

# Component-based Software Engineering – New Challenges in Software Development

Ivica Crnkovic

Malardalen University, Department of Computer Engineering, Västerås, Sweden

ivica.crnkovic@mdh.se, <http://www.idt.mdh.se/personal/icc>

tel: +56 70 533 75 57 fax: +46 21 10 41 60

## Summary

We are witnesses of enormous expansion of software in business, industry, research, everyday life. Software is becoming the key factor for success even in traditionally non-software areas. Most of the software users are non-experts. These trends raise new demands on software. Usability, robustness, simple installations and integrations become the most important features of software. As software supports more and more different areas, the requirements for integration of different areas become stronger. The consequence of this is that software is increasing in size and complexity. The main challenge of software today is to manage the complexity and adaptability to the changes. A new approach, focusing on reuse of existing pieces of software and developing reusable entities and based on new component technologies, such as COM/DCM such as Javabeans, is becoming dominant: Component-based development. Although very promising, component-based development shows weakness in many aspects, which is the consequence of lack of methodology, process and tools support. A new discipline of software engineering Component-based Software Engineering is in focus of researchers and industry. The primary role of Component-based Software Engineering is to deal with developing systems from parts (components), developing parts as reusable entities, and maintaining and improve systems by customising and replacing those parts. This requires established methodologies and tool support covering the entire component and product lifecycle, the organisational, marketing, legal, and

other aspects. In addition to new technologies, also other disciplines from software engineering need new methodologies. For example, the development lifecycle model is different for building components from building systems from components. Some of the disciplines become more important. One example is Software Architecture, which primary goal is to define the system structure consisting of components and relation between them. In component-based approach the components exist not only as logical, but also as physical entities, recognizable and deployable at run-time. For this reason the architectural questions are as significant for the run-time phase as for the design phase. There are many other disciplines that require new approach. The success of the component-based development depends directly of successful research and implementation of component-based software engineering.

**Keywords:** Software Components, Software Engineering, Middleware, Software Architecture, COTS, Component-based development, Component-based Software Engineering

## Software Development Challenges

In recent years we are witnesses of dominance of software role in industry, business, administration and research. Importance of software in technical systems is changing from a marginal role to a core part of business. System features based on software functionality, rather than other characteristics, are becoming dominant for competition on the market. Software parts of the systems are the most suitable to introduce new product variants with new features. Software is becoming dominant in business. The Internet is changing the way of communication between customers and suppliers, the way of placing the products on the market, and the way of providing support to the customers. The new Internet-related technologies give enormous opportunities to expand business by entering new business areas and by increase the product quality in terms of better support. For example, it is obvious today, that every product placed on the market, regardless if it is about a student-series book, a latest variant of a car, or a new movie, has a joined extensive Web page which provides any possible information or other services related to it. In service-oriented business, the web-based services become the first and foremost way of communication between customers and suppliers. Finally, software is giving opportunities to significantly improve the development and other internal processes in an organisation. At the same time the computer power, in terms of memory, CPU and communication possibilities, is increasing continuously. Computer technology is today an integral part of organisation independently of if their core business is related to computer development and services, or not.

The consequence of these trends is increasing demands on software products. Software meets new areas, not necessary related traditional software areas. New areas do not require only the insight in their processes, know-how and technologies, but also new technologies and process in

software development. The profiles of software users have changed from a relative small number of and highly skilled experts to broad groups of people not necessarily trained or having interests in computers themselves. A number of software users has increased dramatically, and different users exploit software in many different, often unpredictable, ways. Their expectations of software are also different. This requires another approach to software. Usability, robustness, simplicity become the most important features of software. As a consequence of enlarging the area of software utilization, the requirements for integration of different areas become stronger. We identify *vertical integration*, where data and processes on different levels are integrated, and *horizontal integration* where similar type of data and processes from different domains are integrated. A typical example of vertical and horizontal integration we can find in industrial process automation, Where we distinguish levels of management: on the lowest level (*Field Management*), we have data collected and controlled directly from the process, integrating on the plant level (*Process Management*), further processed data to analysis and combine them with data provided from the market and finally publishing selected data on the Web (*Business Management*).

