A3 Architecture Overviews

Focusing architectural knowledge to support evolution of complex systems*

P. Daniel Borches **G. Maarten Bonnema**
Laboratory of Design, Production and Management,
Department of Engineering Technology, University of Twente
P.O.Box 217, 7500 AE Enschede
The Netherlands
p.d.borches@ctw.utwente.nl g.m.bonnema@utwente.nl

Abstract. Creating complex systems from scratch is time consuming and costly, therefore a good development strategy often chosen by companies is to evolve existing systems. The understanding that a company has about the impact change has on the system determines its ability to cope with system evolution. Reuse of knowledge and experience becomes therefore, essential. Complex systems are usually the result of a multidisciplinary team, which means that an effective way to capture, organize and present this knowledge, in a fashion that can be used by different disciplines and departments is crucial. Typically, some of this knowledge is present in the form of text documents. However, much of that knowledge is usually lost or hidden, especially in long-lived systems. This leads to unexpected problems that could be prevented if the company had reused the knowledge it already has.

In this paper system evolution barriers are discussed, and a method to cope with them is provided. Some companies such as Toyota have already identified the advantages of using an A3 approach[2] to capture and share knowledge. We share this idea and take it several steps further. A method, based on the creation of 'A3 Architecture Overviews' is proposed to capture and share architectural knowledge. The method is the result of the work carried at Philips Healthcare MRI Group. We show that the proposed method to capture architectural knowledge provides an effective framework to support decision making when evolving complex systems.

Introduction

System requirements change over time; consequently, companies need to systematically evolve their products to cope with those changes. Since developing a system from scratch is time consuming and costly, new systems are often created by evolving an existing system (Suk suh et al. 2008; Borches and Bonnema 2009). However, the effort and resources required to adapt complex systems to changing requirements can be significant; if a change of requirements occurs, the effect can ripple through the entire system due to dependencies, known and hidden, in the system.

Hence, a company's ability to undertake and manage system evolution, can be influenced greatly by their understanding and knowledge of the impact that a change

*This work has been carried out as a part of the DARWIN project at Philips Healthcare under the responsibility of the Embedded Systems Institute. This project is partially supported by the Dutch Ministry of Economics Affairs under the BSIK program.

[2] A3 is an international paper size standard (American metric equivalent of 11" x 17").

has on the overall design and implementation (Isaac and McConaughy 1994; Steiner 1998). Companies already have a large amount of knowledge about the domain, partially implicit -mainly in the expert's minds- and partially explicit -in the form of documents and repositories-. Yet, few companies know how to capture knowledge effectively, and fewer companies know how to reuse that knowledge (Domb and Radeka 2009). Many people have difficulties sharing knowledge across organizational boundaries, and sometimes even with persons in the same office but different backgrounds. Decision making therefore fails to take advantage of the knowledge the company already has.

To provide guidance to architect new generations or variations of the system architectural knowledge is needed. However, this knowledge is usually not structured nor captured effectively. An effective way to capture this architectural knowledge, in a fashion that can be understood and accepted by all stakeholders is then required. The challenge is how to reduce the large amount of information spread within the company to manageable proportions, and how to present it in an easy to use way without losing essential details in the process. In the following sections the method to cope with system evolution by capturing this architectural knowledge, and observations from its application in a real industrial case -a Magnetic Resonance Imaging (MRI) system- are presented.

Industrial case: Magnetic Resonance Imaging (MRI) scanner

Magnetic resonance imaging (MRI) is an medical imaging modality that detects small changes in the magnetism of the atom's nucleus. An MRI system requires a multidisciplinary design team with competences in areas such as mechanics, electronics, physics, software and clinical science. The MRI process is in itself very difficult and involves many parameters and several domains (Weishaupt, Köchli, and Marincek 2006). All the disciplines have to work together on different aspects of the design. However, people are usually specialized in a single discipline, and each discipline uses its own vocabulary. This adds up to the complexity of the design process in a large company such as Philips. To illustrate this complexity, some indicative numbers of Philips Healthcare MRI products and its development are provided in Table 1. All this is excluding research, marketing and so on.

Parameter	Value
Developers	~250
Disciplines	Physics, Mechanics, Electronics, Software, Medical applications etc.
Development sites	3
Subsidiary sites	All around the world
Technologies	~50
Lines of SW code	$7*10^6$ (~10 different languages)

Table 1: Parameters of the MRI system development.

As shown in Figure 1, since Philips released the first commercial scanner back in the 80's, despite the challenges, Philips has successfully evolved it several times leading to the present system. The main architecture of the system and the design principles behind it have remained almost unchanged compared to the original system, while implementations and technologies used have changed completely in those 30 years.

If "*the test of a good architecture is that it will last*" (Robert Pinrad, 1993), we can argue that the Philips MRI architecture is a good one. The complexity of the system however, has increased rapidly. The need to reuse existing knowledge to provide guidance and prevent problems is more relevant than ever. Taking all this into account, we believe the MRI system is an ideal case to study the evolvability of complex systems.

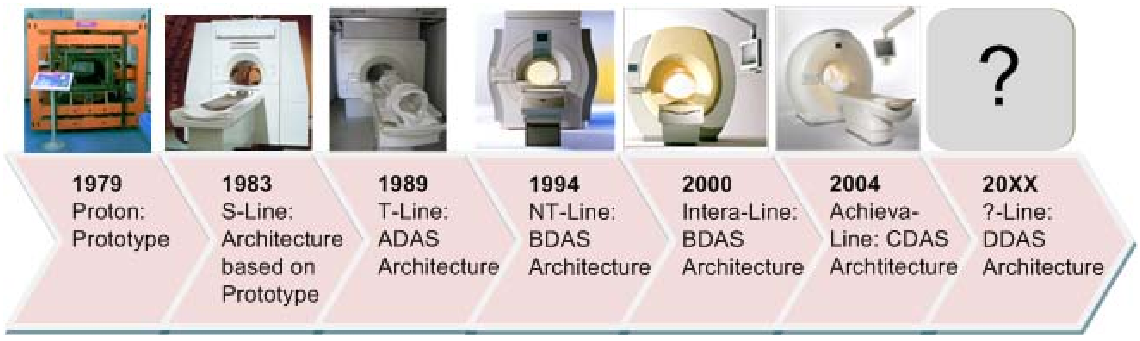

?
1979
Proton: Prototype
1983
S-Line: Architecture based on Prototype
1989
T-Line: ADAS Architecture
1994
NT-Line: BDAS Architecture
2000
Intera-Line: BDAS Architecture
2004
Achieva-Line: CDAS Archtitecture
20XX
?-Line: DDAS Architecture

