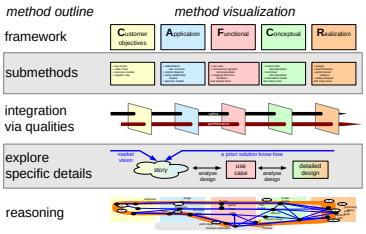
Architectural Reasoning Explained



Gerrit Muller
Buskerud University College
Frogs vei 41 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

This document addresses the way an architect can do his work. It shows many tools that belong to the architect toolkit: The CAFCR model, tools per 5 views (Customer Objectives, Application, Functional, Conceptual and Realization), Qualities, Story telling and Reasoning in multiple dimensions

version: 3.4 status: preliminary draft August 4, 2016

Contents

List of Figures

Introduction

This book integrates the "CAFCR" model, design via qualities, story telling and architectural reasoning into an open-ended architecting method. The background, goal, context and case are introduced in part I. The theoretical framework is described in part II. Part III describes the case: medical imaging workstation.

Figure 1 shows the (preliminary) planned structure of the book.

The purpose of the "CAFCR" chapters is to illustrate the means and methods for the different views. Many more methods are available. It is not feasible to cover all these methods with significant depth, every method in itself can be expanded into a book. I hope to bootstrap designers and potential architects by showing a reasonable set of methods, enabling them to choose, change and expand their tool set.

At this moment the book is in its infancy. As a consequence some chapter references are not yet correct. Most articles are updated based on feedback from readers and students. The most up to date version of the articles can always be found at [11]. The same information can be found here in presentation format.

Chapters can be read as autonomous units. The sequence choosen here is more or less top down, hopping from one viewpoint to the next.

Recommended literature and other resources:

- "The Art of Systems Architecting", Rechtin [19]
- "Systems Engineering Guidebook", Martin [10]
- "Resources for Software Architects", Bredemeyer [1]
- "Role of the Software Architect", Bredemeyer [2]

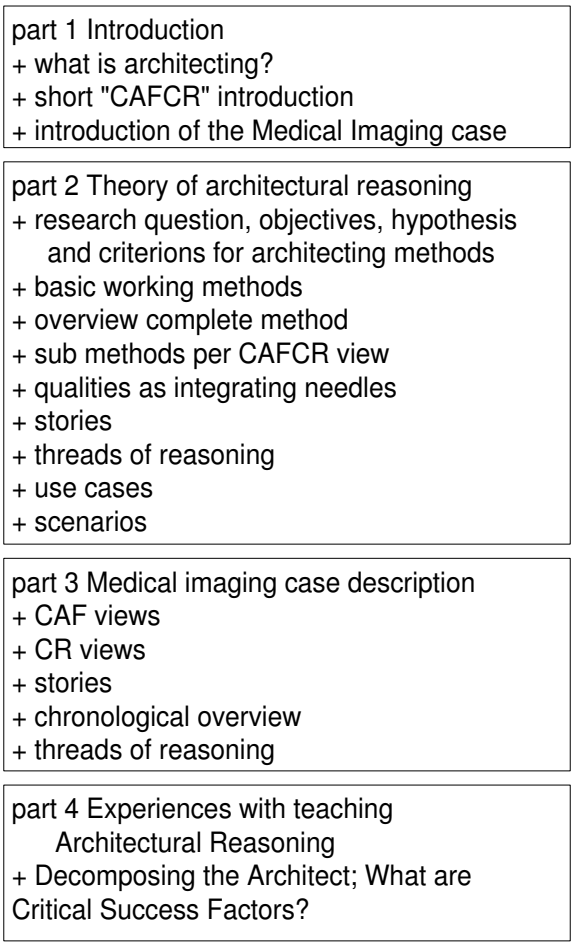


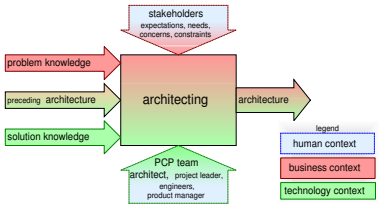
Figure 1: Structure of “Architectural Reasoning Explained”

Part I

Introduction

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 attention.

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 [8] for a much wider variation of definitions for architecture. Essential in this definition is the inclusion of the user context in architecture.

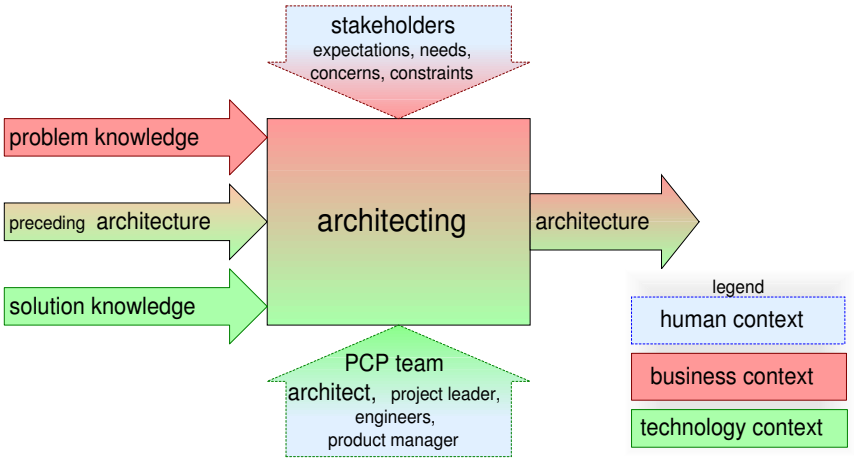


Figure 1.1: Architecting = creating an architecture

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 [16]. 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.

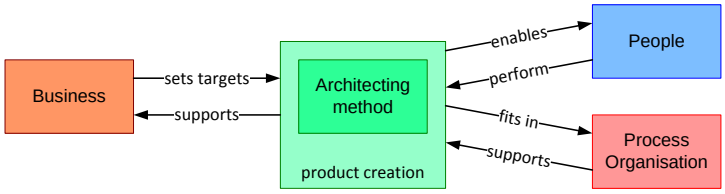


Figure 1.2: The business context of architecting methods

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.

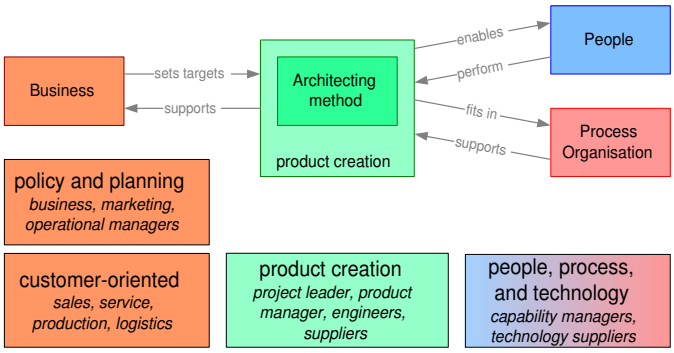


Figure 1.3: Stakeholders of the product creation within a company itself

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.

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.

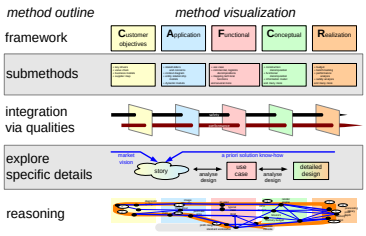
The *people, process, and technology management* is concerned with processes, methods, tools, skills of people, intellectual property, and technology development. These concerns will sometimes result in operational requirements. Care should be taken that the justification of these requirements is clear. 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 **??**.

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.

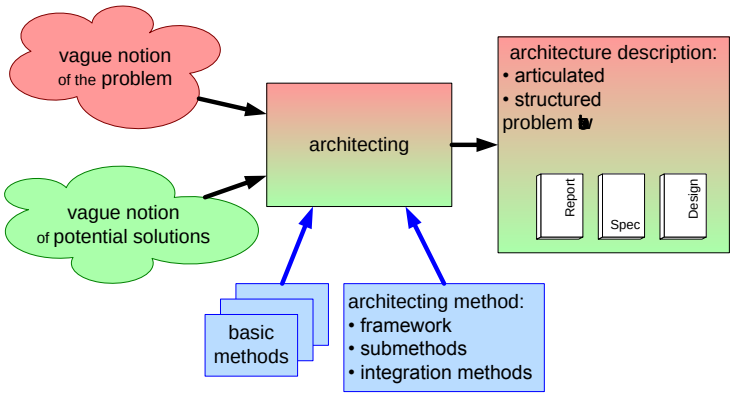


Figure 2.1: An architecting method supports the architect in his process to go from a vague notion of the problem and a vague notion of the potential solutions to a well articulated and structured architecture description

2.2 Architecting Method Overview

Figure 19.6 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 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

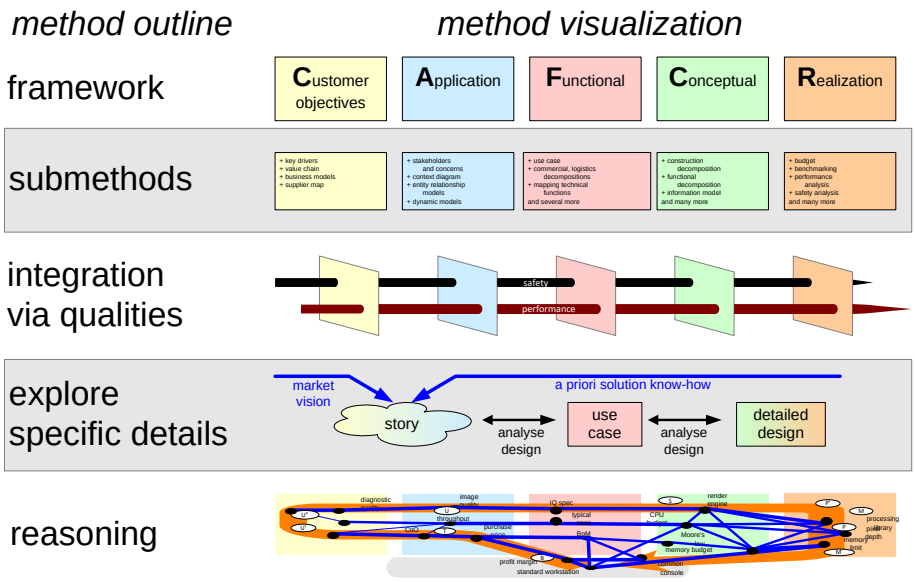


Figure 2.2: The outline of the architecting method with the corresponding visualization that will be used in the later chapters.

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

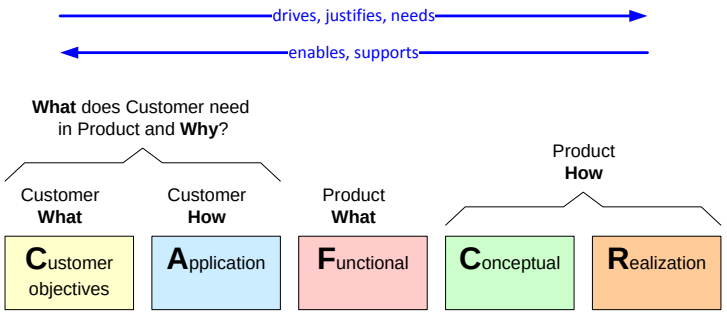


Figure 2.3: The "CAFCR" model

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!).

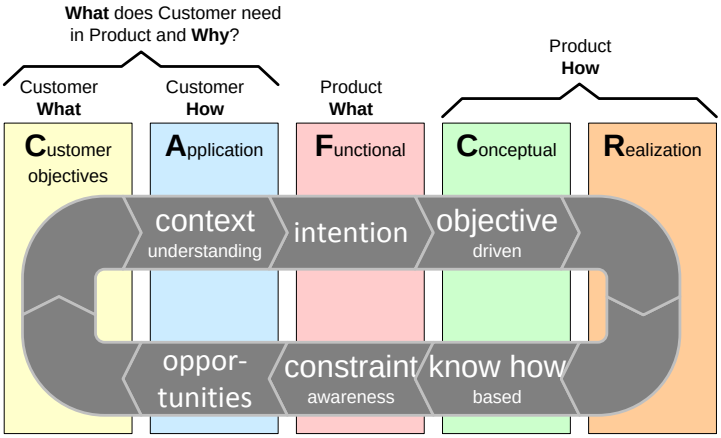


Figure 2.4: Iteration over the CAFCR views and the operational view. The task of the architect is to integrate all these viewpoints, in order to get a *valuable*, *usable* and *feasible* product.

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 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 [17]. The *justification* and the *needs* are expressed in the Customer Objectives View, the Application View, and the operational 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.

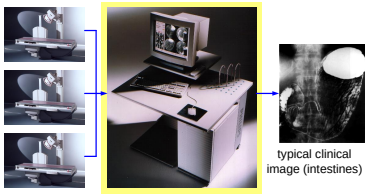
The "CAFCR" model can be used recursively: many customers are part of a longer value chain and deliver products to customers themselves. Understanding of the customer's customer improves the understanding of the requirements.

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.

[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.

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.

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.

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

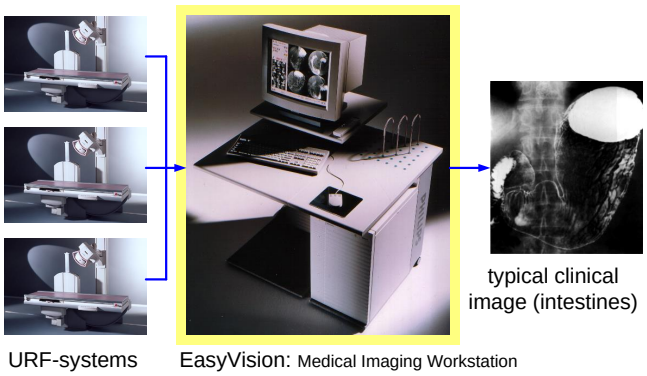


Figure 3.1: Easyvision serving three URF examination rooms

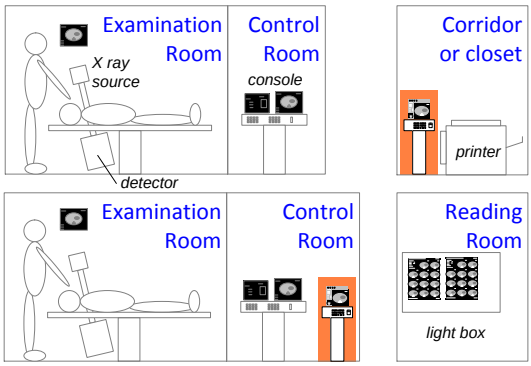


Figure 3.2: X-ray rooms with Easyvision applied as printserver

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. The digital availability of images opens all kinds of possibilities. The simplest

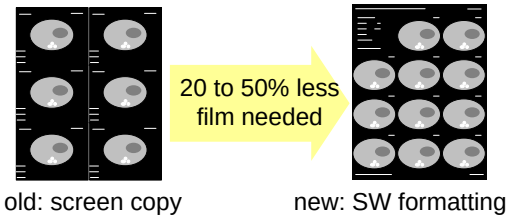


Figure 3.3: Comparison screen copy versus optimized film

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

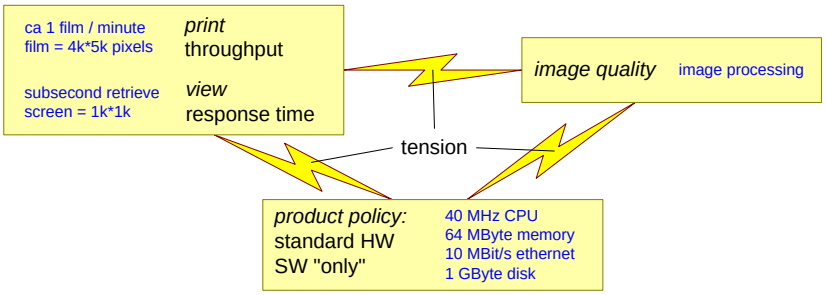


Figure 3.4: Challenges for product creation

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 response 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

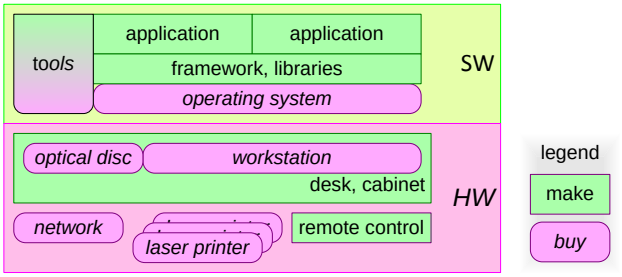


Figure 3.5: Top-level decomposition

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

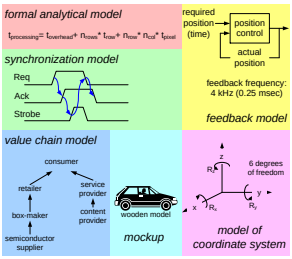
a family of products, with many application innovations. In Part III we will show some of these innovations in more detail and in relation to the product value.

Part II

Theory of architectural reasoning

Chapter 4

Basic Working Methods of a System Architect



4.1 Introduction

The basic working methods of the 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 4.2.
- Decomposition, breaking up a large problem in smaller problems, introducing interfaces and the need for integration, see section 4.3.
- Quantification, building up understanding by quantification, from order of magnitude numbers to specifications with acceptable confidence level, see section 4.4.
- Decision making when lots of data is missing, see section 4.5.
- Modelling, as means of communication, documentation, analysis, simulation, decision making and verification, see section 4.6.

- Asking Why, What, How, Who, When, Where questions, see section 4.7.
- Problem solving approach, see section 4.8.

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 [19], [13] and [14] for additional descriptions of the work and skills of the architect.

4.2 Viewpoint hopping

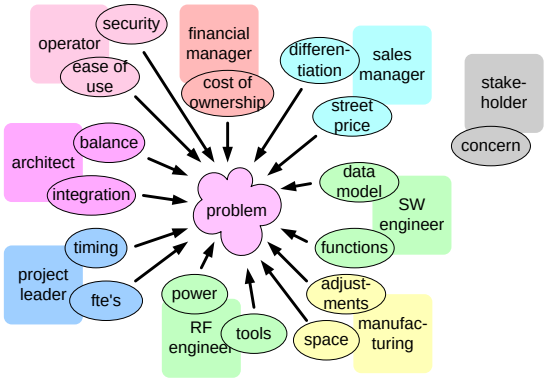


Figure 4.1: Small subset of viewpoints

The architect is looking towards problems and (potential) solutions from many different viewpoints. A small subset of viewpoints is visualized in figure 4.1, where the viewpoints are shown as stakeholders with their concerns.

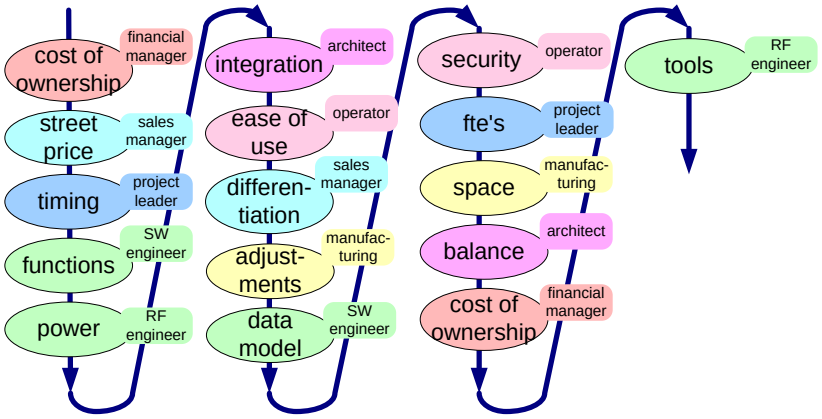


Figure 4.2: Viewpoint Hopping

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 4.2 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.

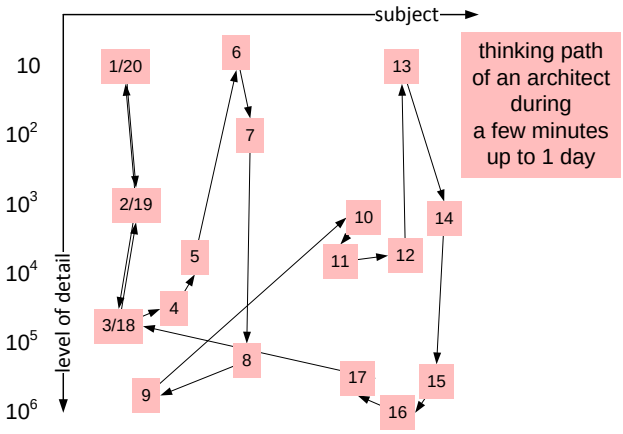


Figure 4.3: The seemingly random exploration path

Viewpoint hopping is happening quite fast in the head of the architect. Besides changing the viewpoint the architect is also zooming in and out with respect to the level of detail. The dynamic range of the details taken into account is many orders of magnitude. Exploring different subjects and different levels of detail together can be viewed as an exploration path. The exploration path followed by the architect (in the architect's head) appears to be quite random. Figure 4.3 shows an example of an exploration path happening inside the architects head.

The plane used to show the exploration path has one axis with *subjects*, which can be stakeholders, concerns, functions, qualities, design aspects, et cetera, while the other axis is *the level of detail*. A very coarse (low level of detail) is for example the customer key driver level (for instance cost per placement is 0.1 milli-cent/placement). Examples at the very detailed level are lines of code, cycle accurate simulation data, or bolt type, material and size.

Both axis span a tremendous dynamic range, creating 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 4.4 for these 2 modes of scanning.

The scanning approach taken by the architect can be compared with *simulated*

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

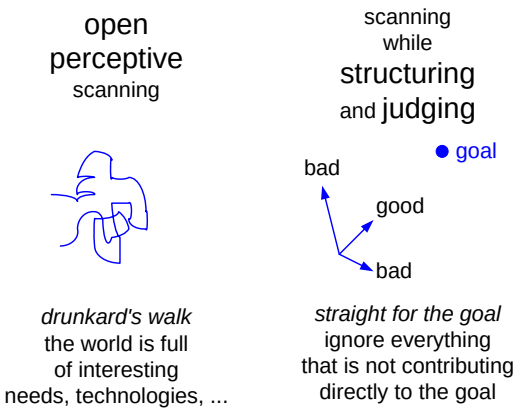


Figure 4.4: Two modes of scanning by an architect

annealing methods for optimization[18]. An interesting quote from this book, comparing optimization methods:

> 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...

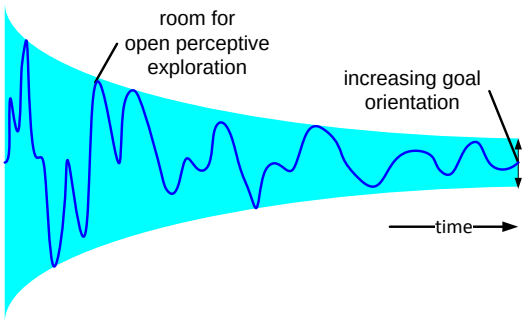


Figure 4.5: Combined open perceptive scanning and goal-oriented scanning

See also figure 4.5 for the combined scanning path. The perceptive mode is used more early in the project, while at the end of the project the goal oriented mode is dominant.

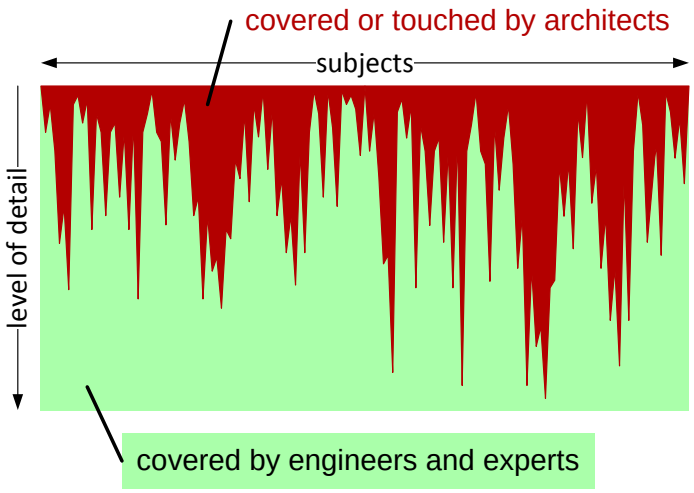


Figure 4.6: The final coverage of the problem and solution space by architect and engineers

The coverage of the problem and solution space is visualized in figure 4.6. Note that the area covered or touched by the architect(s) is not exclusively covered, engineers will also cover or touch that area partially. The architect needs experience to learn when to dig deeper and when to move on to next subjects. Balancing depth and breadth is still largely an art.

4.3 Decomposition and integration

The architect applies a reduction strategy by means of decomposition over and over, as shown in figure 4.7. Decomposition is a very generic principle. Decomposition can be applied for many different problem and solution dimensions, as will be shown in the later sections.

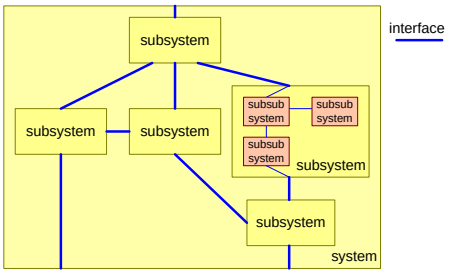


Figure 4.7: Decomposition, interface management and integration

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 decoupling 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. The synthesis or integration part is more difficult and time consuming, and will be perceived as the main contribution by the architect.

4.4 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.

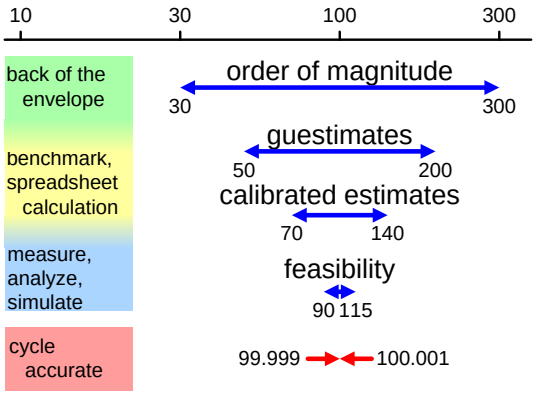


Figure 4.8: Successive quantification refined

The precision of the quantification increases during the project. Figure 4.8 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 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 4.8 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 4.9 shows an example how the quantification evolves in time. The dotted red line represents the required performance as defined in the specification. The

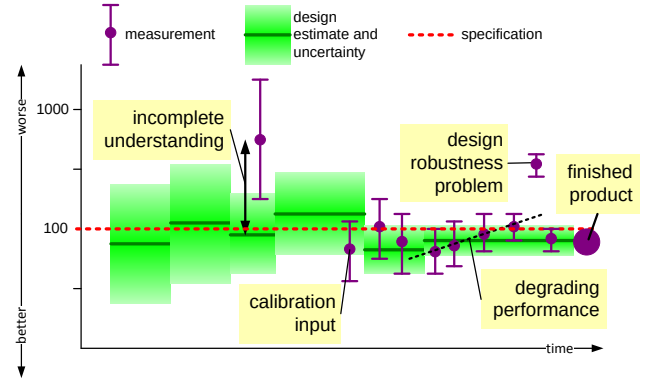


Figure 4.9: Example of the evolution of quantification in time

shaded area indicates the "paper" value, with its accuracy. The measurements are shown as dots with a range bar. A large difference between paper value and measurement is a clear indication of missing understanding. Later during the implementation continuous measurements monitor the expected outcome, in this example a clear degradation is visible. Large jumps in the measurements are an indication of a design which is not robust (small implementation changes cause large performance deviations).

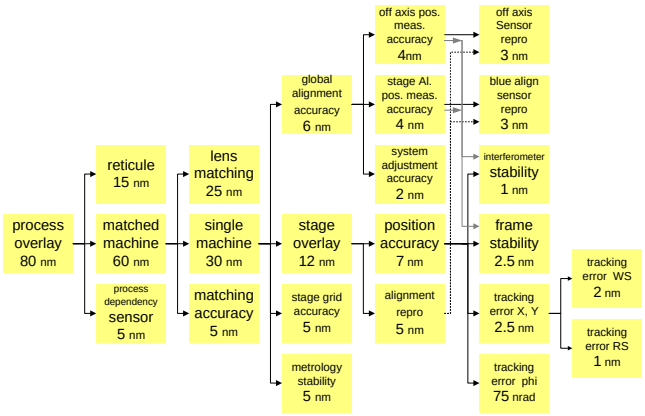


Figure 4.10: Example of a quantified understanding of overlay in a wafer stepper

Figure 4.10 shows a graphical example of an "overlay" budget for a wafer stepper. 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 wafer stepper. 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.

4.5 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.

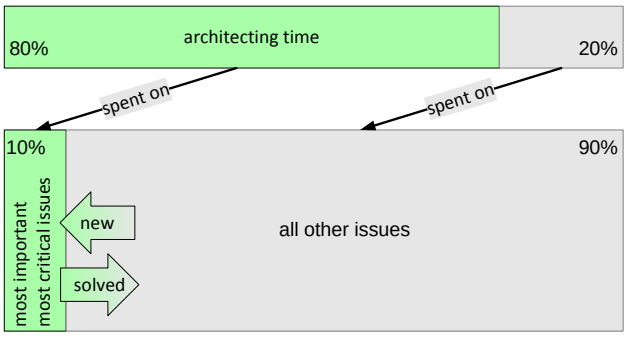


Figure 4.11: The architect focuses on important and critical issues, while monitoring the other issues

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 4.11 visualizes the time distribution of the architect: 80% of the time is spent on 10% of the issues.

The architect will, often implicitly, work on the basis of a top 10 issue list, the ten most relevant (important, urgent, critical) issues. Figure 4.12 shows an example of such a “worry”-list.

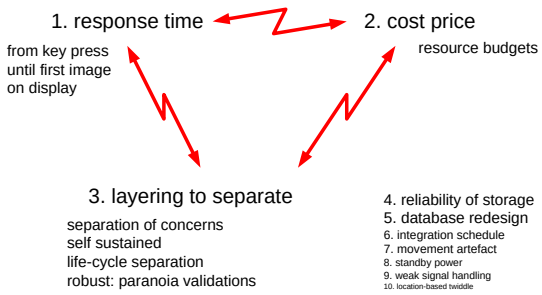


Figure 4.12: Example worry list of an architect

4.6 Modelling

Modelling is one of the most fundamental tools of an architect.

A **model** is
a **simplified** representation of
part of the **real world** used for:

communication, documentation
analysis, simulation,
decision making, verification

In summary models are used to obtain insight and understanding, facilitating communication, documentation, analysis, simulation, decision making, verification. 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.

