Assessing the Influence on Processes when Evolving the Software Architecture

Stig Larsson
ABB Corporate Research
Västerås
Sweden
stig.bm.larsson@se.abb.com

Anders Wall
ABB Corporate Research
Västerås
Sweden
anders.wall@se.abb.com

Peter Wallin
Mälardalen University
Västerås
Sweden
peter.wallin@mdh.se

ABSTRACT
Software intensive products and systems evolve over the life-cycle. Changing business objectives may drive architectural or process changes. Altering either architecture or process might influence the other. Also the organization may influence and be influenced. This paper describes these relationships and proposes a method for assessing the influence on process that a proposed architectural change can have. The method includes the use of scenarios and process reference models. A case study where the method has been used is described, identifying the need for changes in the processes to be able to utilize the advantages made possible due to the architectural evolution. The case study supports our proposal that a structured method to assess the impacts on process when changing the architecture of a system helps to reduce risks and to facilitate the envisioned business benefits. This also identifies the need to devise methods for other types of changes, e.g. how a process change may influence architecture or organization.

Categories and Subject Descriptors
D.2 SOFTWARE ENGINEERING: D.2.7 Distribution, Maintenance, and Enhancement, D.2.9 Management, and D.2.11 Software Architectures.

General Terms
Management, Design, Economic.

Keywords
Architecture, Process, Organization, Business Objectives.

1. INTRODUCTION
As the architecture of a system or product changes, the processes used for the development may change, and vice versa. One example on this is when a system is modularized and new ways of ensuring the integrity of interfaces are needed. Another example is when new business requirements based on possibilities for distributed development require the organization to structure the software into a platform and applications but also to define new processes of how these parts of systems are to be integrated before sent to an end customer.

Moreover, we have observed that changes in business drivers, organization, and technology are common during the life-cycle of a long-lived industrial system. Examples of changes are commercial components that get obsolete and need to be replaced, distribution of development is initiated, companies merge, organizations targets new markets, or gets changed customer focus. It is consequently necessary to have a continuous evolution in all three dimensions: architecture, processes, and organization.

Still, there is a lack of a thorough analysis of interdependences of these factors. While there are many methods for analysis of software evolution based on software architecture, or methods for process improvements, it is practically unknown how they are dependent of each other. We see that there is a clear need for building knowledge of interdependencies between evolution of architectures, development processes, and changes in the development organizations. Our experience is that this is in particular important for long-life products. Examples of such systems are industrial products and systems.

Development of industrial control products and systems is often performed as an evolution rather than frequently developing new products from scratch. The reason is that these products are complex, requirements from customer forces focus on time-to-market, and that a substantial investment is needed before the functionality of a new product matches or exceeds earlier generations of the product [4]. The focus on evolving systems combined with the complexity in today's industrial systems requires that the integrity of the architecture of the system is kept intact. If system architecture integrity degrades, or enters the *servicing* stage as described by Bennet and Rajlich in [1], it is no longer possible to add substantial functionality to the system. To protect the investments in the development of the product, this should be avoided as long as possible.

In this paper different relationships between changes in architecture and the effects on product development processes as well as changes in process and the effects on architecture are discussed. A method for assessing one type of change is proposed, and is illustrated on an industrial case.

IWPSE'07, September 3-4, 2007, Dubrovnik, Croatia
Copyright 2007 ACM ISBN 978-1-59593-722-3/07/09...$5.00.

The remainder of this paper is organized as follows. Section 2 describes the relationships between changes in architecture, organization, and process as well as the proposed investigation method. The case where the use of the proposed method has been illustrated is described in section 3. Section 4 describes related work while conclusions and further work are found in Section 5.

2. METHOD DESCRIPTION

This section describes the relationships between architecture, organization, or development processes when changes occur due to changed business objectives. In addition, a method is proposed for assessing the requirements on changes in processes when an architectural change is initiated.

2.1 Types of Relationships

The reasons for changing the development processes or the architecture should always be motivated from a business perspective. Our experience is that a change in the architecture should never be driven by technology without a specific business motivation. Examples of such motivations are changes in customer focus or introducing distributed development. Also seemingly architecturally driven changes should only be done based on business needs, e.g. a complex architecture that needs refactoring should be changed only if a business benefit can be identified. For example, reduced cost for maintenance or easier evolution of the system may be the original business reason. The business-drivers for our case-study are described in Section 3.1.

