On the Expected Synergies between Component-Based Software Engineering and Best Practices in Product Integration

Stig Larsson
ABB Corporate Research, Västerås, Sweden
stig.larsson@mdh.se

Ivica Crnkovic
Mälardalen University, Västerås, Sweden
ivica.crnkovic@mdh.se

Fredrik Ekdahl *ABB Corporate Research, Västerås, Sweden*

Abstract

The expectations for a well working integration process are described in the Capability Maturity Model Integration (CMMI). Often during the integration process, weaknesses of the entire development process become visible. This is usually too late and too costly. Particular development processes and use of particular technologies may help to improve the performance of the integration process by providing proper input to it. For example, by the use of a component-based approach, the development process changes. Some of these changes may help in performing according to the process expectations. In this paper, examples of problems that have been observed in the integration process are described. Through a case study we describe a number of practical problems in current development projects. Based on this case study, we analyze how a component-based approach could help and lead to a more effective integration process.

1. Introduction

Product integration is a specific activity in the software development process. Very often this is also the activity where most of problems become visible and when it is either too late or at least very expensive to solve the problems. This is especially true for large and complex software products and systems which parts are developed and tested separately and when different mismatches are invisible until the products are integrated. The problems of integration usually have roots in previous phases, and most often in the lack of coordination between these phases. There are several reasons for this. First, it can be a communication problem and differences in goals between engineers conducting requirements analysis and specification, development, integration, testing and delivery of the products. Further there can be differences in the project goals (personified by project managers) and long-term goals (personified by system architects and domain experts). Second, a source of the problem is inadequate preparation of parts for the final integration. While being tested and verified on a part level, the product parts do not fit together. The reason for this problem can be inadequate test environments that are sufficient for testing particular functions of each part in isolation, but which do not reflect the impact of a particular part on the entire product. A third source of problems is inadequate information provided from parts. Very often there are many unwritten rules and "default" assumptions known on the part level that are invalid for the whole product. A fourth type of problems is features added into particular parts that are unknown to other parts and the entire product. By adding new features (such as improvement of particular functions or protocols) the architecture of the entire system can degrade or even break down.

Many of these problems originate from the ambiguity of separations of activities in the development process. While a separation of the different parts of the development processes exists in practice, this separation is often not well defined and formalized.

In component-based software engineering (CBSE), a separation of the development of components from the product integration is one of the main characteristics [1]. This raises several questions as described in [2]: What is a component, what is included into a component specification, what are the possibilities of predicting the product properties from component properties, how does a component interact with other components and its environment and similar.

So far the research focus for component-based engineering has primarily been on technical issues, and considerably less on process issues. It is however very important to know if the development process and CBSE are synergistic; will it be more efficient and effective or will it meet new challenges and maybe unsolved problems?

In this paper our aim is to investigate what the opportunities for improvement of the integration process and the development process in general by introducing a component-based development. Can the problems described be (at least partially) solved?

To investigate this possibility our research approach is the following. From a case study of a development process that has many similarities to a component-based approach, but still is not explicitly designed so, we highlight to the main challenges and problems that become visible in the integration phase. Further we analyze these challenges and discuss the possible changes and improvements in the process by introduction of a component-based development process.

The definition of a software component used in a product follows in this paper is broad, and the term is used to describe a part of a software system. However, in the discussions regarding CBSE, the notion of a component follows to a large extent [1], i.e. *software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system.* We also use the definition of a product as an application that can be sold and distributed independently, and has a clear customer value on its own.

The remainder of the paper is organized as follows. Section two describes the main characteristics of the integration phase of a development process, the main characteristics of a component-based development process, the changes in the integration process implied by component-based software engineering and related work. In section three, a case study is presented to show examples of how the integration process is performed today. Section four analyzes how the use of component-based software engineering would resolve today's challenges. Finally section five contains the conclusion and proposed future work.

2. Product Integration in relation to CBSE

The product integration process for software products addresses the assembly of software components. The target is to integrate components into a product and to ensure that the product works appropriately so that it can be delivered to customers. An integration process that is working well is expected to increase the probability that a development project delivers quality products in a timely manner. Component-based software engineering is targeting similar goals; to improve the productivity through use of high-quality components with predictable behavior. This section describes these two independent methods for improving the performance in development projects, and lists possible synergies.

2.1 Product Integration Best Practices

The Capability Maturity Model Integration, CMMI, [3] defines three goals for the product integration process. These are that (i) the product integration should be prepared, (ii) interface compatibility should be ensured and that (iii) the product components should be assembled and delivered.

