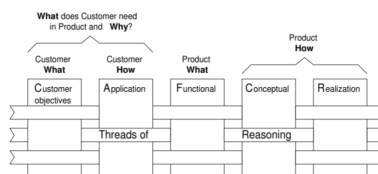
Multi-view Architecting

-



Gerrit Muller, JürgenMüller, Jan Gerben Wijnstra
Embedded Systems Institute, Philips Research
Den Dolech 2 (Laplace Building 0.10) P.O. Box 513, 5600 MB Eindhoven The Netherlands
gerrit.muller@embeddedsystems.nl

Abstract

The development of large SW-intensive products needs to take requirements of multiple stakeholders into account. A design of such a system has to address functional and quality requirements adequately. However, for most of the required qualities no straight-forward design method exists even for a single quality.
A multi-view architecting model is described based upon a decomposition of an architecture in 5 architectural views, ranging from customer objectives to realization. It is the task of the architect to keep these views consistent and to balance design decisions in the perspective of the stakeholder needs.
We derived this model from our experience in developing software intensive industrial products, 2 cases are described from the medical domain.

This paper has been written as part of the "Composable" project in Philips and the Gaudí project.

All Gaudí documents are available at:
http://www.gaudisite.nl/

version: 1.0 status: finished February 10, 2011

1 Introduction

The decomposition of an architecture in 5 views, as depicted in figure 1, separates a number of concerns (the **what** and **how** of the customer or problem domain and the **what** and **how** of the product or solution domain), which increases insight and overview and helps amongst others to focus communication between stakeholders.

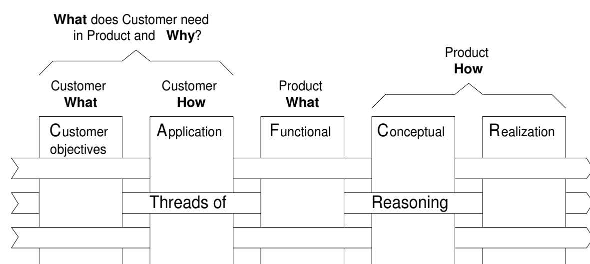


Figure 1: From Customer to Realization, 5 architectural views

A decomposition is a blessing in disguise, the structure and insight created by the decomposition belong to the blessing part, while a new problem is introduced: many stakeholders forget the cross-view relationships. Multi-view architecting tackles this problem by searching for a limited set of integrating-views, which cover the most essential relations between stakeholder needs and technology solutions. These integrating views are visualized in figure 1 in an interweaved fashion.

The more customer oriented views are inherently at a multi-disciplinary level. At the conceptual level design choices start to appear which involve implementation technologies and hence allocation to software or hardware. The system architecture defines these choices, while the software architecture describes the software concepts and realization. In this article we focus on the relation of customer needs to *software* realization.

In this article first a historic case is presented to illustrate a mult-view way of working. Then the historic way of working is rationalized into a reference model, which can be used as a starting point for developing a multi-view architecting method. A first step in creating this method is taken by showing heuristic checklists for issues per view. A second case is presented to illustrate the model.

2 Multiple Views in a Medical Imaging Workstation

The Medical Imaging Workstation development as described here took place from 1990 until 1996 [3]. The main functionality of these workstations is processing, viewing and printing of medical images, acquired by different scanners and X-ray systems. The workstations form a family of products, which are sharing a platform of common functions.

The development documentation of the Medical Imaging Workstation consisted of functional specifications and designs per application, low level designs per module and so-called "system level aspect design specifications" to describe integrating concerns. The aspect documents described integral stakeholder concerns or the requirements analysis, system qualities and design concerns. These aspect documents are the predecessor of the *threads of reasoning* described later in this article. Figure 2 shows the top layers of this documentation.

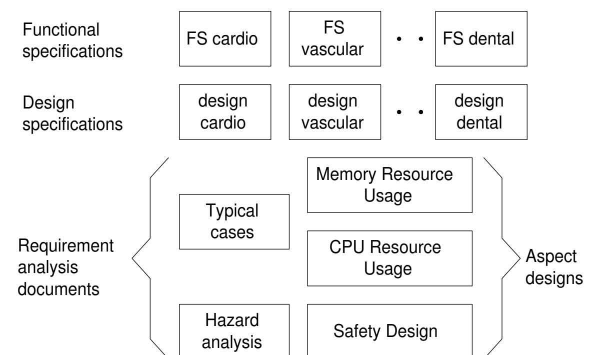


Figure 2: The toplayers of the documentation for the Medical Imaging Workstation.