Since processes, organization, and architecture all must be synchronized in order to support a cost-effective product development, a change along one of these dimensions will require a review of the others in the light of the proposed change. Figure 1 depicts the relationship between changes in business objectives (ΔB), process changes (ΔP), organization changes (ΔO), and changes in the architecture (ΔA).

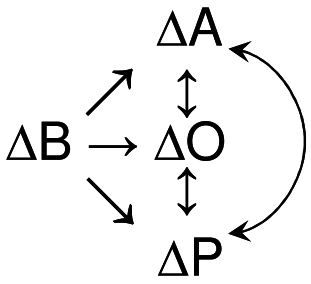

Figure 1. Relationship between different types of changes

The change based on business objectives can be initiated from any of the three dimensions, e.g. a development group proposes a change in the processes to reach the business objectives which may have influences on the software architecture or the other way around. Changes in the organization, e.g. a decision to distribute development to get presence in a specific geographic market, may influence both architecture and development processes.

The method proposed in this paper should not only provide guidance concerning specific changes in existing architecture, organization, and processes but should also give an indication on the cost and risks of the proposed changes. Typically, the motivation for the kinds of changes discussed in this paper is related to reducing product development- and maintenance costs.

2.2 Business-Architecture-Process Method

To investigate and analyze the influences that a change in architecture will have on the development processes we propose the Business-Architecture-Process method. It covers the influence from business objectives and architectural change on processes which is highlighted in Figure 2. It consists of five steps: *Initiate and Motivate the Organization, Find Requirements on Affected Processes, Analyze Different Solutions, Define Alternative Strategies,* and *Decide on Strategy.* An important part of the method is that the underlying business objectives are made visible and should be clearly understood by the organization. Central in the method is also the use of scenarios, i.e. synopsis describing an event or situation. Through the scenarios, an understanding of the business objectives is obtained as the implications of the objectives are made concrete. Finally, the use of reference models is important as this reduces the risk to omit significant process steps.

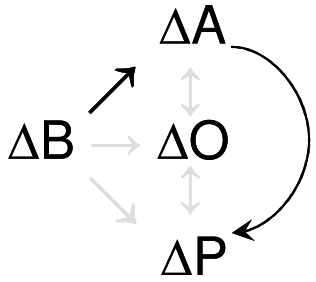

Figure 2. Architectural changes affecting processes

2.2.1 Step 1: Initiate and motivate the organization

Before the investigation can begin, a common motivation must exist for the organization. This is similar to the initiation phase as described in the IDEAL model [9] from Software Engineering Institute used for process improvement. Based on business drivers and a vision for what should be accomplished with the architectural change, the sponsors and other roles for the process investigation should be identified. The sponsors need to communicate the vision, and identify the possible receivers of any process changes. The final activity is to train these receivers in the architectural influence on processes. The outcome of this first step is an organization that is informed and prepared for the process investigation.

2.2.2 Step 2: Find requirements on affected processes

Based on the business drivers as well as the targets and vision for the architectural change, an understanding of what processes are affected should be created. This is done based on scenarios that describe the goals of the architectural change in a concrete way, the currently used practices, and one or more reference models. The results of this step are new requirements on the product development processes used.

The first activity in this step is to create a set of scenarios that describe the vision and purpose of the architectural change in more detail. The reason to work with scenarios is that this makes

the vision concrete for the stakeholders, and promotes a discussion about the activities in the organization. The scenarios limit the scope of the process investigations, making it possible to focus on what is important for this specific change. The scenarios should describe the different activities performed to achieve a goal in an organization. One way to describe a scenario is found in Figure 4.

After the identification of the involved processes, an understanding of current practices should be obtained. The practices need to be captured through an appraisal as it is the used practice that is important, not the documented. Also the problems in the currently used process will be available after the appraisal, and should be part of the material used for further activities. The use of reference models help the investigators to ensure that no process is missed; if some practices are missing in the way the organization is operating, that practice may be ignored if there is no reference to check with.

When data about used processes and the scenarios are available, the next step is to reason about whether a process is affected or not. This can be done in a workshop with affected stakeholders, and will result in conclusions regarding new requirements on the used processes. One essential part of the activity is to capture the rationale for the analysis; the reasoning behind why a process should be modified or added needs to be documented.

