

# Component-based Development Process and Component Lifecycle

Ivica Crnkovic<sup>1</sup>, Stig Larsson<sup>2</sup>, Michel Chaudron<sup>3</sup>

<sup>1</sup>Mälardalen University, Västerås, Sweden, [ivica.crnkovic@mdh.se](mailto:ivica.crnkovic@mdh.se)

<sup>2</sup>ABB Corporate Research, Västerås, Sweden, [stig.bm.larsson@se.abb.com](mailto:stig.bm.larsson@se.abb.com)

<sup>3</sup>Technical University Eindhoven, Eindhoven, The Netherlands

**Abstract.** *In recent years component-based development has in recent years become an established approach. Component-based Software Engineering (CBSE) that deals with the entire lifecycle of component-based products has been focused on technologies related to design and implementation of software components and systems built from software components. The experience has shown that pure technologies alone are not enough. A CBSE approach requires certain changes in development and life cycle processes. However very few CBSE works, either research or practical, have addressed these topics. This paper describes principle differences of component-based and non-component based processes. Also we an overview of a case study from a company that applies component-based approach.*

**Keywords.** Component-based software Engineering, Life cycle processes

## 1. Introduction

Component-based approach has in last years shown considerable successes in many application domains. Distributed and web-based systems, desktop and graphical applications are typical examples of domains in which component-based approach has been very successful. In these domains the general-purpose component technologies, such as COM, .NET, EJB, J2EE are used.

There is however very little knowledge about development processes that is specific for the component-based development.

This paper describes the characteristics of component-based processes, the reasons for this, and the differences from a non-component-based development process. From a case study it shows that component-based approach specific solutions in organization of a company.

The rest of the paper is as follows. Section 2 gives an overview of development processes. Section 3 discusses some basic characteristics of component-based approach and illustrates component-based activities in the “V”

development process model. We illustrate a component-based development approach in an industrial case study in section 4. Finally, section 5 concludes the paper.

## 2. Basic characteristic of lifecycle process models

Lifecycle processes include all activities of a product or a system during its entire life, from the business idea for its development, through its usage and its completion of use. Different models have been proposed and exploit in software engineering, and different models have exhibit their (in)abilities to efficiently govern all activities required for a successful development and use of products. We can distinguish two main groups of models: Sequential and evolutionary. The sequential models define a sequence of activities in which one activity follow after a completion of the previous one. Evolutionary models allow performance of several activities in parallel without requirements on a stringent completion of one activity to be able to start with another one. Well known example of sequential models are waterfall model, or V model, and of evolutionary models, iterative and incremental development, or spiral model.

Independently of the type of the model we can identify the basic activities present in any lifecycle process model. These activities are the following:

**Requirements analysis and specification.** The system’s services, constraints and goals are established (i.e. a specification what the system is supposed to do).

**System and software design.** An overall system and software architecture is established. A detailed design follows the overall design. Software design includes identifying and describing the fundamental software systems abstractions and their relationships.

**Implementation and unit testing.** The formalization of the design in an executable way, which can be composed of smaller units. Testing of the units follows their implementation.

**System Integration.** The system units are integrated.

**System verification and validation.** The correctness of the complete system is verified, and the system is validated in respect to the requirements.

**Operation support and maintenance.** A set of activities that are required for the expected performance of the system.

**Disposal.** A disposal activity, often forgotten in many lifecycle models, includes the phasing-out of the system, i.e. a possible replacement by another system or a complete termination.

Not all models are suitable for all types of system lifecycles. Usually large systems which include many stakeholders and which development last a long period prefer using sequential models. The systems which use new technologies, are smaller, and to which the time-to market is important, usually explore evolutionary models which are more flexible and which can show some results much earlier than sequential models. These models can be applied in a component-based development, but require adoption to the principles of component-based approach.

### 3. Component-based lifecycle process models

CBSE addresses challenges similar to those encountered elsewhere in software engineering. Many of the methods, tools and principles of software engineering used in other types of system will be used in the same or a similar way in CBSE. There is however one difference; CBSE specifically focuses on questions related to components and in that sense it distinguishes the process of “component development” from that of “system development with components”.

