Submethods in the CAF Views

-

logo
TBD

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

Abstract

The customer context and the external characteristics of a system are described in the *Customer Objectives*, *Application* and *Functional* views. This chapter describes submethods to support these views: key drivers, positioning the business of the customer, modelling, use cases and system specification.

version: 1.2 status: finished June 23, 2016

1 Introduction

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

Section 2 describes a submethod to identify key drivers and to relate the key drivers to product requirements.

Section 3 mentions submethods that help in positioning the business of the customer in the context: *analysis of the value chain*, *analysis of business models*, *creation of a map of competitors and complementers*, *application context diagram* and *identification and articulation of the stakeholders and concerns*.

The basic methods modeling and quantification from Chapter **??** are used in Section 4 to identify useful models in the customer world.

Section 5 describes *use cases* as description and communication means for behavioral as well as quantitative characteristics.

The leading document in the product creation process is the *system specification*, which is discussed in Section 6.

Section 7 provides an overview of all the submethods discussed in this chapter, and positions the submethods in the CAFCR views.

2 Key Drivers

The essence of the objectives of the customers can be captured in terms of customer key drivers. The key drivers provide direction to capture requirements and to focus the development. This method has been successfully deployed to focus the product creation process of ASML [3]. A closely related method is "Goal-oriented design" [8], which is also based on analysis of the customer needs and goals in a hierarchical fashion.

The key drivers in the customer objectives view will be linked with requirements and design choices in the other views. The key driver submethod gains its value from relating a few sharp articulated key drivers to a much longer list of requirements. By capturing these relations a much better understanding of customer and product requirements is achieved. In Quality Function Deployment [13], where the term *benefits* is used for key driver, the link between *benefits*, *engineering requirements* and *design concepts* is emphasized.

Figure 1 shows an example of key drivers for a motorway management system, an analysis performed at Philips Projects in 1999.

Figure 2 shows a submethod how to obtain a graph linking key drivers to requirements. The first step is to define the scope of the key driver graph. For Figure 1 the customer is the motorway management operator. The next step is to acquire facts, for example by extracting functionality and performance figures out of the product specification. Analysis of these facts recovers implicit facts. The requirements of an existing system can be analyzed by repeating *why* questions.

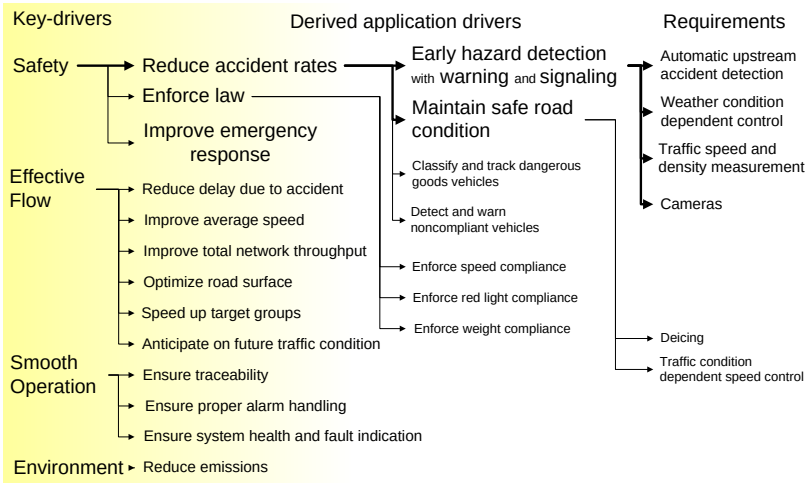

Figure 1: Example of the four key drivers in a motorway management system

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 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 3 shows an additional set of recommendations for applying the key driver submethod. The most important goals of the customer are obtained by limiting the number of key drivers. In this way the participants in the discussion are forced to make choices. The focus in product innovation is often on differentiating features, or unique selling points. As a consequence, the core functionality from the customer's point of view may get insufficient attention. An example of this are cell phones that are overloaded with features, but that have a poor user interface to make connections. The core functionality must be dominantly present in the graph. The naming used in the graph must fit in the customer world and be as specific as possible. Very generic names tend to be true, but they do not help to

• 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 2: 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 3: Recommendations for applying the key driver submethod

really understand the customer's viewpoint. The boundary between the Customer Objectives view and the Application view is not very sharp. When creating the graph that relates *key drivers* to *requirements* one frequently experiences that a key driver is phrased in terms of a (partial) solution. If this happens either the key driver has to be rephrased or the solution should be moved to the requirement (or even realization) side of the graph. A repetition of this kind of iterations increases the insight in the needs of the customer in relation to the characteristics of the product. The **why**, **what** and **how** questions can help to rephrase drivers and requirements. The graph is good if the relations between goals and means are clear for all stakeholders.

