Abstract

The product integration is a particularly critical phase of the software product development process as many problems originating from earlier phases become visible in this phase. Problems in product integration result in delays and rework. One of the measures to decrease the late discovery of problems is the use of development standards and guidelines that define practices to ensure correctness of the product integration. However, even if such standards and reference models exist, they are in not used consistently. One of the reasons is a lack of a proof that they indeed improve the integration process, and even more important, that they are sufficient for performing efficient and correct product integration.

The conclusion of the presented research is that the available descriptions in standards and reference models taken one by one are insufficient and must be consolidated to help development organizations improve the product integration process. The research has resulted in a proposed combination of the activities included in the different reference models. This combination has been based on a number of case studies. Through the case studies performed in seven different product development organizations, a relationship between problems that are observed and the failure to follow the recommendations in reference models is identified. The analysis has indicated which practices are necessary, and how other practices support these. The goal with the research is to provide product development organizations with guidelines for how to perform software product integration.

One additional finding of the research is the existence of relation between software architecture and the development process. A method for identifying dependencies between evolvement of software architectures and adaptation of integration practices has been demonstrated.

Acknowledgements

Foremost, I would like to express my gratitude towards my supervisor professor Ivica Crnkovic. Ivica has guided me throughout the years, and has been there to help me even when schedules have been overloaded. Many thanks go also to my assistant supervisors Dr. Fredrik Ekdahl, for helping me to start this journey and keeping me on track, to Dr. Rikard Land for all cooperation, and to both of them for interesting discussions on everything from integration and research methods to computer games and langoustes.

Special thanks to Christer Persson, Petri Myllyperkiö, and Stefan Forssander for collaboration on the case studies and for helping me remember the realities of product development, and to Per Branger and Clarens Jonsson for great teamwork and helpful comments. Thanks also to all other colleagues at ABB and all the participants in the case studies.

A major advantage with conducting research studies is all the people you meet. I would like to thank my fellow Ph.D. students Jocke, Johan F., Johan K., Markus, Micke, Peter, and Stefan for reviews, interesting meetings, and guidance to great places.

I would also like to thank all my friends and colleagues at Mälardalen University providing a fruitful environment and giving support when I have needed it.

The work would not have been possible without the support from ABB Corporate Research and KKS, providing me with resources for my research.

I have over the last four years been poor in keeping contact with many of my relatives and friends. Hopefully, I will not start another project like this too soon again so that there will be time to meet (if you still remember me).

Ultimately, this has been possible only through the understanding and support from my wife AnnKi and our daughter Camilla. You are my inspiration and I love you both so much.

Stig Larsson

Shanghai, November, 2007

List of Included Papers

Paper A *On the Expected Synergies between Component-Based Software Engineering and Best Practices in Product Integration*, Stig Larsson, Ivica Crnkovic, Fredrik Ekdahl, Euromicro Conference, IEEE, Rennes, France, August, 2004

Paper B *Case Study: Software Product Integration Practices*, Stig Larsson, Ivica Crnkovic, Product Focused Software Process Improvement: 6th International Conference, PROFES 2005, Springer, Lecture Notes in Computer Science, Volume 3547 / 2005, Oulu, July, 2005

Paper C *Product Integration Improvement Based on Analysis of Build Statistics*, Stig Larsson, Petri Myllyperkiö, Fredrik Ekdahl, presented in a shorter version at ESEC/FSE Conference 2007, Dubrovnik, Croatia, September 2007

Paper D *How to Improve Software Integration*, Stig Larsson, Petri Myllyperkiö, Fredrik Ekdahl, Ivica Crnkovic, submitted to the Information & Software Technology journal, Elsevier

Paper E *Assessing the Influence on Processes when Evolving the Software Architecture*, Stig Larsson, Anders Wall, Peter Wallin, presented at IWPSE 2007, Dubrovnik, Croatia, 2007

List of Related Papers

- *Component-based Development Process and Component Lifecycle*, Ivica Crnkovic, Michel Chaudron, Stig Larsson, International Conference on Software Engineering Advances, ICSEA'06, IEEE, Tahiti, French Polynesia, October, 2006
- *Experience Report: Using Internal CMMI Appraisals to Institutionalize Software Development Performance Improvement*, Fredrik Ekdahl, Stig Larsson, 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06), p 216-223, IEEE Computer Society, Cavtat, Croatia, September, 2006
- *Selecting CMMI Appraisal Classes Based on Maturity and Openness*, Stig Larsson, Fredrik Ekdahl, PROFES 2004 - 5th International Conference on Product Focused Software Process Improvement, Springer-Verlag Berlin Heidelberg New York, Kansai Science City, Japan, Editor(s):Frank Bromarius, Hajimu Iida, April, 2004

Additional Publications

Journals

- *Industry Evaluation of the Requirements Abstraction Model*, Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin, in print, Requirements Engineering, 2007.
- *A Model for Technology Transfer in Practice*, Tony Gorschek, Claes Wohlin, Per Garre, Stig Larsson, IEEE Software, vol 23, nr 6, p88-95, IEEE, November, 2006
- *Component-based Development Process and Component Lifecycle*, Ivica Crnkovic, Michel Chaudron, Stig Larsson, Journal of Computing and Information Technology, vol 13, nr 4, p321-327, University Computer Center, Zagreb, November, 2005
- *Integrating Business and Software Development Models*, Christina Wallin, Fredrik Ekdahl, Stig Larsson, IEEE Software, vol 19, nr 6, p28-33, IEEE Computer Society, November, 2002

Thesis

- *Improving Software Product Integration*, Stig Larsson, Licentiate Thesis, Mälardalen University Press, June, 2005

Conferences and workshops

- *Software In-House Integration – Quantified Experiences from Industry*, Rikard Land, Stig Larsson, Ivica Crnkovic, Euromicro Conference, Track on Software Process and Product Improvement (SPPI), IEEE, Cavtat, Croatia, August, 2006
- *Merging In-House Developed Software Systems – A Method for Exploring Alternatives*, Rikard Land, Jan Carlson, Ivica Crnkovic, Stig Larsson, Quality of Software Architecture (QoSA), University of Karlsruhe, Västerås, Sweden, June, 2006

- *Architectural Concerns When Selecting an In-House Integration Strategy – Experiences from Industry*, Rikard Land, Laurens Blankers, Stig Larsson, Ivica Crnkovic, 5th Working IEEE/IFIP Conference on Software architecture, WICSA, p 274-275, IEEE, Pittsburgh, PA, USA, November, 2005
- *Software Systems In-House Integration Strategies: Merge or Retire - Experiences from Industry*, Rikard Land, Laurens Blankers, Stig Larsson, Ivica Crnkovic, Fifth Conference on Software Engineering Research and Practice in Sweden (SERPS), p 21-30, Mälardalen University, Västerås, Sweden, October, 2005
- *Architectural Reuse in Software Systems In-house Integration and Merge – Experiences from Industry*, Rikard Land, Ivica Crnkovic, Stig Larsson, Laurens Blankers, First International Conference on the Quality of Software Architectures (QoSA 2005), Springer Verlag, Erfurt, Germany, September, 2005
- *Process Patterns for Software Systems In-house Integration and Merge – Experiences from Industry*, Rikard Land, Ivica Crnkovic, Stig Larsson, 31st Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Track on Software Process and Product Improvement (SPPI), IEEE, Porto, Portugal, August, 2005
- *Concretizing the Vision of a Future Integrated System - Experiences from Industry*, Rikard Land, Ivica Crnkovic, Stig Larsson, 27th International Conference Information Technology Interfaces (ITI), IEEE, Cavtat, Croatia, June, 2005
- *Component-based Development Process and Component Lifecycle*, Ivica Crnkovic, Stig Larsson, Michel Chaudron, 27th International Conference Information Technology Interfaces (ITI), IEEE, Cavtat, Croatia, June, 2005
- *Towards an Efficient and Effective Process for Integration of Component-Based Software Systems*, Stig Larsson, SERPS'03 - Proceedings of the 3rd Conference on Software Engineering Research and Practice in Sweden, Lund, Sweden, October, 2003
- *Are Limited Non-intrusive CMMI-based Appraisals Enough?*, Stig Larsson, Fredrik Ekdahl, Proceedings of the ESEIW 2003 Workshop on Empirical Studies in Software Engineering WSESE 2003, Fraunhofer IRB Verlag, Stuttgart, Germany, September, 2003

- *Combining Models for Business Decisions and Software Development*, Christina Wallin, Stig Larsson, Fredrik Ekdahl, Ivica Crnkovic, Euromicro Conference, IEEE, Dortmund, September, 2002

Table of Contents

Part 1

Chapter 1. Introduction

The product integration process is a set of procedures used to combine components into larger components, subsystems or final products and systems. Product integration enables the organization to observe all important attributes that a product will have; functionality, quality and performance. This is especially true for software systems as the integration is the first occurrence where the full result of the product development effort can be observed. Consequently, the integration activities represent a highly critical part of the product development process.

We refer to the definition of integration for product and system development found in the glossary of EIA 731.1 (interim standard) [1]:

> ”Integration: The merger or combining two or more elements (e.g., components, parts, or configuration items) into a functioning and higher level element with the functional and physical interfaces satisfied. “

This definition describes the product integration process without limiting its use to an implied product development life-cycle model.

Practices for product and system development are described in a number of standards and models such as ISO/IEC 12207 [2] and CMMI [3]. It is noticeable that most standards and reference models deal with product and system development without distinguishing software as a specific item. Steve McConnel describes integration in [4] as “the software development activity in which you combine separate software components into a single system “. Also with this description, it is easy to use the statement (without the software) for any type of integration. However, when going more into detail, there are important differences between the integration of software and other types of integration.

Product integration is in most organizations performed in an iterative and incremental manner, and it is a central part of any product development project. Figure 1 shows a data flow diagram of the product integration process interaction with other processes as described in the CMMI [3]. The

results from design and implementation in Technical Solution are transferred to Product Integration in a controlled manner. The results from Product Integration are used for Verification and Validation activities. When a version of the product or system is ready, it is made available for internal or external customers.

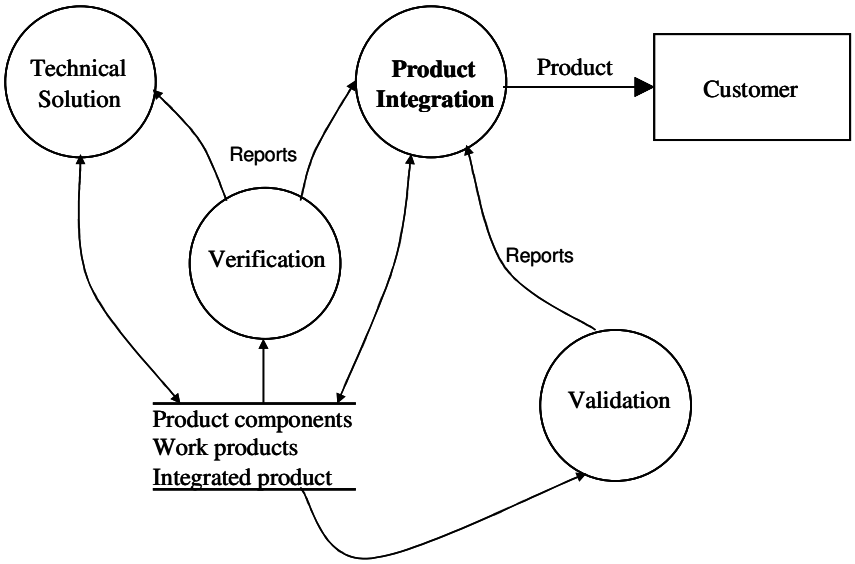

Figure 1. Product integration and related processes

Typical problems in product integration are that the components delivered for integration are not ready, that interfaces between components are insufficiently defined or followed, and that the environment needed for the integration is inappropriately prepared. This leads to the questions if something can be made to improve the product integration process, and what the key elements of product integration are.

1.1 Research Motivation

Good practices for product integration are described in different reference models and standards such as ISO/IEC 12207 [2], CMMI [3], EIA-731.1 [1], and ISO/IEC 15288 [5]. Problems that have been identified include the frequent failure to utilize the knowledge available, or that the recommendations in the reference models are insufficient. This is demonstrated by Campanella who presents an investigation into costs related to different phases in [6], and by RTI describing integration in

relation to testing in [7]. Bajec et al have investigated why available methods are underused in [8] and conclude that inflexibility, and the perceived lack of usefulness are among the reasons. My own experience is that practices described in reference models are too often neglected or misunderstood. This leads to the absence of, inadequate, or insufficient use of activities that would ensure efficient and effective product integration.

There are many examples of how minor mistakes made in an earlier phase complicate and delay the product integration processes. For example, builds fail when code which has not been properly compiled is delivered for integration, or interfaces are changed without checking the impact of that change, or without checking that the corresponding changes in other parts of the system have been done. The consequence is that errors and problems are discovered late in the development process. There are two major negative results from failing product integration processes:

- Activities which use the result and outputs from the product integration process are affected and delayed. These activities include further implementation of functionality, verification, and validation. As integration is performed throughout the project life-cycle, each delay of the results from integration will affect the development effort significantly.
- Work performed in earlier phases must be redone if problems are discovered late in the development process, adding to the needed resources, and further delaying the project results.

Examples from our research include a case study using daily builds showing build statistics that indicate that every fifth build fails due to insufficient use of good practices for product integration. Combining this with the indication that every failed build typically delays the development project by half a day causes a delay in the project of approximately 10% as a direct consequence.

Failure in the integration, which is the result of errors in previous phases can thus be expensive and should be avoided. Practices described in different reference models may help in avoiding these problems. The described practices can be divided into three categories:

- *Preparation of product integration.* This includes decisions on strategy, on integration sequence, and on the criteria for integration
- *Management of interfaces between components.* The integration processes include checking that interfaces are properly defined, and that changes to interfaces are controlled, but not the definition and

design of the interfaces as this is a design issue which is handled in the design process

- *Execution of the product integration.* The execution includes ensuring that the strategy, sequence and criteria are followed, the assembly of components, and the performance of planned tests to verify successful integration

Reference models are of value because they are collected experiences from industry. However, they are not widely used. A question is thus what is needed to help organizations better follow reference models in different product integration undertakings. In addition, the specifics of the reference models differ, and there is a need to understand how these differences may affect the performance of product integration in product development projects.

There is a distinction between products containing software and products that have little or no software: the integration of software products is not tested repeatedly in production through fabrication and manufacturing. This repetition helps the organization prevent problems from reaching the market. This testing is of course most efficient during the development process when the manufacturing organization is active in the project and in the testing to ensure that the production will flow smoothly. Things that can reduce the risk in software product development are the use of continuous or nightly builds, and automated regression testing.

The purpose of this research is thus to determine what changes are required to the current body-of-knowledge for the software product integration process as described in models and standards to be effective.

An additional issue is how the use of the practices described in reference models can be supported in different ways. Examples include training, the use of technologies designed to support product integration, and tools that help engineers define and use components that are well defined are some examples of support that can improve the use of practices described in reference models. Observations made indicate that a closer association between technical and process aspects is needed to ensure the awareness of engineers of the importance of product integration. This means that it is necessary to investigate also the connection and relation between architecture and product integration processes. As a first step, this research considers the influence of architecture on product development processes and proposes a method to find this influence. The use of this method creates

a better understanding of the importance of certain aspects of product integration among engineers.

The importance of architecture in this context is that it affects the possibilities to reach efficient and effective product integration. Product development organizations often have the focus on technical changes, and a wider knowledge of the effects of these changes on the process is needed. If this is developed, we foresee engineers becoming more interested in what affects their work, and how it can be performed more efficiently.

1.2 Research Questions

As described, product integration is a vital part of product development and the main focus and research problem is to understand *what factors influence the possibilities to achieve efficient and effective product integration.* The characteristics of efficient product integration are that unnecessary work is avoided and delays due to integration problems are prevented. Effective product integration is achieved if problems related to the interaction between components are captured, and the planned functionality for a specific integration is achieved. Other important matters related to product integration include delayed time to market, insufficient quality, and inefficient use of resources.

A first step towards understanding how reference models can help was presented in my licentiate thesis [9] and the research presented here is a continued and a more detailed investigation of the reference models. In addition, a first step has been taken in understanding the influence of architecture on processes with emphasis on product integration.

The research questions below make the research topic concrete. The first two questions are related to the use of standards and models as a vehicle for improving product integration, and if there is a need to improve the current reference models. The first question aims to investigate the use of current reference models:

> *Are the practices described in available reference models for product integration necessary and sufficient for visible reduction of problems in the product integration process?* **(Q1a)**

In answering this question, we can find the practices that are most relevant for efficient and effective product integration and what is included in the reference models.

Different reference models cover different aspects of product integration. The reference models represent together the current body-of-knowledge for product integration. Based on the differences and the combined body of knowledge, the next question is:

> *What additions and modifications are needed in the available reference models to take advantage of current body of knowledge in product integration?* **(Q1b)**

Different types of support can help increase use of the described practices. This includes training, tool support, and use of technology that simplifies the product integration.

In addition to the previous question this thesis states a question about integration in a context of software evolution. The reason for this is a recurring observation from the case studies – a relation between changes in the product architecture and a need for changes in the development process. Based on experiences from the case studies we decided to investigate how product development organizations can understand how product integration processes are influenced by changes in the architecture. The evolution of system or product architectures may change the requirements on the product integration process. Failing to change the process when altering the architecture may be one reason why the used product integration practices are not sufficient. We need therefore to understand the influence on process from architectural decisions. This leads to an additional question:

> *How can necessary changes in the integration process due to changes in the product architecture be identified and implemented?* **(Q2)**

1.3 Thesis Overview

The first part of this thesis is an overview of the research, while the second part is a collection of papers that documents details of the research questions, methods, and results.

In part one, chapter 2 relates product integration to different aspects of software product development, including reference models, life-cycles, architecture, product lines, and component-based software engineering.

The method used and the validity of the presented research are discussed in chapter 3, focusing on the whole research project.

Chapter 4 includes a summary of the research results, and an expansion on some of the findings from the papers included.

Chapter 5 wraps up the overview part with conclusions and a look at possible future work.

Part two of the thesis includes the following papers:

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

This paper describes the product integration practices in one product development organization. Problems observed are compared with component-based development practices to investigate if these can help the organization follow good practices as described in the CMMI.

Presented at the Euromicro Conference, Rennes, France, August 2004. Authors: Stig Larsson, Ivica Crnkovic, Fredrik Ekdahl.[10]

I was the main author; I contributed with the description of good practices in product integration, the methodology, the case study, the analysis and conclusions. The co-authors contributed with advice regarding methodology, discussions regarding the analysis and conclusions, and reviews.

Paper B **"**Case Study: Software Product Integration Practices"

This paper includes case studies from three organizations. Practices used in the organizations are compared to EIA-731, and the problems encountered by each of the organizations are described. Problems are mapped to practices, and the conclusion is

that the standard includes activities that can help organizations avoid problems which can appear when integrating components to systems.

Presented at PROFES 2005 Conference, Oulu, Finland June 2005. Authors: Stig Larsson, Ivica Crnkovic. [11]

I was the main author; I contributed with the description of good practices in product integration, the methodology, the case study, the analysis and conclusions. The co-author contributed with advice regarding methodology, discussions regarding the analysis and conclusions, and reviews.

Paper C "Product Integration Improvement Based on Analysis of Build Statistics"

This paper proposes a method for mapping project data to different practices and combines this mapping with project appraisal results to form a basis for focused performance improvement. The product integration processes in four projects from three organizations were examined using the proposed method and the findings are presented. The study demonstrates how the two components, collected metrics and appraisal results, complement each other in the effort to develop product integration process improvement effectiveness.

Presented in a shorter version at ESEC/FSE Conference 2007. Authors: Stig Larsson, Petri Myllyperkiö, Fredrik Ekdahl. [12]

I contributed with the description of good practices in product integration, methodology, and two of the case studies and the analysis for these as well as the conclusions. Petri Myllyperkiö contributed with two case studies, and for these we made the analysis together. Both co-authors contributed through discussions and reviews.

Paper D "How to Improve Software Integration"

This paper consolidates the investigations in paper A, B and C with chapter 4 of my licentiate thesis [9] to show the possibility to enhance current reference models. Seven case studies are compared to five reference models. A combination of the findings from the cases and the models result in a proposed set of 15 practices for successful product integration.

Submitted to the Information & Software Technology journal, Elsevier. Authors: Stig Larsson, Petri Myllyperkiö, Fredrik Ekdahl, Ivica Crnkovic [13].

I contributed with the description of product integration practices in reference models, methodology, and five of the seven the case studies. I also made the analysis which was then discussed with the co-authors, and prepared the conclusion. All co-authors contributed through reviewing the paper.

Paper E "Assessing the Influence on Processes when Evolving the Software Architecture",

This paper expresses different relationships between architectural changes, process changes and the underlying business objectives. As an example of how the understanding of these relationships can be used, we describe a method for assessing the process changes needed when refactoring is performed. Details regarding the consequences for the product integration process are included as examples.

Presented at the 9th International Workshop on Principles of Software Evolution, IWPSE, 2007. Authors: Stig Larsson, Anders Wall, Peter Wallin.[14]

I was the main author and lead the study. I contributed with the description of the proposed method, while the case description, the related work, and the conclusions were made in cooperation with the co-authors.

In addition, the following papers are indirectly related to the thesis. Material from these papers has been used in the preparation of part 1 of this thesis:

- "Component-based Development Process and Component Lifecycle", Ivica Crnkovic, Michel Chaudron, Stig Larsson, International Conference on Software Engineering Advances, ICSEA'06, IEEE, Tahiti, French Polynesia, October, 2006 [15]
- "Experience Report: Using Internal CMMI Appraisals to Institutionalize Software Development Performance Improvement", Fredrik Ekdahl, Stig Larsson, 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06), p 216-223, IEEE Computer Society, Cavtat, Croatia, September, 2006 [16]

- “Selecting CMMI Appraisal Classes Based on Maturity and Openness”, Stig Larsson, Fredrik Ekdahl, PROFES 2004 - 5th International Conference on Product Focused Software Process Improvement, Springer-Verlag Berlin Heidelberg New York, Kansai Science City, Japan, Editor(s):Frank Bromarius, Hajimu Iida, April, 2004 [17]

Chapter 2. Product Integration

This chapter describes product integration, beginning with a general discussion about different interpretations of the nature of product integration. With this background, the problems found in product integration, as used in this thesis, are described. In addition, we discuss the use of reference models, as well as other concepts in software engineering related to and affecting product integration processes.

2.1 Interpretation of "Product Integration"

The terms "product integration", "systems integration" and "software system integration" are used for several different aspects in product and system development literature. Grady claims that *integration* is one of the most misunderstood concepts within systems engineering [18]. Djavanshir and Khorramshahgol [19] have investigated the importance of different process areas related to system integration and observe that professionals in the field relate integration to many areas of systems engineering. This indicates that there is no clear definition of integration when discussing system and software engineering. It is consequently necessary to clearly define the scope of integration, and to be aware of other interpretations of the term. Sage and Lynch provides an overview in [20], and Land elaborates on different meanings of the terms in [21]. The main uses of the terms are:

- Product integration processes:
  This term describes the process used in product development projects when parts are combined into more complex parts and eventually into the product or system to be delivered to the customer. It includes the activities ensuring that the combinations of components are functioning as intended and the management of interfaces between components and between the product and the environment. As earlier described, this is the focus for this thesis.

- Architectural, or technical, product or system integration:
  This concerns the technical solutions used to fulfill requirements on functionality and quality attributes such as reliability and performance. Different levels of integration include export and import facilities, the use of wrappers and adapters, integration through shared databases, and integration on source code level. Interface design is one important issue for all levels of architectural integration, and standard interfaces are available for many applications. Different types of architectural integration is described by Nilsson et al in [22]. Other examples of the use of integration in this meaning is found in [23] where Garlan describes trends in software architecture research, and in [24] where Gorton describes useful architectural practices .
- Enterprise Application Integration (EAI)
  EAI is a specific type of architectural integration where organizations combine and integrate existing and new systems to assist the organization in achieving business objectives. This type of integration is performed to ensure data consistency and to make information accessible to different types of stakeholders, often based on the use of a common middleware. Examples of descriptions of EIA are [25] by Cummins, [26] by Linthicum, and [27] by Ruh et al.
- Software system in-house integration:
  When merging systems with similar purposes, there are both process, architectural, and technical considerations to be managed. This has been described by Land in [21].
- Integrated product and process development:
  The integration of product and process development aims at having a focus on collaboration between all stakeholders in the product development. An emphasis is put on a common vision which is key to fulfill and exceed customer satisfaction. This includes all different disciplines needed to work together in a common effort, often as one project, throughout the project life-cycle. The development processes proceed in an integrated project in parallel, which requires tight cooperation between the participants. The use of integrated product and process development is included in the CMMI [3], and has for example been described by Parsaei et al in [28]

2.2 Product Integration Problems in the Industry

To understand what needs to be improved in descriptions and implementation of product integration processes, it is important to understand what types of problems are found in industry.

Problems in product integration have been described by Ramamoorthy in [29]. According to that study the software system integration problem includes several issues:

- Inconsistencies in the interfaces between modules in the system lead to problems at integration time. The inconsistencies result from the different assumptions made by engineers in earlier phases of the development
- Insufficient use of strategies and planning for the integration effort. This leads to unnecessary dependencies in the product integration for the different modules, and to increased need for interactions between designers to synchronize deliveries
- Insufficient understanding of the dependency structure of the product or system leads to cumbersome debugging and fault finding at integration time

Through the case studies performed in this research project, we have been able to observe a more detailed view of the problems and the following types have been found:

- Related to architecture and design
  - Architectural decisions are done without considering the full system, leading to problems at integration time
  - Changes are made to interfaces without proper control. This leads to errors in the builds or initial integration testing
  - Changes in common resources (e.g. common include files) are not controlled. This results in errors appearing in other components which have not been changed
  - New functions are added and errors are corrected without proper investigation of consequences. The result may be new errors that influence the functionality and performance of the system more than the original problem
  - Errors appear in other components which have not been changed due to changes in interfaces, i.e. changes are made in how two

components interact, while also other components are using this interface

- Related to the inadequate establishment or use of the integration environment
  - Problems appear as tests for the components are not run in the same type of environment as the integration test system. Different versions of hardware and test platform are used
  - The build environment is not prepared for new builds, e.g. results from earlier builds are not removed before a new generation of the system is started
  - Untested changes are introduced in the integration environment e.g. build scripts are changed without proper verification
- Related to inadequate delivery of functions
  - Inconsistent code, i.e. functions that have only been partly implemented, is delivered for integration. Files are not included in the build as planned, resulting in failed builds
  - Functions are not always delivered in time for integration or may be incompletely delivered. This leads to problems in the build process or in integration and system tests
  - Functions are not always fully tested when delivered for integration. This leads to problems in the build process or in integration and system tests

The types of problems are not independent. An example of this is that inadequate coordination of when different components are to be delivered may lead to pressure to deliver components without proper preparation or testing. If no agreed criteria have been defined, it will be even easier to accept this behavior.

To summarize, the problems are in essence related to interaction and planning for interaction, both between different development teams and between the components that are to constitute the final product.

Through the studies performed in industry we have seen that the investigated systems all have some type of legacy. This is an additional factor that creates limitations for how to perform product integration. The legacy can be an inherited code base, connections to other systems such as tools that require certain components to ensure backward compatibility, or standards that require specific behavior from the system. The result of this is

that the product integration depends on a large number of earlier decisions and resulting strategies. In turn, the consequence of this is reduced freedom to select strategy for the integration, and may lead to needs for refactoring or other changes to the architecture before a new strategy can be selected.

2.3 Applying Reference Models

The reference models used in our research describe and propose different activities that should help in achieving efficient and effective product integration.