The preparation for product integration typically includes preparation of an integration sequence. Different integration sequences should be examined and also include test components and equipment. The established sequence should be periodically reviewed to accommodate changes in the development project. The preparation also includes the establishment of the environment needed for product integration. One important decision in the preparation of the integration environment is if it should be developed in-house or bought from outside. In practice, the system will include both components that are bought and that are developed in-house.

A prerequisite for the possibility to ensure the interface compatibility is that the interface descriptions are complete. The design of the interfaces is important for the design of the components, but may also affect the design of the verification and validation environments. The interfaces need also to be managed throughout the project. Note that this is valid also for interfaces with the environment that the product is operating in.

The actual assembly of components should be done in accordance with the selected integration sequence. However, before a component is included in the product, the readiness for integration should be confirmed. The identity of the component needs to be established and the conformance to the specifications and established criteria should be confirmed. This confirmation can include a check of the status of the component, e.g. that the design of the component is reviewed, that the component is tested and that the interface descriptions are followed. Once assembled, the components should be evaluated. This is done based on the integration sequence and the verification specified. Based on the systems created in the product integration process, the system is verified and validated. When all product components have been integrated, the product should be delivered to the appropriate customer. This can be made in an iterative fashion, with part deliveries, internal deliveries and of course as a final delivery for production.

2.2. Developing systems with CBSE

When developing a system based on components, the focus is on the system requirements, the overall system functionality and the mapping these requirements to components. However, the implementation of individual components is not in the focus of the process. The components used in the solutions are thus considered to be developed or acquired independently of the development of the system.

The activities performed when developing a system are similar to those for any non-component-based

development; they include requirement analysis, architectural specification, component selection and evaluation, system design, implementation, integration, verification and validation. A specific activity here is component selection, but also other activities have specific parts that are influenced by the component-based approach. As the dependencies between these activities are strong, it is important to note that they are usually performed in an iterative fashion, and that these iterations should be taken into account when planning the system development.

The requirement analysis is done to transform the collected needs into system requirements. The task is also to define the scope for the system. Based on the system requirements, it is possible to define the system architecture and to derive the component requirements. As the definition of components to be used and the resulting system properties are investigated, it may be necessary to reexamine the system requirements and prioritize what is most important. The reasons may, for example, be that requirements are found to be contradictory, that the selected solution is too expensive or that the time-to-market requirements cannot be met.

When an initial architecture has been created, a decision how to obtain the needed components is taken. If the decision is to develop a new component, specific for the system, the development will be based entirely on component requirements derived from the system requirements. This decision will also make sure that the component fits to the architecture. Preexisting components developed in-house may be used as-is, but may also require modifications. As this reduces the possibilities for reuse, it is more likely that interactions between the components are modified, that adapters are created, or that the architecture is modified to fit the selected components. This is also likely when using commercial components, as these normally require a specific architecture. Both types of pre-existing components may influence the architecture, especially if a specific component framework is required. To find and select components based on the component requirements is a challenge. One reason is that it is difficult to derive these requirements from the system requirements. If the component is not created specifically for the developed system, it is unlikely that a component exactly matching the requirements can be found. In addition to fulfilling the requirements, the components must also coexist in the system, which leads to the need to investigate compatibility issues between the components and also with the selected component framework. It is worth to mention that already in the selection process, integration activities can be performed. Often when validating components they must be composed with other components and integrated in the system environment.

The system construction depends on the chosen architecture and on the selected component technology and framework. The design also depends on what types of components will be used in the system. More reuse and commercial components will reduce the freedom to select different design solutions.

The implementation activities should be limited to adaptations of the components and connections between the components. This should be a minor task, but if the components are not properly selected, the work may be substantial. Also verification of the component behavior in the selected environment should be a part of the implementation. This may lead to additional development of code to handle the components in- and outputs or changes in the way the component is set up.

To ensure that the quality requirements on the system can be met, the integration of the system is crucial and should be started as soon as possible in the development cycle. The activities include determination of integration sequence, verification that the components adhere to the interface description, and provision of systems appropriate for verification and validation. Additional tasks are to identify the need for additional implementation and to monitor the system properties as these emerge when the system is integrated. The integration will depend on the architectural solution, as the possibility to build systems is determined by the selected architecture as well as the component model and framework. The verification that the requirements are met can start as soon as the first integration has been made, while the validation that the customer expectations are met can only be made when the final assembly has been made.