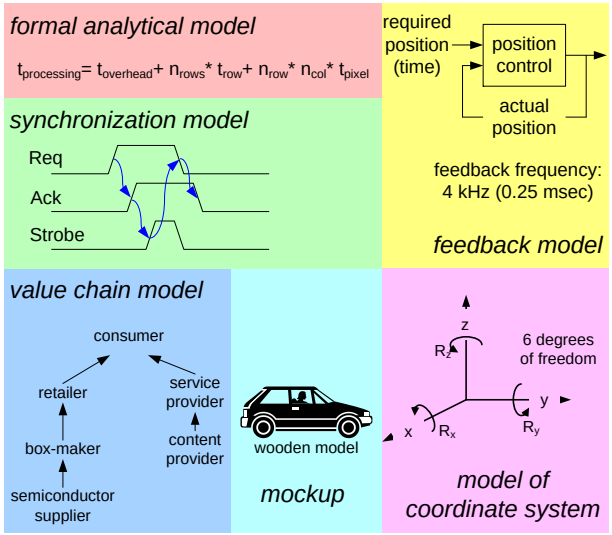


Figure 4.13: Some examples of models

Models exist in a very rich variety, an arbitrary small selection of models is shown in figure 4.13.

Models have many different manifestations. Figure 4.14 shows some of the different types of models, expressed in a number of adjectives.

Models can be *mathematical*, expressed in formulas, they can be *linguistic*, expressed in words or they can be *visual*, captured in diagrams. A model can be

mathematical linguistic	visual
formal	informal
quantitative	qualitative
detailed	global
concrete	abstract
accurate	approximate
executable	read only
←rational—	—intuitive→

Figure 4.14: Types of models

formal, where notations, operations and terms are precisely defined, or 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.

4.7 WWHWWW questions

Why	Who
What	When
How	Where

Figure 4.15: The starting words for questions by the architect

All "W" questions are an important tool for the architect. Figure 4.15 shows the useful starting words for questions to be asked by an 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 4.16 shows that the recursion with **why** questions broadens the scope, and recursion with **how** questions opens more details in a smaller scope.

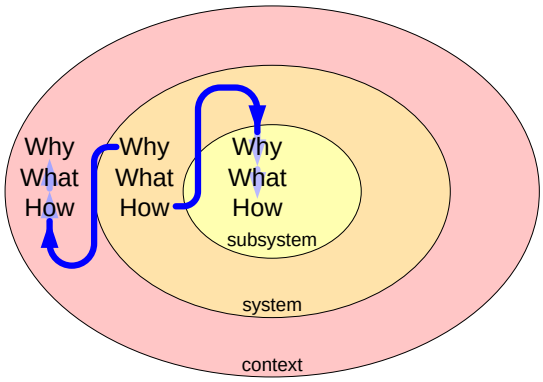


Figure 4.16: Why broadens scope, How opens details

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.

4.8 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.

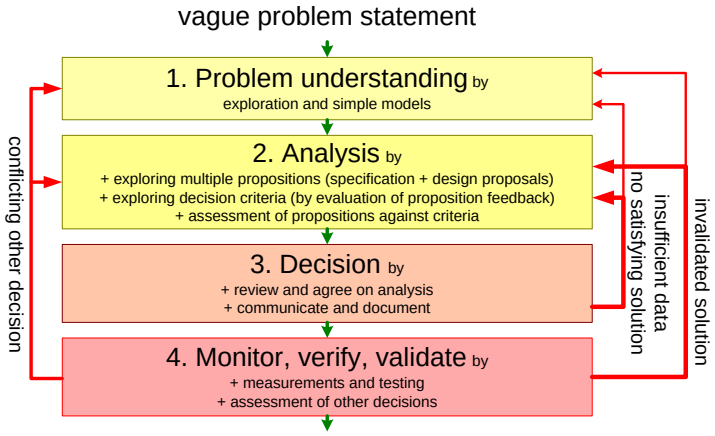


Figure 4.17: Flow from problem to solution

An approach to make these decisions is the flow depicted in Figure 4.17. 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 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.

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 4.18 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

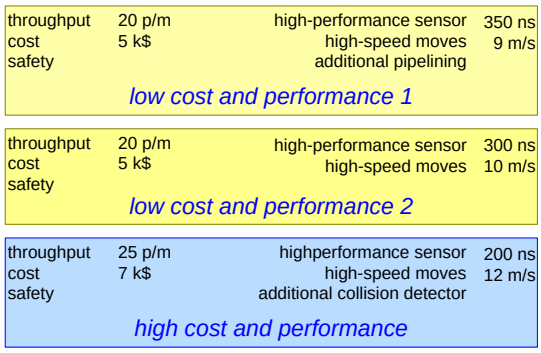


Figure 4.18: Multiple propositions

decision itself is documented and communicated[2]. 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 many assumptions. An significant amount of work is to monitor the consequences and 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, starting again with the understanding for serious mismatches ("apparently we don't understand the problem at all") or direct into the analysis for smaller mismatches.

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 4.19. 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 problem solving might have impact on the more global decisions.

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

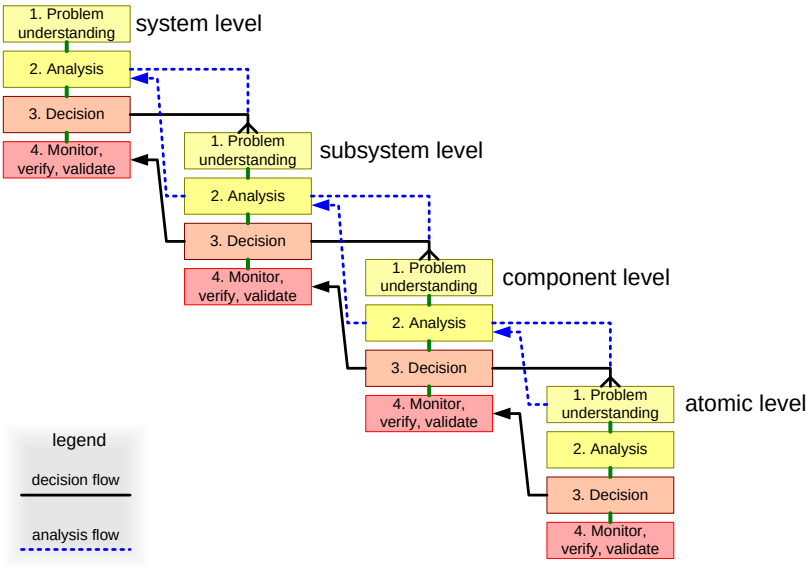


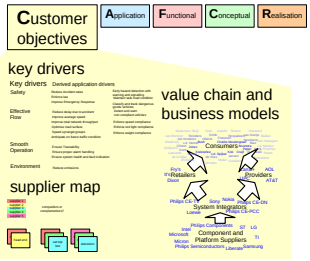
Figure 4.19: Recursive and concurrent application of flow

4.9 Acknowledgements

The team of composable architectures, with the following members Pierre America, Marcel Bijsterveld, Peter van den Hamer, Jürgen Müller, Henk Obbink, Rob van Ommering, and William van der Sterren within Philips Research provided valuable feedback for this article

Chapter 5

The customer objectives view



5.1 Introduction

The customer objectives view describes the goals of the customer, the **what**. The goal of articulating these objectives is to better understand the needs and therefore to be able to design a better product.

In searching the objectives some focus on the product is needed, although the architect must keep an open mind. The architect must prevent a circular reasoning, starting from the product functionality and, blinded by the product focus, finding only objectives matching with this same functionality.

Ideally the trade-offs in the customer domain become clear. For instance what is the trade-off between performance and cost, or size and performance or size and cost. The key driver method articulates the essence of the customer needs in a limited set of drivers.

The customer is often driven by his context. Some of the models and methods described here address ways to understand the customer context, such as value chains and business models. Value chains and business models are used to address the customer's customer. The supplier map addresses the supplying side of the customer.

Figure 5.1 shows an overview of the methods in the customer objectives view.

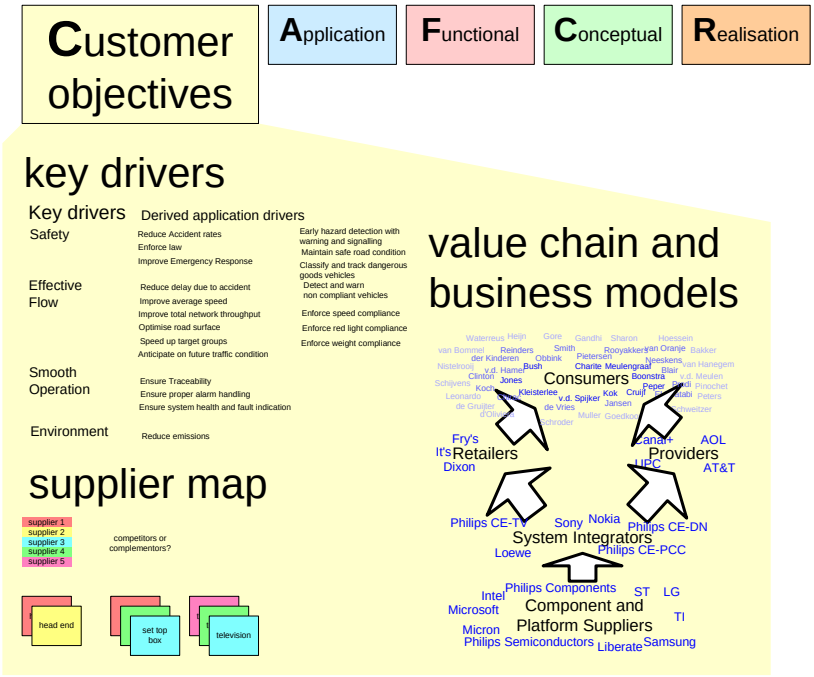


Figure 5.1: Overview of Customer Objectives View methods

5.2 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. 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.

Figure 5.2 shows an example of key drivers for a motorway management system, an analysis performed at Philips Projects in 1999.

Figure 5.3 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 5.2 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 graph, which

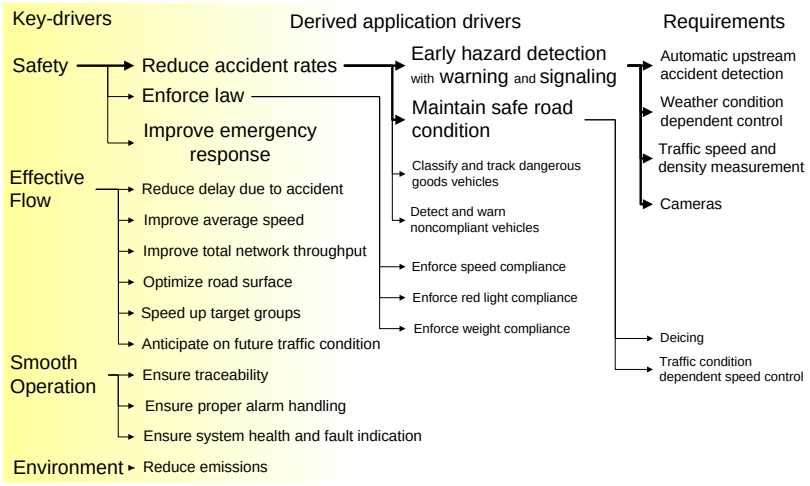


Figure 5.2: Example of the four key drivers in a motorway management system

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.

Figure 5.4 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

• 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 brainstorming 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 5.3: Submethod to link key drivers to requirements, existing of the iteration over four steps

• 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"
• Do not worry about the exact boundary between Customer Objective and Application	create clear goal means relations

Figure 5.4: Recommendations for applying the key driver submethod

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.

5.3 Value chain and business models

The position of the customer in the value chain and the business models deployed by the players in the value chain are important factors in understanding the goals of this customer.

Figure 5.5 shows an example value chain from the Consumer Electronics Domain. At the start of the chain are the component suppliers, making chips and other elementary components such as optical drives, displays, et cetera. These compo-

nents are used by system integrators, building the consumer appliances, such as televisions, set top boxes and cellphones. Note that this value chain is often longer than shown here, where components are aggregated in larger components into subassemblies and finally into systems.

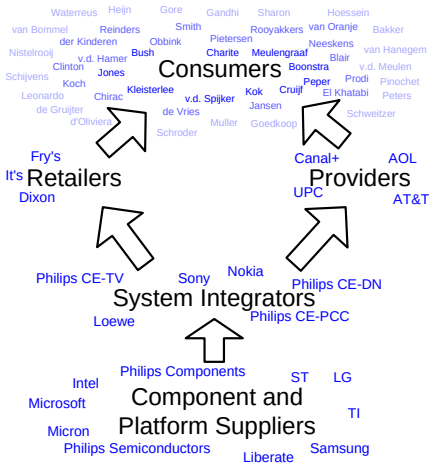


Figure 5.5: Example value chain

The consumer appliances itself are distributed through 2 different channels: the retailers and the service providers. Retailers sell appliances directly to the consumers, earning their money with this appliance sales and sometimes also with maintenance contracts for these appliances. Providers sell services (for instance telecom, internet), where the appliance is the means to access these services. The providers earn their money via the recurring revenues of the services.

Retailers and service providers have entirely different business models, which will be reflected by differences in the key drivers for both parties.

Reality is even much more complicated. For instance adding the *content* providers to the value chain adds an additional set of business models, with a lot of conflicting interests (especially Digital Rights Management, which is of high importance for the content providers, but is often highly conflicting with (legal) consumer interests).

5.4 Suppliers

The value chain must be described from the point of view of the customer. The customer sees your company as one of the (potential) suppliers. From the customer 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

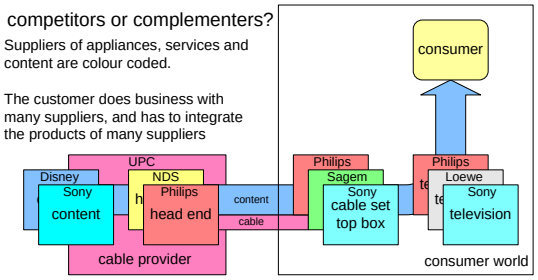
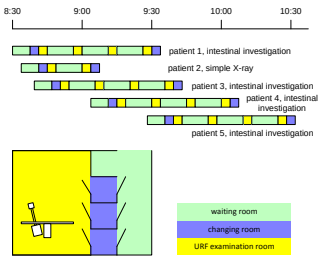


Figure 5.6: Example of simple supplier map for a cable provider

competitors and complementers, which together will supply the solution to the customer. Figure 5.6 shows an example of a simple supplier map for a cable provider. If your company is delivering set top boxes, then some companies can be viewed as competitor and complementer at the same time.

Chapter 6

The application view



6.1 Introduction

The application view is used to understand how the customer is achieving his objectives. The methods and models used in the application view should discuss the customer's world. Figure 6.1 shows an overview of the methods discussed here.

The customer is a gross generalization, which can be made more specific by identifying the customer stakeholders and their concerns, see section 6.2.

The customer is operating in a wider world, which he only partially controls. A context diagram shows the context of the customer, see section 6.3. Note that part of this context may interface actively with the product, while most of this context simply exists as neighboring entities. The fact that no interface exists is no reason not to take these entities into account, for instance to prevent unwanted duplication of functionality.

The customer domain can be modelled in static and dynamic models. Entity relationship models (section 6.4) show a static view on the domain, which can be complemented by dynamic models (section 6.5).

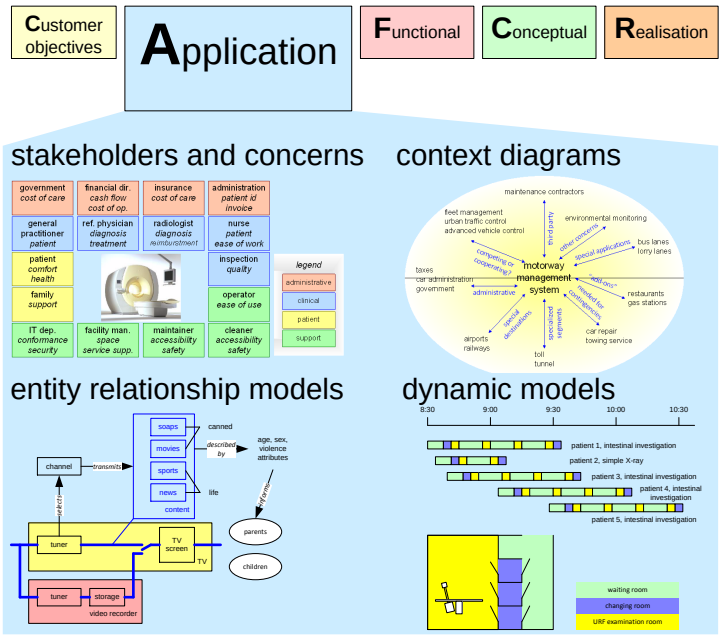


Figure 6.1: Overview of methods and models that can be used in the application view

6.2 Customer stakeholders and concerns

In the daily use of the system many human and organizational entities are involved, all of them with their own interests. Of course many of these stakeholders will also appear in the static entity relationship models. However human and organizations are very complex entities, with psychological, social and cultural characteristics, all of them influencing the way the customer is working. These stakeholders have multiple concerns, which determine their needs and behavior. Figure 6.2 shows stakeholders and concerns for an MRI scanner.

The IEEE 1471 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. The next step can be to gain insight in the *informal* relationships. In many cases the formal relationships, such as organization charts and process descriptions are solely used for this view, which is a horrible mistake. Many organizations function thanks to the unwritten information flows of the social system. Insight in the informal side is required to prevent a solution which does only work in theory.

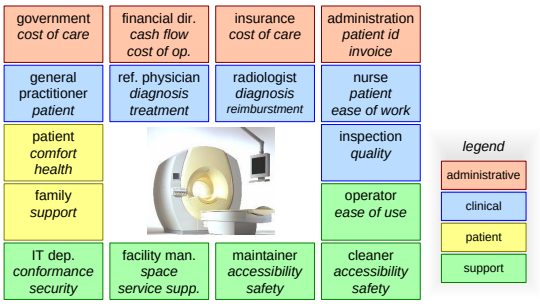


Figure 6.2: Stakeholders and concerns of an MRI scanner

6.3 Context diagram

The system is operating in the customer domain in the context of the customer. In the customer context many systems have some relationship with the system, quite often without having a direct interface.

Figure 6.3: Systems in the context of a motorway management system

Figure 6.3 shows a simple context diagram of a motorway management system. 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 central traffic control center)

6.4 Entity relationship model

The OO (Object Oriented software) world is quite used to entity relationship diagrams. These diagrams model the outside world in such a way that the system can interact with the outside world. These models belong in the ”CAFCR” thinking in the conceptual view. The entity relationship models advocated here model the customers world in terms of entities in this world and relations between them. Additionally also the activities performed on the entities can be modelled. The main purpose of this modelling is to gain insight in how the customer is achieving his objectives.

One of the major problems of understanding the customers world is its infinite size and complexity. The art of making an useful entity relationship model is to very carefully select what to include in the model and therefore also what **not** to include. Models in the application view, especially this entity relationship model, are by definition far from complete.

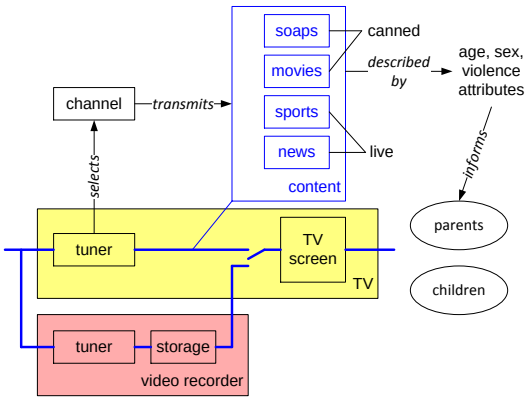


Figure 6.4: Diagram with entities and relationship for a simple TV appliance

Figure 6.4 shows an example of an entity relationship model for a simple TV. Part of the model shows the well recognizable flow of video content (the bottom part of the diagram), while the top part shows a few essential facts about the contents. The layout and semantics of the blocks are not strict, these form-factors are secondary to expressing the essence of the application.

6.5 Dynamic models

Many models, such as entity relationship models, make the static relationships explicit, but don’t address the dynamics of the system. Many different models can be used to model the dynamics, or in other words to model the behavior in time. Examples are of dynamic models are shown in figure 6.5

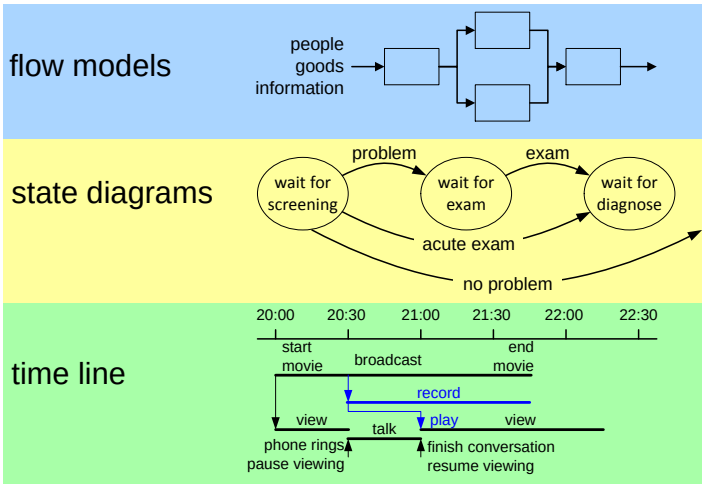


Figure 6.5: Examples of dynamic models

Productivity and Cost of ownership models are internally based on dynamic models, although the result is often a more simplified parameterized model, see figure 6.6.

Figure 6.7 shows an example of a time-line model for an URF examination room. The involved rooms play an important role in this model, therefore an example geographical layout is shown to explain the essence of the time-line model.

The patient must have been fasting for an intestine investigation. In the beginning of the examination the patient gets a barium meal, which slowly moves through the intestines. About every quarter of an hour a few X-ray images-images are made of the intestines filled with barium. This type of examination is interleaving multiple patients to efficiently use the expensive equipment and clinical personnel operating it.

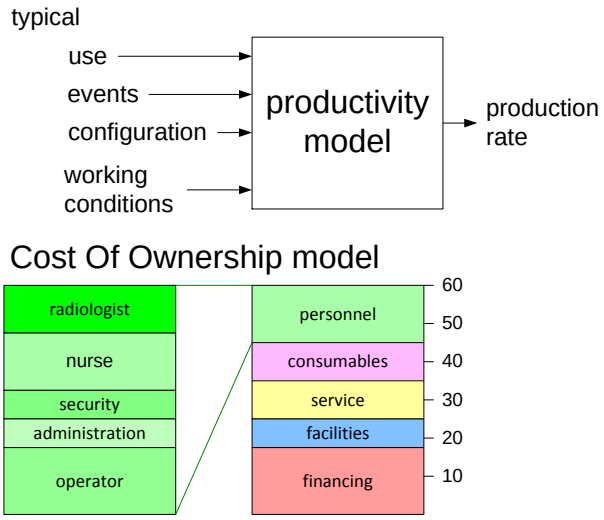


Figure 6.6: Productivity and cost models

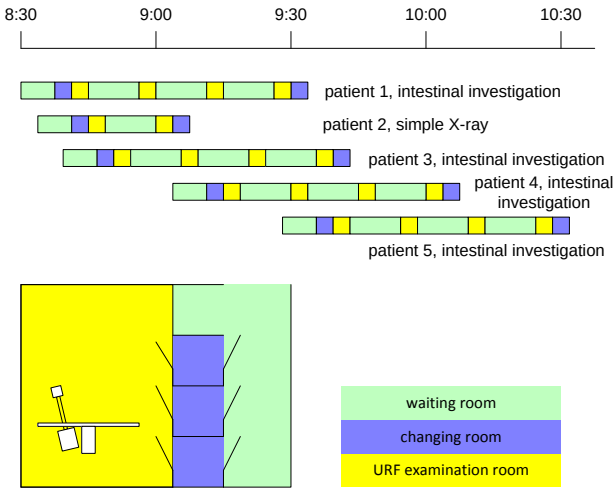
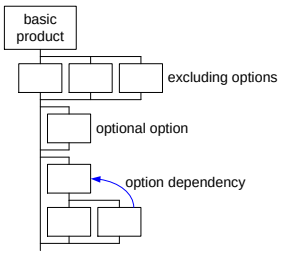


Figure 6.7: Dynamics of an URF examination room

Chapter 7

The functional view



7.1 Introduction

The functional view describes the **what** of the system, or in other words: how is the system perceived from the outside, especially by the customer. The product specification (or requirement specification[1]) covers normally the content of this view. The content of these specs should be observable from outside of the system.

Several methods and models can be used in this view. (Use) Cases, section 7.2, describing the system from user point of view. Commercial, service and goods flow decompositions, section 7.3, describing the product in terms of the commercial packages and options and the other logistics dimensions. Function and feature specifications, section 7.4, focusing on a more functional view or a feature wise view. Performance specification and models, section 7.5, describing performance aspects such as throughput and latency, as a function of the commercial options and the available parameter space.

The information model, described in section 7.6 is especially important when interfacing with other systems. Section 7.7 describes the role of standards in the product specification.

[1] Or any combination of the words: system, product, functional, performance, requirement and specification

7.2 Case descriptions

Use cases are an useful means to describe desired functional behavior in a more cohesive way. An use case describes a set of functions together in typical, worst case or exceptional scenarios. Use cases become really effective if the use case is not limited to the functional behavior, but when the non-functional aspects are described as well.

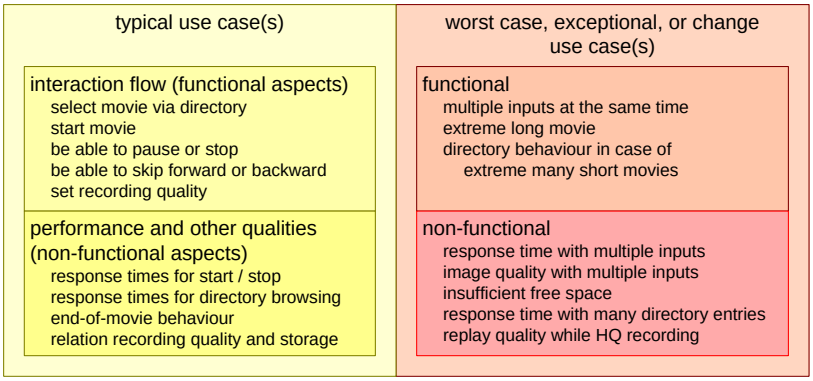


Figure 7.1: Example personal video recorder use case contents for typical use case and worst case or exceptional use case

Figure 12.4 shows the possible content for personal video recorder use cases. The most typical use is to watch movies: find the desired movie and play it. Additional features are the possibility to pause or 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 (maximum amount of video stored is 20 hours standard 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 if the boundaries are rather situational dependent, the circumstances can be described in the use case.

The exceptional use case are comparable to the worst case use cases. Exceptions can be described directly (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 are dependent on the circumstances.

Figure 12.7 summarizes recommendations for working with use cases. Many use case descriptions suffer from fragmentation: every function is described as a separate use case. The overview is lost, and the interaction of functions is missed. The granularity of use cases should match with the external use.

+ combine related functions in one use case
- do not make a separate use case for every function
+ include non-functional requirements in the use cases

+ minimise the amount of required *worst case* and *exceptional use cases*
- excessive amounts of use cases propagate to excessive implementation efforts
+ reduce the amount of these use cases in steps
- a few well chosen *worst case* use cases simplifies the design

Figure 7.2: Recommendations for working with use cases

Another problem is that too many use cases are described, again with the consequence of losing the overview and worse spending too much time at not relevant specification issues. The problem is that up front the knowledge is insufficient to select the most relevant use cases. A somewhat more extensive exploration phase is recommended, where afterwards a reduction of use cases is applied.

7.3 Commercial, service and goods flow decomposition

The commercial granularity of sellable features and the allowed configurations can be visualized in a commercial configuration tree, as shown in figure 7.3. All items in such a tree will appear in brochures, folders, catalogues. Note that the commercial granularity is often somewhat more coarse than the design decomposition. The commercial packaging is optimized to enable the sales process and to 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.

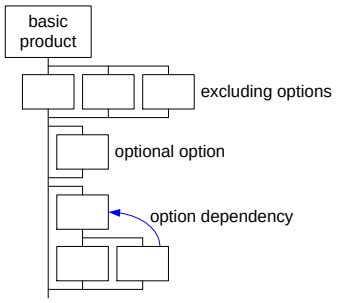


Figure 7.3: Commercial tree as means to describe commercial available variations and packaging

The commercial tree makes clear what the relations between commercial options are. Options might be exclusive (either this printer or that printer can be connected, not both at the same time). Options can also be dependent on other options (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.

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 7.4 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 costs 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 efficient, 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 needs to support both the commercial as well as the service decomposition, has a finer granularity than both these decom-

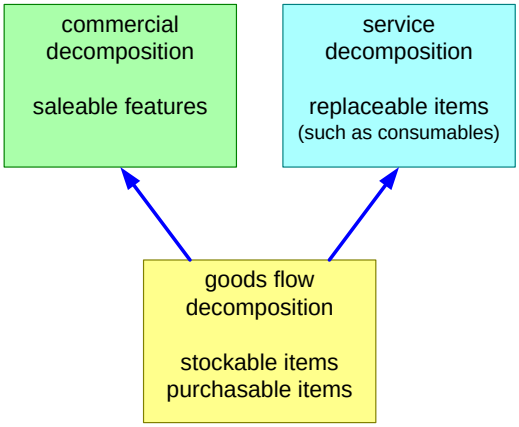


Figure 7.4: Logistic decompositions for a product

positions. At the input side is the goods flow decomposition determined by the granularity of the supply chain.

In Philips all three decompositions used to fit 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. These decompositions are used in logistics information systems (MRP).

7.4 Function and feature specifications

The product specification needs to define the functions and features of the product. The decomposition for this description is again another decomposition than the commercial decomposition. The commercial decomposition is too coarse to use it as basis for the product specification. The technical decomposition in functions and features is kind of a building box to compose commercial products and packages.

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 7.5: Mapping technical functions on products

Figure 7.5 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 and logistics oriented decompositions.