The first release counted 5 aspect documents. At the time of the third release in 1996 some 15 aspect documents with an average size of 6 pages were used. These documents covered the most essential views for this product family. The aspect document structure evolved in time, the increase from 5 to 15 documents is partly the addition of new views, while the number also increased due to refactoring of aspects. Some aspects might disappear, for instance the disk usage aspect was

quite relevant in 1992 with disks of less than 1 Gbyte, while it is a non-issue in this age with disks of 40 Gbyte or more.

The memory usage apect is detailed here as an example, because it played a dominant role in the architecture of the system. Traditionally all software dominated developments for medical products had severe performance problems after significant functional extension. The introduction of many new technologies at the same time made performance (throughput and response time) a high-risk issue.

The memory usage aspect has to be balanced with the processing power usage aspect, because memory and processing power interfere with each other. For that reason a part of the processing power aspect is described as well.

A safety aspect was also described for this product, however safety is much less of a concern for this product, compared to the Medical Image Acquisition product described later on. A workstation has no physical contact with the patient, nor high power or mechanical dangers are involved.

2.1 The Memory Usage Aspect in the Thread of Reasoning

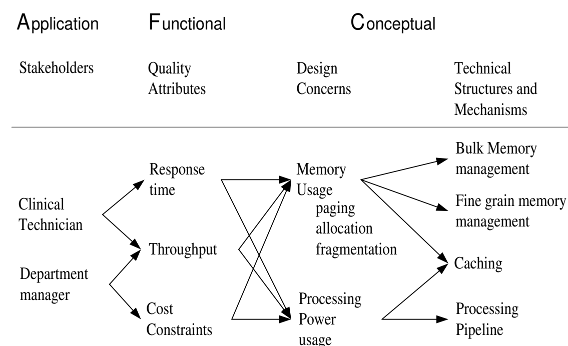


Figure 3: Medical Imaging Workstation: Memory Usage Aspect in the Thread of Reasoning

Figure 3 shows the memory resource usage. The value of the viewing functionality is directly related to the response time of the system, while the value of the print function is determined by the throughput. The available resources are constrained by the costprice of the hardware. The clinical technician, who is the daily user of such a system is mainly concerned with the system performance (response time and throughput), while the department manager as budget owner is mainly interested in a reasonable costprice and throughput.

The quality attributes throughput and response time are transformed into the design concern "resource budgets", especially processor and memory. The memory usage point of view is concerned about the behavior of the virtual memory system, the memory allocation strategy, and memory fragmentation behavior. The processor resource usage is concerned about the amount of CPU time needed for the requested operation.

In this product the memory usage is designed to minimize the activation of virtual memory. Memory budgets are applied which constrain the total amount of memory used to the order of available physical memory.

A bulk memory management mechanism implements the memory allocation. The budget is defined as configuration parameters for the memory allocation. Control over the fragmentation behavior is obtained by separating the fine grain memory allocation and the bulk memory management, and by taking explicit defragmentation measures in the bulk memory management.

An Image Cache mechanism on top of the memory allocation optimizes the use of the limited amount of memory. This cache is also of importance for the processor load, re-use of images in the cache lowers the processor load. This relation shows the trade-off between design concerns to achieve the required system qualities. Reserving more memory for the cache lowers the processor load, but increases the required amount of memory.

The use of an image processing pipeline, with cached intermediate images, is taking this trade-off even further.

3 Multi-view Architecting Model

In every view dominating issues are present, which can be classified as domain specific or more generic. Figure 4 shows typical dominating issues per view.

The rationale in an architecture is expressed in a reasoning from left to right: What is needed in the

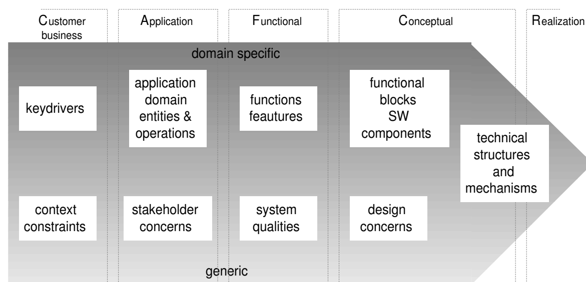


Figure 4: Issues per view

customer business to how is the product realized. The keydrivers in the customer business are realized by domain specific entities and operations. The keydrivers are constrained by the context, for instance economic constraints (material cost, operational cost) or environmental or political constraints. Hence these context constraints influence the realization of the keydrivers by entities and operations.