3 Customer Business Positioning

The position of the customer in the *value chain* and *the business models* [12] deployed by the players in the value chain are important factors in understanding the goals of this customer. The analysis of the *value chain* of the customer and the analysis of the *business models* involved have been deployed as successful

submethods in the *Medical Systems*, *Consumer Electronics* and *Semiconductors* product divisions in Philips.

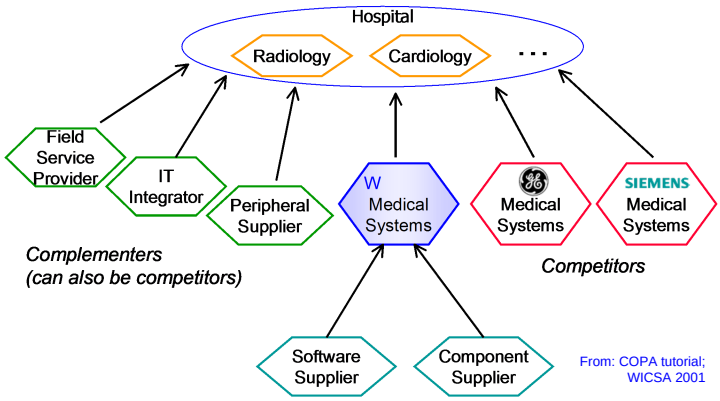

Figure 4: Example map of competitors and complementers from the medical domain

Related to the value chain is the way the customers view their suppliers. The customer sees your company as one of the (potential) suppliers. From the customer's point of view products from many suppliers have to be integrated to create the total solution for his needs. In terms of your own company this means that you have to make a *map of competitors and complementers*, that together will supply the solution to the customer. Figure 4 shows an example of a *map of competitors and complementers* in the medical domain taken from [11].

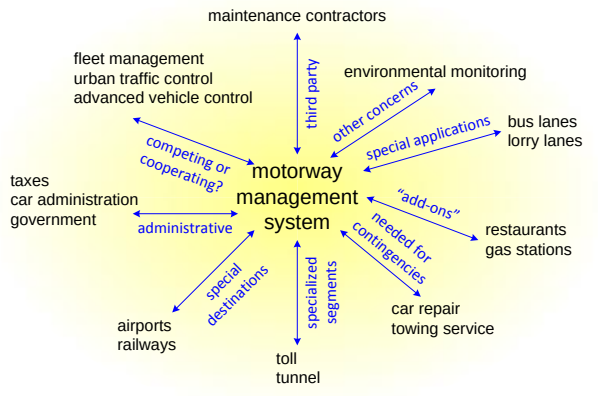

Figure 5: Systems in the context of a motorway management system

An *application context diagram* is a diagram that shows all systems that can

be operational in the application context. The *application context diagram* emphasizes the provided functionality. Many systems in the customer domain have no direct interface with the product under consideration. The interaction of systems in the context with the product happens in many cases via human operators. The value of such a diagram is to understand function allocation in the customer context and to understand the value of potential improvements, such as further integration or automation.

Figure 5 shows a simple context diagram of the motorway management system of Figure 1. Tunnels and toll stations often have their own local management systems, although they are part of the same motorway. The motorway is connecting destinations, such as urban areas. Urban areas have many traffic systems, such as traffic management (traffic lights) and parking systems. For every system in the context questions can be asked, such as:

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

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

The IEEE 1471 [2] standard about architectural descriptions uses stakeholders and concerns as the starting point for an architectural description. *Identification and articulation of the stakeholders and concerns* is a first step in understanding the application domain. This approach matches very well with the CAFCR approach

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

4 Modeling in the Customer World

The customer context can be statically modelled by deploying modeling techniques from the information technology world, for instance entity relationship diagrams [7].

