

Definition of Software Construction Artefacts for Software Construction

M.Zinn, G.Turetschek and A.D.Phippen

Centre for Information Security and Network Research, University of Plymouth,
Plymouth United Kingdom

e-mail: marcuszinn@gmx.de; g.turetschek@fbi.h-da.de, ahippen@cisnr.org

Abstract

The definition of a service based software construction process, published in 2007 (Zinn, 2007), builds on the use of software construction artefacts (SCA). These artefacts form useable building blocks (units of modelling) for software, which can be implemented in a software product by a service. The building blocks are categorized in four different types (GUI, data, function and structure). The foundation for the use of these artefacts is the method of the software construction process which constructs the software similar to the methods used in computer aided design (CAD).

This paper refines and expands on the definition of the artefacts to present the current state of research in software construction procedures. It will show how the artefacts are constructed and how they can be used in software construction. In addition the software construction artefacts will be compared to components, modules and objects. Based on this definition of the application area, further potential scenarios for the use of the artefacts, as well as their properties and points still being worked on, are shown.

The purpose is to create a more exact description, definition of the software construction artefacts to obtain a better understanding of the software construction process.

Keywords

Computer Aided Design (CAD), software construction process (SCP), software engineering, (web) services, software construction artefacts (SCA) and software construction services (SCS), model driven development (MDD)

1. Introduction – about software construction

(Zinn, 2007) shows how software can be “constructed” with a service based software construction process (SSCP). The SSCP consists of three phases. In the first phase, the requirements and information is gathered and converted into an “outer structure” (i.e. source code) by use of a transformation based on model driven development (MDD). In the second phase, this structure is filled with Software Construction Artefacts (SCA). The third phase is optional for the creation of SCAs which have not yet been developed. This procedure is based on the computer aided construction which is used in aircraft and automobile construction. Single elements (parts) are combined to bigger components (products) here. Parts are collections of geometrical shapes (i.e. lines, circles, etc.) and knowledge elements (i.e. parameters, formulas, etc.) (cf. Anderl and Gräß, 2000). An automobile manufacturer, for example, can

develop a car frame in his graphical CAD application. Additional parts, which are not built by the manufacturer himself, can be bought from a third party and integrated into the frame in the application. These external parts (components) are provided with knowledge (i.e. special parameters) to create the according variances of the component. At the assembly, which follows later, the components are delivered by the subcontractor and are integrated by the manufacturer. In the first step, the SSCP creates such a frame and in the following steps fills it with parts (in this case SCAs).

In addition, this procedure is based on the component based software development (CBSD) which combines components to programs or bigger components (cf. (Szyperski, 2002)). Contrary to the CBSD, the SSCP engages at another point of the software lifecycle and thus has another target group. The primary focus of the CBSD is on the components during the runtime of the product. Other service based software construction approaches also follow this path (cf. Stojanović, 2005). The basic idea of the SSCP, however, is to alter only the view of the software developer or designer but not to change the processed components or the final product. This task is assumed by the SCAs. Therefore, they form the core of the SSCP and possess the following features:

- SCAs, in this case software construction artefacts, are of a predefined type
- There are only four possible types of SCAs
- SCAs are provided by special typed based communication services
- Information that defines at what point of time a component is available
- Behind an SCA there can be Source Code, Binary Code or a Service
- SCAs can be combined to achieve a higher purpose

A first definition can be defined by summarising the base publication: “A software construction artefact (SCA) is a type based building block, which represents the foundation of the software construction process.” The definition is expanded by describing the four possible types of artefacts (data, functions, structure and GUI).

2. Aims of this paper

This paper is the second status report of the Ph. D. research of the author. It shows the results and the current state of research about software construction artefacts in the context of ongoing research in the area of “service based software construction”. It can be traced back to (Zinn, 2007). This reference is used as the underlying bases for this work. The purpose is to examine and differentiate, show detected weak points, describe application areas and realise the current definition of the software construction artefacts. This includes research on following topics:

- Integration of artefacts
- Showing time of availability
- First draft of a technical specification
- Showing the supply of artefacts
- Examples of using of artefacts

The research points “Description of the interface of artefacts to the outer structure” and “Limit of technology” are not described in this paper, but this is part of the on going research. In addition to these points two other essential questions appear: “What separates the SCA from a conventional component?” and “What kind of software can be developed with SCA?”