In component-based software systems, components may exist also in runtime. The result of this is that it is possible to change the system while in operation, or at least without replacing the entire system, by replacing components. This simplifies the maintenance and error correction and also makes enhancements possible. A well-designed architecture is however necessary as the dependencies between different parts and components in the system make such changes dangerous if the consequences are not well understood. Special care must be taken when a component is used by several other components.

There are many reasons why component-based approach can improve the integration process. We list here the most important.

- *Component specification.* The basic principle in component-based approach is a separation of component specification from its implementation through its interface. This separation is stronger than in object-oriented approach since all interaction is supposed to be performed through interfaces. This principle drastically decreases the risks for

introduction of unknown properties and architectural mismatches. Though it should be noted that many component models do not follow this principle, in particular for required interface, which may cause many unpredictable problems.

- *Early integration requirements.* For component validation usually a kind of integration procedure must be made. An early integration process can show problems that might remain hidden until the final integration.
- *Standardized interoperation.* Component models define the standards for interconnection between the components. This eliminates a number of potential errors due to architectural mismatches.
- *Integration tool support.* Integration is an inherent part of a basic approach of CBSE. For this reason the component-based technologies focus on this process and usually provide powerful integration tools.

2.3. Related work

This section describes some of the work that has been done related to integration in component based software systems. In the related work, the integration process partly includes what is often described as the composition process.

The notion that all development phases, including the integration activities, need to be reconsidered when working with component-based software is pointed out in [4]. It is also mentioned that the current component models do not take enough of the needs of the system developer into account. A part of the information that is mentioned as underdeveloped is the specific collaboration rules for interfaces and component behavior. This influences the ease with which a developer can determine if the chosen components fulfill the requirements of the system.

The PECOS project [5] [6] describes an approach and a software process to be used for basing embedded systems on component-based technology. The composition process is examined and described. It is, however, not compared to the overall expectations on the integration process.

The OOSPICE project [7] was targeted at overcoming the shortcomings experienced when applying software process improvement approaches to component-based development. In [8], the observation that component-based development is integration-centric is elaborated.

In [9], the risks in the composition phase for component-based software development are listed. Several of the risks are related to the integration process, and a method for how to deal with these risks is outlined.

3. Case study

The case study was performed at an ABB unit developing industrial control systems. The system has evolved through several generations, and a new generation of the system is currently being developed. Compared to the first generation, where the effort was three man months, the effort for software development in the current development is estimated to about 100 man years.

In essence, the controller has layered architecture and within layers, component-based design. The implementation consists of approximately 2500 KLOC of C language source code divided in 400-500 components, organized in 8 technical domains. The software platform defines infrastructure that provides basic services like: a broker for message-based inter-task communication, configuration support, persistent storage handling and system startup and shutdown.

3.1. Research method for the case study

The methods for the case study include interviews, document reviews and an observation. The interviews have been based on a set of open questions, and have been conducted as discussions about the integration process. The document review was performed on the documentation describing the integration process, the training material for the organization as well as the files used for and as a result from the build process. As the purpose of the observation was to identify challenges, it was designed to obtain as much information as possible, i.e. the decision was to perform an unstructured observation.

3.2. Product Integration

The development of the system is conducted in different development groups, and there are separate groups for the integration, verification and validation activities. As the system has evolved over several years and parts of it have been replaced with new solutions, the development environment as also been changed. For example two different configuration management systems are used. Unique tools are used for the integration group that also handles the build process. Developers have their own set of tools for building on local systems. Training of the developers is done as part of the general information about the system given to the staff. The developers also get hands-on training in the projects.

The system evolution is performed in an incremental way. The implementation of a functionality described in the requirement specification is distributed to different integration points (IP), as shown in figure 2.