The result from this step is a set of the requirements on processes and tools used. It is advisable to have a checkpoint after this step; if there are many new requirements, the organization should consider alternatives to the architectural change.

2.2.3 Step 3: Analyze different solutions

The understanding of the practices used in the organization is together with the scenarios used to describe different possible ways to change the affected processes. For each proposed change, the consequences are listed. These typically include changes in roles, authorities, responsibilities, competence, documentation, and communication. It is important at this stage to have a set of different alternatives described for *each process* independently of other processes, as the selected solution may differ depending on combined considerations for several processes.

2.2.4 Step 4: Define alternative strategies

The solutions from step 3 are in this step grouped together to form strategies. Here *combinations* of process changes are investigated. Each strategy should be a combination of proposed process changes that enables a particular scenario or a group of scenarios to be implemented. The reason for combining the process changes into strategies is that they may influence each other. For example, a change in the handling of product integration can affect configuration management, i.e. the way that baselines are managed. The description of a strategy should include associated risks as well as steps and related effort needed to implement the process changes.

2.2.5 Step 5: Decide on strategy

When the different process changes have been described and combined into strategies, the organization is ready for the decision on what strategy to select. The business objectives will be the basis for the decision, as will the risks for each of the strategies. To make a successful implementation likely it is important that the decision on a specific strategy is properly communicated and discussed. In these discussions, the underlying material such as the process investigations based on scenarios can be used. Documenting the decision and the rational for the selected solution is important as the environment may change and new situations appear. Having the background available reduces the effort to adapt to the new situation.

3. CASE STUDY

We have used the proposed method to investigate a product development organization, how the refactoring of an industrial control system is planned and implemented, and how this influences the processes. The investigation has been performed as a participant-observer study, i.e. the research was performed through participation in the refactoring project.

3.1 Case Description

The case that has been studied is the refactoring of an industrial control system at an ABB development unit. The system has evolved through several generations over a ten year period, and new functions are continuously added. Currently, the control system consists of more than three million lines of C/C++ code and several different applications are built on the same basic monolithic system. The refactoring is initiated in order to increase the possibilities to independently develop basic functions and applications, to ensure high quality software, and to increased efficiency in the software development. The most important business drivers in this case are: shortened time-to-market for new applications and new releases of existing applications, and decreased cost for maintenance.

The basic idea of the restructuring is to divide the monolithic software architecture into three parts; a kernel, a set of common extensions, and application specific extensions (Figure 3). The kernel and the common extensions are to be managed by one development group, while the applications is intended to be developed at several different locations. The kernel includes components that provide basic services, e.g. operating system abstractions, which must be a part of the all products, while the common extensions should be selected when defining an application specific product, e.g. support for a specific field bus. The Base Software is the combination of the kernel and the common extensions.

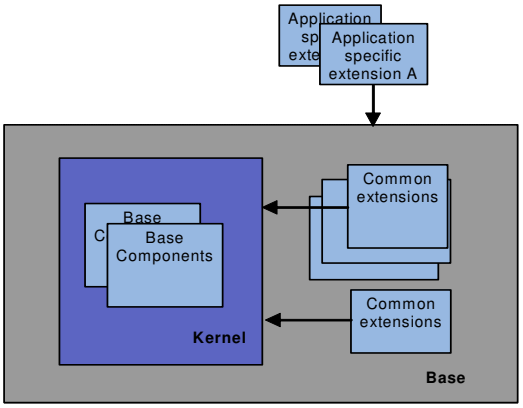

Figure 3. Block diagram of the refactored software

Software components in this context are modules built out of several classes and can have both internal and public interfaces. The idea is that a Base Software SDK (Software Development Kit) should be developed with the public interfaces provided by the Base Software (the API, application programming interface). The SDK should include a well-documented API (a programmers guide), a user guide describing how to develop applications based on the SDK, wizards for developing extensions, and tools for building products based on the SDK and application specific components. These tools should also include e.g. verification tools. The final result from the application development is the load file for the control system, which is added at production time. Additional adaptation for a specific plant can be made, but is not considered a part of the application product.

3.2 Applying the Proposed Method

This section describes how we applied the Business-Architecture-Process method to the industrial case.