Two types of reference material, standards and models, have been considered in this study and are referred to as reference models[1]. Product integration is treated in different ways in the reference models; in some models such as ISO/IEC 12207 [2] and CMMI [3], the subject is handled in a specific part, while in others such as EIA-632 [30], the description of product integration is found in different sections. In most reference models, the product (or system) integration is considered to result in aggregations of components into bigger components. The product integration is repeated over the project life-cycle until the product or system is available and can be delivered to the customer.

The activities that are considered part of product integration can be divided into three areas: preparation, management of interfaces, and execution of the product integration.

Careful *preparation* is in the reference models described as the key to efficient and effective product integration. It includes defining a strategy based on business needs and targets, and organizing the integration sequence to be in accordance with the strategy and synchronized with other project and organization activities. An environment for the integration should be prepared, and requirements on the components to be delivered to integration defined.

Many system and product integration problems occur due to incomplete or misunderstood interfaces. Therefore *management of interfaces*, i.e. the identification and definition of what interfaces should be managed, need to

[1] The difference between the types is that standards have been approved by a standardization body, while a model may be issued by any company or organization.

be a part of the product integration. However, the design and implementation of interfaces should be considered part of the architectural and detailed design. The target for the management of interfaces is to ensure compatibility. This means that the practices needed is (i) to review the interfaces for completeness, and (ii) to manage relationships between interfaces and components to ensure that any changes to interfaces are dealt with in affected components.

When using the reference models as a basis for implementing management of interfaces, it is important that the concept of interfaces is clearly understood. Interfaces are not only the syntactical description of the connection point to a software component. An example of how this is captured in the reference models can be seen in ISO/IEC 15288 [5] where section 5.5.4.3 includes the following

> ”g) Define and document the interfaces between system elements and at the system boundary with external systems
>
> Note Definitions are made with a level of detail and control appropriate to the creation, use and evolution of the system entity and with interface documentation from parties responsible for external interfacing entities. …”

This specific standard covers all types of engineering, and the statement needs to be complemented with more details of what should be considered as a part of interfaces to be practically useful when developing software intensive systems. In addition, each organization needs to determine what is needed. One view of how to regard software interfaces is given by Parnas in [31]. He describes how interfaces need to comprise *the set of assumptions that the developers of the different components can make about other components*, e.g. the behavior in normal and error situations, resource needs, and the need for other components.

The *execution* takes advantage of the preparations, and includes checks that the criteria for when components can be delivered are fulfilled, that the components are delivered as planned, and the integration including evaluation and test of the assembled components.

A detailed description of where information about the three areas *preparation*, *management of interfaces*, and *execution* can be found for different reference models is available in paper D [13].

In [32], Stavridou investigates integration standards for critical software intensive systems. The examination focuses on military policies and standards, but includes also ISO/IEC 12207 in the comparison. The

conclusion is that the majority of the examined standards address integration testing, but that the standardization is not appropriate for many integration issues. The descriptions of the included activities are insufficient as support for a project manager running a product development project. An additional conclusion is that the integration activities should be considered as a separate phase of system development.

Incorrect use of reference models and software development models is described by Fitzgerald [33]. The reason for not using the models as intended is claimed to be the perceived lack of contribution to successful product development, and inflexibility in the models, not allowing for customization to specific organizational and project needs. This has also been highlighted by Bajec et al in [8]. They describe and prescribe a method for adapting the development model to the specific project. This should of course also include the product integration part of the process.

One reason for the industry to be slow in adopting useful practices from reference models may be their format and their content. Reference models in general cover projects and organizations with a large range of attributes; projects with significant differences in size, distribution, complexity and novelty should be covered in the same models. This means that the models often describe *what* should be performed, but not necessarily *how*. The interpretation of a specific method or practice becomes important, and the insufficient knowledge in how to implement the practices may prevent the organizations from adhering to a model.

The extent to which the models describe how a practice should be implemented differs. However, none of the models used in our research are explicit and give detailed advice on how the models can be used for different types of projects and organizations. Most detailed is the CMMI [3] which describes subpractices and expected work products, while standards such as ISO/IEC 12207 [2] give only high level direction.

2.4 Product Integration and Related Software Engineering Concepts

Several areas related to product integration have been identified in literature. That the areas are related to product integration has been confirmed in the examination of the organizations and projects that form the basis for the research presented in this thesis.

Two basic topics are how the selection of the project and software lifecycle influences the product integration, and the effect architectural decisions have on product integration. Other areas such as distributed development and the use of the software product line concept are phenomena that make the product integration more complex.

In this section, a set of these software engineering concepts are discussed from the perspective of product integration. The selection has been made based on literature search and investigations of areas covered in major software engineering conferences.

2.4.1 Project Life-cycle Models

McConnel stresses the importance of selecting the life-cycle model that is appropriate for a specific type of development and provides a selection guide in [34]. The selection of the project life-cycle model also determines what options the project will have from which to select integration strategy. In [35], Pressman differentiates between three types of models: the waterfall model, incremental models, and evolutionary models. Each of these types has an influence on how the product integration processes can be implemented.

The *waterfall model* requires that each phase of the development is concluded before the next is started. A strict use of this model will force the project to begin the integration when all components are ready and apply a big-bang approach. However, the strategies and sequences for integration can be selected on the basis of the needs from the organization if the schedule permits. This includes incrementally integrating components based on architectural or other considerations even though all components are available. There is a risk that errors found in integration requires the project to modify components or interfaces which will delay the project. Modifications to the model permit overlapping phases, enabling the organization to select integration sequence by giving priority to which components should be ready first. Also, by applying the model separately on different components and subsystem, a more flexible integration process can be implemented. The waterfall model used in this way resembles an incremental model.

Using *incremental models* increases the number of possible strategies for product integration. The selection of integration strategy of may determine what should be developed in each increment. Considerations for the project planning with regards to increments will resemble the considerations for the integration selection. Examples of different strategies are to provide the

basic functionality in the first increment, and making more advanced versions of each feature available as the project proceeds with further increments, or to develop the most important feature with all functionality available for the first increment. One important aspect for the product integration is that a strategy and integration sequence is also needed within an increment, e.g. a specific order is needed to be able to perform the integration tests.

Evolutionary models include two major types: spiral models and evolutionary prototyping. Spiral models have been described by Boehm in [36] and further developed in [37], and focus on minimizing risk by starting the project in small scale, addressing the major risks. The project then iterates a number of steps, including setting a target for the iteration, identifying risks, evaluating alternatives, developing deliverables as described for the iteration and evaluating the results, planning the next iteration and deciding on an approach for the next iteration. The integration process will in most evolutionary projects differ between the iterations based on the approach and purpose of the specific iteration. This also gives the organization and project the option to adapt the product integration as the project proceeds.

Evolutionary prototyping is also iterative, and focuses on aspects of the product that can be evaluated by the customer (or a representative for a customer base). Quick design and implementation lead to early feedback which can be used for refining the requirements. Later versions of the product will be designed with more focus on architecture and quality. Here, the product integration processes will be very important, especially for the handling of interfaces. Early prototypes tend to be built on existing components that may have to be replaced in a later iteration, and the management of interfaces and changes to these is crucial.

Using evolutionary models put higher demands on the project as the focus is on minimizing risks and as the processes are often adapted as the project progresses.

Ramamoorthy presents a proposal how to tackle the challenges in product integration in [29], and relates the activities to different project life-cycle phases. The proposal resembles the activities described in reference models, and give additional views on what can be used as design guidelines when implementing product integration processes. The proposal includes a *preliminary development phase* which incorporates specification of interfaces, establishment of an integration strategy, the implementation of an

integration environment, establishment of criteria for integration, and the development of an integration plan. The second phase is labeled *initial integration* and includes primarily management tasks. Examples of activities are regular feedback on status, improvement of the strategy, and monitoring the integration process. The *final product integration* phase is briefly described. Similar to the reference models, no validation of the proposed method or the included activities is presented.

The activities in the product integration area have also been the subject of interest from the agile community where continuous integration is common and frequent builds is one of the cornerstones. One example is [38] where Fowler describes the requirements on developers: before committing components back to the mainline the developer would need to update his work area with the latest mainline, i.e. build against the latest changes of other developers. Only after that, integration into the mainline would be permitted. The use of continuous integration and frequent builds is one of the strategies that can be selected for product integration, and will also put requirements on other activities such as the preparation of an integration environment.

2.4.2 Architecture and Product Integration

Architecture and design are connected to the product integration processes in several ways. The interface design affects the possibilities to select different integration strategies, while the chosen integration strategy may influence and limit the architectural options available. That management of interfaces is an important aspect of many of the architectural tactics as described by Bass et al in [39].

Sage and Lynch provides a general description of system and product integration in [20] and describes a view of how the integration can affected by architecture. The conclusion is that developing an appropriate architecture for a system will simplify the integration later in the project's lifecycle. It is also stated that the architecture can be the means for communication and knowledge transfer in a project. This is further described by Ovaska et al in [40]. The main idea in the description is that a common understanding of the software architecture between the software development parties will improve the coordination of different teams. They also stress the need for both informal communication and formal descriptions of interfaces.

Eppinger describes in [41] a method to reduced the problems in integration using an architectural and design structure matrix approach. The method

includes three steps: decomposition, identification of interactions between the components based on different types of interaction, and clustering of components based on the analysis of the structure of interactions. The method is closely related to the management of interfaces as described in product integration.

Another area that has been well researched is how software can be reused. One example in the context of architecture and refactoring has been described by Metha and Heinemann in [42] where an evolution model is proposed and a methodology that finds code that can be refactored into components is described. Chioch et al [43] reports on experiences where the process determine the acceptance for an architecture intended for reuse.

The challenge of integrating large systems is discussed by Schulte in [44], who proposes methods for modeling system behavior to handle the uncertainties in resulting system characteristics when integrating components. According to Schulte, three areas need to evolve to provide capabilities powerful enough to assist when modeling real-time systems. These are multi-view modeling, analysis and code generation. Another example of using models to ensure efficient and effective integration has been presented by Karsai et al [45]. The point made is that modeling should be made the central activity when developing systems.

The focus of the research presented here is on embedded industrial systems with specific requirements on different quality attributes such as timeliness, reliability, and availability. This is reflected in the need for specific approaches both regarding the view of computation as described by Lee in [46], and in the fact that in these systems, physical properties are modeled and appear as cross-cutting constraints for the whole system as described by Sztipanovits and Karsai in [47]. To solve these needs and requirements, appropriate architectural solutions will be necessary.

Also with respect to the binding time[2], there are requirements that will influence the product integration. Svanberg et al describes the concept of binding time in the context of product lines in [48]. A distinction is made between pre-delivery binding time which includes product architecture derivation, compiling and linking, and post-delivery binding time which can be at start-up, during runtime, or per call. One characteristic of embedded industrial systems is that most bindings are performed pre-delivery, and that

[2] Binding time is the moment when a decision is made for a possible variation in the product.

binding at start-up primarily is performed through configuration. Binding per call, during run-time, is rarely used in this kind of systems.

2.4.3 Software Product Lines

Software product line engineering is a technique used to utilize a common set of core assets in the development and preparation of a series of products. This concept requires a different approach than traditional software product development. Core asset development and their utilization to build products need planning, and require efforts. This includes strategies that encompass several products, management that enforces the development and utilization of common assets, and technologies, methods and processes adapted to software product line development. Bass et al observes that the use of product lines will replace design and coding with integration and testing as the predominant activities in [39].

The software product line concept is described in detail by Clements and Northrop in [49], and by Pohl et al in [50]. SEI provides additional information, references and examples on the "Software Product Lines Home Page" [51].

SEI describes the particular aspects of system and product integration in [52]. One specific topic which differs from One-off product development is the pre-integration that is made on the core assets. This pre-integration has two purposes. The first is a verification to ensure that components that are part of the core assets can be integrated as intended. The second purpose is to prepare larger components that can be reused. This is done to reduce the effort when instantiating products.

Some recent research describes additional advancements. In [53], Krueger describes three new methods which can increase the usefulness of software product lines. These are "Software Mass Customization", "Minimally Invasive Transitions", and "Bounded Combinatorics". The first method can affect the product integration process and to a large degree reduce the effort for integration. It builds on the concept that a software product line (SPL) configurator uses predefined product definitions to create product instances. Besides reducing the need for application development, the recreation of products when changing core assets can be automated. Using an SPL configurator also changes the organizational needs: the development will be performed on the core assets that will contain all software necessary for all the product instantiations. However, there is still a need to exercise the practices for product integration when developing and verifying the core assets. Additional activities include decisions on variation points, and

verification strategies for these. The third method, "Bounded Combinatorics", reduces the variations and resembles the pre-integration technique described by SEI in [52].

2.4.4 Distributed Development

Distribution of development efforts in a project increases the need for monitoring communication between the participants in the project, and also with other stakeholders. The general considerations has for example been examined by Herbsleb and Mockus in [54], and Paulish et al in [55]. Conclusions from these studies show that distributed development takes more time and requires more effort that single-site development. The investigations also provide guidance on how to minimize the negative effects, emphasizing communication on all aspects of the development. Paulish et al note that the architectural work primarily was performed in face-to-face meetings and workshops, focusing on specific topics in each meeting. Interface design and communication regarding content of builds were considered very difficult.

This is also described by Vand den Bulte and Moenaert in [56] where they show that organizational boundaries, and especially physical distance, may hinder communication between distributed development teams, especially for technical know-how.

Sosa et al combine the perspectives of product architecture and organizational structure in [57], and investigate the communication patterns in organizations based on interfaces described in the product architecture. The three main findings are that misalignment of interfaces is greater across organizational and system boundaries, that indirect interaction is important to achieve coordination, and that modularization may hinder alignment of interfaces and interactions. All three findings support the need for careful management of interfaces which is one of the main themes of product integration.

Komi-Sirivö and Tihinen present an investigation into the factors which determines the success or otherwise of distributed development in [58] and lists interfaces as being the most important source of software errors after misinterpreted, changing, or missing requirements. This highlights the importance of interface management in distributed environments.

Product integration is affected by distribution of development efforts as the management of dependencies both between development activities and between parts of the system becomes more cumbersome. This underlines the

importance of the three areas covered in reference models for product integration: 1) preparation, to get a common vision and agree on plans, environments, etc, 2) interface management, to ensure that information that affects components developed in different locations is communicated in an efficient and effective way, and 3) execution, monitoring the cooperation as the project proceeds.

2.4.5 Component Based Software Engineering and Development

Component based software engineering may be one tool in improving the engineering practices and simplify product integration practices. However, there are indications described by Crnkovic in [59] that changes are needed in the established development and life cycle models. The differences between component-based development and non-component based development require the use of new patterns, and a distinction between development *of* components and development *with* components. The product integration process is one area in which we can anticipate changes in current practices as the use of general-purpose components for product development of embedded systems is increasing. There are several definitions of a software component is in this context, and in this section we use the focused definition by Heineman and Councill found in [60]:

> "A *software component* is a software element that conforms to a component model and can be independently deployed and composed with modification according to a composition standard.
>
> A *component model* defines specific interaction and composition standards. A *component model implementation* is the dedicated set of executable software elements required to support the execution of components that conform to the model.
>
> A *software component infrastructure* is a set of interacting software components designed to ensure that a software system or subsystem constructed using those components and interfaces will satisfy clearly defined performance specifications."

There are several approaches to architecting and implementing component based development (CBD). Dogru and Tanik describe in [61] a fully component-oriented approach and contrasts this with modifying object oriented approaches, stressing that CBD takes no account of inheritance and capitalizes on composition. Van Ommering describes in [62] a component model that is used as the basis for development of product families in a

distributed environment. One interesting part of this description is how the process and organization have been aligned with the new way of developing products. This example describes specific changes in the organization; the division into an asset team, handling the basic system, and product teams that build the applications on top of the system and integrates the final product. The conclusion arrived at, with respect to process, was the increased importance of the role of the "quality officers" ensuring that the standards for the specific development methods were followed.

The differences between the development of component based systems and non-component based systems are described in [63]. The separation of component development and development of systems based on components is highlighted and this description gives input to how integration can be organized. Morisio et al [64] also describe the difference in the development of components and system in the context of COTS (commercial off-the-self) components. The investigation included fifteen projects, and the integration was, for many of the investigated projects, the activity that consumed most effort. A solution by means of which decisions regarding requirements and candidate components are made together is proposed. The method also includes early analysis of integration issues in the design phase.

The importance of following and performing all process steps is described by Tran et al in [65]. Their investigation shows an increased risk of failure if a project omits any of the defined steps: identification, selection, evaluation, procurement, integration, and evaluation of software components.

de Jonge finds that the goals of reusing building blocks and the goals of integration are difficult to combine, but proposes techniques of how this can be done [66]. These include the concept of source code components and source tree composition that integrates source files, build and configuration processes.

One area that is related is the use of generic component architectures. In [67], Lichota et al describes a generic component architecture and proposes a process for selecting and integrating software products as components. This process is described as five steps:

- **Identification:**
  Determination if a candidate component can be considered to be included in the product.
- **Screening:**
  Components to be further investigated are selected on the basis of a

review of all available information about the components selected in the identification phase.

- **Stand-Alone Test:**
  Each component is tested to determine if it fulfills the expectations described in the documentation. In addition, the components should be tested to determine its potential reliability, reusability, and general applicability to component requirements.
- **Integration Test:**
  The integration test is performed to understand how effectively the component can be integrated into the selected component architecture. Components that are found suitable are candidates for inclusion in a library of reusable components.
- **Field Test:**
  To finally determine its usefulness the component should be tested in a user environment. This will show how effectively the component fulfills user and inter-operability requirements.

The described process was used for large components in the case described in [67], but can of course be used also on more granular components.

Crnkovic et al have described the development with component as being three separate but coordinated processes in [15]. These are

- **System development,**
  in which components are combined into specific products and systems based on existing or new components,
- **Component assessment,**
  which includes activities to select components that can be a part of a component repository or being selected from a repository for a specific system and product,
- **Component development,**
  which describes the activities to develop independent components and ensuring that they are made available to the intended user of the component.

On the basis of the references above and the reference models investigated, we conclude that the practices described in the models will also support component-based product development. We also see that there is a need to add specific requirements through the description of more detailed activities that can be useful, in addition to the ones currently described.

An example of how this can be done for one reference model, CMMI, is found in Table 1. The goals for CMMI related to Product Integration match the extent of the product integration process in other reference models. The additions are needed to handle the consequences of separate processes for system development, component assessment, and component development. Major parts include handling of the infrastructure for a component model, availability, and suitability of components, and interdependencies between components. The effort for performing the described activities will increase the cost for the development of systems, but the reuse of existing components in combination with higher quality of components when delivered to integration should counter and outweigh this.

2.5 Conclusion

My conclusion is that the different aspects of software engineering, such as project life-cycle models, software architecture, and the organization of the projects, must be considered and taken into account when performing product integration. For all these aspects, a careful management of the interfaces and interaction between both components in the system and the participants in the development projects is vital for success. I conclude also that the descriptions available today in different reference models are insufficiently used and additional effort is needed to make them useful.

Table 1. Proposed changes to Product Integration process in the CMMI

Specific Goal 1: Prepare for Product Integration
System development • Consider component availability when determining the integration sequence • Ensure that the chosen component model is supported by the infrastructure
Component assessment • Investigate component interdependencies
Component development • Ensure that the anticipated infrastructure is specified, i.e. component model and framework • Describe tests and expected results as a part of component specifications
SG 2: Ensure Interface Compatibility
System development • Include checks that the chosen interfaces adhere to the overall architectural decision on patterns and strategies
Component assessment • Check the consistency of interfaces
Component development • Adhering to the component model helps ensure that the definition of interfaces is complete • Check that all functions, including built-in-test facilities, can be accessed, i.e. that the defined interfaces permit the intended functionality
SG 3: Assemble Product Component and Deliver the Product
System development • (No additions proposed)
Component assessment • Prepare assemblies of components at assessment time to ensure that the components fit the system
Component development • Assemble test systems to show suitability for different applications • Test verification procedures that are a part of the component delivery • Make components available in an internal repository, or on the market

Chapter 3. Research Method

This chapter includes an overview of the research methods used in software engineering and how these are used in the research presented. Each of the papers included in the thesis contains the method applied in that part of the research as well as a discussion about validity, and limitations of the studies. The general research strategy and the overall validity are discussed here.

3.1 Method

Software engineering research uses a number of methods to ensure progress in the area. Basili has presented four approaches [68]: 1) the scientific method, 2) the engineering method, 3) the empirical method, and 4) the analytical method. The three first are classified as being part of the scientific paradigm, while the fourth is the analytical paradigm. Understanding these different methods also help distinguish research from development. Research aims at understanding a phenomenon, e.g. why and how the use of a process can help a product development organization improve performance, while development is performed to implement, e.g. to describe, and train people in the use of a process.

Software engineering research is in many respects different from other types of computer science research, and mathematics, as it heavily depends on human behavior through the people developing the software products and systems. This is described by Wohlin et al in [69] and is especially true in the research regarding processes. This makes it difficult to use the analytical paradigm.

In [70], Shaw describes different aspects of research in the area of software engineering. One of her conclusions is that initial research may result in informal and qualitative results, which give incentives for continued research. As the research in an area matures, more empirical models are presented, and finally result in formal models which justify larger

investments to introduce the research outcome on a larger scale. These different steps require different methods as the expected results differ.

One way to distinguish between different types of results from research is based on the type of research that has produced them. This has been expressed for human computer interaction by Brooks and adapted for software engineering by Shaw in [70]. It is necessary to distinguish between different types of results because research results based on experiments that are possible to control, and can show statistical results, are limited in scope, while broader results that are based on observations are more difficult to validate. It is necessary to know the background to be able to understand the implications of the presented research.

The proposed classification of research results includes Findings, Observations and Rules-of-thumb. Findings are the results from soundly-designed research, and with clear declaration of the domain for which a generalization is valid. Observations report on actual phenomena that are interesting, but may be from under-controlled environments and/or observations from limited samples. Finally Rules-of-Thumb are generalizations over a domain that is larger than the tested one. All three types of results should be judged for freshness, and it should be clear for all reports to what type the results belong. There is also a need for all three types; Observations and Rules-of-Thumb will give guidance to practitioners and help generate basis for further research that eventually could lead to Findings.

In [71], Redwine and Riddle describe different phases in software engineering research from the aspect of maturation of software technology. These are *basic research*, *concept formulation*, *development and extension*, *internal enhancement and exploration, external enhancement and exploration*, and *popularization*. Each of these phases requires different methods and tools, and will also bring the knowledge area forward in different ways. The *basic research* is used to investigate basic concepts, and to formulate basic research questions in the area. *Concept formulation* comprises the forming of a research community, and a convergence of different ideas. Solutions to specific problems are also published. The next phase, *development and extension*, includes making preliminary use of ideas and concepts and aim at a generalization of solutions and approaches. *Internal enhancement* refines the solutions and broadens the use to other domains, and the research should in this phase start to show value as it can be used to solve real problems. *External enhancement* brings the technology to other people that have not been involved in the development of the

concepts and the use of the research results shows it substantial value. Finally, the *popularization* includes a full embracement of the technology, with commercialization and an expanded user community as a result.

A substantial set of ideas and guidelines is available for product integration, but the methods are not validated. In order to understand the problems and how different activities, tools, and methods can help in achieving more efficient and effective product integration, it is necessary to have empirical data as a basis. The data in this research has been collected in an industrial environment with case studies involving project developing commercial products, which leads to an under-controlled environment. The results are therefore *observations* [70]. We have chosen to work in the *development and extension* phase as described in [71]. Our aim is thus to make use of existing ideas to determine if any generalizations can be made, and to clarify the ideas underlying what is described in the reference models.

The method used in the research presented consists of three steps. The first two steps have been made in iterations, and step three is the final analysis:

(i) Examination of existing standards and reference models that includes practices for product integration;

(ii) Based on knowledge from the reference models, case studies have been performed to obtain an understanding of the connection between the use of practices and problems found in product integration;

(iii) Analysis of the combination of the results from the case studies and the content in the reference models.

The case studies are planned and executed based on methods described by Yin [72] and Robson [73]. This includes the preparation and the implementation of the studies through interviews and document reviews, and the analysis based on the observations. The three iterations that have been performed are shown in Figure 2.

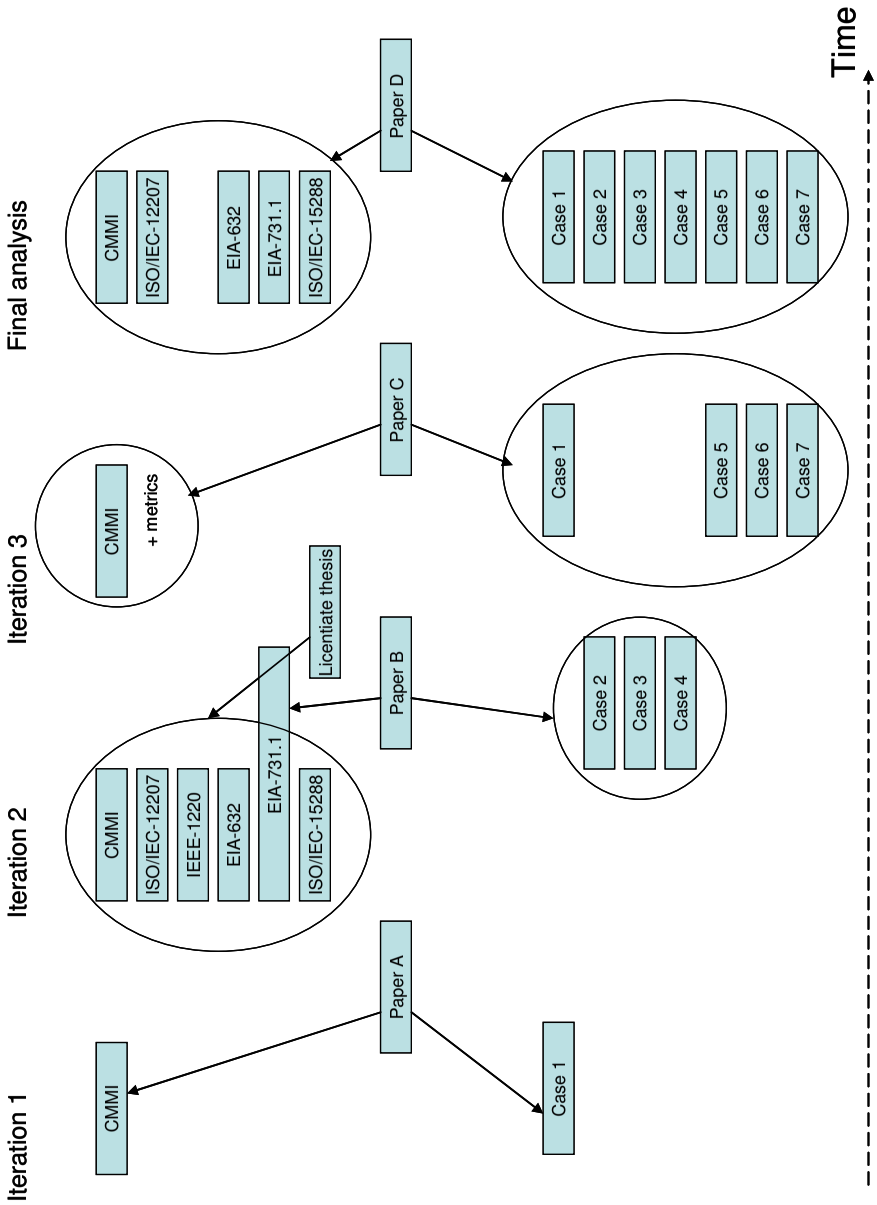

Figure 2 Research iterations

The first iteration included an investigation of CMMI [3], and a case study of one product development organization and is presented in paper A [10]. The case study included two parts; an investigation of the practices used in the company based on CMMI, and an identification of the problems found in product integration. The data for the case study was collected through interviews, document reviews, and reviews of the organizations' process documents.