Figure 1 Philips MRI Evolution (intermediate releases not included)

To cope with evolution of complex systems such as an MRI, we need a way to effectively capture and share architectural knowledge. For that, we propose an method to **collect**, **abstract** and **present** architectural knowledge to support decision making during the evolution of complex systems. The method is based on the creation of **A3 Architecture Overviews**. As will be discussed in the following chapters, the main goal of an A3 architecture overview is to have a manageable representation of the architectural knowledge related to a system aspect, enabling stakeholders to reason and communicate the consequences of system changes.

System Evolution and Architecture Descriptions

In technical sciences such as computer science, significant research has been carried out exploring the relation between design and evolution (MacCormack, Rusnak, and Baldwin 2008). This is due to the fact that software projects rarely start from scratch but rather prior versions are used as a platform. In the systems field, many years have passed since the first paper regarding 'system evolvability' was published (Simon 1962). System evolvability can be defined as "*system's ability to adapt to changing requirements and different environments throughout its lifespan in a time-efficient and cost-efficient way*" (Borches and Bonnema 2008). The importance of adopting evolvability has been discussed by several authors (Isaac and McConaughy 1994; Steiner 1998; Rowe and Leaney 1997; Christian III and Olds 2005; Ring and Fricke 1998) and its role in the system architecture has been described in (Isaac and McConaughy 1994; Steiner 1998). Since then some work has been done (Isaac and McConaughy 1994; Rowe and Leaney 1997, 1998; Christian III 2004; Christian III and Olds 2005), yet those approaches are usually theoretical and hard to apply in industry. In conclusion, although some attempts were made in the past, no satisfactory way to deal with system evolution exists and it is usually delegated to the architect's intuition.

As stated in (Rechtin and Maier 2000), "*if you do not understand the existing system, you can't be sure you're re-architecting a better one*", the first step towards evolving a system is understanding the existing system. (Churchman 1968) stated that "*how can we design improvement in large systems without understanding the whole system, and if the answer is that we cannot, how is it possible to understand the whole system?*". It is to say, to understand the existing system as a whole is essential to effectively evolve it.

To better understand the system we need to collect architectural knowledge of all relevant aspects of the system. Architectural knowledge is the kind of knowledge that provides guidance to architect new generations or variations of the system.

Then, a challenge to cope with system evolution then is how to effectively capture this architectural knowledge, and how to share it in a fashion that can be used by a broad set of stakeholders. Companies have a large amount of knowledge about the domain -both technical and contextual-, however that knowledge is only partially explicit (e.g. in the form of documents) while a majority of it is implicit (e.g. in expert's minds). To support correct decision making, companies need to take advantage of the knowledge they already have. By using this knowledge to understand and assess the consequences of changes, design effort can be directed towards avoiding undesired impacts and guiding the design of the system to one that enables easy evolution.

To better understand challenges to evolution in an industrial environment, and therefore be able later to assess the improvement or our method, we conducted a survey to Philips Healthcare employees. The target of this questionnaire was the MRI development organization (~250 employees), where 35 people (~1/7 of the population) filled in a questionnaire with ~50 questions. The questionnaire addressed the main development challenges and the effectiveness of the current way to capture architectural knowledge. For the analysis, a classification was made according to job title (HW/SW, architect, designer, engineer, domain expert, manager) and MRI experience. This questionnaire will not be discussed in detail in this paper, but the findings will be used to support some of the statements.

System Evolution Barriers

Barriers to evolution and challenges to design and develop complex systems can be found in literature (Ring and Fricke 1998), and they have to do mostly with handling complexity. Evolution pose a great challenge to companies, yet it is taken as a fact of live that is costly to change and therefore it doesn't deserve too much attention from management.

From the questionnaire it was found that main evolution barriers when dealing with new developments are; managing system **complexity**, **communication** across disciplines and departments, **knowledge sharing**, and finding the necessary **system information**. Those barriers, and specially the lack of knowledge sharing, were identified as the root cause of many development problems and bad decisions.

The reaction to the survey findings was, as stated by Philips' management, *"you have not discovered anything new; however you have shown that the magnitude of the issue is bigger than we thought, and consequently deserves more attention"*. After some discussions with representatives from other companies about those findings, we believe this is a common situation in most companies.

Architecture Descriptions

Architecture descriptions aim to ease those evolution barriers. IEEE 1471-2000 is an example of an architecture description standard. It provides with a very generic information model to guide the creation of architecture descriptions. Core idea behind it is that architecture description should consist on models. However, text documents are usually the way to document architectures and related knowledge.

Why should companies pay more attention to the design and representation of architectures? As supported by many authors, a good architecture, **well documented** and **accepted** within an organization, has many benefits: Faster time to market (TTM), easier to add new features, smaller crew, less mutual dependency between projects, more satisfied employees (they know more precisely which part of the system they are responsible for and how their work fits into the whole), less dependencies between new system releases, smaller and clearer development organization, more specific and accurate answers to questions of the market department, fewer problem reports (thus a larger satisfaction of our customers), lower service costs, etc. If a good architecture is not well documented, it can be rendered ineffective and lose those benefits.

For an existing system, this architecture description and design (should) already exist. However, in most companies, complex systems are typically poorly documented. The main architecture knowledge resides in the expert's minds, and only part of that knowledge is documented. Some key knowledge may be lost, especially in long-lived systems, due to experts leaving the company, design decisions and rationale not documented and so on. This means that usually the system architecture representation has to be, largely at least, reconstructed. Doing this is called **reverse architecting**.