3. Software Construction Artefacts

(Wang and Fung, 2004) shows a comparison between object, component and service oriented software engineering. Each methodology has units of modelling (i.e. Objects are the units of modelling for object oriented software engineering). SCAs are the units of modelling in the “Service based Software Construction” approach. The following discussion considers the issues regarding SCA and is divided into three sections – The first shows the SCA from the view of a developer or designer, the second explores the relationship between SCA and other units of modelling and the last contains a first draft of SCA specification.

3.1. Application of artefacts in software construction

The basic task for the SCA is to grant the software developer / designer) a simplified view of given information and functionality. To use SCA as a unit of modelling the designer has to consider several points: the process of modelling, time of use and availability.

The process of modelling: SCA are used as unit of modelling. In the software construction process approach the developer uses an outer structure to fill in different SCA instances. The outer structure is defined by the developer and a customer for example as a process structure, which is presenting the resulting application. This (process) description will be transformed (by using MDA rules) into the outer structure (i.e. a source codes construct). By using graphical tools the developer add some SCA using a “Drag & Drop” approach. Each SCA has to be configured. At the end the structure is filled and configured and can be compiled into a real application.

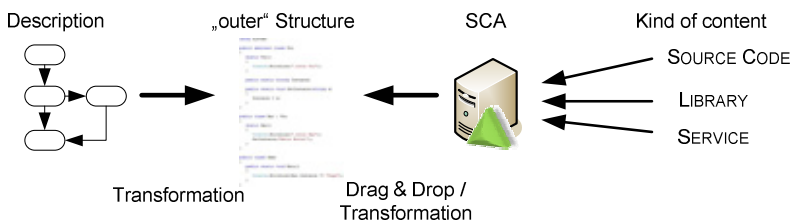


Figure 1: SCA Content Transformation

Each SCA can have different content: Source Code, Binary Code and Services. The developer has to decide how he wants to use an SCA. The content will be automatically transformed and configured by the developer to be a part of the outer structure. I.e. If the developer decides to use the content of an functionality SCA as Source Code the system will give the developer the source code information. Figure

1 shows the artefacts of an SCA transformation. The content, possibilities and services of an SCA are defined by the SCA-Developer. The topic “SCA-Transformation” is part of the ongoing research.

Time of use: One of the basic questions a designer has to ask of himself is: “At what point in time do I use the SCA?” When looking at the different approaches such as SOA, OOP, CBD (Cervantes and Hall, 2005; SEI, 2007; Meyer, 1998; Singh *et al.* 2005) it becomes clear, that information has to be available at different points in time in the lifecycle. There are generally three points in time, when a “part” of software is needed: Time of construction, time of compilation and runtime. In the “normal case” of project oriented programming all classes and methods are available at the time of construction. The developer can work with all classes. Other concepts (e.g. SOC/SOA) define that information and functionality are not available until the time of execution. In most cases of component oriented software development however, only an interface has to be present at the time of compilation. To support the known approaches the SCA process is required to provide the SCA to any defined point in time. At what point in time the designer needs a specific artefact depends on the proceeding of the individual designer and the underlying project. Therefore the personal need of SCAs can deviate from the points in time. I.e. (Kumar and Pragallapati, 2007) show the need of services during construction (design), compilation and runtime.

The transformation represents the use of model driven development (MDD) and model driven architecture (MDA). As shown by (Meimberg, 2006), MDD and MDA are usable on different levels of software development. The use of MDD and MDA respectively within the construction process / a component environment has not yet been analysed. The transformation used here refers to the approach of (Meimberg *et al.* 2006). The transformation, like the interaction, depends on the respective implementation. The transformation however is based on the technology information provided by the SCA and can be used at any point in time.