The second iteration began with a study to find additional suitable reference models. A set of models containing requirements or directions for product integration were selected. The study to find suitable reference models was based on information from standardization organizations such as ISO [74], ANSI [75], and IEEE [76] and organizations such as SEI [77] and INCOSE [78]. The criterion for selecting a model was that the reference model should be relevant to product development of products that include substantial part software.

That a reference model fulfilled the criterion was determined by its purpose as in the model documentation. The result of the investigation is described in my licentiate thesis [9]. The selected reference models were:

- ISO/IEC 12207 Information technology - Software life cycle processes [2]
- EIA-632 Processes for Engineering a System [30]
- CMMI Capability Maturity Model Integration [3]
- EIA-731.1 Systems Engineering Capability Model [1]
- ISO/IEC 15288 Systems Engineering – System life cycle processes [5]
- IEEE Std 1220 for Application and Management of the Systems Engineering Process [79]

Also ISO 9001 [80] was initially considered, but was rejected since the expectations on the product integration process are limited. The standard describes the requirements on design and development, and the general requirements such as planning, input, output, review, verification, validation, and control of design and development changes are all applicable to product integration. However, as the expectations on the product integration process are not mentioned explicitly, this standard has not been analyzed further.

From among the selected reference models, EIA-731.1 [1] was chosen for use in the second case study which included three development groups in two organizations and is described in paper B [11]. The data was collected in the same way as for the first case study; through interviews, document reviews, and review of process documents.

Before the third iteration of cases was initiated, the selected set of reference models was thoroughly investigated. The analysis was performed through careful examination of the standards, and a compilation of a union of what was included in the reference models was developed.

The third and final iteration included a case study with four development organizations, and is described in paper C [12]. In addition to the data collection based on a reference model (CMMI), data was also collected from the build activities. The purpose was to see if it is possible to combine the use of a reference models with existing data from the activities in the organization. The data was collected was through interviews, document reviews, and through the collection of data made by the practitioners (e.g. build failure frequency), either automatically or manually.

Based on the three iterations of step one and two of our research method, the findings from all case studies and the investigation of reference models were analyzed as step three. The results are available in paper D [13]. The analysis was for each of the case studies performed, based on available reference models. Five reference models were selected as they had explicit expectations on product or system integration. This was an additional criterion compared to iteration 2 of the research. One reference model was excluded at this point, IEEE Std 1220 [79], as most of the references to product integration in this standard were implicit and not useful for our purpose.

All original material from the cases was used, but no additional information was collected. The problems found through the case investigations as well as the implementation of practices were mapped to all reference models. One factor in this phase was that the material for each case was collected on the basis of only one of the reference models. Through the use of the notes and reference documents collected, it was possible to determine if the practices were implemented for most cases. The only exception was for the practice related to the product strategy. Information was missing in three of the cases, and it was impossible to draw conclusions if that practice was performed or not.

There is an important distinction between insufficient support in the reference models, and the unsatisfactory implementation of good practices by the product development organizations. The difference has guided us in our research; the focus of the case studies being on insufficient use. The compilation based on all cases studies includes a discussion about the indications of insufficient support for product integration working well in the reference models.

As an additional part of the final step of our research, the relationship between architecture and the product integration process has been investigated. As architectures are developed and evolve, the implementation of the product integration process may be affected. There are several related subjects: i) interface management is an important part of software product architectures and product integration, ii) the division of subsystems and components performed in the architectural development influences the possibilities to select different integration strategies and sequences, iii) and the product integration strategy may introduce constraints on the architectural design.

This part of the research focuses on the evolution of the architecture for a system and has proposed and piloted a method to understand what changes to processes are necessary to achieve the business goals that are the reason for an architectural change. The results are presented in paper E [14]. The method was based on existing methods for assessing architectures and processes, primarily ATAM [81] and CMMI [3, 82]. The investigation in the pilot study was performed as a participant-observer study with two researchers participating in the use of the method. After the pilot study, the method was evaluated based on two criteria as defined before the study.

3.2 Validity and Limitations

Four types for validity based on Robson [73], Yin [72], and Wohlin et al [69]; construct validity, internal validity, external validity (or generalizability), and reliability (or conclusion validity) have been considered in this thesis.

The ***construct validity*** relates to the data collected and how this data represent the phenomenon investigated. This is addressed in the case studies through multiple sources for the data in the project appraisals. This is accomplished through more than one interviewee for each case as well as using document reviews. The use of reference models as a basis for the

interviews and document reviews secure that the data collected is relevant. A concern here is that we have used different reference models for the different cases. However, the collected data also includes information about other practices than those available in the reference model used for the data collection. It has been possible to use this additional data in the comparison with all reference models. If no data is available in the case material for a specific practice in any of the models this is presented in the research results. One specific problem that has been observed in the interviews is that even if an interviewee responds that a practice is performed, we have found that the activities for that practice may actually not be performed. This is treated through corroboration of data through several interviews and document reviews.

The ***internal validity*** concerns the connection between the observations and the proposed explanation for these observations. This has been addressed in several ways. For the appraisals using reference models, several steps have been taken to ensure that the mapping and understanding are correct. A detailed description of the methods used for the appraisal can be found in [16]. One risk related to the internal validity is that, through the investigations and through participation in the discussions of product integration, we affect the processes while collecting data. The results that we collect are however a combination of the performed practices and the problems occurring in the organization. Thus, the data we collect reflects the state in the organizations at the time for the data collection. The results are valid, and useful for our purposes, even if they might have been different if we had not influenced the organization through the investigation. For the case studies that have been performed in the company where I am working, there is the risk that an internal researcher would get different answers than an external. This can go both ways: persons responding to questions may be more open to an external researcher, than to an internal or vice versa. This has not been investigated specifically. One advantage in the case studies performed inside the company is that I have better background knowledge and can understand the responses and ask better clarification questions. Access to different projects has been easier through the internal case studies, and this is probably also an advantage.

The possibilities to generalize the results from a study are dealt with by studying the ***external validity.*** This is addressed through the selection of cases from different domains including telecommunication, power protection and control, process automation, and industrial robot control. The investigations cover primarily embedded systems, but workstation software

products have also been included. In these applications, the workstations are a part of a larger industrial system, typically as operator or engineering stations. The focus is on industrial applications as we see that the requirements and expectations differ from those associated with consumer products, ERP systems, and banking applications. We have also limited the research to products that are delivered to more than one customer, i.e. the cases we have investigated do not include any bespoke development. Additional aspects that have been considered to address the external validity are to ensure that the case studies include different countries and different types of organizations. One disadvantage is that several cases are from the same multinational company. However, the investigated organizations are from different divisions, have distinctly different development processes, and the products are intended for different application domains. The result is that a broad spectrum of different types of products and organizations has been investigated.

High ***reliability*** increases the possibilities to reach the same conclusions as those of another researcher repeating the study. The reliability aspect of the studies has been addressed through the detailed description of the procedure used in each case and have been included as a part of the publication for each case study. Additionally, the method for collecting data from organizations and projects use techniques described in [16].

3.3 Conclusion

In this chapter, I have described how the research presented in this thesis is based on current knowledge, theories and guidelines for software engineering research. I have also described how the planning and execution of the research as well as the selection of case studies contribute to the different aspects of validity. The conclusion of the discussions is that the validity for this research includes industrial software products, intended for use by more than one customer.

Chapter 4. Research Results

This chapter summarizes the research results and relates the research questions with the individual papers included in this thesis.

The main question for our research is to understand *what factors influence the possibilities to achieve efficient and effective product integration.* Efficient and effective product integration is manifested through a minimum delay of the flow of components to larger components or products and systems.

To investigate the factors, we have used different types of reference models. We have examined what effect the use of, or failure to use, the practices described in the reference models have on the performance in product integration. This was performed by investigating product development organizations and through examining development projects. We have further examined how changes in architecture can influence processes, and how this influence can be captured. The relationship between the areas we have investigated and the research papers A-E can be seen in Figure 3.

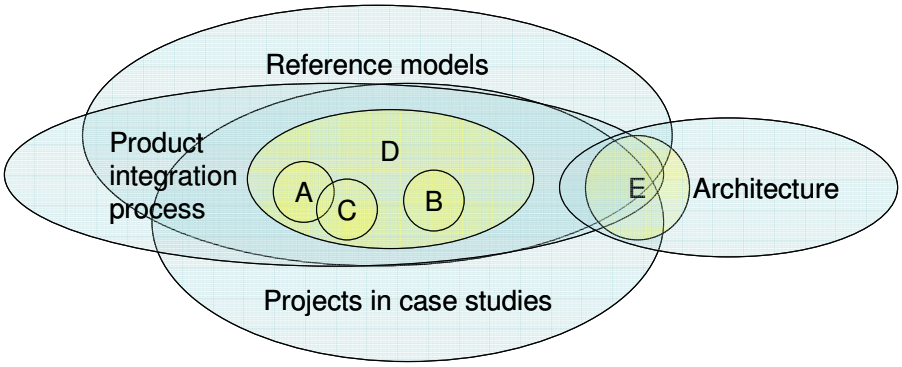

Figure 3. Relationship between the research papers A-E and the investigated areas

4.1 Results related to questions 1a and 1b

Papers A, B and C present the investigation of product integration processes in product development projects based on different reference models. Paper A and C share one of the case studies. Paper D summarizes and expands on the findings from paper A, B and C. Finally, paper E is the first step in a new research direction based on our findings intended to determine additional influences and considerations that need to be taken into account when defining and improving product development processes in general, and specifically product integration processes.

The research questions presented in section 1.2 make our research more concrete, and the response to them is based on papers included in this thesis.

The first question is used to investigate the use of current reference models:

> *Are the practices described in available reference models for product integration necessary and sufficient for visible reduction of problems in the product integration process?* **(Q1a)**

Our investigations (papers A-C) [10-12] compare the performed activities in different organizations with practices described in different reference models. The problems related to product integration have also been captured. The problems in the case studies are associated with practices, which gives us an understanding of what practices can actually help avoid product integration problems. The case studies from seven development organizations in the three papers A-C give at hand that the types of difficulties encountered in product integration can be reduced through following the practices described, but the specific practices in each of the reference models are not sufficient. In particular the different reference models cover different aspects of product integration and a parallel investigation has been directed by the question:

> *What additions and modifications are needed in the available reference models to take advantage of current body of knowledge in product integration?* **(Q1b)**

The answer to this question is a combination of an analysis of the selected reference models, and a compilation of the cases in papers A - C. The results of these steps are presented in detail in paper D [13], and are summarized here. The reference model analysis resulted in a union consisting of 15 practices which describes what can be considered the current level of knowledge in product integration.

Of the 15 practices, four are concerned with preparation of the product integration:

1. Define and document an integration strategy
2. Develop a product integration plan based on the strategy
3. Define and establish an environment for integration
4. Define criteria for delivery of components

The following five practices describe design and interface management

5. Identify constraints from the integration strategy on design
6. Define interfaces
7. Review interface descriptions for completeness
8. Ensure coordination of interface changes
9. Review adherence to defined interfaces

One practice defines the preparation of the verification to be performed in the product integration:

10. Develop and document a set of tests for each requirement of the assembled components

The actual integration of components is made up of four practices:

11. Verify completeness of components obtained for integration through checking criteria for delivery
12. Deliver/obtain components as agreed in the schedule
13. Integrate/assemble components as planned
14. Evaluate/test the assembled components

Finally, a single practice ensures that the integration is documented:

15. Record the integration information in an appropriate repository

Note that this division of practices is more detailed than is common in the reference models that have been used in this research, and may be seen as addressing the responsibility for different roles in the organization. The preparation is the responsibility of the project manager with assistance of the product integrator. The second part, interfaces, is an architectural task, involving architects and developers. The test preparation as well as the actual integration is the product integrator's responsibility, while again the project manager with the assistance of all the product integration participants will be responsible for recording the results.

As a second step to answer research question Q1b, the union of practices from the different reference models has also been compared to the different problems found in the case studies to clarify which practices will directly reduce the number and the effects of problems in product integration. A detailed description of this analysis is available in paper D [13]. The results are summarized in Figure 4 and show the product integration problems that can be related to a practice in each reference model. Our conclusion is that none of the standards include all necessary practices needed to help the organizations in avoiding the problems. As an example, for ISO/IEC 15288 we could associate 11 of the 17 problems found in the case studies to any one of the product integration practices in that standard. The result confirms the need of a broader approach than is available in any of the examined reference models. Using a combination of the examined reference models as we propose will cover activities and procedures that address all the problems encountered in our case studies. Further analysis is presented in Table 2. This analysis shows that a combination of CMMI and either ISO/IEC 15288:2002 or EIA-733.1 would be sufficient to include all needed practices.

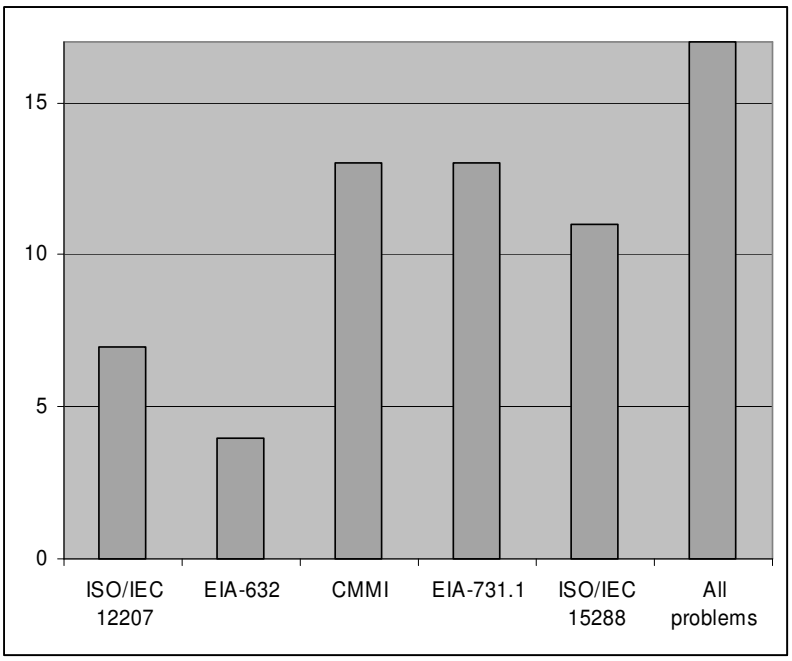

Figure 4. # of unique problems related to the practices in each standard and the total # of unique problems for all cases

Table 2. Problems from cases related to reference models

Problem identity	ISO/IEC 12207	EIA-632	CMMI	EIA-731.1	ISO/IEC 15288
Case 1, problem A	Covered			Covered	Covered
Case 1, problem B	Covered	Covered	Covered	Covered	Covered
Case 1, problem C			Covered	Covered	
Case 2, problem A			Covered		
Case 2, problem B				Covered	Covered
Case 2, problem C			Covered	Covered	
Case 3, problem A	Covered	Covered	Covered	Covered	Covered
Case 4, problem A			Covered	Covered	
Case 5, problem A	Covered			Covered	Covered
Case 5, problem B			Covered		Covered
Case 6, problem A	Covered	Covered	Covered	Covered	Covered
Case 6, problem B	Covered			Covered	Covered
Case 6, problem C			Covered		Covered
Case 6, problem D			Covered	Covered	
Case 7, problem A	Covered	Covered	Covered	Covered	Covered
Case 7, problem B				Covered	Covered
Case 7, problem C			Covered	Covered	

Of the 15 Product Integration practices, we have observed that problems are likely if any of the following five are neglected:

4, Define criteria for delivery of components

7, Review interface descriptions for completeness

8, Ensure coordination of interface changes

11, Verify completeness of components obtained for integration through checking criteria for delivery

12, Deliver/obtain components as agreed in the schedule

For PI practice 1, “Define and document an integration strategy”, and PI Practice 3, “Define and establish an environment for integration” we have seen that there may be problems even if the practices are performed.

For eight of the practices, we have not seen any problems:

2, Develop a product integration plan based on the strategy

5, Identify constraints from the integration strategy on design

6, Define interfaces

9, Review adherence to defined interfaces

10, Develop and document a set of tests for each requirement of the assembled components

13, Integrate/assemble components as planned

14, Evaluate/test the assembled components

15, Record the integration information in an appropriate repository

One important additional factor when determining what practices need to be performed is the dependencies between them. Some of the practices are necessary as a preparation for others, i.e. support the necessary practices while additional practices may be more indirectly connected.

Through reasoning about the different practices we have identified the dependencies, and the result of this analysis is presented in Figure 5.

We claim that for the interface handling, also PI practice 6 “Define interfaces” is important as PI practice 7 “Review interface descriptions for completeness” relies on it. A weaker dependency is also identified between PI practice 6 and PI practice 5 “Identify constraints from the integration strategy on design”. The same reasoning can be applied on PI practice 2 “Develop an integration plan based on the strategy” which is recognized as a prerequisite for PI 12. PI practice 11 “Verify completeness of components obtained for integration through checking criteria for delivery” depends on the checks done through PI practice 9 which is “Review adherence to defined interfaces”. Finally, PI practice 1 “Define and document an integration strategy” can be depending on PI practice 15 “Record the integration information in an appropriate repository” as the collected data is important when deciding on changes and improvements in the strategy for product integration.

A conclusion is that the set of practices that need to be followed is larger than the set that we have seen causes problems in the development organizations. The additional practices that support the crucial ones are PI

practices 2 “Develop a product integration plan based on the strategy”, 6 “Define interfaces”, 9 “Review adherence to defined interfaces”, 15 **“**Record the integration information in an appropriate repository”, and indirectly also PI practice 5 “Identify constraints from the integration strategy on design” as PI practice 6 is depending on it. Note that the dependency that PI practice 15 has on all other practices has been omitted. To be able to record the results from the PI activities so that this information can be used for future improvement, information from all activities should be included.

The remaining three practices that are not connected to or supporting the crucial practices are PI practices 10, 13, and 14. PI practice 10, “Develop and document a set of tests for each requirement of the assembled components”, is likely to give problems through later discovery of errors, and resulting problems are not connected to the product integration. Our interpretation is that this could explain why it is not connected. The same reasoning is applied to PI practice 13. “Integrate/assemble components as planned”, and PI practice 14, “Evaluate/test the assembled components”. These practices include activities that are performed if integration is performed. Failure in the practices would be that the defined sequences and procedures are not followed which would give problems in alter phases. Problems found executing PI practice 13. “Integrate/assemble components as planned”, and PI practice 14, “Evaluate/test the assembled components” are normally related to the preparation of product integration or the environment.

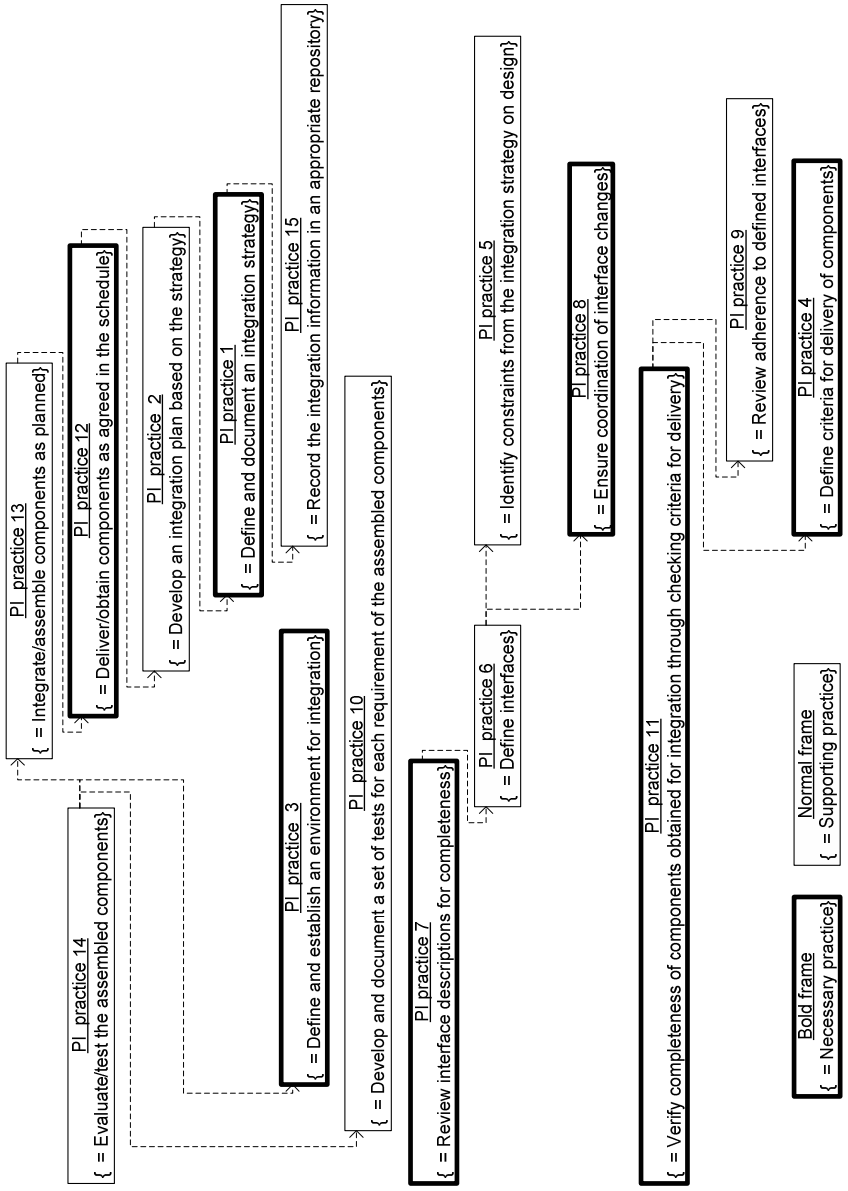

Figure 5 Dependencies between PI practices.

For PI practice 1 and 3, we need to understand how the problems occur even if our investigation indicates that the practices are followed. For PI practice 1, “Define and document an integration strategy”, we have observed that problems were encountered even when this practice is followed. This may be explained by the strategy selected being les suitable for the environment of that project. PI practice 1 sets the stage for the integration process, and even if a strategy is chosen, it may not fit the specific conditions under which a project is run. However, we have not been able to determine what the root cause was, as the rationale for the selection of the strategy has not been documented. In this context, the difference between deciding on an integration strategy and the preparation of an integration plan is important.

When selecting a strategy for the product integration in a project or for an organization, several different factors will determine what will be the best technique for the sequence. Among important factors are:

- Risk elimination
- Customer needs for functionality
- Availability of components/resources(people/technology/tools)
- Technical infrastructure
- Dependencies between components (anatomy)
- Testing possibilities

McConnel describes different strategies in [4]. The first observation described is that the incremental strategies are superior to the phased (also called “big bang” integration). The techniques mentioned for the incremental integration are top-down integration, bottom-up integration, sandwich integration, risk-oriented integration, feature-oriented integration, and T-shaped integration. It is also pointed out that these techniques are to be tailored to fit each specific project, and not to be dogmatically followed – in the end, a project need to have its own integration strategy with the resulting integration sequence. One difficult issue is to measure how well this practice is followed. One proposal is to use checklists as the one presented in [4].

PI practice 3, “Define and establish an environment for integration”, disclosed problems during the appraisals for two cases even when the practice was considered to be performed. A closer investigation into these cases show that the effectiveness with which the practice is performed was not considered, and can be classified as a false positive from the appraisal. It highlights however an important point: the use of practices need to be

verified by information about the performance, and an analysis of the connection between the actual performed practices and the measured results is key to improving the product integration process.

4.2 Results related to question 2

It was observed in the case studies that the architecture of a product or system is very often changed without the processes to further develop the system being altered to reflect this evolvement. One practice described in ISO/IEC 15288 is related to this and is included as PI practice 5 of the union: “Identify constraints from the integration strategy on design”.

To ensure that architectural changes are reflected in the process and strategies for product integration, tools are needed that can provide support to find the needed changes. As many organizations today start introducing changes by redefining or evolving the architecture, there is a need to understand and map the influence on process from architectural decisions. This leads to an additional research question and corresponding answers:

> *How can necessary changes in the integration process due to changes in the product architecture be identified and implemented?* **(Q2)**

Through an investigation of different models used for supporting architectural decisions, and appraisal methods for process improvement, a method has been proposed and piloted (paper E [14]). The method was successful in helping the organization to understand what process changes are needed to benefit from the architectural changes. This was especially true for the product integration process as the architectural changes called for new strategies. This fact was not identified by all parts of the organization before the use of the proposed method. The proposed method has also been updated based on the results from the study. An additional result from the investigation is that the understanding for needed process changes, including steps in product integration, has increased in the pilot organization.

4.3 Conclusion

From our case studies I conclude that the available knowledge regarding the product integration process is inconsistently used by different product development organizations.

It can be questioned if the use of the PI Practices is valid for all types of projects and organizations. As seen in the analysis above, all the practices described are either crucial in them selves, supporting the crucial ones, giving problems later in the process, or dependent on other practices and thus not indicating any problems. My conclusion is that all practices are needed, but that an adaptation is necessary to get the proper level of activities in each project. A future research area can be to investigate the need for a method for the adaptation.

When investigating the product integration area, we have seen that organizations are aware of practices that are described in reference models. However, as the information in the models is too limited, the usefulness is also limited and additional information such as examples and hands-on methods are needed. Consequently, the models should primarily be used as guidelines for what to improve, and information about how the practices should be implemented need to be found elsewhere.

I also conclude the product integration processes may be influenced by evolvement of the architecture and design of software systems, and that the method provided, the needed changes of processes can be understood and implemented.

Chapter 5. Conclusions and Future Work

The goal of the research presented in this thesis has been to understand what factors are most important when trying to achieve efficient and effective product integration, and how these factors influence the software product development processes.

The origin of the research is in the needs identified in organizations developing products for industrial use with significant software content. These needs have been confirmed in case studies. Organizations experience problems in product integration due to insufficient and inconsistent strategies and plans for integration, lack of understanding of how interface management and other architectural decisions influence product integration, and inadequate control of components delivered for integration. The focus in the research is on industrial software products, with real-time requirements. This implies specific needs to understand and be able to manage quality attributes, such as performance, reliability and availability of the resulting product.

I have through investigations of information available in reference models regarding product integration practices and a series of case studies identified the key elements for software product integration practices. These have been organized in five categories: preparation of the product integration, design and interface management, preparation of the verification to be performed, the execution of product integration, and documentation of the product integration results. The collection includes the practices available today in relevant reference models, which have been made accessible through the compilation. The validity of the practices has been examined through case studies. The work to reach an agreed body-of-knowledge for software product integration processes should be continued. This can be done through relevant research in other application domains, and reference models applicable for these domains.

Table 3. Collection of Product Integration Practices

Preparation of product integration	
Define and document an integration strategy	Necessary
Develop a product integration plan based on the strategy	Supporting
Define and establish an environment for integration	Necessary
Define criteria for delivery of components	Necessary
Design and interface management for product integration	
Identify constraints from the integration strategy on design	Supporting
Define interfaces	Supporting
Review interface descriptions for completeness	Necessary
Ensure coordination of interface changes	Necessary
Review adherence to defined interfaces	Supporting
Preparation of product integration verification	
Develop and document a set of tests for each requirement of the assembled components	Problems likely related to other practices or processes
Integration of components	
Verify completeness of components obtained for integration through checking criteria for delivery	Necessary
Deliver/obtain components as agreed in the schedule	Necessary
Integrate/assemble components as planned	Problems likely related to other practices or processes
Evaluate/test the assembled components	Problems likely related to other practices or processes
Documentation of the integration	
Record the integration information in an appropriate repository	Supporting

The collection of practices that I have described provides support for software product development organizations. The collection is summarized in Table 3. Of the 15 practices, there are indications in our case studies that five are necessary, as shown in the rightmost column in Table 3, to avoid problems in product integration. Two practices have been seen to be necessary, but that problems arise if the practices are inadequately performed. An additional five practices support the practices considered necessary. For the remaining three practices, there are no indications that organizations will have problems if not implementing them. However, the nature of these three practices is such that any problems would most likely be related to other practices such as the preparation of the integration or integration environment, or verification.