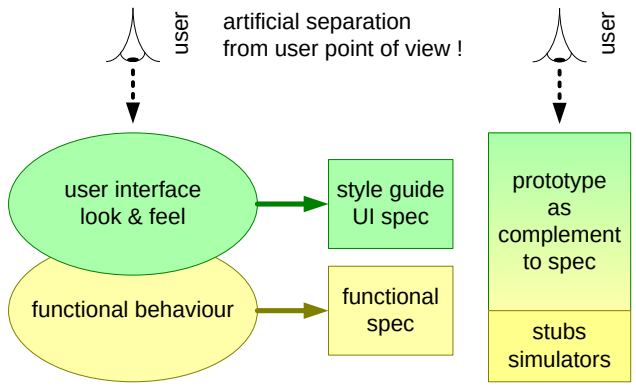


Figure 7.6: Relation between user interface and functional specification

The struggle in nailing down the functional specification is the degree in which user interface and functional specification are decoupled and separated. Separation eases the delivery of look and feel variants. However this separation from user point of view is rather artificial, see figure 7.6, which shows that the user experiences the system behavior via the user interface. As design team we create then artifacts as style guides, user interface specifications and functional specifications.

Another consideration is the high dynamics of user interface details versus the relative stability of the functions itself. Hard coupling of user interface description and functional specification propagates the dynamics of user interface details into the entire functional specifications.

Figure 7.6 offers an alternative solution for this dilemma by using a prototype as complement to the specification for the user interface details. Such an approach allows the team to limit the functional specification, style guide and user interface specification to the essentials. A clear description of the way of working is required for quality assurance purposes: the specification is leading and is verified, is the prototype archived and a formal part of the specification?

7.5 Performance

The performance need to be specified quantitatively and verifiable in the functional view. This means that the performance needs to be specified in conjunction with the circumstances in which this performance specification is valid. In easy cases a simple maximum value is sufficient, which is valid under all circumstances. In many systems the performance specification is more complicated: the system performance depends on the user settings of the system.

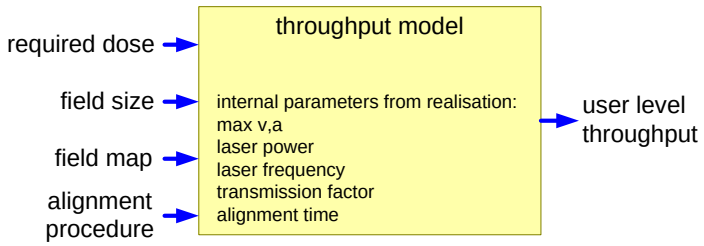


Figure 7.7: Example of performance modelling: throughput as function of user controlled values

In not too complicated 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, which describes the required relation between performance and user settings. Figure 7.7 shows an example of such a performance model for waferstepper throughput.

Throughput models are the result of several iterations between problem and solution space. Sufficient understanding of the solution space is needed to know which user parameters are relevant in the throughput model.

From the functional view *(the what perspective)* the internal design parameters are not relevant. In the iteration and decision process this model with external and internal parameters is a means to understand the consequences of design choices and to understand the consequences (cost) of customer needs.

The notion of *internal* and *external* is also somewhat artificial. In this example many customers measure the dose and do expect a certain relationship between dose and throughput. These customers perceive dose as externally known parameter.

Other examples of performance data are: standby time of a cell phone, gas consumption of a car, average monthly cost of a lease car. Note the increasing need in these examples for specification of the context.

7.6 Information Model

The information model is layered, as shown in figure 7.8. The highest layer is the understanding of the humans using the information model. This understanding is always biased by the individual human knowledge, emotional state and many other human factors, see [15]. The real meaning of information for human beings is never completely defined, humans always add interpretation to the definition.

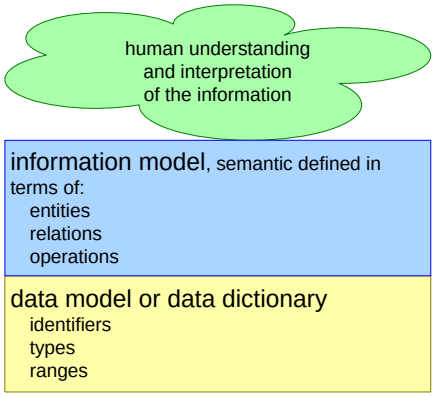


Figure 7.8: Layering of information definitions

The information model itself describes the semantics of the information. The syntax and representation aspects are described in the data model or data dictionary.

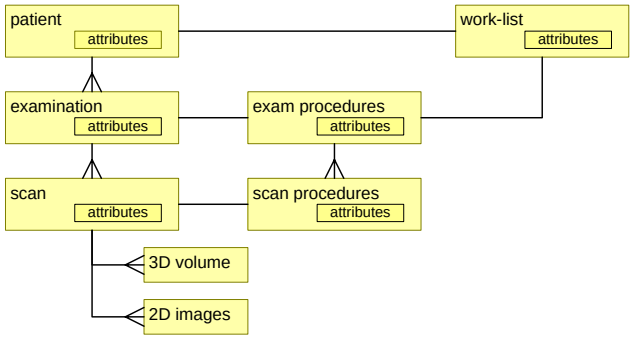


Figure 7.9: Example of a partial information model

The information model describes the information as seen outside of the system. It should not contain internal design choices. The 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 the information model. Figure 7.9 shows an example of a part of an information model.

The ingredients of an internal 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. The information model defines the intended meaning of the information, in terms of entities, their meaning, the relation with other entities and possible operations which can be applied on these entities.

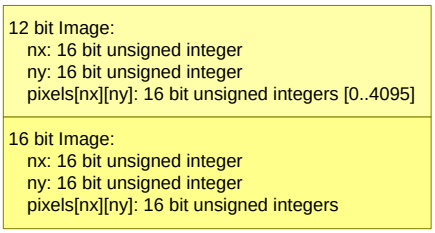


Figure 7.10: Small part of a datamodel

The technical details of the information model, such as exact identifiers, data types and ranges is defined in the datamodel. Figure 7.10 shows a small part of a datamodel defining 12 and 16 bit images. The term data dictionary is also often used for this lower level of definitions.

7.7 Standards

Compliance with standards is part of the product specification. The level of compliance and eventual exceptions need to be specified. Duplication of information in the standard must be avoided (minimize redundancy). The nice characteristic of standards in general is that the standards are extensively described and well defined. Most standard related implementation effort is straight forward engineering work, without the uncertainty of most other parts of the product specification.

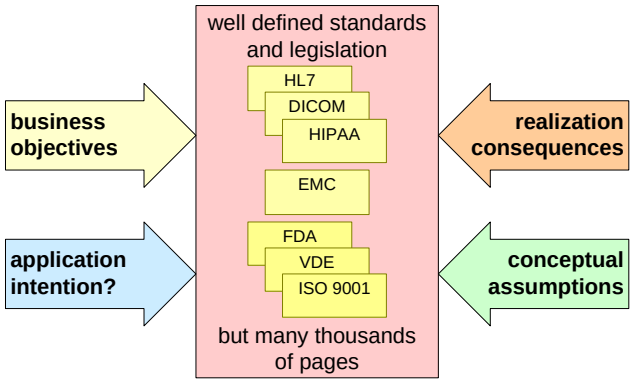


Figure 7.11: The standards compliance in the *functional view* in a broader force field.

Nevertheless architecting work is required in deciding on standards and in designing the implementation. Figure 7.11 shows the forces working upon the standards selection. The market and business environment more or less dictate a set of standards, if the product not comply the system is not viable. Some of these standards are mandatory due to legislation (for instance VDE or FDA related), some 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?* At the other hand standards are created by domain experts, which make all kinds of conceptual assumptions. If the standard is used in a way which does not correspond well with these assumptions, then it creates 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 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 which 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[2]. For communication purposes the architect can make this model explicit.

[2] the CAFCR model is in fact the architecture of the standard itself.

7.8 Summary

The functional view is concerned with all the required externally observable characteristics of the system. The CAFCR model puts a lot of emphasis on the customer. The operational viewpoint, from the producer point of view, determines also part of the system. Figure 7.12 summarizes the content of the functional view, where the left hand side shows the customer specifications and the right hand side the company operational specifications.

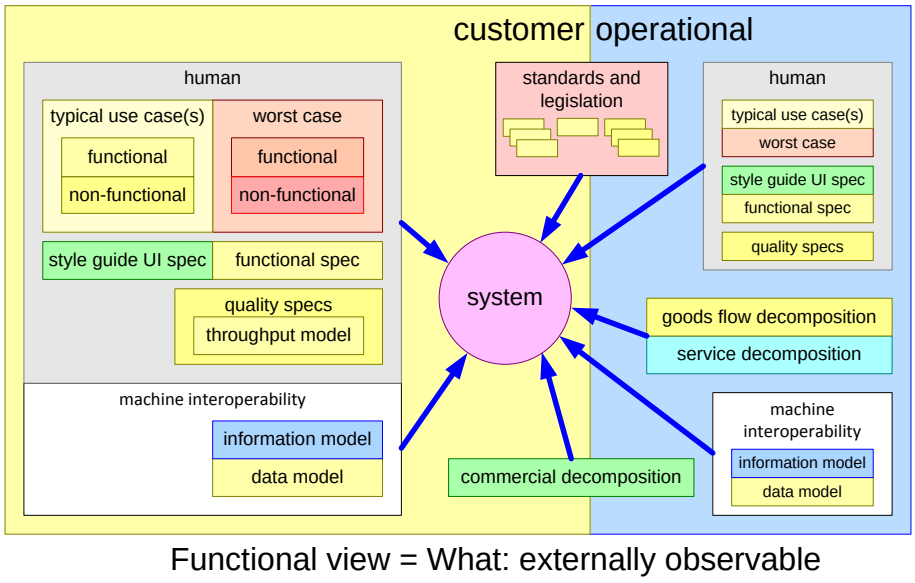


Figure 7.12: Summary of functional view

In the previous chapters we discussed the use cases, user interface, functional specification, quality specifications, and information model from customer point of view. As shown in this figure the same aspects need to addressed from the operational point of view, for example:

- typical use case for service and/or production
- functional specification and user interface for service
- performance of adjustment and verification measurements
- information interface for SPC (Statistical Process Control) purposes

Another classification used in figure 7.12 is human oriented or machine interoperability oriented. Again such a classification is artificial. For some products with a lot of human user interaction this is a useful separation. Other products, for

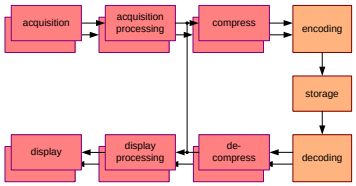
instance electronic or software components to be used in other systems, don't have immediate human users.

7.9 Acknowledgements

William van der Sterren was very fast in providing relevant feedback.

Chapter 8

The conceptual view



8.1 Introduction

The conceptual view is used to understand how the product is achieving the specification. The methods and models used in the conceptual view should discuss the *how* of the product in conceptual terms. The lifetime of the concepts is longer than the specific implementation described in the *Realization* view. The conceptual view is more stable and reusable than the *realization* view.

The dominant principle in design is decomposition, often immediately coupled to interface management of the interfaces of the resulting components. It is important to realize that any system can be decomposed in many relevant ways. The most common ones are discussed here briefly: construction decomposition, section 8.2, functional decomposition, section 8.3, class or object decomposition, other decompositions (power, resources, recycling, maintenance, project management, cost, execution architecture...), and related models (performance, behavior, cost, ...).

If multiple decompositions are used then the relationships between decompositions are important. One of the methods to work with these relationships is via allocation. Within a decomposition and between decompositions the dependency structure is important.

From development management point of view it is useful to identify the infrastructure (factoring out shareable implementations), and to classify the technology in *core*, *key* and *base* technology.

The complement of decomposition is integration. Articulating the integrating concepts (start up, shutdown, safety, exception handling, persistency, resource management,...) provides guidance to the developers and helps to get a consistently behaving system.

8.2 Construction decomposition

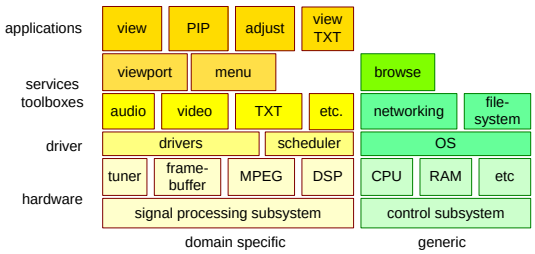


Figure 8.1: Example of a construction decomposition of a simple TV

The construction decomposition views the system from the construction point of view, see figure 8.1 for an example and figure 8.2 for the characterization of the construction decomposition.

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 in compound design units, which are used as unit for testing and release and this often coincides with organizational ownership and responsibility.

management of design	SW example	HW example
unit of creation storage update	file	PCB IP cells IP core
unit of aggregation for organisation test release	package module	box IP core IC

Figure 8.2: Characterization of the construction decomposition

In hardware this is quite often a very natural decomposition, for instance in cabinets, racks, boards and finally IC's, IP cores and cells. The components in the hardware components 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.

8.3 Functional decomposition

The functions as described in the functional view have to be performed by the design. These functions often are an aggregation of more elementary functions in the design. The functional decomposition decomposes end user functions in more elementary functions.

Be aware of the fact that the word *function* in system design is heavily overloaded. It does not help to define sharp boundaries with respect to the functional decomposition. Main criterium for a good functional decomposition is its useability for design. A functional decomposition provides insight how the system will accomplish its job.

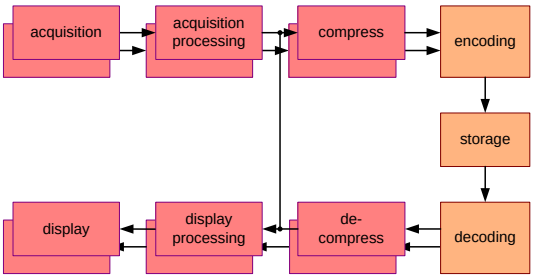


Figure 8.3: Example functional decomposition camera type device

Figure 8.3 shows an example of (part of) a functional decomposition for a camera type device. It shows 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, but from control point of view.

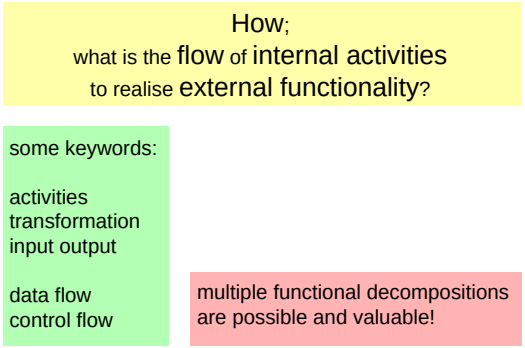


Figure 8.4: Characterization of the functional decomposition

8.4 Designing with multiple decompositions

The design of complex systems always requires multiple decompositions, for instance a construction and a functional decomposition. Many designers in the design team need support to cope with this multiplicity.

Most designers don't anticipate cross system design issues, for instance when asked in preparation of design team meetings. This limited anticipation is caused by the locality of the viewpoint, implicitly chosen by the designers.

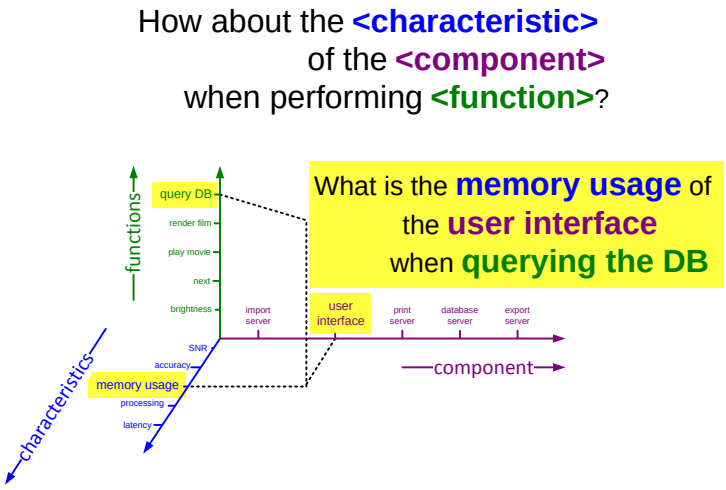


Figure 8.5: Question generator for multiple decompositions

Figure 8.5 shows a method to help designers to find system design issues. A three dimensional space is shown. Two dimensions are the decomposition dimension (component and functional), the last dimension is the design characteristic dimension.

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. Luckily most of the not answered questions will not be relevant. The answer to the memory usage question above might be *insignificant* or *small*.

The architect has to apply a priori know how to select the most relevant questions in the 3D space. Figure 8.6 shows a set of selection factors that can be used to determine the most relevant questions.

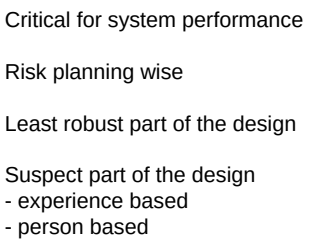


Figure 8.6: Selection factors to improve the question generator

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 part of the design Other parts of the design might be suspect for several reasons. For instance experience learns that response times and throughput do not get the required attention of software designers (experience based suspicion). Or for instance we allocated an engineer to the job with insufficient competence (person based suspicion).

Figure 8.7 shows another potential optimization, to address a line or a plane in the multi dimensional space. The figure shows an example of a memory budget for the system, which is addressing all memory aspects for both functions and components in one budget. The other example is the design specification of a database query, where the design addresses the allocation to components as well as all relevant design characteristics.

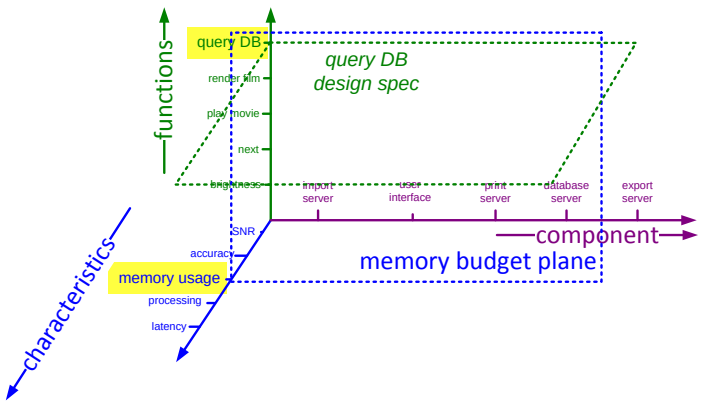


Figure 8.7: Addressing lines or planes at once in the multiple dimensions

8.5 Internal Information Model

The information model as seen from the outside from the system, part of the functional view, is extended into an internal information model. The internal information model is extended with design choices, for instance derived data information is cached to achieve the desired performance. The internal data model might also be chosen to be more generic (for reasons of future extendibility), or less generic (where program code is used to translate the specific internal models in the desired external models.

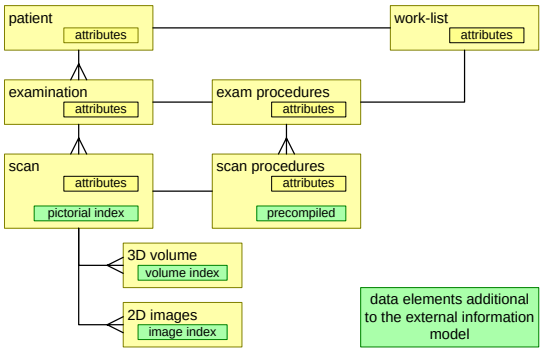


Figure 8.8: Example of a partial internal information model

The internal information model is an important means to decouple parts of the design. The functional behavior of the system is predictable as long as components in the system adhere to the internal information model.

Figure 8.8 shows an example of a part of an information model. In this example several information elements which are derived from the primary data are stored explicitly to improve the response time. The pictorial index, existing of reduced size images, is an example of derived information, which takes some time to calculate. This index is build in the background during import, so that the navigation can use it, which makes the navigation very responsive.

All considerations described in section 7.6, such as the layering hold also for the internal information model.

8.6 Execution architecture

The execution architecture is the run time architecture of a system. The process decomposition plays an important role in the execution architecture. Figure 8.9 shows an example of a process decomposition.

One of the main concerns for process decomposition is concurrency: which concurrent activities are needed or running, how to synchronize these activities. A

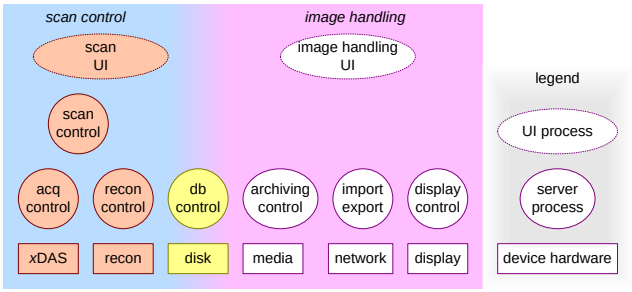


Figure 8.9: Example process decomposition

process or a task of an operating system is a concept which supports asynchronous functionality as well as separation of concerns by providing process specific resources, such as memory. A thread is a lighter construction providing support for asynchronous activities, without the separation of concerns.

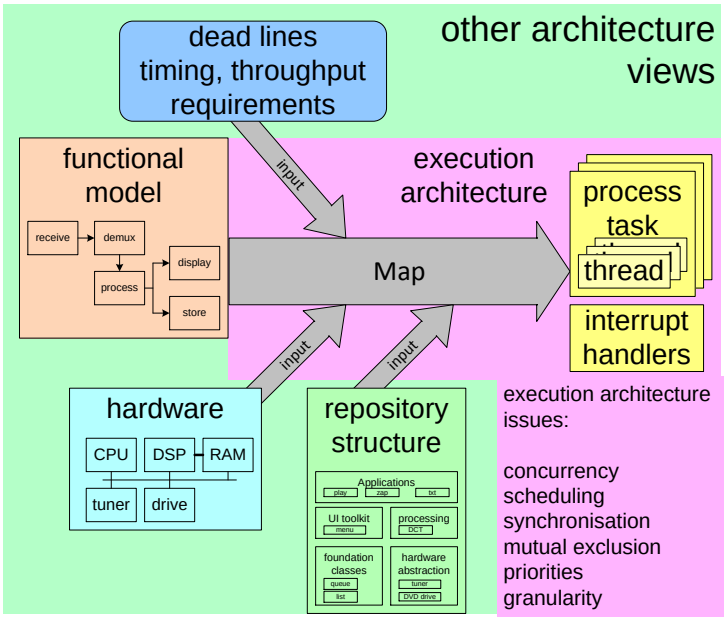


Figure 8.10: Execution architecture

The execution architecture must map the functional decomposition on the process decomposition, taking into account the construction decomposition. In practice many building blocks from the construction decomposition are used in multiple functions mapped on multiple processes. These shared building blocks are aggregated in shared (or dynamic link) libraries. Sharing is advantageous from memory

consumption point of view, some attention is required for the configuration management side[1].

Figure 8.10 shows the role of the execution architecture. The main inputs are the real time and performance requirements at the one hand and the hardware design at the other hand. The functions need to be mapped on processes, threads and interrupt handlers, synchronization method and granularity need to be defined and the scheduling behavior (for instance priority based, which requires priorities to be defined).

[1]The *dll-hell* is not an windows-only problem. Multiple pieces of software sharing the same library can easily lead to version problems, module 1 requires version 1.13, while module 2 requires version 2.11. Despite all compatibility claims it often does not work.

8.7 Performance

The performance of a system can be modeled by complementing models. In figure 8.11 the performance is modelled by a flow model at the top and an analytical model below. The analytical model is entirely parameterized, making it a generic model which describes the performance ratio over the full potential range.

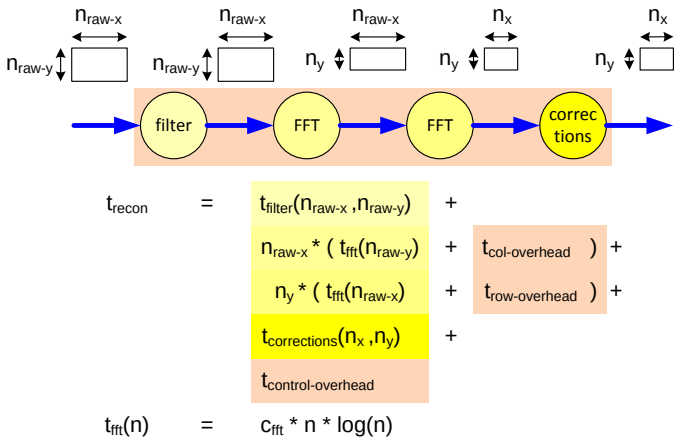


Figure 8.11: Performance Model

Later in the realization view it will be shown that this model is too simplistic, because it focuses too much on the processing and does not take the overheads sufficiently in account.

8.8 Safety, Reliability and Security concepts

The qualities *safety*, *reliability* and *security* share a number of concepts, such as:

- containment (limit failure consequences to well defined scope)
- graceful degradation (system parts not affected by failure continue operation)
- dead man switch (human activity required for operation)
- interlock (operation only if hardware conditions are fulfilled)
- detection and tracing of failures
- black box (log) for post mortem analysis
- redundancy

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.

8.9 Start up and shutdown

In practice insufficient attention is paid to the start up and shutdown of a system, since these are relatively exceptional operations. However the design of this aspect has an impact on nearly all components and functions in the system. It is really an integrating concept. The trend is that these operations become even more entangled with the normal run-time functionality, for instance by run-time downloading, stand-by and other power saving functionality.

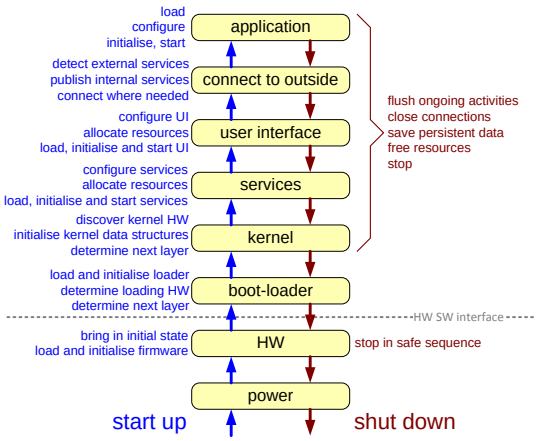


Figure 8.12: Simplified start up sequence

Figure 8.12 shows a typical start up shutdown pattern. The system is brought step by step to higher operational levels. Higher levels benefit from more available support functions, lower levels are less dependent on support functions.

One of the considerations in the design of this system aspect is the impact of failures. The right granularity of operational levels enable coping with exceptions (for example network not available). For shutdown the main question is how power failures or glitches are handled.

8.10 Work breakdown

Project leaders expect a work breakdown to be made by the architect. In fact a work breakdown is again another decomposition, with a more organizational point of view. The work in the different work packages should be cohesive internally, and should have low coupling with other work-packages.

Figure 8.13 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

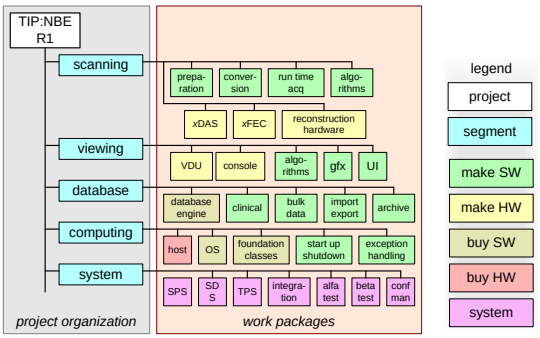


Figure 8.13: Example work breakdown

development work or purchasing work. Both types of work require domain know how, but different skills to do the job.

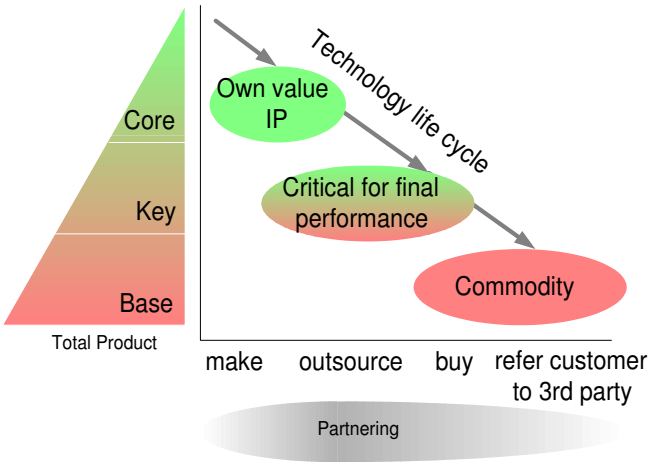


Figure 8.14: Core, Key or Base technology

Make versus Buy is a limited subset of an entire spectrum of approaches. The decision how to obtain the needed technology should be based on where the company intents to add value. A simple reference model to help in making these decisions is based on *core*, *key*, and *base* technology, see figure 8.14.

Core technology is technology where the company is adding value. In order to be able to add value, this technology should be developed by the company itself.

Key technology is technology which is critical for the final system performance. If the system performance can not be reached by means of third party technology

than the company must develop it themselves. Otherwise outsourcing or buying is attractive, in order to focus as much as possible on *core* technology added value. However when outsourcing or buying an intimate partnership is recommended to ensure the proper performance level.

Base technology is technology which is available on the market and where the development is driven by other systems or applications. Care should be taken that these external developments can be followed. Own developments here are de-focusing the attention from the company's core technology.

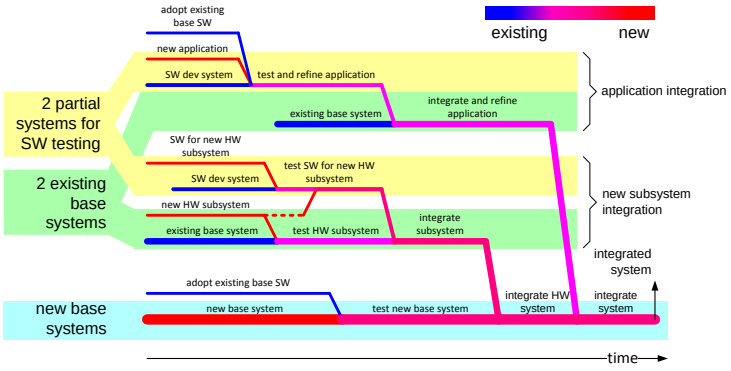


Figure 8.15: Example integration plan, with 3 tiers of development models

Schedules, work breakdown and many technical decompositions are heavily influenced by the integration plan. Integration is time, effort and risk determining part of the entire product creation process. The integration viewpoint must be used regular because of its time, effort and risk impact.