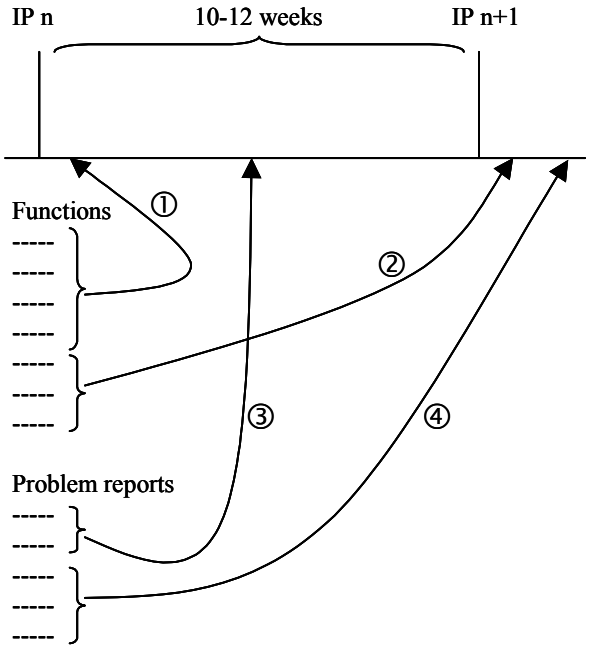

Fig 2. Distribution of functions and error corrections

The changes may occur in a project where the intended functionality for IPn is redistributed to IPn (1) and to IPn+1 (2). This redistribution is based on the progress in the project, the priorities for the different functions as determined by product management and the possibilities to alter the decided integration strategy. Also the problem reports and the error corrections related to them are assigned to the different integration points (3 and 4). Product and technology management decides what errors should be corrected for a specific integration point.

The procedure used when reaching an integration point is shown in figure 3. The width of the arrows in the figure (4) represents the amount of new functions or error corrections that are accepted for integration. As an integration point is approached, the possibility to add new functionality is reduced and increasingly monitored. This is illustrated by the narrowing towards the point of the arrow (1). As the "beta drop" is reached, the version is branched to a release track. All release tracks are made available to the organization for use in testing and further development. Errors that are found in the verification and validation are considered for correction for the new integration point (2). After the release "beta drop", the development groups have the possibility to add new functionality again (3).

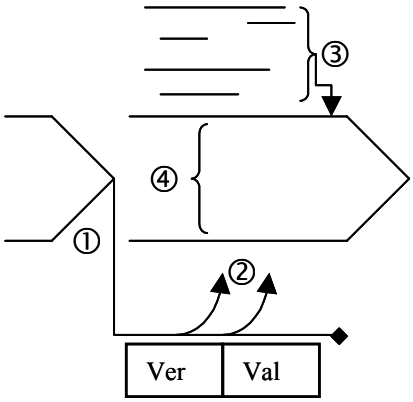

Fig 3. Integration point activities

An important prerequisite for a working product integration process is an appropriate build process. It is also in the build process that many of the problems with the product integration process appear. For our case study system, the current build process has been in place for four years and is continuously updated and improved. Each day, the full system is built and generated for several target systems with a total of more than 15 versions. A separate build machine is used, and each build takes seven hours. As soon as a build is started, it is possible to start delivering to the next one. New code to be included in a system build is put on a build queue. Once put in the queue, the component cannot be deleted from the queue. The two different software configuration management (SCM) systems used give different protection against mistakes. One prevents mistakes, as there are no possibilities to check code directly into the build directories. The other SCM system makes a direct merge into the release directory without the delivery through the queue.

The build is normally done during night, so the result of the build is known in the morning. The person responsible for execution of the build process examines the log files. In case of problems, the responsible persons are notified and asked to correct the problem. The result of a severe problem is normally that the build will be delayed one day. However, as the deliveries in the new build queue can be included, the setback may be different for different parts of the project. Today, no metrics or statistics are captured how often the problems occur or to see what causes the problems in the integration process. The error reports from the findings are however tagged with the build identity to make error correction easier.

The problems identified in the case study relate to three main areas. The first issue is the delivery of code to the build process. The code may be delivered late, or a function is not fully delivered. Also, the two different ways to deliver the code for integration is a concern. One system handles this automatically, while the other requires manual checking that the right things are

included. The second issue is the low quality, e.g. errors that cause the builds or initial integration tests ("smoke tests") to go wrong. This can be due to insufficient tests and system generation by the developers. They normally test only a few of the possible combinations. The result may be that the system generated works for the tested configurations but fails in the others. The final issue relates to components that influence other parts of the system. It may be that changes in include-files affect other components. This is possible as no routine or mechanism for how to handle the communication of changes has been established. This and the second issue may be discovered in the smoke test following the system generation.

4. Analysis

When we compare the problems discovered in the case study to the product integration expectations as described in [3], we see several activities that can be put in place to improve the process. The improvements of course can be made without the introduction of CBSE. However, our analysis of three main problem areas supports the idea that a CBSE solution would reduce the difficulties.

A first improvement is related to the checks at integration time and deals with the first two problems, delivery of incomplete functions and code with low quality. The rules for including a component at an integration point should be appropriate so that they can be followed both for major additions of functionality and for minor error corrections. This means that the rules should be suitable for different types of changes, but need to be followed for all inclusions at an integration point. To enable this, additional power must be given to the integration team. The development groups will through this lose some control but in return less often get unstable systems or broken builds. The improved check at integration time would be supported by CBSE as the delivery of code to integration would be done as ready-made components. This would also reduce the problem of functions delivered before they are ready. Through the use of CBSE, the poor quality can be reduced, as components should be tested in all environments they are envisioned to be used in.