The influence that architectural decisions have on product development processes is seldom investigated in the industry. Through a case study, we have demonstrated the usefulness of a method to examine and identify changes necessary to be made to development processes. When evolving the architecture of the product in a case study, the processes influenced include the product integration process. By providing a method to understand how different changes affect the processes, proposed improvements for better product integration can be understood and assessed.

The research presented here has been performed based on available theories and guidelines for research in software engineering, and care has been taken to address different types of validity. This is done through careful planning and execution of the studies, and through selecting relevant case study organizations working with software products in industrial applications. The conclusion is that the validity for this research includes industrial software products, intended for use by more than one customer.

Future research includes additional validation of the collection of practices, also in other application domains. A subject which needs to be investigated is implementation of proposed practices, to understand why available practices are not used, and why the implementation sometimes fails.

The impact the presented research will have when applied in industry remains to be seen and additional investigations are needed to explore this. Each organization using the practices described in this thesis needs to implement the practices, adapted to the organizations needs. Further investigations are needed to understand how an impact can be achieved with reasonable effort.

Additional research is also needed to look at other methods, tools, and technologies to help product development organizations improve product integration. Through the compilation of practices based on the available reference models and an understanding how these can help, a foundation is available for future research. This can also be the starting point to investigate different types of project and development models to understand if there are specific requirements that should be taken into account.

References

[1] EIA-731.1, "Systems Engineering Capability Model," Electronic Industries Alliance, 2002.
[2] ISO/IEC12207:1995, "Information technology - Software life cycle processes," ISO/IEC, 1995.
[3] SEI, "CMMI® for Development, Version 1.2.," Pittsburgh, PA, USA, Technical Report CMU/SEI-2006-TR-008, 2006.
[4] S. McConnel, Code Complete, 2nd ed. Redmond, Wa, USA: Microsoft Press, 2004.
[5] ISO/IEC15288:2002, "Systems engineering - Systems life cycle processes," ISO/IEC, 2002.
[6] J. Campanella, Principles of Quality Costs: Principles, implementation and Use, 3rd ed. Milwaukee, WN, USA,: ASQ Press, 1999.
[7] RTI, "The Economic Impacts of Inadequate Infrastructure for Software Testing." Gaithersburg, MD, USA,: National Institute of Standards and Technology, 2002.
[8] M. Bajec, D. Vavpoti, and M. Krisper, "Practice-driven approach for creating project-specific software development methods," Information and Software Technology, vol. 49, pp. 345, 2007.
[9] S. Larsson, "Improving software product integration." Västerås: Dept. of Computer Science and Electronics Mälardalen University, 2005, pp. xi, 108.
[10] S. Larsson, I. Crnkovic, and F. Ekdahl, "On the expected synergies between component-based software engineering and best practices in product integration," presented at Proceedings - 30th EUROMICRO Conference, Aug 31-Sep 3 2004, Rennes, France, 2004.
[11] S. Larsson and I. Crnkovic, "Case Study: Software Product Integration Practices," presented at 6th international conference Profes, June, 2005, Oulu Finland, 2005.
[12] S. Larsson, P. Myllyperkiö, and F. Ekdahl, "Product Integration Improvement Based on Analysis of Build Statistics," presented at ESEC/FSE, Dubrovnik, Croatia, 2007.

[13] S. Larsson, P. Myllyperkiö, F. Ekdahl, and I. Crnkovic, "Examination of Product Integration Practices in Reference Models," Submitted to Information & Software Technology, 2007.
[14] S. Larsson, A. Wall, and P. Wallin, "Assessing the Influence on Processes when Evolving the Software Architecture," presented at IWPSE 2007, Dubrovnik, Croatia, 2007.
[15] I. Crnkovic, M. Chaudron, and S. Larsson, "Component-Based Development Process and Component Lifecycle," presented at Software Engineering Advances, International Conference on, 2006.
[16] F. Ekdahl and S. Larsson, "Experience Report: Using Internal CMMI Appraisals to Institutionalize Software Development Performance Improvement," presented at 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 06), 2006.
[17] S. Larsson and F. Ekdahl, "Selecting CMMI Appraisal Classes Based on Maturity and Openness," presented at Product Focused Software Process Improvement (PROFES), Kansai Science City, Japan, 2004.
[18] J. O. Grady, System Integration: CRC press, 1994.
[19] G. R. Djavanshir and R. Khorramshahgol, "Key Process Areas in Systems Integration," in IT Professional, vol. 9, 2007, pp. 24-27.
[20] A. P. Sage, Charles L. Lynch, "Systems integration and architecting: An overview of principles, practices, and perspectives," Systems Engineering, vol. 1, pp. 176-227, 1998.
[21] R. Land, "Software Systems In-House integration," in Department of Computer Science and Electronics: Mälardalen University, 2006.
[22] E. G. Nilsson, E. K. Nordhagen, and G. Oftedal, "Aspects of systems integration," presented at Systems Integration, 1990. Systems Integration '90., Proceedings of the First International Conference on, 1990.
[23] D. Garlan, "Software architecture: a roadmap," presented at Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000.
[24] I. Gorton, Essential Software Architecture: Springer, 2006.
[25] F. A. Cummins, Enterprise Integration: An Architecture for Enterprise Application and Systems Integration: John Wiley & Sons, 2002.
[26] D. S. Linthicum, Enterprise Application Integration: Addison-Wesley, 1999.
[27] W. A. Ruh, F. X. Maginnis, and W. J. Brown, Enterprise Application Integration: Addison-Wesley, 2000.

[28] H. R. Parsaei, J. M. Usher, and U. Roy, Integrated Product and Process Development: Methods, Tools, and Technologies: Wiley-IEEE, 1998.
[29] C. V. Ramamoorthy, "Distributed techniques in software system integration," presented at Workshop on Future Trends of Distributed Computing Systems, Cheju Island, Korea, 1995.
[30] ANSI/EIA-632-1999, "Processes for Engineering a System." Government Electronic and Information Technology Association: Electronic Industries Alliance, 1999.
[31] D. L. Parnas, "Information distribution aspects of design methodology," presented at Information Processing 71 Proceedings of the IFIP Congress 1971 Volume 1, 23-28 Aug. 1971, Ljubljana, Yugoslavia, 1972.
[32] V. Stavridou, "Integration standards for critical software intensive systems," presented at Software Engineering Standards Symposium and Forum, 1997. 'Emerging International Standards'. ISESS 97., Third IEEE International, 1997.
[33] B. Fitzgerald, "An empirical investigation into the adoption of systems development methodologies," Information & Management, vol. 34, pp. 317-328, 1998.
[34] S. McConnel, Rapid development. Redmon, Wa, USA: Microsoft press, 1996.
[35] R. S. Pressman, Software Engineering, 6th ed: McGraw-Hill, 2005.
[36] B. W. Boehm, "A spiral model of software development and enhancement," Computer, vol. 21, pp. 61-72, 1988.
[37] B. W. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, "Using the WinWin spiral model: a case study," Computer, vol. 31, pp. 33-44, 1998.
[38] M. Fowler, "Continuous Integration, http://www.martinfowler.com/articles/continuousIntegration.html," 2006.
[39] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice: Addison-Wesley Professional, 2003.
[40] P. Ovaska, M. Rossi, and P. Marttiin, "Architecture as a coordination tool in multi-site software development," Software Process: Improvement and Practice, vol. 8, pp. 233-247, 2003.
[41] S. D. Eppinger, "A planning method for integration of large-scale engineering systems," presented at International Conference on Engineering Design, ICED, Tampere, Finland, 1997.

[42] A. Mehta and G. T. Heineman, "Evolving legacy system features into fine-grained components," presented at Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on, 2002.
[43] F. A. Cioch, J. M. Brabbs, and L. Sieh, "The impact of software architecture reuse on development processes and standards," The Journal of Systems and Software, vol. 50, pp. 221-236, 2000.
[44] M. Schulte, "Model-based integration of reusable component-based avionics systems - a case study," pp. 62-71, 2005.
[45] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," Proceedings of the IEEE, vol. 91, pp. 145-164, 2003.
[46] E. A. Lee, "Embedded Software," in Advances in Computers, vol. 56, M. Zelkowitz, Ed.: Elsevier, 2002.
[47] J. Sztipanovits and G. Karsai, "Embedded Software: Challenges and Opportunities," in Embedded Software: First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October, 8-10, 2001, Proceedings, 2001, pp. 403.
[48] M. Svahnberg, J. V. Gurp, and J. Bosch, "On the Notion of Variability in Software Product Lines," presented at Working IEEE/IFIP Conference on Software Architecture (WICSA'01), Amsterdam, The Netherlands, 2001.
[49] P. Clements and L. Northrop, Software product lines: practices and patterns, 2002.
[50] K. Pohl, G. Böckle, and F. J. v. d. Linden, Software Product Line Engineering: Foundations, Principles and Techniques: Springer, 2005.
[51] SEI, "Software Product Lines," http://www.sei.cmu.edu/productlines/index.html, 2007.
[52] SEI, "A Framework for Software Product Line Practice, Version 5.0," in Software System Integration. http://www.sei.cmu/productlines, 2007.
[53] C. W. Krueger, "New methods in software product line practice," Commun. ACM, vol. 49, pp. 37-40, 2006.
[54] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," IEEE Transactions on Software Engineering, vol. 29, pp. 481, 2003.
[55] D. J. Paulish, M. Bass, and J. D. Herbsleb, "Global software development at siemens: experience from nine projects," presented at Software Engineering, 2005. ICSE '05. Proceedings of the 27th International Conference on, 2005.

[56] C. Van den Bulte and R. K. Moenaert, "The effects of R&D team co-location on communication patterns among R&D, marketing, and manufacturing," Management Science, vol. 44, pp. S1-S18, 1998.
[57] M. E. Sosa, S. D. Eppinger, and C. M. Rowles, "The Misalignment of Product Architecture and Organizational Structure in Complex Product Development," Management Science, vol. 50, pp. 1674-1689, 2004.
[58] S. Komi-Sirvio and M. Tihinen, "Lessons learned by participants of distributed software development," Knowledge and Process Management, vol. 12, pp. 108-122, 2005.
[59] I. Crnkovic, "Component-based software engineering - new challenges in software development," Software Focus, vol. 2, pp. 127-133, 2001.
[60] G. T. Heineman and W. T. Councill, Component-based Software Engineering, Putting the Pieces Together: Prentice-Hall, 2001.
[61] A. H. Dogru and M. M. Tanik, "A process model for component-oriented software engineering," IEEE Software, vol. 20, pp. 34-41, 2003.
[62] R. van Ommering, "Building product populations with software components," presented at Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on, 2002.
[63] I. Crnkovic, S. Larsson, and M. Chaudron, "Component-based development process and component lifecycle," presented at 27th International Conference on Information Technology Interfaces, 2005., 2005.
[64] M. Morisio, C. B. Seaman, V. R. Basili, A. T. Parra, S. E. Kraft, and S. E. Condon, "COTS-based software development: Processes and open issues," Journal of Systems and Software, vol. 61, pp. 189-199, 2002.
[65] V. Tran, L. Dar-Biau, and B. Hummel, "Component-based systems development: challenges and lessons learned," presented at Eighth IEEE International Workshop on Incorporating Computer Aided Software Engineering, 1997.
[66] M. de Jonge, "Package-based software development," presented at Euromicro Conference, 2003. Proceedings. 29th, 2003.
[67] R. W. Lichota, R. L. Vesprini, and B. Swanson, "PRISM Product Examination Process for component based development," presented at Assessment of Software Tools and Technologies, 1997., Proceedings Fifth International Symposium on, 1997.

[68] V. R. Basili, "The Experimental Paradigm in Software Engineering," 2000.
[69] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in software engineering; an introduction. Norwell, Massachusetts, U.S.A.: Kluwer Academic Publishers, 1999.
[70] M. Shaw, "What makes good research in software engineering?," STTT - International Journal on Software Tools for Technology Transfer, vol. 4, pp. 1-7, 2002.
[71] S. T. Redwine and W. E. Riddle, "Software technology maturation," in Proceedings of the 8th international conference on Software engineering. London, England: IEEE Computer Society Press, 1985, pp. 189-200.
[72] R. K. Yin, Case Study Research: Design and Methods, 3rd ed: Sage Publications, 2003.
[73] C. Robson, Real World Research, 2nd ed: Blackwell Publishers, 2002.
[74] ISO, "International Standardization Organization," http://www.iso.org, 2007.
[75] ANSI, "American National Standards Institute," http://www.ansi.org/, 2007.
[76] IEEE, "The Institute of Electrical and Electronics Engineers," http://www.ieee.org/, 2007.
[77] SEI, "Software Engineering Institute," http://www.sei.cmu.edu/, 2007.
[78] INCOSE, "International Counsil on Systems Engineering," http://www.incose.org/, 2007.
[79] IEEE1220-2005, "IEEE Standard for Application and Management of the Systems Engineering Process," Institute of Electrical and Electronics Engineers, 2005.
[80] ISO9001:2000, "Quality management systems -- Requirements," ISO, 2000.
[81] R. Kazman, M. Klein, and P. Clements., "ATAM: Method for architecture evaluation. CMU/SEI-2000-TR-004," Carnegie Mellon University, Software Engineering Institute, Technical Report CMU/SEI-2000-TR-004, 2000.
[82] SEI, "Standard CMMI® Appraisal Method for Process Improvement (SCAMPISM) A, Version 1.2: Method Definition Document," Carnegie Mellon University, Software Engineering Institute, Handbook CMU/SEI-2006-HB-002, 2006.

Part 2

Paper A

ON THE EXPECTED SYNERGIES BETWEEN COMPONENT-BASED SOFTWARE ENGINEERING AND BEST PRACTICES IN PRODUCT INTEGRATION

Stig Larsson, Ivica Crnkovic, Fredrik Ekdahl
Presented at the Euromicro Conference,
Rennes, France, August 2004

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 1.

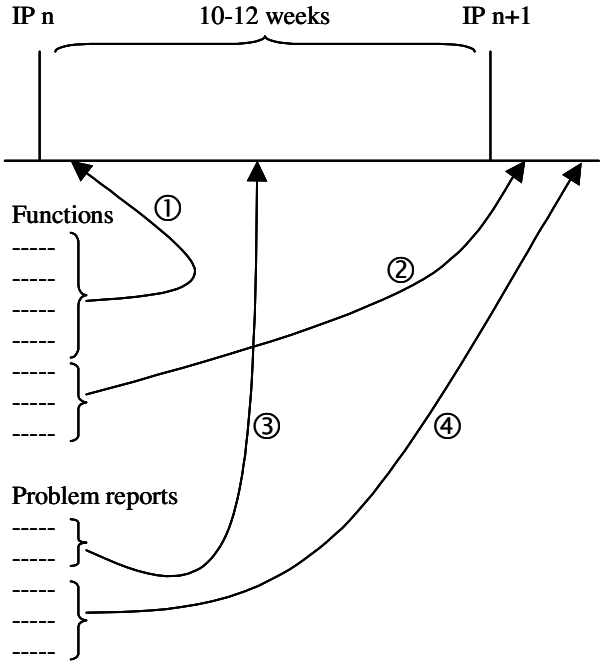

Figure 1. Distribution of functions and error corrections

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 2. 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).

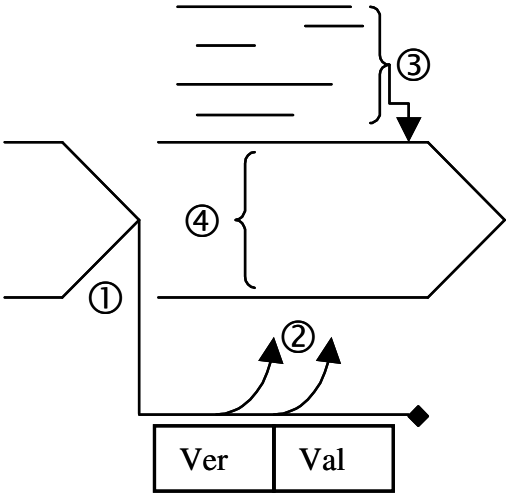

Figure 2. Integration point activities

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. (Link valid April 2005.)

[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., A. Rashid, “A strategy for Managing Risk in Component-based Software Development”, Euromicro 2001 CBSE workshop, Warsaw, Poland, 2001.

Paper B

CASE STUDY:
SOFTWARE PRODUCT INTEGRATION PRACTICES

Stig Larsson, Ivica Crnkovic
Presented at the PROFES 2005 Conference,
Oulu, Finland, June 2005

Abstract

Organizations often encounter problems in the Product Integration process. The difficulties include finding errors at integration related to mismatch between the different components and problems in other parts of the system than the one that was changed. The question is if these problems can be decreased if the awareness of the integration process is increased in other activities. To get better understanding of this problem we have analyzed the integration process in two product development organizations. One of the organizations has two different groups with slightly different integration routines while the other is basing the development on well defined components. The obstacles found in product integration are highlighted and related to best practices as described in the interim standard EIA-731.1. Our conclusion from this study is that the current descriptions for best practices in product integration are available in standards and models, but are insufficiently used and can be supported by technology to be accepted and utilized by the product developers.

1. Introduction

Through investigations of many development organizations developing products with software as an important part, we have seen that the product integration is one of the processes where many of the problems in product development become visible. The origin of the problems is often in other processes performed early in the development cycle. These problems can be reduced through an increased understanding of the needs from an integration standpoint. Today, not enough care is taken to ensure that the system requirements are considered when components and parts developed. Proper preparation, understanding and performance of the product integration are believed to resolve part of this problem.

Integration of products that include software is described in several standards and collections of best practices. These best practices are collected from different companies and organization and include areas that are considered to be of good use for the development organizations in different application areas. There is however a lack of independent research which shows whether the practices described in these collections give the intended result when implemented in different organizations; a systematic validation of the practices is needed.

There are different perspectives from which the use of descriptions found in standards and models can be investigated and different questions to be answered. The first question is how it can be determined that the processes described in the standards and models are suitable for different types of development and the use of different life cycle models; are the generic principles of the descriptions valid for all types of product development? Another question is if an organization may run into problems even if the principles and descriptions are followed in a proper way. Are there ways to fulfill the principles described but not achieve the intended results? A third question is how to determine if the reason for an organization having problems is the fact that the principles are described as the prescribed working method, but are still not followed. Our approach to these different perspectives is to look at the performance of the process in the investigated organizations and compare the activities with the ones prescribed in the standards and models regardless of the development model used. We also look at the problems in the organizations and analyze these with respect to the practices that are not followed by the organization.

We claim that we by investigating a number of organizations and the practices in use can obtain support for the practices described in standards and models or determine a need for revisions of the standards and models. This leads to the following research questions for this paper: (i) How well can the practices described in a specific standard be expected to reduce problems encountered in the integration of products? and (ii) What deficiencies or incompleteness can we observe in the proposed practice?

We have in this paper selected to use the interim standard EIA-731.1 [1] as the reference model. The rational for this is that the interim standard model has been used as one of the inputs to CMMI [2], and is specifically intended to be used for internal process improvement, not for qualification of suppliers. In addition to this, the development of this interim standard has been carried out in cooperation between a number of national and international organizations such as EIA[3] and INCOSE [4] involving a

large number of organizations and companies with substantial experience in software and system product development.

Our proposition in this paper is that the problems encountered in the investigated units relate to the lack of execution of practices that are described in the interim standard. We also propose that successful execution of the product integration can be mapped to specific implementation of practices described in the interim standard.

This case study is a continuation of the work described in [5], where a different case has been compared to CMMI. The purpose of this paper is to investigate one additional source for best practices, compare it to current industrial problems and to establish if there are connections between the problems and the lack of execution of proposed activities.

The remainder of the paper is organized as follows. Section two describes general structure of the interim standard EIA-731.1 as well as the main characteristics of the integration processes of a development process. In section three, the case study design is described with explanations about the data collection method, the analysis method and the threats of validity of the study. Section four includes a description of the findings from the case study. Section five analyzes how the findings relate to best practices. Finally section six contains the conclusion and proposed future work and is followed by the references list.

2. Product Integration in EIA-731.1

The interim standard EIA-731.1 describes a number of focus areas useful for organizations developing products and systems. The focus areas described are organized in three categories; technical, management and environment. For each focus area, a number of themes describe the suggested activities. All themes include a description, typical work products and specific practices for the focus area. For some of the focus areas there are comments that normally contain clarifications or suggested implementation details. In addition to the specific practices, there are a number of generic practices applicable for all specific practices with the different focus areas. The generic practices include tasks such as planning of the activities to perform the process, monitoring and checking that the activities performed are according to plan and the execution of corrective measures when these are identified and needed.

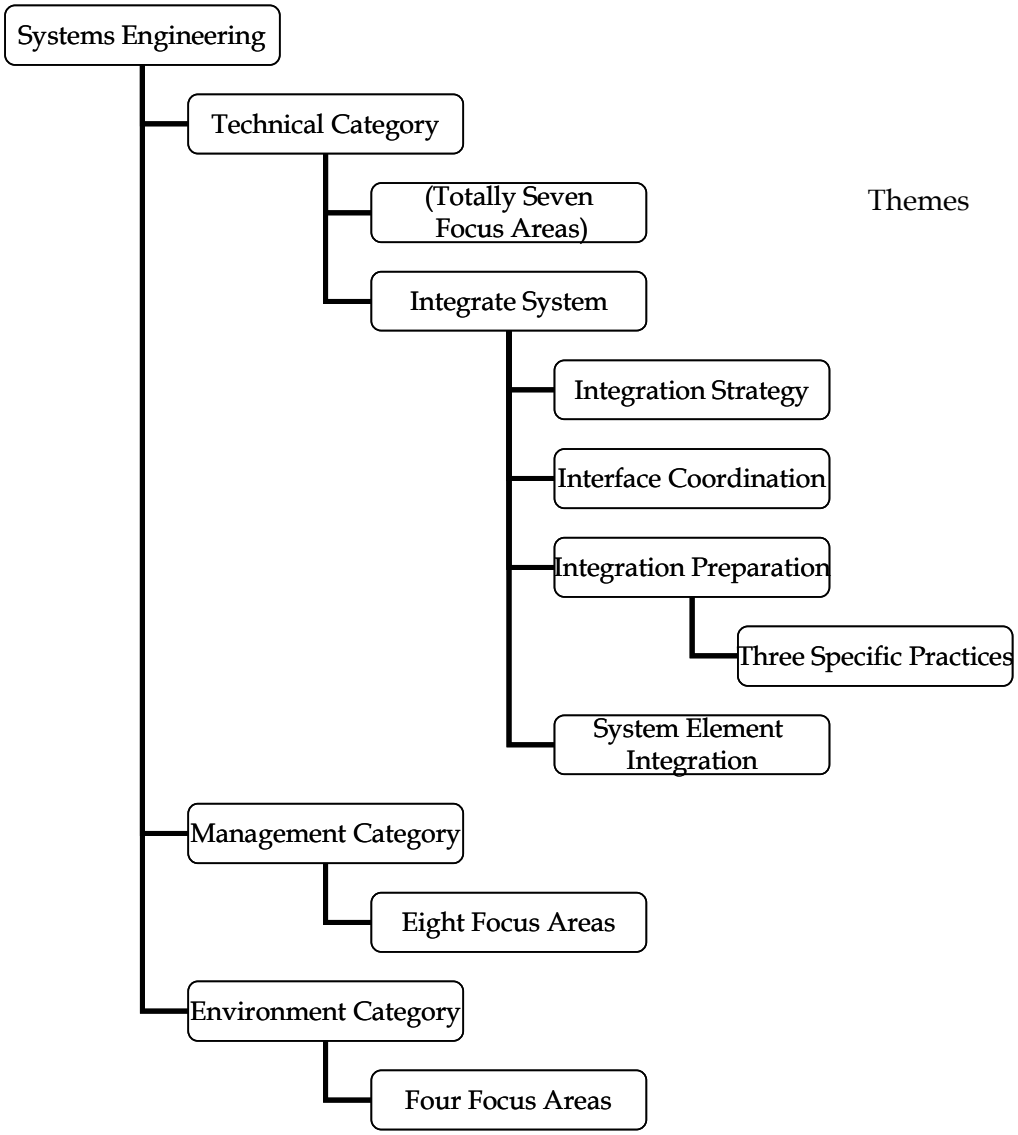

Figure 1. Structure of EIA-731.1

The interim standard includes a possibility to determine the capability level of an organization in a specific area. This is based on the observation that organizations typically take observable distinct steps in the effort to improve the performance. In EIA-731.1 these levels are intended to be used as means to help the organization in the planning and implementation of the improvement efforts. Six different capability levels have been defined. Level 0 indicates that the specific practices are not performed. Level 1 indicates that the specific practices on level one are performed. For level 2 to 5 both the specific and generic practices on these levels are performed. Note that no effort has in this study been made to determine the capability level of the organizations investigated as the target is to understand if the specific practices for product integration give the intended result.

The rest of this section summarizes the product integration process as it is described in EIA-731.1. The standard prescribes a set of specific practices that are considered to be essential for accomplishing the purpose of the focus area designated Integrate System (Focus Area 1.5).

The purpose of the Integrate System focus area is to ensure that the product and system works as a whole based on the components that have been integrated. Interfaces between components and functions that extend over many components in the system are in the center of attention. It is also noted that the integration activities should start early and are typically iteratively performed.

Four themes have been identified for the focus area. An Integration Strategy (1) is considered to be the basis for the integration process. This theme includes the development of a strategy that contains an integration sequence and a plan for the integration tests to be performed. The Interface Coordination (2) is the second theme and includes handling of the requirements on the interfaces as well as specifications and detailed descriptions. As a third theme, the Integration Preparation (3) describes how components are received for integration and the checking that the components are in accordance with the strategy and interface documentation. The final theme is the actual integration: System Element Integration (4). The components are integrated according to the plan and the inter-operations between the components are checked. It should be noted that the actual verification is described in a different focus area in the interim standard EIA-731.1, FA 1.6 – Verify System.

The different specific practices on capability level 1, 2 and 3 for all themes can be found in Table 4. The descriptions in the interim standard are short

and need to be interpreted with the description of the theme as a basis. Some guidance can be found in EIA-731.2 [6] that describes an appraisal method for EAI-731.1. However, the sample questions in this guide are also on a high level and require substantial expertise to be used.

3. Case Study Design

The case study was performed on three different product development groups in two different organizations. As the development methods are different in all three groups, the case study has been designed as a multiple-case holistic study as described by Yin [7]. The units of analysis are the processes for integration as perceived by members of the development groups in the three different cases. The focus of the study was on processes used at the time for the investigation, not described in quality systems or handbooks and not on processes that were under development.

3.1 Research Method

The interviews made with members of the development groups are the main sources of data in this investigation. Additional information was obtained from descriptions and examples of how the integration was planned and performed. For each case at least two persons were interviewed. The selection of subjects for the interview was based on two criteria. The first was that for each organization, both a manager and a developer should be interviewed. The second criterion was that the subjects should have extensive experience spanning over several years from the development in the investigated group.

The interviews were performed as open-ended discussions and all interviews were made by the same researcher. The researcher was guided by a discussion guide to ensure that different aspects of product integration were covered in the discussion. The guide was developed by two researchers and included questions related to three different areas; organization, implementation, and effectiveness of the product integration. The questions included in the discussion guide were not taken from the standard, but were designed to give an understanding of the used processes independent from descriptions in standards and models. During the interviews, the guide was used to ensure that the interesting topics were covered, and the specific questions asked were depending on how much information was obtained through the explanations from the interviewees. The use of open-ended questions allowed the researcher to follow up interesting statements that

lead to more information and a deeper understanding of the used process. Each interview was between one and two hours. The documentation from the data collection consists of notes taken during the interviews complemented with information from the written documentation.