#### 3.1 Building systems from components

The main idea of the component-based approach is building systems from pre-existing components. This assumption has several consequences for the system lifecycle. First, the development processes of component-based systems are separated from development processes of the components; the components should already been developed and possibly used in other products when the system development process starts. Second, a new separate process will appear: Finding and evaluating the

components. Third, the activities in the processes will be different from the activities in non-component-based approach; for the system development the emphasis will be on finding the proper components and verifying them, and for the component development, design for reuse will be the main concern.

There is a difference in requirements and business ideas in these two cases and different approaches are necessary. Components are built to be used and reused in many applications, some possibly not yet existing, in some possibly unforeseen way

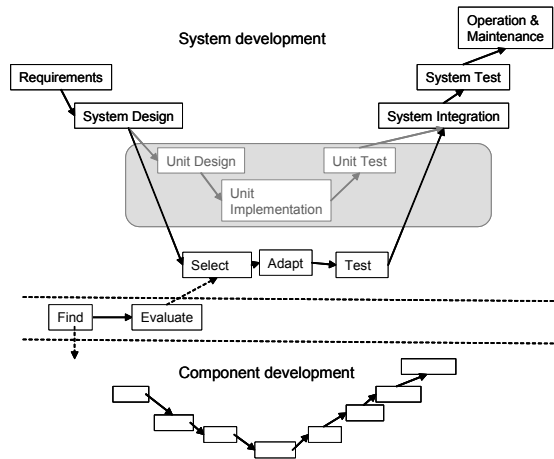
System development with components is focused on the identification of reusable entities and relations between them, beginning from the system requirements and from the availability of components already existing [1][2]. Much implementation effort in system development will no longer be necessary but the effort required in dealing with components; locating them, selecting those most appropriate, testing them, etc. will increase [3].

We do not only recognize different activities in the two processes, but also find that many of these activities can be performed independently of each other. In reality the processes are already separate as many components are developed by third parties, independently of system development. Even components being developed internally in an organization which uses these very same components, are often treated as separate entities developed separately.

Let us discuss these differences in more detail. Figure 1 shows a V development model adopted to component-based approach.

We use V model as this model is widely used in many organizations – typically large organization building complex long-life products, such as cars or robots. In this model the process starts in a usual way by requirements engineering and requirements specification, followed by system specification. In a non-component-based approach the process would continue with the unit design, implementation and test. Instead of performing these activities that often are time and efforts consuming, we simply select appropriate components and integrate them in the system. However, two problems appear here which break this simplicity: (i) It is not obvious that there is any component to select, and (ii) the selected component only partially fits to our overall design. The first fact shows that we must have a process for finding components. This process includes activities for finding the

components, and then the component evaluation. The second fact indicates for a need of component adoption and testing before it can be integrated into the system. And of course there must be a process of component development, this being independent of the system development process.



**Figure 1. V development process for CBD**

Figure 1 still shows a simplified and an idealized process. Its assumption is that the components selected and used are sufficiently close to the units identified in the design process, so that the adaptation process requires (significantly) less efforts than the units' implementation. Further it does not consider what happens in the maintenance process; what happens if a system malfunctions due to a problem occurred in a component, or due to incompatibilities of the components. This indicates that the component-based approach is not only limited to the development process, or part of the development process, but to the entire life cycle. Already in the very early phase, in the Requirements and Design phases the system requirements engineers and system architects must be aware about availability of the existing components.

A more realistic process is shown on Figure 2. Let us look at the activities at different phases of the development process in more detail.

#### **Requirements analysis and specification.**

In this phase one important activity is to analyze the possibility of realizing the solutions that will meet these requirements. In a component-based approach this implies that it is necessary to analyze whether these requirements can be fulfilled by available components. This means that the requirements engineers must be aware of

components that can possibly be used. Since it is not likely that appropriate components can always be found, there is a risk that the new components have to be implemented. To keep with component-based approach (and utilize its advantages) one possibility is to negotiate the requirements and modify them to be able to use the existing components.

