Real World Influences on Software Architecture - Interviews with Industrial System Experts

Goran Mustapic, Anders Wall, Christer Norström, Ivica Crnkovic,
Kristian Sandström, Joakim Fröberg, Johan Andersson
Dept. of Computer Science and Engineering, Mälardalen University
PO Box 883, SE-721 23 Västerås, Sweden
{goran.mustapic, anders.wall, christer.norstrom, ivica.crnkovic,
kristian.sandstrom, joakim.froberg, johan.x.andersson}@mdh.se

Abstract

Industrial systems are examples of complex and often long-lived systems in which software is playing an increasingly important role. Their architectures play a crucial role in maintaining the properties of such systems during their entire life cycle. In this paper, we present the results of a case study based on a series of interviews and a workshop with key personnel from research and development groups of successful international companies in their Swedish locations. The main goal of the investigation was to find the significant factors which influence system and software architectures and to find similarities and differences between the architecture-determining decisions and the architectures of these systems. The role of the architect was an important subject of the investigation. Our findings result in recommendations relating to the design and evolution of system architectures and suggestions regarding areas in which future research would be beneficial.

1. Introduction

There are many large and complex software-intensive systems which have been successful, not only in commercial terms, but in providing reliable bases for several products over many years. One of the key factors in successfully managing a system, i.e. maintaining the system, introducing new features etc. is its architecture. There are several factors that distinguish the management of large and complex industrial systems from the management of smaller systems. The requirements of smaller systems can often originate with and be understood by a single person, while the requirements of large systems have many dimensions and involve many stakeholders, which makes them much more complicated and difficult to grasp and manage.

It is not feasible to study the important architectural factors affecting large systems by constructing and reasoning about small "toy-systems". The architectural work in real systems has so many different aspects that it is unrealistic to experiment on system models and to expect to draw conclusions of value. Large and complex systems can have a lifetime of 20-30 years, which makes experimenting with these systems even more difficult. A realistic study of factors important for successful system management requires that we can study a system over a long period of time, or at least have access to a reliable source of information regarding the history of the system.

We have studied seven large and complex industrial systems in which software has an important and expanding role. We looked at several aspects of these systems related to their architecture at different points in their lifecycles, i.e. requirements, design and implementation of the system, system evolution and retirement. Among many others, the following issues are analyzed in their relationship to architectural decisions: reuse, legacy, the effects of the choice of technology, standards, organization and development process.

Even though we have primarily focused on software architecture, it is important to have in mind that a typical industrial system incorporates computer hardware, other hardware and software. One of our goals was to find out how much importance is given to software architecture in the design of these systems. In [1], the authors "persist in speaking about software architecture primarily, not system architecture, …, because most of the freedom is in software choices, not hardware choices." However for the systems we have studied, the following statement by Maier and Rechtin [6] may be more appropriate: "Even if 90% of the

system-specific engineering effort is put into software, … it is the system, not the software inside that the client wishes to acquire."

The systems we have studied are: the electronic control systems of cars and construction equipment provided by Volvo Car Cooperation and Volvo Construction Equipment respectively, a robot control system provided by ABB Automation Technologies AB/Robotics, a train control system provided by Bombardier Transportation, the software system of radio base stations for 3G provided by TietoEnator Telecom & Media, and the Ericsson telecom system and platforms of some of its nodes. The study is based on interviews with specialists in the system and software architectures of the companies' system and software development organizations. The results from the interviews were summarized and further discussed during a workshop in which the interviewed architects participated.

Most software architecture-related research is focused on architectural analysis, architectural descriptions, and tools. Consequently, most of the documented case studies are focused on these issues [9] [2]. Other reports present the utilization of related architectural activities in the software development process in terms of descriptions and analysis [10]. A case study similar to ours but of relatively limited scope is presented in [5]. We have found no other relevant work that addresses important factors related to the software architecture in successful software intensive industrial companies to the same extent as this paper.

The contribution of this paper is a description of the state-of-practice in industry with respect to software architecture and a set of observations and findings from an analysis of interviews and a workshop with the chief architects from these companies. The findings include architecture-related differences between and similarities of the systems studied. Moreover, we provide speculative findings and trends that were not explicitly found in the case study, but were rather the results of the intuitive reasoning of the interviewers.

The outline of the paper is the following: in Section 2 we present a detailed description of the method used in performing the interviews. In Section 3 we give a brief overview of the systems studied. Section 4 presents a comparative analysis of the data collected in our case study. Section 5 contains our conclusions and some suggestions for our future work in this field. We express our gratitude to the individuals who participated in the project and their companies in Section 6 and provide a list of references in Section 7.