These trends put a lot of pressure on software development. Software continuously increases in size and complexity. At the same time new software technologies appear faster and faster on the market. Traditionally, software development addressed challenges of increasing complexity and dependence on external software by focussing on one system at a time, and on delivery deadlines and budgets, while ignoring the evolutionary needs of the system. This has lead to a number of problems: the majority of projects fail to meet their deadline, budget, and quality requirements; and costs associated with software maintenance continue to rise.

To meet these challenges software development must be able to manage complexity and fast adaptation to changes. The key point to achieve this is reusability. As far as software products are

each time built from scratch, these goals cannot be achieved. From this perspective *Component-based Development* (CBD) comes as a proper approach: Software systems are built by composing components which are already completed and prepared for the integration. There are many advantages of component-based development [1]: Better management of complexity, reduced time to market, increased productivity, better quality, improved consistency, improved usability.

However, there are several disadvantages and risks that can jeopardize success of CBD:

- **Development time and efforts required for development of components.** There are several factors that can discourage development of reusable. First, building a reusable unit requires three to five times (B. Spencer, Microsoft, Presentation at 22<sup>nd</sup> ICSE, 1999) more efforts to develop than produce a unit for one specific purpose.
- **Unclear and ambiguous requirements.** In general, requirements management is an important part of development, where the main goal is to define consistent and complete requirements. As intention of building components is reusability, the goal is to reuse them in different applications, many of them still unknown, and thus the application requirements are not yet defined. The component developers must be able to predict the demands on the component that can be stated from the applications. This is valid for both functional and non-functional requirements.
- **Divergence of usability and reusability.** To achieve better reusability, a component must be general enough, scalable and adaptable, which makes a component more complex (and thus more complicated to use), and more demanding for computing resources (and thus more difficult to use).
- **Component maintenance costs.** While the application maintenance costs can decrease, the component maintenance costs can be very high since the component must respond to

different requirements of different applications which run in different environments, have different reliability requirements and may require different level of maintenance support.

- Reliability and sensitiveness on changes. As components and applications have separated lifecycles and different kind of requirements, there is a higher risk that a component does not completely fulfil the applications requirements and that it includes hidden characteristics not known for application developers. When introducing changes on the application level (changes such as update of operating system, update of other components, changes in the application, etc.), there is a risk that the change introduced will break the component's performance.

To utilize the advantages and overcome the problems and risks, we need a systematic approach to component-based development on the process and technology levels.

## **Component-Based Software Engineering**

The idea about building software from components is not new. A “classical” design of complex software systems always starts with identification of system parts, called subsystems or blocks, and on lower level modules, classes, procedures and so on. Reuse approach in software development has existed in many years. However, in recent years, the emergence of new technologies has significantly increased possibility to facilitate the concept of building systems and applications from reusable components. The expectations are huge from both customer and management side. Often these expectations are not met. The experience has shown that component-based development requires a mature and a systematic focus on component aspects in software development [10]. Traditional software engineering disciplines must be improved and adjusted to the new approach, and new disciplines must be developed. *Component-based*

*Software Engineering* (CBSE) has been recognized as a new subdiscipline of Software Engineering.

The primary goal of CBSE is to deal with developing systems from parts (components), developing parts as reusable entities, and maintaining and improve systems by customising and replacing those parts [2]. Building systems from components and building components for different systems requires established methodologies and processes not only related to the development/maintenance aspects, but also covering the entire component and product lifecycle, covering the organisational, marketing, legal, and other aspects. In addition to specific CBSE disciplines, such as component specifications or compositions, and technologies, such as different component models, there are many software engineering disciplines that require specific methodologies for component-based development. Most of these methodologies are not yet established in practice or even not yet developed. A lack of established methodologies leads in many cases to unsuccessful results and disbelief in component-based approach in general. The success of software development will in the nearest future depend very much of the success of establishing of CBSE disciplines, which is recognized both in industry and academia. All major software engineering conferences have sessions related to CBSE or CBD and there are many CBSE or CBD workshops [3][4][5][6][7]. According to the Gartner Group [8] “By 2002, 70 percent of all new applications will be deployed using component-based application building blocks.”

