The Missing Approach for Component Specification

Benneth Christiansson and Marie-Therese Christiansson
Information Technology Department
Karlstad University
Sweden

Benneth.Christiansson@kau.se
Marie-Therese.Christiansson@kau.se

Abstract:
For component-based software development to be successful in organizations, the software developers must give close attention to the design of components as independent abstractions with well-specified behaviors. Without well-specified behaviors the possibility to distribute and acquire software components will be limited. We have studied 20 approaches to software component specification focusing on problem area, formality, usage of specification and identification of requirements. We show that the main focus in the software engineering community is towards the 'datalogical' side of software component specification. The major part of research concerning software component specifications is aimed at finding solutions regarding assembly and composition. The approaches are also oftentimes formal in their modelling approach. We argue for the need of an informal approach to capture requirements and enable acquisition of existing component. This approach needs to be based on business practice and people who are best suited to model requirements.

Introduction

Software systems form an essential part of most organizations business infrastructure, and becomes increasingly complex. In today's global market, these organizations have to continuously adjust and improve their business practices to maintain a competitive edge. This conveys that the demands and requirements on the organizations software systems change at the same rate. This is a big challenge for the software development community, and has been a big issue in the software engineering field for at least two decades. The challenge is to increase the productivity of software system development and to augment the flexibility of software systems to react to business process changes. One approach to achieve this is taking clues from traditional production techniques. Software systems should be constructed from prefabricated, easily identifiable software components (Szyperski, 2002, Christiansson, 2001) that can be widely used. To develop software systems with a component-based approach is one of the newer trends within the software development community. However for component-based software development to be successful in organizations, the software developers must give close attention to the design of components as independent abstractions with well-specified behaviors. Without well-specified behaviors the possibility to distribute and acquire software components will be limited.

Langefors (1995, p. 142) describes the development of software systems as finding the solutions to: *"Two fundamental problems with information systems were pinpointed at the outset: (1) The "infological" problem of how to define the information to be made*

available to the information system user, and how to design data that may represent the information to the user; and (2) The "datalogical" problem of how to organize the set of data and the hardware so as to implement the information system.". We take our point of departure in this description. We believe that emphasis during software development needs to be on both these problem areas also regarding component-based software development or even more so regarding component-based development. We believe this is accurate due to two facts firstly software component development is about assembly not about construction. This means that we do not have to focus on how the actual development is done; the software component is an existing artifact. Secondly software component development is about acquisition, we need to be able to identify which components we need when assembling systems; this conveys the need for a specification of the components behavior (Szyperski, 2002, Christiansson, 2000, Heineman & Council, 2001). This acquisition we believe needs to be based on an 'infological' specification as well as a "datalogical" one, using Langefors (1995) terms. In this paper we describe the need for a new approach to specify software components; we also show that the major part of research in this area is aimed at finding the solution to the 'datalogical' problem. In the few approaches focusing on the 'infological' problem the strategy is based on creating formal specifications. We argue for the need of an informal approach for addressing the 'infological' problem of software development.

The software component

The term software component isn't easy to define, it does not have a clear-cut definition in the software development community, but the meaning fluctuates. Atkinson et. al (2002, p. 67) expresses this as: *"Although components have been in vogue for some time, there is still no general consensus about precisely what constitutes a component and exactly how they should be put together to build useful systems."* This paper does not focus on the issue of defining the term software component. Instead we support the definition that Christiansson (2001, pp. 235-236) makes:

"A software component:

- *is independent and reusable;*
- *provides a defined functionality using a specific interface;*
- *can affect/be affected by other software components;*
- *should have a specification (in which the software component is described on a high level of abstraction);*
- *can have multiple implementations, meaning that the same component can be implemented in several programming languages; and*
- *can have several executable (binary) forms, i.e. the same component can be executed in different operating systems."*

The fact that a component is independent and reusable shows that a component can be used without other components present, the services provided by the component should be accessible without any external help except from the software glue and necessary run-time environment. A component can affect and be affected by other software components. This means that two components can 'work together' and 'as a whole'

create a greater service than used separately. In figure 1 we illustrate a software component with a context.