Figure8.15 shows an example integration plan. This plan is centered around 3 tiers of development vehicles:

- SW development systems
- existing HW system
- new HW system

The SW development system, existing from standard clients and servers, is very flexible and accessible from software point of view, but far from realistic from hardware point of view. 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 for software development from an accessible system to a stable HW system to the more real final system. In general integration plans try to avoid stacking

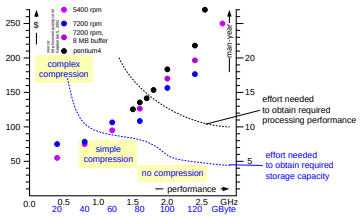
too many uncertainties by looking for ways to test new modules in a stable known environment, before confronting new modules with each other.

8.11 Acknowledgements

Constructive remarks from Peter Bingley, Peter van den Hamer, Ton Kostelijk, William van der Sterren and Berry van der Wijst have been integrated in this document.

Chapter 9

The realization view



9.1 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 explicit

Figure 9.1 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

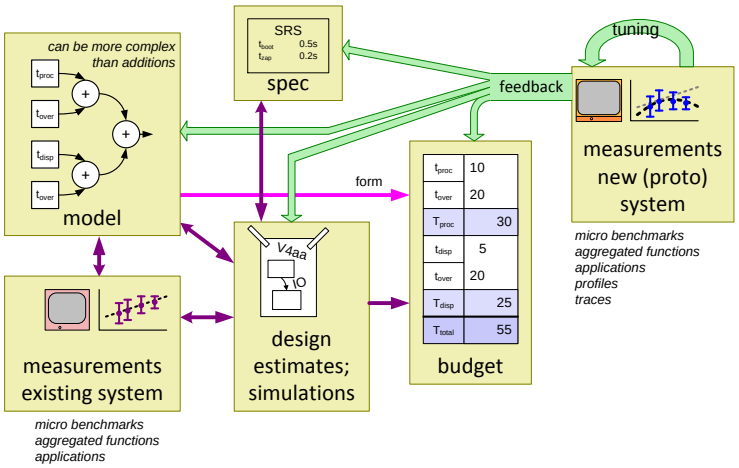


Figure 9.1: Budget based design flow

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.

memory budget in Mbytes	code	obj data	bulk data	total
shared code	11.0			11.0
User Interface process	0.3	3.0	12.0	15.3
database server	0.3	3.2	3.0	6.5
print server	0.3	1.2	9.0	10.5
optical storage server	0.3	2.0	1.0	3.3
communication server	0.3	2.0	4.0	6.3
UNIX commands	0.3	0.2	0	0.5
compute server	0.3	0.5	6.0	6.8
system monitor	0.3	0.5	0	0.8
application SW total	13.4	12.6	35.0	61.0
UNIX Solaris 2.x				10.0
file cache				3.0
total				74.0

Figure 9.2: Example of a memory budget

Figure 9.2 shows an example of an actual memory budget. This budget decomposes the memory in three different types of memory use: code (”read only” memory with the program), object data (all small data allocations for control and bookkeeping purposes) and bulk data (large data sets, such as images, which is explicitly managed to fit the allocated amount and to prevent fragmentation). The

difference in behavior is an important reason to separate in different budget entries. At the other hand the operating system and the system infrastructure provide means to measure these 3 types at any moment, which helps for the initial definition, for the integration and the verification.

The second decomposition direction is the *process*. The number of processes is manageable, processes are related to specific development teams and again the operating system and system infrastructure support measurement at process level.

9.2 Logarithmic views

A logarithmic positioning of requirements and implementation alternatives helps to put these alternatives in perspective. In most designs we have to make design choices which cover a very large dynamic range, for instance from nanoseconds up to hours, days or even years. Figure 9.3 shows an example of requirements and technologies on a logarithmic time axis.

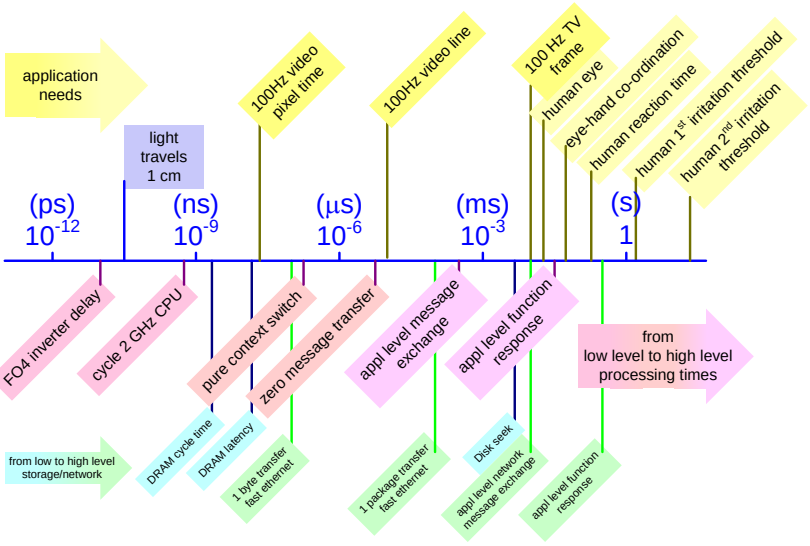


Figure 9.3: Actual timing represented on a logarithmic scale

”Fast” technologies can serve many slow requirements, but often slower technologies offer other benefits, which offset their slowness. ”Slow” technologies offer more flexibility and power, at the cost of performance. For instance real time executive interrupt response time are very short, while reacting in a user task is slower, but can access much more user level data and can interact more easy with other application level functions. Going from real time executive to a ”fat” operating system slows down the interrupt response, with a wealth of other operating system

functionality (networking, storage, et cetera) in return. Again at user process level the response needed is again bigger, with a large amount of application level functionality in return (distribution, data management, UI management, et cetera).

Requirements itself also span such a large dynamic range from very fast (video processing standards determining pixel rates) to much slower (select teletext page).

For every requirement a reasonable implementation choice is needed with respect to the speed. Faster is not always better, a balance between fast enough, cost and flexibility and power is needed.

9.3 Micro Benchmarking

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 9.4 lists a typical set of micro-benchmarks to be performed. The list shows infrequent and often slow operations and frequently applied operations, which are often much faster. This classification implies already a design rule: slow operations should not be performed often[1].

	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 9.4: Typical micro benchmarks for timing aspects

The results of micro-benchmarks should be used with great care, the measurements 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 performance measured in a micro benchmark is often dependent on a number of parameters, such as the length of a transfer. Micro benchmarks are applied with a variation of these parameters, to obtain understanding of the performance as a function of these parameters. Figure 9.5 shows an example of the transfer rate performance as a function of the block size.

For example measuring disk transfer rates will result in this kind of curves, due to a combination of cycle time, seek time and peek transfer rate. This data can

[1] 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 if a slow operation is in fact not very slow and the brute force approach is both affordable as well as extremely straightforward (simple!) then this is better than over-optimizing for efficiency.

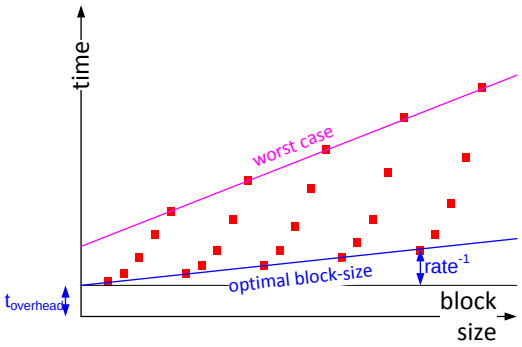


Figure 9.5: The transfer time as function of block size

be used in different ways: the slowest speed can be used, a worst case design, or the buffer size can be tuned to obtain the maximum transfer rate. Both choices are defensible, the conservative choice is costly, but robust, the optimized choice is more competitive, but also more vulnerable.

9.4 Performance evaluation

The performance is conceptually modelled in the conceptual view, which is used to make budgets in the realization view. An essential question for the architect is: Is this design *good*? This question can only be answered if the criteria are known for a *good* design. Obvious criteria are meeting the need and fitting the constraints. However an architect will add some criteria himself, such as balanced and future-proof.

Figure 9.6 shows an example of a performance analysis. The model is shown at the top of the figure, as discussed in the conceptual view. The measurement below the model shows that a number of significant costs have not been included in the original model, although these are added in the model here. The original model focuses on processing cost, including some processing related overhead. However in practice 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).

9.5 Assessment of added value

The implementation should be monitored with respect to its quality. The most common monitoring is problem reporting and fault analysis. The architect should

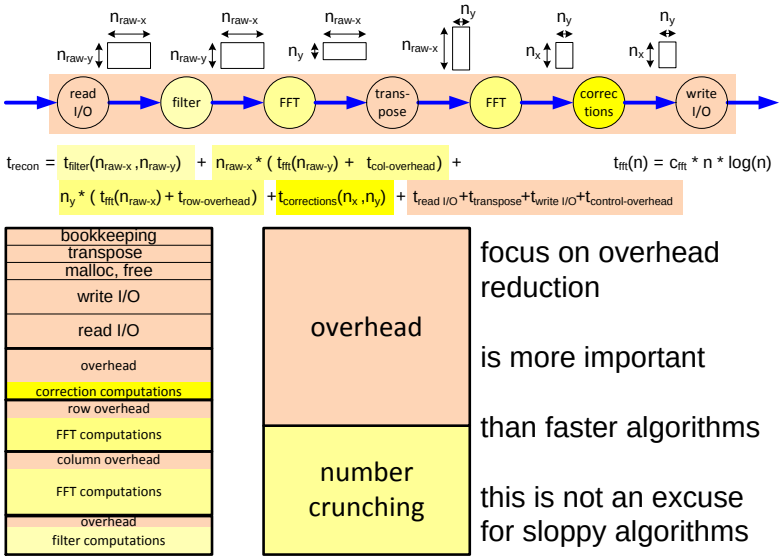


Figure 9.6: Example of performance analysis and evaluation

maintain a quality assessment, based on the implementation itself. This is done by monitoring size and change frequency. In order to do something useful with these metrics some kind of value indicator is also needed. The architect must build up a reference of ”value per size” metrics, which he can use for this a priori quality monitoring.

Figure 9.7 shows an example of a performance cost curve, in this example Pentium4 processors and hard disks. Performance and cost are roughly proportional. For higher performance the price rises faster than the performance, At the low performance side the products level out at a kind of bottom price, or that segment is not at all populated (minimum Pentium4 performance is 1.5 GHz, the lower segment is populated with Celerons, which again don’t go down to any frequency).

The choice of a solution will be based on the needs of the customer. To get grip on these needs the performance need can be translated in the sales value. How much is the customer willing to pay for performance? In this example the customer is not willing to pay for a system with insufficient performance, neither is the customer willing to pay much for additional performance (if the system does the job, then it is OK). This is shown in figure 9.8, with rather non-linear sales value curves.

Another point of view is the development effort. Over-dimensioning of processing or storage capacity simplifies many design decisions resulting in less development effort. In figure 9.9 this is shown by the effort as function of the performance.

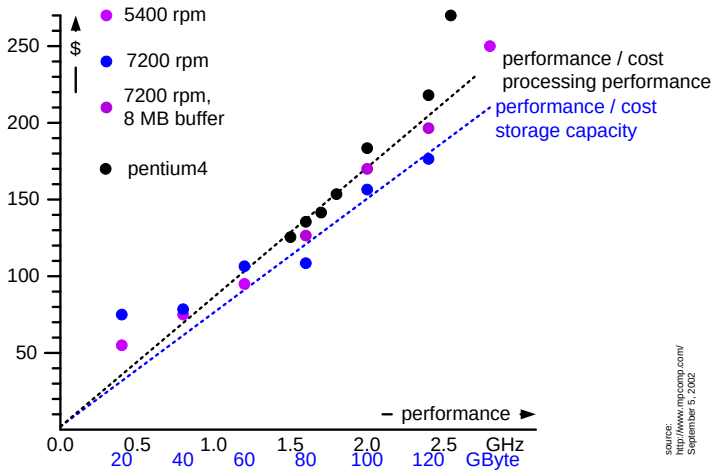


Figure 9.7: Performance Cost, input data

For example for the storage capacity three effort levels can be distinguished: with a low cost (small capacity) disk a lot of tricks are required to fit the application within the storage constraint, for instancing by applying complex compression techniques. The next level is for medium cost disks, which can be used with simple compression techniques, while the expensive disks don't need compression at all.

Figure 9.10 show that many more issues determine the final choice for the "right" cost/performance choice: the capabilities of the rest of the system, the constraints and opportunities in the system context, trade-offs with the image quality. All of the considerations are changing over time, today we might need complex compression, next year this might be a no-brainer. The issue of effort turns out to be related with the risk of the development (large developments are more risky) and to time to market (large efforts often require more time).

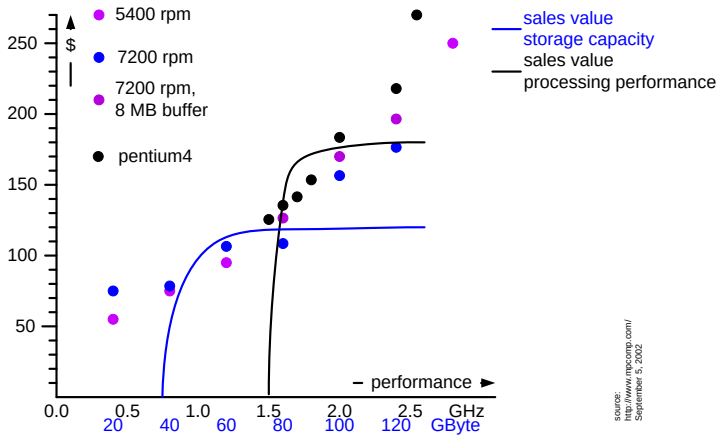


Figure 9.8: Performance Cost, choice based on sales value

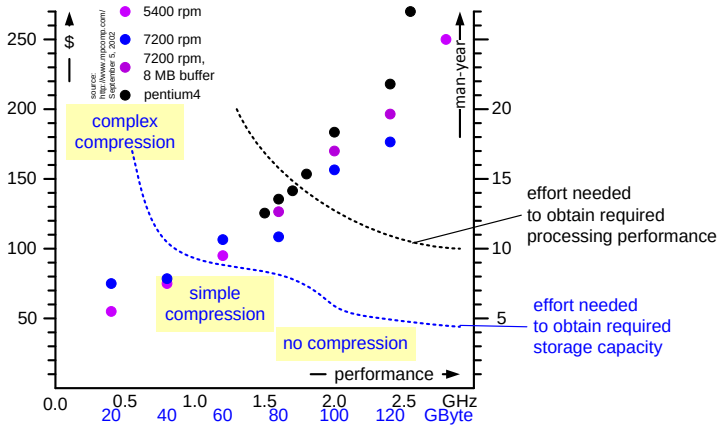


Figure 9.9: Performance Cost, effort consequences

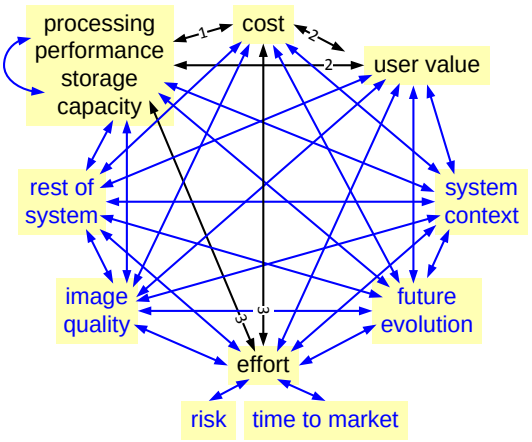


Figure 9.10: But many many other considerations

9.6 Safety, Reliability and Security Analysis

Qualities such as safety, reliability and security depend strongly on the actual implementation. Specialized engineering disciplines exists for these areas. These disciplines have developed their own methods. One class of methods relevant for system architects is the class of analysis methods, which start with a (systematic) brainstorm, see figure 9.11.

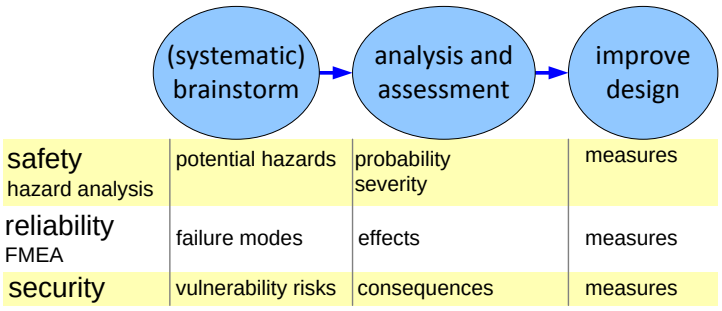


Figure 9.11: Analysis methods for safety, reliability and security

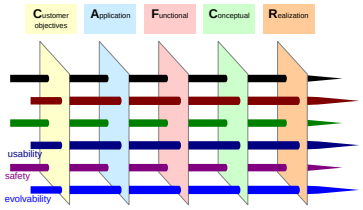
Walk-through is another effective assessment method. A few use cases are taken and together with the engineers the implementation behavior is followed for these cases. The architect will especially assess the understandability and simplicity of the implementation. An implementation which is difficult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

9.7 Acknowledgements

William van der Sterren and Peter van den Hamer invented the nice phrase micro benchmarking.

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.

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 [9].

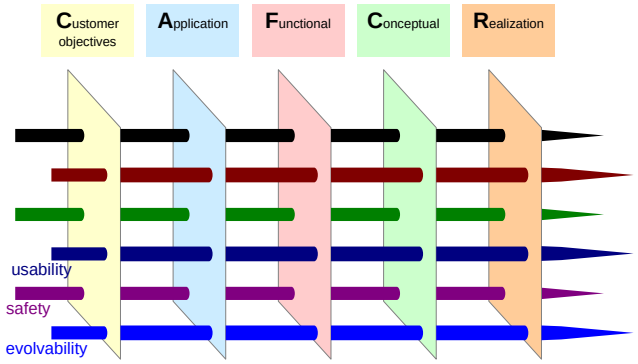


Figure 10.1: The quality needles are generic integrating concepts through the 5 CAFCR views

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 [3]. 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.

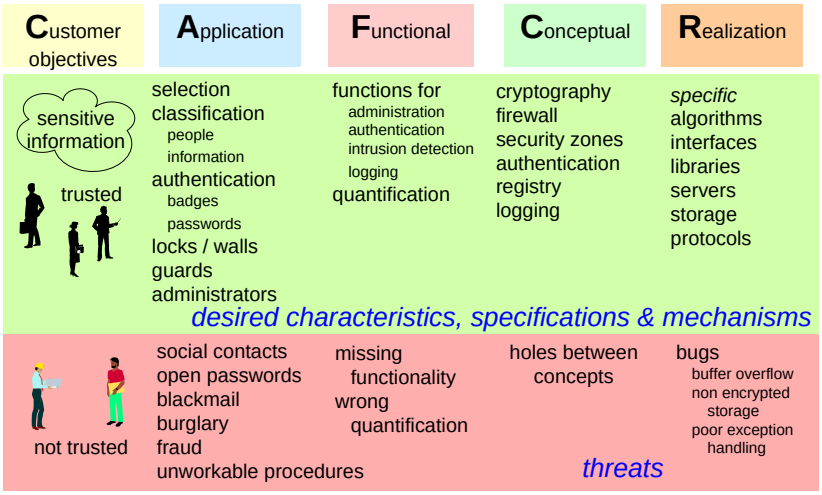


Figure 10.2: Example security through all views

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 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

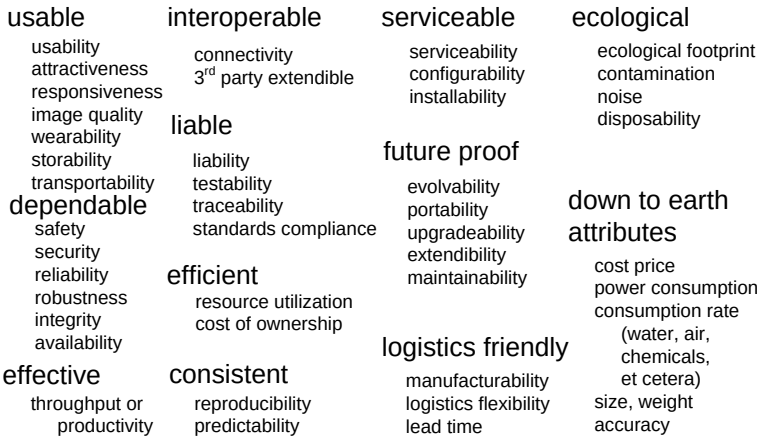


Figure 10.3: Checklist of qualities

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.

availability 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

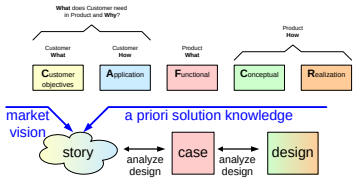
accuracy

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 How To



11.1 Introduction

Starting a new product definition often derails in long discussions about generic specification and design issues. Due to lack of reality check these discussions are very risky, and often way too theoretical. 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.

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.

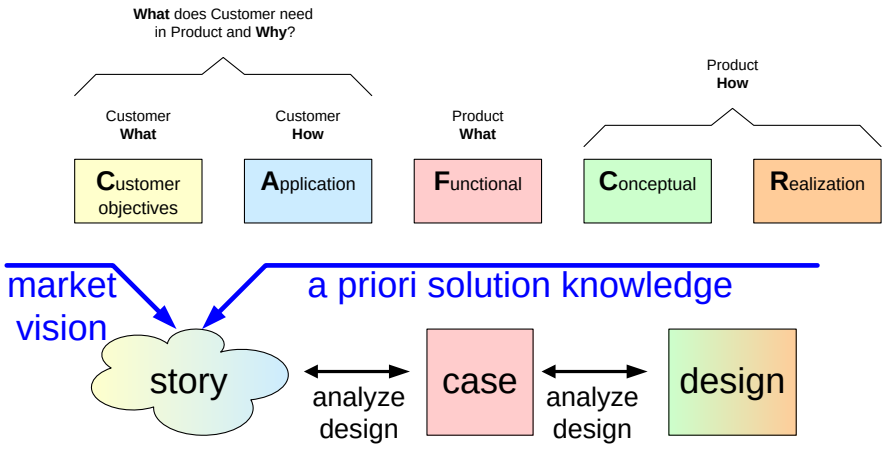


Figure 11.1: From story to design

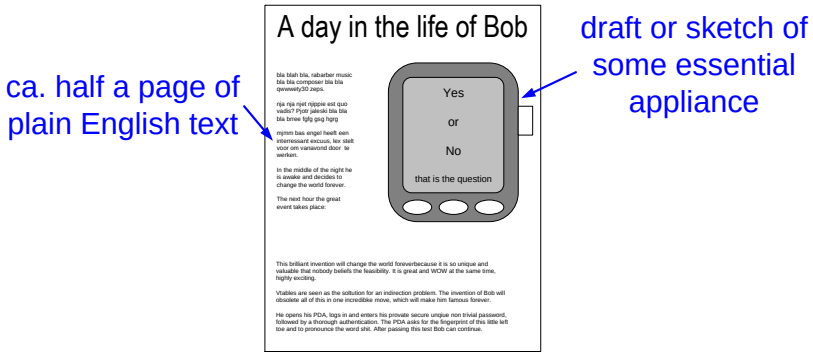


Figure 11.2: Example story layout

11.2 How to Create a Story?

A story is a short single page story, as shown in Figure 11.2, 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, animations, 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 useful to understand a wide context, but leaves many details unexplored. An 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.

The *size* of the story is rather critical. Only short stories serve the purpose of discussion catalyst. At the same time all stakeholders have plenty of questions that can be answered by extending the story. It is recommended to really limit the size of the story. One way of doing this is by consolidating additional information in a separate document. For instance, in such a document the point of the story in customer perspective, the purpose of the story in the technology exploration, and the implicit assumptions about the customer and system context can be documented.

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 a story such that it meets all the criteria or to 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.

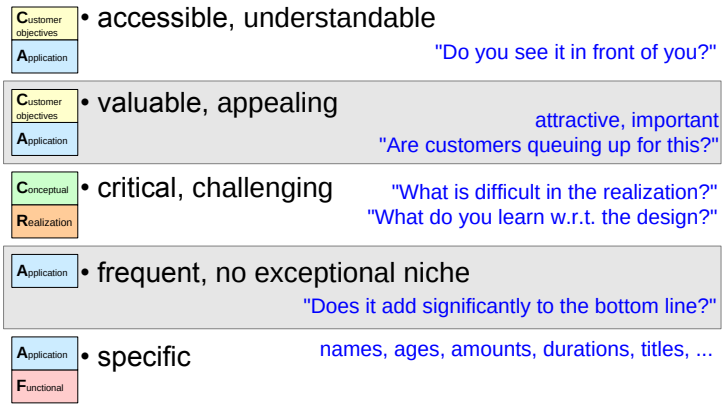


Figure 11.3: criteria for a good story

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

Important, valuable, appealing, attractive 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 an useful level of importance.

Critical, challenging The purpose of the story is to learn, define, 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, than 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.

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.

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.

Specificity can be achieved in social, cultural, emotional or demographic details, such as names, ages, and locations. "Eleven year old Jane in Shanghai" is a very different setting than "Eighty two year old John in an Amsterdam care center". Note that these social, cultural, emotional or demographic details also help in the engagement of the audience. More analytical stories can be too "sterile" for the audience.

Another form of specificity is information that helps to quantify. For example, using "Doctor Zhivago" as movie content sets the duration to 200 minutes. Stories often need lots of these kinds of detail to facilitate later specification and design analysis. When during the use of the story more quantification is needed, then the story can be modified such that it provides that information.

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**)

Story writers sometimes want to show multiple possibilities and describe somewhere an escaping paragraph to fit in all the potential goodies (Aardvark works, sleeps, eats, swims et cetera, while listening to his Wow56). Simply leave out such an paragraph, it only degrades the focus and value of the story.

11.5 Example Story

Figure 11.4 shows an example of a story for hearing aids. The story first discusses the problem an elderly lady suffers from due to imperfect hearing aids. The story continues with postulated new devices that helps her to participate again in an active social life.

Figure 11.5 shows for the value and the challenge criteria what this story contributes.

Betty is a 70-year-old woman who lives in Eindhoven. Three years ago her husband passed away, and since then, she lives in a home for the elderly. Her two children, Angela and Robert, come and visit her every weekend, often with Betty's grandchildren Ashley and Christopher. As with so many women of her age, Betty is reluctant to touch anything that has a technical appearance. She knows how to operate her television, but a VCR or even a DVD player is way to complex.

When Betty turned 60, she stopped working in a sewing studio. Her work in this noisy environment made her hard-of-hearing with a hearing-loss of 70dB around 2kHz. The rest of the frequency spectrum shows a loss of about 45dB. This is why she had problems understanding her grandchildren and why her children urged her to apply for hearing aids two years ago. Her technophobia (and her first hints or arthritis) inhibit her from changing her hearing aids' batteries. Fortunately, her children can do this every weekend.

This Wednesday, Betty visits the weekly Bingo afternoon in the meeting place of the old-folk's home. It's summer now and the tables are outside. With all those people there, it's a lot of chatter and babble. Two years ago, Betty would never go to the bingo: "I cannot hear a thing when everyone babbles and clatters with the coffee cups. How can I hear the winning numbers?!". Now that she has her new digital hearing instruments, even in the bingo cacophony, she can understand everyone she looks at. Her social life has improved a lot, and she even won the bingo a few times.

source: Roland Mathijssen
Embedded Systems Institute
Eindhoven

That same night, together with her friend Janet, she attends Mozart's opera The Magic Flute. Two years earlier, this would have been one big low rumbly mess, but now she even hears the sparkling high piccolos. Her other friend Carol never joins their visits to the theaters. Carol also has hearing aids; however, hers only "work well" in normal conversations. "When I hear music, it's as if a butcher's knife cuts through my head. It's way too sharp!". So Carol prefers to take her hearing aids out, missing most of the fun. Betty is so happy that her hearing instruments simply know where they are and adapt to their environment.

Figure 11.4: Example of a story

11.6 Acknowledgements

Within the IST-SWA research group lots of work has been done on scenario and story based architecting, amongst others by Christian Huiban and Henk Obbink. Rik Willems helped me to sharpen the *specificity* criterium. Melvin Zaya provided feedback on the importance of the story context and the "point" of the story. Roland Mathijssen provided an example story.

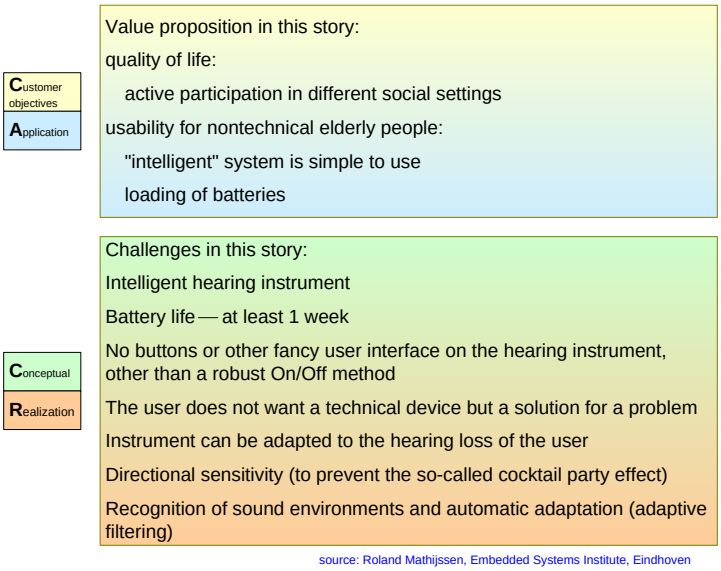
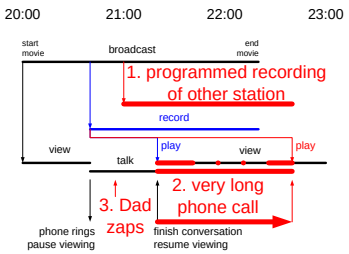


Figure 11.5: Value and Challenges in this story

Chapter 12

Use Case How To



12.1 Introduction

The *use case* technique is frequently used in specification and design, for example RUP[20] is advocating it is as tool. The power of use cases is that they put specifications in *user* perspective. Use cases facilitate *analysis and design* and *verification and testing* by providing concrete inputs. In practice the following problems arise when use cases are used:

- designers apply the technique too local, for example software only
- the use cases are limited to functionality, ignoring quantified information

The purpose of this article is to explain the use case technique at system level, applied in a multi-disciplinary way. We will show how to obtain understanding from use cases of typical use and how to analyze the specification and design for worst cases and boundary conditions.

