Component Based Software Development: Processes and Problems

Wes J. Lloyd
Computer Science, Colorado State University
Fort Collins, Colorado, USA 80521
wlloyd@acm.org

Abstract: Component-based software development (CBSD) is emerging as a new software development paradigm that promises to revolutionize how to approach reuse in software development. CBSD promises to allow developers to develop entire software systems by simply integrating together commercial off-the-shelf software components or internal components from company repositories. This paper presents an introduction and overview to CBSD. In this paper the general view of what it means to be developing software using the CBSD paradigm is discussed including several challenges impeding the realization of maximum benefit from using component based development techniques. Development processes tailored to the unique needs of component-based development can help address many of these challenges. A summary and comparison of several existing component based software development processes is presented. This paper concludes by summarizing the capability of existing processes and by suggesting how future process enhancements may better address the challenges of component based software development.

Key Words: CBSD, CBSE, CBD, components

1. Introduction

Components are objects in object-oriented software that emphasize encapsulation, abstraction and reusability. Developers have been developing software-using components for several years now. For example an application developer may wish to present data in a spreadsheet like display in a graphical user interface. This developer instead of spending time designing and developing a customized spreadsheet like display uses a commercial component that easily plugs into the application and provides the required functionality. The cost of the component is small, generally a few hundred dollars, where the development time, and hence the cost to create custom code to implement the spreadsheet functionality is high. Development managers know about the timesavings of developing software-using components and frequently ask the question "is there a tool out there that will do this for us?" Finding commercial components to provide functionality similar to that in this example is much faster than "reinventing the wheel", rewriting the same code that was previously written. In this case, writing custom code to implement a spreadsheet display could take several weeks or even months to complete depending on the extent of the requirements. Using a component to realize the functionality will be somewhat faster with development activities for the functionality piece limited to integration and testing.

Component based software development is the development of software through the integration of preexisting components. The definition of exactly what is a software component, is often as varied as the scope of the software in which they are used in, but in general a component is considered to be a separable piece of executable software that stands alone as a unit, can inter-

operate with other components, and is accessible only through a published interface. Today many software components are developed using a technology specific component model. A component model (CM) defines the building blocks and methods by which components interoperate. Components require a component model just as algorithms require a computer language in order to realize their implementation. Without a component model components cannot work or interoperate together. [3] Popular component models include:

1. Microsoft Windows based CM: Microsoft Transaction Server (MTS), Component Object Model (COM), .NET
2. Java based CM: Javabeans and Enterprise Javabeans (EJB)
3. Cross-Platform CM: Common Object Request Broker Architecture (CORBA)

The MTS/COM/.Net component model allows for the development and deployment of components for operation with Microsoft operating systems and development languages. EJB and Javabeans are component models for component based software development in the popular cross-platform programming language Java. CORBA is the leading component model that supports the development and deployment of cross-platform language independent components. [20]

Components can vary greatly in functionality, purpose, and scope. Some components are concerned only with presentation and are considered graphical user interface components, while others are data related and provide content and information for various server applications. Still other components exist as middleware, software that enforces and implements business rules and logic that separate the presentation and user interaction from the application's data model.

Component Based Software Development (CBSD) has two commonly realized variants. [19] Some view CBSD as the practice of selecting commercial off-the-shelf software components and integrating them together in a similar process as described in the example above. [5] The development of such systems that relies heavily on using external, of the shelf components relies on selection decisions in the design process. Designers focus attention on selecting the best component available to meet the system requirements. Even small companies without internal component development group can benefit from CBSD in this way.

The other popular view of CBSD is for each company to have a component software group. This group is responsible for the internal development of custom components. Instead of developing custom software only to meet specific project requirements, commonalities across projects are identified and used to drive the development of components within an organization. These components are then deployed and reused across many projects at a company. It seems sensible for a company developing a family of products with similar user interfaces to use this approach. In [14] Kang suggests that companies form a central support group to identify cross-cutting requirements among related systems to develop and promote component use. One challenge is justifying the cost of component software groups within a company. Proving the cost effectiveness of component development groups is likely to be as difficult as measuring the return-on-investment of any new software practice.