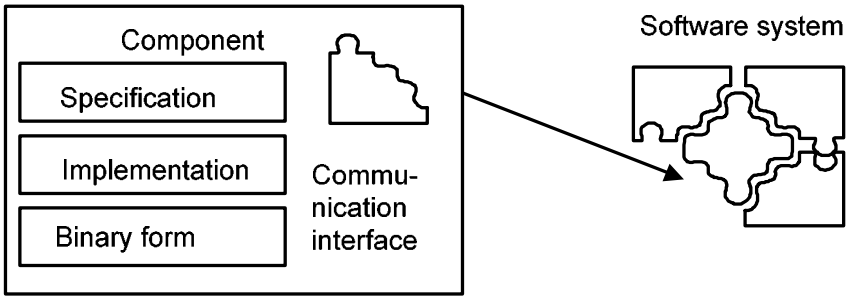

Figure 1. A software component with a context (Christiansson, 2001, p. 236)

The need for a documented specification for a software component is obvious if one consider the process of acquiring a component. How can one find a software component if one doesn't have something to look for? This is a factor that can decrease the gap between the 'infological' and 'datalogical' issues when developing a software system. If there are documented specifications in the 'infological' area, these can be described in such ways that they are useful when dealing with the capturing of requirements and acquisition of components to be used in software systems. The component as such is directly usable in the 'datalogical' sense as an implementation and/or binary form (Christiansson, 2000).

Our approach to software component specification

We have chosen to describe our research effort as: *"The meeting between business processes and software components – in a well founded specification."* Our approach is focused mainly towards capturing requirements on components, and creating a specification that can be used for acquisition, approaches towards assembly do already exist. Our approach for specifying software components is based on the integration of results from two research fields; 1) Component Based Software Engineering and 2) Business Process Modelling. As Langefors (1973, p. 53) so elegantly put it: *"Experience shows that different groups in organizations tend to neglect the importance or the difficulty of the other peoples 'field'"*. Our approach integrates business process models, the capturing of software requirements in an informal manner and software component specifications into one type of document. We believe it is possible to short-circuit the traditional software development process when using software components. This is due to two facts 1) it is not a pragmatically based assumption that the approach to formally break down requirements into more and more detailed software specifications is successful in the real world; and 2) software components are already constructed, so the problem of software development is solved, however we still have the problem of acquisition and assembly of the existing components. Acquisition and assembly is based on the knowledge of what is needed to acquire and assemble and also on what is possible to acquire. Vigder et al. (1996, p. 13) claims that *"… in order to realize the benefits of COTS software a procurement process must be in place that defines requirements according to what is available in the marketplace, and that is flexible enough to accept*

COTS solutions when they are proposed.”. What is needed is an approach to specify requirements on software components to identify actual needs. To be able to capture requirements based on the desired business processes expressed by people who run and perform business practise. We believe this should be done close to the business practise to cope with the problem of missing and/or inaccurate requirements as well as constant business change, se figure 2. Beck (2000, p. 3) illustrates this as *“Business misunderstood – the software is put into production, but it doesn’t solve the business problem that was originally posed. Business changes – the software is put into production, but the business problem it was designed to solve was replaced six months ago by another, more pressing, business problem.”*