12.2 Example Personal Video Recorder

We use *time shift recording* as a use case of desired user functionality. Figure 12.1 shows the concurrent activities that occur when straightforward time shifting is

used. In this example the user is watching a movie, which is broadcasted via conventional means. After some time he is interrupted by the telephone. In order to be able to resume the viewing of the movie he pauses the viewing, which starts invisible the recording of the remainder of the movie. Sometime later he resumes viewing where he left of, while in the background the recording of the not yet finished movie continues.

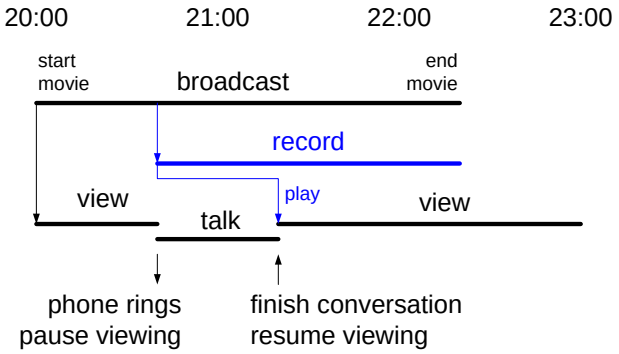


Figure 12.1: Example use case Time Shift recording

In this simple form (pause/resume) this function provides freedom of time to the user. This appears to be very attractive in this interaction modus. However when such an appliance is designed limits out of the construction world pop up, which intrude in the user experience. The list below shows a number of construction limits, which are relevant for the external behavior of the appliance.

- number of tuners
- number of simultaneous streams (recording and playing)
- amount of available storage
- management strategy of storage space

Construction limits, but also more extensive use cases, see figure 12.2, show how the intrinsic simple model can deteriorate into a more complex interaction model. Interference of different user inputs and interference of appliance limitations compromise the simplicity of the interaction model.

12.3 The use case technique

Figure 12.3 shows what elements should be present in a use case. The purpose of the use case is to make the specification clear of functionality or behavior of the

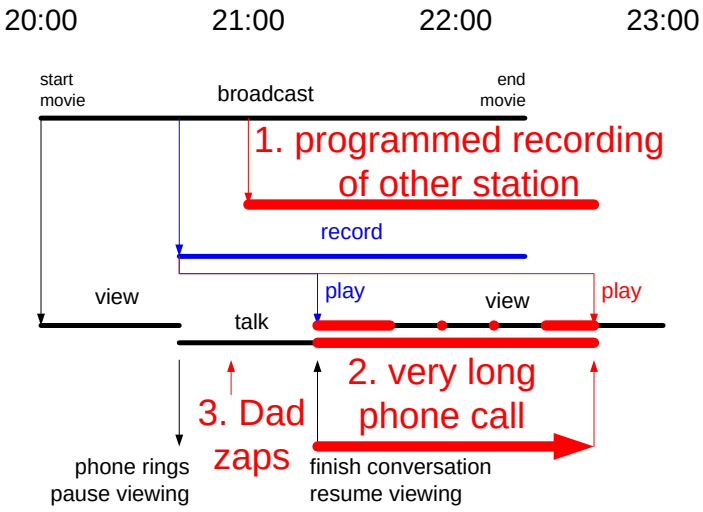


Figure 12.2: What if conflicting events happen during the pause interval?

system and what the desired non-functional requirements (or qualities) are. The use case technique can also be applied for technical interfaces, where the use case illustrates the specification from the perspective of the using system.

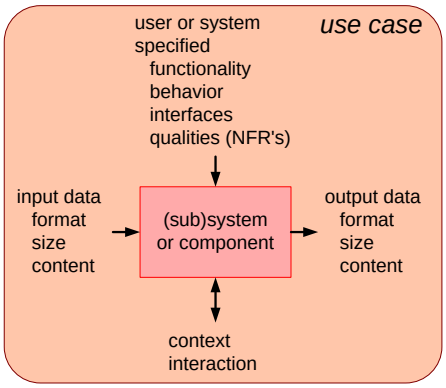


Figure 12.3: Content of a Use Case

The use case also described the input of the (sub)system in terms of format, size and content. The expected outputs are described with the same attributes. Then the interaction with the context of the system must be described, as far as relevant for this specific use case.

Figure 12.4 shows the elements of two use cases also from the personal video recorder domain. At the left hand a typical use case is presented: watching a

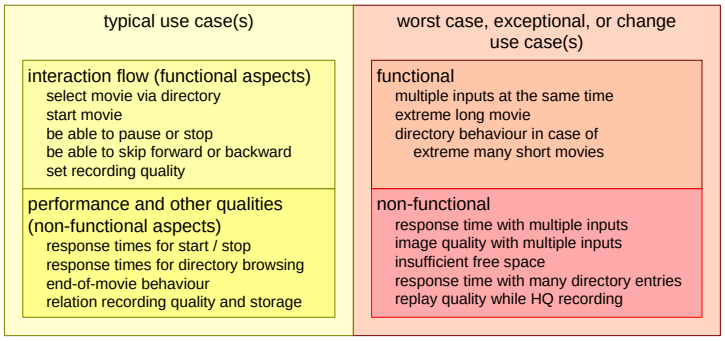


Figure 12.4: Example personal video recorder use case contents

pre-recorded movie. The right side shows the elements of examples of worst cases or boundary cases. At the bottom of both use cases the possible quantification is shown. For example in the typical case user response times can be specified or image quality in relation to required storage capacity. For worst cases many more numbers are relevant for design. These worst case numbers arise from the confrontation of the extremes of the user needs with the quantification of the technology limitations.

12.4 Example URF examination

This use case example focuses on the quantification aspect. Figure 14.7 shows the typical case for URF (Universal Radiography Fluoroscopy) examinations when used image intestines. 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).

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 light-box. 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 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

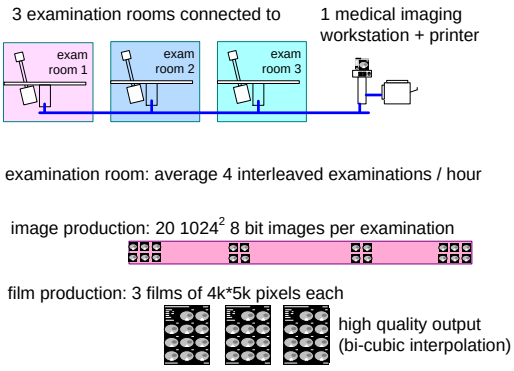


Figure 12.5: Typical case URF examination

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.

The amount of worst case and boundary situations is very large, so selection of relevant ones is needed. The danger of working out too many use cases is that all this work is also transformed in realizations and verifications resulting in excessive implementation efforts. Reduction of the amount of use cases can be done in steps, by replacing several detailed use cases by one slightly more generalized use case. The consequence of such a transformation is that also the design is simplified, where the focus should be on excellent performance of typical use cases and acceptable performance and behavior for worst cases and exceptional cases.

12.5 Summary

Figure 12.7 summarizes the recommendations for use cases. A common pitfall is that people describe use cases at single function level. The usage aspect disappears in this way and many small problems become invisible. Therefor a good use case combines several functions into one user activity. The use case should be quantified to make it useful for design, analysis and verification. The amount of use cases should be limited.

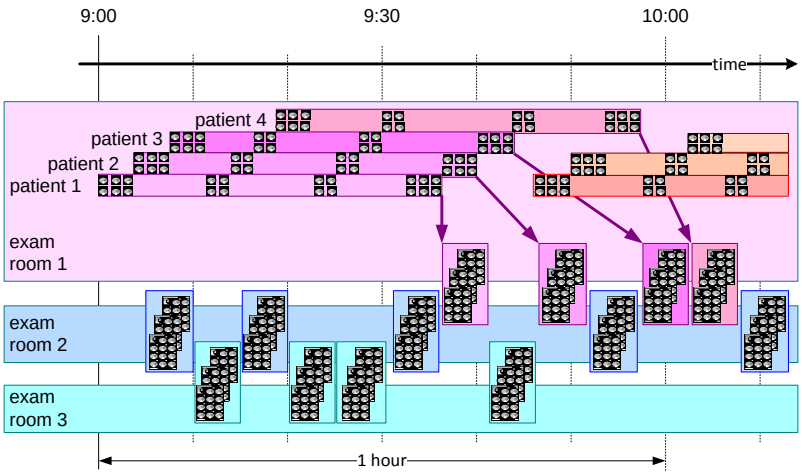


Figure 12.6: Timing of typical URF examination rooms

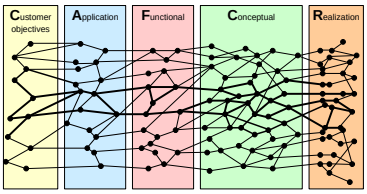
+ combine related functions in one use case
- do not make a separate use case for every function
+ include non-functional requirements in the use cases

+ minimise the amount of required *worst case* and *exceptional use cases*
- excessive amounts of use cases propagate to excessive implementation efforts
+ reduce the amount of these use cases in steps
- a few well chosen *worst case* use cases simplifies the design

Figure 12.7: Recommendations for working with use cases

Chapter 13

Threads of Reasoning



13.1 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 13.2 provides an overview of the approach and gives a short introduction to each step. Section 13.3 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 13.4 explains how the threads of reasoning fit in the complete method.

13.2 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 **??**. 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 **??** and **??**. 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).

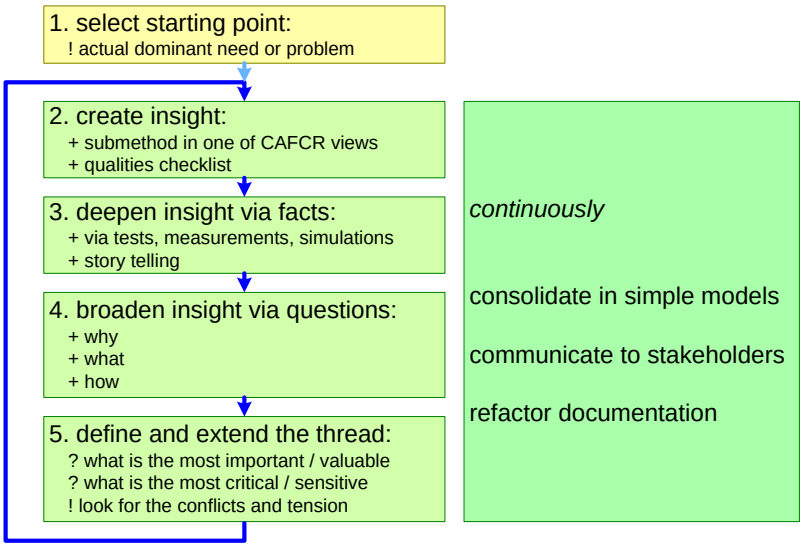


Figure 13.1: Overview of reasoning approach

Figure 13.1 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.

13.2.1 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.

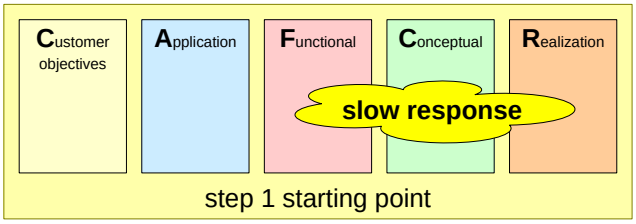


Figure 13.2: Example of a starting point: a slow system response discussed from the designer's viewpoint

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 13.2 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.

13.2.2 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 13.3 shows the *response time model* as potential submethod.

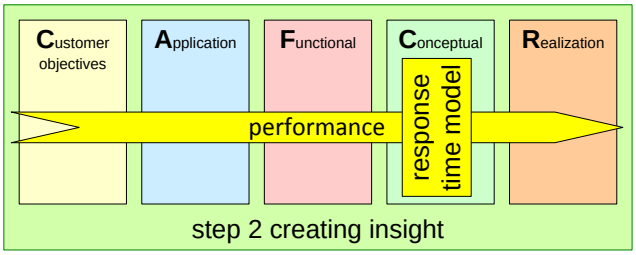


Figure 13.3: Example of creating insight: to study the required performance a response model of the system is made

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 13.3 shows the *performance* as quality to be used to understand the response time issue.

13.2.3 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 13.4.

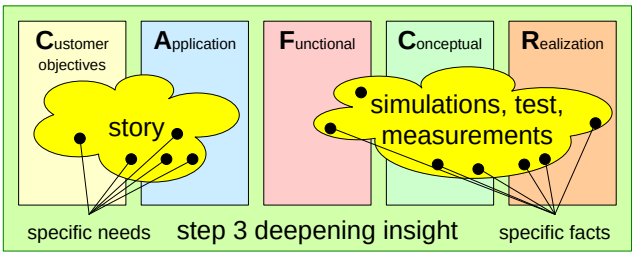


Figure 13.4: Deepening the insight by articulating specific needs and gathering specific facts by simulations, tests and simulations

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.

13.2.4 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 **??** and shown in Figure 13.5.

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?

13.2.5 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

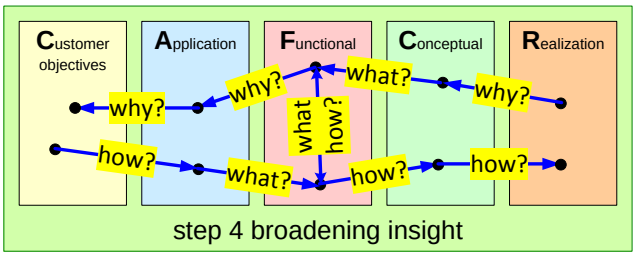


Figure 13.5: Broadening the insight by repeating why, what and how questions

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

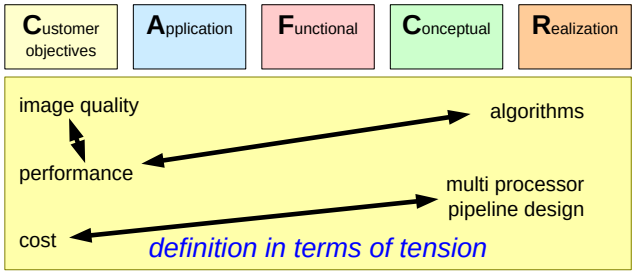


Figure 13.6: Example definition of the thread in terms of tension for a digital TV

Figure 13.6 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 13.6 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

[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.

processors. The cost of the number crunching capacity easily exceeds the cost target.

13.3 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.

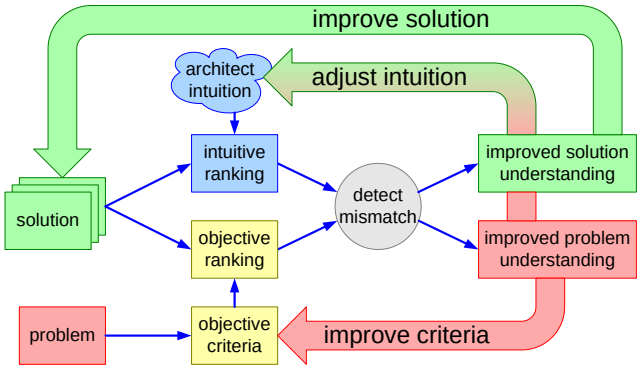


Figure 13.7: Reasoning as a feedback loop that combines intuition and analysis

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

During the reasoning a network of related issues emerges, as shown in Figure 13.8. Figure 13.8 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

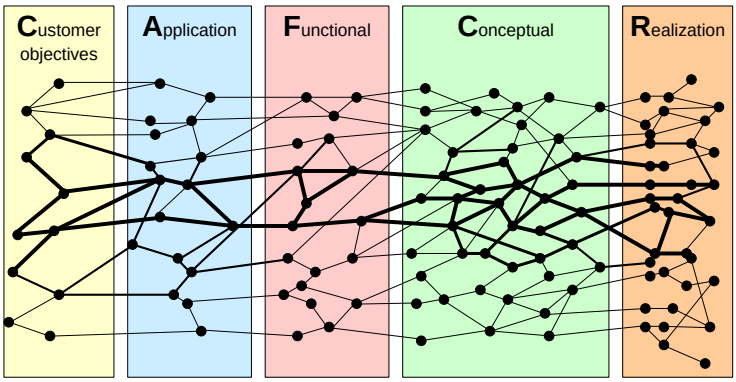


Figure 13.8: One *thread of reasoning* showing related issues. The line thickness is an indication for the weight of the relation.

problem(s), without drowning in the real world complexity.

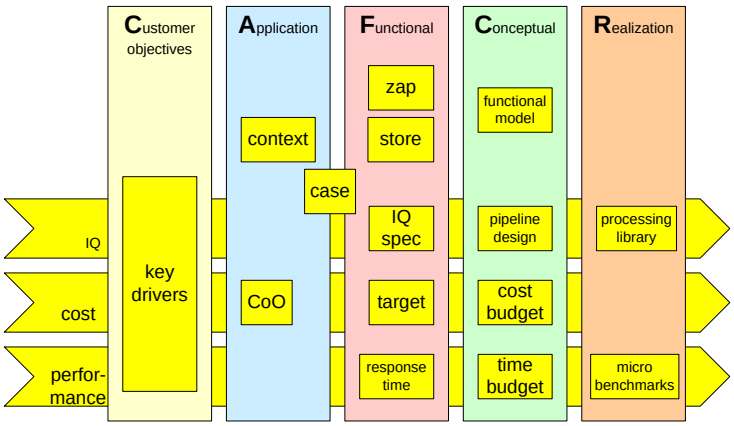


Figure 13.9: Example of the documentation and communication for a digital TV. The thread is documented in a structured way, despite the chaotic creation path. This structure emerges after several iterations.

A continuous concern is to communicate with the stakeholders and to consolidate the findings, for instance in documentation. Figure 13.9 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 13.6. This single thread of reasoning addresses three key drivers as shown in Figure 13.9: *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*.

13.4 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 13.7. 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.

13.5 Summary

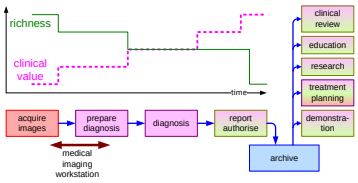
The reasoning approach is a means to integrate the CAFCR views and the qualities to design a system that fits entirely in the customer needs. The *threads of reasoning* approach is described by five steps. The result can be visualized as a graph of many related customer needs, specification issues and design issues. In this graph the core reasoning can be indicated around a limited set of key drivers or quality needs. In Chapter 18 the graph will be visualized for the Medical Imaging Workstation case.

Part III

Medical Imaging Case description

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.

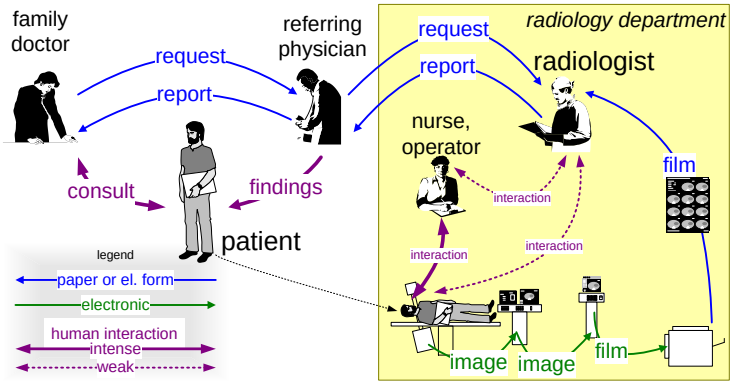


Figure 14.1: The clinical context of the radiology department, with its main stakeholders

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. 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.

The insurance company and the government have a strong interest in cost

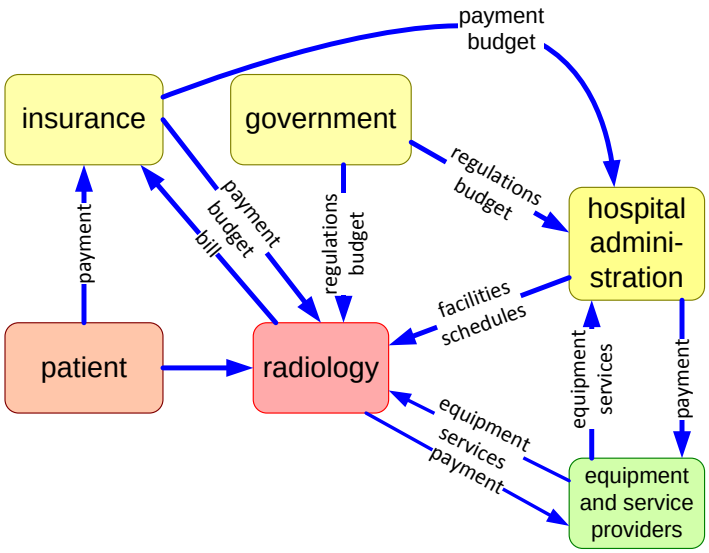


Figure 14.2: The financial context of the radiology department

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 [5]. 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 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

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

medical imaging workstation then an analysis is needed of the greatest common denominator to be able to define a rigorous interface.

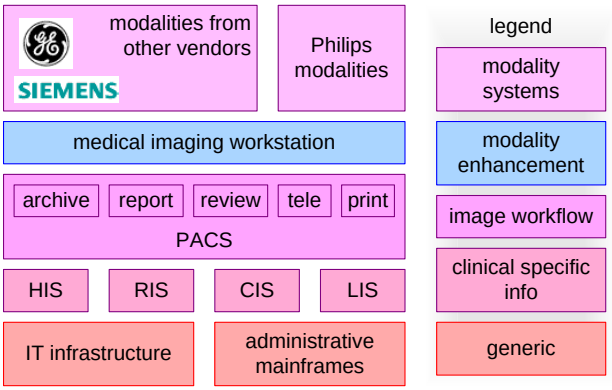


Figure 14.3: Application layering of IT systems

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 shows a reworked copy of the reference model for image handling functions from the "PACS Assessment Final Report", September 1996 [4]. 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

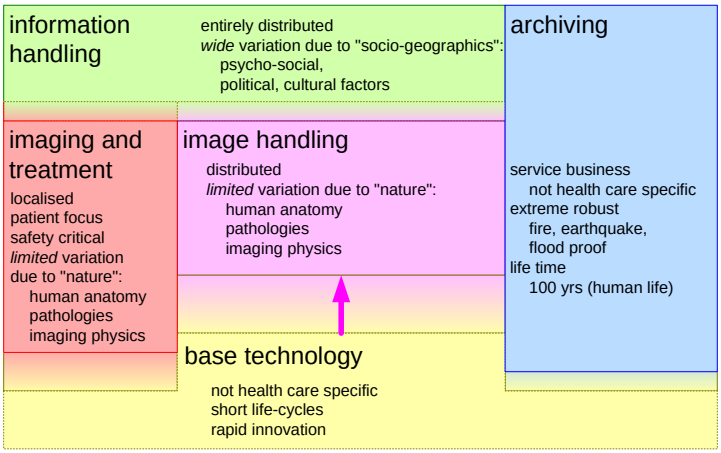


Figure 14.4: Reference model for health care automation

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.

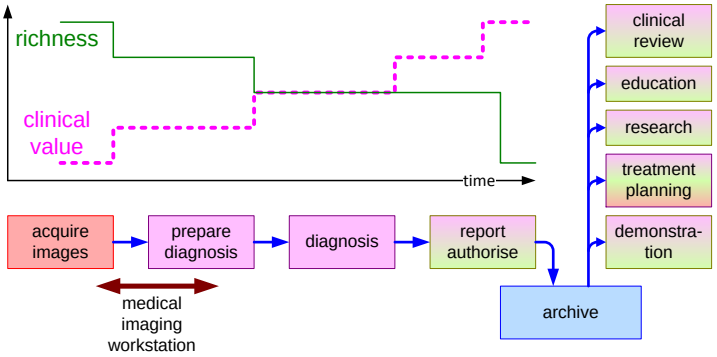


Figure 14.5: Clinical information flow

Figure 14.5 comes from the same report [4] 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 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

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

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.

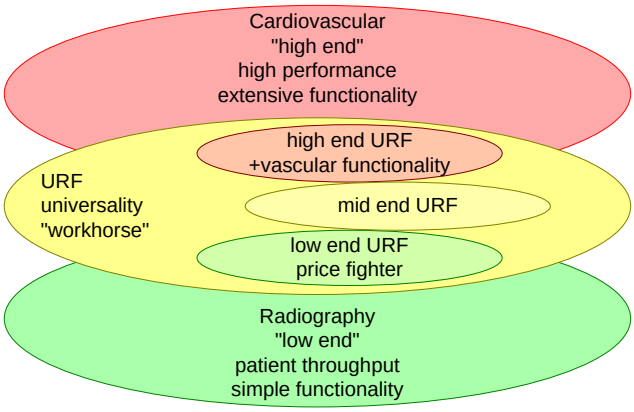


Figure 14.6: URF market segmentation

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).

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 $24 * 30 cm^2$. 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

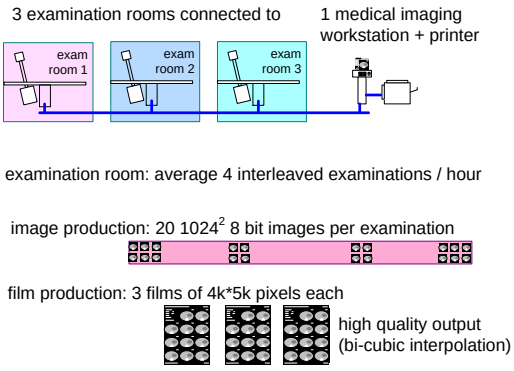


Figure 14.7: Typical case URF examination

printed on 3 film sheets. Image quality of the film sheets is crucial, which translates into the use of bi-cubic interpolation.

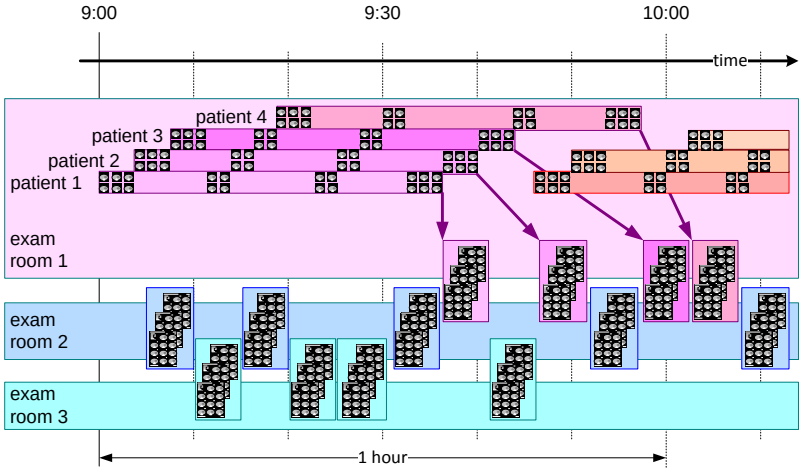


Figure 14.8: Timing of typical URF examination rooms

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 **??**. 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.

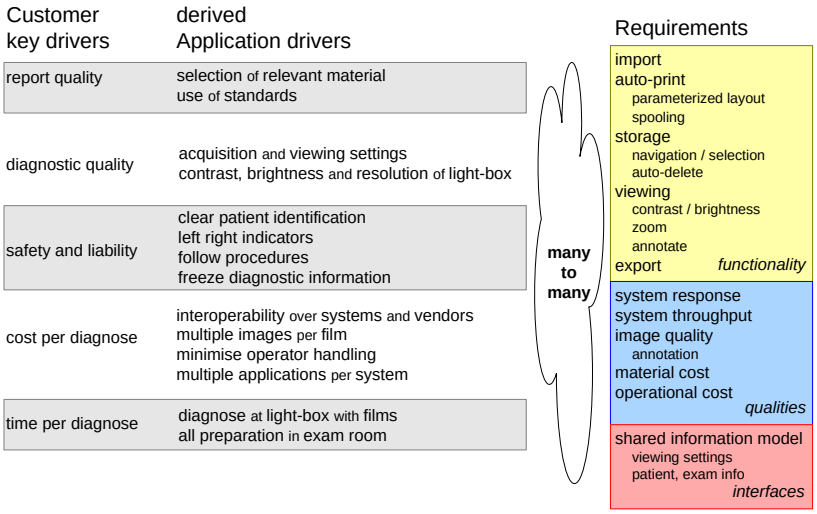


Figure 14.9: Key drivers, application drivers and requirements

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.

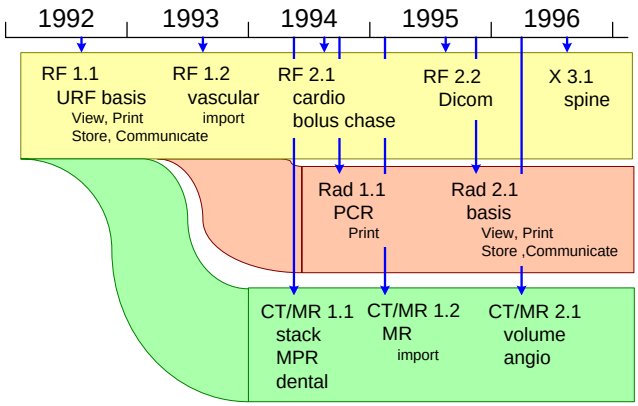


Figure 14.10: Retrospective functionality roadmap

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.

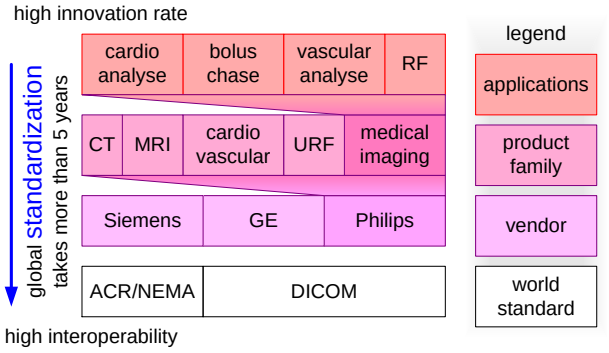


Figure 14.11: Information model, standardization for interoperability

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.

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.