3.2.1 Initiate and Motivate the Organization

The first step is to *Initiate and Motivate the Organization* both for the architectural change, and the need to investigate the influence on process. The organization had two clear business objectives: to reduce cost for verification, and to increase capability to perform distributed development. Through the research and development project, the vision and goals for refactoring were communicated to the stakeholders. One problem in this case was that the sponsor assigned the project manager to communicate the vision, both internally in the project and externally to the rest of the organization, giving perceived less importance to the message. However, through this approach, also the architectural influence on the product development processes where covered, and the receivers of the process changes were involved. The communication was also continued throughout the project to ensure that new information and status was given to the receivers.

3.2.2 Find requirements on affected processes

To investigate the influence on the product development processes, the second step, *Find requirements on affected processes,* was performed. The first activity was to develop a set of scenarios to be used together with two reference models, CMMI [14] and ISO/IEC 15288:2002 [7] . The scenarios describe different roles and activities and serves as a source of requirements for the processes. Through the use of process models, the processes can be structured and investigated with a specific process area in focus. The second activity has been to look at the current process to understand how the system is developed today. Throughout the appraisal, it has been important to understand the different needs from different stakeholders such as product managers, application developers, and base system developers. Each process area has been discussed and analyzed using the specific practices as described in CMMI and the different requirements described in "Systems engineering - Systems life cycle processes" (ISO/IEC 15288:2002). Based on the information from the two first activities, the requirements for the process have been defined and described.

In our investigation, four different scenarios have been defined, with different levels of independence for the application development units. In this context, application development is the process of combining application specific extensions, and the Base Software. This process may be performed by an organization separated from the one developing the kernel and common extensions.

Each scenario involves different roles that may be involved in the product development process when developing an application. These include an application development team, a Base Software integration team, a verification team, and a production team.

The example scenario in Figure 4 describes one alternative for how the integration of a new or modified application is done. In this example, the Base Software Integration Team is responsible for the integration of the application specific extensions. The application tests are performed by the application development team and further tests of the total system is performed by the verification team. Note that this example is a simplification of the real case which includes additional processes such as product management, release handling, and production.

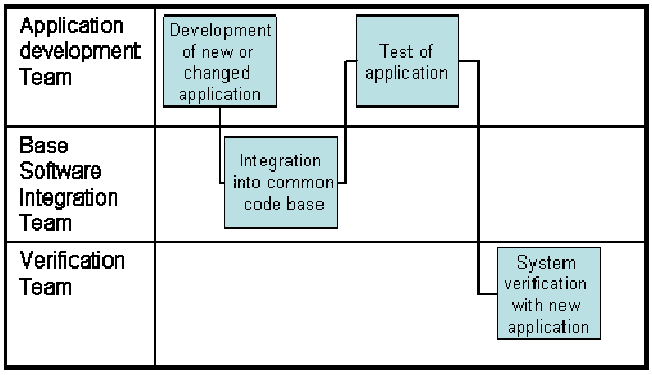

Figure 4. Example scenario. (Boxes denotes activities, and lines are showing flow of information and data)

The specific process areas that were identified as subject to most requirements for change were product management including release planning, requirement development, requirement management, configuration management, product integration, and verification. In this paper we describe the requirements and proposed solutions for product integration, configuration management, and verification. The practices that are described build on good practices identified in industry and have been examined using the different scenarios for use of the Base Software and the development of the applications.

3.2.3 Analyze different solutions

The third step, *Analyze different solutions,* was performed in discussions with experts in the different processes, and the findings where validated through a review. Here, each of the affected areas in the example process areas is described, with the different solutions discussed.

Configuration management: The parts of configuration management that are affected by the refactoring and changes in how applications are developed include handling of the code-base, documentation of builds (i.e. the process of compiling and linking software or the result of this process) and the increased need for availability of stable versions for development, tests and integration purposes.

A decision on how to handle the code-base is needed as this will create different requirements on the infrastructure. One common

code-base can be used for the whole development organization, including the application development centers. If one code-base is maintained, processes need to be defined for how the applications are included, and what baselining strategy should be used. The handling procedures should also include naming rules, version handling, and library structures that are common for the whole system. A description of the rules and procedures should be included in the Base Software SDK to ensure that they are available to the application engineers.

If several repositories are used, localization for structures, names, and documentation can be introduced. This can, however, result in issues regarding availability for support, service, maintenance, and production functions that must be resolved. Hence, if local repositories are introduced, rules for accessibility, backup, release notifications and for error corrections must be defined and implemented in each part of the organization.