2. Overview of CBSD Processes

In order to define, design, and implement software products it is generally accepted that higher quality systems result when the software development organization applies a well understood software process. Component based software development is a significant new paradigm for software development. Several activities unique to component based software development must be performed in addition to traditional activities. These new activities include but are not limited to: identification of common cross-project requirements, component specification, component selection, component optimization, and component integration. It is impractical to define a single all inclusive software process for CBSD. There are many possible process variants for CBSD and any good CBSD software process should be customized for the organization in which it is to be used.

Traditional software development processes do not address the unique needs and requirements of CBSD. [16] This has encouraged the development of several component based software processes which intend to focus on the unique aspects of CBSD during software development. Some challenges unique to component based software development which new processes should attempt to address include:

- Component Specification Documentation is informal and lacks detail – Commercial components and components developed within an organization often lack consistent formal specifications. Specifications are limited to natural language descriptions describing basic functionality. It is difficult to understand the performance characteristics and testing implications with limited access to the internal implementation details of the components. Presently there are no formal methods for specifying non-functional requirements such as performance and reliability. [7] [10] [5]
- Components are volatile – Components, especially commercial ones are subject to frequent changes, updates, bug fixes, etc. Components can also grow in size and complexity over time as features are added, and bugs are fixed. The ever-changing nature of components can create challenges in component integration and the maintenance of component based software systems. [13] [5] It is desirable to perform system-wide regression testing when individual components are upgraded to newer versions. Dependencies and interoperabilities between components may change as components are upgraded to newer versions.
- CBSD currently favors building new components – Presently, CBSD efforts seem biased towards building components from scratch to meet the requirements for the projects they are to be used in. Components are commonly viewed as a product of software development efforts rather than an integral part of software specification and design. [1] In order realize the time and money saving benefits promised by CBSD more focus needs to be placed on component searching, discovery, and selection activities in component based software processes. [19]
- Lack of commonality between component models – Presently there exists numerous component models in which components can be developed. There is a general lack of inoperability between these standards that restricts the reuse of components cross-models. In order for component-based software development to prosper a unified standard is desired. [5] [16] [20]
- Difficult to test and retest integrated components – The lack of design and specification details for black-box components can lead to difficulties in writing tests that fully

exercise the component's code. Unless hooks are available to access private attributes and component details, testing is restricted to running tests via public interfaces. Component testing is also difficult because it is hard to inspect internal component variables. Only the resultant variables returned through the component's interface are easily available for verification against a test oracle. [7] [9]

This paper presents a summary of some existing processes described in the literature, and tries to point out their advantages, disadvantages as well as commonalities among them.

3. Component Based Software Development Processes

Many of the prevailing component based software processes recognize two distinct and separate activities in software development: [9] [14] [19] [21]

- Design and Development of components
- Design and Development of application software using components

A recognized problem with CBSD is that present component models and development methodologies are typically focused on the implementation and deployment of components. Components are often the result of software development rather than an integral part of the process. [1] In order to realize maximum benefit from CBSD, component processes should consider the design and development of components apart from the design and development of the immediate application software in which they are to be deployed. The focus of component development within organizations should move from the immediate needs for a specific project's requirements to the cross cutting needs for many projects' requirements.