The partial study of: automation of the construction with SCA has not yet been conducted. The basic conditions for the use of SCAs in the construction process however have already been defined (for the basic conditions for Runtime see chapter “Runtime”):

- the existence of a structure to implement the SCA in (i.e. source code)
- the existence of a transformation rule to integrate the SCA
- the existence of the SCA infrastructure (repositories)
- the expansion of the development environment for the use of SCA

Time of construction: At the time of construction, the designer has access to all SCAs and can decide freely on when to use them. The designer uses a set of SCAs provided by repositories. During this “drag & drop” of SCAs into the existing code, the SCAs are being transformed from SCA-(meta)-information into the code. The designer decides how (i.e. as source code or service) and where (i.e. in a specific method) he wants to implement the information. How exactly the interaction

between the development environment and the designer looks, depends on the respective implementation.

Time of compilation: The compilation represents a significant point in the deployment of SCAs. There are several different possibilities on how to get from the construction to the final product.

Scenario 1: Standardised compilation ($A1 \rightarrow B2 \rightarrow C3$) If the compiler first translates all SCA information, by use of the technological specifications of the corresponding SCA, into known standards and then starts the compilation, the result is a “conventional” product. This calls for all information needed for the translation to be present as SCA meta information.

Scenario 2: SCA based compilation ($A1 \rightarrow B1 \rightarrow C1$) In the second possibility, the compiler only uses the SCA information. The final product also uses SCA information to request/check for further information. This result is called “software construction product (ScP).”

Scenario 3: SCA based and standardised compilation ($A1 \rightarrow B1 \& B2 \rightarrow C2$) In the third possibility, standard components and SCA components are compiled together. The result of this approach is called “hybrid software construction product” (HScP). Scenario 1 is the target of the SCA approach. In this case only the “view” of the designer is simplified to the point where he can “construct”. For the compilation only the translation of the information into compiler known standards is necessary. This possibility only affects the designer but not the following software development process. In the context of this study a first prototype following this approach was created in 2006 and successfully tested. This methodology is the precondition, for using SCA to build all kinds of applications (i.e. office and game applications). Scenarios 2 and 3 however require alterations in the compiler as well as in the product. In the case of the compiler, all interfaces have to be fully translated to be available for the product. In addition the SCA repositories, which were only used for the construction in possibility 1, are now required to be integral parts of the software product. This heightens the contextual dependency. (cf. chapter “Problem of contextual dependency”).

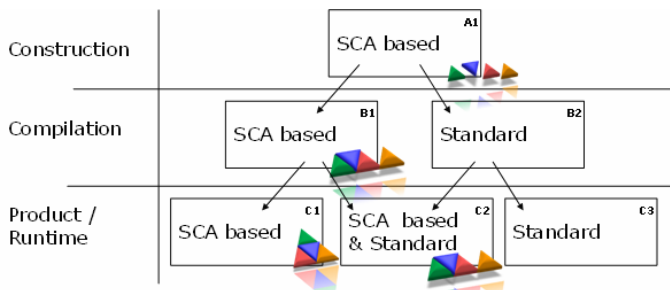


Figure 2: SCA based and standard development

Runtime: The first scenario, shown in the “Compilation” chapter, provides a “conventional” software product. For the user, there is no difference to a product

created without SCA construction. It is different with scenarios 2 and 3. Products created with these scenarios possess special dependencies. On the one hand dependencies on specific (local) libraries exist since the SCA functionality uses them to grant the product the ability to handle internal and external SCAs. On the other hand, depending on the implementation, the product is capable of accessing external SCAs and using their information/functionality during its running time. An SCA repository has to be present for this. In contrast to the time of construction, during the running time the user has no knowledge of the use of SCAs.