Rules for how builds should be documented are needed and should be standardized. The information should contain information about included components/modules, tools and hardware used. It is also important to document the versions of software and hardware used for the build activity. This is a change from current handling where this is done in one location. Today, the handling can differ between the different application developers as the builds they initiate always are made for development purpose only and are not documented as well as a production build. The product builds are today made centrally, but may be made by the application developers once the new product architecture is launched.

Stable versions of the different components/modules are needed for developers, integrators, and test engineers. Base Software development builds should be made available for development purposes to Base Software developers, but not to application developers or the verification function as changes are introduced between different builds as a part of the development process. Instead, baselines with well defined content should be made available at agreed milestones. As with the build documentation, this is due to the fact that the versions provided to the application developers may be the version that is included in the product shipped to an end customer.

Product Integration Strategy: The product integration is the inclusion of functionality into the common code base, and should not be confused with builds. The ability to build systems must be given to all developers, but with different degrees of freedom. Base Software developers should be able to build new kernel and Base Software systems, but the integration into the common code base should be performed by a kernel integration function. Application developers should be able to build systems that include a pre-built Base Software module and new functions, but the integration into the application code base should be performed by the application integration function.

Three different types of integrations are needed with the chosen architecture:

- Kernel integration,
- Base Software integration
- Application integration.

Kernel integration includes only the parts that are needed for all systems. This integration is performed by the Base Software integration function. As there may be applications built without any of the common extensions selected, there is a requirement to test the kernel as a basic version of the software.

Base Software integration is starting from the kernel, adding the common extensions, resulting in Base Software. This integration is also performed by the Base Software integration function. It should ensure that the extensions can be selected as described and that the expected interfaces are available after the integration.

The sequence for when functionality is integrated is determined for the Base Software, including both the kernel and the common extensions.

Application integration is based on the product definitions and is after verification and validation delivered for production. An application integration function is responsible for the inclusion of new functionality into the integration. This function should also be responsible for the inclusion of new versions of the Base Software for use with the specific application. Verification is needed to ensure that the new Base Software version is compatible with the application.

The whole strategy of the integration will be changed through the use of a new layered architecture. This gives also the organization the possibility to change boundaries and responsibilities.

Development of new applications and functions need to be built on stable releases of the Base Software. This implies that intermediate versions of the Base Software should not be broadly available, and that the versions made available should have been tested.

Requirements on application development units: Each development unit that will develop products based on the Base Software SDK will have to fulfill a number of criteria. This section describes the areas where criteria are needed.

As the target system is an embedded controller, the final deployment is done as one executable even if the development of applications is made separately. The first requirement is that the development unit has the competence required for the specific development that is performed on the Base Software SDK. This calls for training of all engineers in how to use the Base Software SDK as well as general purpose software tools that are used. In addition to this, domain knowledge for both the type of embedded system that is used and for the specific application is needed.

The second requirement is that a specification of the equipment needed for the application development and integration units must be developed. It should include specification for the development environment, with external and internal SDK, hardware requirements for development computers, tools for verification including automated tests and build machines. Development units that are performing the integration function also need equipment for integration tests.

Finally, a third requirement, certification, can be introduced. This should be done to ensure that quality development is performed through guaranteeing that the competence and skills needed are in place The certification should check that training has been provided to all development engineers as defined in the training requirements, that verification procedures are defined and validated and that the development equipment and development environments are available.

The certification should be performed by the unit responsible for the Base Software development and be performed for individual engineers as well as for the organizations developing applications.

Product integration delivery and criteria: To accept a solution or function for integration, the readiness of the delivered modules must be checked. This should be done using criteria for when a module can be delivered. If development of applications is distributed to many parts of the organization, a set of criteria that can be used for all levels of integration is needed. This will ensure that the documentation and quality is maintained on a common level, and the transfer of functions between different parts of the system is simplified. An example is when an application specific extension is generalized and made available as a common extension.

Examples of criteria for allowing a function to be integrated are that code reviews and module/class tests have been performed with satisfactory results, the level of expected remaining errors is documented, and design documentation is available.

Tool support is recommended to simplify the checking of criteria for delivery to integration. One example of this is tools used for static and dynamic analysis as a complement to manual code review. Tools are available that can assist with workflow functions and process templates.