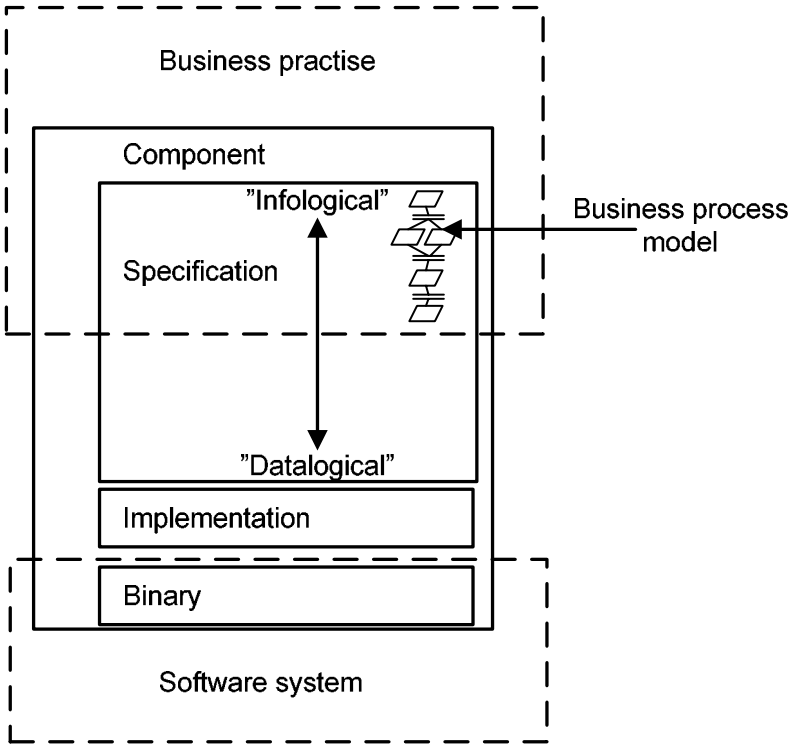

Figure 2. Our approach to well founded component specifications.

Our approach is based on the following assumptions and demarcations:

1) Our approach is applicable after the performance of a business change analysis that resulted in the need for software systems development.

2) Our approach is based on a component-based strategy for software systems development.

3) A useful (pragmatically) business model is small and flexible. To appeal to the software engineering community as well as the potential procurer of software systems the models that capture requirements should be easy to understand and

result in the absolute minimal extent required to capture requirements. We suggest an approach similar to what Beck (2000) call extreme programming. He states *"Business often doesn't like Development. Relations between people who need systems and the people who build systems is so strained, they often resemble the relations between centuries-old enemies"* (Beck, 2000, pp. 86-87). This indicates the need of a minimal, understandable and useful method for capturing, documenting and communicating requirements. We call this an extreme modelling strategy (Christiansson & Christiansson, 2003).

4) We believe that people are best suited to model requirements and therefore the models should be adapted to human needs. We need to delimit the information amount to its bare essentials and describe this information on such a high level of abstraction that for instance atomization isn't needed. The amount of information should be easily overviewed, browsed and understood by people, both systems engineers and clients, without the rendering of 'information overload'. We regard formal models as important when creating models for software development, but not when describing requirements for acquisition of existing software components. We argue for a human-based capturing of requirements and a manually performed acquisition.

5) We believe a business process based approach for capturing software requirements in the form of component specifications is useful.

6) A usable strategy for capturing requirements is based on the language used in the business practise.

7) We regard component-based systems development as being closer to the field of ERP-systems acquisition than traditional tailoring of Software systems. Szyperski (2002) presents component-based development as a middle path somewhere between traditional 'tailoring' of software and the acquisition of ERP-packages. The case isn't development any more but acquisition and assembly of already existing software in the shape of components.

8) Our approach is only focused towards capturing requirements and using them for acquisition of COTS (Components Off The Shelf) software components. The captured requirements can of course be used as a basis for traditional software development but this is not our intent with the approach.

9) We use Langefors (1995) idea about two major problem areas 'infological' and 'datalogical' and consequently with our demarcations choose to focus on the 'infological' problem area. Assuming the actual components already exist indicates that capturing 'infological' requirements is sufficient to acquire components.

10) By integration we mean the merging of business process models and software component specifications. We also mean that these models are linked to other

necessary documents both regarding business process models, the multifaceted nature of business processes calls for pluralistic and multidisciplinary modelling approaches (Melão & Pidd, 2000), as well as more detailed software specifications focusing on for instance assembly of components.