Further on we shall give overviews of certain CBSE disciplines and some of the trends and challenges in the nearest future.

## Component Specification

To get a common agreement about component-base development, the start point is to understand what a component is and what a component is not. As a generic term the definition is pretty clear – a component is a part of something. This definition is however too weak to be useful. There have been many discussions about the component specification [11][12]. We shall take Szyperski's definition [13] that is mostly used today and that is most appropriate to implementations in different component models:

*A software component is a unit of composition with contractually specified interface and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parts.*

The most important feature of component is a separation of interface from the implementation. This separation is different from those we can find in many programming languages (such as ADA or Modula-2), where we separate declaration from implementation, or those in object-oriented programming languages where we separate class definitions from class implementations. We require that the component integration into an application happens independently of the component development lifecycle. We do not need to recompile or re-link the application to update the application with a new component version. Another important factor of the separation is that the component implementation is unknown for the application, and this even can be the case for developers if a component is delivered by third party. An implication of that is the requirement for a complete specification of a component including its functional interface, non-functional characteristics (performance, resources required, etc.), use cases, tests, etc. While the present component-based technologies successfully manage functional interface, other parts of specifications are far beyond the requirements.

The component definition specified above is focused on use of components. It does not say too much about how to design, implement and specify a component. For this reason there are many other definitions which point on different aspects of component-based development. For example there is a strong relation between object-oriented programming (OOP) and components. Component models (also called component standards) COM/DCOM [14][15], .NET[16], Enterprise Java Beans (EJB)[17][18], and CORBA Component Model (CCM) [19] relate Component Interface to Class Interface or to an interface of set of classes. Components adopt object principles: Unification of functions and data, encapsulation. According to some opinions [20] there exists several forms which component can take: Component Specification (component characteristics, what component does), Component Interface (a part of specification, a definition of component's behaviour), Component Implementation (A realisation of Component Specification), Installed Component (deployed instance of a Component Implementation) and Component Object (an instance of Installed Object). Not all researchers agree that components are extension of OOP. On the opposite the difference between components and objects are in fact that an object has state and is a unit of instantiation, while a component is stateless and is a unit of deployment.

There is also a different understanding about CBD in academia and industry [21]. While researchers from academia define components as well defined entities (often small, and easily to understand functional and non-functional features), industry sees components as parts of a system that can be reused, but not necessary well defined with explicit interface and with a weak or not at all conformance to components models. A component can be an amorphous part of system which may require a lot of efforts for adaptation. Still these types of components (or better to say

reusable entities) are of extreme importance, as the bigger components are the higher productivity can be achieved by reusing them.

Component specification is still a research topic. The component standards are mostly concentrated on interface definition part, while non-functional properties are specified (if specified at all) informally in form of separate documentation. Some steps in that direction, by functional characteristics, i.e. interface, and design characteristics, have been done in new Microsoft Component Model .NET.

## **Component-based Development Life Cycle**

CBSE meets the requirements, challenges and problems known in software engineering. Many methods, tools, principles from software engineering can be used in the same of in similar way as for other types of applications or systems. Still, there is one distinction. CBSE covers component development and development with components. There is a slightly difference in the requirements and business ideas in this two cases and different approaches are needed. Of course, when developing components, other components can be (and should be) used, but the main concern is put on reusability: The components are built to be (re)used in many applications, many of them still not existing. A component must be well specified, easy to understand, enough general, easy to adapt, easy to deliver and deploy, easy to replace. The component's interface must be as simple and strictly separated (both physically and logically) from its implementation. Marketing factors play important role, since the development costs will be payed by future (this is especially true for COTS). However, the main problem in developing components belong to acquiring and elicitation of requirements in combination wit COTS selection[21] since the process goes through multi-criteria decision. If the process starts with requirements selection,

there is a high probability that you will never find COTS that meet all your requirements. If you select components too early in the process you may develop a system that does not meet all requirements.