Figure 5 shows the relations for the issues per views in the customer business and application views. The issues in the next views have similar relations.

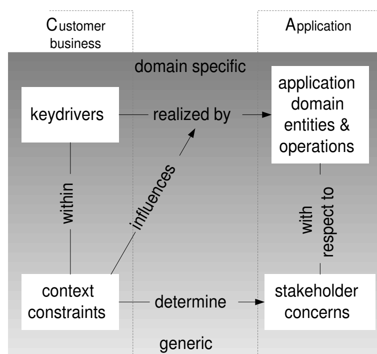


Figure 5: Zooming in on the relations between the issues in the Customer Business and Application views

Real world products have a large amount of details which play a role. Figure 6 shows the amount of detail per view that inherently play a role. The documentation contains a subset of all these details, which is shown as explicit facts. The challenge in complex product creation is to address the relevant issues, because it is impossible to address **all** facts in a reasonable amount of time.

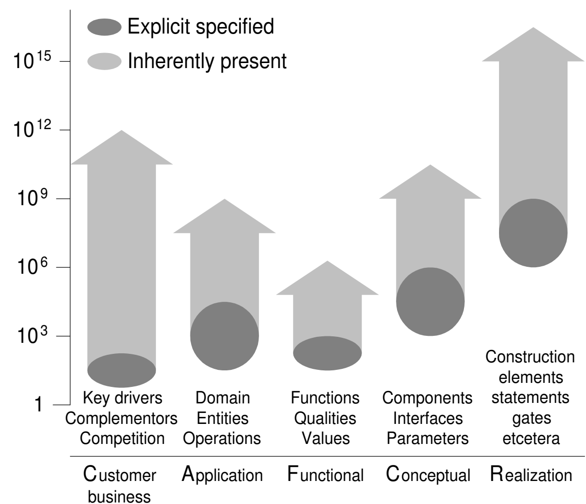


Figure 6: Number of explicit facts and inherent details per view

The explicit facts from Realization to Conceptual to Functional are abstractions, which explain the large reduction. The realization view is often described at statement level, gates, netlists et cetera. Todays products have descriptions of multi-million lines of code, multi-million gates et cetera.

The conceptual view abstracts this to classes or components, interfaces and parameters. Typical systems have multi-thousand classes and tens of thousands methods and parameters. Not all these facts are conceptual by nature, however the order of magnitude in total is tens of thousands.

The functional view abstracts further to a black box description of the system, in terms of functions and quantified qualities like performance. Quite often this specification level is limited to a reasonable communicatable amount, in the order of hundreds of functions and values.

At the other side of the diagram is the customer business and application view, which is from product creation point of view the context. These views need to be made explicit as far as needed to understand the context. This means that the customer business view is reduced to a small explicit abstraction in terms of keydrivers, competition and complementors.

Some more explicit data is needed of how the customer works, in order to understand the role of our

product in the context. The entities in the domain are modeled as well as the operations performed on or by these entities. Hundreds or even thousands of entities can be present. For example the DICOM information model for medical equipment consists of this order of magnitude.

The amount of inherently present facts in every view is orders of magnitude larger. For instance in the realization view the real behavior is always dynamic and the amount of inherent details is proportional with the product of amongst others the state space, the amount of statements and the amount of relations between statements, configurations and parameters.

In all other views the same kind of combinatorial explosion determines the diffrence between explicit specified and inherently present details. While at the context side the ratio of inherently present to explicit specified details increases much more, due to the fact that our explicit facts where highly abstracted. In the context views somewhere the individual consumers are present as "detail", with todays population of billions of people the number of inherently present facts becomes quite large.

Figure 7 shows a network of relations for specific individual issues. For architectures of real products the amount of issues is several orders of magnitude larger than shown here, see figure 6. This large quantity forces the architect in an approach of focusing on the most important (from customer point of view) and most critical (from technical point of view) issues. In practice this means that the architect spends a significant amount of time in a limited set of *threads of reasoning*. Figure 7 shows one *thread of reasoning* as a bold set of relations.