Dynamic models are used to model the logical behavior or the behavior in time. Examples of dynamic models are shown in Figure 6:

- flow models that model the flow of goods, people or information

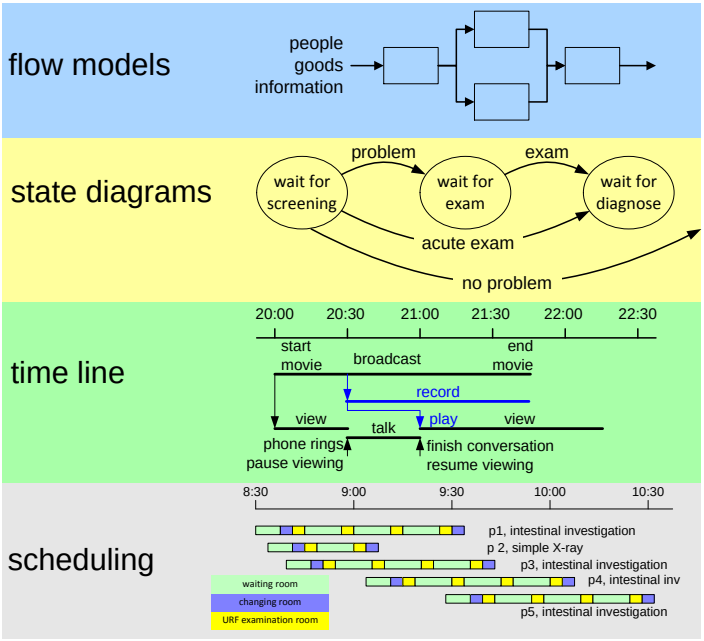

Figure 6: Examples of dynamic models

- state diagrams make the dynamics explicit by means of states and state transitions -
- time line that show the events as a function of time
- resource management by means of scheduling models

Seitz [9] discusses the use of *time domain* and *sequence domain* models in electronic circuit design context.

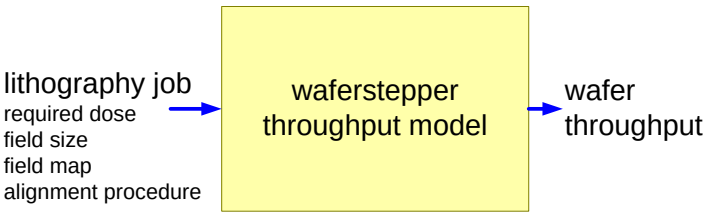

Figure 7: A throughput model that estimates the throughput as function of user controlled values

A comparable class of modelling techniques focuses on economical aspects: for instance *Productivity models* and *Cost of ownership models*. The throughput

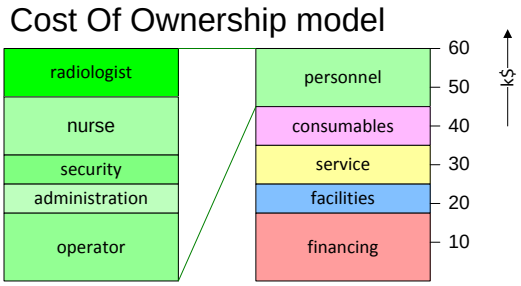

Figure 8: Example of a Cost of Ownership model

model shown in Figure 7 has been used for wafersteppers. This throughput model is internally based on a dynamic model. The throughput model can be used as black box, with a simple parameterized model. In this example the parameters are the *job* that the customer wants to repeat, the *configuration* of the system at the customer side and the *working conditions*. An example of a *Cost of Ownership* (CoO) model is shown in figure 8. Combination of productivity and CoO models are used for cost benefit analysis. Gartner [1] has done a lot of work in the area of cost benefit analysis. The Gartner models are well appreciated in industry. These models can be used as started point for modeling the customer world.

5 Use Cases

Use cases [4] are widely used in software development to describe how the system is used. Most software people use the *use case* submethod only for behavioral descriptions. In embedded systems design this submethod is also very useful for quantitative descriptions of the system, for instance for performance.

Figure 9 shows a classification of use cases, with several examples from the Personal Video Recorder (PVR) domain per category. The most typical use of a PVR is to watch movies: find the desired movie and play it. Additional features are the possibility to pause or to stop and to skip forward or backward. The use case description itself should describe exactly the required functionality. The required non-functional aspects, such as performance, reliability and exceptional behavior must be described as well.