Interface handling: Insufficient control of interfaces is a source of mistakes and problems in development of products. The requirements on and designs of interfaces need to be captured and documented to assist in the development of components/modules. To ensure proper use of interfaces, a standardized way of documenting is needed to reduce ambiguity and misunderstandings. This documentation should include the following:

- Functionality
- Expected environment
- Limitations for use
- Usage
- Returned results
- Ownership

Note that this documentation complements the description of the interfaces in the SDK and is primarily used for Base Software design and implementations. This is also one area where the architecture may be influenced by the changes in the process: the attributes that can be retrieved from the system at runtime should also include information about possibilities for tests of the different modules. This ensures that proper verification can be done also in late stages of the integration process, i.e. when integrating the application.

It is important to ensure that the interface documentation also includes implicit dependencies that are related to generation of target code. This includes internal changes in modules that may not affect its interface but requires recompilation or linking.

Once an interface has been included in a Base Software release, the changes must be controlled and communicated. A decision process is needed to ensure that proper handling of changes in interfaces. Also, the product road map should be considered as any change of an interface may affect applications that need to ensure that the change affects the application as expected.

Changes of interfaces in Base Software need to be documented to ensure that they can be communicated to users and also to ensure that changes can be tracked. The documentation of a change should include the rational, a listing of affected parts of Base Software, as well as a description of how the change can influence the applications. Changes to interfaces in the applications should be handled in a similar way as application specific extensions may be transformed into common extensions.

Verification strategies: As the system is integrated iteratively in steps, there is a need to also have verification performed in steps. The verification of the kernel needs to ensure that the specified functionality is available and that the described interfaces are working correctly. As the kernel cannot be tested without an application that uses the interfaces, a test application is needed. This test application should include enough functionality to ensure appropriate coverage of the functions in the kernel. A second set of verifications is needed to ensure that the common extensions are working as specified. This calls for a different test application. Finally, the applications need to be tested. As the applications affect the functionality and the performance of the final system, parts of the tools and methods used for verification of the kernel and the common extensions need to be made available to the application engineers as part of the SDK.

As the Base Software is used for the development of many applications, deficiencies that remain after the verification will increase the risk that this error will affect one of the applications and causing problems in the field. This calls for higher standards in the verification of the Base Software than for the applications. The verification also needs to ensure that different combinations of the kernel and the chosen extensions are working. We note that the need to have well working verification of the Base Software also create a need for specific interfaces that enable the verification team to test the system sufficiently.

However, as the customer of the product will not distinguish the Base Software from the application, an error in the application may create a market problem that is as sever as a problem in the Base Software. Thus, the support for testing the applications is important, and should be a part of the Base Software SDK. It should also be part of the training and, if used, in the certification of the development unit.

3.2.4 Define alternative strategies

Define alternative strategies is the fourth step. In our case the strategies are divided from a business perspective and are based on how the application products are packaged, distributed, and verified. Two of the areas requiring process changes, product integration delivery criteria and interface handling, which are needed independent of the chosen strategy, and is based on the analysis of the changes in combination with the scenarios. The strategies are summarized in Table 1. Also the pros and cons as well as the risks have been captured, documented, and reviewed for each one of the strategies. These are related to the business objectives and are as such specific for the business situation the organization is performing under.

3.2.5 Decide on a strategy

The final step, *Decide on a strategy,* was in the case study delayed as the business implications for changing the product development to be more distributed needed further investigation

by product management. The final decision was to stay with the current model, strategy 1, and gradually move towards strategy 4. The decision was also to allow different locations to work in different ways, i.e. use different strategies, and develop their capabilities over time. As a consequence, the organization developing the Base Software SDK will deal with a diverse set of internal customers, requiring different levels of support. How this will be handled from a business and organizational perspective needs to be further investigated, e.g. how costs for the support should be divided between the different users of the Base Software SDK.