Reverse Architecting

As stated in (Krikhaar 1997); *reverse architecting is a flavor of reverse engineering that concerns all activities for making existing architectures explicit, and the main goal of reverse engineering is to increase comprehensibility of the system for maintenance and new development.* Literature regarding reverse architecting is scarce, and it is mainly from software engineering and building architecture fields. Complex systems however are usually the creation of a multidisciplinary team. Yet we believe the lessons learned from reverse architecting in other fields such as software engineering (Mayrhauser, Wang, and Li 1999) and building architecture (Galal-Edeen 2002) apply to the systems engineering discipline.

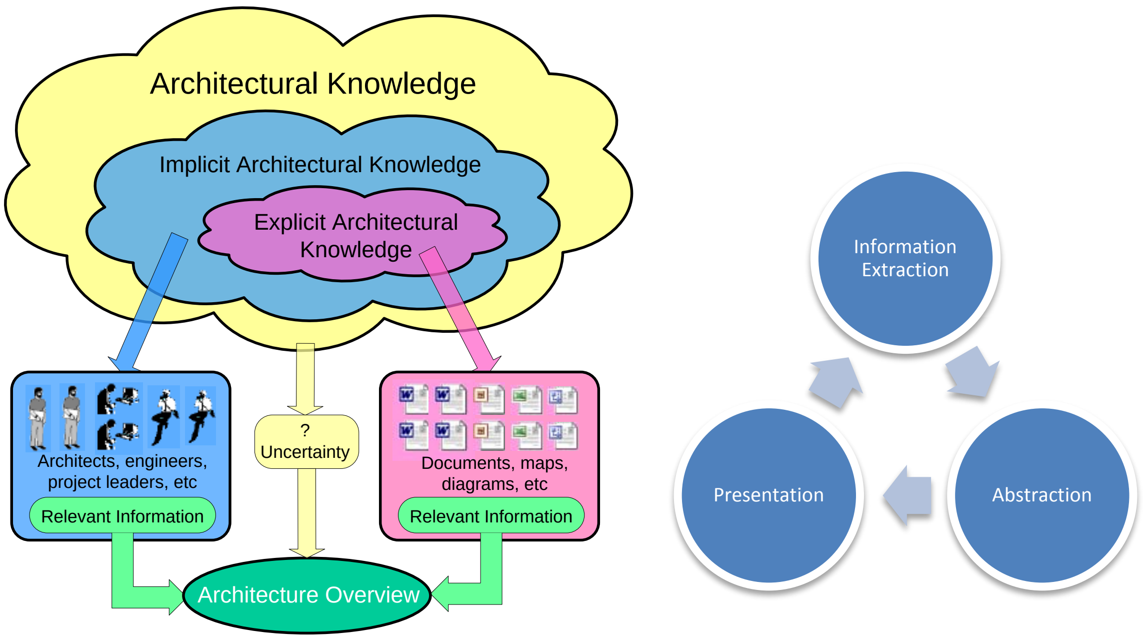


Figure 2 Reverse Architecting as a means to recover architectural knowledge

Reverse Engineering is the process of analyzing a subject system to identify the system's components and their interrelationships, and to create representations of the system in another form or at a higher level of abstraction that are less implementation-dependent. It is a process of examination, not a process of change or replication (Chikofsky and Cross II 1990). In (Muller 1996), it is mentioned that the reverse engineering of a system consists of three phases; **information extraction**, **abstraction** and **presentation.** As shown in Figure 2, to reverse architect a system the same process can be applied in order to create an architecture overview. This is an iterative process; the information extracted is abstracted and presented many times, and then new findings are incorporated in each iteration. Although not explicitly mentioned, after several iterations the architecture overview becomes stable enough (no information is added nor removed). Then, we can say that the architecture overview is validated when no new information is added and no information is removed in the process.

Information extraction

In complex systems the architectural body of knowledge is abundant. Part of that knowledge is documented and conforms the architecture description of the system. Documents, diagrams and system history are the most common forms of storing this information. Another rich source of architectural knowledge is the architects, experts and other employees who can be interviewed for this purpose. It should be noticed that recovering the complete architectural knowledge is an illusion (Borches and Bonnema 2009), there will be always some degree of uncertainty that will be translated into the architecture representation. This uncertainty is reduced as new insights are discovered in the process. The challenge is to extract effectively relevant information from those sources, and not be deviated during the process. Documents and similar sources are used as a starting point to create the draft architecture representation, which will be used as a baseline for the rest of the process. It is not possible to recover all architectural knowledge at once, it is necessary to focus on specific system aspects at a time. The initial draft will evolve during the reverse architecting process, and it is used as a discussion tool to extract information from experts; it is much easier for an expert to modify a wrong representation than creating one from scratch.

Abstraction

Abstraction consists of grouping and filtering extracted information to obtain a manageable set of information[3]. During the information extraction phase, keeping the essentials and removing the unnecessary information is vital. Humans can only handle a limited amount of information, therefore it is important to provide only the necessary information. An overview provides a broad and comprehensive view. In an overview, only relevant information is present (Bonnema and Borches 2008). The challenge is how to determine what information is relevant. An A3 (metric equivalent of 11” x 17”) can be used as a means to keep only relevant information and provide overview.

A well known example is the ‘A3 report’, which derives its name from the paper size used to print such reports. An A3 report is a tool that Toyota Motor Corporation uses to propose solutions to problems, give status reports on ongoing projects, and report results of information gathering activity. Toyota uses it to systematically guide problem-solvers through a rigorous process, document the key outcomes of that process and propose

[3] It is important to distinguish between levels of abstraction. Different parts of the life-cycle may require different levels of detail.

improvements. Some companies have already identified the advantages of the A3 thinking approach as a way to encourage focus and brevity to enhance communication (Sobek II and Jimmerson 2004, 2005). However so far it has been used mainly as a report or analysis tool at the production floor rather than an architecting method.

The A3 paper size works well for presenting the essential elements of a system topic, with enough information to make a decision about it. Larger sizes contain too much information, and the large paper format can become cumbersome. An A3 has enough room for a concise chunk of knowledge and fits well within the average person's field of view. Readers may focus on one part of it at a time, but they can always see the whole. The guiding principle behind the A3 is to include whatever information is needed to create a complete picture of the issue at hand, and eliminate everything else until only the essentials remain.