The data collected can be divided into two types. The first type was descriptions of how the integration process was performed for each case and what activities were carried out. The second was descriptions of the problems that the units perceived in the integration process.

3.2 Analysis Method

After the interview sessions, the data collected was analyzed in several ways. This was done as a separate activity and without the involvement of the development organizations. For each case in the case study, the activities captured during the data collection were compared and mapped to the practices described in EIA-731.1. The result from the mapping showed if the development in the different cases were performed in accordance with the interim standard. As a second step, the problems identified were mapped to the specific practices in EIA-731.1 that are intended to ensure that the problems should not occur. Finally, the relations between activities performed and the problems were investigated. This resulted in Table 4 that indicates the relation between practices from EIA-731.1, activities performed and identified problems. A second phase of the analysis was to propose how the practices in EIA-731.1 should possibly solve the encountered problems. The results from this analysis in found in Table 5. The analysis was made by one researcher and reviewed by two other researchers.

3.3 Validity

Four types of validity threats are of interest for case studies [7]. In this section, we discuss these and the preventive measures to reduce them. Construct validity relates to the data collected and how this data represent the investigated phenomenon. Internal validity concerns the connection between the observed behavior and the proposed explanation for this behavior. The possibilities to generalize the results from a study are dealt with through looking at the external validity. Finally, the reliability covers the possibilities to reach the same conclusions if the study was repeated by another researcher.

The construct validity is dealt with through multiple sources for the data through more than one interview for each case. Additional interviews with other stakeholders as well as additional document investigations would have increased the construct validity. However, this would have required more intrusive investigations and would limit the availability to the organizations. The design of the discussion guide was based on available standards and methods and involved more than one researcher to ensure that the questions to be discussed were relevant. The researchers experience in software product development provided a basis for relevant discussions under the interview sessions.

The internal validity was secured in three ways. First, the connection between the behavior and the interim standard was done in several steps to avoid predetermined connections. Secondly, rival explanations have been listed and examined to exclude other causes to the findings. Finally, the analysis of the data and the connection to the interim standard has been reviewed by two additional researchers to avoid personal bias.

The external validity is dealt with through the use and description of three cases in two different application domains and through the use of several different standards and methods when defining the investigation area.

The reliability of the study has been secured through the description of the procedure used in the study and the documentation of the discussion guide.

4. Case Descriptions

Two product development organizations have been investigated, both developing systems for monitoring and control of different types of networks, but in different application domains. The systems operate in industrial settings with real-time requirements as well as high demands on availability and reliability. One of the units is developing products for two different environments. This has lead to the use of different processes and in this study they are treated as two cases resulting in a total of three cases. For each case the following sections contain a brief description of the product and the product development process. The descriptions also include the problems that were identified and described in the interviews. The problems are presented in tables where each problem is labeled with a P, the case number and a reference character.

4.1 Case One

The product in case one is a stand-alone product that is connected to a real-time data collection system. The development is done in one group with less than 20 developers and follows a clearly defined process. The product development of a specific release is based on a definition of the product that contains what should be included in each release. The first step in the development is the implementation of requirements on the functions for the release. Based on this, the unit and system verifications to be performed are defined. Development of the functions is done in units called components. The Rational Unified Process is used, and a document list defines the development process. The planning is made so the development is done in increments. The unit verification is performed by software developers. The strategy is that tests should not be done by the developer producing the software. The unit tests are often done through automatic testing. Specifications and protocols from the tests are reviewed by peers and system integrators. The tests are performed in the developer's environment and consist of basic tests. Functional tests are performed before the system tests.

The product integration is not defined as a separate process, but the product is integrated by the developers before the system verification. Before a component is checked in, it should be included in a system build to ensure proper quality. Delivery to the system test is done of the whole system. The test protocols and error reports from the unit verifications are reviewed with the system integrator before the system test. The system tests are performed by a core of system testers and temporary additional personnel. This strategy builds on well defined and detailed tests. The tests are focusing on functions and performance and are performed on different hardware combinations. This includes different variants of the product and different versions of the operating system. The test period takes approximately 12 weeks, with new versions of the assembled components received to system test every week. Although the development builds on increments, no integration plan is used for the product. The integration plan used is one for the whole system where this product is included. Typical time for the development of a release is less than one year.

The three most serious problems were captured for case one as described in Table 1. The routines are mainly followed, but due to tight deadlines, shortcuts may be taken. Sometimes uncontrolled changes are introduced in the software. This is typically done when a part of the system is changed due to an existing error that is uncritical and not planned to be corrected. Due to

the dependencies in the system, new errors may appear in parts that have not been changed. Also other connections between components that are not explicit generate this problem.

Table 1. Problems captured for case one

Label	Problem description
P1-A	Functions are not always fully tested when delivered for integration. This leads to problems in the build process or in integration and system tests
P1-B	Errors are corrected that should not be. This results in new errors with higher influence on functionality and performance
P1-C	Errors appear in other components which have not been changed

4.2 Case Two

The second case is a product that includes software close to the hardware. The development group is small and follows a common development process. This process includes rules for what should be checked and tested before a component is integrated. The tests include running the application in simulators and target systems before the integration. A specification for what should be ready before start of functional and system test are available. The architect is responsible for implementation decisions. The target system includes a complex hardware solution with the application divided on two target systems. Typical time for the development of a release is 1.5 year. This includes the full development cycle from defining the requirements to system testing.

Most of the problems appear because of the incapability and version mismatch of the test system, the final product and the test and final hardware platform (Table 2). Efforts are now made to go towards incremental development, and to increase the formalism in the testing. The tests will be made in three stages with basic tests performed by the designer, functional tests performed by a specific functional tester and system tests with delivery protocol.

Table 2. Problem captured for case two

Label	Problem description
P2-A	Problems appear as a consequence that tests for the components are not run in the same environment as the test system. Different versions of hardware and test platform are used.

4.3 Case Three

The development organization in this case is responsible for the design of a user interface that acts as a client to a database server. The organization is small, around 15 developers.

The current architecture has been recently improved. The old version of the system suffered from problems with many common include files. Through global variables and similar solutions permitted by the selected technology, unintended side-effects made debugging and error correction tedious. Different attempts to reduce the problems within the available technology lead to the insight that a design that was built on isolation of interfaces should be beneficial. The solution was to start building a new system. Included in this decision was a strategy to design interfaces carefully and to use technologies that permitted isolated components to be used.

The system is built up of components that primarily implements different parts of the user interface. Each component handles the communication with the server. This design was used to allow the development of services that are independent and dedicated for each component. The component framework defines the required interface for each component and provides a number of services, such as capturing of key strokes. The technology used permits the developers to easily isolate problems and to minimize the uncontrolled interference and dependencies between the components.

The development is organized with frequent builds and continuous integration of new functions. The integration is handled by the integration responsible. However, the checks before the inclusion of new functions are done by the developers. There are no specific routines in place for handling the interfaces. Changes are in practice always checked by the system architect.

The new system design has reduced the implementation time for a function with 2/3. The turn-around time for a system release has been reduced from six months to between one and three months. At the same time, a need for

maintaining the base platform has emerged. Also, some of the technical solutions have been questioned and may increase the need for maintenance (Table 3).

Table 3. Problem captured for case three

Label	Problem description
P3-A	Scattered architecture on the server side as a result of the decision to handle communication in each component

5. Collected Data and Analysis Results

In these three cases we found may similarities: size of the development groups, similar concerns, requirements of the products, similar product life cycle. What we have seen are the differences in the development processes and in used technologies and approaches. Our intention is to analyze what are the sources of the main problems and if they could have cause in deviation or absence of the activities pointed out in the best practices.

This section contains two parts. The first includes a table containing the analyzed data from the case study, while the second lists the problems found in the cases with a suggested implementation of the practices that could improve the performance.

5.1 Analyzed Case Study Data

The three steps of the analysis have been summarized and presented in Table 4. The table includes two parts for each practice. The first two columns show the description from EIA-731.1 for the specific practices for the focus area Integrate System. The first number in column one shows what theme the practice belongs to, and the second number is the capability level (i.e., 1-2 shows that the practice belongs to theme one and is placed on capability level 2). Finally, if two or more practices exist on a capability level for a theme, these are distinguished by a character. The following three columns include data from each of the cases. These columns include two things: (i) an indication for each case if the practice has been observed as performed (+) or not observed (-), and (ii) if there are indications of problems connected to the practice (*). The indicated problems are further described and analyzed in section 5.2.

Table 4. Specific practices for Integrate System compared to data from case 1, 2 and 3

SP	Description	Case		
		1	2	3
1-1	Develop an integration strategy	+ *	+	+
1-2	Document the integration strategy as part of an integration plan	-	+	-
1-3a	Develop the integration plan early in the program	-	+	-
1-3b	When multiple teams are involved with system development, establish and follow a formal procedure for coordinating integration activities	-	-	-
2-1a	Coordinate interface definition, design, and changes between affected groups and individuals throughout the life cycle	- *	-	+
2-1b	Identify interface requirement baselines	- *	+	+
2-2a	Review interface data	-	-	-
2-2b	Ensure complete coverage of all interfaces	-	-	-
2-3a	Capture all interface designs in a common interface control format	-	-	-
2-3b	Capture interface design rationale	-	-	- *
2-3c	Store interface data in a commonly accessible repository	-	-	-
3-1a	Verify the receipt of each system element (component) required to assemble the system in accordance with the physical architecture	- *	- *	+
3-1b	Verify that the system element interfaces comply with the interface documentation prior to assembly	- *	+	+
3-2	Coordinate the receipt of system elements for system integration according to the planned integration strategy	-	+	-
4-1a	Assemble aggregates of system elements in accordance with the integration plan	+	+	+
4-1b	Checkout assembled aggregates of system elements	+	+	+

5.2 Analysis of Observed Problems

In each of the cases, problems encountered in the performed product integration process were captured and discussed. The problems are in Table 5 cross-referenced by the researcher to the specific practices for the Integrate System focus area of EIA-731.1. Each problem has a label composed of a P, the case number and a reference character as in the tables in section 4. In addition to the description and the reference, a proposed action based on the specific practice has been included in the table.

Based on the data, we have made two observations regarding the perceived problem situation. The first is that all the problems for case one and two are related to capability level 1 specific practices. This may indicate that additional problems may be observed once all capability level one practices are performed, or it may indicate that higher capability level practices have less influence on the actual product integration results. The second observation is that case three had a similar culture for process adherence as case one, but the developers were forced by the technology to perform the specific practices.

5.3 Analysis of Propositions

As a summary of the analysis, we conclude that case two is performing the product integration most in line with the specific practices described in EIA-731.1 It is also clear that case two and three follow almost all the recommendations from capability level 1 specific practices. We see that case one has the most problems, and that all these problems are related to capability level 1 specific practices and we have noticed that in case three, the technology may help the development team in following the capability level 1 practices. The results are displayed in Table 6.

Table 5. Cross-reference between observed problems and relevant specific practices

Label	Problem description	Relevant specific practices and proposed actions
P1-A	Functions are not always fully tested when delivered for integration. This leads to problems in the build process or in integration and system tests	3-1a Ensure a handover to a dedicated integration responsible
P1-B	Errors are corrected that should not be. This results that new errors are introduced, with higher influence on functionality and performance	1-1 Ensure that the strategy and decision are followed through a handover procedure
P1-C	Errors appear in other components than the changed	2-1a, 2-1b, 3-1b Specify and enforce interface descriptions for all dependencies between the components
P2-A	Problems appear as a consequence that tests for the components are not run in the same environment as the test system. Different versions of hardware and test platform are used.	3-1a Ensure that the proper test equipment as described in the integration strategy is made available to the developers. Check that proper tests are performed through a clear handover to an integration responsible
P3-A	Scattered architecture on the server side as a result of the decision to handle communication in each component	2-3b Ensure that the rationale for design decisions are documented and communicated

Table 6. Summary of analysis

	# of specific practices performed of total number # of problems found		
	Capability level 1	Capability level 2	Capability level 3
Case 1	3 /7 5 problems	0/4 No problem	0/5 No problem
Case 2	5/7 1 problem	2/4 No problem	15 No problem
Case 3	7/7 No problem	0/4 No problem	0/5 1 problem

The first of our two propositions was that the problems encountered in the investigated units relate to the lack of execution of practices that are described in the interim standard EIA-731.1. In the analysis of the data and the comparison, we conclude that the problems found can be mapped to specific practices which support our proposition. We have also observed that it is primarily the inability to perform capability level 1 specific practices that have lead to observable problems.

The second proposition was that successful execution of the product integration can be mapped to specific implementation of practices described in the interim standard. For many of the practices on capability level 2 and 3, no observations have been made that they were performed, but only one problem has been reported that could be related to level 2 or 3 practices. Based on this and the observations regarding capability level 1 practices, an additional proposition has evolved and should be tested in future studies. This can be formulated as follows: A successful execution of the product integration can be mapped to specific implementation of practices described in the interim standard for capability level 1.

5.4 Rival Explanations

The conclusion regarding the propositions above can be challenged and in this section we examine rival explanations and analyze the possibility that these give better reasons to the data found in the study.

The first explanation examined is that there is no real connection between the performance and the specific practices described and that the data match only is coincidental. We consider this explanation to be unlikely due to two facts. The first is that the interim standard build on long industrial experience from companies and organizations from a wide set of areas and applications. The second fact is that the pattern shown in this study is clear and builds on three cases from two different organizations.

The second alternative explanation could be that the organizations due to other factors succeed in the product integration process. However, if there are other factors involved, these may also help in following the proposed practices. This is also the situation in case three where the selected technology has imposed a way of working on the product developers.

6. Conclusions and Future Work

Data regarding the product integration process from two development organizations have been collected and compared to the requirements described in a standard description of the product integration process. The problems observed in the case study have been compared to practices that describe activities that should improve the performance in the product integration.

We can from the observations conclude that the basic level of practices described in the interim standard EIA-731.1 includes activities that can help the organizations to avoid problems which can appear when integrating components to systems. Basic activities include (i) development and a clear specification of the strategy for the integration, (ii) keeping well defined interface descriptions up to date throughout the life cycle, (iii) that the integration of components follow the strategy and (iv) that the assembly is verified as planned.

We have also observed that there are indications that skilled use of component technologies as described in [8] facilitates the integration process. The factors contributing to this support are well described interfaces, the need to test components before integration and the explicit definition of the environment required by the components.

Through this investigation, partial answers have been found to our research questions, but additional research is needed. Future work should include steps to strengthen and further investigate the propositions made in this paper. They are (i) improvement of validation of the results by providing the feedback to the case participants in a form of discussions of accuracy of

collected data and the results at a common workshop, and (ii) additional case studies in industry. Additional descriptions of practices in standards and models need to be investigated in relation to industry practices. There is also a need to analyze the similarities and differences in the different standards and models. One additional research direction has been indicated with the purpose to confirm or refute the indications in this paper and in [5] that component technologies assist in the implementation of successful software product integration. Of specific interest may the integration problems related to COTS be.

References

[1] EIA/IS-731.1, Systems Engineering Capability Model, Electronic Industries Alliance (Interim Standard), (01 Aug 2002)

[2] Chrissis, M.B., M. Konrad, S. Shrum, CMMI, Addison-Wesley, Boston, MA, (2003).

[3] http://www.eia.org/. (Link valid April 2005.)

[4] http://www.incose.org/. (Link valid April 2005.)

[5] Larsson, S., I. Crnkovic, F. Ekdahl, "On the Expected Synergies between Component Based Software Engineering and Best Practices in Product Integration", Euromicro Conference, France, August 2004, IEEE

[6] EIA/IS 731.2, Systems Engineering Capability Model Appraisal Method, Electronic Industries Alliance (Interim Standard), (01 Aug 2002)

[7] Yin R. K., Case Study Research: Design and Methods (3rd edition), ISBN 0-7619-2553-8, Sage Publications, 2003

[8] Szyperski, C. et al, Component Software -- Beyond Object-Oriented Programming, (2nd edition), ISBN 0-201-74572-0, ACM Press, New York, (2002)

Paper C

PRODUCT INTEGRATION IMPROVEMENT BASED ON ANALYSIS OF BUILD STATISTICS

Stig Larsson, Petri Myllyperkiö, Fredrik Ekdahl
Presented in a shorter version at the ESEC/FSE Conference, Cavtat, Croatia, August 2007
Avaiable from ACM: http://doi.acm.org/10.1145/1287624.1287696

Abstract

Process improvement efforts based on best practices and standards such as CMMI use appraisal results as input and focus on implementing processes as described in reference models. Since these models are of general character the conclusions from the assessments could easily overlook problems experienced in the daily work. In addition, process improvement programs often fail to engage practitioners. To improve this, data that can be related to the daily work can help. This paper reports on the results from a study performed to understand how data based on measurements can complement project appraisals in finding improvement possibilities. The effect is that conclusions from appraisals can be corroborated by metrics, and also that additional areas for improvement can be identified. A method for mapping project data to different practices and combine this with project appraisals to form a basis for focused performance improvement is proposed. In our study the product integration processes in four projects from three organizations have been examined using the proposed method and the findings are presented. The study demonstrates how the two components, the collected metrics and appraisal results, complement each other in the effort to develop product integration process improvement effectiveness.

1. Introduction

In industrial software projects, it is common that product integration becomes a bottleneck as many problems are first revealed in this process [1][5]. Industrial standards and models include practices that describe what is considered to be useful practices for product integration [3][6][8][14][15]. Main themes in these descriptions are; preparation of the environment, handling of interfaces to ensure that different parts of the system can interact, preparation of parts that are to be integrated, and the actual integration including the initial verification of the resulting system. The practices can be used as a basis for appraisals to provide guidance for

improving the process. One risk organizations using this method face is that it typically involves managers and high-profile developers as these are knowledgeable in the processes [2].

An additional way that is used to identify improvement activities is to define a metrics program for the organization. This is often part of the models and standards for product development; the implementation of a full metrics program is often a challenging task [4]. However, data is often collected from parts of the processes. This gives a possibility to use this data for identifying improvements. For product integration, statistics from build activities are examples of data that are often collected or easily obtained. When already existing metrics are used, there is a risk that these to not result in improvement activities where the biggest gains can be made.

We propose that already existing data from the development projects can be used in combination with appraisal results when selecting and prioritizing improvement activities. This paper describes a method that combines the use of process data and appraisal results for identifying and prioritizing improvement activities.

Our hypothesis is that the method we propose can increase the accuracy and extend the findings and results from existing appraisal methods in the identification and prioritization of improvement activities. We also propose that this can be achieved with an effort that is substantially smaller than what can be gained.

The method has been tested on four different projects in three organizations with focus on the product integration process. All these organizations are developing industrial control systems, with real time requirements. This makes the integration very important as many of the resulting performance characteristics are determined by how well the parts of the product or system work together.

The study presented in this paper is a continuation and extension of the research presented in [16][17][18] which investigates the use of project and organization appraisals for identifying improvement activities, and what support can be found in standards and models regarding product integration.

The remainder of this paper is organized as follows. Section 2 gives an introduction to the product integration process area and section 3 describes related work. Section 4 explains the research method, including a description of the proposed method for identifying improvement activities. Section 5 includes a brief description of the organizations and projects that have been investigated. Section 6 contains the results from the data

collection and the appraisals performed, and a discussion of the results. A discussion on the limitations of the results and the threats to their validity are found in section 7. Finally, conclusions and future work is presented in section 8.

2. Product integration

This section describes product integration and the relation between build activities and product integration. The main purpose of product integration is to assemble the product from product components, and ensure that the result of the integration functions properly. For software products, the integration may be made in several steps, building components from other components. A component is in this context used with a broad definition as in [11], describing "a recognizable 'chunk' of software".

In software product development the concepts of integration and build are closely related. Each component needs to be built, and included in builds together with other components as one of the activities of the integration. However, not all builds lead to integration. In concurrent software engineering, the development of components often takes place in an independent development environment, in which the developer creates, debugs, and tests the component. In order to execute the component for debugging or unit testing purposes it needs to be built together with other components. The integration takes place when the separately developed and tested components are checked back to the development mainline and built together with the other separately developed components. This process is depicted in the Figure 1.

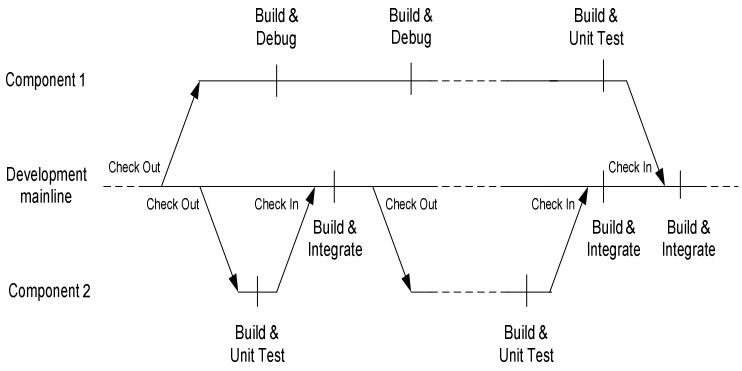

Figure 1. Example of build and integration timeline in a project

Product integration is included in different ways in standards and models used for describing product development. When selecting a reference model as a guideline for the activities, it is important to use one that fits the organization and the intended use. In this paper we use the definition and description of product integration from the Capability Maturity Model Integrated (CMMI) from the Software Engineering Institute at Carnegie Mellon University [6]. The motivation to use CMMI is that the model separates the product integration process from other processes explicitly which simplifies the reasoning about the integration activities.

3. Related research

This section describes research related to process improvements based on metrics and reference models.

Using metrics as a basis for improvements have been extensively described. One important reference is [4] where practical guidelines for basing process improvements on measurements are described. The authors define practical guidelines based on the Goal Question Metrics paradigm, and propose that explicit modeling of the process help in structuring the measurement plan. They also acknowledge that setting up a measurement program is challenging. How software process improvement can be based on measurements is described in [7], where the combination of using metrics and process appraisal is stressed. However, in neither of these papers, the direct mapping between the measurements and practices as described in the used reference models is used.

In [12], an example of an implementation of a metrics program for small projects is described. Here, one of the conclusions is that a metrics collection program should start with a small set of metrics that will show the benefits of collecting data. This is in line with our proposal that already existing measurements should be used to as a starting point. Also, that the measurements are directly connected to the activities performed by practitioners is supported as this will encourage software personnel to collect data.

Houston [13] studied the integration problems occurring in an avionics system. Types of integration problems were identified based on the functionality where the problem occurred. By classifying the integration problem reports and estimating the expected handling time through simulations of the report types several possible SPI activities were

identified. The study complements our research as one alternative route to find improvement activities.

Product integration processes are included in different reference models for product development [3][6][8][14][15]. These can be used as a source for improvements, and are a part of the method described in this paper. One concern with reference models is the inadequate validation of them; it is difficult to find research looking into the validity of the content of these reference models. However, the reference models are developed based on experience from a large number of organizations, ensuring that important considerations are taken into account. The activities in the product integration area have also been the subject of interest from the agile community where frequent builds is one of the cornerstones. One example is [10] where Fowler describes the requirements on developers: before committing back to mainline the developer would need to update his work area with the latest mainline, i.e. build against the latest changes of other developers. Only after that, integration into the mainline would be permitted. However, the connection between metrics and changes in the procedures has not been investigated.

The conclusion from reviewing related research is that the combination of using metrics and appraisals is described, but that mapping the metrics directly to the practices and activities appears to be novel.

4. Research method

This section describes the three steps performed to investigate the proposed working method. The aim of this research study is to understand how problems in product integration in an industrial setting can be identified and reduced through combining appraisals and metrics collection. It has been conducted as a multiple-case study, with the units of analysis being the methods for process improvements for product integration processes.

In this research we have identified two research questions based on our hypothesis: 1) Can the additional activities in the proposed method increase the accuracy and extend the results from appraisals?, and 2) Is the effort needed to perform the additional activities significantly less than the gains achieved through avoiding unnecessary work by implementing the identified improvement activities?

The research has been performed in three steps. First, a method which combines project appraisals and data collection was described. The proposal was based on our observations indicating that project appraisals do not take

full advantage of data already available in the organizations, and that additional support for making use of available data in a structured way was needed.

The second step was to use the method in an industrial environment. Four projects in three different organizations have been examined.

Finally, the method was evaluated, and improvement proposals were discussed.

4.1 Proposed working method

The proposed method for using appraisals in combination with project data is presented in this section as laid out in Figure 2.

According to the proposed method a project appraisal is conducted and process data collection is performed independently of each other. The project appraisal requires an established appraisal methodology based on an appropriate reference model that will provide valid and repeatable results over time.

The data collection requires an agreement on which data to be collected and on the procedures for collecting, storing, analyzing and maintaining the data. To ensure that the available data is relevant for the use with the appraisal results, we propose conducting a workshop as Step 1 where the data collection routines are examined. The purpose is to ensure that the collected data can be used for the mapping to the chosen reference model.

Step 2 of the method is the actual data collection. This is done by practitioners and should as much as possible be automated, based on tool support.

Step 3 contains an identification of the direct cause of the data. The collected data is classified to ensure proper use of the domain knowledge held by the practitioners. The identification terms are depending on the process area and the terminology used by the practitioners. Note that this identification should describe the direct event resulting in the data. An example of the events identified for the build and integration process can be found in Table 3.

As a fourth step, a mapping based on the chosen model or standard is made by process experts to ensure that the root cause is targeted in the improvement activities. One important aspect is the criteria used for the mapping in relation to the selected model. In this investigation, the mapping has been based on the experience of the researchers knowledgeable in

process development and improvement, and the definition of general criteria needs to be performed for a large number of process areas and remains a subject for further research.

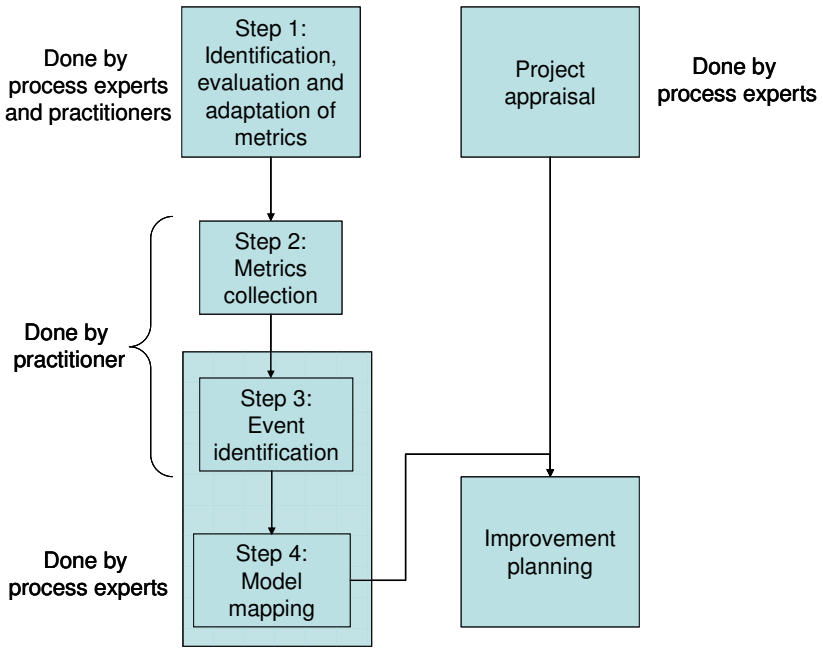

Figure 2. Proposed method for finding improvement activities

The invention in this method is the activities leading to the mapping between the collected data and the reference model used as a basis for improvement planning. The added value is that indications from the appraisals can be substantiated through supporting data, but also that new areas for improvement needs can be identified.