3.1. Cheesman and Daniels Component-Based Software Development Process

Cheesman and Daniels define a component-based software process centered on these activities [3]:

Requirements Identification: Projects requirements are first identified using a business concept model and a use-case model.
System Specification: From the requirements specification, potential software components are identified, then the component interactions are specified, and finally a complete specification of the required components is made.
Provisioning: Provisioning is the phase which involves acquiring the components. Components meeting the requirements can be implemented from scratch, found in an existing archive, or purchased from the open market. Preexisting components will require integration with varying degrees of adaptation. Adaptation may involve alteration of the component or various ad hoc activities to allow the component to comply with requirements.
Assembly: Once all components are implemented or found, the system development involves integration of components, application logic, and user interface aspects to form a working application

Cheesman and Daniels' process does not center on the need to identify commonalities of applications within a domain or organization. They justify this by stating that the organizational

and process discipline needed to adopt a common CBSD practice is difficult. Cheesman further says that consequently their process has placed emphasis on specification because without the ability to describe the specifications of components they won't ever be reused. We can conclude that common standards for defining and describing components and for sharing them are required before large-scale component reuse will occur.

Chessman and Daniels' process provides a methodology for the modeling and specification of component based software systems with the UML. [3] [4] They describe techniques for using the UML integrally to conduct requirements analysis, build a business type model, and specify component interfaces and interactions. The product of the design and modeling process is complete component specifications in the UML that describe pre and post conditions (using OCL), offered and used interfaces, and component interactions. Their methodology does not provide a method for specification of performance and non-functional requirements, an aspect that is lacking by most component specification methods. [7] [10] [5]

3.2. Kang's Component Based Software Development Process

Kang proposes a two-tier process for component based software development. [14] The Domain Engineering tier focuses on domain analysis to discover commonalities among requirements that can result in the development of common architectures, and reusable components. Kang's Domain Engineering process includes: Domain Analysis, Reference Architecture Development, and Reuasable Code Component Development. Application Engineering is the other tier of Kang's model. The Application Engineering tier is similar to a traditional software lifecycle but integrates reuse of existing architectures and components. Kang's Application Engineering process includes: user requirements analysis, application architecture selection, application design, and application software integration phases.

Domain engineering, the process of discovering commonalities of systems within application domains involves the identification and development of a family of applications based on similar architectures and components. Once a family of applications exists developing new applications within the family can benefit significantly through the reuse of the associated preexisting architecture and component libraries. A significant problem with domain engineering is the large cost involved in developing reusable components. Kang states that the cost could be five times as much to develop reusable software components. The additional cost can be attributed to the additional development time required to discover common domain requirements and provide better interfaces and documentation.

3.3. Componentware Process

Componentware is a software process that provides a method for describing, structuring and developing component-based systems. [2] Componentware resembles a traditional software development process except that the design phase separates business-oriented design from technical design. Business design concerns the business-relevant aspects of the application including interactions and algorithms with respect to the business aspects, or "work" of the system. The technical design addresses the specification of technical aspects such as database components, persistence, distribution, and communication. Componentware includes an analysis

phase (Interaction, Responsibility, Risk Analysis, and Market Study), a split business and technical design phase (Architectural design, component design, evaluation, and search), a specification phase (Architecture specification, Component Specification, Component Test, and Component Assignment) and an implementation phase (Code and System Test).

By separating business logic from technology, the Componentware process encourages reuse since components will not couple the work of the system with technology specific details. By encouraging the development of decoupled components, components will be independent of each other. Kang points out that components should only implement a unique singular function. To improve maintainability the implementation of a single functional requirement should be implemented centrally and not across a whole set of components.

3.4. PECOS Process

The PECOS Software Process seeks to enable component-based development for embedded systems. [21] Many aspects of the PECOS process can also be used as a general component-based software development process. Similar to Kang's process, PECOS splits software development into two distinctive processes: a component development process, and an application development process. The component development process includes: requirements elicitation and analysis, interface design, component implementation, testing, profiling which is the benchmarking of performance and non-functional requirements, and documentation/release. The application development process is referred to as application composition, which entails the assembly of components to form an application. The application development process includes: requirements elicitation, architecture specification, component identification, query/searching for components, selection of components, composition of components, application testing, and application documentation and deployment.

It is interesting to note that the PECOS process places documentation in the final phase of each process. For the component development process it is arguable that the documentation of the interfaces and non-functional requirements could be done integrally during earlier phases of development. The precise results of component profiling may not be available but much of the interface specification information should be available. Documentation should also be continually updated as changes to requirements occur.