Presentation

It is difficult to determine what makes a good architecture representation. It is not without reason that there are so many architecture frameworks. What we do know is that a useful architecture representation has different views of the system (Zachman 1987; Muller 2006). The most common view in any architectural representation is probably a block diagram consisting of physical components and their interfaces. For complex systems however, as supported by authors such as (Shaw 1989), other high-level abstraction views that are independent of the components should also be included. From (Retching and Maier 2000) we know that "*except for good and sufficient reasons, functional and physical structuring should match*", and from (Bass, Clements, and Kazman 2003) we know that *"the architectural design of a system can be described from (at least) three perspectives: functional partitioning of its domain of interest, its structure, and the allocation of domain-function to that structure*". This supports the idea that an architectural representation needs at least a physical and a functional view.

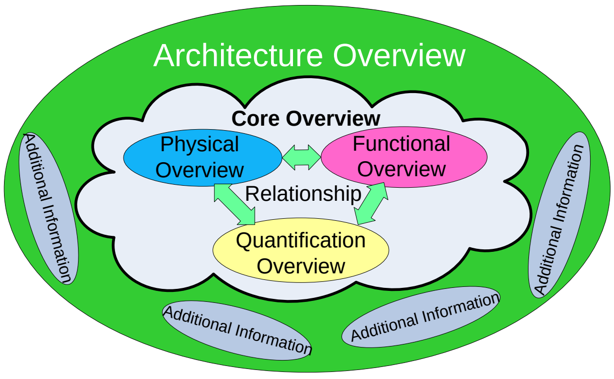


Figure 3 Architecture Overview elements

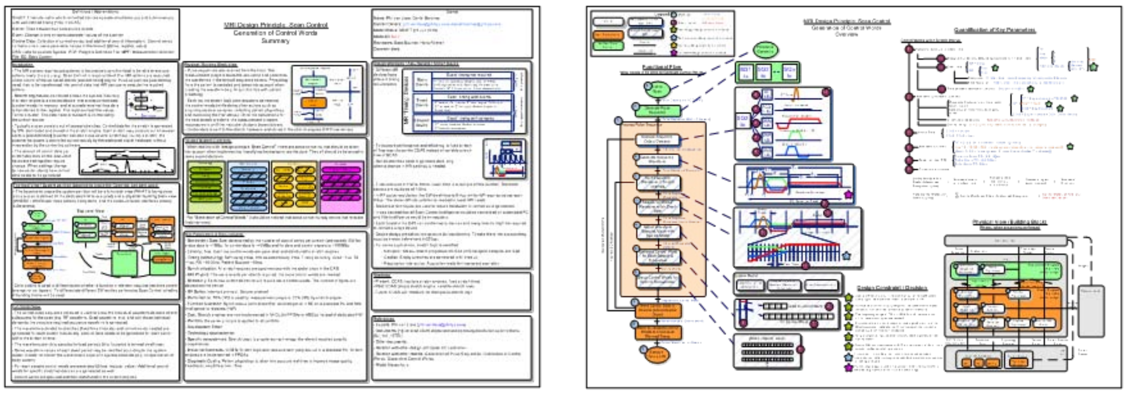
As shown in Figure 3, what architecture representations usually lack is a quantification view; it is to say, figures of merit of key system parameters to support the other views. Numbers are needed to grasp the relevance of the issue at hand. In addition, in a real system, many instantiations deviate from the ideal architecture, and there is always some design constraints or choices that are imposed to the architecture such business drivers, technology choices, etc. This additional information should also be included in the architecture representation. These views; physical, functional and quantification, should not be isolated from each other, mapping of elements among them is necessary.

Once we know what to include in the architecture representation, the question that often arises in the mind of the architect is whether the architectural knowledge should be captured in a model[4] or in text. Standards such as IEEE 1471-2000 encourage the use of models, and in our survey we found that this is also preferred by most disciplines but maybe Managers and Domain Experts. However, as we will discuss later, this doesn't mean that we can forget about the text and provide just models; text to support the models is needed. We found that models that do not provide textual descriptions usually ended up embedded in related documents losing their utility -overview is lost-. In conclusion, a model view is preferred, but text is needed to support the model to prevent the need for a document to contain the model.

Another point of discussion is whether or not formal notation (e.g. SysML or UML) should be used in the architecture representation. From the survey we found that this is supported mostly by employees with a software background, who are more used to formal notations, while other disciplines prefer not to use formal notations. This is also corroborated by our own experiments, in which SysML models were created to model the same views that are part of our current architecture representation. The main issue with them was that most of the meetings with experts was spent discussing the notation itself rather than the content. Although the benefits of formal notations are widely recognized, for the purpose of sharing architectural knowledge where communication among stakeholders is a major goal, we believe they are more a barrier than a support. Formal notations become more useful as the process moves down into detail design, in which having a formal and correct notation provides many benefits.

A3 Architecture Overviews

From the psychology field we have learnt that; *"If the concepts in the mind of one person are very different from those in the mind of the other, there is no <u>common model</u> of the topic and no communication"* (Taylor and Fiske 1975). Hence, for effective communication a common model that represents the system close to the concept in the mind of the reader is needed. The most widely-used form of communication is the drawing. Drawings range from rather general descriptions that give an 'overview' of the system, to the most specific that use precise details. Because all of them have to communicate, all drawings must be subject to agreed rules, codes and conventions. Learning how to read and make those drawings is an important part of education.



a) A3 Textual Overview **b) A3 Model Overview**

Figure 4 A3 Architecture Overview example

[4] Model: An approximation, representation or idealization of selected aspects of the structure, behaviour, operation or characteristics of a real-world process, concept or system (IEEE 610.12-1990).

