ISSUES FOR ASSESSING COMPONENT-BASED SYSTEMS

Alejandra Cechich[1] and Mario Piattini[2]

[1]Departmento de Informática y Estadística - Universidad Nacional del Comahue
Buenos Aires 1400, 8300 Neuquén, Argentina
E-mail: acechich@uncoma.edu.ar

[2]Escuela Superior de Informática - Universidad de Castilla-La Mancha
Paseo de la Universidad 4, Ciudad Real, España
E-mail: Mario.Piattini@uclm.es

ABSTRACT

The use of component-based software has become more and more important in state-of-the-art and state-of-the-practice of software and system development. Using COTS software promises faster time-to-market, which can yield substantial advantages over competitors with regards to earlier placement of a new product on a market. At the same time, component-based software introduces risks such as unknown quality properties of the components in use that can inject harmful side effects into the final product. This paper proposes a multidimensional classification scheme for assessing component-based systems. The classification scheme provides insight into what quality characteristics, managerial features, or assessment methods and techniques might be used for evaluating different component artefacts.

KEY WORDS: Component-based systems • Component evaluation • Component-based system evaluation • Component metrics

1. INTRODUCTION

One of the major steps towards achieving reusability of software lies in using techniques, which support the identification and creation of components that are abstract in the sense that they are relevant in applications other than those for which they were originally created.
Given the large amount of component assessment methods and component-based evaluation proposals available, there is some confusion as to which methods are suitable for component-based systems (CBS) or Commercial-off-the-shelf (COTS) components, or just traditional components (bespoke components). We present a three-dimensional classification space for "component artefacts", i.e. component elements, generally speaking, that we abstract as components, interactions, and compositions. The dimensions of the classification space are the *perspective*, the *artefact*, and the *software element*. The classification scheme we use provides insight into which existing assessing methods are useful for evaluating a component artefact along dimensions, even though not every dimension value applies on every component artefact. We find that a number of models exist for the *artefact* and the *perspective dimension*; however, while several assessing methods (e.g. [1][2][3][4][5][6]) have been proposed for the evaluation of COTS components, there is a need for a conceptual model that can facilitate the

evaluation of component-based applications along the *software element* dimension, such that the applications are more extensively evaluated.
On the other hand, there is a natural affinity between software architecture and software component technology. This affinity is expressed by the central role of components and connectors as abstractions, and by the correlation of architectural styles, as an abstraction of design constraints, and component models and frameworks. Several methods and techniques have used those abstractions as a way of evaluating component interactions and component compositions (e.g. [7][8][9][10]), i.e. interaction and composition values respectively in our *artefact* dimension.

We propose a conceptual model that facilitates the evaluation of component-based systems by providing a scheme, which splits component software into three constituting parts according to ISO/WD-Std 12119 [11]. Different approaches, such as those mentioned above, have been considered to build the schema. Classifying and grouping the relevant ideas into a schema achieves the following:

- *Helps transfer the potential of component-based system evaluation into reality*. Although still immature, the concept of component-based system evaluation is timely. The necessary methods and techniques are available to support and exploit it, and it offers potential gains in productivity and reliability among other quality aspects.
- *Brings together disparate perspectives on component evaluation*. Thinking about component evaluation incorporates a wide range of views, from those that reflect primarily business objectives to those that are almost entirely concerned with technical issues.
- *Begins to identify the key research questions*. CBS development implies specific problems (such as COTS selection, integration, maintenance, and security). These are just a few problems. To select a COTS product, developers must also know the effort required for overcoming these problems. For example, a sequence of questions has to be answered: What is the difference between the system's requirements and the COTS products? What integration strategies can be used by the developers to integrate COTS software products? What is the productivity of the developers for the applied integration strategy?
  The importance of discussing CBS evaluation shows up when considering that component products are developed to be generic, however, being integrated into a system, they are used in a specific context with certain dependencies.