2. The case study setup

The case study performed was an investigation of the architecture of software. According to [8] the following steps are involved in a case study: conception, hypothesis setting (particularly important as it describes what we measure and how the results are analyzed), design, preparation, execution, analysis, dissemination and decision-making. Also, because a case study usually compares one situation with another, some measures must be taken to avoid bias and to make sure that hypothetical relationships are tested. Possible techniques are: sister project, comparison with a general baseline and random selection. We have performed these general steps in the following way. In the preparation phase of the case study we held several brainstorming sessions. The outcome of these meetings was a plan with the following agenda:

- Formulate a list of question to guide the interview.
- Request and arrange interviews with people who have an architect, chief designer or similar important role and know the history of the system. The interview was estimated to take at least two hours, and was conducted by one or preferably several of the authors of this paper. We decided to use the following criteria in the selection of companies to participate in the case study:
  - Successful products on the market
  - Complex, established industrial products in which software is an important part of the entire system
  - Availability of specialist architects
  - Availability of data from several system generations
- After the interviews, write summaries to send to the architects concerned for review.
- Assemble the results of the interviews in a technical report.
- Make a preliminary analysis of the interviews and determine the similarities of and the differences between the cases.
- Organize a workshop with the interviewees and perform a further analysis of the results.
- Publish the results of the case study as a scientific article.

The purpose of submitting a questionnaire was to give a common structure to all the interviews and to form a basis for comparison and analysis of the results. The interviews were not restricted to the questions in the questionnaire. Wherever possible, we advanced counter arguments to provoke discussion, well aware that some of the statements were a summary of the

interviewee's experience with multiple systems over a period longer than 10-15 years. In some cases, we have conducted several interview sessions to penetrate different subjects and e.mail exchanges to clarify some uncertainties.

3. Systems overviews

In this section, we present a brief overview of the systems covered by the case study. The way we present the system overviews reflects the views and responsibilities of the interviewees.

3.1 ABB Automation Technologies/ Robotics

ABB Automation Technologies Products – Robotics is a manufacturer of industrial robotic systems. The interviewee is the chief system architect for the two most recent system generations. Industrial robots are systems consisting of one or more mechanical units (robot arms that can carry different tools), electrical motors, robot controller (computer hardware and software) and clients (used for on-line and off-line programming of the robot controller). Industrial robots can be characterized as generic tools that can be configured and programmed for a specific purpose such as painting, welding, palletizing etc.

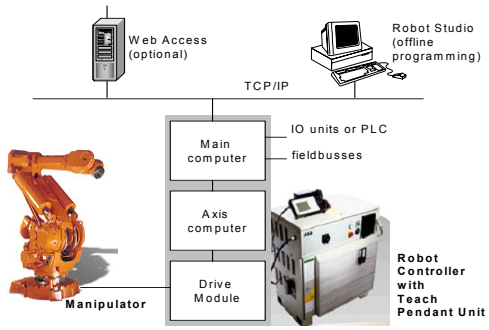

Figure 1 System hardware view of an industrial robot system from ABB Robotics

A hardware view of a robotics system is shown in Figure 1. The clients of the robot controller are optional, while the main computer and axis computer are tightly coupled, with real-time constraints on their communication.

The system has evolved through four generations, and the fifth generation of the system is currently being developed. Compared with the first generation (S1), which used the first microcomputer-based electrical robot control system, and whose software design required about three man months of work, the effort required to develop the software for the fifth generation (S5) is estimated to be about 100 man years.

One of the initial requirements was that the same controller should be used for all the different types of robots, and thus the architecture can be characterized as product line architecture.

In essence, the controller has layered architecture and within layers, an object-oriented design. The implementation consists of approximately 2500 KLOC of C language source code divided into 400-500 "classes" and organized in 8 technical domains. The software platform of the robot controller defines the infrastructure that provides basic services such as: a broker for message-based inter-task communication, configuration support, persistent storage handling, system startup and shutdown, etc. These basic services constraint the implementation of the software system, as defined by the architecture.

3.2 Ericsson AB (R&D System Management)

The interviewee is a member of the R&D System Management group, the task of which is to maintain an overall technical view of how the Ericsson product portfolio and platform technology is evolving. The subject of our discussion was the Ericsson telecom system as a system of systems. Telecom systems have a long history and have been standardized to permit a global communication system. Standards are the dominating factor for system functionality in this domain - a sign of the maturity of the telecom industry. Packet switching networks are more recent and these systems are still subject to more dynamics and changes in their requirements. A telecom system is a complex system and an example of only a part of a telecom network is shown in Figure 2. The figure shows a radio access network, the radio base station acting as the radio modem, converting digital information to analog radio signals and vice versa. This particular part of a telecom network was illustrated because the software architecture of one of its nodes is discussed in the following section.

A telecom system requires exceptional reliability and availability. For example, a switch is started only once, and after that it should be possible to perform almost all maintenance during operations. Interoperability is another key property of a telecom system.