We discussed in the previous section that to deal with architectures we need different views of the system. At the same time, in order to keep the overview, we need to have those views within the A3. We have expanded the A3 approach by introducing structure and guidelines to its presentation, in order to effectively display the architectural knowledge. As shown in Figure 4, unlike the A3 reports, we provide structure and use both sides of the A3 to create the architecture overview. One side displays the model view of the system aspect, while the other displays the textual view to support and complement the model view. In this way we close the 'gap' between a model description and a text description of the system, by using both in combination. This may seem as a lot of information for a single sheet of paper. In practice, the A3 Architecture Overview contains only the essentials, and it may serve as a baseline for a more comprehensive description with more detailed information. The amount of paper -if printed- is the same, so it does not overload the reader -reader can't read both sides at the same time-.

Structure and Elements of an A3 Architecture Overview

Providing structure to the A3 improves readability and comprehension. Common elements in the A3 help the reader identify at a glance whether or not the architecture overview of a system aspect is of any interest, and quickly locate where the specific information is.

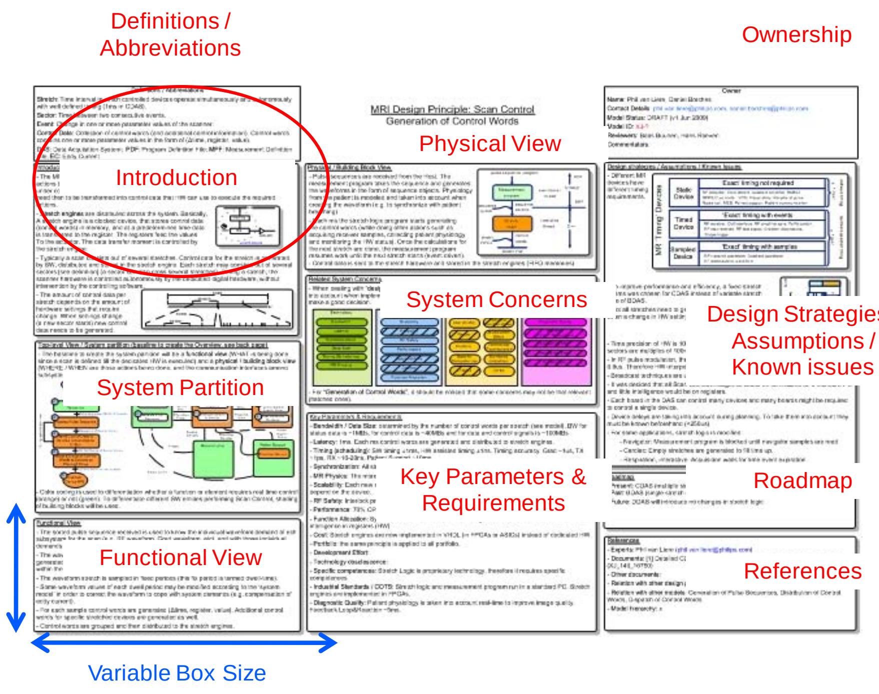

Definitions / Abbreviations
Ownership
Physical View
Introduction
System Concerns
Design Strategies / Assumptions / Known issues
System Partition
Key Parameters & Requirements
Roadmap
Functional View
References
Variable Box Size
MRI Design Principle: Scan Control
Generation of Control Words

Figure 5 A3 Text Summary Structure and Elements (image not readable due to confidentiality reasons)

As shown in Figure 5, the A3 text summary has the following elements[5]:

- **Definitions / abbreviations:** Definitions of concepts used within the A3 Architecture Overview as well as the list of abbreviations.

[5] Visual models and pictures are, as they say, worth a thousand word each, thus, as the room to display information is limited, it is encouraged to use them also in the text summary.

- **Introduction:** Description of the context of the topic under discussion.
- **System Partition:** For each system aspect an appropriate system partition is created -which may also be used also by other A3s discussing topics related with this system aspect-. Here the partitioning is described and the notation used (e.g. the color coding). The system partition consists on a functional and physical view of the system.
- **Functional[6] View:** Textual information to support the functional view in the model view.
- **Physical View:** Textual information to support the physical view in the model view.
- **System Concerns:** The top level concerns related with the system aspect are captured in a model and presented here. A 4-column model; technology, functional, business and customer concerns is used. They are collected at the beginning of the creation process and used later for the key parameters and requirements elicitation. When used in combination with other A3s describing related system aspects, it supports the creation of budgets.
- **Key Parameters & Requirements:** The topic under study contributes to specific system concerns (e.g. requirements are imposed on the system concerns). Related requirements and key parameters values (e.g. performance, size) are captured here.
- **Ownership[7]:** The person in charge of maintaining this architecture overview is provided (and the contact details), as well as the version of this overview, the status (draft, reviewed, approved), reviewers, commentators, document ID, etc.
- **Design Strategies / Assumptions / Known issues:** The specific strategies applied to deal with the issue at hand are described here, as well as the assumptions used (e.g. power is not lost in heat dissipation) and known issues (e.g. network may overload if broadcast not used).
- **Roadmap:** Current, past and (possible) future instantiations that apply, and the reasons for change.
- **References:** Sources to broaden the knowledge are provided; experts (and their contact details), documents (and their identification numbers), relation with other system aspects, relation with other A3s, and the hierarchy of this A3.

It should be noticed that the box size in Figure 5 is variable. This means that if more room is needed for one element it can be expanded. However as the A3 limitation is imposed to maintain the overview, this implies that there will be less room for other elements. Elements in the A3 are distributed in columns, from left to right. The order of this elements is structured according to the user needs; from little detail to more detail.

[6] The term 'Function' can be interpreted in many ways. We have used this term in the sense *"a specific or discrete action that it is necessary to achieve a given objective"* (Blanchard and Frabryky 1998)

[7] For maintainbility reasons, all A3s are not created by one single person. One person should be responsible of collecting all A3s and establish the links among them, as well as their hierarchy. However the creation of individual A3s and their update should be the responsibility of individuals related to an A3 topic.