The rest of this paper is organised as follows. First, we present a 3-dimensional classification space for component and CBS quality assessments. Then, we propose high-level features along one dimension of the space and we show its use. Conclusion is addressed at the end.

2. A CLASSIFICATION SPACE FOR CBS ASSESSMENTS

In this work, we consider a *component* as a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard [12]. A *component model* defines specific interaction and composition standards, meanwhile a *component model implementation* is the dedicated set of executable software elements required to support the execution of components that conform to the model. Finally, a *software component infrastructure* is a set of interacting software components designed to ensure that a software system or subsystem constructed using those components and interfaces will satisfy clearly defined performance specifications.
One underlying concept of a component is that it has clearly defined *interfaces*. A component support a provided interface if the component contains an implementation of all operations defined by that

interface. A component needs a required interface if the component requests an interaction defined in that interface and expects some other software element to support that interface. A component may have an explicit context dependency on the operating system, a software component, or some other software element. An *interaction standard* specifies the type of explicit context dependencies a component may have. In general, an *interaction* is defined as an action between two or more software elements.
A *composition* is the combination of two or more software components yielding new component behaviour at a different level of abstraction. The characteristics of the new component behaviour are determined by the components being combined and by the way they are combined.
Commercial-off-the-Shelf (COTS) components are the main building blocks of a composition process. They are components that can be used by understanding its interface and without knowing the details of how its internals are constructed.

These concepts are common to all CBS development processes and they are also what make a CBS different from a traditional application. Next we propose three dimensions along which component assessments and CBS assessments differ: the *perspective*, which considers organisations capable of deploying trusted components for use within a CBS or organisations capable of developing trusted components for distribution; the *artefact*, which considers COTS and traditional components, interactions, and compositions; and finally the *software element*, which considers components as build upon several parts such as its functional and non-functional specifications, its documentation, and its implementation, i.e. programs and data.

The three dimensions serve the purpose of providing insight into what quality characteristics or managerial features are relevant to assessment, and the role of different evaluation methods and techniques appropriated for assessing a CBS and/or individual components. Dimensions are shown in Figure 1.

2.1 Perspective dimension

This dimension looks at the perspective of people/organisations capable of deploying trusted components for use within a CBS (users/acquirers), people/organisations capable of developing trusted components for distribution (developers), and people/organisations concerned with managing a CBS development process as well as assessing quality features.
Values along this dimension include developers, users/acquirers, and managers/quality assurance staff. We now explain each of these values.
Users/acquirers focus on building business solutions by consuming components. The challenge for the solution builder is to assemble a large system from components. The process of finding the right components is important under this dimension. This process can have one of two outcomes: either an ideal (or similar) component is found and used or no relevant component exists. This means that the process is not just about areas for built components, but also a decision-making process. A process on configuration management is also important for users/acquirers.

The second value along this dimension is *developers*. They are responsible for building completed components, usually to predefined specifications. The component specification may come from the component producer's desire to provide a generic component or the solution builder's specific needs. The specification may also derive from an industry standard for component interactions. The component developer starts with this specification and iteratively develops a design that satisfies the specification. A component implementation occurs when you decide on the language to use to develop the component. Because a component specification can be implemented using more than one language, multiple versions of an implementation may occur during the course of the life cycle. The developer will need to

publish accurate information about a component, such as its available interfaces and its services. Component testing is an important activity for developers.

The third value along this dimension is *managers/ software quality staff*. A software development project manager must understand and have sufficient experience to conduct CBS development activities, such as eliciting and modelling business rules, selecting COTS components, building components, developing test cases, estimating selection effort, estimating integration effort, etc. Additionally, managers and quality assurance staff deal with component quality aspects such as certification, measurement, configuration management, etc. Component cost models, component quality standards, and measurements are important issues for managers/software quality staff.