4.2 Method implementation

This section describes how the method has been used to find improvement activities for the Product Integration process area to test our hypothesis. Information has been compiled through appraisals in the four projects involved and through collection of data from the corresponding build processes. The appraisals were based on CMMI and performed in accordance with the requirements for Class C appraisals as defined in by the

Software Engineering Institute [19]. The build process data was collected by the practitioners who also made a first identification of the cause of build failure. A further mapping of the build failures was then done by the researcher based on the Product Integration process area as defined in the CMMI.

Project appraisals were conducted for all projects as a means to understand how the product integration was performed and what problems could be expected. In our study, the appraisals were performed by one researcher for each of the cases. To ensure consistent appraisal findings, a standardized appraisal method is used [9]. Interviews and document reviews were carried out, and evidence was collected on the execution of Product Integration practices. The CMMI describes goals for all process areas included, and for Product Integration, three goals are defined: Prepare for Product Integration, Ensure Interface Compatibility, and Assemble Product Component and Deliver the Product. Each of the goals is supported by specific practices that are used as guidance for organizations that would like to achieve the goals. The specific practices are listed in Table 1 together with the goals they support. Note that Specific Practice 3.4 has not been considered further in this study as the activities are not directly related to the integration activities.

The interviews were performed in groups with at least two representatives for the organization with good knowledge about the practices used. The discussion was lead by researchers using the description of the different practices from the reference model as a discussion guide. Notes where captured on observation forms, structured in accordance with the Product Integration specific practices. The document reviews included investigations of the process descriptions for each organization as well as project documents such as project plans and configuration management plans. Also for the document reviews, the observation forms based on specific practices were used. An example of the observations compiled based on the interviews and document reviews and the results of the analysis can be found in Table 2.

Table 1. Product Integration Specific Goals (SG) and Specific Practices (SP) in CMMI

SG 1: Prepare for Product Integration
SP 1.1: Determine Integration Sequence
SP 1.2: Establish the Product Integration Environment
SP 1.3: Establish Product Integration Procedures and Criteria
SG 2: Ensure Interface Compatibility
SP 2.1: Review Interface Descriptions for Completeness
SP 2.2: Manage Interfaces
SG 3: Assemble Product Component and Deliver the Product
SP 3.1: Confirm Readiness of Product Components for Integration
SP 3.2: Assemble Product Components
SP 3.3: Evaluate Assembled Product Components
(**SP 3.4:** Package and Deliver the Product or Product Component)

The inclusion of additional process areas in the investigation was also considered and analyzed. The related process areas as described in [6] that were judged to be of interest are Technical Solution, Verification, Validation, and Configuration Management. However, as the collected statistics from build activities are less direct as a basis for the examination of the performance for practices in these process areas than in Product Integration, a decision was made to focus only on Product Integration.

Table 2. Example Observation

SP 1.1 Determine the product-component integration sequence.	
Observation: Project manager plans the integration points and communicates to the development team. The plan contains the high level description of functionality of components intended to be in the integration. Integration sequence doesn't define the expected functionality in detail; usually all that is available is taken into build. Some exceptions exist.	**Analysis** + Integration points planned in advance and coordinated with the relevant stakeholders - Integration sequence defines functionality on a too high level to be useful for the developers, and ad-hoc deliveries are done **Conclusion** SP 1.1 is not performed as intended.

In parallel to the appraisals, the data collection method was implemented. The described working method includes four steps. The first is to identify and decide on what metrics is useful. After discussions and preliminary data collection, it was decided to use statistics from build activities for each of the projects in the study. The second step, data collection, was made either automatically by tools or performed by practitioners. For each build, the result was recorded to be a successful build or a failure. For failing builds, the immediate cause was recorded, which is the third step in the method. Examples of immediate cause include; files not delivered for integration as decided, faulty include files specified in the build script, insufficient space for result files in the build environment and errors in linking due to changes in interfaces. Each of these types of errors can depend on missing practices and working methods, insufficient tools or a combination of them. Also the delay in making a planned build available for the project was recorded. This information can be used in the prioritization of the changes in working methods and procedures. The period over which the data was collected ranges from two to nine months for the different projects.

After the build data collection and an initial fault categorization in the projects, a mapping to the used reference model was made as the final and fourth step in the method. This mapping was based on the experience of the researchers through reasoning about each of the errors, linking them to a specific practice as specified for Product Integration in [6]. Through the investigation, we have collected a number of typical cases of mappings between the errors and the specific practices. The set of connections obtained from this study can be found in Table 3. This table can be seen as a first step towards formulating a set of criteria for the mapping the relationships. To verify the mapping, the results were reviewed by representatives for the participating organizations.

To implement this method in an organization, three prerequisites are necessary. First, the organization needs to have or decide on a reference model that can be used for appraisals. Secondly, sufficient knowledge of the selected method must be available in the organization; relying on external resources for the implementation will lead to problems when these are not longer available. Finally, the organization must have data from the process areas of interest, or have the means to easily define and implement data collection.

4.3 Evaluation of the Method

The evaluation of the proposed method has been performed through an assessment based on the collected data. It was performed as workshops, both by the research team and in separate workshops together with representatives for the development organizations. Two criteria were formulated to support the evaluation of the model and were based on our research questions and our hypothesis. The first criterion was to understand if different results are obtained from the appraisals and the mapping based on data, giving additional information when selecting and prioritizing improvement activities. The second was that the additional effort needed using the proposed working method should be smaller that what can be saved.

Table 3. Relating errors to specific practices

Error and/or immediate cause for error	Specific practice
Syntax error in build script	1.2 Establish the Product Integration Environment
Build computer crash	
Problem in configuration management system, delivery not included in build	
Problems overwriting files from earlier files due to errors in scripts	
Error to clean build area	
Build area too small, build fails, no routine to check this	1.3 Establish Product Integration Procedures and Criteria
Compile error due to changed type in common interface	2.1 Review interface Descriptions for Completeness
Compile error in one unit due to changes in another	2.2 Manage Interfaces
Compile error due to “old” changes of interface	
Undefined identifier, code obviously not compiled	3.1 Confirm Readiness of Product Components for Integration
Functional error in code resulted in new build	
Component not compiled in all environments before delivery	
Linker error, identifier conflict	
Error at smoke test	
Unidentified identifiers, definition not checked in	3.2 Assemble Product Components
Component included in library, but not in build	
New component not checked in	
Component depending on changes in other module not included in the build	
Failed to synchronize delivery with other subproject	

5. Description of cases

This section includes a description of each of the four case projects. Three product development organizations have been investigated. For two organizations one project has been investigated, Case 1 and 2. For both of these organizations, the project investigated is the integration project which mainly serves to integrate functionality developed in several other separate projects. This way of integrating enables the organization to run function development in parallel, but puts high demands on the integration process. For the third organization, two product development projects have been incorporated in the study, Case 3 and 4. In these, the product integration is performed in the same project as the functional development.

All products developed in the case organizations are used for industrial purposes and have high real-time, availability and reliability requirements. For each of the cases, characteristics such as size of software product and organization, and project organization are described.

Case 1 The study was performed at a 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 an infrastructure that provides basic services like: a broker for message-based inter-task communication, configuration support, persistent storage handling and system startup and shutdown.

Case 2 The organization in Case 2 is the largest in the study. The product consists of more than three MLOC and the number of developers involved is close to 80. The organization develops a complex real time control product including event, trend and error handling, data collection, communication, and operator interface. The product is part of a suite of about 30 products, forming a system that is used in process industries. The development is tightly coordinated with the development of these other products. The development process varies between different groups in the organization, but all parts are delivered to the build and integration process. The build team delivers the results from the builds to the group performing automated

regression tests. These two teams have possibilities to capture integration problems and to analyze background problems. The builds are performed as daily builds, weekly builds and baseline builds. The daily builds are performed to ensure the stability of the code base, and have the build environment adapted to additions in functionality. Weekly and baseline builds are made available to other parts of the organization for verification purposes. Development cycles are between 12 and 18 months.

Case 3 The project develops embedded software for new generation of protection relays for electrical network. The product architecture consists of base software, application components, and higher level services, such as communication services. The base software is a platform that provides the running environment for the application components and services, and separates them from the hardware. The software is built of source-code level components that are compiled and linked to a single executable binary. Most of the components are developed by the organization but also some 3rd party components are used. The project is responsible of developing the base software and integrating it with the service and application components, which are partly developed in separate projects. The outcome of the project is a reference configuration which will be the baseline for the actual productization projects, including base software, service components and reduced subset of application components. Thus, the studied integrations include all types of the components used in the product.

The development organization is small with around 15 developers. The main development team is located in one site, but few persons from other sites have participated in the development partially. At the time of the study the project size was around 500 KLOC, however the final products built on technology will be larger than that.

Case 4 The project develops embedded software for new generation of protection relays for electrical network. The product architecture resembles that of Case 3, i.e. it consists of base software, application components, and higher level services. The product has some common components with Case 3, but it is based in other hardware than Case 3. In addition some components have a longer history, being released in the existing products. However there is remarkable development work ongoing in the project. As in Case 3, the software is built of source-code level components, either developed by the organization or 3rd party, that are compiled and linked to single executable binary. The project is responsible of further development of the base software and integrating it with the service and application

components, which are partly developed in separate projects. The outcome of the project is a new product.

The development organization is small with around 15 developers. The main development team is located in two sites in two different time zones. At the time of the study the integrations were performed only to the components of base software and service components, while as the application components were not yet included in the integration. The project size was over 500 KLOC, but when integrated with the application components, it is expected to be >1000 KLOC.

6. Results and discussion

This section includes the results from the data collection and appraisals, and a discussion about the results. The data is presented in four tables to show the different types of data that have been collected. These are the percentage of failed builds, the portion of failed builds related to the specific practices in CMMI combined with the results from the project appraisals, the delays that the failed builds have caused, and the extra effort needed to use the working method proposed in this paper. Each of these tables gives feedback to the organizations and can be used as a basis for process improvement activities.

In Table 4, the basic statistics from build activities for each of the projects are listed. The table includes the number of builds investigated and the portion of the builds that have failed (including results from builds not passing startup tests).

Table 4. Statistics from build activities

	Number of builds	% builds with errors
Case 1	240	28
Case 2	126	19
Case 3	776	16
Case 4	107	32

The projects in Cases 1 and 2 are typically performing daily builds. For the project in Cases 3 and 4, the builds are performed continuously, and occur several times every day. In Case 4, the developers did not have the process

to do frequent check-ins. This was done for Case 3 and we note that this seems to reduce the portion of failing builds, which needs further research.

In Table 5, the errors have been divided on the different specific practices of the process area Product Integration as described in the CMMI. The numbers indicates the ratio of errors that can be related to a particular practice as classified according to our proposed method. The table also includes the results from the appraisals in the development units. If the conclusion from the appraisal is that a practice is performed by the project, this is indicated with a plus sign (+). If the results are that the practice is not performed, this is indicated with a minus sign (-).

It is noticeable that for some of the specific practices no build problems have been found. In the mapping of erroneous builds, no problems were attributed to specific practice "1.1 Determine Integration Sequence" and "3.3 Evaluate Assembled Product Components". As the statistics from build activities do not match any of these practices, this is in line with the expectations. However, it is important to remember that these practices have been included in the model as they are considered as important parts of the product integration process. Also, the practices depend on each other and the root cause for some of the problems linked to specific practice "3.2 Assemble Product Components" could for example be in the area of the integration sequence. However, this needs to be further investigated in each improvement activity.

An additional interesting observation is that for two practices, errors were found even if the appraisal results showed that the practices are performed. Three possible reasons for this have been found. The first is that the practices may not be fully followed even if the appraisal results indicate that they are implemented. This can be the result of an inadequate appraisal method, or the perception in the organization that a practice is made even if it is not. The second is that the mapping is incorrectly made. However, this explanation has been abandoned after discussions with practitioners indicating that the mapping is correct, whereas the adherence to practices is perceived to vary between individuals. A third possible reason is that the chosen reference model does not fully describe the requirements on activities needed to ensure a successful implementation of the processes. To understand if this is the case, a validation of the model is needed which in the case of CMMI is subject for further research.

It should be noted that a number of the erroneous builds in Case 2 were not possible to classify and link to a practice. The reason for this is the lack of complete data. In this case, the direct cause of the failure has not been recorded in a way that makes it possible to do the analysis. These fault instances occurred early in the collection of data, and through a discussion with the practitioners, better quality of the data for later builds was obtained. This led to an additional step in the proposed method; the evaluation and enhancements of existing data collection is needed as the first step. Even if data is already collected in the projects, small changes can enhance the usefulness both for the organization and for research purposes. It can also be noted from all cases in the study that the interest for the results from the investigation increased over time as results became available.

One important factor for product development projects is to avoid or reduce delays. This is especially important for activities that are providing results for many project members and other stakeholders. Typically, results from the product integration are used as a basis for further development, and for verification and validation activities. This means that a delay of an expected build with a specific functionality may affect the project progress substantially. This makes it important to understand and measure the delays in the integrations made available through builds. By linking the errors and the corresponding delays to specific practices, it will be possible to determine what is causing the most severe delays. Table 6 shows the delay times as measured in the build process in the four case projects divided on specific practices, based on the mapping presented in Table 5.

The information in Tables 5 and 6 can be used to prioritize the improvement activities. An example of this is in Case 1 where more than half of the problems are due to practice “3.1 Confirm Readiness of Product Components for Integration” and where the average time for correcting the error is more than 3 hours. Even if the “3.2 Assemble Product Components” also is a frequent source for problems, the average delay is less than half. This gives the organization input for actions to ensure that components are ready for integration. It is also interesting to see that this is also one practice that was not considered to be performed in the appraisal of Case 1.

Table 5. Portion of build errors and performed activities related to CMMI Specific Practices for Product Integration

Specific Practice	Case 1	Case 2	Case 3	Case 4
1.1 Determine Integration Sequence	0% +	0% -	0% -	0% -
1.2 Establish the Product Integration Environment	6% +	29% -	11% +	11% +
1.3 Establish Product Integration Procedures and Criteria	2% -	0% +	0% -	0% -
2.1 Review Interface Descriptions for Completeness	2% -	0% -	0% +	0% +
2.2 Manage Interfaces	8% -	8% -	9% -	26% -
3.1 Confirm Readiness of Product Components for Integration	53% -	17% -	45% -	63% -
3.2 Assemble Product Components	30% +	29% +	35% -	0% -
3.3 Evaluate Assembled Product Components	0% +	0% +	0% -	0% -
Errors not possible to classify (no detailed enough cause for error recorded	0%	17%	0%	0%

Table 6. Maximum and average time in hours between failed build and correct build classified per practice

Specific Practice	Case 1	Case 2	Case 3	Case 4
1.1 Determine Integration Sequence	-	-	-	-
1.2 Establish the Product Integration Environment	8 2.9	1.3 0.3	6.5 2.5	6.4 3.3
1.3 Establish Product Integration Procedures and Criteria	8 8	-	-	-
2.1 Review Interface Descriptions for Completeness	2.5 2.5	-	-	-
2.2 Manage Interfaces	8 5.5	2 2	2.4 0.8	34 11.5
3.1 Confirm Readiness of Product Components for Integration	8 3.3	14 7.25	6 0.8	15 6.75
3.2 Assemble Product Components	8 1.8	7 2.5	14 1.4	-
3.3 Evaluate Assembled Product Components	-	-	-	-
Errors not possible to classify (no cause for error recorded or not detailed enough for mapping)	-	5 5	-	-

The first criterion for evaluation of the proposed method is to understand if we get additional information through using a combination of appraisals and collection of data that is linked to practices. Our conclusion from the data in this section is that additional information is obtained. Through the proposed method, the problem areas with regards to product integration for each of the organizations were identified. Improvement activities based on this evaluation method can address problems that are experienced by the practitioners in the organization and where improvements can be measured.

The second criterion used to evaluate if the proposed method is helpful is to look at spent versus reduced effort in the projects. To examine this, we have estimated the additional effort used to execute the method. Unfortunately, for the data collection step in the method it is difficult to make a distinction between the additions and what would have been performed anyway. However, Table 7 includes an estimate for the total effort.

This effort should be compared to the delays in the projects due to erroneous builds. For the most frequent reasons for erroneous builds, the delay is more than 2 hours for case 1, 2 and 4. If the project can avoid one erroneous build which would have delayed five persons for two hours, this is equal to the additional effort in case 1 and 4. As projects in these organizations are big, the organizations judge that more than five people normally are delayed until an erroneous build is corrected. The indication we get from comparing the extra effort for the proposed method with the reduction of delays in the projects is that the benefits are substantial, but depends on how successful the resulting improvement activities are.

To summarize, we examine the two criteria that have been used to evaluate the method and to investigate our two research questions. The first criterion concerns the additional information that can be gained. We have seen in the case studies that observations from the appraisals can be supported by data, but also that additional improvement activities can be identified. The data expose deficiencies not perceived by the organization as problems.

The second criterion used is to understand if the additional effort spent using the method is considerable less than what can be gained in a project by avoiding unnecessary work. The collected data is partly based on estimations, but indicate that the savings from implementing improvement activities based on the findings from using the method are substantial.

Table 7. Additional effort spent implementing the proposed method

Activity	Average effort in person-hours			
	Case1	Case2	Case 3	Case 4
Workshop for changing data collection method per project (measured by researchers)	0	5	0	0
Additional data collection effort (estimation by development organizations)	0	0	0	0
Description of errors per build (estimation by development organizations)	0	0,1	0,1	0,1
Mapping errors to practices per failed build (estimated by researchers)	0.1	0.1	0.1	0.1
Summarizing data for improvement activity proposal per project (measured by researchers)	3	2	4	2
Total extra effort per project WS+DC*NoB+(DE+ME)*NoFB+SD WS: Workshop DC: Additional data collection NoB: Number of builds DE: Build error description ME: Mapping errors to practice NoFB: Number of faulty builds SD: Summarize data	10	18	27	9

7. Limitations and validity threats

This section examines the threats to validity in this paper, and through this also describes the limitations of the study

As proposed in [20], four types for validity have been considered in this study; construct, internal, external, and reliability.

Construct validity relates to the data collected and how this data represent the investigated phenomenon. Internal validity concerns the connection between the observations and the proposed explanation for these observations. The possibilities to generalize the results from a study are dealt with through looking at the external validity. Finally, the reliability covers the possibilities to reach the same conclusions if the study was repeated by another researcher.

The construct validity is addressed through multiple sources for the data in the project appraisals through more than one interviewee for each case as well as using document reviews. Additional interviews with other stakeholders would have increased the construct validity. However, this would have required more intrusive investigations and would limit the access to the organizations. The use of a model as a basis for the interviews and document reviews ensures that the data collected is relevant. For the build data, discussions were held with the practitioners in all cases to ensure that the data was relevant, and adjustments where done to reflect the way of working. One example from case 2 is that intermediate builds are used to verify that the build environment is working. This data was included in the first analysis, but after discussions with the integration team, this was clarified. These considerations are clearly indicated in the underlying data.

The internal validity was addressed in several ways. For the statistics from build activities, the mapping to the process practices was done as described in several steps to avoid predetermined connections. Also, through involving practitioners in the direct identification and review of the mapping to the practices reduces the risk regarding the internal validity. Also for the appraisals several steps are taken to ensure that the mapping and understanding is correct. A detailed description of the methods used can be found in [9]. One risk related to the internal validity is that we through the investigations and through participation in the discussions of product integration affect the processes while collecting data. This is specifically relevant as the data collection is done over a long period of time.

The external validity is addressed through the use of our proposed method in four cases in three different application domains. One threat is that all cases are from the same multinational company. However, the investigated organizations are from different divisions, have distinctly different development processes, and the products are intended for different application domains.

The reliability of the study has been addressed through the detailed description of the procedure in this paper as well as using a procedure for appraisal based on [9] used in the study.

8. Conclusion and future work

This paper proposes and validates a method in which appraisals and data collection are combined to more efficiently and accurately identify improvement activities. The method has been examined through use in four cases. The results from the case studies support our hypothesis that the proposed method can increase the accuracy and extend the findings from the existing appraisal methods in the identification and prioritization of improvement activities. The study indicates that this can be achieved with an additional effort that is significantly smaller than what can be gained in one project through implementing the resulting additional improvement activities.

The proposed method combines project appraisals with a structured way of mapping data from the process to practices described in a selected reference model. The steps in the method are to identify and define what metrics are to be collected, to collect the data, to identify the direct cause for the metrics, and finally to map the direct causes to the practices. The third and the fourth step are additions to current used practices and add value as results from appraisals and also extend the results by the identification of additional areas needing improvements.

There are four main reasons why this method can enhance the effectiveness of improvement activities in the area of build and product integration activities. First, the focus in the development projects is normally on getting the build through. This method gives possibilities with small means to look for the underlying problems avoiding the repetition of problems. The second reason is that it helps in using proven good practices as a model for finding root causes in a structured way. A third effect is that areas that are considered to be performed in the organization are exposed as not being effective. Finally, the method gives an increased focus on the big picture,

and will help establishing deeper understanding in the organization of the importance of product integration.

There are several pointers for further investigations. The current study only looks at the erroneous builds. Additional information can be gained from investigating integration problem reports similar to what has been performed in [13]. The collected data also indicates that continuous integration reduces the total "down-time", i.e. the average time to fix a build: does continuous integration affect that if used properly. However, to draw any conclusions we think that a dedicated study on this topic would be valuable. Finally, additional investigations into practices that were not connected to any erroneous builds are interesting as a means to find root causes.

Acknowledgements

The authors wish to thank ABB and all participants in the study for all support. This work is partly funded by the Swedish Knowledge Foundation (KKS).

References

[1] The Economic Impacts of Inadequate Infrastructure for Software Testing, RTI, National Institute of Standards and Technology, Gaithersburg, MD, USA, May 2002

[2] Aaen, I. Software Process Improvement: Blueprints versus Recipes, IEEE Software, Volume 20, Issue 5 (September/October 2003), 86-93

[3] ANSI/EIA-632-1999, Processes for Engineering a System, Government Electronic and Information Technology Association, Electronic Industries Alliance, 1999.

[4] Briand, L.C., Differding, C.M:, Rombach, H.D. Practical Guidelines for Measurement-Based Process Improvement, Software Process: Improvement and Practice, Volume 20, Issue 5, (1996), 253-280

[5] Campanella, J., editor, Principles of Quality Costs: Principles, Implementation, and Use, 3rd edition, ASQ Press, ISBN 0-87389-443-X, Milwaukee, WN, USA, 1999

[6] CMMI® Product Development Team, CMMI® for Development, Version 1.2. Technical Report CMU/SEI-2006-TR-008, Pittsburgh, PA, USA, 2006

[7] Dybå, T., Skogstad, O. Measurement-based software process improvement, Telektronikk, Volume. 93, Issue. 1, (1997), 73-82

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

[9] Ekdahl, F. and Larsson S. Experience Report: Using Internal CMMI Appraisals to Institutionalize Software Development Performance Improvement. In Proceedings of 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (Euromicro'06) (Cavtat, Croatia, August 29-September 1, 2006). IEEE, Los Alamitos, CA, USA, 2006, 216-223.

[10] Fowler, M., Continuous Integration, http://www.martinfowler.com/articles/continuousIntegration.html, (2006)

[11] Gorton, I. Essential Software Architecture. Springer, Berlin Heidelberg NewYork, 2006

[12] Grable, R. et al., Metrics for Small Project: Experiences at the SED, IEEE Software, Volume 16, Issue 2, (March/April 1999), 21-29

[13] Houston, D. An Experience in Facilitating Process Improvement with an Integration Problem Reporting Process Simulation. Software Process Improvement and Practice 11,4 (Jul 2006), 361 – 371.

[14] ISO/IEC 12207:1995, Information technology – Software life cycle processes, ISO/IEC 1995.

[15] ISO/IEC 15288:2002, International Standard, Systems engineering – Systems life cycle processes, ISO/IEC 2002.

[16] Larsson, S., Crnkovic I. and Ekdahl F. On the Expected Synergies between Component Based Software Engineering and Best Practices in Product Integration, Euromicro Conference, France, (2004), 430-436

[17] Larsson, S. and Crnkovic, I. Case Study: Software Product Integration Practices, PROFES 2005 Conference, Oulu, Finland, (June 2005), 272-285

[18] Larsson, S. Improving Software Product Integration. Licentiate Thesis, Mälardalen University, Västerås, Sweden, 2005

[19] SCAMPI Upgrade Team, Appraisal Requirements for CMMI, Version 1.2 (ARC, V1.2). Technical report CMU/SEI-2006-TR-011, Pittsburgh, PA, USA, 2006

[20] Yin R. K., Case Study Research: Design and Methods (3rd edition), Sage Publications, Thousand Oaks, CA, USA, 2003

Paper D

HOW TO IMPROVE SOFTWARE INTEGRATION

Stig Larsson, Petri Myllyperkiö, Fredrik Ekdahl, Ivica Crnkovic
Submitted to Information & Software Technology Journal

Abstract

In software-intensive systems the integration becomes complex since both software and hardware components are integrated and run in the execution environment for the first time. Support for this stage is thus essential. Practices for Product Integration are described in different reference models. We have investigated these and compared them with activities performed in seven product development projects.

Our conclusion is that current descriptions of best practices in product integration are available in reference models, but need to be merged. Through case studies we see that the described practices are insufficiently used, and that organizations would benefit from adhering to them.

1 Introduction

Integration of software products, as well as products that include software, is described in several standards and other collections of best practices, i.e. reference models. Even if product integration process is a part of many development process models and included in the iterations, the process is in many cases an isolated and recognizable process. In particular, the development of complex systems in a distributed environment where components are developed in different locations and even companies requires a coherent, comprehensive and distinguished description of that phase.

These best practices are compiled from experiences from different companies and organizations. Practices are selected as they are considered to increase the effectiveness and efficiency as well as contributing to the quality of the product. The source for the described practices is collective experiences and the resulting documents are rarely validated through independent research. This article is an attempt to investigate product integration practices, compare the reference models with experiences from different development projects, and to aim at a first step of validation.

In order to define the context and scope for this paper, we use the definition of integration for product and system development found in the glossary of EIA 731.1 (interim standard) [7]:

”Integration: The merger or combining two or more elements (e.g., components, parts, or configuration items) into a functioning and higher level element with the functional and physical interfaces satisfied. “

In spite of the existence of many reference models the problems in integration persists. Experiences from many organizations developing products tell us that product integration often is where problems occur [3]. This is especially true for software which is a part of a larger system [24]. The difficulties found during integration includes mismatch between the different components, problems with properties of the system (e.g. performance, response time) that are observable first after integration, and problems in other parts of the system than the one that was changed or added. To better understand of this problem we have analyzed the integration processes in two companies and a total of seven product development projects from different organizations within the companies. As a starting point for our study we have stated the following questions:

- How are the practices described in reference models useful for product development units for improving product integration?
- What is the core set of practices that can be identified to reduce problems in product integration?
- Is it appropriate to combine reference models to provide better support to product development units, and how can this be done?

Our approach to these different perspectives is to study the performance of the process in the investigated organizations and compare the activities with the ones prescribed in the reference models regardless of the development model used. We also look at the problems in the organizations and analyze these with respect to the practices that are not followed by the organization.

We claim that we by investigating a number of projects and the practices in use have been able to determine to which extent the practices described in reference models are useful as a support for development units. We also claim that we through the investigation can identify needs for revisions of the reference models.

Our proposition in this paper is that the problems encountered in the investigated cases relate to the lack of execution of practices that are described in the reference models. We also propose that successful execution of the product integration can be mapped to specific implementation of practices described in the reference models.

However we find that no reference model provides a full support for the product integration, although by combining them a significant improvement can be made. For this reason we propose a more complete, consistent and integrated combination that the reference models should be updated to include the union of practices described.

The remainder of this paper is outlined as follows: Section two refer to related work. Section three describes the research method used in the studies. Section four provides an analysis of product integration included in different reference models. Section five describes the case studies and presents the data from these. Finally in section six we discuss the results of the studies, give conclusions, and propose further research based on the found results.