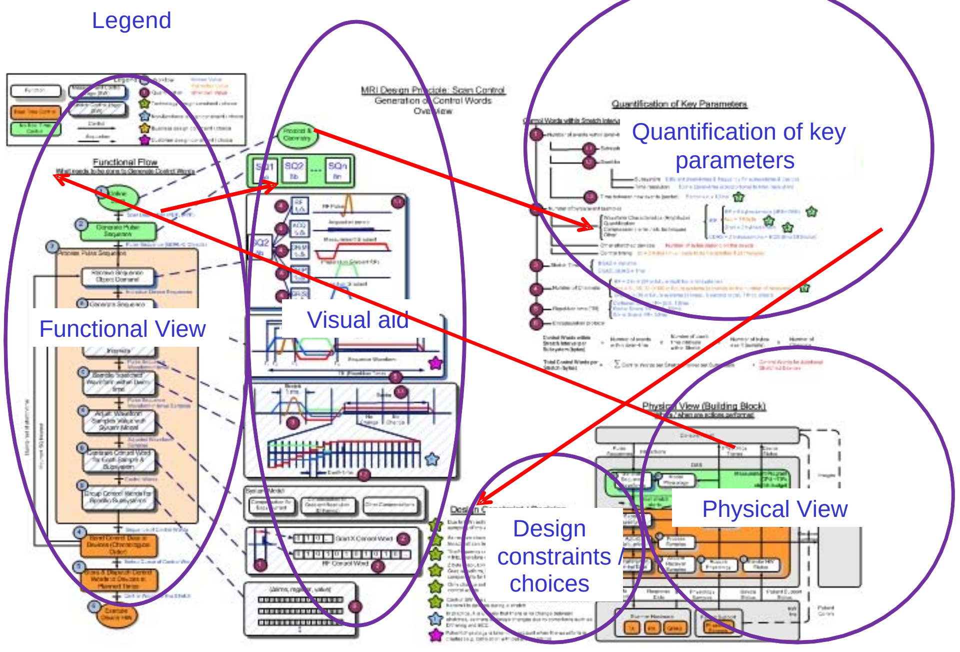

Legend
Functional View
Visual aid
Quantification of key parameters
Physical View
Design constraints / choices

Figure 6 A3 Model Overview Structure and Elements (image not readable due to confidentiality reasons)

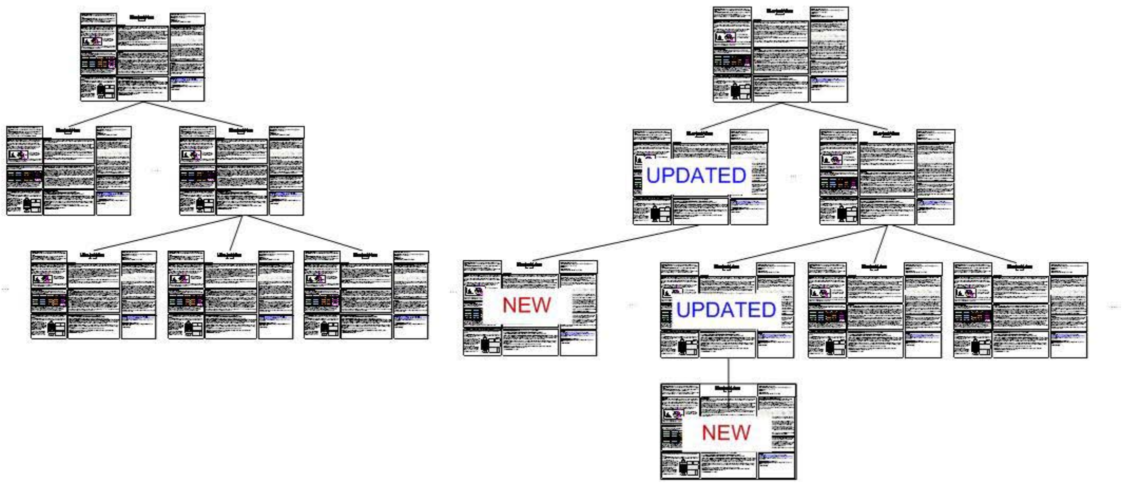
As shown in Figure 6, the A3 model overview has the following elements:

- **Functional view:** The set of basic steps required to perform the action related to the topic under study. Boxes represent functions and the text within is in the form verb+noun. It should be noticed that this functional view (or functional flow) is an extension of the system partition chosen for this system aspect.
- **Visual aid:** As functions are sometimes confusing or ambiguous, and in order to come closer to the mental model of the reader, pictures or visual representations are used to describe the function and its context.
- **Physical view:** Typically a block diagram consisting of physical elements (software in this sense is a 'physical' element) and their interfaces. This view is also an extension of the system partition.
- **Quantification of key parameters:** Numbers are needed to support the other views. Some key parameters need further decomposition, that can be related to the views. Measurements, experts estimations and best guesses -if nothing more is available- are included (with different colors).
- **Design constraints / choices:** No system is ideal. There are usually some constraints imposed in the design (e.g. technology, functional, business, customer) that should be taken into account are mentioned here.
- **Legend:** Describes the notation used in this A3. Color coding, shading, shapes and other element used to help the reader visualize the information are described here.

As represented by the red arrows in the Figure 6, the views within the A3 are not isolated from one another. Links among the views (e.g. by using number links) enable to map one view to another. The predominant view on the A3 is the functional view. As functions are usually more stable over time than implementations, it is likely that this view will remain, even if the implementation changes.

A3 Architecture Overviews as Repository of Architectural Knowledge

The set of A3 architecture overviews forms a repository of architectural knowledge. Each A3 architecture overview is an independent chunk of architectural knowledge that focus on a specific system aspect, yet to get an even broader picture, or to understand cross system concerns, more A3s can be used; A3s are linked to other A3s and arranged in hierarchy (described in the reference section of the A3 textual overview).



a) Set of A3 architecture overviews **b) Extended set of A3 architecture overviews**

Figure 7 A3 repository

As shown in Figure 7, this repository of architectural knowledge can be expanded and updated easily. New A3s can be created and added while those that need to be updated can be modified without having to modify the rest.