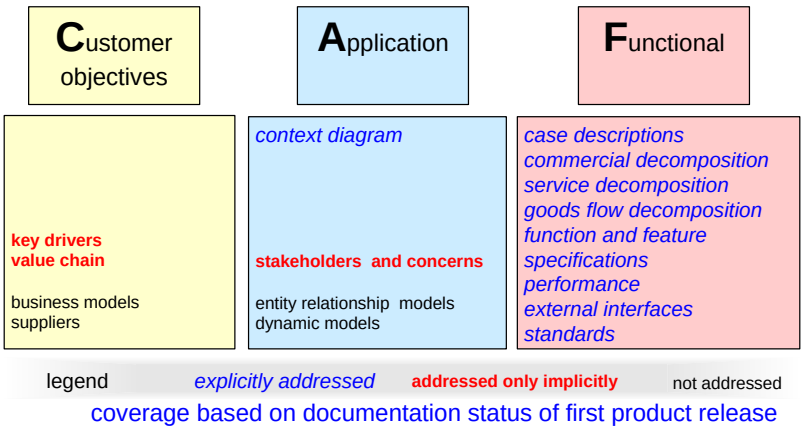
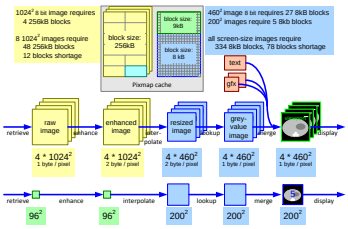


Figure 14.12: Coverage of submethods of the CAF views

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.

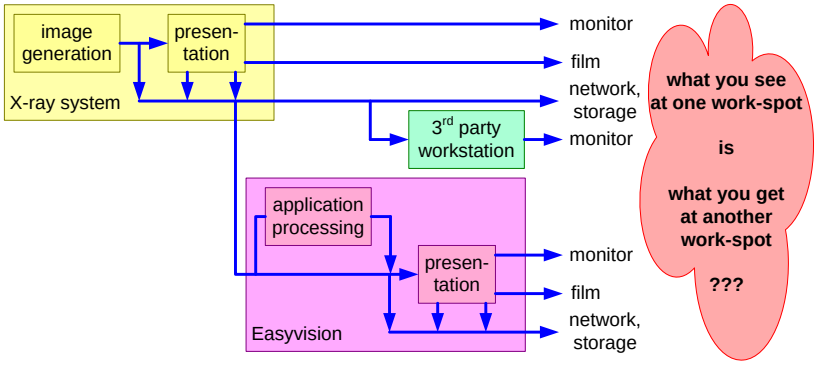


Figure 15.1: The user expectation is that an image at one work-spot looks the same as at other work-spots. This is far from trivial, due to all data paths and the many parties that can be involved

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 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.

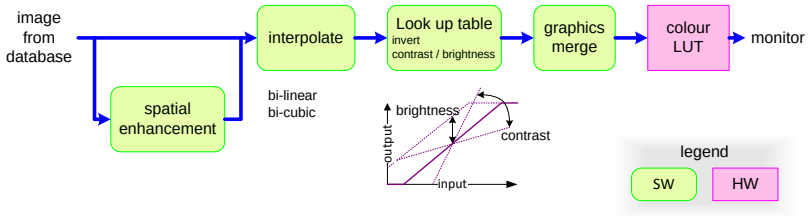


Figure 15.2: The standard presentation pipeline for X-ray images

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.

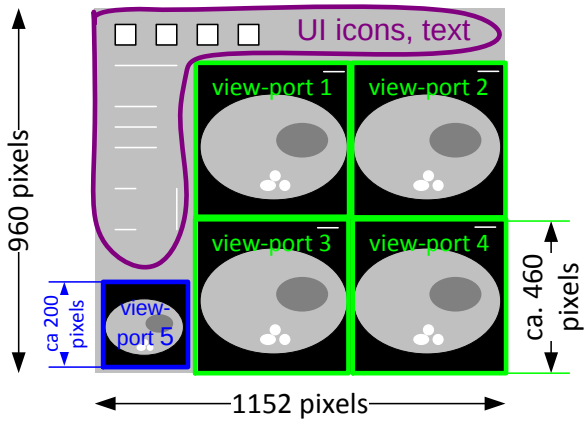


Figure 15.3: Quadruple view-port screen layout

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.

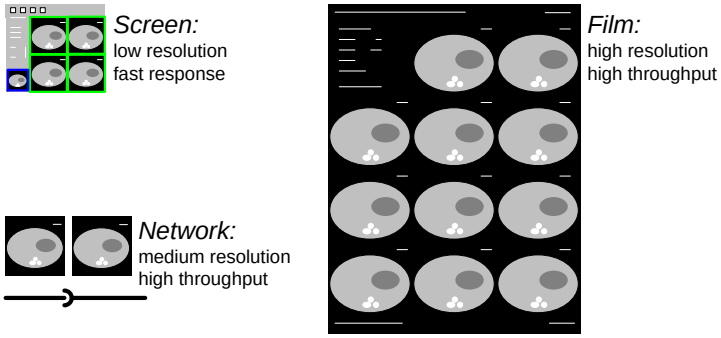


Figure 15.4: Rendered images at different destinations

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 provides *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* [6] 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 shows four types of processes: *client processes*, *server processes* *database process*, and *operational processes*. A client interacts with a user (remote

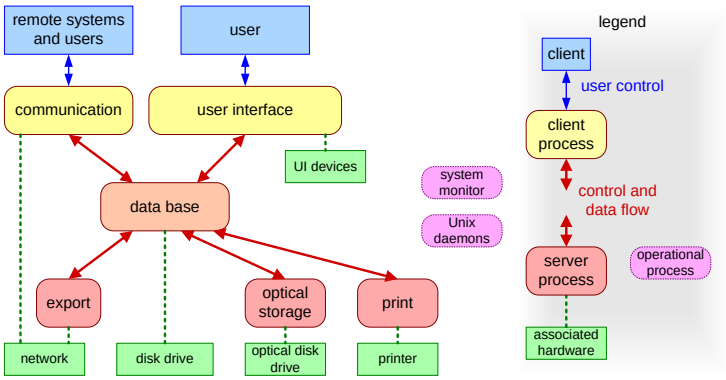


Figure 15.5: Software processes or tasks running concurrently in Easyvision

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.

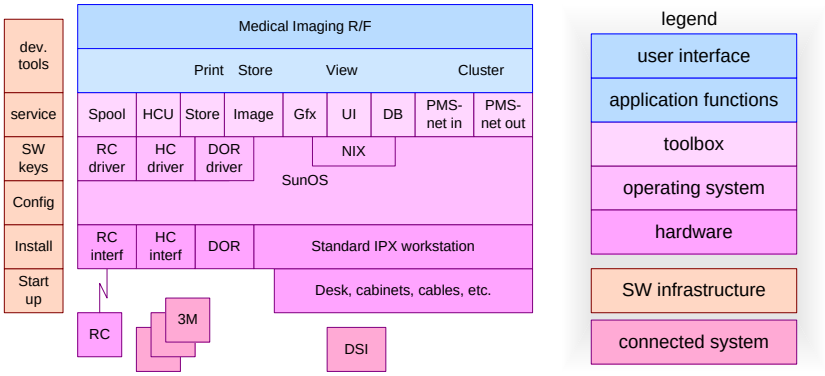


Figure 15.6: Simplified layering of the software

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.

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.

	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

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.

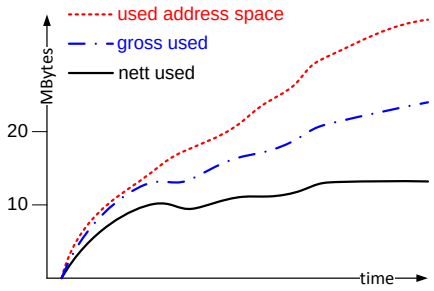


Figure 15.8: Memory fragmentation increase. The difference between gross used and nett used is the amount of unusable memory due to fragmentation

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.

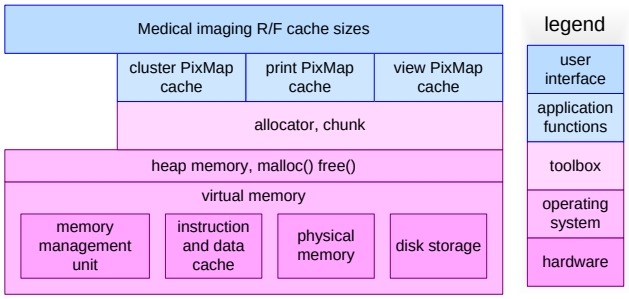


Figure 15.9: Cache layers at the corresponding levels of Figure 15.6

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 malloc() 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.

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

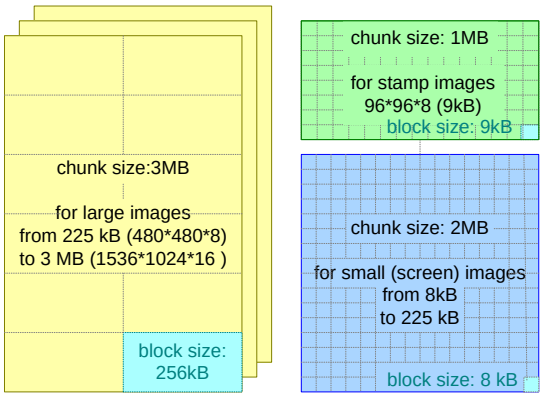


Figure 15.10: Memory allocators as used for bulk data memory management in Easyvision RF

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.

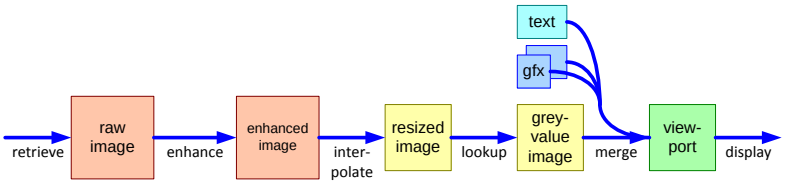


Figure 15.11: Intermediate processing results are cached in an application level cache

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.

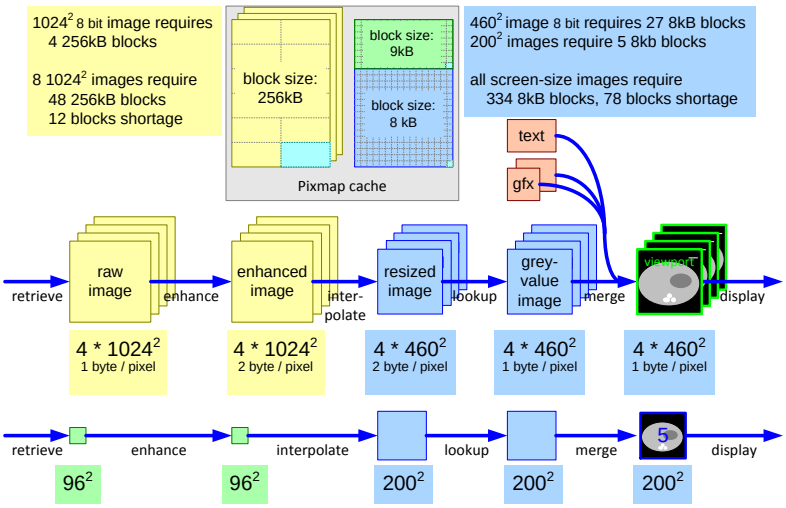


Figure 15.12: Example of allocator and cache use. In this use case not all intermediate images fit in the cache, due to a small shortage of blocks. The performance of some image manipulations will be decreased, because the intermediate images will be regenerated when needed.

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 process, 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

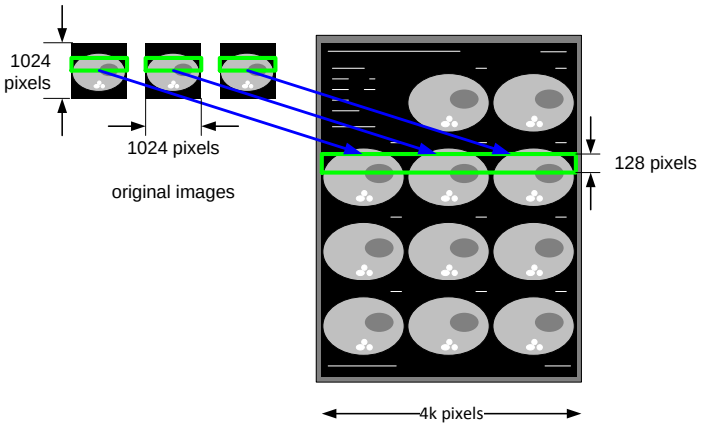


Figure 15.13: Print server is based on different memory strategy, using bands

not provide any significant value for this server usage, because the image data flow is straightforward and predictable.

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.

For background tasks a CPU budget is used, expressed in CPU seconds per

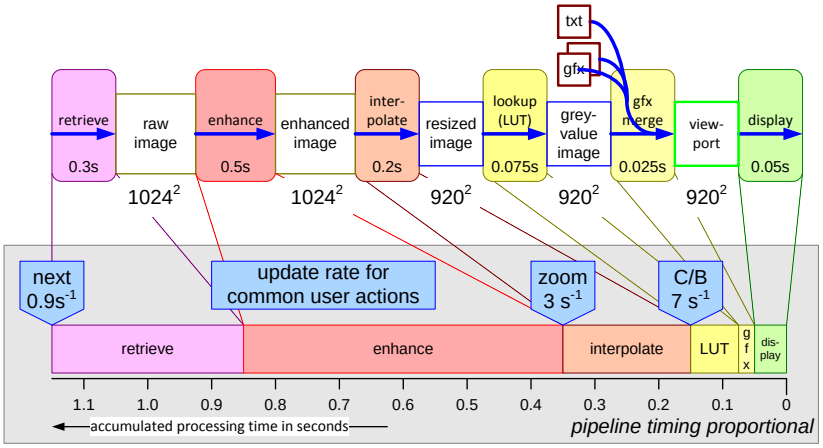


Figure 15.14: The CPU processing times are shown per step in the processing pipeline. The processing times are mapped on a proportional time line to visualize the viewing responsiveness

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 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

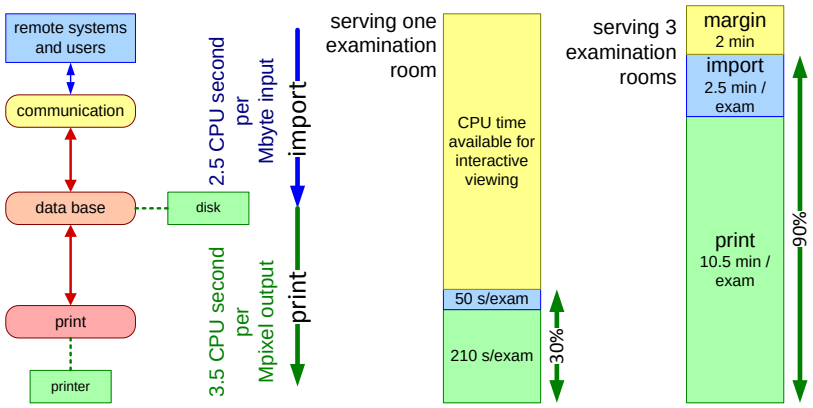


Figure 15.15: Server CPU load. For a single examination room sufficient CPU time is left for interactive viewing. Serving three examination rooms fits in 90% of the available CPU time.

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.

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

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

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.

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

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

Figure 15.17: Overview of benchmarks and other measurement tools

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.

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 documentation of the first release. This coverage is high for most submethods. Safety, reliability and

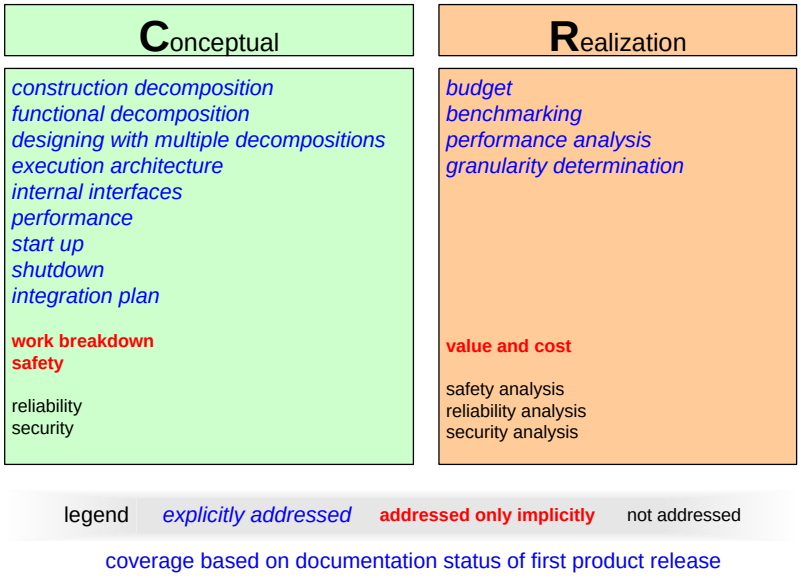
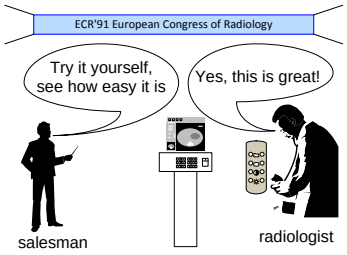


Figure 15.18: Coverage of submethods of the CR views

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 **??**, 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 **??**.

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.

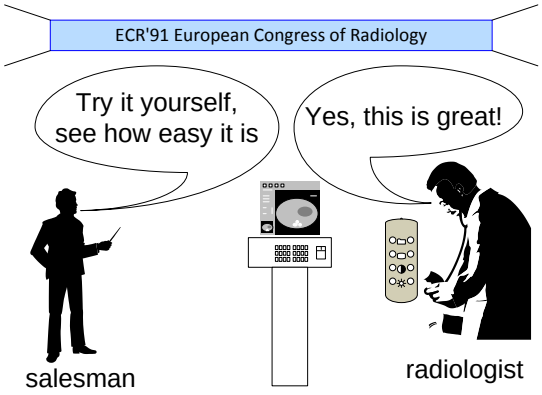


Figure 16.1: The main sales feature is easy viewing

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.

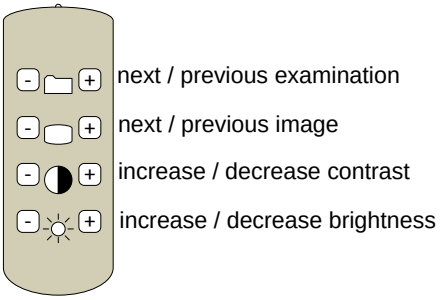


Figure 16.2: The simple remote control makes the viewing easy

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.

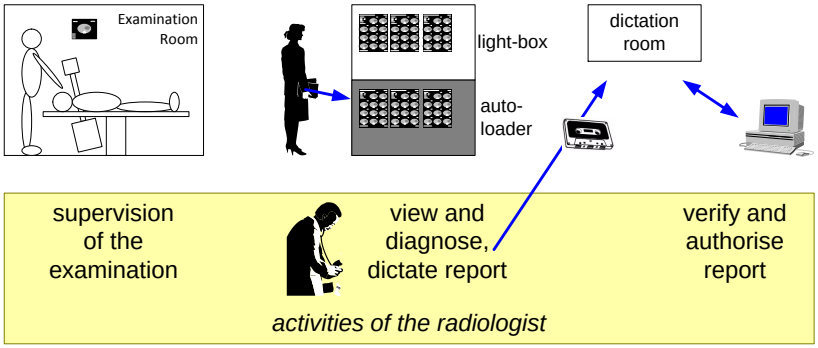


Figure 16.3: Radiologist work-spots and activities

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.

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

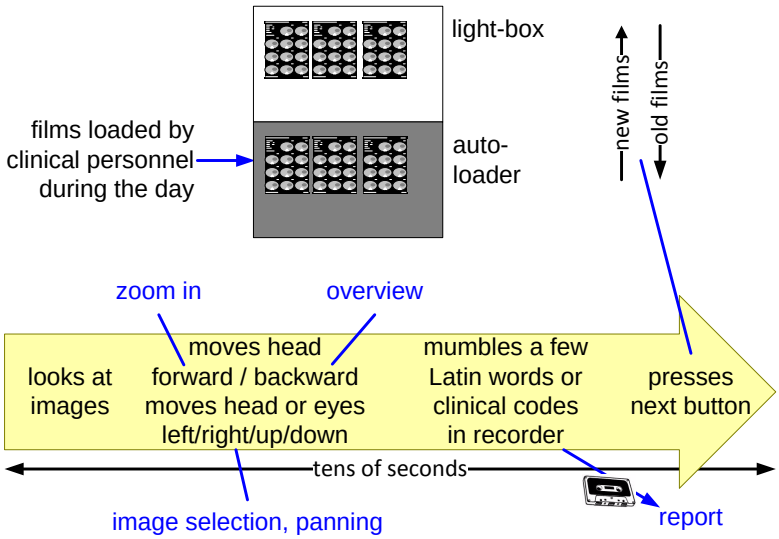


Figure 16.4: Diagnosis in tens of seconds

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 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

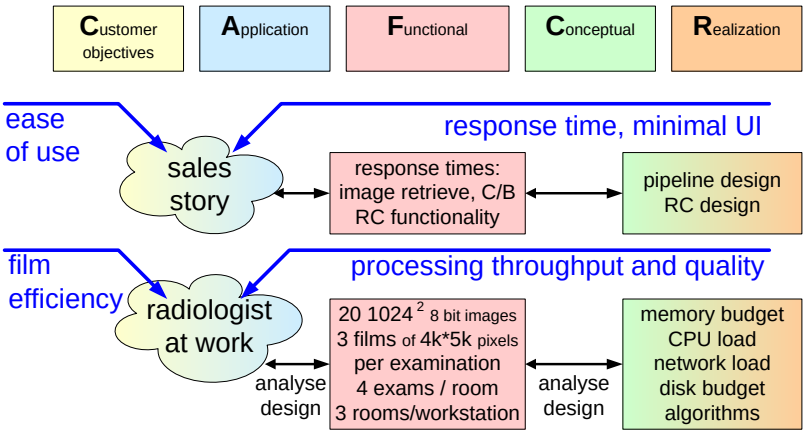


Figure 16.5: The stories in relation to the CAFCR views and the derived requirements and design choices

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.

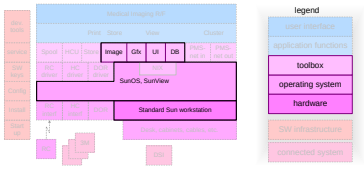
The original software completely ignored the need for printing images on film, it was not even present! The developer crew assumed that radiologists would use the workstation for “soft” diagnosis. Soft diagnosis is diagnosis from the monitor screen instead of film. A better understanding of the radiologist 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

Medical Imaging in Chronological Order



17.1 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.

17.2 Introduction

The context of the first release of Medical Imaging is shown in Section 17.1. The chronological development of the first release of the medical imaging workstation is described in Section 17.3. Sections 17.4 and 17.5 zoom in on two specific problems encountered during this period.

17.3 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.

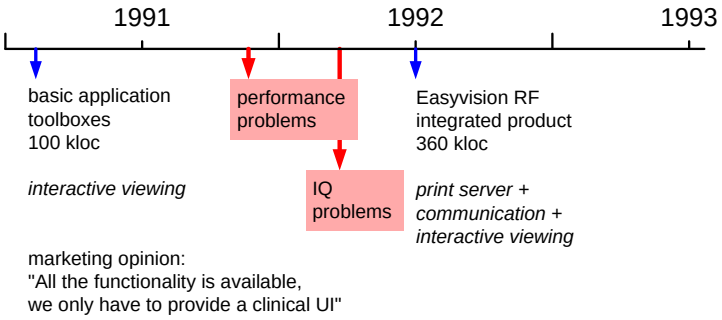


Figure 17.1: Chronological overview of the development of the first release of the Easyvision

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 17.1 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 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

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

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.

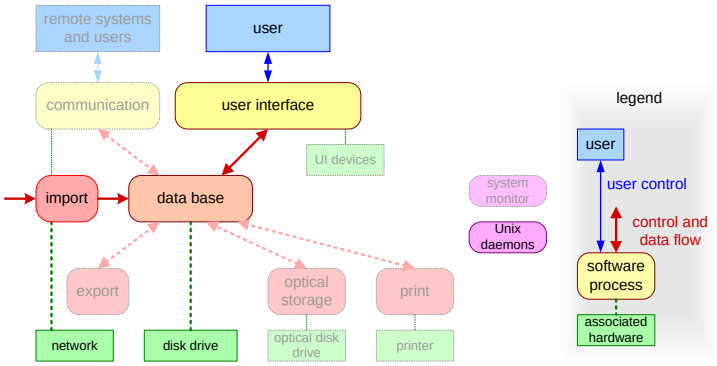


Figure 17.2: The functionality present in the Basic Application shown in the process decomposition. The light colored processes were added to create the Easyvision

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 17.2 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 17.2 and 17.3 are explained further in Chapter 15.

Figure 17.3 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.

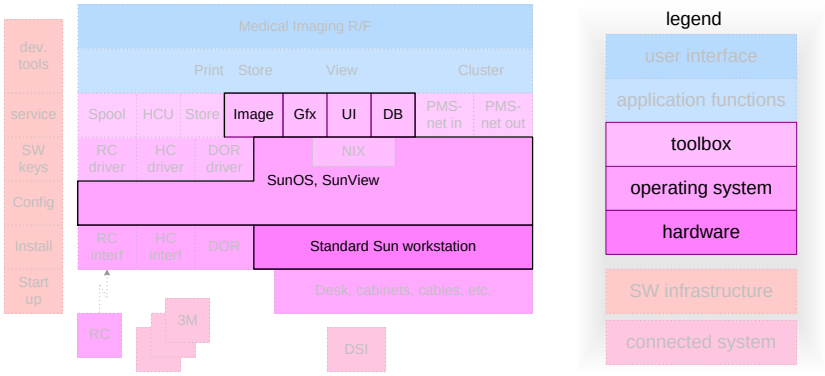


Figure 17.3: The functionality present in the Basic Application shown in the construction decomposition. The light colored components were added to create the Easyvision

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 17.1 both a performance and a (image quality related) safety crisis happened in that period.

17.4 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 17.4 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.

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 17.5 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 17.6 shows the situation per process. Here the shared libraries are shown separate of the processes. The category *other* is the accumulation of a number

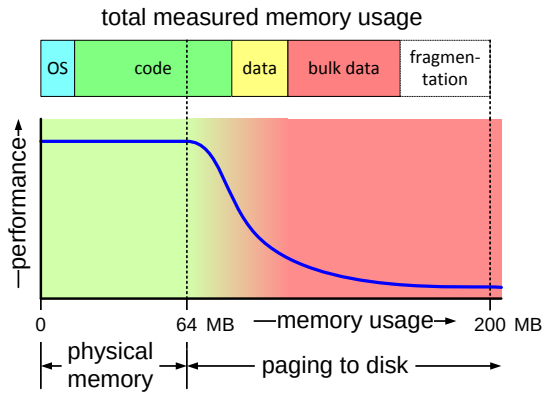


Figure 17.4: Memory usage half way R1

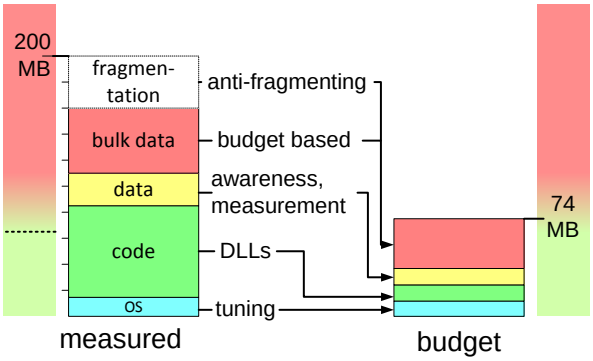


Figure 17.5: Solution of memory performance problem

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.

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 17.7.

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 trans-

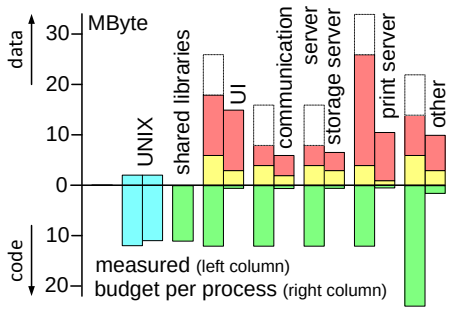


Figure 17.6: Visualization per process

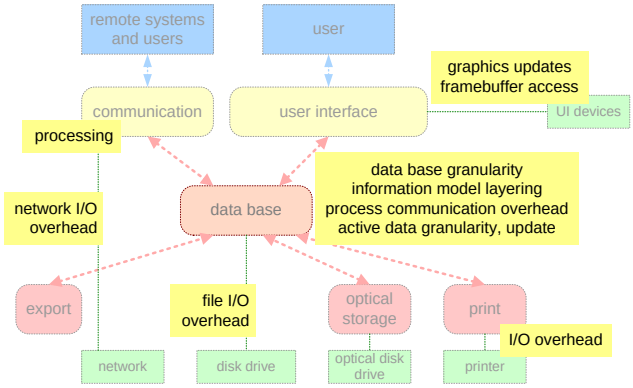


Figure 17.7: Causes of performance problems, other than memory use

action 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 appli-

cation, 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.

17.5 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.

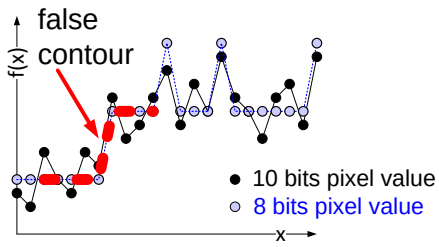


Figure 17.8: Image quality and safety problem: discretization of pixel values causes false contouring

Figure 19.18 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 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.

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

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

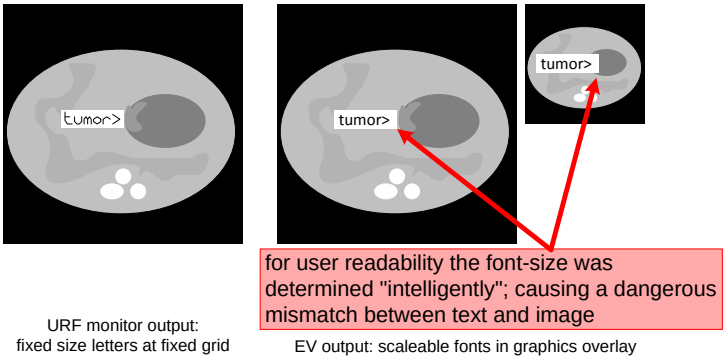


Figure 17.9: Safety problem caused by different text rendering mechanisms in the original system and in Easyvision

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 17.9.