Availability: To assure the access to an SCA a so called software construction service (SCS) was defined. Each SCA can have a SCS. A SCS provides the function and information of an SCA to the outside and screens the user from the actual implementation-/ execution unit and from the position of the artefact. The form of the implementation is not predetermined as long as the information of the SCA is retrievable and executable. In the scope of the previous research, SCA and SCS are firmly connected. It is aspired however, to connect the units only loosely, i.e. the SCA is placed on a system and the SCS should be a part of a repository, which can be hosted by another system. The target is to switch the administration to a more abstract level like it is done in the component oriented approach (Stovanović, 2005). Through the aid of accumulative units for SCS, so called SCS-Repositories, the administration (i.e. versioning) and handling (i.e. access management) of SCS is made possible for the designer. Figure 3 shows that repositories can be used not only locally (within the department- and company borders) but also in a global context. Through this it is possible to adapt the availability of SCA to the company structure. Figure 4 shows that a user can query an SCA and can also create and insert data into a repository by use of a SCS. With enough “starter SCS” the complete system “software construction with SCAs” can be self sustaining. For this kind of repository query, a method based on the UDDI (Melzer *et al.* 2005) is used for this study.

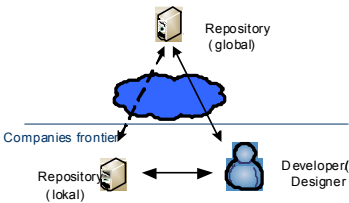


Figure 3: Repository /global and local

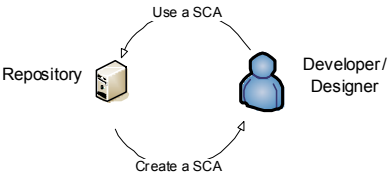


Figure 4: SCA life cycle

3.2. Components, objects, modules and services vs. SCA

There are two possible perspectives on SCA. The first is an SCA as a unit of construction and the second is the content carried by an SCA. Because of an SCA can be seen as component, object, module or services in both perspectives, it is important to compare SCA with these units.

Components: When comparing components to SCA it is easily assessed that SCAs are actually components. Nevertheless the differentiation depicted by the underlying publication is still correct. SCAs are components that differ in their characterizations from conventional components. (Szepersky, 2002) and (Stojanović, 2005) shows several different definitions of components, for the research the definition of (Szepersky, 2002) will be used: *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition of third parties*”

		Program/ script languages	Libraries	Interfaces	Messages and protocols	Frame- work	System architecture
SCA-Type	Data	X	X	(X)	X	(X)	
	Function	X	X			X	
	Structure	X	X	X	X	X	X
	GUI	X	X			(X)	

Table 1: Design level and SCA types

The main target of SCAs is the simplification of the view to get from the development to the construction. The result can be an ordinary program (cf. chapter “construction”). Normal components however, have the target to use a functionality or information as an external unit. So SCAs target to another point of a software lifecycle: the development phase. The content of an SCA also has some differences to record. On the one hand an SCA can offer its content in different forms (cf. chapter “specification”), on the other hand an SCA can carry several different pieces of design level information. Table 1 shows the results of the study of SCA types and design level information. SCAs and SCSs are, like other components, part of a component model. Actually component model is defined as follows: Definition: *“A component model defines a frame for the development and execution of components. The frame makes structural demands concerning link and composition possibilities as well as behaviour oriented demands concerning possibilities of collaboration on the components. Further, through the use of a component model an infrastructure is provided, which can implement frequently needed mechanisms like distribution, persistence, message exchange, security and versioning.”* (Gruhn and Thiel, 2000) An allocation to a component world is not planned and dependent on the corresponding implementation.

Comparing to the problem with the component markets, SCA exist in a vertical market. The target group “designers” and the target area “software construction / development” define this narrow market. Another differentiator is SCA types (cf. SCA types). They are mainly used for the division of the SCAs into a specific context and a functional content. Therewith SCAs deviate from the conventional description of a component (like it is shown for example in the Microsoft COM technology). Generally components do not have a state. If it has a state it is not detectable from the outside (cf. table 1) (Szepersky, 2002), but the author do not correspond to this opinion. SCA do have a state to adapt themselves to the new

requirements for components (Heckmann, 2007) like i.e. policies, monitoring and provisioning. The most important difference between an SCA and a component is in its use. The sections “Construction” and “Running time” show that an SCA can implement more than one kind of use. By the use of MDA transformation, the reuse can even be heightened. By using transformation the content can be an ordinary component.