Application: Philips MRI New Style SDS

A System Design Specification (SDS) is used by companies in the development process to consolidate design specifications, to support ‘development memory’ and for educational purposes. It is usually the **main description of the system’s design**. It is meant to specify how requirements are met. It serves to consolidate the partitioning of the system into design entities; mapping of requirements onto system elements and define interfaces between them, budget between components when they need to cooperate, description of the behaviour, etc. They vary in structure from company to company, but they all have one thing in common; they are large text documents with few drawings in them.

SDS acceptance and use within Philips MRI

Consolidating the design of a complex system such as an MRI is difficult, therefore it requires a large SDS document. Despite the great effort required to update and maintain the SDS document for different system releases, it was found from the survey that it was seldom used. Architects, designers and engineers are the expected users of the SDS, however while architects were familiar with the SDS, designers and engineers were not. Surprisingly, while architects have their way to find the information they need without relying on the SDS, for designers and engineers is very difficult to obtain the system information they need for their work, and the SDS is of little support for them. The main users of the SDS were found to be managers, which see the SDS as the main source to gain insight on the system.

Regarding whether the SDS provided enough architectural knowledge outside the user's domain of expertise, the majority thought it did not. The lack of architectural knowledge was identified in the survey as the root cause of many problems and poor decisions. Communication across disciplines and departments as well as MRI system complexity was perceived as the main problem for most employees when dealing with new developments. The SDS was found of little use in this sense. The SDS was too detailed for most users but domain experts, leading to communication barriers. SDS was found useful for new employees as a means to learn about the system.

Having a system overview was considered very important to support development by all disciplines, however only half of the people thought the SDS provided the system overview they needed. Managers do obtain the overview they need from the SDS, however it did not provide overview to architects, and little overview to designers and engineers. It was concluded from the survey that a change in the syle was desired. While half of the manages and all domain experts preferred to continue with a text-based document, almost all architects, designers, and many engineers preferred a model-based description for the SDS.

New Style: A3 Architecture Overview SDS

From all mentioned above, it was decided that the A3 Architecture Overview style would be introduced as a way to describe the system and become the new SDS style. A 'proof of concept SDS' was developed to assess the benefits and concerns of the new style. After an initial discussion with main MRI system architects, 16 system aspects were considered essential for the MRI system description. It was estimated that each of those system aspects may need between one and five A3s once divided into topics. Then, a initial amount of 40-60 A3 architecture overviews were estimated to describe every relevant system aspect of the MRI system. This may look as a lot of work (previous SDS document was 200 pages long), however it should be noticed that A3s describing a system aspect are created by different individuals or teams.

Two topics were selected for the SDS concept; scan control and MRI calibrations. These system aspects were chosen as they were complex aspects that crossed system and organizational boundaries, resulting in interesting study cases. Scan control is in charge of controlling the different subsystems in a timely and synchronized fashion. MRI systems need to have precise timing and be synchronized to the nanosecond, requiring complex solutions. Three A3 architecture overviews; *Generation of control data*, *distribution of control data* and *store and dispatch of control* data were created. MRI calibrations is in charge of correcting imperfections and tune different parameters for optimal image quality. Two A3 architecture overviews were created for two of the main MRI calibrations; *Resonance frequency calibration* and *eddy current calibration*. Although they may seem independent system aspects, a change in an MRI calibration parameter has an impact on the generation of control data, resulting in links among A3s.

After the proof of concept SDS was created, accepted and validated, a workshop with system architects and experts was organized to complete the SDS. A short training in the creation of A3 was given and there was some coaching by the main author. An owner for each system aspect was assigned, and the task to create the A3 and maintain it was given. To support the creation of those A3 architecture overviews after the workshop, a cookbook was provided. The cookbook (and A3 itself) described the process and provided guidelines to the creation process. The SDS owner is now in charge of collecting the A3s and maintain the relations among them.

Benefits and Concerns of the New Style

Although the complete SDS is not yet finished by the time of publication of this article, the A3s developed for the SDS concept were used at some projects. It was stated by users that one of the main values of the A3 architecture overviews is that they enable to get more insight on the system. This in itself has a great value during the development process, *"one insight is worth a thousand analysis"* (Charles W. Sooter, 1993).

For project meetings, A3 architecture overviews were populated among the members. The first reaction of those not used to the A3 layout was to complain about the new format (hard to fit in the screen, problems with the printers, etc). However in later meetings we found out that all had read and studied the A3 provided (maybe out of curiosity), while they didn't read the equivalent text document that was also provided. This situation happened several times, leading us to the conclusion that the A3 is the maximum amount of information employees are willing to read to prepare for a meeting or discussion.

In addition, people attended meetings with plenty of annotations in the A3, triggering discussions and improving the A3 contents, proving that it is a good tool for rich discussions. The model view helped discussions while the text part helped broaden the information on individual use. People of different backgrounds were able to use them without much explanation. Unlike the equivalent views provided in SysML formats, content discussions in the A3 format started right away.

From the management point of view, it become clear that this new style improved maintainbility and upgradeability of the design specification. New A3s could be added without having to touch the ones already created, and future SDS could reuse much of existing A3s.

Conclusions

A company's ability to cope with system evolution is influenced by its understanding of the impact that a change has on the overall design and implementation. Companies already have a large amount of knowledge about the domain, yet few companies know how to capture knowledge effectively and how to reuse that knowledge.

A challenge to cope with system evolution is how to effectively capture architectural knowledge, and how to share it in a fashion that can provide guidance to architect new system generations. By understanding and assessing the consequences of changes, design effort can be directed towards avoiding undesired impacts and guiding the design of the system to the one that enables easy evolution.

From a survey performed at Philips MRI, it was found that main evolution barriers when dealing with new developments were; dealing with **complexity**, **communication** across disciplines and departments, **knowledge sharing**, and finding the necessary **system information**. Those barriers were identified as the root cause of many development problems and bad decisions.

Architecture descriptions aim to ease evolution barriers. Having a well documented architecture poses many benefits to the company. However, the main architecture knowledge resides in the expert's minds, and only part of that knowledge is documented. This means that the system architecture representation has to be, largely at least, reconstructed.

Reverse architecting enables recovering the architectural knowledge. It is an iterative process which consists of three phases; information extraction, abstraction and presentation. This process leads to the creation of A3 architecture overviews.