Positioning of existing approaches

We have performed an extensive literature study and studied 20 different approaches towards software component specification. We realize that we haven't studied 'all' existing approaches but still believe we have found a sufficient amount to be able to state that our approach is a novel one. We have chosen four dimensions to focus on in this study. These dimensions are based on our description of a useful approach to component specification and the capturing of software requirements, se preceding section.

The first dimension is what *problem area*, according to Langefors (1995) division of the software development activities as the solutions of the 'infological' and 'datalogical' problems, the approach is focused on.

The next dimension is the *degree of formality* in the required documentation. Here we see a scope from informal natural language descriptions using the business practitioner's language to very formal definitions of each of the constructed software's attributes and behaviors. In this perspective atomization is often addressed. To automate the allocation of components we need to have a very high level of formalism regarding the specifications we use. This leads to a lot of information specified in graphical or textual models. Another approach we believe could be to delimit the information amount to its bare essentials and describe this information on such a high level of abstraction that atomization isn't needed.

The third dimension is the *expected usage* of the specification, what the specification is aimed towards. We have noticed that the larger amounts of approaches are aimed towards assembling or composition of component-based software systems. These approaches indicate that the components already exist within reach of the software engineers. But we argue for specification approaches that support the actual acquisition of components regarding identification and evaluation as well as assembling existing ones. So we identify the third dimension as expected usage mainly acquisition and/or assembly.

The fourth dimension is regarding *identification of requirements*. By this we mean any kind of methodological support for identifying the actual requirements. We have noticed that the specification approaches described in literature is based on the notion that the requirements already are defined and described somehow; we call this a descriptive focus. By descriptive we mean that the specification approach only supports the description of known requirements. We believe that another way of doing this is enabling and supporting both the actual description of requirements but also in identifying them. We call this an explorative approach, where explorative indicates the support for exploration and identification of requirements as well as describing them.

The following section gives a brief summation of each identified approach and a positioning within our described dimensions, we end this section with a table containing the whole classification, see table 1.

1. Business Objects.
Sims (1994) presented his idea regarding business objects, where a business object is a component supporting an activity in the business practice. Business objects are, according to object-orientation, large objects that supports one or more clear business functions. This approach is focused towards 'datalogical' issues and focuses mainly on assembly. It is formal in the way business objects are documented and descriptive regarding capturing of requirements.

2. Melding Structured Abstracts and the World Wide Web for Retrieval of Reusable Components.
The intention of this approach is to according to Poulin & Werkman (1995): *"enable a way to quickly assess the important aspects of a piece of software so programmers can decide whether or not to reuse it."* (p. 1). The approach has a structured informal approach of representing a software component specification using *"…an orderly, concise, natural-language narrative."* (p. 6). Furthermore we classify the approach as descriptive in the sense that all requirements for acquisition is known beforehand, it is directed towards the 'datalogical' problem area .

3. Retrieving software Components That Minimize Adaptation Effort.
Jilani et. Al (1997) describes an approach for software component retrieval from a software repository. *"Given a software library whose components are represented by formal specifications, we distinguish between two types of retrieval procedures: exact retrieval, whereby, given a query K, we identify all (and only) the library components that are correct with respect to K; approximate retrieval which is invoked in case exact retrieval fails, and which (ideally) identifies the library components that minimize the required adaptation effort…"* (p. 1). We distinguish this approach as a formal, descriptive and 'datalogical' approach with focus on acquisition and assembly of existing components.

4. The Magma approach to CBSE.
Hallsteinsen & Skylstad (1999) describes an approach for component-based software engineering and is based on experiences from the Norwegian software development industry. The approach is based on practice and is being distributed through a software engineering Handbook. The approach uses Object-Oriented modeling with UML. This means that the approach has a main focus towards 'datalogical' issues and uses a formal strategy for specifications. The approach has an explorative focus regarding identifying software requirements. The main focus is towards development and assembly of applications and components.