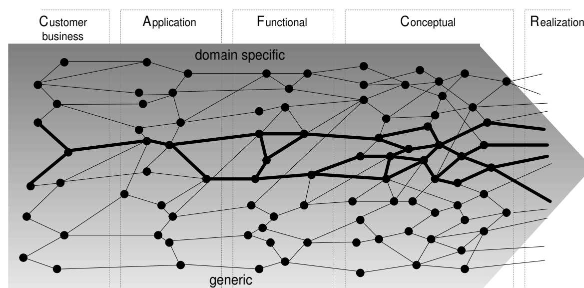


Figure 7: Network of issue relations with one *thread of reasoning* in bold

3.1 Activities in multi-view architecting

Figure 8 shows the main activities of multi-view architecting. These activities are part of the responsibility of the system architect, see [4]. As mentioned above these activities address the challenge of creating an immensely complex product for an even more complex environment, within a limited amount of time. A continuous concern is the level of conciseness, pragmatism is required to keep the conciseness at a workable level, without loosing the critical details.

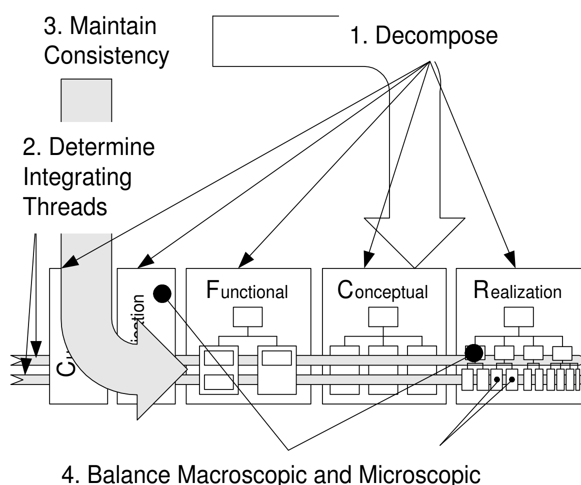


Figure 8: Activities in multi-view architecting

3.1.1 Decompose

Every view requires one or more decompositions. For instance the functional view is often decomposed in functional requirements and quality or non-functional requirements. Kruchten [1] and Soni *(referentie?)* show specific decompositions for the conceptual and realization views in SW architecture.

3.1.2 Determine integrating threads

The decomposition in main views and the further decomposition fragments the system. Likewise repsonsibilities are scattered and many decisions are taken within a limited scope. Integrating threads make the reasoning in a broader perspective explicit.

The most difficult aspect of determining the integrating threads is the selection of a *limited*, approximately 5 in the first generation, set of threads. For professional products as described in this article about 50 threads are required to obtain a reasonable coverage of the system. The know-how, skills, manpower and time constraints require a pragmatic trade-off between coverage and timeliness[1] The main criteria for selection are:

- Important for customer and the business
- Critical with respect to technical realization

These criteria can be phrased as: focus on the important, for instance valuable customer issues and create a thread if the technical implementation is critical, for instance risky or costly.

3.1.3 Maintain consistency

The repeated decompositions in different views can easily introduce inconsistencies. The explicit thread of reasoning helps to detect inconsistencies accross views. Note that this is by far not waterproof, since the threads of reasoning by far do not cover all issues in all views. A verification and feedback process is essential to detect inconsistencies as early as possible.

Note also that the threads of reasoning can be mutually inconsistent. By looking along the main decomposition direction some of these inconsistencies are detected.

3.1.4 Balance

Balancing is one the main responsibilities of an architect. Balancing is required at every design level, from microscopic level, statements for instance, up to macroscopic level, business key drivers for instance.

The balance between integrating views is a concern during the selection of the integrating views. For instance functionality or performance oriented threads need to be complemented with cost or power constrained threads. Ideally both aspects are covered in one thread, for instance performance cost ratio.

[1] For many people this may sound dangerous and irresponsible. However most people do not realize that many systems are developed, where most people are not even aware of the integrating concerns! At the other hand systems, where a large coverage is pursued, are often not successfull due to lack of feedback.