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.

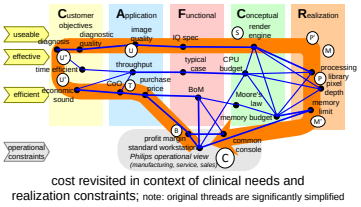
The solution that has been chosen is to define an encompassing rectangle at the interface level and to render the text in a best fit effort within this encompassing rectangle. This strategy maintains the image-text relationship.

17.6 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 18

Threads of Reasoning in the Medical Imaging Case



cost revisited in context of clinical needs and realization constraints; note: original threads are significantly simplified

18.1 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 18.2 the outline of the thread is explained. Section 18.3 describes the 5 phases as defined in Chapter 13:

1. Select starting point (18.3.1)
2. Create insight (18.3.2)
3. Deepen insight (18.3.3)
4. Broaden insight (18.3.4)
5. Define and extend the thread (18.3.5)

18.2 Example Thread

Figure 18.1 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.

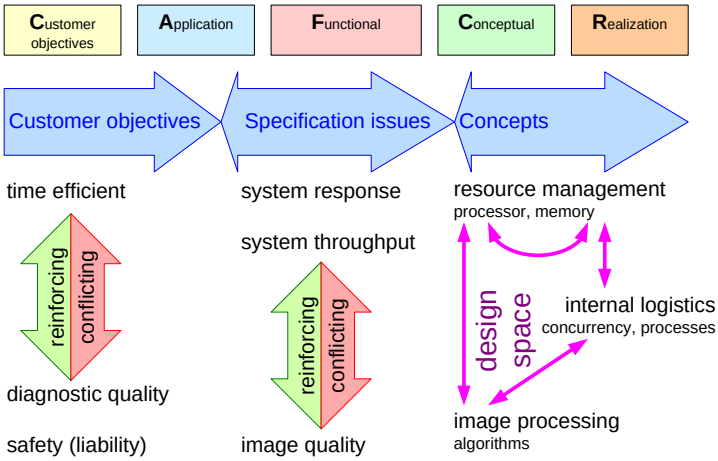


Figure 18.1: The thread of reasoning about the tension between time efficiency on the one hand and diagnostic quality, safety, and liability on the other hand. In the design space this tension is reflected by many possible design trade-offs.

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

18.3 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 18.2 to 18.6, described in Subsections 18.3.1 to 18.3.5, show the phases as described in Chapter 13. 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.

18.3.1 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.

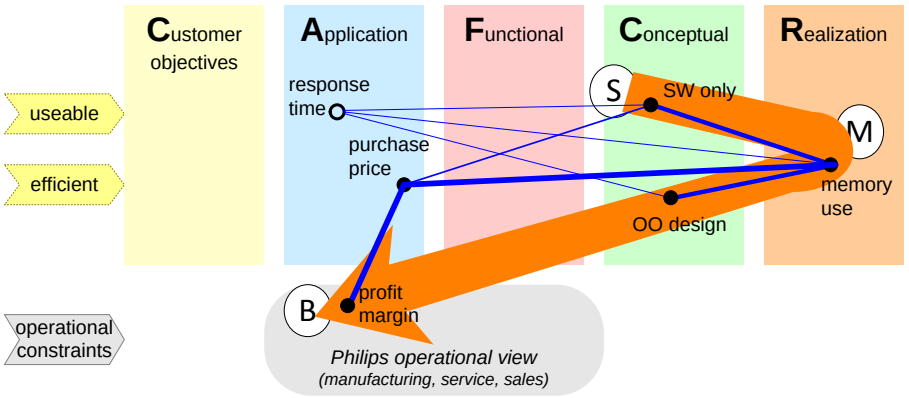


Figure 18.2: Thread of reasoning; introvert phase. The starting point (S) is the a priori design choice for a SW only solution based on Object Orientation. The consequence for resource usage, especially memory (M) and the business (B), especially product margin are explored.

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 18.2 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.

18.3.2 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?

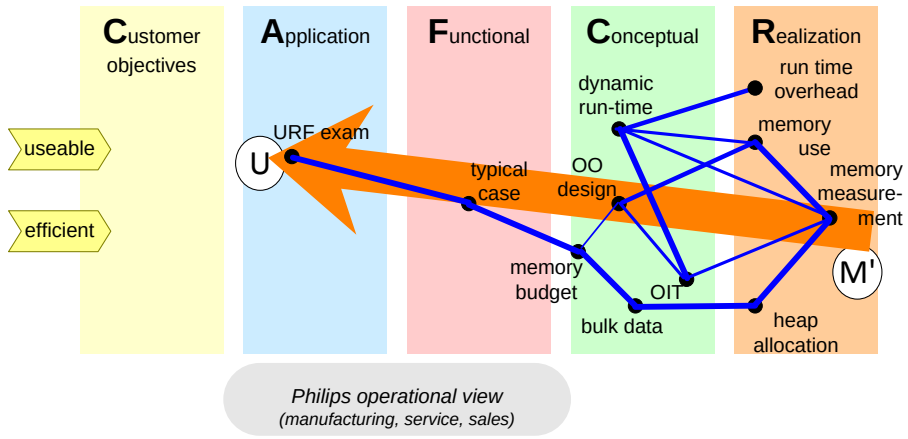


Figure 18.3: Thread of reasoning; memory needs. Create insight by zooming in on memory management (M'). Requirements for the memory management design are needed, resulting in an exploration of the typical URF examination (U).

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.

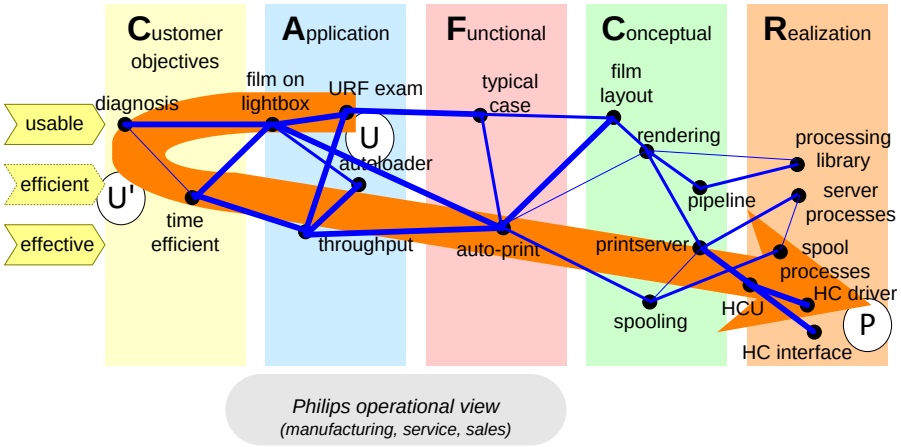
The need arose to define relevant cases to be measured and to be used as the basis for a memory budget. An URF examination was used to define a typical case. Now the application knowledge starts to enter the reasoning process, and the reasoning starts to become more extrovert. Efficiency and usability are the main qualities addressed.

Figure 18.3 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.

18.3.3 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.



Radiologists diagnose from film, throughput is important
Extrovert view shows conceptual and realization gaps!

Figure 18.4: 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).

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 is mostly an introvert concern about resource constraints. Effectiveness is a more extrovert concern about the quality of the result. Hitchins clearly explains in [7] 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 18.4 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.

18.3.4 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.

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 determines 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.

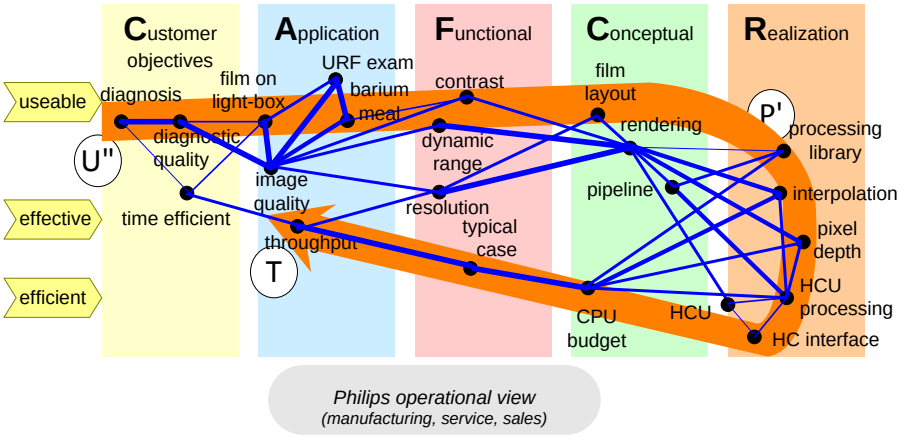


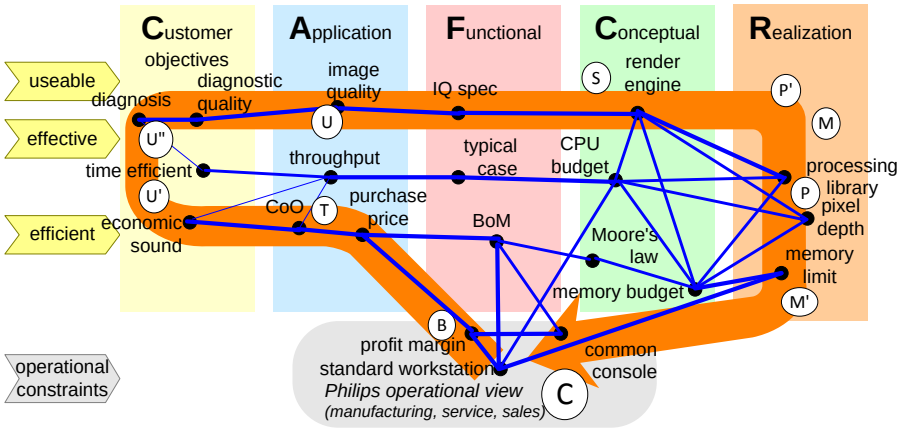
Figure 18.5: Thread of reasoning; phase 4. The insight is broadened. Starting at the objective to perform *diagnosis* efficient in time (U"), the application is further explored: type of examination and type of images. The specification of the imaging needs (contrast, dynamic range, resolution) is improved. 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.

Figure 18.5 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.

18.3.5 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.



cost revisited in context of clinical needs and realization constraints; note: original threads are significantly simplified

Figure 18.6: 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.

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 previous steps are still valid and present. However Figure 18.6 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 18.6 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.

18.4 Conclusion

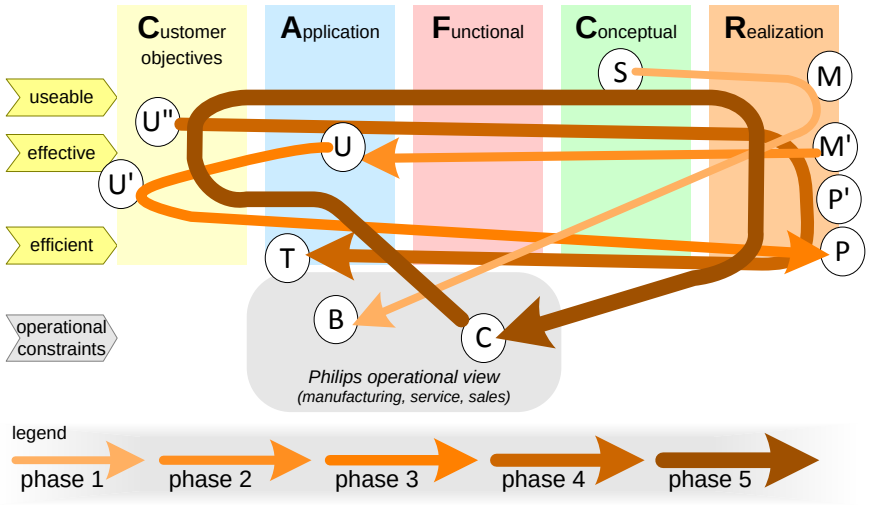


Figure 18.7: All steps superimposed in one diagram. The iterative nature of the reasoning is visible: the same aspects are explored multiple times, coming from different directions. It also shows that jumps are made during the reasoning.

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 18.7 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.

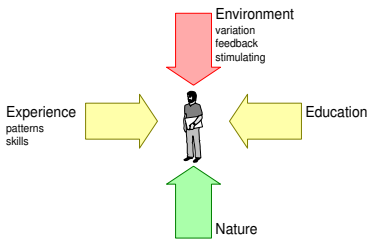
- The positioning of the issues and their relationships in the CAFCR views enable a compact description of the reasoning during the product creation.
- Submethods are used to address one issue or a small cluster of issues.
- Qualities are useful as integrating elements over the CAFCR views.
- The *threads 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

Experiences with Teaching Architecural Reasoning

Chapter 19

Decomposing the Architect; What are Critical Success Factors?



19.1 Introduction

One of the big challenges of today is: How do we get more, and effective, system architects? At Philips and the Embedded Systems Institute we have been very successful in teaching the non-technical aspects of systems architecting to people. This course, called SARCH, has been given 36 times (May 2006) to about 570 participants. We also provide the Embedded Systems Architecting course (ESA), that has been given more than 20 times to more than 300 participants, which addresses the technical broadening of designers. We identified a number of missing steps in between these courses: addressing multi-disciplinary design. We fill this hole by ”single aspect” courses that address one aspect at the time, for instance, performance or reliability. The performance oriented course, that has been given 7 times to about 100 people, is also successful. The next course that we developed to fill this hole is the Multi-Objective System Architecting and Design (MOSAD) course. The evaluation after 3 courses revealed a problem: the participants are satisfied, but the teacher is not satisfied. The dissatisfaction of the teacher is that the participants pick up many submethods and techniques provided in the course, but they struggle to integrate this into an architecting approach.

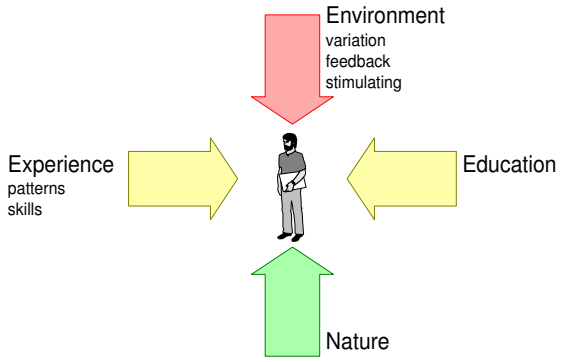


Figure 19.1: Decomposing Contributing Factors

This conclusion triggered the analysis of critical success factors for system architects. We decomposed these factors into four categories: *education*, *experience*, *environment*, and *nature*, as shown in Figure 19.1. We will discuss these four categories in separate sections. We will start with a section about the architect, to create a baseline for the further analysis.

19.2 What is an Architect?

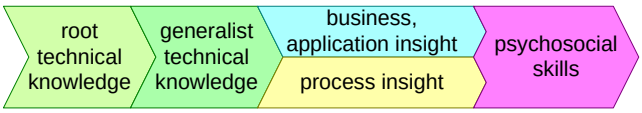


Figure 19.2: Typical Development of a System Architect

System architects need a wide range of knowledge, skills and experience to be effective. Figure 19.2 shows a typical development of a system architect.

The system architect is rooted in technology. A thorough understanding of a single technological subject is an essential underpinning. The next step is a broadening of the technical scope.

When the awakening system architect has reached a technological breadth, it will become obvious that most problems have a root cause outside of technology. Two main parallel streams are opened:

- The business side: the market, customers, value, competition, logistics, service aspects
- The process side: who is doing what and why

During this phase the system architect will broaden in these two dimensions. The system architect will view these dimensions from a technological perspective. Again when a sufficient level of understanding is attained an awareness starts to grow that people behave much less rationally than technical designs. The growing awareness of the psychological and the sociological aspects is the next phase of growth.

Most developers of complex high tech products are specialists. They need an in-depth understanding of the applicable technology to effectively guide the product development. The decomposition of the development work is most often optimized to create a work breakdown enabling these specialists to do their work with as much autonomy as possible.

Most generalists are constrained in the depth of their knowledge by normal human limitations, such as the amount of available time and the finite capacity of the human mind. The figure also shows that a generalist has somewhere his roots in in detailed technical knowledge. This root is important for the generalist himself, since it provides him with an anchor and a frame of reference. It is vital in the communication with other specialists, because it gives the generalist credibility.

Both generalists and specialists are needed. Specialists are needed for their in depth knowledge, while the generalists are needed for their general integrating ability. Normally there are much more specialists required than generalists. There are more functions in the Product Creation Process which benefit from a generalist profile. For instance the function of project-leader or tester both require a broad area of know how.

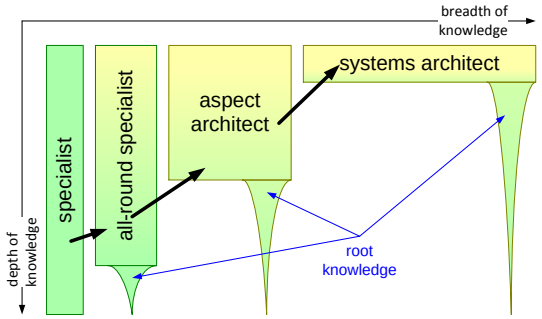


Figure 19.3: Growth in technical breadth, intermediate functions from specialist to system architect

Architects require a generalist profile, since one of their primary functions is to generate the top-level specification and design of the system. The step from a specialist to a generalist is of course not a binary transition. Figure 19.3 shows a more gradual spectrum from specialist to system architect. The arrows show that intermediate functions exist in larger product developments, which are natural

stepping stones for the awakening architect.

Examples of aspect architects are:

- subsystem architects
- SW, mechanics or electronics architects

For instance a software architect needs a significant in-depth knowledge of software engineering and technologies, in order to design the software architecture of the entire system. On the other hand a subsystem architect requires multi-disciplinary knowledge, however the limited scope reduces the required breadth to a hopefully realistic level.

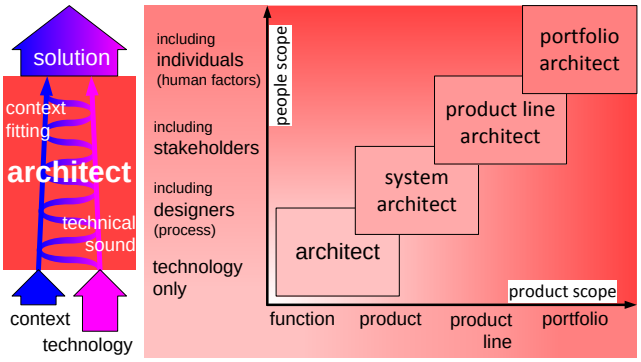


Figure 19.4: Different Architecting Scopes

Many products are becoming so complex that a single architect is not capable of covering the entire breadth of the required detailed knowledge areas. In those cases a team of architects is required, that is complementing each other in knowledge and skills. It is recommended that those architects have complementary roots as well; as this will improve the credibility of the team of architects.

Figure 19.4 shows that the scope of architects widely varies. The common denominator for all these architects is the bridge function between context and technology (or problem and solution). An architect needs sufficient know-how to understand the context as well as the technology, in order to design a solution, which fits in the context and is technical sound at the same time.

In general increasing the product scope of an architect coincides with an increase in people scope at the same time.

19.3 Education

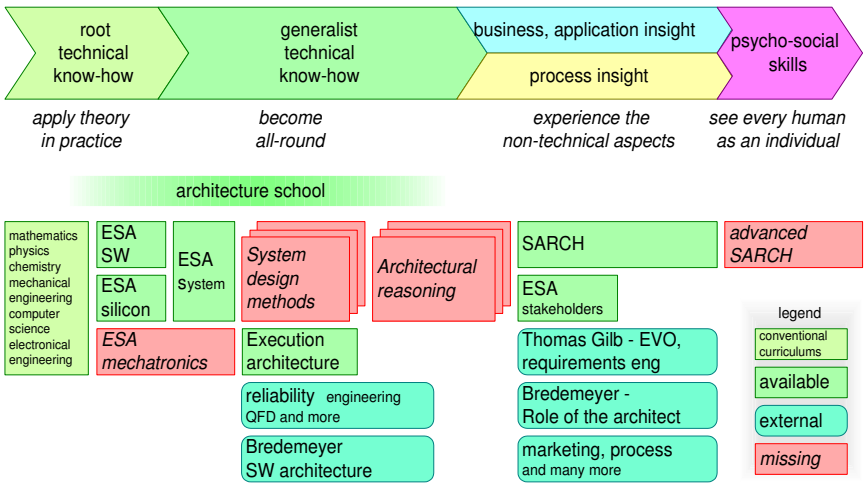


Figure 19.5: Proposed Curriculum for System Architects

A curriculum proposal for architects is shown in Figure 19.5. At the top of the figure the growth path of a system architect is shown. Below the courses or course subjects are shown which fit in the architect career path. Note that this is not a unified list for all architects. Instead it is a palette of courses, where the architect must select the courses which best fit his current needs. In color coding is indicated if courses are available internal or external.

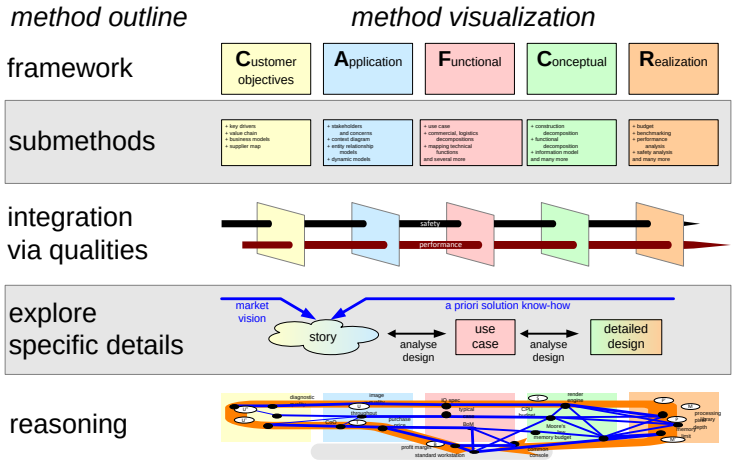


Figure 19.6: The outline of a CAFCR based architecting method

Figure 19.6 shows the overall outline of an architecting method, as it is being

used in the MOSAD or *Architectural Reasoning* course. The right hand side shows the visualization of the steps of the method. The *framework* is a decomposition into five views, the "CAFCR" model, *Customer Objectives*, *Application*, *Functional*, *Conceptual*, and *Realization* views.

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 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.

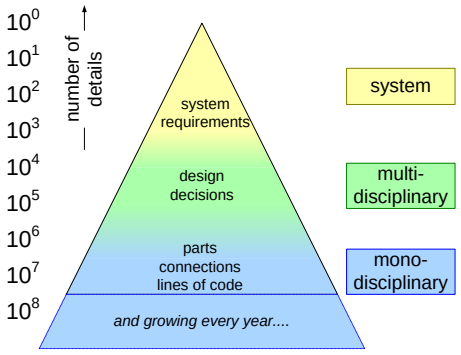


Figure 19.7: Connecting System Design to Detailed Design

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.

The translation of system requirements into detailed mono-disciplinary design decisions spans many orders of magnitude. The few statements of performance, cost and size in the system requirements specification ultimately result in millions of details in the technical product description: million(s) of lines of code, connections, and parts. The technical product description is the accumulation of *mono-disciplinary* formalizations. Figure 19.7 shows this dynamic range as a pyramid with the system at the top and the millions of technical details at the bottom.

The combination of Figures 19.6 and 19.7 brings us to a very common organizational problem: the disconnect between customer oriented reasoning (breadth,

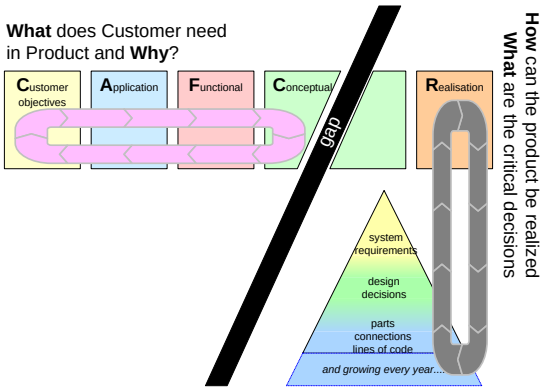


Figure 19.8: Organizational Problem: Disconnect

CAFCR) and technical expertise (depth, the mono-disciplinary area in the pyramid). Figure 19.8 shows this disconnect.

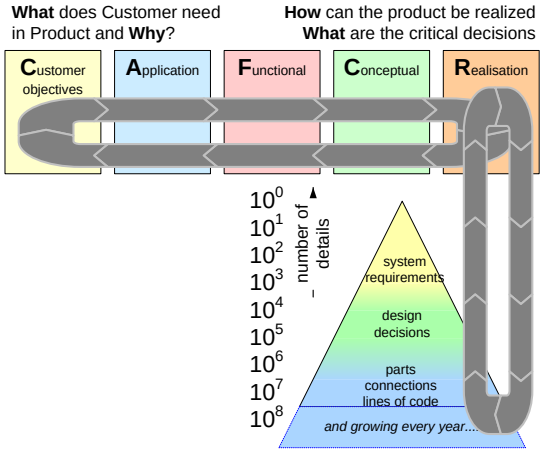


Figure 19.9: Architect: Connecting Problem and Technical Solution

Our definition of the work of an architect places this role as a bridge between these two worlds, as shown in Figure 19.9. In essence the architect must combine and balance breadth and depth iterations.

We should realize that this architect role is quite a stretching proposition. The architect is stretched in customer, application and business direction and at the same time the same architect is expected to be able to discuss technological details at nuts and bolts level. By necessity the architect will function most of the time at higher abstraction levels, time does and brain capacity don't allow the architect to spend all time at detailed design level. Figure 19.10 shows that different people fill

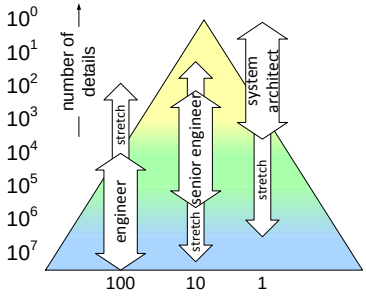


Figure 19.10: Major Bottleneck: Mental Dynamic Range

different spots in the abstraction hierarchies. For communication purposes and to get a healthy system design the roles must have sufficient overlap. This means that all players need to be stretched regularly beyond their natural realm of comfort.

The MOSAD course provides means to address:

- the breadth of systems architecting
- the depth of technological design
- the connection of breadth and depth

If we look back at the first editions of the MOSAD course, then we see that participants have the tendency to either go for breadth or for depth. But exploring both breadth and depth, and even more challenging connecting breadth and depth appears to be very difficult.

19.4 Nature

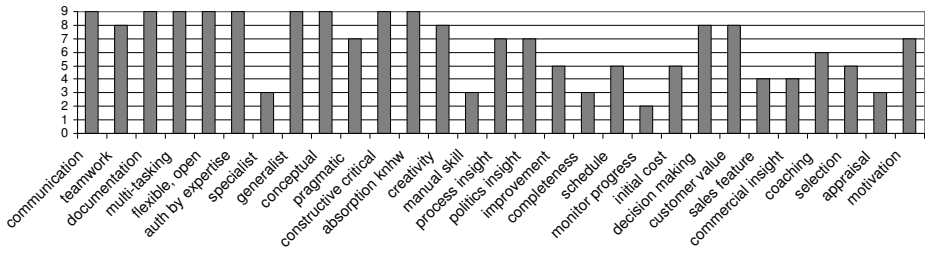


Figure 19.11: Profile of an ”Ideal” System Architect

The profile of the ”ideal” system architect shows a broad spectrum of required skills, as shown in Figure 19.11. A more complete description of this profile and the skills in this profile can be found at[14]. Quite some emphasis in the skill set is on *interpersonal skills*, *know-how*, and *reasoning power*.

This profile is strongly based upon an architecting style, which is based on technical leadership, where the architect provides direction *(know-how* and *reasoning power*) as well as moderates the integration (*interpersonal skills*).

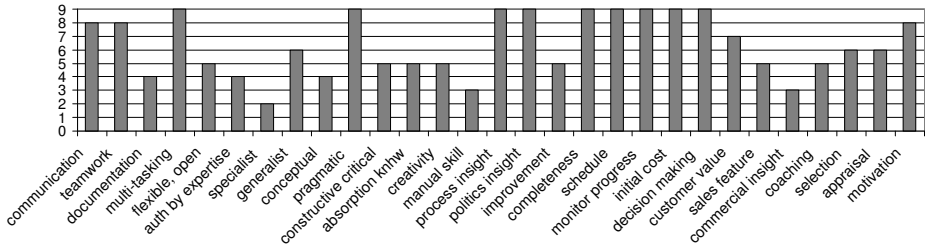


Figure 19.12: For Comparison: Profile of a Project Leader

The required profile is so requiring that not many people fit into it, it is a so-called **sheep with seven legs**. In real life we are quite happy if we have people available with a reasonable approximation of this profile. The combination of complementary approximations allows for the formation of architecture teams, which as a team are close to this profile.

For comparison the profile of a project leader is shown in Figure 19.12. A project leader is totally focused on the result. This requires project management skills, which is the core discipline for project leaders. The multi-tasking ability is an important prerequisite for the project leader. If this ability is missing the person runs a severe risk on a burn out.

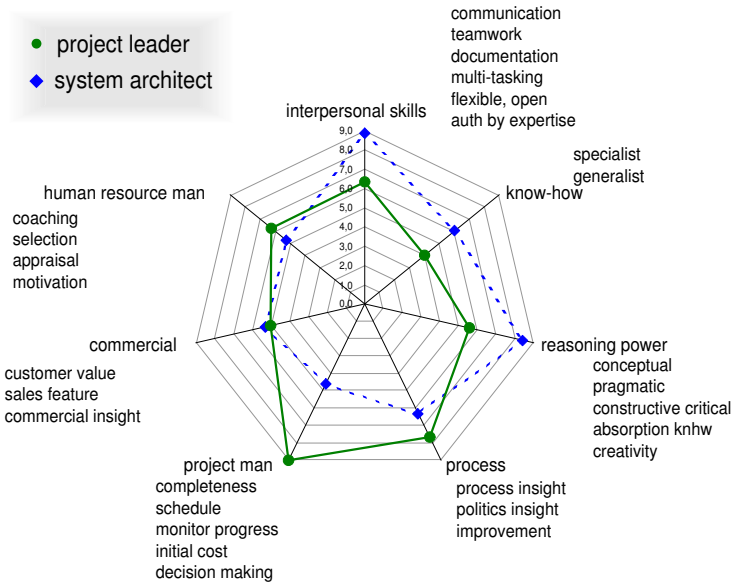


Figure 19.13: Project Leader versus System Architect

The comparison is further visualized in Figure 19.13, where the more detailed skills from Figures 19.11 and 19.12 are grouped together.