An A3 Architecture Overview is an architecture representation that provides a manageable view of a system aspect. It displays the architectural knowledge related to that system aspect in both sides of an A3 sheet; one side with a textual view and the other with a model view. Views are structured and provide common elements to enhance readability and comprehension.

The A3 method has successfully been applied as a new style of the Philips MRI system design specification. Architectural knowledge of different system aspects were captured using this method. The results prove that the A3 architecture overview, is an effective tool for communication among diverse disciplines and departments, and enables gaining deeper insight into a system.

Acknowledgements

The research is conducted in cooperation with the MRI group of Philips Healthcare. We thank all the experts from Philips Healthcare for their support and contributions. We want to thank specially Phil van Liere for the time and effort dedicated to this project. We also want to thank the support of Gerrit Muller and other members of the Darwin project.

References

Bass, L., P. C. Clements, and R. Kazman. 2003. *Software Architecture in Practice*. Reading, MA.: Addison-Wesley.

Blanchard, B. S., and W. J. Frabryky. 1998. *Systems Engineering and Analysis*. Edited by U. S. River. New Yersey: Prentice Hall.

Borches, P. D., and G. M. Bonnema. 2008. On the Origin of Evolvable Systems: Evolvability or Extinction. *Proceedings of the TMCE* 2:1351-1353.

———. 2009. Coping with System Evolution- Experiences in Reverse Architecting as a Means to Ease the Evolution of Complex Systems. Paper read at 19th Annual International INCOSE Symposium, at Singapore.

Chikofsky, E. J., and J. H. Cross II. 1990. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software* 7 (1):13-17.

Christian III, John A., and John R. Olds. 2005. A Quantitative Methodology for Identifying Evolvable Space Systems. In *1st AIAA Space Exploration Conference January 2005, Orlando, FL*. Orlando, FL.: Georgia Institute of Technology.

Churchman, C. W. 1968. *Challenge to Reason*. Edited by Chichester, *Sussex*. New York: Wiley.

Domb, E., and Katherine Radeka. 2009. LAMDA and TRIZ: Knowledge Sharing Across the Enterprise. *TRIZ Journal*.

Galal-Edeen, G. H. 2002. Reverse architecting: seeking the architectonic. Paper read at Ninth Working Conference on Reverse Engineering

Isaac, D. , and G. McConaughy. 1994. The Role of Architecture and Evolvability Development in Accommodating Change. *INCOSE'94*:541-545.

Krikhaar, R. L. 1997. Reverse architecting approach for complex systems. Paper read at IEEE International Conference on Software Maintenance, at Bari.

MacCormack, A., J. Rusnak, and C. Baldwin. 2008. The Impact of Component Modularity on Design Evolution: Evidence from the Software Industry: Harvard University.

Mayrhauser, A. von, J. Wang, and Q. Li. 1999. Experience with a reverse architecture approach to increase understanding. Paper read at IEEE International Conference on Software Maintenance, at Oxford.

Muller, G. *How to Create an Architecture Overview* 2006 [cited. Available from www.gaudisite.nl.

Muller, H. A. 1996. Understanding software systems using reverse engineering technologies research and practice. Paper read at Tutorial 18th ICSE at Berlin.

Percivall, G.S. 1994. System Architecture for Evolutionary System Development. In *Proceedings of the 4th Annual Symposium of the National Council on Systems Engineering*. San Jose.

Rechtin, E., and R. W. Maier. 2000. *The Art of Systems Architecting*: CRC.

Ring, J., and E. Fricke. 1998. Rapid Evolution of All Your Systems - Problem or Opportunity? *Proceedings of IEEE 17th DASC, Seattle*.

Rowe, David, and John Leaney. 1997. Evaluating evolvability of computer based systems architectures - an ontological approach. *IEEE Proc ECBS*:360-367.

Shaw, Mary. 1989. Larger scale systems require higher-level abstractions. *Proceedings of the 5th International Workshop on Software Specification and Design*.

Simon, H. A. 1962. The Architecture of Complexity. Paper read at American Philosophical Society.

Sobek II, Durward K. 1997. Principles that Shape Product Development Systems: A Toyota-Chrysler Comparison, University of Michigan, Ann Arbor.

Sobek II, Durward K., and Cindy Jimmerson. 2004. A3 Reports: Tool for Process Improvement. Paper read at Industrial Engineering Research Conference, at Houston.

———. 2005. A3 Reports: Tool for Organizational Transformation. Paper read at Industrial Engineering Research Conference.

Steiner, R. 1998. Systems Architecture and Evolvability - Definitions and Perspective. *Proceedings of the 8th Annual Symposium of the International Council on System Engineering*.

Suk suh, Eun, Michael R. Furst, Kenneth J. Mihalyov, and Oliver L. de Weck. 2008. Technology Infusion: An Assessment Framework and Case Study. Paper read at International Design Engineering, at New York, USA.

Taylor, S. E., and S. T. Fiske. 1975. Point of view and perceptions of causality. *Journal of Personality and Social Psychology* 32 (3):439-445.

Weishaupt, D., V. D. Köchli, and B. Marincek. 2006. *How does MRI work*. Edited by Springer-Verlag.

Womack, J., D.T. Jones, and D. Ross. 1990. *The Machine that Changed the World: The Story of Lean Production* Edited by HaperPerennial. New York.

Zachman, J. 1987. A framework for information systems architecture. *IBM Systems Journal* 26:3.

Biography

P. Daniel Borches received his Master's degree in Telecommunication Engineering from the University of Madrid Carlos III in 2004. He has worked several years in companies such as Telefonica R&D and Nokia. From 2006 he is working at Philips Healthcare Netherlands while doing his Ph.D. at the University of Twente. The main focus of his research is Systems Engineering and Systems Architecting applied in the industrial sector.

G. Maarten Bonnema is an assistant professor at the Laboratory of Design, Production and Management of the Faculty of Engineering Technology at the University of Twente. He studied Electronic Engineering, and did a two year designer course on Technical Systems. He has worked as a Systems Engineer at ASML. His research involves supporting system designers, conceptual design and complex system design.