	Objects	SCA
Have a state	YES	YES
Have a state controllable from the outside	YES	YES
Have basic description (i.e. class)	YES	NO
Instancing needed	YES	NO

Table 2: Differences between Objects and SCA

Objects: The perspective of SCA is the content of the SCA. In the section “construction time” it is shown that the content of the SCA can be used as an object for itself (i.e. when the content of a structure SCA is a class from which in a further process an object is created). Significantly more interesting is the second case: the comparison between an object and an SCA as a component. Table 2 shows the most important differentiators between objects and SCAs. In addition the properties are compared to SCAs. For the designer an SCA is not an object, it is a (standardised) means to an end to obtain the desired data. The two alternative methods of use, shown in chapter time of compilation pose an exception to this. If SCA information is not transformed, but reused in the compilation and the product respectively, SCA components present themselves as components (in an object oriented environment). This case has not been studied in the current research, but is part of the research plan.

Modules: The presentation of modules defined by (Wirth, 1989) is very conformable with the SCA since SCAs are able to encapsulate module information. The statement that an SCA is a module is nevertheless wrong and can be used as content information like objects (see above).

Service: Because of the SCA is a service the first perspective of SCA is simple. To use a service as a content of an SCA is also very simple. In this case the SCA needs a service description. The research plan includes also checking the current research questions for services (i.e. Quality of Service (QoS) , Service Level Agreement, life cycle, dynamic binding, instantiating and security etc. (Papazoglou *et al.* 2007). These questions can have influence over the SCA research and the result. The research in the service area will be done in the near future.

3.3. Specification

The specification is under development and based on the requirements description of component architecture of (Rütschlin, 2000). An SCA is divided into four segments: Meta information, general SCA functionality, meta information for the implementation and technological specification.

Meta information: The potential of SCA (discussed above) assumes that the user and the provider of the SCA have access to certain information or are provided with it. The following meta-information is, at the current state of development, necessary:

Name of the SCA: The name of the SCA is only used as an identifier for the user or administrator. It eases the use of the SCA.

Type of the SCA: The Type decides on the general context of the artefact. It is differentiated into UI, data, structure, function and user interface (cf. “SCA-Types”).

Possibilities of (re)use: At the current point of research there are three commonly used kinds of application: *IsLibrary*, *source code* and *service*. Thereby it is possible for an artefact to have several possible uses, for example as “service” and as “source code”. *IsLibrary* indicates to the user that the artefact can be referenced as a library. This reflects the way most software products are built. The SCA acts as *source code* when it offers its content as source code. (Välimäki, 2005) shows the idea of distributing source code in more detail. The literature also shows the necessity for the exchange of source code (Chávez *et al.* 1998). The third point *service* describes the possibility of the SCA functionality to work as a service, similar to today’s web services (Singh *et al.* 2005).

Version information: An important point with the partial or full use of software is the version. Therefore it is necessary for the SCA to carry its full versioning information (Baerisch, 2005; Sommerville, 2004): ID (unique identifier), Version number (unique version ID), Time of build (identification of creation time), Last change (time of the last change)

Common SCA functionality and Meta information of the implementation: The common SCA functionality is at the current time limited to: the query of meta information, the query of the SCA as source code, the query of the SCA as library and the query of the SCA as a service. The most important information is the details of the implementation about an SCA. Since an SCA essentially only aims at influencing the development perspective, but not the final product, it needs to contain the *Description* information. Essentially the designer/developer needs a description of the provided function. At a later point in time additional information like policies, monitoring and provisioning (Heckmann, 2007) Quality of Service (QoS) , Service Level Agreement, life cycle, dynamic binding, instantiating and security etc. (Papazoglou *et al.* 2007) will become interesting. The current state of research however deals with the assembly of artefacts.