5. Characterizing a Software Component.
Yacoub et. Al (1999) focuses on component characterization they *"proposes a set of features to characterize a software component."* They define a component

characterization as existing on three levels 1) Informal description with "human-related issues" 2) Externals which means its interactions with other application artifacts and with the platform and 3) Internals which reflects its internal aspects. This is an approach that we classify as addressing both 'infological' and 'datalogical' issues. The approach identifies the need for informal descriptions and has an acquisition /assembly perspective. The strategy is descriptive and gives no hints on how to identify the needed parts of the specification.

6. The Catalysis Approach.
D'souza & Wills (1999) describes an approach to develop component-based software systems. The approach contains a specification strategy which the authors describe as *"A system requirement spec often reflects the business model closely"* (p. 17) This quotation indicates a informal and 'infological' approach to component specification but they also state that *"Gathering all the specs for the actions the system is required to take part in and the static models needed to draw snapshots for those specs, we compile a formal functional requirements model."* (p. 17). We classify this approach as mainly dealing with the 'datalogical' problem area, it has an explorative focus for capturing requirements. The approach is formal and focus is on development and assembly.

7. An Approach to Software Component Specification.
Han (1999) describes their approach as *"… component specification aims to provide a basis for development, management and use of components."* (p. 2). The approach is claimed to handle the syntactic issues concerning the exact interface of a component implementation and also to include the semantics of the interface elements, their relationships, the assumed user contexts and the quality attributes. In what way all of this is captured in the suggested specification is however not clear. We understand this approach as being a formal one dealing with assembling components on the 'datalogical' level. The approach has a descriptive focus dealing wit already known requirements.

8. Enhancing Component Reuse Using Search Techniques.
Zhang (2000) has narrowed the specification issue down to be applied in a specific type of component repository. This repository exists in a so-called metaCASE environment. Zhang (2000, p. 1) states the purpose of this approach as enabling acquisition *"To reuse and integrate a component in the metaCASE environment, users must be able to locate and understand them."* The approach is based on the use of a CASE tool where the actual components and their specification are stored. This indicates the specification strategy to be formal, with both syntactic and semantic rules for the specification schemes. The approach has a descriptive aim where all requirements are known and focuses on the 'datalogical' problem area.

9. Declarative Descriptions of Component Models as a Generic Support for Software Composition.
Presso (2000) describes a concept called component models which *"…define standards for describing components and mechanisms to build applications out of components. These standards are specified using natural language and embedded into composition tools…We propose the use of logic meta-programming to describe the components,*

describe the protocols for composition and the code that realizes them, specify an application built from connected components and generate the code for the application. " (p. 1). This quotation illustrates the 'datalogical' focus as well as a formal approach towards specifications. The approach also incorporates a descriptive outlook regarding requirements and aims towards assembly of component rather than acquisition.

10. Using UML Software Engineering With Objects and Components.
Stevens & Pooley (2000) describes an approach to software component specification using UML diagrams to model components. They state that "A component must be properly documented with specification…. Components by definition realize interfaces and have context dependencies; both aspects should obviously be documented." (p. 217). The approach is formal an focused towards solving the 'datalogical' problem area. The approach is descriptive regarding requirements capturing and focus on assembly of existing components.

11. A Grey-Box approach to Component Composition.
Bruin (2000) describes an approach based on trying to solve the problems with incomplete component contracts and undocumented dependencies between components. The approach is based on a *"scenario-based technique called Use-Case-Maps (UCM), which uses scenarios to describe how several components operate at a high abstraction level"* (p. 195). We classify this approach as formal and dealing with assembly of existing components. It is focusing towards 'datalogical' issues and is descriptive regarding requirements.

12. Business Modeling with UML.
Eriksson & Penker (2000) states that a specification is a model element, they use the UML notion of interface to document specifications. *"The interface is a specification of a collection of operations that can be implemented by one or more classes."* (p. 27). The approach is based on UML and uses the 'use case' diagrams to model the 'infological' issues. We classify this approach as descriptive regarding capturing of requirements. The approach handles informal issues in models, the approach does not focus on acquisition of existing components.