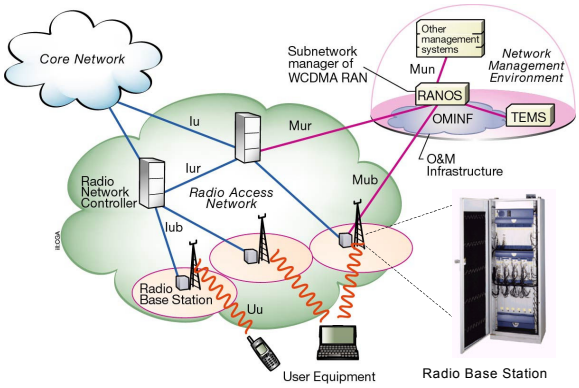

Figure 2 Radio Access Network

3.3 TietoEnator Telecom & Media

TietoEnator Telecom & Media, in partnership with Ericsson, develops 3G-base stations for the new mobile telecommunication system UMTS. We interviewed a senior software specialist who acts as one of the software architects of the Radio Base Station (see Figure 2). One of the most significant characteristics of 3G base stations is that they are sold in very large numbers (for example, each 3G provider in Sweden requires about 12,000 base stations). They must have very high degrees of availability and ease of maintenance.

The system currently being developed is the first version of the new 3G-base stations. The development of the base station system was preceded by an initial prototype and experimental implementation, gathering experience and evaluating architectural solutions. In the actual product development, experience from the prototype development was very useful, but no software from the prototype was reused. The radio base station controller software consists of approximately 2000 KLOC of code organised in about 5000 UML-RT model elements, i.e. capsules, protocols and classes. About 80% of the code is generated automatically from Real-time UML. The base stations are built on a platform delivered by Ericsson (CPP, Connectivity Packet Platform). The platform is based on the OSE-Delta real-time operating system.

Figure 3 shows the functional architecture and the major functional components of a Radio Base Station. The functional components can be realized both in software and hardware, the Traffic Control and Operation & Maintenance being the most SW-centric components.

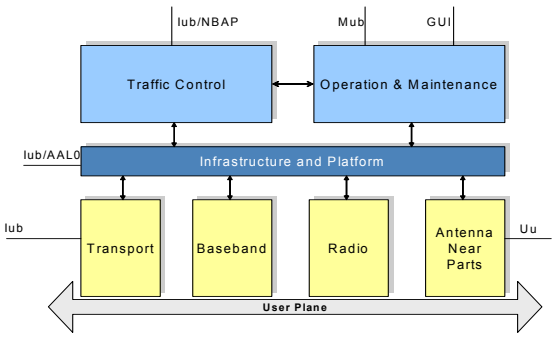

Figure 3 Radio Base Station Functional Architecture

The main characteristic of the software architecture is that the functional architecture has been arranged in a layered structure. The layering can be seen in three distinct dimensions: From the Traffic Control point of view, from the Operation & Maintenance point of view and from the Platform point of view. The main focus in the layered structure from the Traffic Control point of view is on the hardware abstraction layer, which decouples the higher layers from the actual HW realization in the Transport, Baseband, Radio and Antenna-near parts. The decoupling is achieved by means of reusable components that provide the User Plane functionality in the same way, irrespective of how the HW is being realized, i.e. irrespective of which kind of radio base station is being built.

3.4 Ericsson AB (Core Network Development)

In this interview, we met two software technology specialists at the Ericsson Core unit, Core Network Development. This group develops platforms which provide the basic hardware and software in different types of telecommunication systems. Examples of platforms are CPP, AXE switch platform and WPP platform (for GSN nodes). Platforms include both hardware and software. Examples of systems built on these platforms are nodes in telecom networks: Digital switching systems (e.g. AXE108), Mobile Base Stations (e.g. RBS discussed in section 3.3 above), SGSN (GPRS Support Node).

Many architecture patterns can be recognized in each of those platforms, e.g. “client server", "blackboard" and especially “pipes and filters”. The system has a layered structure both on the system and subsystem levels. Special attention was devoted in the architectural design of these systems to concurrency and availability.

3.5 Volvo Construction Equipment

Volvo Construction Equipment (Volvo CE) develops and manufactures a wide variety of construction equipment vehicles such as articulated haulers, excavators, graders, backhoe loaders, and wheel loaders. We interviewed a technical specialist in electronic systems, who have been involved in the architectural design of several generations of the Volvo CE system.

Compared with passenger cars, most construction equipment vehicles are equipped with less complex electronic systems and networks. Using a distributed electronic system reduces the cost of the product by permitting the use of sensors and displays for several purposes and enabling the use of control solutions which permit the use of les expensive mechanical components.

Figure 4 shows the basic architecture of the electronic system consisting of several ECUs (Electronic Control Units) connected by busses. A unified hardware is currently used for all nodes except for the display ECUs that differ due to space and appearance requirements. A unified hardware means, in this case, a common design with configurable I/O.