- Generalist
- Multi-tasking
- Authority by expertise
- Constructive critical
- Balance between conceptual and pragmatic

Figure 19.14: Most Discriminating Characteristics

In practice the characteristics shown in Figure 19.14 are quite discriminating when selecting (potential) system architects: The first reduction step, when searching for architects, is to select the generalists only. This step reduces the input stream with one order of magnitude. The next step is to detect those people which need time and concentration to make progress. These people become unnerved in the job of the system architect, where frequent interrupts (meetings, telephone calls, people walking in) occur all the time. Ignoring these interrupts is not recommendable, this would block the progress of many other people. Whenever these

people become system architect nevertheless they are in sever danger of stress and burn out, hence it is also the benefit of the person itself to fairly asses the multi-tasking characteristic.

The attitude of the (potential) architect is important for the long term effectiveness. Roughly two attitudes can be distinguished: architects that ask for formal power and architects that operate on the basis of build-up authority. Building up authority requires know-how and visible contribution to projects. We have observed that architects asking for formal power are often successful on the short term, creating a single focus in the beginning. However in the long run the inbreeding of ideas takes its toll. Architecting based on know-how and contribution costs a lot of energy, but it pays back in the long term.

The balance between conceptual thinking and being pragmatic is also rather discriminating. Conceptual thinking is a must for an architect. However the capability to translate these concepts in real world activities or implementations is crucial. This requires a pragmatic approach. Conceptual-only people dream up academic solutions.

19.5 Experience

The effectiveness of an architect depends on experience. In all years of being an engineer, designer and architect, a lot of different needs in different contexts with different solutions with different complicating challenges pass by. If all these events are processed by the (potential) architect, then a frame of reference is created that is very valuable for future architecting work.

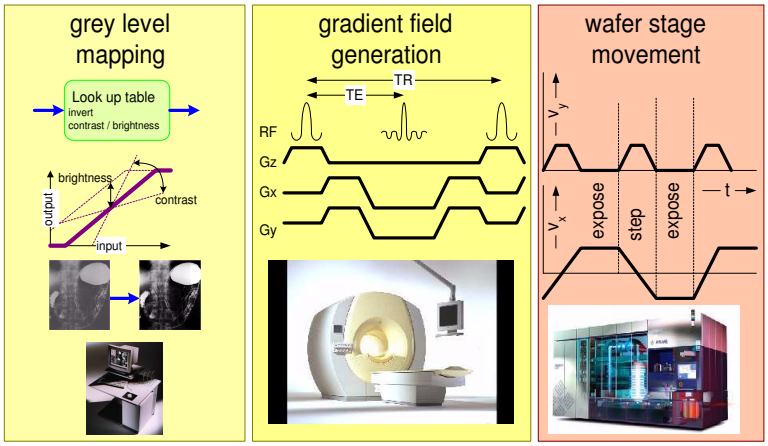


Figure 19.15: Example: Trapezoid Pattern

In this section we will illustrate the experience factor by means of a few architecture patterns that repeatedly popped up in completely different domains. For this purpose we look at the *Trapezoid Pattern*, as shown in Figure 19.15. One of the very common technical problems is the actuation by software of some physical entity, for instance for positioning, moving or displaying. In these cases the software often has to create set-points for one parameter, where this parameter is constant at different levels for some time and switches linearly from one level to another level. For instance, a sample table is at a given position (constant), moves with a constant velocity to the next position, and then stays at the next position for some time (constant). This same behavior is also present in the actuation of gradient fields in MRI scanners, and in the grey level mapping in imaging displays (although the last example uses grey levels as running parameters instead of time).

In the system a chain of transformations takes place to get from a high level software representation towards an actual physical behavior by physical objects. Figure 19.16 shows such a chain of three steps: *computation*, *conversion*, and *actuation*. Note that this chain is often more complex in real systems with more software steps (controller algorithms, corrections), more electronic steps (controllers, amplifiers), and more mechanical steps (motors, transmission). The high level software representation that is the starting point is unconstrained (high precision

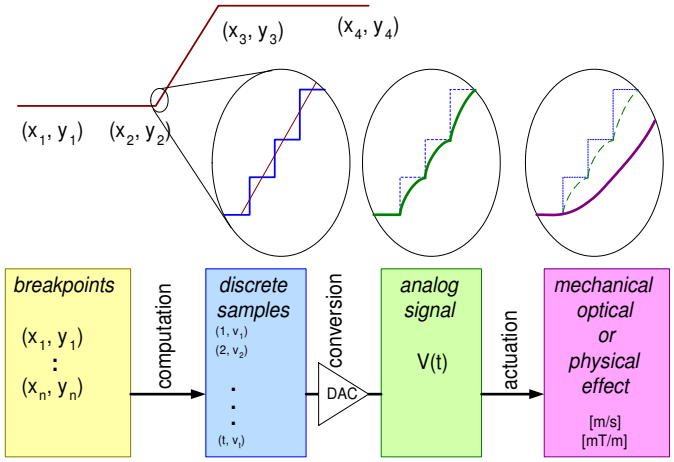


Figure 19.16: From SW input to physical Effect

in time as well as in value). The most common representation is break-point based: the coordinates, where the running parameter changes the linear behavior, are specified.

The conversion and actuation steps have their own particular transfer functions. These steps may introduce additional delays, noise, variations et cetera. The virtual model in the high level software does not take this into account or makes (calibrated) assumptions.

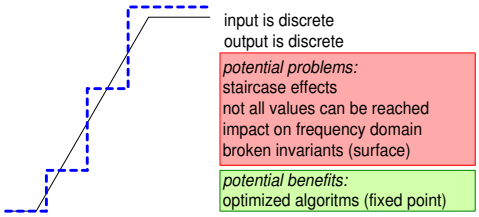


Figure 19.17: Discretization effects

The *computation* step transforms the unconstrained representation into a constrained sampled list of values. This transformation is a discretization in two directions: time and value, see Figure 19.17. This discretization may introduce system level problems:

Staircase effects the linear shape is approximated by many staircase-like steps. The question is how this software output is transformed into the actual physical actuation and if artifacts will be observable in the physical performance.

Not all values can be reached . Normally the digital to analog conversion is a bottleneck in the values that can be reached. This conversion can be very

much limited in low cost solutions (8-bits, 256 values) to limited (16-bit, 65536 values). The time-values are also limited, varying from sub-microsecond for more expensive solutions to milliseconds for simple low-cost controls. The consequence of this limitation is that the physical reality may differ in a systematic way from the virtual model in the high level software. For example the high level software may have determined that at moment $t = 3.14159$ the system should be at position $x = 2.718281$, while actually the system is controlled to stop at $t = 3.1$, $x = 2.7$.

Impact on frequency domain The staircase approximation of linear behavior introduces many higher frequencies in the frequency domain. Many of the higher frequency artifacts are filtered out in the analog and physical part of the chain. However, due to aliasing-like problems the system performance might degrade in unexpected ways.

Broken invariants (surface) The high level software model in many systems is based on invariants. For instance, if we control velocity linear, then we expect that we now the position as the integral of velocity. Discretization, at lower software level, will violate the higher level assumption. If the model assumes we move with $v = 3.14159m/s$, while we actually move with $v = 3.1m/s$, then the position will deviate significant. Interestingly, the low level software can compensate for this error by modulating the value: 58% of the time $v = 3.1m/s$ and 42% of the time $v = 3.2m/s$. These solutions work, but introduce again their own next level of problems and artifacts. In this example the frequency of the modulation may introduce unexpected physical behavior, such as vibrations.

A priori use of the need for discretization can also turn into a benefit. Especially the consequent use of integer representations (with some pragmatic normalization, such as $255 = 5Volt$) reduces processor load, memory use and may increase system performance.

Discretization problems, the artifacts introduced by discretization, the measures against artifacts are also universally applicable. However, the exact consequence and the right countermeasure are domain dependent.

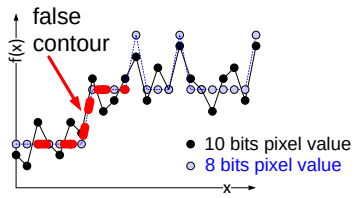


Figure 19.18: Example of Discretization Problem

As example of discretization problems Figure 19.18 shows a typical image quality problem that popped up during the integration phase of a Medical Imaging Workstation. 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 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.

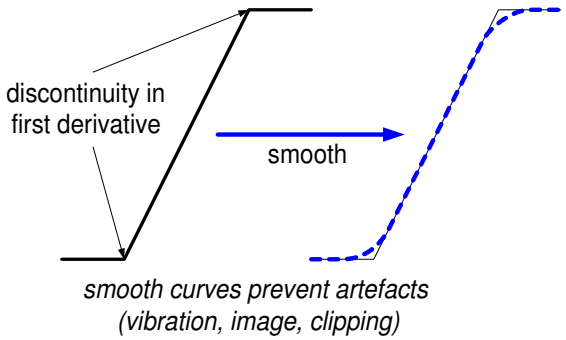


Figure 19.19: Example of Generic Smoothing Consideration

An example of a pattern that builds further on this transformation chain is shown in Figure 19.19. Physical systems in general start to show artifacts with discontinuous inputs. The linear approximation used in the trapezoid pattern has a discontinuity in the derivative. For example, if we control velocity, then the acceleration jumps at the break-point. A solution for this discontinuity is to *smooth* the input function, for instance by a low-pass filter. Note that most analog and mechanical systems are already natural low-pass filters. Despite the low-pass characteristic of the later part of the chain artifacts might still be induced by the discontinuity. These remaining artifacts can be further removed by using an explicit low-pass filter in the high level software model. Again this is an example of a pattern that is universally applied in multiple domains.

The example showed a small subset of patterns that an architect experiences. This subset as its has been discussed here is highly technical. However, in real life technical patterns and organizational patterns are experienced concurrently. For example in the trapezoid example also a number of organizational patterns pop up, related to mono-disciplinary experts and multi-disciplinary design, and system integration.

In Figure 19.20 the career of an architect is shown with the repeated encounters

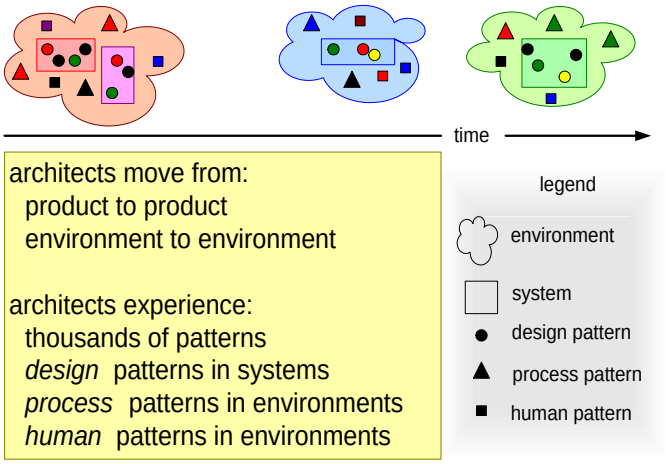


Figure 19.20: Architects Collect a Rich Set of Patterns

of patterns in different products and in different environments. We estimate that an experiences architect encounters (and files and uses) thousands of patterns. All these patterns form a frame of reference for the architect as an individual. This frame of reference helps the architect to assess new architectures very quickly. Potential problem areas are identified and design issues are weighted very fast, thanks to this frame of reference.

19.6 Environment

The business process for an organization which creates and builds systems consisting of hardware and software can be decomposed in 4 main processes as shown in figure 19.21. This process decomposition model is more extensively discussed in[12].

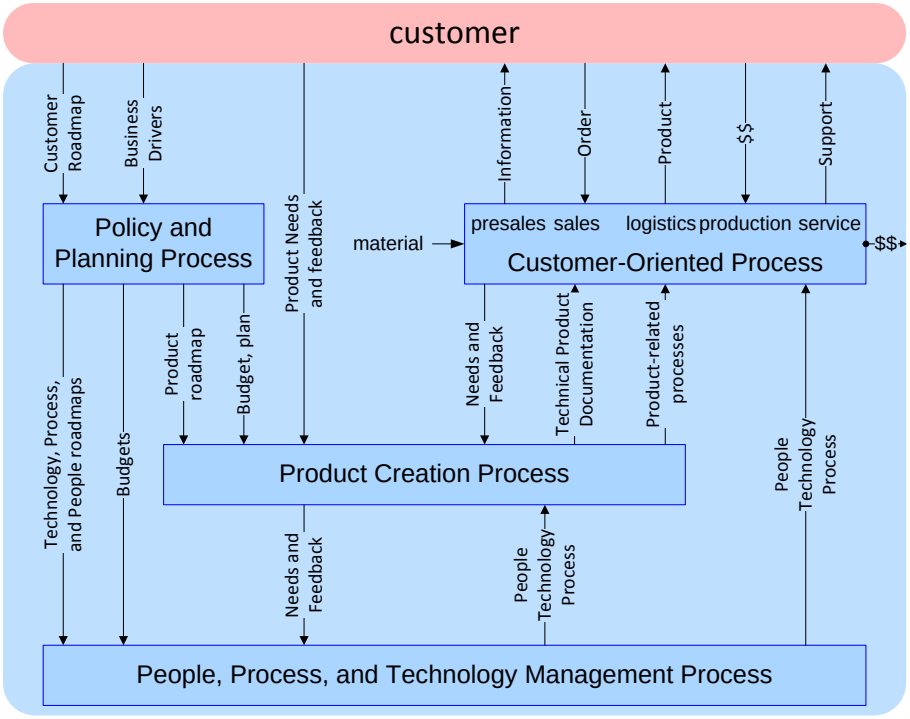


Figure 19.21: Simplified decomposition of the Business

The decomposition in 4 main processes leaves out all connecting supporting and other processes. The function of the 4 main processes is:

Customer Oriented Process This process performs in repetitive mode all direct interaction with the customer. This primary process is the cash flow generating part of the enterprise. All other processes only spend money.

Product Creation Process This Process feeds the Customer Oriented Process with new products. This process ensures the continuity of the enterprise by creating products which enables the primary process to generate cash flow tomorrow as well.

People and Technology Management Process Here the main assets of the company are managed: the know how and skills residing in people.

Policy and Planning Process This process is future oriented, not constrained by short term goals, it is defining the future direction of the company by means of roadmaps. These roadmaps give direction to the Product Creation Process and the People and Technology Management Process. For the medium term these roadmaps are transformed in budgets and plans, which are committal for all stakeholders.

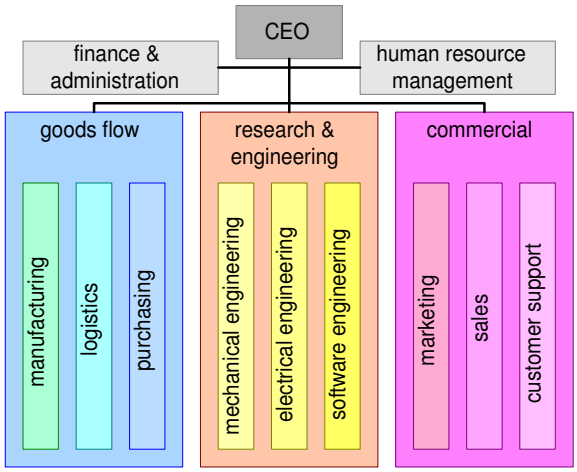


Figure 19.22: Line Organization Stovepipe

The challenge for companies is to organize themselves in a way that support these 4 different types of processes. Rather common is that the People and Technology Management Process is mapped on the line organization, see Figure 19.22. This figure also shows a common problem of hierarchical organization structures: the organizational entities become (over)specialized stovepipes.

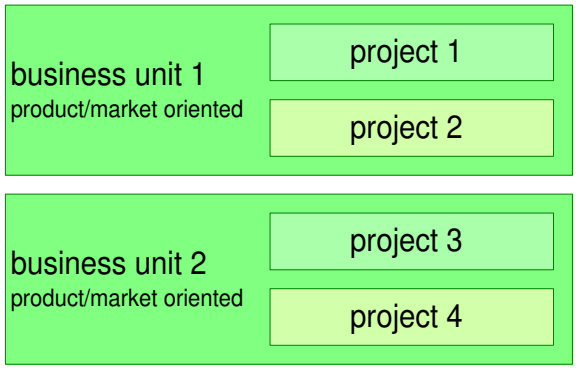


Figure 19.23: Business Organization Stovepipe

The *Product Creation Process* maps often on a business oriented project organi-

zation, as shown in Figure 19.23. The stovepipe problem is here also present, although the stovepipes are now in the product/market direction.

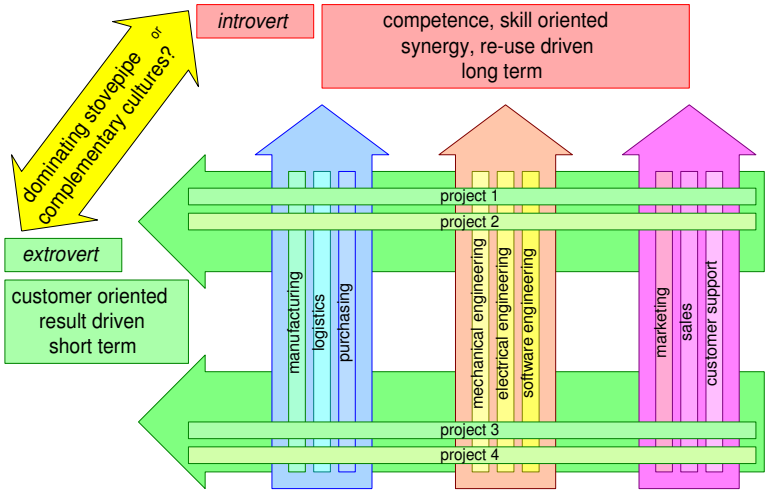


Figure 19.24: Different Concerns

The combination of both organization models results in a matrix organization, where the two types of organizations have different concerns. The line organization is competence and skill oriented, looking for synergy and re-use opportunities. The line organization typically has a long term focus, but an introvert perspective. The business organization is customer oriented and result driven. The business organization typically has a short term focus, but an extrovert perspective.

Figure 19.25 positions the *System Architecture Process* in the simplified process decomposition. The System Architecture Process bridges the Policy and Planning Process and the Product Creation Process. The roadmaps made in the policy and planning process are the shared understanding of direction of the company:

- It positions the products in the market and within the product portfolio.
- It shows the relations between products, such as re-use of technology.
- It positions the product in the technology life-cycle.
- It relates products and technology to the (long lead) development of people and process

The System Architecture Process is the process that:

- Gathers input for the Policy and Planning Process
- Brings in technical overview and common sense in the Policy and Planning Process and the Product Creation Process

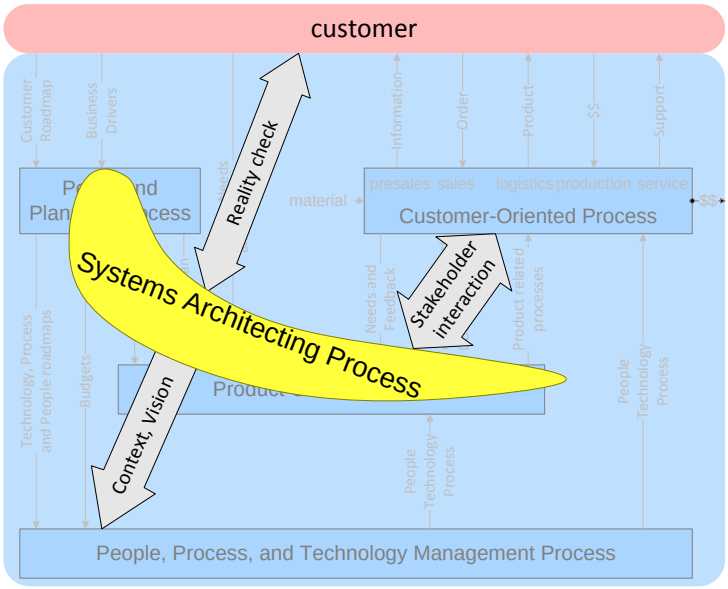


Figure 19.25: Positioning System Architecting

- Transfers the intention of the Policy and Planning Process into the Product Creation Process
- Performs the system level Requirement analysis, Specification, Design and Verification
- Maintains the consistency, integrity and balance.

systems engineering as discipline
job rotation
stimulate architect exposure
stretch all engineers
cultivate customer & market oriented culture
share and invest in future exploration and vision

Figure 19.26: What Can We Do to Improve the Environment?

Until now we have sketched the organizational and process environment in which the system architect operates. A complex environment that is full of human factors, such as conflicting interests and complementing (or opposing?) characters. The natural growth direction in this environment is specialization. In some organi-

zations the security or standardization efforts hurt the architecting effectiveness. For example, we have seen organizations where customer key drivers, cost of ownership models, and market roadmaps are marketing confidential. The gap as described in Figure 19.8 is here imposed by the organization.

Figure 19.26 shows what we can do to improve the environment from system architecting perspective.

Systems engineering as discipline Conventional disciplines are technology oriented, for instance: mechanical, electrical, and software engineering. However, systems engineering has grown into a discipline itself. Most organizations have a severe lack of systems engineers and systems architects. Organizational ownership for systems engineering as a discipline counter-balances the natural tendency towards specialization.

Job rotation is one of the means to broaden employees. The cultivation of a systems attitude requires such a broadening, it is a prerequisite to become systems engineer

Stimulate architect exposure to help them overcome their introvert nature and to help them bridge the gap between managers and architects.

stretch all engineers The broadening mentioned before should not be limited to (potential) system architects. The extremely challenging job of a system architect becomes somewhat more feasible if the engineers are at least system-aware.

cultivate customer and market oriented culture Especially in large organizations the distance from local organizational concerns to customers and market can become large. System architects suffer tremendously from introvert organizations, because the architect has to connect the customer and market needs to technological decisions.

share and invest in future exploration and vision Good architects base their work on a vision. Some investment is needed to create a vision and to keep the vision up-to-date. A vision becomes much more powerful if it is shared throughout the organization.

19.7 Discussion and Conclusions

This paper was triggered by the not yet satisfactory results of our newly developed MOSAD course. Analysis of the critical success factors for system architects provides us with the following insights:

- Only a limited set of technical educated people have a personality profile (the *nature* component) that fits with the architecting role.
- System architecting education for people that do **not** fit in this architect profile is, nevertheless, a good investment. System aware designers ease the job of the system architect.
- Environmental issues, such as organization and processes, have a big impact on the effectiveness of architects.
- Architects need to be stimulated and supported to break through roadblocks imposed by the environment.
- To integrate and use multi-disciplinary design techniques a broad frame of reference is needed. Such a frame of reference helps to position, relate and weight issues, and to identify risks. Without the ability to quickly determine value, relevance and criticality, designers drown in the practical infinite space of problems and solutions.
- A frame of reference grows over time and is the result of experience. This process can be supported by explicit reflection, for instance triggered by a mentor or intervision by peers.

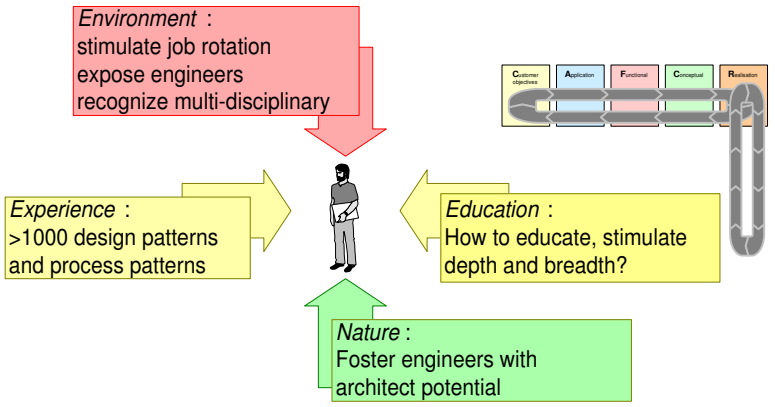


Figure 19.27: Conclusion

Figure 19.27 summarizes the conclusions:

Education How do we stimulate and educate breadth and depth synthesis?

Nature People with architecting genes are scarce; We have to foster and stimulate those people that fit in the architecting profile.

Experience plays a very critical role in cultivating architects. Good architects have a very rich frame of reference with thousands of patterns.

Environment has a big impact on architect effectiveness. Stimulation of job rotation helps to enrich the frame of reference. By exposing engineers to multi-disciplinary aspects the awareness for system issues increases The environment (management, rewarding system) must recognize the value of multi-disciplinary design.

19.8 Acknowledgements

Louis Stroucken detected a painful copy-paste error and provided other useful feedback.

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 Architecture 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] Dana Bredemeyer and Ruth Malan. Resources for software architects. http://www.bredemeyer.com/, 1999.

[2] Dana Bredemeyer and Ruth Malan. Role of the software architect. http://www.bredemeyer.com/pdf_files/role.pdf, 1999.

[3] 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.

[4] 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.

[5] Remco M. Dijkman, Luï¿½s 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.

[6] EventHelix.com. Publish-subscribe design patterns. http://www.eventhelix.com/RealtimeMantra/Patterns/publish_subscribe_patterns.htm, 2000.

[7] Derek K. Hitchins. Putting systems to work. http://www.hitchins.co.uk/, 1992. Originally published by John Wiley and Sons, Chichester, UK, in 1992.

[8] 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.

[9] 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.

[10] James N. Martin. *Systems Engineering Guidebook*. CRC Press, Boca Raton, Florida, 1996.

[11] Gerrit Muller. The system architecture homepage. http://www.gaudisite.nl/index.html, 1999.

[12] Gerrit Muller. Process decomposition of a business. http://www.gaudisite.nl/ProcessDecompositionOfBusinessPaper.pdf, 2000.

[13] Gerrit Muller. The role and task of the system architect. http://www.gaudisite.nl/RoleSystemArchitectPaper.pdf, 2000.

[14] Gerrit Muller. Function profiles; the sheep with 7 legs. http://www.gaudisite.nl/FunctionProfilesPaper.pdf, 2001.

[15] Gerrit Muller. Communicating via CAFCR; illustrated by security example. http://www.gaudisite.nl/CommunicatingViaCAFCRPaper.pdf, 2002.

[16] 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.hitech-projects.com/SAE/COPA/COPA_Tutorial.pdf, 2000.

[17] 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.

[18] William H. Press, William T. Vetterling, Saul A. Teulosky, 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.

[19] Eberhardt Rechtin and Mark W. Maier. *The Art of Systems Architecting*. CRC Press, Boca Raton, Florida, 1997.

[20] Wikipedia. Rational unified process (rup). http://en.wikipedia.org/wiki/Rational_Unified_Process, 2006.

History

Version: 3.4, date: July 3, 2007 changed by: Gerrit Muller

- added Use cases Chapter

Version: 3.3, date: June 8, 2007 changed by: Gerrit Muller

- Part IV: changed into experiences with teaching architecural reasoning

Version: 3.2, date: December 15, 2004 changed by: Gerrit Muller

- added Part IV; background reading

Version: 3.1, date: September 17, 2003 changed by: Gerrit Muller
- added Appendix "Abbreviations"

Version: 3.0, date: July 28, 2003 changed by: Gerrit Muller
- changed the book title in "Architectural reasoning explained"
- refactored the book in a PhD thesis and a more explaining book
- removed the scientific justification and explanation chapters
- performed overdue spell checking

Version: 2.0, date: July 15, 2003 changed by: Gerrit Muller
- added "The future of architecting research"
- changed status in preliminary draft

Version: 1.7, date: July 11, 2003 changed by: Gerrit Muller
- added "Reflection on research method to study architecting methods"

Version: 1.6, date: July 9, 2003 changed by: Gerrit Muller
- added "Balancing genericity and specificity"

Version: 1.5, date: July 7, 2003 changed by: Gerrit Muller
- added "Evaluation of architectural reasoning by diverse sources"

Version: 1.4, date: June 12, 2003 changed by: Gerrit Muller
- added "Evaluation of architectural reasoning"

Version: 1.3, date: May 21, 2003 changed by: Gerrit Muller
- added "Overview of architecting method based on CAFCR and architectural reasoning"
- updated the introduction
- refactored "Architectural reasoning" chapters:
    - moved "problem solving approach" to "basic working methods of an architect"
    - split case chapter in "chronological description" and "thread of reasoning"

Version: 1.2, date: April 9, 2003 changed by: Gerrit Muller
- added "Story telling in Medical Imaging"

Version: 1.1, date: March 26, 2003 changed by: Gerrit Muller
- moved "Story how to" before"Threads of Reasoning"

Version: 1.0, date: March 20, 2003 changed by: Gerrit Muller
- updated figure with book structure

Version: 0.9, date: February 17, 2003 changed by: Gerrit Muller
- added "Medical Imaging workstation: CAF views"

Version: 0.8, date: February 6, 2003 changed by: Gerrit Muller
- updated figure 1 with the structure of the book
- added "Threads of reasoning in the medical imaging case"

Version: 0.7, date: January 28, 2003 changed by: Gerrit Muller
- added "Conceptual and Realization View of Medical Imaging workstation"

Version: 0.6, date: January 21, 2003 changed by: Gerrit Muller
- added "Introduction to Medical Imaging case study"

Version: 0.5, date: January 17, 2003 changed by: Gerrit Muller
- added "Positioning Architecting Methods in the business"

Version: 0.4, date: January 15, 2003 changed by: Gerrit Muller
- added "Research method to study architecting"

Version: 0.3, date: November 11, 2002 changed by: Gerrit Muller
- added "Criterions for architecting methods"

Version: 0.2, date: July 3, 2002 changed by: Gerrit Muller
- recommended literature and resources to introduction

Version: 0.1, date: June 20, 2002 changed by: Gerrit Muller
- Added chapter "Basic working methods architect"
- Added chapter "Quality needles"
- removed section "Qualities" from chapter "Threads of reasoning"
- Added chapter "Realization View"

Version: 0, date: March 28, 2002 changed by: Gerrit Muller
- Created very preliminary bookstructure, no changelog yet