2 Related Work

Product integration processes are included in different established reference models for product development such as ANSI/EIA -632 [2], EIA-731.1 [7], ISO/IEC 12207 [15], ISO/IEC 15288 [16], and CMMI [27]. The can be used as a source for assessments and process improvement planning. Reference models are normally articulated as a set of requirements on the development, and not as a set of activities to be performed, giving the organization possibility to implement the process as suitable. One concern with reference models is the inadequate validation of them; it is difficult to find research looking into the validity of the content of these reference models. However, the reference models are developed based on experience from a large number of organizations, making it likely that important considerations are taken into account.

Stavridou has examined product integration from two different perspectives. In [29], integration standards for critical software intensive systems are investigated. The examination focuses on military policies and standards, but includes ISO/IEC 12207 in the comparison. The conclusion is that the majority of the examined standards address integration testing, but that the standardization is not appropriate for many integration issues, and that additional guidance for the project manager is needed. A more technical approach is selected in [30] where the integration is proposed to be considered as a design activity.

The activities in the product integration area have also been the subject of interest from the agile community where frequent builds is one of the cornerstones. One example is Fowler [9] who describes the requirements on

developers: before committing back to mainline the developer would need to update his work area with the latest mainline, i.e. build against the latest changes of other developers. Only after that, integration into the mainline would be permitted.

Schulte [26] describes the integration of large systems as a challenge and proposes a method for handling uncertainties in the resulting system characteristics when integrating components. De Jonge [5] identifies that the integration of components is more difficult when reusable building blocks are applied and propose techniques that promote fine-grained software reuse. In [4], Chittister and Haimes argues that the system integration process is additionally complicated as software is included, and proposes that the risk factors along the software development life-cycle must be identified and through measurements understood and ultimately mitigated.

A more general description of system and product integration has been made by Sage and Lynch [25]. The paper describes various views including lifecycle, architecture, process, interfaces, and enterprise integration. The role of architectures and integration in different reference models are also described. One conclusion is that methodologies and tools for system integration and integration architectures are not well described in literature at that point in time.

In addition to use reference models to improve product integration, metrics can be used. Houston [13] studied the integration problems occurring in an avionics system. Types of integration problems were identified based on the functionality where the problem occurred. By classifying the integration problem reports and estimating the expected handling time through simulations of the report types several possible SPI activities were identified. The study complements our research as one alternative route to find improvement activities.

The term Product Integration can also be used for the combination of systems from an architectural standpoint. The approaches have been outlined by Land and Crnkovic [18] and include component-based software, open standards, and Enterprise Application Integration.

Four different aspects of systems integration is described by Nilsson et al in [23]. The paper addresses the technical characteristics of integration and address integration technology, integration architecture, semantic integration and user integration. The main message from this standpoint is that systems integration is difficult and complicated and that there are no obvious shortcuts.

Kuhn concentrate in [17] on effective use of standards for interfaces when integrating systems, and describes a methodology to application development that focus on an architectural approach.

Component-base software engineering is also considered to simplify the integration if taken to the extreme. In [6], Dogru describes a fully component-oriented approach and put this in contrast with modifying object oriented approaches, stressing that CDB leaves out inheritance and capitalizes on composition. However, the lack of tools and experience is currently preventing full use of the presented ideas.

This research in this article build on the work published as separate case studies. Case 1 is described in [21] and as case 1 in [22], Case 2, 3 and 4 in are described as case 1, 2 and 3 in [20], and Case 6, 7 and 8 as case 2, 3 and 4 in [22]. An early version of the summary of the reference models described in section three can be found in [19].

3 Research methods

Our study includes experiences in seven product development projects from five organizations in two companies. The products developed in these organizations include applications such as manufacturing industries, process industries, telecommunication, power distribution, and power transmission. Both companies are multinational with development in many countries. The experiences from the investigated product development organizations are compared to and classified according to a set of standards and models, i.e. reference models. This is done through three activities: investigation of reference models, data collection, and a mapping between the reference models and the data.

Two types of reference material, standards and models, have been considered in this study and are referred to as reference models. The difference between the types is that standards have been approved by a standardization body, while a model may be issued by any company or organization. The included reference models are typically used by product development organizations to obtain a common language, to ensure that the development performed covers necessary activities, to guide improvement activities, and to show compliance. The selection of reference models is based on available information from standardization organizations such as ISO[14], ANSI[1] and IEEE[11] and references from organizations such as SEI[28] and INCOSE[12]. Based on the focus of our research, product

integration in product development of products that include software, two specific selection criteria have been used in the choice of reference models:

(i) The reference model should be relevant to product development of products that include software.

(ii) The reference model should include requirements on product integration, implicitly or explicitly, as this is our research area.

The standards provided by the listed organizations have been evaluated based on both. The reference models that have been selected are:

- ISO/IEC 12207 Information technology - Software life cycle processes [15]
- EIA-632 Processes for Engineering a System [2]
- CMMI Capability Maturity Model Integration [27]
- EIA-731.1 Systems Engineering Capability Model [7]
- ISO/IEC 15288 Systems Engineering – System life cycle processes [16].

Also ISO 9001 [13] and IEEE Std 1220-2005 [10] were considered, but for both these standards the expectations on the product integration process are limited, and hence, they have not been further analyzed. It should also be noted that efforts are made within the standardization bodies to harmonize several of these reference models such as IEEE Std 1220-2005 [10], EIA-632 [2], ISO/IEC 15288 [16] and ISO/IEC 12207 [15]. New versions of ISO/IEC 15288 and ISO/IEC 12207 are planned to be published third quarter 2007.

To accommodate the comparisons between the different reference models practices related to product integration has been extracted and listed in Table 2. This is done through a detailed review of the reference models based on the following definition of product integration found in CMMI [27]:

> “The scope of this process area is to achieve complete product integration through progressive assembly of product components, in one stage or in incremental stages, according to a defined integration sequence and procedures. Throughout the process area, where we use the terms product and product component, their intended meanings also encompass services and their components.

> A critical aspect of product integration is the management of internal and external interfaces of the products and product components to ensure compatibility among the interfaces. Attention should be paid to interface management throughout the project."

The definition in CMMI has been selected as it explicitly defines the scope of product integration. In the table, the practices described in all reference models have been combined and expressed in a generic way.

For the data collection, four main questions have been formulated. Based on these the questions for the interviews have been expressed. The four main questions are

- How is the preparation for product integration performed?
- How are interfaces to and in the product managed?
- How is the actual integration of the product performed?
- What types of problems have been observed in relation to the product integration?

The data collection is based on interviews and document reviews. The method used for the data collection is described by Ekdahl and Larsson in [8] where also examples of types of observations can be found. For three of the cases (cases 5, 6, and 7), data from the execution of the process has also been collected from the production integration process.

The interviews have been based on a set of questions derived from one or several of the reference models selected for this study. The format of the interviews has been a discussion about the integration process. During this discussion, the researchers have monitored that all questions are covered. Document reviews were 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 product integration process. Besides an understanding of how the process is performed, information about the documented as well as perceived problems related to the product integration process was captured. The information collected from the product integration process in Case 5, 6, and 7 are problems and failures in the build, smoke test and regression test activities, and is further described in [22]. The primary data from builds and tests has been collected by the practitioners and compiled by the researchers.

The final activity in our research has been to map the findings from the interviews to different reference models. A first mapping from the cases to

the reference models was made to find out what practices were performed in each organization. This was done for each practice through searching the collected material for evidence that the practice was performed. This way, all the practices were covered and additional information about the organization besides the practices were captured.

A second mapping was made to understand how the problems found in each case relate to the practices. This was made for each problem through searching the collection of practices for a match. Problems that could not be related to any product integration practices were noted and discussed, but are not used further in this study.

Care has been taken to ensure that all classification is determined by two different researchers. In cases where the researchers have had different conclusions, a discussion has been held to clarify the different opinions, and an agreement is sought. If impossible, this has been clearly indicated in the presentation of results as being undetermined.

4 Practices in standards and models

Each of the selected reference models is in this section described starting with its purpose and intention and continued with details on the description regarding product and software integration processes. The actions and tasks considered to be related to product integration are summarized. Note that these summaries are for information purposes only, and that the original text in the reference models should be used for any implementation.

Based on the acquired knowledge regarding the reference models, a summary of practices and a comparison between the models has been made. The purpose has been to see if there is a set of practices consisting if the union of the used reference models.

The first step was to combine the extracted information from all investigated reference models into a set of practices. After that, all reference models were investigated based on the set of practices. Both explicit and implicit instances of the practices were noted. This classification is relying on the experience and knowledge of the researchers. However, through the stepwise approach, the risk for missing information is less as each reference model is examined twice.

4.1 ISO/IEC 12207, Information technology – Software life cycle process

The purpose of ISO/IEC 12207 is to provide the software industry with a well-defined terminology for software life cycle processes [15]. It contains the different processes, activities and tasks that make up a software life cycle, and applies to the development, operation and maintenance of software products as well as to acquisition and supply of software products, systems and services.

ISO/IEC 12207 includes two parts related to product integration. The first is covering the integration of software units or components into software items that can be integrated into a system. The tasks described are: to develop and document an integration plan for each software item that has been identified in the system architectural design, to integrate and test the aggregates as described in the plan, to update the user documentation and to develop and document a set of tests for each requirement of the software items. The standard also lists a number of criteria that should be used for evaluation of each work product developed in the software integration process as well as a requirement to conduct joint reviews. Note that the update of user documentation is omitted in this investigation as it is not considered to be a part of product integration.

The second part describes the system integration tasks. These are: to integrate the software into the system and to test the requirement of the system. There is also a list of criteria for evaluation of the integrated system.

4.2 EIA-632

The purpose of the EIA-632 standard [2] is to provide developers with fundamental processes that assist in engineering a system. In this context, a developer can be an enterprise or an organization. The use of the standard should help developers to develop requirements that enable delivery of system solutions in a cost-effective way, delivering within cost, schedule and risk constraints and to provide a system that satisfies the different stakeholders over the life-cycle of the products that make up the system.

The integration of parts into products is included in the requirement for implementation. The implementation practices include expectations, that the developers should plan for and execute tasks such as validating the subsystems received for assembling and assembling validated subsystem products into the test items or end products to be verified.

4.3 Capability Maturity Model Integration (CMMI), Version 1.1

The Capability Maturity Model Integration (CMMI) from the Software Engineering Institute describes what is considered as best practices for product and systems engineering [27]. The model includes process areas covering the full product life cycle for the development and maintenance of products and services. The purpose of the model is to provide a basis for process improvement, and includes guidelines for how to select improvement areas.

For each of the process areas described in CMMI, a purpose is described. For Product Integration it is "to assemble the product from the product components, ensure that the product, as integrated, functions properly, and deliver the product". It is detailed in three goals which are supported by a total of nine practices that are specific for product integration. The goals are: Prepare for product integration, Ensure interface compatibility and Assemble product components and deliver the product.

4.4 EAI-731.1

The purpose of EIA-731.1 (interim standard) is to support the development and improvement of systems engineering capability [7]. It is structured to support different activities performed to improve the performance in a development organization such as appraisals, process improvement, and process design.

Product integration is described in the section Integrate System which describes practices connected to product integration strategy, interface coordination, integration preparation and system element integration.

4.5 ISO/IEC 15288, Systems engineering – system life cycle processes

ISO/IEC 15288:2002 is intended to describe the life cycle of systems [16]. The standard is to be applied to the full life cycle of systems from inception, development, production, utilization, and support to retirement of the system. It is noted in the standard that the implementation typically involves a selection of a set of processes applicable for the project or organization.

Product integration is described in the section Integration Process. The purpose with this process is to assemble a system that is consistent with the

architectural design. System elements should be combined to form partial or complete products. The activities includes definition of a strategy for integration, identification of design constraints based on the strategy, preparation of facilities that enable the integration, reception of validated system elements in accordance with a schedule and the actual integration. In addition, there is a requirement to store information about the integration into an appropriate database.

ISO/IEC 15288:2002 introduces a requirement that the constraints from the integration strategy on design should be identified. This requirement is not represented in any of the other standards, and is not investigated in the case studies. However, we believe this is an important area that needs to be further investigated as it is closely related to the requirements on how interfaces are handled.

4.6 Summary of reference model practices

Table 2 summarizes the product integration process as described in different reference models and provides a basis for comparison. In this section, we have extracted practices described in each reference model and combined each of these practices. The combination of the selected reference models has given us 15 different practices that have been expressed in a generic way. These have been selected as they are directly related to product integration. Other practices mentioned in the context of product integration in the different reference models have been excluded. Examples of this are CMMI, Specific Practice 3.4, “Package the assembled product or product components and deliver it to the appropriate customer”, and ISO/IEC 12207, Section 5.3.8.3 “The developer shall update the user documentation as necessary”. The excluded practices are important, but have been considered to be only tangentially related to product integration as defined for this paper.

The proposed practices for each of the activities are described below, and further guidance can be found through pointers to the reference models in Table 2. Note that the practices are not ordered in the sequence that they are expected to be performed. Most of the activities described should be carried out in an iterative manner. However, our advice is to ensure that Product Integration (PI) Practice 1 is performed early in any product development project to set the expectations on the remaining practices.

PI Practice 1. Define and document an integration strategy

Develop an integration strategy and supporting documentation that help in identifying a sequence for the product integration satisfying the requirements while minimizing risks. The documentation should include different considered strategies, and the rational selecting one. The strategy should be based on factors important for the product development such as the need for partial systems, verification and validation strategies, organizational arrangements and selected architectural solutions. As the development proceeds, the strategy should be reviewed periodically to ensure that the basis for the decision is still valid.

Examples of strategies are to start with platform functions to simplify the addition of many applications in parallel, to ensure early customer functionality to be added early enabling demonstrations, or to have continuous integration of product components as they become available. Each of these strategies has advantages and drawbacks that must be understood. The strategy will affect the planning for the whole project, and is essential for the understanding, planning and preparation for product integration.

PI Practice 2. Develop a product integration plan based on the strategy

The product integration plan should define the integration steps as an assembly sequence, the procedures to be used for integration, the integration verification to be performed, resources, and responsibilities. Based on the selected strategy, the plan may include alternate sequences to minimize risks and prepare for different scenarios. The plan should be reviewed periodically and updated based on new information and risks as needed.

PI Practice 3. Define and establish an environment for integration

The product integration plan will together with the product requirements provide requirements on an integration environment. The definition and establishment of the environment can be reused from organizational resources, developed, or acquired. In either case, the requirements and plans for the environment need to be considered in parallel with the development and integration plans.

The integration environment typically consists of simulators, stubs, test equipment, parts of existing components and products, software and hardware tools, and measurement equipment.

PI Practice 4. Define criteria for delivery of components

The criteria defined for delivery of components should be selected so that they can indicate the readiness of a component for integration. The criteria should address what type and level of verification should be performed on the component and interfaces as well as thresholds of the verification results for acceptance.

PI Practice 5. Identify constraints from the integration strategy on design

Several factors can be considered when identifying the constraints that a specific integration strategy may impose on the design. These include rules for what types of interfaces should be available to enable interconnection between components at different stages of the integration. Also the necessity to use simulation, stubs and need to be considered as this may require specific solutions. The environment used for the integration may require that the design is constrained.

Typically, the constraints on architecture and design due to a specific integration strategy require a revision of the architectural and design documentation.

PI Practice 6. Define interfaces

The efficient and effective execution is depending on interfaces that are agreed and used for the different components. This includes physical, functional, and logical interfaces. When interfaces are determined and defined, an agreed set of criteria should be followed. The criteria typically expose attributes important for a specific application, and may include parameters reflecting the requirements on dependability, performance, safety, evolvability and other quality attributes. Once determined based on the evaluation of different criteria based on criteria, the interfaces should be documented and put under configuration management. This documentation should include the rational for the selected definition and design. The interfaces are typically characterized through the source, destination, control and data characteristics for software, and electrical and mechanical characteristics for hardware. Also human interfaces and environmental parameters should be addressed and documented.

Early definition of interfaces reduces the risk for mismatch between product components that are developed in parallel. The drawback is that knowledge is acquired as the implementation of the product components progresses; additional interfaces may be needed, as well as modifications to existing ones.

PI Practice 7. Review interface descriptions for completeness

When interfaces are defined and revised, appropriate stakeholders need to perform a review to ensure that each interface description is complete and fulfill the intentions as described in the requirements. This is however not sufficient. As the product components are developed, there is a need to review the interface descriptions periodically to ensure that they are sufficient and understood by all stakeholders. These reviews should also provide input to proposed interface changes.

To facilitate proper reviews, interfaces categories can be defined. The definition of the categories can be used to decide on what needs to be document for each category. Once established, the categories can be used to organize the interfaces, and made available to relevant stakeholders.

PI Practice 8. Ensure coordination of interface changes

Interfaces affect different stakeholders, and the changes must be controlled to reduce the misunderstanding as well as late discovery of mismatch. Changes should be controlled for different types of interfaces, e.g. between product components, to the environment, to users, and verification equipment.

Change Control Boards can be set up to control changes to interfaces. This is critical for projects that have external suppliers, or depend on other parts of the organization. The responsibility of the CCB goes beyond deciding on changes; consequences should be investigated before decisions are made, relevant stakeholders should be involved, and information regarding decision on changes should be communicated, and the interface documentation updated as appropriate.

PI Practice 9. Review adherence to defined interfaces

As the product component is to be delivered, compliance to the interface documentation should be reviewed and verified. The criteria used for definition of the interfaces can be used as support for the review.

A review of interface adherence may be done in a common session for product components using a specific interface. This enables the development teams to agree on any mismatch and decide on changes to one or several of the components, or a proposal to change the interface.

PI Practice 10. Develop and document a set of tests for each requirement of the assembled components

The requirements considered for the integration tests are the ones related to interfaces and interaction with other components and the consistency with the architectural design.

PI Practice 11. Verify completeness of components obtained for integration through checking criteria for delivery

Each product component to be integrated must be identified as being the intended one in the right version. The completeness of a component can only be confirmed through checking defined criteria. If a component does not fulfill the criteria appropriate measures should be taken; changes may have to be made to the component, or the deficiency can be accepted temporarily or

The responsibility to ensure that a product component meets the defined criteria can be decided in the strategy for product integration. Typically, the developer or development team is responsible to develop the product component in accordance with all requirements, including the criteria for integration. However, it is also common that the integration team is responsible to check are that the criteria are met, and to reject the delivery of components not adhering to the requirements.

PI Practice 12. Deliver/obtain components as agreed in the schedule

As the product components are verified as complete as defined by the product integration delivery criteria, they can be delivered for integration. It is of utmost importance that any slippage in the agreed schedule for the delivery of a component is communicated as soon as it is know, or even as soon as the risk for late delivery is identified. Any delays may affect the integration sequence, and the possibility to provide different stakeholders with intermediate integrations, and with the final product.

The acknowledgement of reception for integration is important, and can be made through an informal or formal handshake procedure. An example of this is to use the configuration management status information to set the product components in different states. This enables relevant stakeholders to get an understanding of the status, and bottlenecks can be identified.

PI Practice 13. Integrate/assemble components as planned

The integration and assembly of the components should be performed as described in the product integration plan. The integration can be made in steps, with aggregates of components being built consecutively, and it may be necessary to perform evaluation activities on the intermediated results. The result of the assembly should be made available to all relevant stakeholders.

PI Practice 14. Evaluate/test the assembled components

The focus when evaluating the assembled components is on interface verification. The defined and described procedures and environments are used to ensure that the product components work as intended when combined. The results from the verification should be recorded and appropriate action taken to handle any issues that may occur.

PI Practice 15. Record the integration information in an appropriate repository

When the integration is performed, it is necessary to record information regarding problems in the product, product components, integration environment, and in procedures for integration. The information can, besides a control of necessary changes to the product and product components, be used to further improve strategies, practices, environment, and process improvements for product development processes that are delivering to the product integration.

Examples of information that can be collected are problems in the integration related to different practices, e.g. build statistics [22].

The list of activities can be used as a guideline for the definition of a product integration process and process improvement in the area. Note that if a reference model is implemented, the original text for that specific reference model should be used. Three different types of indications have been used in Table 2. E is used if the practice is explicitly described in the reference model, I if it is implicitly described and a – if it is not described. Implicit descriptions are identified if there is a generic statement that the type of activity, such as reviews, should be performed. If a practice is covered both explicitly and implicitly, only the explicit occurrence is mentioned in the table. A pointer to the reference model is given for each explicit or implicit description of the practice. The references in the table are numbers of sections, practices, or requirements as defined in Table 1.

Table 1. References for the different reference models

Reference model	Reference
ISO/IEC 12207	Section
EIA-632	Requirement
CMMI	Specific Practice in the Product Integration process area
EIA-731.1	Specific Practice
ISO/IEC 15288	Section

Table 2. Product integration process in selected reference models

Reference models	ISO/IEC 12207	EIA-632	CMMI	EIA-731.1	ISO/IEC 15288
Publication date Generic activity description	Aug 1995	Jan 1999	Mar 2002	Aug 2002	Nov 2002
1. Define and document an integration strategy	-	I Req 32 a	I PI SP 1.1	E SP 1.5-1-1 SP 1.5-1-2	E 5.5.6.3a
2. Develop an integration plan based on the strategy	E 5.3.6 5.3.7 5.3.8	E Req 32 a	E PI SP 1.1	E SP 1.5-1-3a	E 5.5.6.3a
3. Define and establish an environment for integration	-	I Req 32 a	E PI SP 1.2	-	E 5.5.6.3c

Reference models	ISO/IEC 12207	EIA-632	CMMI	EIA-731.1	ISO/IEC 15288
Publication date Generic activity description	Aug 1995	Jan 1999	Mar 2002	Aug 2002	Nov 2002
4. Define criteria for delivery of components	I 5.3.8	I Req 32 a	E PI SP 1.3	I 1.5-3	I 5.5.6.3e
5. Identify constraints from the integration strategy on design	-	-	-	-	E 5.5.6.3b
6. Define interfaces	E 5.3.4 5.3.5 5.3.6	E Req 16 b Req 17 b	E TS SP 2.3	E SP 1.3-1-1c SP 1.3-1-3a SP 1.5-2-3a, 3b, 3c	I 5.5.4.3g
7. Review interface descriptions for completeness	I 5.3.5	I Req 12 d	E PI SP 2.1	E SP 1.5-2-2a, 2b	I 5.5.4.3g
8. Ensure coordination of interface changes	-	I Req 12 d	E PI SP 2.2	E SP 1.5-2-1a	I 5.5.4.3i
9. Review adherence to defined interfaces	-	I Req 12 d	E PI SP 2.2	E SP 1.5-3-1a	E 5.5.6.3f

Reference models	ISO/IEC 12207	EIA-632	CMMI	EIA-731.1	ISO/IEC 15288
Publication date Generic activity description	Aug 1995	Jan 1999	Mar 2002	Aug 2002	Nov 2002
10. Develop and document a set of tests for each requirement of the assembled components	E 5.3.6 5.3.7	E Req 32 a	I PI SP 1.3	I 1.6-2	I 5.5.7.3e
11. Verify completeness of components obtained for integration through checking criteria for delivery	E 5.3.8	E Req 3 b Req 20 b	E PI SP 3.1	E SP 1.5-3-1a	E 5.5.6.3e
12. Deliver/obtain components as agreed in the schedule	E 5.3.8	I Req 20 a	I PI SP 3.1	E SP 1.5-3-2	E 5.5.6.3d
13. Integrate/assemble components as planned	E 5.3.8 5.23.10 6.4.2	E Req 20 c	E PI SP 3.2	E SP 1.5-4-1a	E 5.5.6.3f
14. Evaluate/test the assembled components	E 5.3.9 6.4.2	E Req 20 d Req 32 b	E PI SP 3.3	E SP 1.5-4-1b	E 5.5.7.3e
15. Record the integration information in an appropriate repository	I 5.3.9 5.3.10	-	I PI SP 3.3	I 1.5-4	E 5.5.6.3g

4.7 Similarities and difference between the reference models.

A comparison of the standards based on the PI practices show that there is an on-going development of the area and an increased agreement over time on what can be considered to be best practices. The following observations have been made:

- Integration planning is expected in all reference models
- Only ISO/IEC 15288:2002 mention the aspect that the integration strategy may imply constraints on the system or product design
- Interface definition is explicitly specified in all reference models except ISO/IEC 15288:2002, but other aspects of interface management such as review and control of changes are only specified in CMMI and EIA 731.1
- The verification of completeness as well as the actual integration and verification of the assembled components are included in all reference models.

The comparison between the different reference models indicates that expectations on the preparation for integration and the handling of interfaces have been made more explicit over time; additional practices are added and already existing practices are made more precise for reference models released at later dates.

Older standards are less explicit regarding product integration, while newer focus on different aspects. As EIA has been used as an input to the development of CMMI, there is no surprise that they are handling product integration in a similar way. ISO/IEC 15288 has the best coverage of product integration except for management of interfaces.

Our conclusion is that additional investigations and comparisons are needed to understand how the area evolves, what factors are determining what is added to the reference models and if there are specific considerations that should be made for different types of products and systems. There is also a need to validate the changes that are made through case studies in different types of product development organizations.

5 CASE STUDIES

In order to understand if the reference models can help organizations reduce the problems in product integration as executed in an industrial environment, we have examined seven different projects. All of the projects had the task to develop products used in the manufacturing, process, telecommunication, or power domains. This section describes the projects and products for each case. One notable characteristic is that the projects in both companies are to a certain extent independent in their selection of work processes and supporting tools. This is a strategic decision based on the diverse needs from different types of development and products.

In each of the cases we have captured the problems appearing in product integration. A problem is a reoccurring reason for failure in the integration process. This includes problems in the build, smoke test, and regression testing. For all cases problems have been captured in the interviews and document reviews. In cases five through seven, measurements from the build and integration test phases have been added as a source for finding problems and their causes.

5.1 Case 1

This study was performed at a 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.

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

Table 3 describes the three problems found in case 1. All are related to the coordination: functions are not delivered as promised, functions are not tested before delivered, and changes of common resources are not controlled. This organization is now putting more effort in place to ensure that functions are tested before delivered and to control interface changes.

Table 3 Problems captured for Case 1

Label	Problem description
1A	Functions are not always delivered in time for integration or may be incompletely delivered. In addition to this, delivery is complicated through two different ways to deliver code. This leads to problems in the build process or in integration and system tests
1B	Functions are not tested as required by the developers. This leads to problems in the builds and in initial integration testing.
1C	Changes in common resources (e.g. common include files) are not controlled. This results in errors appearing in other components which have not been changed

5.2 Case 2

The product in case one is a stand-alone product that is connected to a real-time data collection system. The development is done in one group with less than 20 developers and follows a clearly defined process.

The product development of a specific release is based on a definition of the product that contains what should be included in each release. The first step in the development is the implementation of requirements on the functions for the release. Based on this, the unit and system verifications to be performed are defined. Development of the functions is done in units called components. The Rational Unified Process is used, and a document list defines the development process. The planning is made so the development is done in increments. The unit verification is performed by software developers. The strategy is that tests should not be done by the developer producing the software. The unit tests are often done through automatic testing. Specifications and protocols from the tests are reviewed by peers and system integrators. The tests are performed in the developer's environment and consist of basic tests. Functional tests are performed before the system tests.

The product integration is not defined as a separate process, but the product is integrated by the developers before the system verification. Before a component is checked in, it should be included in a system build to ensure proper quality. Delivery to the system test is done of the whole system. The test protocols and error reports from the unit verifications are reviewed with the system integrator before the system test. The system tests are performed

by a core of system testers and temporary additional personnel. This strategy builds on well defined and detailed tests. The tests are focusing on functions and performance and are performed on different hardware combinations. This includes different variants of the product and different versions of the operating system. The test period takes approximately 12 weeks, with new versions of the assembled components received to system test every week. Although the development builds on increments, no integration plan is used for the product. The integration plan used is one for the whole system where this product is included. Typical time for the development of a release is less than one year.