Table 1. Strategies and corresponding changes in processes

	Strategy 1	Strategy 2	Strategy 3	Strategy 4
	Centralized distribution		Distribution by application	
Process area or activity	Central verification	Verification by application	Central verification	Verification by application
Configuration management	One common repository	One common repository	Distributed repositories	Distributed repositories
Product Integration Strategy	Central application integration	Central application integration	Distributed application integration	Distributed application integration
Requirements on application development units	Ability to develop based on SDK	Ability to develop, and verify based on SDK	Ability to develop, and integrate based on SDK	Ability to develop, integrate, and verify based on SDK
Product integration delivery criteria	Common criteria for all levels of integration	Common criteria for all levels of integration	Common criteria for all levels of integration	Common criteria for all levels of integration
Interface handling	Secure interface handling for Base Software	Secure interface handling for Base Software	Secure interface handling for Base Software	Secure interface handling for Base Software
Verification strategies	Stepwise verification	Stepwise verification, with application developer doing final verification	Stepwise verification	Stepwise verification, with application developer doing final verification

3.3 Case Discussion and Lessons Learned

Compared to an ad-hoc method, the Business-Architecture-Process method facilitated the definition of the proposed changes in the development processes and the compilation of strategies. This was concluded by the organization after the investigations were performed, and compared to earlier architectural changes when no method was used for assessing process change. The difference in results is that necessary changes are implemented faster and that the organization is better informed and prepared for the new technology and new processes.

Four observations where made that will affect future use of the method. The first was that as we are using reference models as a basis for the appraisal of used processes, there is a risk that the proposed changes are generic process improvement proposal, and not connected to the change of the architecture. The second observation is that some of the changes in processes might only be depending on the changed business objectives, and not be a result of the architectural change. However, the process changes were not identified when the business objectives were initially analyzed. We conclude that the proposed method also helps the organization to identify these needed changes. The third observation was that it is important to continuously have a dialog with the sponsor. In the case study, the aspect of distributed development was reinforced. Finally, the involvement of some stakeholders was possible first after the strategies were formulated: the interest and time to analyze partial solutions and alternatives with too many degrees of freedom was minimal, and a full strategy was needed to ensure the full attention. Note also that there is substantial effort needed for the method as many stakeholders are involved. We think that to minimize the time and effort, it is important to plan workshops and other interaction early, ensuring that the effort spent is balanced with expected gains in reduced problems. All these observations will affect the next revision of the described method.

4. RELATED WORK

This section describes work that has been done related to influences between architecture, organization, and processes.

Various methods concerning the business objectives impact on both process and architecture exists but none combining the three. For architectural analysis the Architecture Tradeoff Analysis Method, ATAM [8], can be used. The goal of ATAM is to assess the consequences of architectural decisions in the light of quality attribute requirements. Typically there exist competing quality attributes such as modifiability, security, reliability, and maintainability that different stakeholders consider to be the most important. These quality attributes are broken down into scenarios. ATAM is divided into nine steps. These steps involve eliciting a utility tree and identifying risks, sensitivity, and tradeoff points. Since ATAM focuses on technical tradeoffs it can be complemented with the Cost Benefit Analysis Method, CBAM [10]. CBAM aids in the process of making architectural decisions by providing a return of investment (ROI), ratio. This ratio is the benefit divided by cost. A problem with quality attributes is that they are abstract and each stakeholder has it own interpretation of it. Neither ATAM nor CBAM compares different architectures and can therefore be hard to use when it comes to choosing a between different architectures. To aid in selecting a specific architecture over another, a method is presented in [15]. This method uses the elicitation of scenarios from ATAM and then analysis different architectural approaches with the Chainwise Paired Comparison method (CPC). CPC is based on the Analytical Hierarchy Process, AHP [12] but CPC only requires O(n) comparisons instead of the $O(n^{2)}$ needed with AHP. This method provides a structured reasoning why a specific architecture is chosen. The method is also highly scalable and can therefore be adapted to fit the resources available, however it does not consider the implications the chosen architecture has on the process, or how the process affects the architecture.

Another example, where different scenario-based methods have been used as a basis for assessment of architectures, has been described by Del Rosso [2]. This investigation is interesting as it

describes the evolution of a product line, and can be compared to the case study in this paper. It also compares scenario-based methods with performance assessments and experience-based assessments. However, the connections to process and organization are not examined.

Several methods, such as SCAMPI [13] and ISO/IEC TR 15504 (SPICE) [5], are available for assessing processes, and there are also methods available for evaluations of specific processes such as TPI. However, none of these are designed specifically to understand the combined changes of architecture, organization, and processes. Additional support for assessing processes can be found in different standards and reference models for development life-cycles [3, 6, 7, 14].