The third and maybe most important problem area is the need to handle dependencies, i.e. interfaces, between different components more strictly. Changes to interfaces should be controlled and communicated. To achieve this, the interfaces must be sufficiently documented. Also, any changes to the interfaces must be controlled at integration time to ensure that they have been approved and communicated. In CBSE, the separation of the processes for developing components and for building systems into two separate processes helps in better defining the interfaces for the components. A component without a clearly defined interface cannot be used unless the developers of the system have full knowledge about the component. Introducing a clear separation in this manner would also increase the clarity in the dependencies between the components. It would also make it possible to have a more thorough, or strict, procedure for accepting a new version of a component for a specific integration point. Using CBSE, improved descriptions of interfaces would diminish the influence from one component to another, or at least make these dependencies visible.

For all three main problems, we predict that CBSE would help in reducing the problems. The cost is however that the system, processes and organization need to be changed to accommodate CBSE.

A first step would be the introduction of a complete component model. There are experiences that by introduction of component models have significantly improved the development process [2]. Of course introduction of a component model would require additional efforts. First the existing code and basic architecture should be reused as much as possible. This implies that widely used components models such as .NET or EJB are not appropriate. Rather a simple, probably in-house developed component model should be deployed. This component model could be built incrementally, starting with basic principles such as interface specification and automation of integration of components.

A second effort required would be a componentization of the existing code. Since today many of the dependencies between the components are implicit, their separation might be a tedious work. However such a work would pay off in the long run, since errors made today depending on hidden connections between components would be reduced. Efforts to describe the dependencies explicitly are being made in the case study system today, with promising results. A continued work in this direction would result in an architecture that is properly documented and better cohesiveness of components which are the basic prerequisites for efficient system development and evolution.

Finally, the organization of projects and departments to clearly divide the work into development of components and development of the system is needed.

5. Conclusions and future work

A case study has been compared to the generic requirements on a best practice product integration process [3]. In addition to this, we have analyzed what support the current process may get from using component-based software engineering. Our conclusion is

that several of the requirements for a well working integration process can get substantial support through skilled use of well defined components. The support comes from the fact that components should be well documented, tested in the environment they are intended for and that any dependencies to other components (or the environment) should be explicitly highlighted.

Future work should include additional case studies in industry. Both development units working with components and with traditional software need to be further examined. These investigations need to include measurements on the problems caused by an insufficient integration process as well as root cause analysis. The purpose of these investigations would be to confirm or refute the conclusions in this paper that CBSE helps in providing a platform for efficient and effective software product integration.

Further additional analysis should be done on a feasibility of full componentization of the systems. The efforts and return-on-investments for re-architecting and for development and introduction of a component model should be estimated.

6. References

[1] Szyperski, C. *Component Software -- Beyond Object-Oriented Programming*, Addison-Wesley, Reading, MA, 1998.

[2] Crnkovic, I., and M Larsson, *Building reliable component-based software systems*, Artech House, Boston, 2002.

[3] Chrissis, M.B., M. Konrad, S. Shrum, *CMMI,* Addison-Wesley, Boston, MA, 2003.

[4] Zeidler, C., Componentware Glory and Crux for early industrial adopters, Object Oriented Programming conference OOP 2000, Munich, Germany, 2000.

[5] Winter, M., C. Zeidler and C. Stich, "The PECOS Software Process", Workshop on Components-based Software Development Processes, ICSR 7, Austin, TX USA, 2002.

[6] Müller, P., C. Zeidler, C. Stich and A. Stelter, "PECOS — Pervasive Component Systems", Workshop on ”Open Source Technologie in der Automatisierungstechnik”, GMA Kongress, Baden-Baden, Germany, 2001.

[7] The OOSPICE project, http://www.oospice.com

[8] Stallinger, F., B. Henderson-Sellers and J. Torgensson, ”The OOSPICE Assessment Component: Cusomizing Process Assessment to CBD”, in *Business Component-Based Software Engineering*, edited by F. Barbier, Kluwer Academic Publishers, Boston, USA, 2002.

[9] Kotonoya, G. and A. Rashid, A strategy for Managing Risk in Component-based Software Development, Euromicro 2001 CBSE workshop, Warsaw, Poland, 2001.