2.2 Artefact and Software Element dimensions

The *artefact* dimension looks at the CBS artefact to be evaluated, i.e. COTS and bespoke components, interactions, and compositions.
We have previously stated that for the scope of our work, a *component* is a piece of software that conforms to a component model and can be independently deployed and composed without modification according to a composition standard; an *interaction* is defined as an action between two or more software elements; and a *composition* is the combination of two or more software components yielding new component behaviour at a different level of abstraction.

The *software element* dimension looks at the software element to be evaluated, i.e. its specification, its documentation, and the programs and data used for implementation. The values of this dimension are adapted from the ISO/WD 12119 - V4 [11] standard for requirements quality of COTS components and testing.
The first value – *specification* – refers to a document containing any combination of recommendations, requirements or regulations to be met by a component, interaction, or CBS. It includes statements on functionality as well as on reliability, usability, efficiency, and other quality properties. We should note that those quality properties depend on a particular product and a given context.
The second value – *documentation* – refers to the complete set of documents, available in printed or non-printed form, which is provided for the application of the product and also is an integral part of the product. It includes user documentation that shall be complete, correct and consistent. Understandability and ease of overview are also quality properties required to user's documentation.
Finally, the third value – *programs and data* – refers to all implementation that corresponds to all the statements in the specification and the user's documentation. In particular, programs and data shall comply with all the requirements in any requirements document referenced by the product description. Hence, quality properties such as functionality, reliability, usability, efficiency, maintainability, and portability can also be assessed when considering programs and data as elements to be evaluated.

Using the three proposed dimensions is not straightforward, i.e. some combinations of values are senseless. For example, when considering COTS components, programs and data are not available for evaluation from a developer perspective, because only required and provided interfaces are known (along with some other documentation excluding source code). Therefore, the classification scheme needs to provide insight into what approaches could be used in the quality-assessing phase as well as what features should be focused on.