Technology specification: The content of the technical specification has to contain enough information for the designer to adapt his program as easily as possible or even to automate the process. At the current time, this area has not been researched further. It is clear however, that in this area the usability of the SCA-theory as a component framework can be seen. (Szyperski, 2002)

SCA-Types: The underlying problem with the classification of components like the Software construction artefacts is described by (Czarneck and Eisenecker, 2000).

Thus the definition of components (cf. table 2) is expanded by showing components as a natural concept, i.e. a component cannot be defined fundamentally and used identically everywhere. This is in contradiction to the concept of objects since these software artefacts are, according to this definition, an “artificial” concept. Objects have a classical definition with sets of necessary properties defining them. The same problem was focused by (Kumar und Pragallapati, 2007) for next generation services and solved with types for services. This research project acknowledges the problem described. SCA uses meta information (see above) and interfaces to solve the specialization problem. The general division in different context areas, similar to the context information of brokers within UDDI (Dostel *et al.* 2005), is however useful for the application.

Data	The data is all the information that is used. In comparison to functions, data-SCAs have low to no “cost” of data acquisition.
Functions	The function-type describes functions and methods. They are local and/or external present.
Structure	The structure type is a carrier of structure information in the form of interfaces, class structures, muter and architecture guidelines.
Graphical user interface	This type provides a (graphical) user interface. Lightweight (e.g. Extended Markup Application language (XAML) and Scalable Vector Graphics (SVG)) and library based (i.e. windows forms) technologies are best suited for this.

Table 3: Type of SCA

Software construction artefacts can be differentiated into four types, depending on their internal context: data, function, structure and (graphical) user interface. (Rütschlin, 2000) define these types as the basis parts of a component which represents an application. These types are the basic building blocks according to underlying publication. This differentiation serves the designer as an indicator for the software technical content of the artefact. Therefore the view and expectations are limited to these four types. The same principles are being used with components (Beans) in the area of J2EE (DeMichiel *et al.* 2006) and in the area of computer aided design (CAD) (Anderl and Gräß, 2000) and (Brill, 2006) with different adapted limitations of the view.

4. Conclusion and Future work

The description of software construction artefacts, presented in this publication, shows that typified artefacts are usable in the software engineering environment and offer an added value compared to conventional units of modelling. This kind of unit of modelling is new in the area of software construction. But this publication is only an overview and it is an early state of research.

The target to clarify the open points and to adapt the definition has been reached. Many of the open questions could be answered by the use of practical presentations in the chapter “Application of artefacts in software construction”. This section shows that SCAs are components which are wrapping ordinary information like source code, libraries and services. The developer uses the SCAs (in a drag & drop

approach) and the system will transform all information into a given language. The result can be transformed/compiled into a real application. Essential questions on the specification have been analysed in the chapter “Specification”.

With the newly acquired information in reference to expansion, description and differentiation of software construction artefacts the following definition can be reached: *A software construction artefact (SCA) is a typified unit (component) of modelling, which creates the basis for a unified view on data, structure, functions and interaction possibilities. The software construction artefact presents only a simplification of the view and handling at the time of construction to support the designer. It does not necessarily influence the outcome of the construction on a technological level. Software construction artefacts possess their own component model which predefines an interaction- and composition standard to enable an orchestration of the encapsulated information and functionality respectively.*

This adapted definition serves as a basis for the continuing research. By using the shown results of this paper, following steps are planned:

- Research into SCA using object oriented and component information
- Further specification of SCA, SSCP, the “outer structure” and the “Limit of technology”
- More Research in the specification of using services
- A closer comparison between SCA and other service based approaches
- Practical test of the SSCP approach

5. References

Anderl and Gräb (2000), “Parametrics: A Key Technology for Efficient Development of Virtual Products”, *Prostep Science Day Conference*, Germany, Darmstadt, ISBN: 3-8167-5585-2

Baerisch, S. (2005), “*Versionskontrollsysteme in der Softwareentwicklung*”, IZ-Arbeitsbericht Nr. 36, Informationszentrum Sozialwissenschaften, Bonn

Brill, M. (2006), *Parametrische Konstruktion mit CATIA V5*, Hanser Verlag, Sindelfingen, Germany, ISBN: 3-446-40705-7

Cervantes, H. and Hall, R. (2005), “Technical Concepts of Service Orientation” subchapter of *Service Oriented Software System engineering Changes and Practises*, Idee Group Publishing, London, ISBN: 1591404274

Chávez A., Tornabene C. and Wiederhold, G. (1998), “Software Component Licensing. A Primer”, *IEEE Software*, Volume 15, Issue 5, pp. 47-53.