Together with the common hardware platform, Volvo CE uses a common software platform for the on-board ECUs. All of the nodes have a layered structure as shown in Figure 4. Software implementation that is reused between nodes includes: boot code, drivers, communication software, service software, error handling etc. Tools such as compiler, code generators, and scheduler can be used more easily due to the fixed hardware platform. Methods for parameterization of software are also reused.

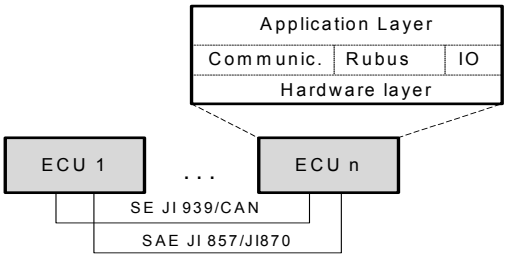

Figure 4 Components in electronic system of Volvo CE equipment.

To permit the reuse of software components and methodology in different products, Volvo CE has incorporated the Rubus component model for the real time application domain [12]. The component model is an important part of the Volvo CE electronic platform since it enables reuse and commonality in terms of tools and methods. The Rubus component model is port-based and provides an architecture that corresponds to the pipes and filters pattern.

3.6 Volvo Car Corporation

Volvo Car Corporation (Volvo CC) is a subsidiary of the Ford Motor Company, manufacturing a premium product aimed at the upper end of the car market. In this interview we met with the Program Manager, Research & Advanced Engineering. Volvo CC manufactures nearly half a million cars per year. To achieve these volumes, and still offer the customer a wide range of choices, the products are built on platforms containing common technology that has the flexibility to be adaptable to different models. A typical configuration of a Volvo CC car includes ECUs from more than 10 suppliers. A Volvo CC car contains a maximum of about 40 ECUs , connected via 4 different networks.

External suppliers work with a number of different car companies (or OEMs, original equipment manufacturers), providing them with similar parts. The role of the OEM is to provide external suppliers with specifications so that the component supplied will be suitable for a particular car model. Currently, external suppliers offer components in the form of ECUs with associated software, but as the computational power of the ECUs increase, it will be more common to include software from several suppliers in the same nodes, this increasing the complexity of the integration. The suppliers develop the ECU software using their tools and structure, but Volvo CC, as an OEM specifies: communication, power consumption, diagnostics, and software download procedure.

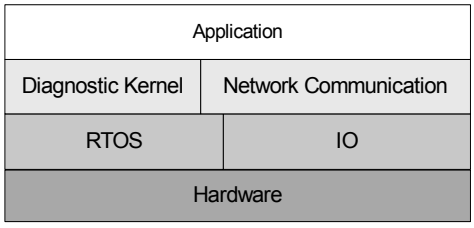

Figure 5 ECU node architecture.

The Volvo CC node architecture is a layered architecture (Figure 5) that must include a diagnostic kernel and a network interface provided by Volvo CC. All these components are integrated by Volvo CC, which is responsible for guaranteeing each node's resource requirements with respect to communication bandwidth.

3.7 Bombardier Transportation

Bombardier Transportation is the global leader in the rail equipment, manufacturing and servicing industry. We interviewed persons with two different roles – a system architect and a technology specialist. Examples of Bombardier products are: passenger rail vehicles and total transit systems, locomotives, freight cars, propulsion & controls, signaling equipment and systems. The group, representatives of which we met, develops components of the control system, propulsion and control systems. The control system components are delivered to the system groups that develop complete solutions for the end customers, and in particular, sub-domains (e.g. InterCity trains, Metro, Railway Control System).

A simplified model of a train with the most important elements of its control system is shown in Figure 6. A standard designated TCN (Train Communication Network), defines the network interconnections between vehicles (WTB – Wire Train Bus) and within vehicles (MVB - Multifunction Vehicle Bus). A common time-triggered protocol is used on both WTB and MVB bus.

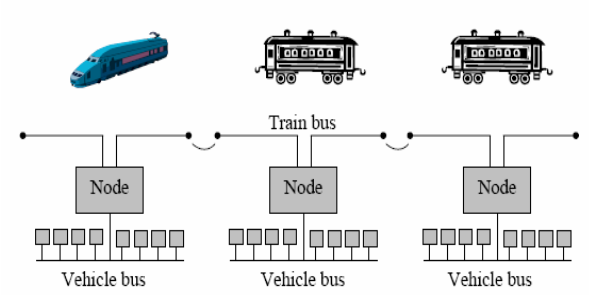

Figure 6 Train communication network

The layers pattern is dominant on the architectural level. Other design patterns such as publisher-subscriber and cyclic execution are also used in addition to own internal patterns to develop applications in a distributed deterministic real-time environment. For application programming, within nodes of the control system, Bombardier uses function blocks as defined in the IEC 61131-3 standard programming languages [11].