Development with components is focused on reuse, but on identifying reusable entities which will fit system requirements. The early design process includes two essential steps: System architecture specification in terms of functional components and their interaction. This gives a logical view of the systems. The second step is specification of system architecture consisting of physical components.

Different lifecycle models, established in software engineering, can be used in CBD. These models will be modified in the way that the component-centric activities will be emphasized. Let us, consider, for example, the waterfall model using extreme component-based approach. Fig. 1 shows the waterfall model and the meaning of the phases. Gathering requirements and design in the waterfall process corresponds to finding and selecting components. Design corresponds to the system architecture design and component identification/selection. Implementation, test, release and maintain correspond to create, adapt, deploy and replace components.

Fig. 1. The development cycle compared to the waterfall model.

The different steps in the component development process are:

- Finding components that may be used in the system. Here all possible components are listed for further investigation.
- Select the components that fit the requirements of the product
- Alternatively, create a proprietary component that will be used in the system.

- Adapt the selected components so that they suit the existing component model or requirement specification. Some component needs more wrapping than others.
- Compose and deploy the components. This is done with a framework or infrastructure for components.
- Replace old versions of the components with new ones. This is also called maintaining the system. There might be bugs that have been fixed or new functionality added.

There are many other aspects of CBD which require specific methods, technologies and management. For example, development environment tools [22][23], component models and support for their use, software configuration management [24], testing, software metrics, legal issues, project management, development process, standardisation and certification issues, etc. It is beyond this article to discuss them, and we shall concentrate here on software architecture in relation to CBD.

## **Software Architecture and Component-Based Development**

Software architecture and components are closely related. All software systems have an architecture which can be viewed in terms of the decomposition of the system into components, and their relations. A commonly used definition of Software architecture is [27]: *“The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them.”* Traditionally, software architecture is focused on in the early design phase when the overall structure of the system is designed to satisfy functional and non-functional requirements. In monolith applications, the architecture specified in the design process is concealed at execution time in one block of executable code. Component technologies focus on composition and deployment, closer to or at execution time. In a component-based system, the

architecture remains recognisable during the application or system execution, the system still consisting of clearly separated components. The system architecture thus remains an important factor during the execution phase. Component-based Software Engineering embraces the total lifecycles of components and component-based systems and all the procedures involved in such lifecycles.

In a classical approach the design of software starts with architectural questions – structuring the system in smaller as much as possible independent parts. The first phase of this structuring is functionality-based architectural design. The second phase is software architecture assessment during which the software architecture is evaluated with respect to the driving quality requirements. Once the software architecture has been defined, the components that make up the system need to be developed or selected. We can differ different categories of components in relation to requirements of the system: a special purpose components, developed for specifically for the system, reused components, internally developed for multiple purpose, and final commercial components (COTS). Pre-existing components typically need to be integrated into the system through the use of glue code or a modification of the components themselves. This top-down approach ensures the fulfilment of the requirements, or at least a better control of requirements fulfilment. However, this approach does not encourage reuse of pre-existing components, especially not commercial components, since there is a high probability that the pre-existing components do not exactly fit to the system. Another approach, a mix of bottom-up and top-down starts with the system requirements and analysis of possible candidates of components. The component specification and selection have impact on the final requirements and the system architecture. In this case, software architecting is primarily concerned with identifying means for optimising the interactions among the given components. Since basic artefacts for both software architecture and component technologies are components and their assembly it is natural that they

will merge, i.e. use common techniques, methods and tools. Architectural definition languages (ADLs), for example ACME [28], can be used for designing component based-systems and even implemented for the de-facto standard component models.