Czarneck, K. and Eisenecker, U. (2000), *Generative Programming*, Addison Wesley, Indianapolis, ISBN: 0-201-30977-7

DeMichiel, L. and Keith, M., (2006), “*JSR 220: Enterprise JavaBeans™, Version 3.0*“, EJB 3.0 Expert Group, Sun Microsystems, Available from <http://java.sun.com/products/ejb/>, (Accessed 4 May 2008)

Heckmann, B. (2007), "Service provision in a utility computing environment", *Third International NRG Research Symposium*, pp. 185-197, Network Research Group, University of Plymouth, Plymouth, England, ISBN: 978-1-8410-2173-7

Meimberg, O. and Petrasch, R. (2006), *Model Driven Architecture*, dpunkt, Heidelberg, ISBN: ISBN 978-0-471-31920-7

Melzer, I., Dostal, W. and Jeckle, M. (2008), *Service-orientierte Architekturen mit Web Services. Konzepte – Standards – Praxis*, 2nd Edition, Spektrum Akademischer Verlag, Netherlands, ISBN: 978-3-8274-1885-2

Meyer, B. (1988), *Object-Oriented Software Construction*, 2nd Edition, Prentice Hall, Santa Barbara, ISBN: 013-6291-554

Papazoglou M. P., Traverso P., Dustdar S. and Leymann F. (2007) "Service-Oriented Computing: State of the Art and Research , available Challenges", pp. 38-45, Online available from <http://ieeexplore.ieee.org>, accessed 29. December 2007

OMG (2004), "Common Object Request Broker Architecture: Core Specification", available from <http://www.omg.org/docs/formal>, [10 May 2008]

Rütschlin, J., (2000), "The Requirements for Component-Based Architectures", *Prostep Science Day Conference*, Germany, Darmstadt, ISBN: 3-8167-5585-2

Singh, M. and Huhns, M. (2005), *Service Oriented Computing Semantics, Process, Agents*, Wiley, England, ISBN: 0-4709148-7

Software Engineering Institute (SEI) | Caregie Mellon University (2007), "Component-Based Software Development /COTS Integration", available from <http://www.sei.cmu.edu/str/descriptions/cbsd.html> (Accessed 7 July 2008)

Sommerville, I. (2007), *Software Engineering*, 8th ed. Chapter 29. Addison-Wesley, Kösel, Germany, ISBN: 382-7370-019

Stojanović, Z. (2005), "A Method for Component-Based and Service-Oriented Software Systems Engineering", Delft University of Technology PHD Thesis, Delft, Netherlands, ISBN: 909-0191-003

Szyperski, C. (2002), *Component Software Beyond Object-Oriented Programming*, 2nd Edition, Great Britain , ISBN: 020-1178-885

Välimäki, M. (2005), *The Rise of Open Source Licening*, Turre Publishing, Finland, ISBN: 952-9187-696

Volker, G. and Andreas, T. (2000), *Komponentenmodelle DCOM, Javabeans, Enterprise Java Beans, CORBA*, pp. 293, Addison-Wesley, ISBN, 382-7317-24X

Wirt, N. (1989), "Programming in Modula-2", 4th Edition, Springer Verlag, Berlin, ISBN: 054-0150-781

Wang G. and Fung C. K. (2004), "Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems", *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, Online available from <http://ieeexplore.ieee.org>, accessed 29. December 2006

Zinn, M. (2007), "Service base software construction process", *Third International NRG Research Symposium*, pp169-184 Network Research Group, University of Plymouth, Plymouth, England, ISBN: 978-1-8410-2173-7