In addition to the traditional control system-related functionality, comfort-related functionality is becoming more important. Each of the computer-based electronic equipment units developed by the organization can be classified in one of the three domains of criticality: comfort (e.g. connection system, passenger information, entertainment, etc), safety (control system that is a part of the safety control) and control.

4. Comparative interview analysis

In this section, we present a comparative analysis of the interview data grouped according to the questions used in the interviews. In addition to a request to the interviewees to provide an overview of their systems, the following groups of questions have been used in the interviews (for detailed lists of questions see [7]):

- Relationship of software and system architecture and propagation of requirements between them;
- Reuse/inheritance/legacy issues;
- Business/application domain factors:
- Choice of technologies:
- Organizational issues and architecture;
- Process issues;
- Resources in architectural activities

4.1 Relationship of system, computer hardware and software architecture

From the brief overviews of system and software architectures presented in Section 3, we can conclude that they have many similarities. All of the systems have complex distributed system architectures with distributed and relatively autonomous units. In some cases we can treat them as systems of systems.

In [6] Maier and Rechtin discuss a shift in architectural design methods from “hardware first” to “software first”. One of the goals of our investigation was to determine the current state of practice regarding the relationship of software and system architectures in industrial systems. All interviewees agreed that the system architecture comes first, where the system consists of other hardware, computer hardware and software. That the “software before computer hardware” approach is becoming increasingly common is demonstrated by several of the systems we have studied; recent system generation has changed from “hardware first” to “software first” at both ABB Robotics and Volvo CE. Increasingly more intelligence is encapsulated in the software of these systems. In the ABB Robotics example, the computer hardware was completely redesigned in the most recent system generation shift, the software platform remaining the same. In the Ericsson cases, hardware still dominates the design of the software system. In the Volvo CC case, the hardware architecture remains dominant as it is the basis for integration of external functionality but the importance of software and software architecture is increasing.

The following are views regarding software vs. system knowledge expressed by interviewees. One

stated that a "good enough knowledge, or at least understanding of the system and the HW... with respect to managing, supervising and controlling all the different HW is needed. However, I find that my experience in SW engineering and good general knowledge of SW is much more valuable in getting a good SW architecture." Another commented "I have seen 'strange' software solutions built by application/domain specialists who did not have sufficiently good general software architecture design knowledge." This indicates that system expertise is necessary, but not sufficient.

4.2 Reuse and legacy in architectural design

As described in Section 2, several of the systems described have passed through clearly defined generations and within the systems, there have been corresponding generations of the (control) system hardware and software. We analyzed the relative importance of the following three factors: experience, subsystems and code, in the design of a major new generation of a system.

All of the interviewees stated that *experience* in developing similar systems, or previous generations of the system, was of the greatest importance. This is because the design of a new system generation seldom, if ever, begins with a blank sheet.

The reuse of subsystems in a new system design was considered to be an important economy and therefore can have more impact on the architectural design than might be expected. There are examples (ABB, Ericsson), in which complete subsystems were either reused from a previous system generation or acquired from a third party, when a new generation was designed. In the case of ABB, the design of the S4 system generation would have never been approved, because of the unacceptably high cost, if the new architecture required all subsystems to be changed or replaced. It is hard to disregard the legacy in long-lived systems. In the case of Bombardier, we have seen an example in which new hardware was designed with the explicit requirement that it should run old system software without change. In safety critical systems, e.g. train safety control, it is highly desirable to reuse a critical code that has been proven to work well in practice.

As the amount of software in a system increases it becomes increasingly important to reuse software components to minimize the investment in a new system or generation. It is therefore important to package software components in such a way that they can be used in a variety of architectures. A stable base platform infrastructure and easily reconfigurable connections between components on higher-levels is economically advantageous as seen in the ABB Robotics example. The software architecture in the system is considered to be the same in the most recent system generations, because although some new, very different features have been added, the basic patterns, message-based communication and similar have not changed. From a structural point of view, as connectors and components, this architecture would be considered to be a new architecture because a number of components and the way they are connected have changed. Of course different system properties, such as timing properties or reliability can be changed, due to changes in the structure, and must therefore be re-verified.

4.3 Business and application domain factors

In this section, we analyze the impact of business and domain related factors on system and software architecture. More specifically, we have investigated the influence of the following factors: standards, type of customers, production volumes, product lifetime, and non-functional requirements.

The influence of *Standards* varies on the basis of the domains in which the systems are used. Standards completely dominate the telecom domain, leaving space for competition based on optional features in standards and other non-functional requirements. Standards are not only used for interoperation between different systems, but also for interoperation between the nodes within a system. This is the case not only in telecommunication but also in the automotive industry in which e.g. the standardization of bus protocols is used as the integration point. The importance of standards and interoperability go hand-in-hand in the telecom domain. It is very important, to remain competitive in the market, to participate in standardization activities, to be at the leading edge and deliver solutions as soon as standards are finalized. For other systems and domains, we were told that standards have the greatest impact on subsystem level (e.g. a safety subsystem). This says basically that system partitioning was applied in the solution space, in order to comply with standards more easily (i.e. to certify a subsystem rather than the whole system). It is common to use software to implement more advanced safety features and have software independent core safety function as a backup (e.g. robotics system safety, ABS systems for car brakes, trains safety). Industry standards have a considerable impact on system architecture, e.g. for robotic systems - support for different kinds of field busses, for train control systems - TCN standard for train networks, other

industry standards for network communication. These systems often have architectural level variability points to be able to support, for example, multiple industry standard protocols.