In [11], Ovaska et al describes how the architecture supports the product development processes in a multi-site environment, and the influence between the two is implicitly described. The study suggests that coordination efforts for activities are not enough, but that interdependencies between activities must be handled. This requires that a common understanding of the architecture. There is however no discussion about how the changes of architecture or process would influence each other.

5. CONCLUSIONS AND FUTURE WORK

Development of business objectives may initiate changes in the product development. The changes can affect architecture, organization, and process. However, our observation is that a change in one of these three aspects may influence the other two as a secondary consequence. We have described a method for assessing the influence a proposed architectural change can have on the process. Central to this method is the use of scenarios and process reference models. Combining solutions to process requirements into strategies gives a possibility for stakeholders to easy understand the implications of different decisions. By applying the proposed method during the refactoring of an industrial control system, we have based on the proposed method identified key areas and changes to these that need to be implemented in the development process. The case study has also resulted in the identification of additional details as useful input to the method. The case study supports our proposal that a structured method supports efficient and effective investigations of process changes due to architectural changes.

Threats to validity are that the reference models may be inappropriate for the investigation and that the selected scenarios are not representative and exhaustive for the product and organization. We argue that by selecting different reference models that are used in the organization today we cover current knowledge of processes for product and system development in this context. We have also ensured that the scenarios have been validated through review with product management, as well as with process owners, developers, and architects.

Future work includes detailing the description in the method, adding details on how each step should be performed, and also give additional examples. Additional details need to be added regarding scalability and resource needs for using the model in different types of organizations. There is also a need to expand the method to describe also remaining relationships depicted in Figure 1. This involves finding appropriate reference models for investigating organizations and architectures, and including the use of scenarios and combined solutions as strategies into the additional methods.

6. REFERENCES

[1] Bennett, K.H. and Rajlich, V.T., Software maintenance and evolution: a roadmap. in Proceedings of the Conference on The Future of Software Engineering, (Limerick, Ireland, 2000), ACM Press, 73-87.

[2] Del Rosso, C. Continuous evolution through software architecture evaluation: a case study. Journal of Software Maintenance and Evolution: Research and Practice, 18 (5). 351-383.

[3] EIA-731.1. Systems Engineering Capability Model, Electronic Industries Alliance, 2002.

[4] Greer, D. and Ruhe, G. Software release planning: an evolutionary and iterative approach. Information and Software Technology, 46 (4). 243-253.

[5] ISO/IEC15504:2004. Information technology - Process assessment, ISO/IEC, 2004.

[6] ISO/IEC12207:1995. Information technology - Software life cycle processes, ISO/IEC, 1995.

[7] ISO/IEC15288:2002. Systems engineering - Systems life cycle processes, ISO/IEC, 2002.

[8] Kazman, R., Klein, M. and Clements., P. ATAM: Method for architecture evaluation. CMU/SEI-2000-TR-004, Carnegie Mellon University, Software Engineering Institute, 2000.

[9] McFeeley, R. IDEAL^{SM}: A User's Guide for Software Process Improvement, Carnegie Mellon University, Software Engineering Institute, 1996.

[10] Nord, R.L., Barbacci, M.R., Clements, P., Kazman, R., Klein, M., O'Brien, L. and Tomayko, J.E. Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method (CBAM), CMU/SEI-2003-TN-038, Carnegie Mellon University, Software Engineering Institute, 2003.

[11] Ovaska, P., Rossi, M. and Marttiin, P. Architecture as a coordination tool in multi-site software development. Software Process: Improvement and Practice, 8 (4). 233-247.

[12] Roper-Lowe, G.C. and Sharp, J.A. The Analytic Hierarchy Process and Its Application to an Information Technology Decision. The Journal of the Operational Research Society, 41 (1). 49-60.

[13] SCAMPI update team, Standard CMMI® Appraisal Method for Process Improvement (SCAMPI^{SM}) A, Version 1.2: Method Definition Document, Carnegie Mellon University, Software Engineering Institute, 2006.

[14] SEI. CMMI® for Development, Version 1.2., Pittsburgh, PA, USA,, 2006.

[15] Wallin, P., Fröberg, J. and Axelsson, J., Making Decisions in Integration of Automotive Software and Electronics: A Method Based on ATAM and AHP. In Fourth International Workshop on Software Engineering for Automotive Systems (SEAS 2007), (Minneapolis, USA, 2007).