**System and software design.** Similar to the requirements specification phase the system specification and design is strongly related to the availability of the components. The potential components are complying with a particular component model. One could assume that it would be possible to use components implemented in different component technologies, but in practice it is very difficult to achieve interoperability between different component models. Particular component model requires a particular architectural framework, and the application is supposed to use this framework. This directly has impact on architectural decisions. For example if the component model requires a client-server architecture style, it is obvious that the application will use that style and not another (for example pipe-filter). This will put limitations on the system design. Also, other properties of components can have a direct influence on the design decisions. For this reason the design process is tightly connected to the availability of the components.

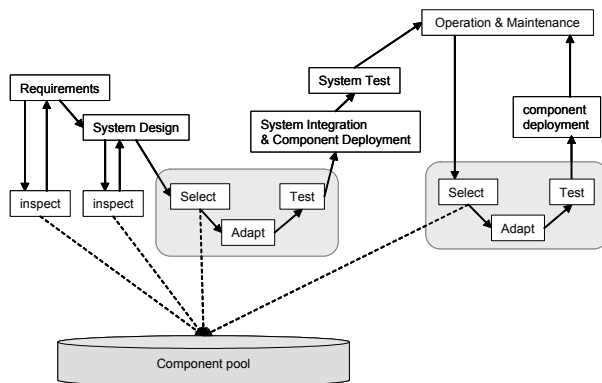
**Implementation and unit testing.** When building component-based system, an ideal case is to build an application by direct integration of components, i.e. directly connecting components. The "glue code" is a code that specifies this connection. In practice the role of the glue code will also include adaptation of the components, and even implementation of new functions. In an ideal case the components themselves are already built and tested. However the component tests in isolation are not sufficient. Often design units will be implemented as assemblies of several components and possibly a glue code. These assemblies must be tested separately, since an assembly of correct components may be incorrect although the components themselves are correct [5].

**System Integration.** The integration process includes integration of standard infrastructure components that build a component framework and the application components. The integration of a particular component into a system is called a component deployment. In difference to the

entire system integration a component deployment is a mechanism for integration of particular components – it includes download and registering of the component.

**System verification and validation.** The standard test and verification techniques are used here. The specific problem for component-based approach is location of error, especially when components are of “black box” type and delivered from different vendors. Typically a component can exhibit an error, but the cause of the malfunction lies in another component. Contractual interfaces play an important role in checking the proper input and output from components. These interfaces enable a specification of input and output and checking the correctness of data.

**Operation support and maintenance.** The maintenance process includes some steps that are similar to the integration process: A new or modified component is deployed into the system. Also it may be necessary to change the glue code. In most of the cases an existing component will be modified or a new version of the same component will be integrated into the system. Once again new problems caused by incompatibility between components, or by broken dependencies may occur. This means, one again that the system must be verified (either formally, or by simulation, or by testing).



**Figure 2. A Detailed V development process for CBD**

In comparison with a non-component-based approach, in a component-based development process there are significantly less efforts in programming, but the verification and testing require considerably more efforts. The verification activity repeats in several phases, with slightly different goals:

- Verifying component in an isolation;
- Verifying components in an assembly;

- Verifying the systems when a component has been deployed into the system.

### 3.2 Building reusable components

The process of building components can follow an arbitrary development process model. However any model will require certain modification to achieve the goals; in addition to the demands on the component functionality, a component is built to be reused. Reusability implies generality and flexibility, and these requirements may significantly change the component characteristics. For example there might be a requirement for portability, and this requirement could imply a specific implementation solution (like choice of programming language, implementation of an intermediate level of services, programming style, etc.). The generality requirements imply often more functionality and require more design and development efforts and more qualified developers. The component development will require more efforts in testing and specification of the components. The components should be tested in isolation, but also in different configurations. Finally the documentation and the delivery will require more efforts since the extended documentation is very important for increasing understanding of the component. An example of extended component specification can be found in ROBOCOP component model [6]; a component is specified by a row of modules: executable model, functional model, simulation model, resource model, etc. Each model includes a corresponding documentation.