The *type of customers* has an appreciable impact on the process of architectural design. In some cases (Volvo CC) there are many small customers. In this case, the development partners are those involved in the architectural design. In other cases (ABB Robotics, Ericsson) there are both large customers and many small. We were told that: "large customers have their opinions not only about what a system should do but also how a system should be built". In the case of special customers, architectural issues may be a matter for discussion between the customer and the architects.

Product volumes also have an important effect on the architectural design. If a product is to be manufactured in large numbers, it is easier to justify the expenditure of more time in optimizing the product with respect to certain properties if this will lower the unit cost. The properties which need to be further optimized depend on the product. Ease of maintenance and fault tracing by non-experts is definitely one important property (Volvo CC, Volvo CE).

Product lifetime is another factor that is important in the architectural design of the systems we have studied. Most of the systems studied use a layered approach to decouple hardware and software, but also to decouple platform software and operating system. When the product lifetime is 20-30 years, any unavailability of the OS used to build the original system is as much a problem as the unavailability of the hardware used in the original system design. We were shown examples of projects in which it was decided to use the Linux OS platform instead of MS Windows, because it was believed to be preferable to own the source code. We are not aware of any systematic approach to dealing with this issue.

We have seen that *non-functional requirements (NFR)* have a significant explicit impact on the architectural design of systems investigated. Each of the systems has particular non-functional requirements that are dominating and explicitly taken into account. Operational NFR are always analyzed from the system level. For some systems (e.g. trains) there is more focus on hardware NFR because of the nature of the operational environment (e.g. temperature variations – 40C to +50C, humidity, vibrations, etc), while for other systems (e.g. telephone switches) software and hardware play an equally important role in providing support for performance/concurrency. The implementation of software concurrency mechanisms and fault-tolerance (especially fault isolation) is given much attention during the architectural design in telecom networks. Understandability of the system architecture was stressed by all of the interviewees as being very important.

4.4 Choice of technologies

A number of questions were asked during the interviews to determine how technologies and architecture have influenced each other in the creation process and the evolution of systems. In particular, we wanted to know if any explicit architectural activities or measures taken were related to choice of technology. We used the term technology in a broad sense - something that includes particular principles, methods and tools - e.g. database-technology, .NET or Java technology or similar.

From the architectural point of view, a common opinion was that technology does not play a crucial role. An ambition is to keep architecture separated from implementation, and in many cases, the use of a particular technology is seen as a matter of particular implementation. New technology is introduced carefully, very often in a part of a system (ABB Robotics, Ericsson). The introduction of new technology is sometimes forced by cost reduction requirements, e.g. the introduction of new cheaper hardware (Bombardier), or because of a possibility of utilizing more efficient tools (Bombardier, ABB Robotics).

However we have seen that the choice of technology may have important impacts on the architectural documentation, design and evaluation. If a particular technology brings some important advantages in analysis and settings of quality attributes (such as timing properties), its use becomes the central paradigm of the development process. One characteristic example is the use of a component-based technology used by Volvo CE in which the software and, to a degree, the system architecture are expressed consequently in terms of components with a standardized specification. Another example is the Model Based approach which is used at TietoEnator in the design of RBS software (section 3.3). UML-RT specifications are strictly used and the code is generated from the specification. In these cases technology plays an important role, achieving not only greater efficiency but also a better understanding of the system architecture.

A somewhat surprising finding related to technology was that technology choices are sometimes even made to motivate the developers of the system and create enthusiasm in the team.

4.5 Organizational factors and architecture

We analyzed the following issues in this section: influence of distributed development on architecture, outsourcing, size, maturity, dynamics, etc., of the organization which was to implement the system.

It is widely accepted that the organization influences the architecture; for example, Conway's Law says that the structure of the organization that builds some software matches the structure of the software [3]. The allocation of resources and people working on the project will have a direct impact on the architecture of the system. The majority of the interviewees stressed that the company's organization often mirrors the system architecture and vice versa. Proper handling of this relationship is important in order to minimize the dependencies in the software that make integration and validation more difficult and also to achieve distinct interfaces between different organizational units.

Several interviewees emphasized the importance of taking into account in architectural design, the fact that implementation will take place in geographically different places. We have also seen examples of distributed development not being taken into account, this resulting in less than optimal architectural support for the distributed development process. However, partitioning that is appropriate in a distributed organization may impair other system properties.