Typical use cases describe the core requirements of the products. The boundaries of the product must be described as well. These boundaries can be simply specified (for example, maximum amount of video stored is 20 hours standard quality or 10 hours high definition quality) or a set of *worst case* use cases can be used. *Worst case* use cases are especially useful when the boundaries are rather situational dependent. These situations can then be described in the use case.

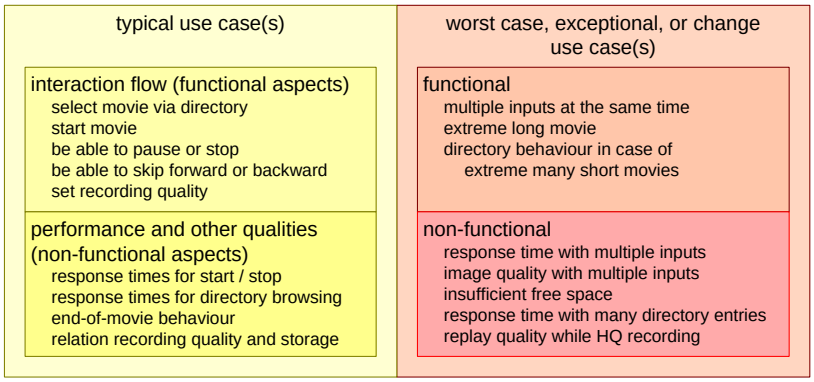

Figure 9: Classification of use cases, with several examples from the Personal Video Recorder domain per category.

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

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

6 System Specification

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

The system specification must cover multiple aspects:

- commercial, service and goods flow decompositions, see 6.1
- functions and features, see 6.2
- quantified requirements, such as performance, see 6.3
- external system interfaces, see 6.4
- standards compliance, see 6.5

6.1 Commercial, Service and Goods Flow Decomposition

The commercial granularity of sellable features and the allowed configurations can be visualized in a commercial configuration graph, as shown in Figure 10. All items in such a graph will appear in brochures, folders, and catalogues. Note that the commercial granularity is often somewhat coarser than the design decomposition. The commercial packaging is optimized to enable the sales process and to increase the margin. In some businesses the highest margin is in the add-ons, the accessories. In that case the add-ons are not part of the standard product to protect the margin.

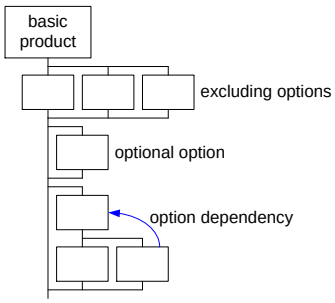

Figure 10: Commercial graph as means to describe commercial available variations and packaging

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

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

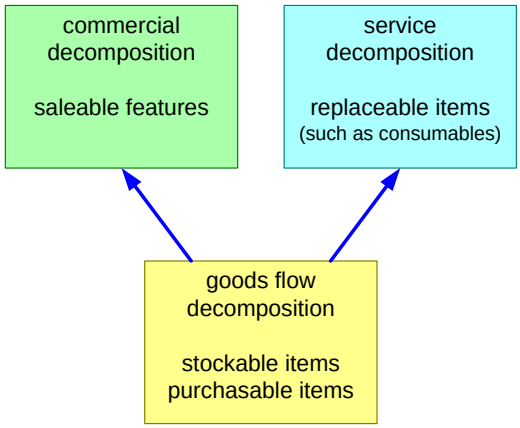

Figure 11: Logistic decompositions for a product

supply chain.

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

6.2 Function and Feature Specifications

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

Figure 12 shows a mapping of technical functions and features onto products. The technical functions and features should still be oriented towards the *what* of the product. In practice this view emerges slowly after many iterations between design decompositions and commercial, service and goods flow decompositions. This type of maps is used in several methods, for instance Quality Function Deployment (QFD) [13], or in PULSE [5].