3.5. Catalysis Process

The Catalysis Process does not provide a strictly specified set of steps for CBSD. [6] Instead Catalysis acknowledges that no single process can fit every project, so rather than define a process, Catalysis provides a number of process patterns, good practices which should be considered in the formulation of the software process that an organization will adopt. D' Souza identifies process patterns many of which are derived from real world experience in CBSD. The process patterns of Catalysis are categorized in figure 1. Note that two patterns from [7] have been classified as both requirements analysis and design/specification process patterns.

Development Phase	Number of Patterns
Requirements Analysis	10
Design/Specification Patterns	15
Testing	1
Software Process Development/Improvement	2
Organization Process Development/Improvement	1
Modeling	6
Implementation	1
Component Design/Specification	8

Figure 1 - Categorization of D'Souza's Process Patterns

D' Souza identifies that component based development should be separated into three areas instead of the two suggested by Kang and the PECOS process. The three areas sited are: architecture definition: the design of common interconnection standards and frameworks, component development: specification and design of components with enhancement through evolution, and product assembly: the rapid development of end user products from assembled components. Catalysis stresses the importance of having a common architecture from which families of products can be developed.

Catalysis suggests that a CBSD process be an evolutionary life cycle with frequent deliverables and milestones. Rapid application development (RAD) should be used to deliver as much as possible early in development to acquire early feedback. A CBSD process should be nonlinear, iterative and parallel. Catalysis suggests that many activities should occur concurrently throughout software development including but not limited to testing, quality assurance, and documentation. Farias notes that Catalysis is a flexible, scalable process supporting component-based development. However it is somewhat limited at specifically addressing the challenges of CBSD because of its broad scope and flexible nature. [8]

In [6] an example for a CBSD process for a typical business system is presented. The suggested steps are: Requirements (Understand problem, system context, architecture, and non-functional requirements), System Specification (Describe external behavior of target system using domain model), Architectural Design (Partition technical and application architecture components and their connectors to meet design goals), and Component Internal Design (Design interfaces and classes for each component; build and test). A complete process would also include steps for implementation and testing.

3.6. Herzum and Sims' Component Development Process

In [11] Herzum and Sims identify three basic processes important for component based software development: a rapid component development process, a system architecture and assembly process, and a architecture and assembly process for creating a "federation", or library of components that can be shared by a family of applications. Organizations begin by developing components, move into developing common system architectures supporting component use, and finally build "federations" of system level components. Herzum like others has recognized the need to separate the design and development of components from the application assembly

process. A significant aspect of Herzum's process is the presentation of "golden" CBSD process characteristics. These characteristics are summarized in figure 2.

Ten Golden Component Process Characterisitics
Component Centric Approach to Development
Architecture Centric Approach to Development
Components should be autonomous
Approach to Collaborations
Iterative evolutionary process
Concurrent activities throughout process
Continuous Integration
Support risk management-driven development
Focus on reuse
Focus on development of quality components

Figure 2 - Ten Golden Component Based Development Process Characteristics

Herzum and Sims present an example of a rapid component development process. This process focuses on the development of the business-level components to implement the business logic and work of a system. The process describes how to go about gathering system requirements, build models, design business components from the models, and then implement the system. The following process steps are repeated in an iterative manner: Requirements Analysis (Feature Lists, Use Cases), System Analysis (Modeling), Design (External Specification, Dependency Specification, Internal Specification), Implementation, and Validation and Verification (Reviews, Testing). The process does not concern itself with user interfaces or non-work related aspects of the system. Using existing off-the-shelf components (COTS), or components from an internal company component archive to maximize reuse, is not addressed by the process.