Software architecture and CBD are successfully used in software product lines development [25][26], where many variants of a product are delivered. Typically product variants contain a set of core-components and a number of additional components. The component-based approach and architectural design play important role in product configuration management.

## **UML and component-based systems modelling**

UML (Unified Modelling Language) is a natural choice for the component and can be used for both component and system modelling, as shown in [20]. Component-driven design concentrates on interface definitions and collaboration between the components through the interfaces. However the design continues with modelling the system with physical components. It can handle about pre-existing components, with already specified interface, which might require wrappers. One logical component, identified in the first phase of design, may consist of several physical components. Finally, there is a deployment aspect, where components may be executed on different computers in a distributed network. UML [29] provides sufficient support for designing component-based systems covering all these aspects[1],[2]. Interfaces are presented as multiple subsystems (also multiple interfaces may be realised by a subsystem), which indicate the possibility of changing the implementation without replacing the interface. An interface can be presented in two ways (see Fig. 2), the second alternative is more common presentation.

Fig. 2. UML component

Fig. 3 shows the three aspects of system architecture.

Fig. 3. Example of different aspects of component-based architecture

UML is however not specialised for CBD and certain extension to standard UML are required (such as naming convention, or stereotypes). The component interfaces cannot be described to such detailed level that it can be directly used. For this reason there exists extension to UML, for example Catalysis [29]. Further work on UML related to CBSE is expected. The next major version of UML (UML 2.0) [31] includes proposals for extensions for describing Enterprise Java Beans, data modelling entities, real-time components, XML components, etc., many of them related directly or indirectly to CBSE.

## **Future Of Component-Based Software Engineering**

It is obvious that CBD and CBSE are in the very first phase of their life. The CBD is recognized as new, powerful approach that will, if not revolutionary, at least significantly change the software development and software use in general. We can expect that components and component-based services will be widely used by non-programmers to build up their applications. Tools for building such applications by component assembly will be developed. Automatic component update over the Internet, already present today in many applications, will be a standard way of application improvement. Another trend we can see in standardisation of domain-specific components on the interface level. This will make it possible to build applications and system from component purchased from different vendors. The standardisation of domain-specific components requires standardisation of domain-specific processes. Work on the standardisation in different is already widely outspread (a typical example is OPC Foundation [32], working on a standard interface that makes possible interoperability between automation/control applications, field systems/devices and business/office applications). Support

for exchange of information between components, applications, systems distributed over the Internet will be further developed. Works related to XML [33] will be further expanded.

CBD approach is especially sensitive for safety-critical domains, real-time systems, different process-control systems where the demands on reliability are higher than in a “standard” web-based application. One biggest problem with CBD is low control of quality requirements of component-based systems and extensive work must be done in that direction.

The goal of CBSE is to standardise and formalise all disciplines supporting the activities related to CBD. The success of the CBD approach depends directly of successful research and implementation of CBSE.

## References

- [1] Brown A. *Large-Scale Component-Based Development*. Prentice Hall, 2000
- [2] Heineman G. and Council W. *Component-based Software Engineering, Putting the Pieces Together*. Addison Wesley, 2001
- [3] ICSE 2000, Workshop on Component-Based Software Engineering (CBSE 3), <http://www.sei.cmu.edu/cbs/cbse2000/CFP2000.html>, Access date 2001-07-14
- [4] ICSE 2001, Workshop on Component-Based Software Engineering (CBSE 4), <http://www.sei.cmu.edu/pacc/CBSE4-Proceedings.html>, Access date 2001-07-14
- [5] ECOOP 2000, Workshop on Component-Oriented Programming, <http://www.ipd.hk-r.se/bosch/WCOP2000/>, Access date 2001-07-14
- [6] Euromicro 2001, Workshop on Component-Based Software Engineering, <http://www.idt.mdh.se/ecbse/>, Access date 2001-07-14