3.2 Pitfalls in multi-view architecting

The list of activities can be complemented by a list of pitfalls, as shown in table 1

- too few views
- completeness
- general formalization

Table 1: *Pitfalls in multi-view architecting*

3.2.1 Too few views

As a counterexample we sketch the situation when working with a one-dimensional approach. Take, for instance, a naïve object-oriented approach working only with objects, classes and relations. For the application domain, domain objects are identified. These domain objects are detailed to describe the precise functionality of a product. These objects are extended with design objects to address design issues. The design objects are implemented as classes and instances. Traceability links show the derivation of one workflow to the next. The number of objects is growing from workflow to workflow. The implemented system is just a network of classes. The entire structure of objects within and between workflows becomes strongly coupled by functional dependencies and therefore hard to manage and evolve.

As a consequence of such a one-dimensional approach issues like functional structure or subsystems can only be handled as adornments of certain domain objects. In the same way, error handling and initialization only refer to sets of design objects, which represent the according design decisions.

The proposed multi-view architecting approach allows dealing with each of the issues within the relevant workflow within a specific view.

3.2.2 Completeness

Another pitfall is the pursuit of completeness. Often an attempt is made to document requirements completely to have a sound basis for further development steps. Completeness is aimed for to gain complete certainty that the right work is being performed.

The problem with this attempt is that a criterion for completeness is hard to define and that most development work is done in an evolving context. Large extensive documents easily become outdated and they are hard to keep up to date.

A multi-view approach addresses the issue of certainty by approaching the workflows from multiple viewpoints. Complete certainty is an illusion. It is better to look for early feedback on the work performed and to adjust it, if necessary. This leads to an incremental way of developing.

3.2.3 General Formalization

The executable system is completely formal. To improve the development quality often such rigor is already pursued in early development phases. For instance by describing requirements using a formal notation.

Formalization of requirements should only be used were requirements are available in a precise form. Formality of representations grows as the development moves from requirements to design. A set of lightweight notations is preferable to one heavy notation.

The use of formal models in early development such as executable simulations may be very useful for feasibility analysis. In general, formalization should be used for well-defined problems with specific scope where it can make a specific contribution.

3.3 Evolution and Feedback in multi-view architecting

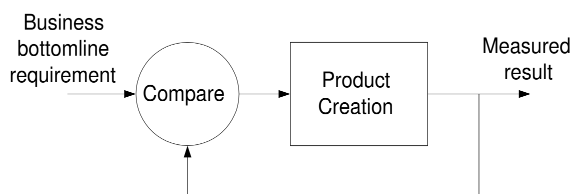


Figure 9: The product creation process of a complex product requires continuous feedback

The scope and complexity of today's products are so large that no single human being can oversee and understand the system completely. The limits of human know-how and skills require an approach which takes care of these limits. A very effective apporach is via continuous feedback, as shown in figure 9.

Working incremental enables obtaining feedback to control the next incremental development steps. It is crucial to have both a criterium for success as well as a measure of the result to compare with the success criterium. The success criterium as well as the measure for success must be very close to the "real-world" objectives. For instance measuring customer satisfaction is better than measuring system performance, while measuring system performance is better than measuring module performance et cetera.

Typical bottomline business requirements are related to financial parameters like cashflow, margin, return on investment, but also future oriented parameters like innovative power and potential and satisfied customer base.

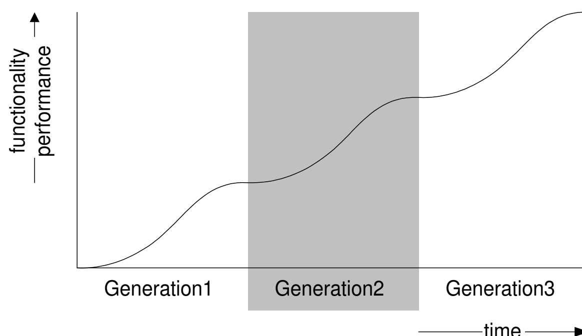


Figure 10: Succesive product generations increase the functionality and performance stepwise.

The feedback control mechanism is applied within a product generation as well as from one generation to the next generation. Figure 10 show the normal stepwise increase of functionality and performance for succesive product generations. The views and integrating threads should evolve together with the product family. From one generation to the next the thread structure should be refactored and extended. The number of threads will more or less increase proportional with the generations, increasing the coverage of the system.