3.7. COTS-Based Development Process

Kotonya, Onyino, Hutchinson, Sawyer, and Canal present a COTS-Based Development Process in [16]. They present a four-phase process based on these activities: negotiation, planning, development and verification. Their process is unique in that the traditional process activities of requirements gathering, design, and implementation appear under the larger phase known as the development phase. Testing is the separated from development into its own phase known as verification. Interesting is that all phases proceed in tandem so that while development is underway, continuous negotiation, planning, and verification are done as well. The COTS-Based development process has as its centric goal the use of existing components for application development. The traditional process activities of requirements gathering, design and composition are biased towards using available COTS rather than developing new components.

To optimize the use of existing components the COTS-Based development process suggests that the capabilities of available components could drive the requirements specification. Various reasons including: budgetary restrictions, schedule limitations, and project quality requirements may force specific components to be used, thus constraining the requirements and system design.

Scoping and ranking of requirements may be required to better understand possible tradeoffs and make compromises in order to maximize the use of preexisting components versus developing new components from scratch. Requirements negotiation with customers and end users may be necessary to scope the requirements to best reuse existing components. Through negotiation customers may perceive weaknesses in component based software design and even the software developers themselves if they sense that project requirements and features are being eliminated or changed in order to accommodate the restrictions imposed on the developers by the COTS.

System design using the COTS-Based development process involves partitioning system requirements into logical components taking into account business concerns, architectural concerns and the availability of pre-existing components. Next follows the process of component selection. Components are evaluated and the component that is most "fit for use" is selected. The most fit component best operates in the context in which the component will be used.

Following design, system composition proceeds by integrating together components. Components can be used directly off the shelf, or adapted for use. Components are glued together using script languages or with code from high-level languages. Verification and testing is done concurrently throughout the process. Thorough testing of existing components is suggested prior to and after they are integrated into the system. Regression testing should be conducted when components are upgraded or the system is enhanced by additional new components.

3.8. The Rational Unified Process

The Rational Unified Process (RUP) is a generalized software process, not specifically for component based software development in the sense that we have discussed thus far. The RUP states explicitly that it is "component-based". In the case the process itself is component-based, but RUP is not only for the development of component-based software. The RUP is a generic process framework that can be specialized for many specific classes of software systems, including component-based systems. [12] The RUP has several process attributes common to many of the component based processes we've presented. The RUP is architecture-centric. Several CBSD processes presented here have noted the importance of having standard architectures and common frameworks for promoting component reuse. The RUP is iterative and incremental. Iterative evolutionary based lifecycles are generally recognized as the preferred way to produce component-based software as the majority of the processes reviewed here are iterative. RUP supports modeling and specification of component-based systems with the UML. RUP also provides a workflow of activities for CBSD. [17]

The RUP presents traditional software life cycle activities organized into four phases similar to Kotonya's COTS process discussed earlier. The phases include: Inception, Elaboration, Construction, and Transition. Each phase includes a complete iteration of the five core workflow activities: Requirements, Analysis, Design, Implementation and Testing. Each phase should allocate different resources to each of the activities as shown in figure 3.

Activities:	Inception	Elaboration	Construction	Transition
Requirements	**Maximum**	**Maximum**	*Minimum*	*Minimum*
Analysis	Moderate	**Maximum**	*Minimum*	*Minimum*
Design	Moderate	**Maximum**	Moderate	Moderate
Implementation	*Minimum*	*Minimum*	**Maximum**	Moderate
Testing	*Minimum*	*Minimum*	**Maximum**	Moderate

Figure 3 - RUP's Activity Resource Allocation per Phase

RUP's implementation activities include the development of an implementation model, components, and their interfaces. From the implementation model, model elements are combined into components for implementation. Components encapsulate a series of design classes and provide an interface to their functionality. Components can be more than just traditional source code components in RUP. Components can also be: an executable program, file, database table, or document. Components that we are familiar in RUP are considered to be static/dynamic source code libraries. Components encapsulate the classes they are composed of forming black box like entities in the architecture. Dependencies can exist between the components and can be shown in the design model. The Rational Unified Process benefits from process automation by the existence of many commercially available tools. One of the reasons why the automation of RUP has been so successful is that the process has been standardized to simplify the creation of tools.