Several interviewees mentioned that changes in the organization are more frequent than changes in the architecture. Some of the reasons for organizational changes are company mergers and changes in the market. Moreover, some employees leave and others join the development organization and it is important that the newly employed people can become productive quickly. That is why NFR mentioned in 4.3 - *understandability* is of the utmost importance.

Some of the interviewees had fundamentally different opinions concerning understandability. An Ericsson architect opinion is that it is impossible to achieve quality if the developers do not "have the big-picture", i.e. understand how a sub-system is used and operates with other units. On the contrary, at ABB Robotics, opinion is that it is more important that many developers can be good development contributors, without knowing and understanding the entire system. This is because the ABB Robotics system is relatively multi-disciplinary; control engineers, mechanical engineers, and software engineers must be able to contribute to the same system without necessarily having any deeper knowledge in the other's fields. Other possible factors that have an impact on this issue are, for instance, related to the turnover of engineers, and the magnitude of the system.

One organizational and process issue related to system evolution which is particularly interesting is how to balance long-term strategic goals and short-term project goals. The conflict begins as early as in the architectural phase (experiences from Ericsson) – should more time be spent in establishing a good solid base for a long-term product evolution, or in getting a single product to the market as soon as possible?

4.6 Process related factors

There was surprisingly little interest among the architects (with a few exceptions) in life cycle and development processes. While it is obvious that architecture is a main means of preserving system properties in an evolution process, it seems that the process itself is not a direct concern of the architects. This implies that the mutual relations and influences are not under direct control but happen more or less ad-hoc. Examples in which a process view of the architecture would be beneficial are the designing of a testable architecture or, for managing future changes in, e.g. technologies, by having a life-cycle process associated with the architecture. We believe that the absence of a process view in these architectures may be one of their weaknesses and also that the indirect impact of the processes is appreciable. An example in which a more defined process could be advantageous was given by the architect at TíetoEnator. Integrating software and hardware for the very first time is usually a source of friction. One potential remedy of this problem could be software/hardware co-design.

Standards, e.g. safety standard such as IEC 61508 and ISO 15998, specify the development and maintenance processes. These standards will affect Volvo CE and have already been considered at Bombardier Transportation in different domains of criticality (comfort, control, and safety). Processes for development in the safety domain are much more strict and regulated, than e.g. those for the comfort domain.

4.7 Resources used for architectural design

Factors that were discussed included the time and effort invested in architectural activities and the number of people involved in architectural design.

The companies that participated in this investigation spend several years in developing a new architecture. Typically, architectural evaluation is based on prototypes and pilot systems.

All the architects who participated in this investigation agreed upon the importance of small core-teams that decide the architecture, i.e. *single-mind-consistency*. Typically, architects should be people with knowledge of the domain, the technology, and the architecture. One interviewee described an architect as "a person who could implement the complete system by himself/herself, if only time (to market) permitted."

The fundamental principles of the system i.e. its basic infrastructure are a result of the architectural design. Examples of fundamental principles in this context are the infrastructures for communication and concurrency. The fundamental principles should be developed and stable before too many developers are involved. We were given a contra-example of an unsuccessful project (Ericsson), one of the main reasons identified being that too many people were involved before the basic principles were stabilized.

5. Conclusion

In this paper we have presented the results of a case study of several complex industrial systems. We will conclude the paper by providing a synthesis of our interviews with respect to the life cycle of a system. For each phase of the life cycle we will present our main findings.

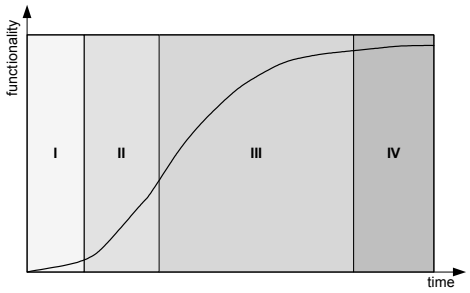

Figure 7 A system life cycle plotted as functionality over time

A system's life cycle can be divided roughly into four different phases as depicted in Figure 7: (I) inception, (II) initial development, (III) maintenance and evolution, and (IV) end of life time. The curve in Figure 7 plots the functionality in the system over time. Hence, for a successful system it is desirable to stay in the phase III as long as possible with a curve that has an inclination as steep as possible, because this implies a high degree of productivity. The architecture is the means that should ensure a long life cycle.

It is the initial phase that lays the foundation for a long life cycle. A core team of experienced designers should be responsible for deciding the important architectural principles and the infrastructure. The architectural principles typically manifest themselves as guidelines, handbooks and an infrastructure as a platform. It is the architectural principles that ensure that the most important non-functional requirements are fulfilled. The most common mistake reported is to involve too many people in the initial phase. Several of the interviewees reported experiences in which too many people were involved without having a clear understanding of how to implement what. At best, the development is hesitant and in the worst case, many creative engineers design diverging system components which are incompatible because there are too few constraints on their work. This problem often occurs because higher management requires too much result in too short a time and believes that increasing the number of engineers employed in the project will solve the problem.