13. UML Components A Simple Process for Specifying Component-Based Software.
Cheesman & Daniels (2001, p. 24) defines a component specification as *"…defines what is to be built and what units will exist at runtime. The component specification defines the set of interfaces supported and any constraints on how they are to be implemented."*. The approach is based on using UML-notation and has a both 'infological' and 'datalogical' approach. The approach is descriptive in the sense that the authors delimit themselves from identifying requirements *"This isn't a book about requirements gathering"* (p. 67). The focus is on development of new components and to some extent on the assembly of components. The approach has a both formal and informal perspective and for instance argues the usage of 'system envisioning' using techniques such as storyboarding.

14. A Formal Approach to Software Component Specification.

According to Lau & Ornaghi (2001) the purpose of this approach is to allow formal reasoning about components. The reasoning is to be done regarding the components construction and composition as well as their correctness. They define a component specification as a specification of its interface. "The specification of a component is therefore the specification of its interface, which must consist of a precise definition of the component's operations and context dependencies, and nothing else." (p. 1). We classify this approach as a formal, descriptive approach towards the assembly of already known components.

15. A Goal-driven Appoach to Enterprise Component Identification and Specification.

This approach is described by Levi & Arsanjani (2002) and has a web service focus. They have developed a systematic method of component identification and boundary definition. The approach is based on the usage of UML notation. And the approach uses a straightforward decompositional strategy for breaking down the requirements according to the possibilities within UML. We classify this approach as a formal approach using UML with main focus on assembling software components with a 'datalogical' focus. They use an explorative approach regarding the identification of requirements but do not document them in any informal way.

16. Software Component Specification Using Role-Based Modeling Language.

Kim et. Al. (2002) have developed an approach for component specification with a Role-Based Modeling approach. A role model is defined as *"…a structure of roles, where a role defines properties that determine a family of UML model elements (e.g., class and generalization constructs."* (p. 2). Further they claim that a component specification is a package of 'role models' expressed using the Role Based Modeling Language. This is a formal approach to support assembly of software components with a 'datalogical' focus. The approach is based on a descriptive notion where the requirements are known beforehand.

17. Software Component Specification Using Design by Contract.

Liu & Cunningham (2002) presents an approach to software component specification with the intent to *"…give close attention to the design of components as independent abstractions with well-specified behaviors."* (p. 1). The focus is to understand precisely what a component does, based on the specification of the operations in its interfaces. We regard this approach as a formal, descriptive approach directed mainly towards assembling existing components.

18. Components retrieval systems.

Khayati & Giraudin (2002) presents an approach for retrieving software components already existing in a repository. This implies that focus is on acquisition and assembly of existing components. They state that *"The main problem encountered when reusing the components libraries is component retrieval i.e. finding in the library the components that can be used in the construction of a specific information system."* (p. 1). We classify this approach as a semi-formal strategy due to the fact that they on one hand states that *"using external information provided as human description of the components."* (p. 4).

This statement indicates an informal approach towards specification but they also state that the engineer should be able to *"…formally specify their needs."* (p. 5). They assume that the components already are specified i.e. the requirements are known. We believe the approach to be mainly focused on specifications in the 'infological' problem domain.

19. The KobrA Approach.

Atkinson et al (2002) describes a method for component-based software engineering based on a product-line perspective. Product-line indicates that all software components can be related to a family of software products within the organization. These families are described as frameworks. These frameworks are generic and reusable. They define a concept called Komponent (KobrA component), each Komponent in a framework is then described by a suite of UML diagrams as if it where an independent system in its own right. This approach is formal using UML and focuses on implementation of software. The focus is towards development of components and applications and the approach assumes requirements to be known beforehand.