According to Farias [8], the RUP is not really a component-based process. As we mentioned previously the process itself is component-based, but it is not customized for the development of component based software. Farias states that components are an after thought in the RUP. The RUP exclusively relies on UML for modeling purposes. The originial RUP relies on the modeling constructs for component specification in the UML. It is generally agreed that the component specification capabilities in the UML prior to version 2.0 are limited. [15] With future upgrades of the UML, the RUP will likely become a more amenable component based software development process. [18]

4. Discussion

By analyzing key attributes the table in figure 4 can be presented showing common attributes among the CBSD processes presented in this paper.

Several processes (Kang, PECOS, Herzum) divide the component and application development, into separate life cycles. By dividing the development of components and applications, a software process can recognize that components are not merely the product of application development. By applying domain engineering organizations can identify cross cutting requirements in families of applications that can help to identify cases of component reuse.

Many component development processes (Cheesman and Daniels, Kang, PECOS, Catalysis, Herzum, COTS, Rational Unified Process) stress the need for iterative evolutionary lifecycles. Short development life cycles emphasizing frequent deliverables allow for frequent user evaluation and feedback. Use of off-the-shelf components can reduce development time and allow more iterative release cycles. Frequent user feedback can help assure that the correct

software is built to meet user needs to help reduce development time incurred from misunderstandings of software requirements.

Process	Component Based Software Development ONLY	Separate Component and Application Lifecycles	Defines Modeling Methods	Iterative / Evolutionary Life Cycle	Parallel / Concurrent Activities
Cheesman & Daniels	√		√	√	
Kang	√	√		√	
Componentware	√			***Weak***	√
PECOS	√	√		√	***Weak***
Catalysis			√	√	√
Herzum	√	√		√	
COTS	√			√	√
Rational Unified Process			√	√	√

Figure 4 - Comparison of Component Based Software Processes

Software processes should be malleable and adaptable to the individual attributes of the software project and organization. Several CBSD processes (Componentware, Catalysis, COTS, Rational Unified Process) recognize the concurrent nature of CBSD activities. Process activities are not strictly sequential in nature, but can occur concurrently throughout iterative lifecycles. Testing, documentation, requirements identification, and specification activities can occur dynamically through an iterative process.

5. Conclusions

Several challenges remain in component based software development and further enhancements to component-based software processes may help bring to fruition the many promises of using component-based development practices. Component-based design and development processes used by organizations will most likely be influenced by specific business needs and requirements of end users. Once a development process is adopted, it should be customized and improved after successive process iterations. Component based development requires the use of either off-the-shelf commercial components, or internally built custom components. By applying domain engineering, organizations begin to identify common software requirements between projects to allow for the development of cross-project reusable components.

Traditional development activities within the software process should be refocused to best enable and enhance software development using components. Processes such as the COTS process take an aggressive stance by going as far as influencing software requirements based on available components. Additional process evolution may help CBSD live up to its expectations. Current CBSD processes have already identified activities such as: requirements partitioning (for easy

mapping to components), repository searching, evaluation of components (ranking), component assembly, and component testing. The eventual goal should be to balance the cost benefits of CBSD so that productivity gains are realized and project delivery times shrink. Merely proposing software development processes will not improve the state of the art. Component based processes must be used in real projects in conjunction with case studies and empirical investigations. This research can help us to understand how processes and their related development activities need to change in order to help CBSD become the leading software development paradigm in the future.

6. Acknowledgements

I would like to recognize and acknowledge the guidance and support from Dr. Robert France, Dr. Suditpo Ghosh and Dae-Kyoo Kim.

7. References

[1] Atkinson, C., Bayer, J., Laitenberger, O., Zettel, J., Component-Based Software Engineering: The KobrA Approach, in Proceedings of the 2000 International Workshop on Component Based Software Engineering held in conjunction with ICSE2000, Limerick, Ireland, 2000.