Safety
Security
Reliability
Robustness
Useability
Appeal, Appearance
Throughput or Productivity
Response Time
Image Quality
Reproduceability
Predicatability
Accuracy
Transportability
Wearability
Storability

Manufacturability
Testability
Serviceability
Configurability
Installability
Evolvability
Portability
Upgradeability
Extendability
Maintainability
Logistics flexibility
Lead time
Standards Compliance

Cost price
Cost of operation
Interaction with environment
Power consumption
Consumption rate (water, air, chemicals, etcetera)
Disposability
Size, weight
Resource utilization

Figure 11: Emperical checklists for for qualities in all 5 views

4 Emperical checklists

From our experiences in the development of software intensive systems we compiled some checklists, shown in figure 11 and 12.

The qualities shown in figure 11 are relevant in all 5 views. In the *customer objective* view the ultimate objectives of the customer with respect to the quality are articulated. In the *application* view the customer way of working is depicted to achieve the quality. The *functional* view described the product requirements regarding this quality. The design strategy and the choosen concepts to ensure the quality are described in the *conceptual* view. Finally the realization in lines of code, electronics circuits et cetera form the *realization* view.

granularity
scoping
containment
cohesion
coupling
interfaces
allocation
budgets
information model
entities
relations
operations
characteristics
static
dynamic
configuration man.
packages
components
files
objects
modules
interfaces

meta-functional
operational
image processing
handling calls
..
initialization
start-up
shutdown
bootstrap
discovery
negotiation
fault handling
exceptions
logs
traces
diagnostics
configuration handling
data replication
performance observation
capability query
testing
automation
special methods
harness
suites
off-line guidance

supply chain
outsource
co-design
buy
interoperate
source vs binary
technology choices
lifecycle
obsolesence
core, key, base
SW development
environment
repository
tools
feedback tools
monitoring
statistics
analysis
call graphs
message tracing
object tracing
licensing
SW keys

synchronization
signalling
messaging
call-back scheduling
notification
active data
watchdogs
time-outs
locking
semaphores
transactions
checkpoints
deadlock detection
roll-back
priorities
pre-emption
concurrency
processes
tasks
threads
persistence
caching
versioning,
prefetching
lazy evaluation

identification
uniqueness
naming
data model,
registry
scoping
configuration
database
inheritance
resource management
allocation
anti-fragmentation
garbage collection
distribution
allocation,
transparency
component,
client/Server
multi-tier model
infrastructure

Figure 12: Emperical checklists for SW architecture aspects in conceptual and realization views

In the *conceptual* and *realization* view well known styles, patterns and mechanisms are (re-)used to ensure an appropriate quality. Figure 12 is an attempt to capture the relevant aspects we met in a checklist.

5 Safety and Reliability for Medical Image Acquisition

Medical image acquisition systems are used to acquire and display images for interventional support or diagnostics. In case of interventional usage, for example a vascular surgeon is operating the patient using a catheter. For this surgeon it is of the utmost importance that he always sees what he is doing inside the patient, i.e. the acquired images must be displayed real-time (fluoroscopy). During acquisition, the patient and in a much lesser degree the surgeon are exposed to a dose of radiation. This dose must be a small as possible. Next to interventional usage, these systems are also used for diagnostics by radiologists, i.e. acquiring images, applying image processing functionality, and using them to make a diagnosis. To obtain the best images, the image detector must be carefully positioned with respect to the patient. For this, the acquisition systems contain massive moving parts that can move at high speed. Relevant views for such systems include response-time and throughput, similar views as for imaging workstations. We will focus here however in this example on reliability and safety which are of particular importance in these acquisition systems (see figure 13).

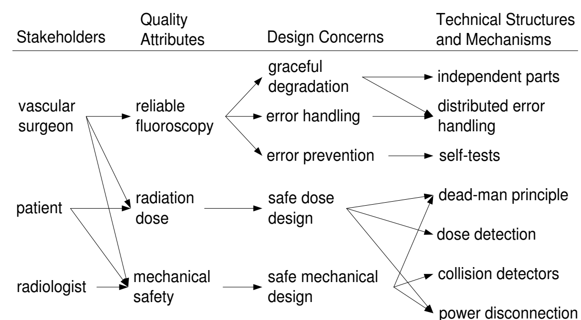


Figure 13: Safety and Reliability Views in a Medical Imaging Acquisition System

Two important stakeholders for the system are thus the vascular surgeon and the radiologist. The patient that is being examined is also a stakeholder. For the vascular surgeon it is important that fluoroscopy is available when using a catheter. Especially for the patient it is important that the dose to which he/she is exposed is limited. Furthermore, when using the system, the physician or patient should not be hit by mechanical parts.