### 4. Industrial Case of component-based process model

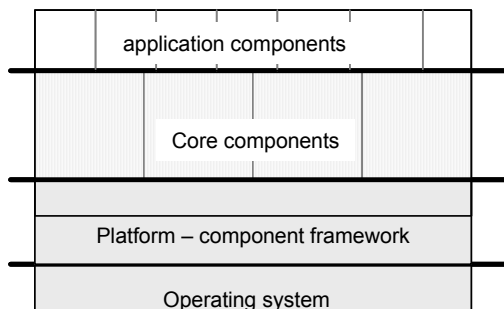
We give here a short overview of a case study: a process model used in a large international company in consumer electronics. The case study was performed by four researchers in intensive interviews with different stakeholders of the development projects: System architects, component architects, developers, project leaders, the management, the quality assurance and test people, and principal specialist.

The development divisions of the company are placed in four different countries and they produce numerous products with different variants and models. The company has adopted component-based development using product-

line architecture. The component model is internally developed and most of the tools are internally developed. The reason for that are the specific requirements of the domain: low resource usage, high availability, and soft real-time requirements.

The component model follows the basic principles of CBSE: The components are specified by interfaces which distinguish “require” from “provide” interfaces. In addition to functional specification, the interface includes additional information; the interaction protocols, the timeliness properties, and the memory usage. The component model enables a smooth evolution of the components as it allows existence of multiple interfaces. The model has a specific characteristic; it allows a hierarchical compositions: a composite component is treated as a standard component and in can further be integrated in another component. The components are also developed internally, but their development is separated from the development of the products.

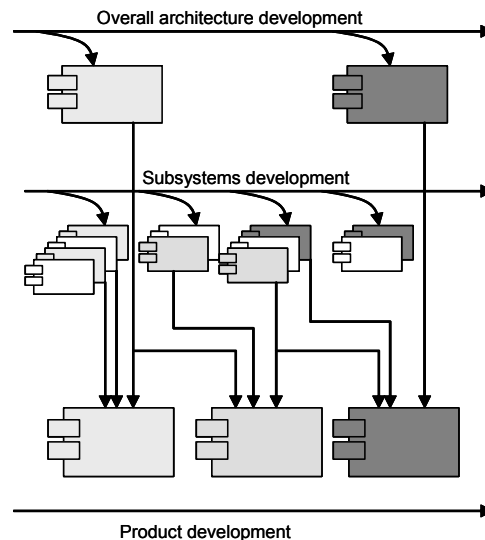
The product-line architecture identifies the basic architectural framework. The product architecture is shown on Figure 3.



**Figure 3. Product Software Architecture**

The product architecture is a layered architecture which includes (i) operating system, (ii) the component framework which is an intermediate level between domain-specific services and operating, (iii) core components which are included in all product variants, and (iv) application components that usually are different for different product variants.

Complementary to this horizontal layering there is a vertical structuring in form of subsystems. Subsystems are also related to the organizational structures; they are responsible for development and maintenance of particular components. The overall process is designed as follows as shown on Figure 4.



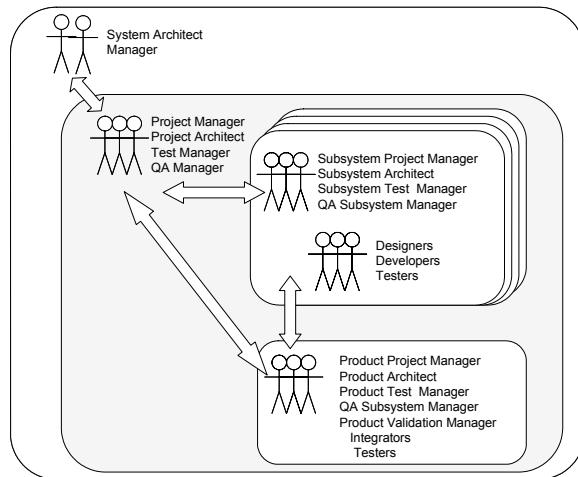
**Figure 4. Overall development process**