[2] Bergner, K., Rausch, A., Sihling, M., Vilbig, A., Componentware – Methodology and Process, in Proceedings of 1999 International Workshop on Component Based Software Engineering held in conjunction with ICSE99, Los Angeles, CA, USA, pp. 194-203, 1999.

[3] Cheesman, J., Daniels, J., UML Components: A Simple Process for Specifying Component-Based Software, Addison Wesley, 2001.

[4] Clemente, P. J., Sánchez, F., Pérez, M. A. Modelling with UML Component-based and Aspect Oriented Programming Systems, in Proceedings of the Seventh International Workshop on Component-Oriented Programming, Malga, Spain, June 2002.

[5] Crnkovic, I., Hnich, B., Jonsson, T., Kiziltan, Z., Specification, Implementation, and Deployment of COMPONENTS. Communications of the ACM, 45(10):pp. 35-40, 2002.

[6] D. D'Souza and A.C. Wills, Catalysis: Objects, Frameworks, and Components in UML, Addison-Wesley, 1998.

[7] Edwards, S., Toward Reflective Metadata Wrappers for Formally Specified Software Components, in Proceedings of the Specification and Verification of Component-Based Systems, OOPSLA Workshop, October 2001.

[8] Farias, C.R.G., Pires, L. F., Sinderen, M., Quartel, D., A Combined Component-Based Approach for the Design of Distributed Software Systems, in Proceedings of the Eigth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'01), Bologna, Italy, 2001.

[9] Ghosh, Sudipto, Improving Current Component Development Techniques for Successful Component-Based Software Development, ICSR7 2002 Workshop on Component-Based Software Development Processes, Austin, Texas, 2002.

[10] Han, Jun, An Approach to Software Component Specification, 1999 International Workshop on Component Based Software Engineering held in conjunction with ICSE99, Los Angeles, CA, USA, 1999.

[11] Herzum, P., Sims, O., Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise, John Wiley & Sons, Inc, 2000.

[12] Jacobson, I., Booch, G., Rumbaugh, J., The Unified Software Development Process, Addison Wesley, 1999.

[13] Jarzabek, S., Knauver, P., Synergy between Component-Based and Generative Approaches, Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, Toulouse, France, pp. 429-445, 1999.

[14] Kang, K., Issues in Component-Based Software Engineering, in Proceedings of 1999 International Workshop on Component Based Software Engineering held in conjunction with ICSE99, Los Angeles, CA, USA, pp. 209-214, 1999.

[15] Kobryn, C., Modeling Components and Frameworks with UML. Communications of the ACM, 43(10): pp. 31-38, 2000.

[16] Kotonya, G., Onyino, W., Hutchinson, J., Sawyer, P., Canal, J., COTS Component-Based System Development: Processes and Problems, appears in Business Component-Based Software Engineering, Kluwer Academic Publishers, pp. 228-245, 2003.

[17] Kruchten, P., Modeling Component Systems with the Unified Modeling Lanugage, in Proeceedings of the 1998 International Workshop on Component Based Software Engineering held in conjunction with ICSE 1998, Kyoto, Japan, 1998.

[18] UML Revision Task Force, OMG Unified Modeling Language Specification, v 2.0, document ad/02-??-??. Object Management Group, Fall 2002.

[19] Wallnau, K., Hissam, S., Seacord, R., Half Day Tutorial in Methods of Componen-Based Software Engineering: Essentials Concepts and Classroom Experience, Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, Vienna, Austria, pp. 314-315, 2001.

[20] Wanapan, N., and Kerethom S., A Method for Developing Component-Based Applications: Role Based Coordination Component Model (RBCCM) Approach, ICSR7 2002 Workshop on Component-Based Software Development Processes, Austin, Texas, 2002.

[21] Winter, M., Zeidler, C., Stich, C., The PECOS Software Process, ICSR7 2002 Workshop on Component-Based Software Development Processes, Austin, Texas, 2002.