20. Modelling with UML Component-based and Aspect Oriented Programming Systems.

Clemente et al (2002). The approach is based on using UML diagrams to specify important aspects of software components. According to the authors: *"Interfaces specification, component specification, components implementation, package and assembly and deployment"* (p. 2) are important aspects that needs modelling. A component specification is a specification of its interfaces through a certain concept in UML. We classify this approach as a formal approach with a focus on 'datalogical' issues. The strategy does not give any help of how to identify requirements and has an emphasis on assembly of existing components, implying acquisition is already done.

Table 1. The classification of existing software components specification approaches

	Problem-solving area		Degree of formality		Usage of specification		Support for capt. Req.	
Approach#	'infological'	'datalogical'	Informal	Formal	Acquisition	Assembly	Descriptive	Explorative
1		X		X		X	X	
2		X	X		X		X	
3		X		X		X	X	
4		X		X		X		X
5	X	X	X		X	X	X	
6		X		X		X		X
7		X		X		X	X	
8		X		X	X		X	
9		X		X		X	X	
10		X		X		X	X	
11		X		X		X	X	
12	X		X			X	X	
13	X	X	X	X		X	X	
14		X		X		X	X	
15		X		X		X		X

16		X		X		X	X	
17		X		X		X	X	
18	X			X	X	X	X	
19		X		X		X	X	
20		X		X		X	X	

Conclusions

We have in this paper focused on what we believe is a missing approach to specify software components. We believe that emphasis during software development needs to be on both solving the 'infological' and 'datalogical' problems (Langefors, 1995). We have shown through our study of 20 existing approaches to software component specifications that the main focus in the software engineering community is towards the 'datalogical' side. We claim that focus should be towards the 'infological' problems, we do not mean that the 'datalogical' problems are irrelevant, but component-based development gives the possibility to regard the software as already constructed. We believe this is accurate due to two facts 1) software component development is about assembly not about construction. This means that we do not have to focus on how the actual development is done; the software component is an existing artifact. 2) Software component development is about acquisition, we need to be able to identify which components we need when assembling systems; this conveys the need for a specification of the components behavior in the 'infological' sense. To create this 'infological' specification we have turned towards the business process modeling community and suggest an approach that integrates business process models, the capturing of software requirements and creation of software component specifications into one type of document.

We claim that people are best suited to model requirements and therefore the models should be adapted to human needs. We need to delimit the information amount to its bare essentials and describe this information on such a high level of abstraction that for instance atomization isn't needed. The amount of information should be easily browsed and understood by a human, both systems engineer and the client, without the rendering of 'information overload'. We believe our claim is even stronger, regarding component-based development vis-à-vis traditional development, due to the fact that we are not dealing with development of software but mere the acquisition and assembling of existing software. Formal models are important, but not when describing requirements for acquisition of software components. We argue for an informal human-based capturing of requirements and a manually performed acquisition, we have in our positioning of existing component specification approaches showed that emphasis is towards formal models. This is maybe due to the fact that the major part of the studied approaches is focused towards assembly of components on the 'datalogical' level. We believe that we need an approach focused towards acquisition; we need to acquire components before we can assemble them into software systems. We also show that the larger share of studied approaches have a descriptive view regarding the capturing of requirements. With descriptive we mean that they only show how to describe existing known requirements, they do not aid the process of exploring requirements. Our approach has emphasis on the capturing of requirements.

Referenses

Atkinson C., Bayer J., Bunse C. & Kamsties O. (2002) Component-based Product Line Engineering with UML Addison Wesley Longman, Inc., California, Menlo Park, USA

Beck K. (2000) Extreme Programming Explained. Addison Wesley Longman, Inc., California, Menlo Park, USA

Christiansson, B. (2000) Komponentbasera informationssystem - Vad säger teori och praktik?, Linköpings Universitet (in swedish)

Christiansson, B. (2001) ”Component-Based Systems development” In: Nilsson A.G. & Pettersson, J.S. (eds.).*On Methods for Systems Development in Professional Organisations*, Studenlitteratur, Lund.