6.3 Quantified requirements

The system requirements must be specified quantitatively and verifiable in the functional view, see for instance Gilb's recommendations in [6]. This holds for

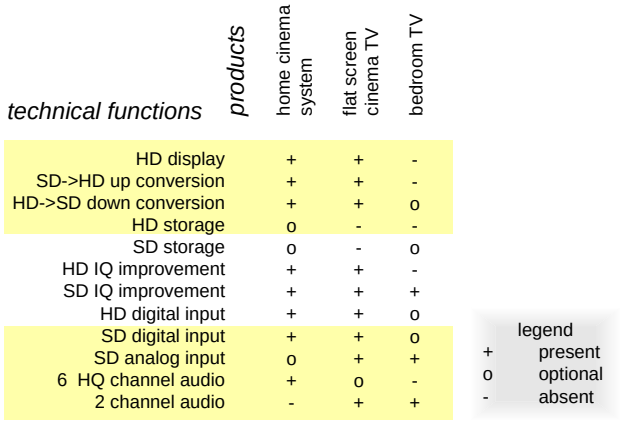

Figure 12: Mapping technical functions on products

many requirements from performance and reliability to qualities that are more difficult to quantify such as extendibility.

Quantification can only take place in conjunction with the circumstances in which this quantification is valid. In easy cases a simple maximum value, which is valid under all circumstances, is sufficient. In many systems quantification is more complicated: for instance the system performance depends on the user settings of the system.

In moderately complex systems it is sufficient to define a limited set of performance points in the parameter space. For more performance-critical and complex systems an external performance model might be required. This describes the required relation between performance and user settings. An example was shown in Figure 7, which shows a throughput model for a waferstepper, see Section 4.

6.4 External Interfaces

The external interfaces of the system can be described in a layered fashion, such as the OSI-ISO Model [14]. A large part of the interface descriptions will be covered by referring to standards, see Subsection 6.5.

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

An information model in the *functional view* describes the information as seen from outside the system. It should not contain internal design choices. This information model is an important means to decouple interoperating systems. The functional behavior of the systems is predictable as long as all systems adhere to this information model. Figure 13 shows an example of a part of an information

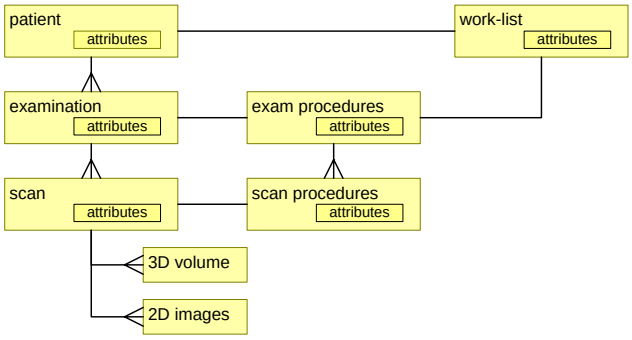

Figure 13: Example of a partial information model described by an entity relationship diagram

model, based on an entity relationship diagram [7]

The ingredients of an external information model are:

- entities
- relations between entities
- operations on entities

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

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

6.5 Standards

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

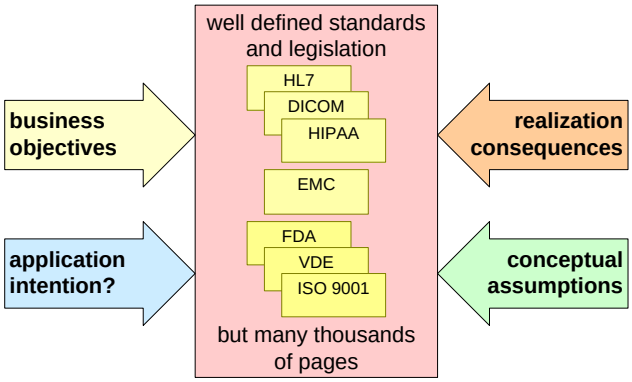

Figure 14: The standards compliance in the *functional view* in a broader force field.

Architecting work is, nevertheless, required in deciding on standards and in designing the implementation. Figure 14 shows the forces working upon the selection of standards. The market and business environment more or less dictate a set of standards. If the product does not comply with those standards the system is not viable. Some of these standards are mandatory due to legislation (for instance mandated by the VDE in Germany or the FDA in the United States), others are de facto musts (for instance DICOM, the medical imaging communication standard).

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

The standard can have significant implementation consequences, for instance in the amount of effort needed or the amount of license costs involved in creating the implementation. These costs must be balanced with the created customer value.