This leads to the requirement that fluoroscopy must be highly reliable. This results in a number of design concerns. Amongst others, the system must support graceful degradation, i.e. when one part of the systems fails then other parts should continue as good as possible. Especially the parts concerned with fluoroscopy must be able to continue. The system is built from a number of relatively independent hardware and software parts, limiting the consequences of a failure in one of these parts. For example, the software is divided into a number of functional units, and each of them has its own process space, limiting the consequence of a failure in one of the units. Furthermore, each unit is responsible for its own error handling, in order to avoid cascading errors. The issue of preventing errors is addressed by providing self-tests, which help to identify possible sources of problems in time. When reliability and error prevention are concerned, also other qualities play a role, for example the testability of a system so that possible errors are more easily detected during development, and the serviceability where service-engineers can for example inspect log-files and can replace equipment with intermittent failures.

Other important quality attributes that must be dealt with are safety with respect to the dose and the mechanical parts. A thorough design must be made for both of them. The dead-man principle is important for both of them. This principle means that only dose is generated or the system moves as long as the physician holds a button; when the button is released, the dose generation stops, or the system stops moving. For dose control, dose detection is in place. Similarly, collision detectors are applied, stopping the moving parts when a collision with the environment is detected.

An important organization for admission of such systems for amongst others the US market is the FDA (Food and Drug Administration). This organization imposes a number of legal requirements on such systems. For example, a button must be provided that can disconnect all equipment from power in case uncontrollable behavior occurs. This button can however lead to dangerous situations when used during an intervention. It is important for the FDA to validate that the legal requirements are met, making traceability an important quality (not shown in figure 13).

The above description focused on reliability and safety. More qualities for these systems and the relation to design concerns and implementation mechanisms are described in [5].

6 Related work

In the previous section, we described and illustrated our multi-view architecting approach. We focussed on an integral usage of views that go through the workflows. Below, we will describe a selection of related work that deals with multiple viewpoints in the various workflows and how this work is related to the ideas in our paper.

6.1 IEEE 1471

Figure 14: IEEE 1471 model

IEEE1471 [**?**] defines stakeholders, concerns, views, viewpoints and models, and the relationships between those concepts, see figure 14. The IEEE 1471 definition makes the relationship of **why** (stakeholders and their concerns) and **what** (described per viewpoints in models) explicit. It is a useful definition, because it acknowledges the fact that viewpoints are not orthogonal and not a priori consistent.

The threads of reasoning build upon these concepts to provide some means of navigating, integrating and

using these widely different concepts.

6.2 ISO 9126

- Functionality suitability, accuracy, interoperability, compliance, security, *traceability*
- Reliability maturity, fault tolerance, recoverability, *availability, degradability*
- Usability understandability, learnability, operability, *explicitness, customisability, attractivity, clarity, helpfullness, user-friendliness*
- Efficiency time behaviour, resource behaviour
- Maintainability
- Portability

Table 2: *ISO 9126 quality framework*

ISO 9126 [?] provides a quality framework, close to the emperical checklist we compiled ourselves. A classification in 6 categories is used, see 2.

7 Conclusions

8 Acknowledgements

We like to thank Pierre America, Hans Jonkers, Henk Obbink, and William van der Sterren for commenting on earlier versions of this paper. This research has been partially funded by ESAPS, project 99005 under Eureka Σ! 2023 Programme, ITEA.

References

[1] Philippe B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, pages 42–50, November 1995.

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

[3] Gerrit Muller. Case study: Medical imaging; from toolbox to product to platform. http://www.gaudisite.nl/MedicalImagingPaper.pdf, 2000.

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

[5] Jan Gerben Wijnstra. Quality attributes and aspects of a medical product family. submitted to the Software Track of the HICSS-34, January 2001 http://nlww.natlab.research.philips.com:8080/research/swa_group/wijnstra/ExternalPublications/hicss34/HICSSPaperJGW.pdf, 2000. This article describes the flow of non functional requirements to (software) implementation.

History
Version: 0, date: 2000 changed by: Gerrit Muller
- Created, no changelog yet