Christiansson M-T. & Christiansson B. (2003) Extreme Modelling – Less is More. (submitted paper)

Clemente P. J., Sanchez F. & Perez M. A. (2002) Modelling with UML Component-based and Aspect Oriented Programming Systems. Proc. 7:th Int. Workshop on Component-Oriented Programming.

de Bruin H. (2000) A Grey-Box Approach to Component Composition. First int. symp, Generative and Component-Based Software Engineering, Springer-Verlag, Berlin

D’souza D. & Wills A. C. (1999) Objects, Components, and Frameworks with UML The Catalysis Approach. Addison Wesley Longman, Inc., California, Menlo Park, USA

Eriksson H-E. & Penker M. (2000) Business Modeling with UML, John Wiley & Sons, Inc., USA

Hallsteinsen S. & Skylstad G. (1999) The Magma approach to CBSE. Proc. Int. Workshop on Component-Based Software Engineering ICSE99 Los Angeles, CA, USA

Heineman G. T. & Councill W. T. (2001) Component-Based Software Engineering Putting the Pieces Together. Addison Wesley Longman, Inc., California, Menlo Park, USA

Khayati O. & Giraudin J-P. (2002) Components retrieval systems. Reuse in Object-Oriented Information Systems Design, OOIS workshop Montpellier 2002.

Kim D. K., Ghosh S. France R. B. & Song E. Software Component Specification Using Role-Based Modeling Language", 11th OOPSLA Workshop on Behavioral Semantics: Serving the Customer, November 4, 2002.
Langefors B. (1973) Theoretical Analysis of Information Systems, 4:th ed., Studentlitteratur, Lund.

Langefors B. (1995) Essays on Infology – Summing up and Planning for the Future. Studentlitteratur, Lund.

Lau K. K. & Ornaghi M. (2001) A Formal Approach to Software Component Specification. Proc. Specification and Verification of Component-Based Systems Workshop at OOPSLA 2001, Tampa Bay, Florida

Levi K. & Arsanjani A. (2002) A Goal-driven Approach to Enterprise Component Identification and Specification. Commuication of the ACM. October/Vol. 45, No. 10, pp. 45-52.

Liu Y. & Cunningham H. C. (2002) Software component specification using design by contract. Proceedings of the SoutEast Software Engineering Conference, Tennessee Valley Chapter, National Defense Industry Association, Huntsville, AL, April 2002

Melão, N. Pidd, M. (2000) A conceptual framework for understanding business processes and business modeling, *Information Systems Journal*, Vol. 10, pp.105-129.

Poulin J. S. & Werkman K. J. (1995) Melding Structured Abstracts and World Wide Web for Retrieval of Reusable Components. M. H. Samadzadeh M. H. & Zand M. K. (Eds.): Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'95), April 23-30, 1995, Seattle, WA, USA pp. 160-168

Presso M. J. (2000) Declarative Descriptions of Component Models as a Generic Support for Software Composition. Proc. 5:th Int. Workshop on Component-Oriented Programming, Blekinge Institute of Technology.

Sims O. (1994) Business Objects, Addison Wesley Longman Inc., California, Menlo Park, USA.

Stevens P. & Pooley R. (2000) Using UML Software Engineering with Objects and Components. Addison Wesley Longman, Inc., California, Menlo Park, USA

Szyperski C. (2002) Component Software Beyond Object-Oriented Programming. Addison Wesley Longman, Inc., California, Menlo Park, USA

Vigder M. R, Gentleman W. M. & Dean J. (1996). COTS Software Integration: State of the art. National Research Council of Canada., Toronto, Canada.

Yacoub S., Ammar H. & Mili A. (1999) Characterizing a Software Component. Proc. Int. Workshop on Component-Based Software Engineering ICSE99 Los Angeles, CA, USA

Zhang (2000) Enhancing Component Reuse Using Search Techniques. Proc. IRIS 23, Laboratorium for Interaction Technology, University of Trollhättan Uddevalla.