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

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

7 Overview of the Submethods in the CAF views

Figure 15 shows an overview of the submethods that are discussed in this chapter. These submethods are positioned in the *Customer Objectives View*, *Application View* and the *Functional View*. This positioning is not a black and white proposition, many submethods address aspects from multiple views. However, the positioning based on the essence of the submethod helps to select the proper submethod.

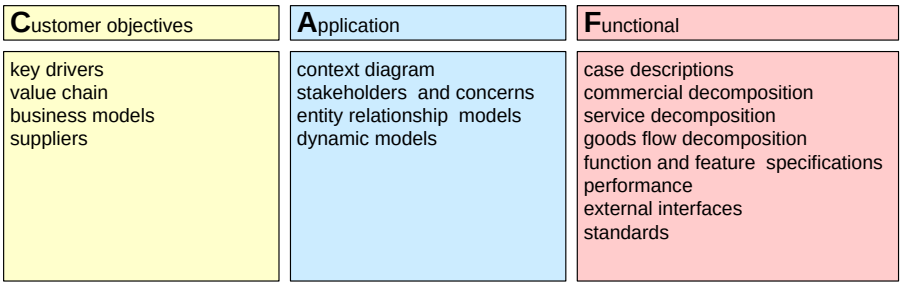

Figure 15: Overview of the submethods discussed in this chapter, positioned in the CAF views

References

[1] Audrey Apfel. BVIT: Frameworks and methodologies that work. `http://www3.gartner.com/resources/113500/113516/113516.pdf`, 2003.

[2] Architecture Working Group (AWG). *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. The Institute of Electrical and Electronics Engineers, Inc., 2000.

[3] Arthur D Little Global Management Consultants. Technology management study, 1998. confidential report at ASML. Complete derivation from 5 customer key driver to about 250 critical waferstepper technologies.

[4] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000.

[5] Jean-Marc DeBaud and Klaus Schmid. A systematic approach to derive the scope of software product lines. In *21st international Conference on Software Engineering: Preparing for the Software Century*, pages 34–47. ICSE, 1999.

[6] Thomas Gilb. *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Elsevier Butterworth-Heinemann, London, 2005.

[7] Volker Haarslev. A fully formalized theory for describing visual notations. In *Proceedings, International Workshop on Theory of Visual Languages, Gubbio, Italy*, 1996.

[8] Laurence Holt. *Goal-oriented design*. unknown, 2004 (planned). A preview was presented at the Gilb Systecture event 2003, London, June 2003.

[9] Carver Mead and Lynn Conway, editors. *Introduction to VLSI systems*. Addison-Wesley, 1980. Chapter 7: System Timing, by Charles L. Seitz.

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

[11] 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.

[12] Karen Peterson, David Hope-Ross, Andrew White, and Marc Halpern. Deriving competitive advantage from product value chains. `http://www3.gartner.com/Init`, 2002.

[13] QFD Institute. QFD institute. `http://www.qfdi.org/`, 2000.

[14] H Zimmermann. OSI reference model - the ISO model of architecture for open systems interconnection. *IEEE Transactions on communications*, COM-28, No. 4, April 1980.

History

Version: 1.2, date: April 6, 2004 changed by: Gerrit Muller

- small textual improvements
- corrected reference to figure key driver method.
- added change use cases

Version: 1.2, date: February 27, 2004 changed by: Gerrit Muller

- small textual improvements
- added some short descriptions to references.

Version: 1.1, date: November 20, 2003 changed by: Gerrit Muller

- added Section "Overview"
- refined application context diagram description
- added Figure "Complementor and competitor map"
- refactored paragraph about IEEE 1471 in section "Business Positioning"
- made throughput model figure more specific
- changed status to concept

Version: 1.0, date: October 10, 2003 changed by: Gerrit Muller

- split figure productivity and cost model
- split key driver figure in method and recommendations
- added refrences
- many language improvements

- changed status to draft

Version: 0.2, date: September 3, 2003 changed by: Gerrit Muller

- added citations to QFD
- added section "use case "
- added section "system specification"
- added abstract
- changed status to preliminary draft

Version: 0.1, date: August 4, 2003 changed by: Gerrit Muller

- adapted the original section "key drivers"
- added section "Customer business positioning"
- added section "Modelling in the customer world"

Version: 0, date: July 28, 2003 changed by: Gerrit Muller

- Created, no changelog yet