The three problems captured for Case 2 are described in Table 4. The routines are mainly followed, but due to tight deadlines, shortcuts may be taken. Sometimes uncontrolled changes are introduced in the software. This is typically done when a part of the system is changed due to an existing error that is uncritical and not planned to be corrected. Due to the dependencies in the system, new errors may appear in parts that have not been changed. Also other connections between components that are not explicit generate this problem.

Table 4 Problems captured for Case 2

Label	Problem description
2A	Functions are not always fully tested when delivered for integration. This leads to problems in the build process or in integration and system tests
2B	Errors are corrected that should not be. This results in new errors with higher influence on functionality and performance
2C	Errors appear in other components which have not been changed

5.3 Case 3

The third case is a product that includes software close to the hardware. The target system includes a complex hardware solution with the application divided on two target systems.

The development group is small and follows a common development process. This process includes rules for what should be checked and tested before a component is integrated. The tests include running the application in simulators and target systems before the integration. A specification for

what should be ready before start of functional and system test are available. The architect is responsible for implementation decisions. Typical time for the development of a release is 1.5 years. This includes the full development cycle from defining the requirements to system testing.

Most of the problems appear because of the incapability and version mismatch of the test system, the final product and the test and final hardware platform (Table 5). Efforts are now made to go towards incremental development, and to increase the formalism in the testing. The tests will be made in three stages with basic tests performed by the designer, functional tests performed by a specific functional tester and system tests with delivery protocol.

Table 5. Problems captured for Case 3

Label	Problem description
3A	Problems appear as a consequence that tests for the components are not run in the same environment as the test system. Different versions of hardware and test platform are used.

5.4 Case 4

The development organization in this case is responsible for the design of a user interface that acts as a client to a database server. The organization is small, around 15 developers, and most developers participated in the investigated project.

The current architecture has in recent years been improved. The old version of the system suffered from problems with many common includes files. Through global variables and similar solutions permitted by the selected technology, unintended side-effects made debugging and error correction tedious. Different attempts to reduce the problems within the available technology lead to the insight that a design that was built on isolation of interfaces should be beneficial. The solution was to start building a new system. Included in this decision was a strategy to design interfaces carefully and to use technologies that permitted isolated components to be used.

The system is built up of components that primarily implement different parts of the user interface. Each component handles the communication with the server. This design was used to allow the development of services that are independent and dedicated for each component. The component framework defines the required interface for each component and provides a

number of services, such as capturing of key strokes. The technology used permits the developers to easily isolate problems and to minimize the uncontrolled interference and dependencies between the components.

The development is organized with frequent builds and continuous integration of new functions. The integration is handled by the integration responsible. However, the checks before the inclusion of new functions are done by the developers. There are no specific routines in place for handling the interfaces. Changes are in practice always checked by the system architect.

The new system design has reduced the implementation time for a function with 2/3. The turn-around time for a system release has been reduced from six months to between one and three months. At the same time, a need for maintaining the base platform has emerged. Also, some of the technical solutions have been questioned and may increase the need for maintenance (Table 6).

Table 6. Problems captured for Case 4

Label	Problem description
4A	Scattered architecture on the server side as a result of the decision to handle communication in each component

5.5 Case 5

The organization in case 5 develops a complex real time control product including event, trend and error handling, data collection, communication, and operator interface. The product is part of a suite of about 30 products, forming a system that is used in process industries. The development is tightly coordinated with the development of these other products. The organization is the largest in the investigation and the number of developers involved is close to 80. The development process varies between different groups in the organization, but all parts are delivered to the build and integration process.

The product consists of more than three MLOC and consists of applications on a workstation. The architecture is distributed and care has been taken to define different layers to achieve separation-of-concerns.

The project that has been investigated is the integration project for a major release program. The integration project consists of a build team and a team responsible for the automated regression tests. These two teams have possibilities to capture integration problems and to analyze background

problems. The builds are performed as daily builds, weekly builds and baseline builds. The daily builds are performed to ensure the stability of the code base, and have the build environment adapted to additions in functionality. Weekly and baseline builds are made available to other parts of the organization for verification purposes. Development cycles are between 12 and 18 months.

A primary problem for this organization has been the complex build and integration environment, which is reflected in the identified problems as described in Table 7.

Table 7. Problems captured for Case 5

Label	Problem description
5A	Inconsistent code is delivered, and files are not included in the build as planned. The result is failed builds.
5B	The build environment sometimes contain traces of earlier builds or fail due to unstable applications used, resulting in failing builds.

5.6 Case 6

The project develops embedded software for new generation of protection relays for electrical network. The development organization is small with around 15 developers. The main development team is located in one site, but few persons from other sites have participated in the development partially.

At the time of the study the project size was around 500 KLOC, however the final products built on the technology will be larger than that.

The product architecture consists of base software, application components and higher level services, such as communication services. The base software is a platform that provides the running environment for the application components and services, and separates them from the hardware. The software is built of source-code level components that are compiled and linked to a single executable binary. Most of the components are developed by the organization but also some 3rd party components are used. The project is responsible of developing the base software and integrating it with the service and application components, which are partly developed in separate projects. The outcome of the project is a reference configuration which will be the baseline for the actual productization projects, including base software, service components and reduced subset of application

components. Thus, the studied integrations include all types of the components used in the product.

Table 8 summarizes the problems for the organization in Case 6. A major problem for this organization is that changes are introduced without proper testing. Also late deliveries of functions are a problem as the functions often consists of sub-functions that are combined in the integration.

Table 8. Problems captured for Case 6

Label	Problem description
6A	Changes are sometimes untested before integration, resulting in errors in initial integration testing.
6B	Files are not delivered to integration as planned and required, resulting in errors in the build process.
6C	The build environment contains sometimes traces of earlier builds, resulting in failing builds.
6D	Changes are made to interfaces without proper control. This leads to errors in the builds or initial integration testing.

5.7 Case 7

The project develops embedded software for a new generation of protection relays for electrical network. The development organization is small with around 15 developers. The main development team is located in two sites in two different time zones.

At the time of the study the integrations were performed only to the components of base software and service components, while as the application components were not yet included in the integration. The project size was over 500 KLOC, but when integrated with the application components, is expected to be >1000 KLOC.

The product architecture resembles that of Case 6, i.e. it consists of base software, application components and higher level services. The product has some common components with Case 6, but it is based in other hardware. In addition some components have a longer history, being released in the existing products. The software is built of source-code level components, either developed by the organization or 3rd party, compiled and linked to single executable binary. The project is responsible of further development of the base software and integrating it with the service and application

components, which are partly developed in separate projects. The outcome of the project is a new product.

For this case, three problems were found as described in Table 9. These are problems in the build environment, problems appearing in parts of the system that has not been changed, and that functions sometimes are untested when delivered for integration.

Table 9. Problems captured for Case 7

Label	Problem description
7A	Functions are not always fully tested when delivered for integration. This leads to problems in the build process or in integration and system tests
7B	Untested changes to build scripts
7C	Errors appear in other components which have not been changed

5.8 Problems in relation to reference models

The problems found in the cases have in this section been related to the reference models. This gives an indication on how the reference models can help in avoiding the types of problems found. If a problem can be related to a practice that is not performed, but is described in the reference model, there is a possibility that the practice could help if implemented. However, if there are problems that we can relate to product integration, but cannot be related to the practices for a specific reference model, following that specific reference model does not support the organization in preventing that type of problem. The purpose of this analysis is thus to understand if there are problems in product integration that are possible to avoid if each of the reference models are expanded with the practices that are not explicitly described today.

For each reference model, a table summarizes the described product integration practices and the adherence to the tasks as observed in the cases. The seven columns describing the results from the case studies include: an indication for each case if the practice has been observed as performed (+), observed as not performed (-), not investigated or not possible to determine (?), and if there are indications of problems connected to the practice (indicated with the problem label). Only the explicitly described practices

have been included in the tables as this leaves less interpretation to the user of the reference model.

Two issues limit the value of this analysis. The first is that explicit coverage of all the practices described in all reference models was not made in all cases. This means that practices may be performed even if no evidence can be found in the material from the interviews and document reviews. This has been indicated with a questions mark (?) in the tables. The second issue is that problems may exist in the organization without indications in the table, based on the fact that not all practices where explicitly covered in the case studies. However, all problems that have been captured have been possible to relate to practices that have been investigated.

ISO/IEC 12207:1995 compared to cases

The practices in ISO/IEC 12207:1995 cover 7 of the 17 unique problems found in the case studies. These are related to the practices used to ensure that the integrated software is ready for verification. This standard has no requirements on the handling of interfaces, which represents the cause of many of the problems found in the case studies

Table 10. ISO/IEC 12207:1995 compared to cases

Generic activity description	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
2. Develop an integration plan based on the strategy	+	-	+	-	-	-	-
6. Define interfaces	-	-	-	+	-	-	-
10. Develop and document a set of tests for each requirement of the assembled components	+	-	+	-	+	-	-
11. Verify completeness of components obtained for integration through checking criteria for delivery	- 1B	-	- 3A	+	-	- 6A	- 7A
12. Deliver/obtain components as agreed in the schedule	- 1A	-	+	-	- 5A	- 6B	-
13. Integrate/assemble components as planned	+	+	+	+	+	-	-
14. Evaluate/test the assembled components	+	+	+	+	+	-	-

EIA-632

The requirements in EIA-632 are concrete, but do not include requirements in all the areas where we have found problems in the case studies. 4 of the 17 unique problems are related to practices described.

Table 11. EIA-632 compared to cases

Generic activity description	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
2. Develop an integration plan based on the strategy	+	-	+	-	-	-	-
6. Define interfaces	-	-	-	+	-	-	-
10. Develop and document a set of tests for each requirement of the assembled components	+	-	+	-	+	-	-
11. Verify completeness of components obtained for integration through checking criteria for delivery	- 1B	-	- 3A	+	-	- 6A	- 7A
13. Integrate/assemble components as planned	+	+	+	+	+	-	-
14. Evaluate/test the assembled components	+	+	+	+	+	-	-

Capability Maturity Model Integration (CMMI), Version 1.1

13 of the 17 problems encountered in the case studies regarding product integration can be related to practices that are described in the CMMI. Of the four that are not covered are three related to late delivery of components to integration, and one is related to the strategy regarding error correction.

Table 12. CMMI compared to cases

Generic activity description	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
2. Develop an integration plan based on the strategy	+	-	+	-	-	-	-
3. Define and establish an environment for integration	+	+	+	+	- 5B	+ 6C	+ 7B
4. Define criteria for delivery of components	- 1B	- 2A	+	-	+	-	-
6. Define interfaces	-	-	-	+	-	-	-
7. Review interface descriptions for completeness	- 1C	- 2C	-	- 4A	-	+	+
8. Ensure coordination of interface changes	- 1C	- 2C	-	-	-	- 6D	- 7C
9. Review adherence to defined interfaces	-	-	+	+	-	-	-
11. Verify completeness of components obtained for integration through checking criteria for delivery	- 1B	-	- 3A	+	-	- 6A	- 7A
13. Integrate/assemble components as planned	+	+	+	+	+	-	-
14. Evaluate/test the assembled components	+	+	+	+	+	-	-

EAI-731.1

As with CMMI, many problems found in the case studies can be related to practices in EIA-731.1. Of the 17 unique errors found in the cases, 13 are covered by the practices. The remaining four errors are related to the build environment and the criteria for integration.

Table 13. EIA-731.1 compared to cases

Generic activity description	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
1. Define and document an integration strategy	?	+ 2B	+	+	+	?	?
2. Develop an integration plan based on the strategy	+	-	+	-	-	-	-
6. Define interfaces	-	-	-	+	-	-	-
7. Review interface descriptions for completeness	- 1C	- 2C	-	- 4A	-	+	+
8. Ensure coordination of interface changes	- 1C	- 2C	-	-	-	- 6D	- 7C
9. Review adherence to defined interfaces	-	-	+	+	-	-	-
11. Verify completeness of components obtained for integration through checking criteria for delivery	- 1B	-	- 3A	+	-	- 6A	- 7A
12. Deliver/obtain components as agreed in the schedule	- 1A	-	+	-	- 5A	- 6B	-
13. Integrate/assemble components as planned	+	+	+	+	+	-	-
14. Evaluate/test the assembled components	+	+	+	+	+	-	-

ISO/IEC 15288, Systems engineering – system life cycle processes

The standard covers 11 of the 17 problems found in the case studies, and most of the other errors are related to the interface handling which is not explicitly covered.

Table 14. ISO/IEC 15288:2002 compared to cases

Generic activity description	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
1. Define and document an integration strategy	?	+ 2B	+	+	+	?	?
2. Develop an integration plan based on the strategy	+	-	+	-	-	-	-
3. Define and establish an environment for integration	+	+	+	+	- 5B	+ 6C	+ 7B
5. Identify constraints from the integration strategy on design	?	?	?	?	?	?	?
9. Review adherence to defined interfaces	-	-	+	+	-	-	-
11. Verify completeness of components obtained for integration through checking criteria for delivery	- 1B	-	- 3A	+	-	- 6A	- 7A
12. Deliver/obtain components as agreed in the schedule	- 1A	-	+	-	- 5A	- 6B	-
13. Integrate/assemble components as planned	+	+	+	+	+	-	-
14. Evaluate/test the assembled components	+	+	+	+	+	-	-
15. Record the integration information in an appropriate repository	-	-	-	-	-	-	-

Table 15 summarizes the use of practices derived from the reference models, and the problems identified in the case studies. Through our case studies and the investigation of different reference models, we have found the following:

- Five of the practices (PI Practice 4, 7, 8, 11, and 12) indicate that problems may appear when the practice is not followed.
- In three instances, identified problem areas can be related to practices that are performed in the organizations. Two of these are related to the strategy definition (PI Practice 1), and are in fact referring to the lack of rules for corrections of errors. Hence, it may be that the integration strategy is available, but does not cover the rules for error corrections. The final problem is related to the build environment definition (PI practice 3) and is also an indication that the descriptions of practices do not cover the quality of the implementation.
- Table 16 illustrates also that the problems related to product integration could not all be related to a practice in the reference model for any model, e.g. for ISO/IEC 15288 we could associate 11 of the 17 problems found in the case studies to the product integration practices in that standard. The results confirm the need of a broader approach than is available in any of the examined reference models

Table 15. Performed activities and problems identified in the case studies

Generic activity description	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
1. Define and document an integration strategy	?	+ 2B	+	+	+	?	?
2. Develop an integration plan based on the strategy	+	-	+	-	-	-	-
3. Define and establish an environment for integration	+	+	+	+	- 5B	+ 6C	+ 7B

Generic activity description	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
4. Define criteria for delivery of components	- 1B	- 2A	+	-	+	-	-
5. Identify constraints from the integration strategy on design	?	?	?	?	?	?	?
6. Define interfaces	-	-	-	+	-	-	-
7. Review interface descriptions for completeness	- 1C	- 2C	-	- 4A	-	+	+
8. Ensure coordination of interface changes	- 1C	- 2C	-	-	-	- 6D	- 7C
9. Review adherence to defined interfaces	-	-	+	+	-	-	-
10. Develop and document a set of tests for each requirement of the assembled components	+	-	+	-	+	-	-
11. Verify completeness of components obtained for integration through checking criteria for delivery	- 1B	-	- 3A	+	-	- 6A	- 7A
12. Deliver/obtain components as agreed in the schedule	- 1A	-	+	-	- 5A	- 6B	-
13. Integrate/assemble components as planned	+	+	+	+	+	-	-
14. Evaluate/test the assembled components	+	+	+	+	+	-	-
15. Record the integration information in an appropriate repository	-	-	-	-	-	-	-

Our finding is that there is a small set of practices that need to be implemented to have working product integration. However, they are not sufficient, which is indicated by the larger set of practices described in different reference models.

Table 16. Number of errors for each case related to practices in the reference models

	ISO/IEC 12207	EIA-632	CMMI	EIA-731.1	ISO/IEC 15288
Case 1 3 problems	2	1	2	3	2
Case 2 3 problems	0	0	2	2	1
Case 3 1 problem	1	1	1	1	1
Case 4 1 problem	0	0	1	1	0
Case 5 2 problems	1	0	1	1	2
Case 6 4 problems	2	1	3	3	3
Case 7 3 problems	1	1	3	2	2
Total 17 problems	7	4	13	13	11

6. Discussion

The difficulties for product development organization found during integration are disrupting the progress of development projects, and increases time-to-market. Problems origin for example in the lack of integration planning, insufficient management of interfaces, and inadequate preparation of components delivered for integration.

Our intent with this research has been to examine to which extent the practices described in reference models are useful as a support for development units. Five reference models have been analyzed and practices as well as problems from seven development projects have been captured. We have based the investigation on the following questions and summarize here the results:

- How are the practices described in reference models useful for product development units for improving product integration?

The reference models can be used as a tool for identifying weak areas in the product integration processes. Care must however be taken when selecting the reference model so that sufficient coverage is obtained.

- What is the core set of practices that can be identified to reduce problems in product integration?

Through the case studies, five Product Integration practices have been identified to be necessary to perform to have successful product integration. These are PI 4 Define criteria for delivery of components, PI 7 Review interface descriptions for completeness, PI 8 Ensure coordination of interfaces changes, PI 11 Verify completeness pf components obtained for integration through checking criteria for delivery, and PI 12 Deliver/obtain components as agreed in the schedule. For the interface handling, also PI 6 Define interfaces is important as PI 7 relies on that practice. The same reasoning can be applied on PI 2 Develop an integration plan based on the strategy which is a prerequisite for PI 12.

- Is it appropriate to combine reference models to provide better support to product development units, and how can this be done?

The analysis of existing reference models show that none of the investigated models cover the problem situations for the investigated product development organizations regarding product integration. This leads to our conclusion that a combination of the content in the reference models can be

helpful for development organizations when designing and improving the product integration process.

Our suggestion to companies that would like to improve the product integration processes is to use the set of 15 practices described in section 4.6 and perform an assessment on the current practices. In addition to this, the problem areas should be captured, and together with the assessment result be the basis for any improvement effort.

One additional conclusion is that a continued development towards an agreed body-of-knowledge for the product integration area is needed. This can be achieved through consolidation and further validation of existing reference models. Finally, as a result of our studies, we see the need to perform additional investigations to understand the reasons for the lack of use of proven good practices, and to understand why the implementation of product integration practices sometimes fails.

Several different additional directions for future research have been identified. Additional organizations using different technologies should be investigated and compared to clarify if there are dependencies between the type of application and the needed practices. A related direction is to look at the influence architectural decisions have on product integration. Also, methods for how to determine the best improvement proposals for product integration for different types of organizations should be investigated, enhanced, and possibly developed. This probably requires an agreed body-of-knowledge for product integration that supports different types of organizations, and the use of different development models. The reference models investigated in this article do not prescribe specific development models, but the selection is likely to influence the ability to follow the practices and to be successful in the product integration.

References

[1] ANSI, American National Standards Institute, http://www.ansi.org/, 2007.

[2] ANSI/EIA-632-1999, Processes for Engineering a System, Electronic Industries Alliance, Government Electronic and Information Technology Association, 1999.

[3] J. Campanella, Principles of Quality Costs: Principles, implementation and Use, ASQ Press, Milwaukee, WN, USA,, 1999.

[4] C.G. Chittister, and Y.Y. Haimes, Systems integration via software risk management. Systems, Man and Cybernetics, Part A, IEEE Transactions on 26 (1996) 521-532.

[5] M. de Jonge, Package-based software development. Euromicro Conference, 2003. Proceedings. 29th (2003) 76-85.

[6] A.H. Dogru, and M.M. Tanik, A process model for component-oriented software engineering. IEEE Software 20 (2003) 34-41.

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

[8] F. Ekdahl, and S. Larsson, Experience Report: Using Internal CMMI Appraisals to Institutionalize Software Development Performance Improvement. 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06) (2006) 216-223.

[9] M. Fowler, Continuous Integration, http://www.martinfowler.com/articles/continuousIntegration.html, 2006.

[10] IEEE1220-2005, IEEE Standard for Application and Management of the Systems Engineering Process, Institute of Electrical and Electronics Engineers, 2005.

[11] IEEE, The Institute of Electrical and Electronics Engineers, http://www.ieee.org/, 2007.

[12] INCOSE, International Counsil on Systems Engineering, http://www.incose.org/, 2007.

[13] ISO9001:2000, Quality management systems -- Requirements, ISO, 2000.

[14] ISO, International Standardization Organization, http://www.iso.org, 2007.

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

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

[17] D.R. Kuhn, On the effective use of software standards in systems integration. Systems Integration, 1990. Systems Integration '90., Proceedings of the First International Conference on (1990) 455-461.

[18] R. Land, and i. Crnkovic, Existing Approaches to Software Integration - and a Challenge for the Future, Fourth Conference on Software Engineering Research and Practice in Sweden, Linköping, Sweden, 2004.

[19] S. Larsson, Improving software product integration, Dept. of Computer Science and Electronics Mälardalen University, Västerås, 2005, pp. xi, 108.

[20] S. Larsson, and I. Crnkovic, Case Study: Software Product Integration Practices, 6th international conference Profes, June, 2005, Oulu Finland, 2005, pp. 272-285.

[21] S. Larsson, I. Crnkovic, and F. Ekdahl, On the expected synergies between component-based software engineering and best practices in product integration, Proceedings - 30th EUROMICRO Conference, Aug 31-Sep 3 2004, IEEE Computer Society, Los Alamitos;Massey University, Palmerston, CA 90720-1314, United States;New Zealand, Rennes, France, 2004, pp. 430-436.

[22] S. Larsson, P. Myllyperkiö, and F. Ekdahl, Product Integration Improvement Based on Analysis of Build Statistics, To be presented at ESEC/FSE, Dubrovnik, Croatia, 2007.

[23] E.G. Nilsson, E.K. Nordhagen, and G. Oftedal, Aspects of systems integration. Systems Integration, 1990. Systems Integration '90., Proceedings of the First International Conference on (1990) 434-443.

[24] RTI, The Economic Impacts of Inadequate Infrastructure for Software Testing, National Institute of Standards and Technology, Gaithersburg, MD, USA,, 2002.

[25] A.P. Sage, Charles L. Lynch, Systems integration and architecting: An overview of principles, practices, and perspectives. Systems Engineering 1 (1998) 176-227.

[26] M. Schulte, Model-based integration of reusable component-based avionics systems - a case study. (2005) 62-71.

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

[28] SEI, Software Engineering Institute, http://www.sei.cmu.edu/, 2007.

[29] V. Stavridou, Integration standards for critical software intensive systems. Software Engineering Standards Symposium and Forum, 1997. 'Emerging International Standards'. ISESS 97., Third IEEE International (1997) 99-109.

[30] V. Stavridou, Integration in software intensive systems. Journal of Systems and Software 48 (1999) 91-104.

Paper E

ASSESSING THE INFLUENCE ON PROCESSES WHEN EVOLVING THE SOFTWARE ARCHITECTURE

Stig Larsson, Anders Wall, Peter Wallin
Presented at IWPSE,
Cavtat, Croatia, August 2007

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.

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.

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).

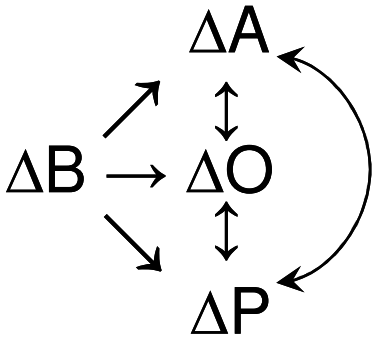

Figure 1. Relationship between different types of changes

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.

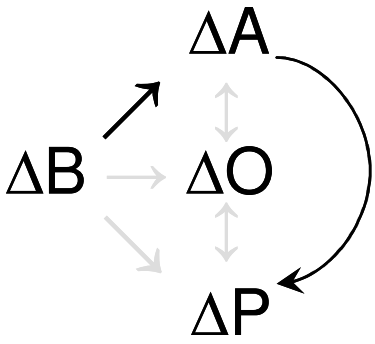

Figure 2. Architectural changes affecting processes

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.

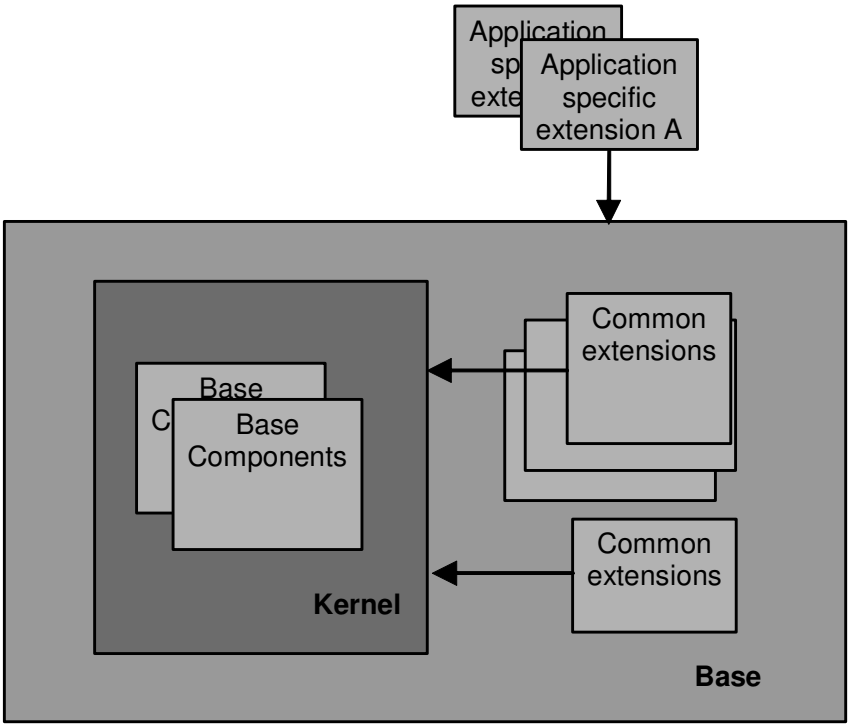

Figure 3. Block diagram of the refactored software

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.

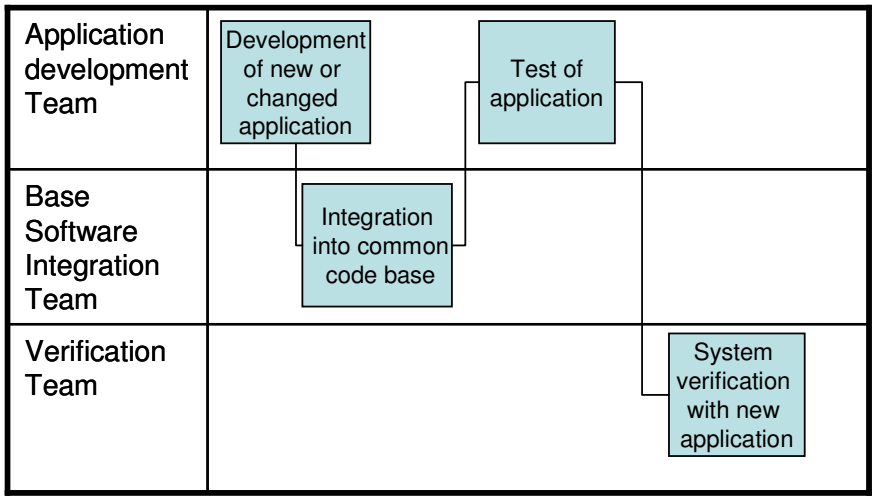

Figure 4. Example scenario. (Boxes denotes activities, and lines are showing flow of information and data)

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

Process area or activity	Strategy 1	Strategy 2	Strategy 3	Strategy 4
	Centralized distribution		Distribution by application	
	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(n2) 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. Conclusion 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. IDEALSM: 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 (SCAMPISM) 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).