In the overall process there are three sets of the independent parallel processes: (i) An overall architecture and platform development process responsible for delivering new platforms and basic components, (ii) Subsystem development processes which deliver a set of components that provide different services, and (iii) the product development process which is basically an integration process. This process arrangement makes it possible to deliver new products every six months, while the development of subsystem components takes typically between 12 and 18 months. The specifics of these projects are that all deliverables have the same form. A deliverable is a software package defined as a component. The two main documents belong to every deliverable: Component interface specification and Component sheet; the first document describing the interconnection, the second describing the component internals.

Although the overall development and production is successful, the process suffers from several problems. The most serious is late discovery of errors, due to interface or architectural mismatches, insufficient specifications of semantics of the components, or due to inappropriate interfaces. Also the problems related to encapsulation of a service in components often occur; due to functional overlaps, or some requirements that affect the architecture, it is difficult to decide in which components a particular function will be implemented. All these problems point out that it is difficult to perform the processes independently; negotiation between different subsystems and agreements in many technical details between different teams are necessary.

For this reason the processes are not completely separated. The processes are distributed among several projects and there is an overall project that coordinates these projects.

The processes have a strong support in the project and organization structure (see Figure 5).



**Figure 5. Project organization structure**

The system architect and management have overall responsibilities for requirements, policies, product line architecture, products visions, and long term goals. The project architect has a responsibility for the overall project which results in a line of products. He/she coordinates the architectural design of the product family and subsystems. The test and quality-assurance (QA) managers have similar role in their domains: to ensure coordination and compatibility of tests and quality processes. The subsystem architects provide with the designs of their subsystems and coordinate the design decisions with other subsystems. Each subsystem has a test team and a QA manager which responsibility is the quality of the delivered subsystem components. The integration team which work in the delivery projects is represented by a product architect, QA and test managers who coordinate the activities with other projects. We can observe that the project teams have many “non-productive” stakeholders. This is in line of the component-based approach – more efforts must be put on overall architecture and test, and less on the implementation itself. The development processes in our case is manly of an evolutionary model. The platform, the subsystems and the products are developed in several iterations until the desired functionality and quality is achieved. This requires synchronizations of iterations.

## 5. Conclusion

A component-based approach cannot be fully utilized if the development processes and even the development organizations are not adopted according to basic principles of CBSE. Since this approach aims for increased reusability of existing components, the efforts for the implementations decrease, and the efforts for system verification increase. This requires adjustments of the development processes.

By an industrial case study we have pointed out the difficulties to achieve a complete separation of the development processes of systems from the components, as well as the need for a project organization which puts a more important role on the architectural issues, and system and components verification.

## 6. Acknowledgements

The authors would like to thank to Chritiene Aarts for his enormous help in organizing the interviews and to all interviewees which took their valuable time for the interviews.

## 7. References

- [1] Bass L., Clements P., and Kazman R., *Software Architecture in Practice*, Addison-Wesley, 1998.
- [2] Garlan D., Allen R., and Ockerbloom J., *Architectural Mismatch: Why Reuse is so hard*, *IEEE Software*, Vo.12, issue 6, 1995.
- [3] Morisio M., Seaman C. B., Parra A. T., Basil V. R., Kraft S. E., and Condon S. E., "Investigating and Improving a COTS-Based Software Development Process", In *Proceedings , 22nd ICSE*, ACM Press, 2000.
- [4] Borghoff V, Paresi R, editors. *Information Technology for Knowledge Management*. New York: Springer Verlag; 1998.
- [5] Ivica Crnkovic and Magnus Larsson (editors), *Building Reliable Component-Based Software Systems*, Artech House Publishers, ISBN 1-58053-327-2, 2003
- [6] ITEA project, *ROBOCOP- Robust Open Component Based Software Architecture for Configurable Devices Project* <http://www.hitech-projects.com/euprojects/robocop>