It is important that the architectural principles are properly communicated through the development organization in order to preserve them. However, no or limited strategies for communicating the important architectural principles were found in the case studies. One strategy that was reported as successful was to appoint members of the core architecture team as technical leaders in the development projects. In this way technical leaders become the medium that carries and transfers information of importance to the development organization.

The architects or technical leaders also have a very important role in bridging the gap between architecture and technology, acting as mentors or guides [4]. It is crucial that the architect is a technically very competent person, able to handle both detailed technical issues in the implementation view as well as the coarse-grained big picture represented by the architecture and the domain. Only this kind of architect will be trustworthy in the eyes of the developers who implement the system. Architects should also be able to rise above the small-detail problems and find a better solution on a higher level, when appropriate.

Communicating the important architectural principles is a continuous process that must be considered throughout the life cycle of the system. It is important to define processes and strategies, not only for communicating architectural principles, but also for managing for example, the satisfaction of new requirements. A possible reason for finding process issues rather insignificant could be that engineers who develop systems and those responsible for process related issues tend to have a different focus – product and process quality.

The majority of the systems we have studied are continuously exposed to new requirements from the

customers. If these are not properly handled, the system evolution may result in architectural deterioration. As a system is maintained and new functions are added the technical complexity will increase. This is especially true if the architecture does not completely adopt and support the new functions.

Moreover, the cognitive complexity also becomes a problem if there is a large turnover in personnel. Consequently, it is important to have continuity in the people working with the system since they are carriers of undocumented and important knowledge. In the majority of the cases in our study, tools or processes for handling new requirements were not used. As a consequence, the focus during system evolution is on the new requirements only. This creates a risk that old requirements are violated while new requirements are being implemented. Ideally, old requirements should always be verified when new requirements are being implemented. We found no systematic approach to solving this problem.

Finally, the system reaches a point (phase IV in Figure 7) where the current architecture cannot support the new requirements or they become too difficult i.e. too expensive, to implement within the frame provided by the architecture. As the inclination of the functionality curve in Figure 7 decreases, the effort of adding new functions becomes excessive. Paradoxically, this is likely the time at which the company makes the most profit from the system, and will probably continue doing so for quite a while. It is important to set aside funds from that profit for investment in a new architecture. Such an activity should begin towards the end of the phase III.

We believe that the results presented in this paper are applicable in general to complex industrial systems in which software has an important and growing role.

In this paper, we have identified several areas that deserve more attention and each can be the profitable subject of a new study. Additionally, in our future work, we plan to perform additional interviews with smaller organizations and study their systems and compare the results with our current observations and conclusions. We have performed one such interview, which is not discussed in this paper. Before we can publish any results we need to perform additional investigation.

6. Acknowledgments

We are very grateful to the following people who participated in this case study and their companies: Peter Ericsson (ABB Automation Technologies AB/Robotics); Ulf Olsson, Hans Brolin and Mike Williams (Ericsson AB); Nils-Erik Bånkestad (Volvo Construction Equipment), Jakob Axelsson (Volvo Car Corporation); Peter Cigéhn (TietoEnator Telecom & Media); Erik Gyllensvärd and Peter Sandberg (Bombardier Transportation).

7. References

[1] Bass L. et al, *Software Architecture in Practice*, Addison-Wesley, 2003.

[2] Bril R.J., et al, *Embedding Architectural Support in Industry*, International Conference on Software Maintenance, 2003.

[3] Conway M., *How do committees Invent?*, Datamation, 14 (4), 1968.

[4] Fowler M., *Who needs an architect?*, *IEEE Software*, 2003.

[5] Graaf B. et al, *Embedded Software Engineering: The State of the Practice*, IEEE Software, 2003.

[6] Maier M. and Rechtin E., *The art of systems architecting*, CRC Press, 2000.

[7] Mustapic G. et al, *Influences between Software Architecture and its Environment in Industrial Systems - a Case Study*, MRTC Technical Report, http://www.idt.mdh.se, 2004

[8] Pfleeger S.L., *Software Engineering - Theory and Practice*, Prentice Hall, 2000.

[9] Smolander K. et al, *What is Included in Software Architecture? – A Case Study in Three Software Organizations*, proceedings of the 9th IEEE International Conference and Workshop on Engineering of Computer-Based Systems, 2002.

[10] Soni D. et al, *An Empirical Approach to Software Architectures*, proceedings of the 7th International Workshop on Software Specification and Design, 1993.

[11] Application and Implementation of IEC 1131-3, Standard provided by the International Electrotechnical Commission, May 1995.

[12] The Rubus Operating System Manual, www.arcticus.se