- [7] Workshop on CBSE – ABB Corporate Research Centre, Switzerland, 2000,  
[http://icawww2.epfl.ch/~opreiss/CBSE\\_Conference2000/](http://icawww2.epfl.ch/~opreiss/CBSE_Conference2000/), Access date 2001-07-14
- [8] Gartner Group, <http://www.gartner.com>, Access date 2001-07-14
- [9] Brown A. and Wallnau K., The current state of CBSE, *IEEE Software*, 1998
- [10] Crnkovic I. and Larsson M, A Case Study: Demands on Component-based Development, ICSE 2000 Proceedings, ACM Press, 2000, 23:31
- [11] Szyiperski C. and Pfister C., Workshop on Component-Oriented Programming, Summary. In Mühlhäuser M. (ed.) *Special Issues in Object-Oriented Programming – ECOOP96 Workshop Reader*, Springer 1997
- [12] ICSE 1999, Workshop on Component-Based Software Engineering (CBSE 2),  
<http://www.sei.cmu.edu/cbs/icse99/cbsewkshp.html>, Access date 2001-07-14
- [13] Szyperski C., *Component Software –Beyond Object-Oriented Programming*. Addison-Wesley, 1998
- [14] Box D. *Essential COM*, Addison-Wesley, 1998
- [15] Microsoft Component Object Model, <http://www.microsoft.com/com/>, Access date 2001-07-14
- [16] Microsoft .NET Component Model, <http://www.microsoft.com/net>, Access date 2001-07-14
- [17] Enterprise Javabeans technology, <http://java.sun.com/products/ejb/>, Access date 2001-07-14
- [18] Matena V. and Stearns B *Applying Enterprise JavaBeans(TM): Component-Based Development for the J2EE(TM) Platform*, Addison-Wesley, 2000
- [19] OMG, CORBA, <http://www.omg.org/corba>, Access date 2001-07-14

- [20] Cheesman J. and Daniels J. *UML Components – a Simple Process for Specifying Component-Based Software*, Addison-Wesley, 2001
- [21] Maiden N. and Ncube C. Acquaering Requirements for Commercial Off-The\_shelf Package Selection, *IEEE Software*, Vol. 15, No. 2, Mar., 1998
- [22] Microsoft Visual Studio, <http://msdn.microsoft.com/vstudio/>, Access date 2001-07-14
- [23] Development tools – Forte <sup>TM</sup> tools, <http://www.sun.com/forte/>, Access date 2001-07-14
- [24] Larsson M. and Crnkovic I., New challenges for configuration Management, Proceedings of 9<sup>th</sup> Symposium on System Configuration Management, *Lecture Notes in Computer Science*, Springer, 1999, 232-243
- [25] Bosch J. *Design & Use of Software Architecture*, Addison Wesley, 2000
- [26] Bosch J. Software Product Lines: Organisational alternatives, *ICSE 2000 Proceedings*, ACM Press, 2001, 91-100
- [27] L. Bass, P. Clements, R. Kazman, *Software Architecture In Practice*, Addison Wesley, 1998
- [28] ACME architecture definition language, <http://www.cs.cmu.edu/~acme/>, Access dated 2001-07-14
- [29] Booch G., Jacobson I and Rumbaugh J. *The Unified Modeling Language User Guide*, Addison-Wesley, 1998
- [30] D'Souza D. and Wills A., *Objects, Components, and Frameworks With UML : The Catalysis Approach* , Addison-Wesley, 1998
- [31] OMG UML, <http://www.omg.org/technology/uml>, Access date 2001-07-14
- [32] OPC Foundation, <http://www.opcfoundation.org/> , Access date 2001-07-14

[33] Extensible Markup Language (XML) <http://www.w3.org/XML>, Access date 2001-07-14