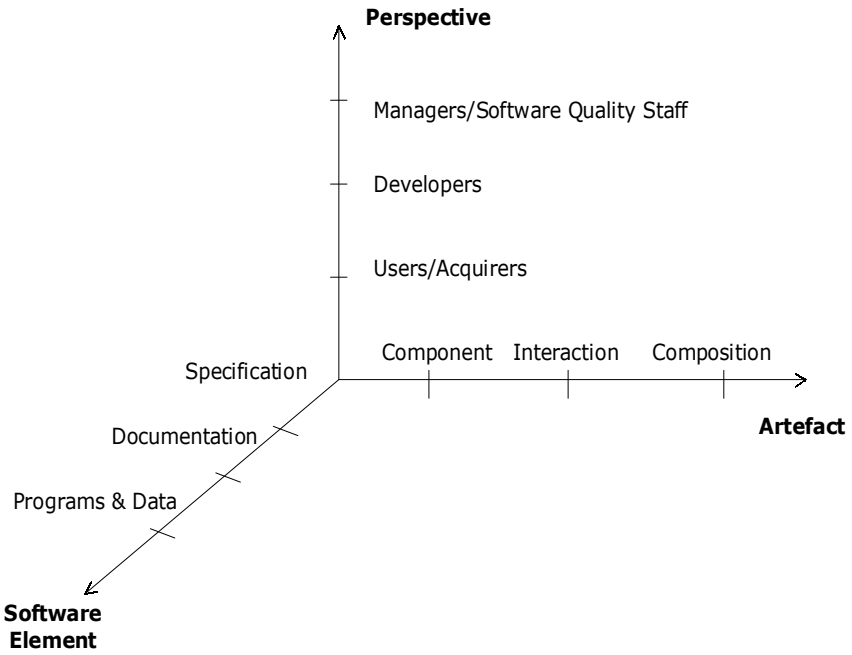

Figure 1: Space for classifying CBS assessments

3. INSIGHTS FROM THE CLASSIFICATION SCHEME

The classification scheme provides insight into what quality characteristics or managerial features are relevant to each assessment. We also point out to methods and techniques that could be used in the evaluation of different component artefacts. CBS assessments focus on evaluating COTS and traditional components, interactions, and compositions. We consider this assessment focus as a classification where different goals, features, procedures, and techniques are associated to each class of assessment. For example, *component quality assessment* focuses on evaluating COTS components as well as traditional components. COTS assessment and selection is the most crucial phase in the COTS-based cycle. Here long-term decisions on which COTS will be used in a software system are made. Non-optimal COTS software used in the development of a system can become extremely costly for a software organisation. Apart from the general reuse problems (selection, integration, maintenance, etc.), COTS products have by their own specific problems:

- Incompatibility: COTS components may not have the exact functionality required; moreover, a COTS product may not be compatible with in-house software or other COTS products.
- Inflexibility: usually the source code of COTS software is not provided, so it cannot be modified.
- Complexity: COTS products can be too complex to learn and to use imposing significant additional effort.
- Transience: different versions of the same COTS product may not be compatible, causing more problems for developers.

Therefore, repeatable and systematic methods to assess and select COTS software are an important issue in COTS-based software engineering [1][2][3][4][5][6].

On the other hand, traditional (bespoke) components are evaluated by identifying many component properties (e.g., functionality, limitations, pre-conditions) recording which properties are in conflict with other expected properties. Current approaches in aiding program understanding can be characterised as "white-box" understanding because they rely primarily on source code [12]. However, white-box understanding is not useful for COTS components because source code of components is not available. As a result, other methods for understanding, referred to as black box understanding methods, must be used [13].
The second class of assessment is *interaction quality assessment*. For example, the work presented in [7] proposes a general classification of possible types of mismatches between COTS products and software systems, which includes architectural, functional, non-functional, and other issues. The incompatibilities are essentially failures of components' interactions, so authors claim that finding and classifying these interactions will help to find and classify the incompatibilities. As another example, in [8] a reference model that relates key abstractions of software architecture and software component technology is presented. In this case, there is a strong condition to integrate architectural reasoning and component technology: that the architecture that is used to reason about the quality attributes of a system is very similar to (approaching isomorphism) the structure of a deployed component assembly that implements the system. Hence, interaction quality assessments constitute a core and particular value when evaluating CBS.
Finally, the third class of assessment is *composition quality assessment*. Component development is a traditional development process since all the usual lifecycle phases are traversed. However, the application composition process differs from a traditional process in some phases. In requirements, and even more so in design, the component market must be taken into consideration. Finding components that match arbitrary requirements will be difficult or impossible without considering availability and functionality of components. Moreover, once the decision of reuse a certain component is made, it will have to be configured within a component composition. As a way of alleviate these problems, the method presented in [9] focuses on the role of the connectors to provide causal connection between the components and coordinate the component behaviours to satisfy the scenario specific ordering constraints. As another example, proposal in [10] uses software metrics to guide quality and risk management in a Component-Based Systems (CBS), accurately quantifying various factors contributing to the overall quality, and identifying and eliminating sources of risk.
COCOTS model [14] is actually an amalgam of four related sub-models, each addressing individually primary sources of COTS software integration costs. COCOTS model can be used along with most of the methods previously mentioned.
We should notice that many other assessment proposals exist. For brevity reasons, only some of the most widespread methods or techniques that would be applicable to each assessment are mentioned here. Interested readers are referred to [15] for more information.

3.1 Using the multidimensional model

To facilitate CBS assessment understanding, the multidimensional schema refines the analysis along combined dimensions. The paragraphs below highlight the major features in some cases.
First, for the *perspective* dimension we consider the *specification* value on the *software element* dimension and the *component* value on the *artefact* dimension.

In the first case, *developers* evaluate a traditional component from three different viewpoints:

- As a component itself: when considering a component, developers should evaluate component's functionality, reliability, and robustness using techniques such as white-box testing, etc.;

- Its interactions: interoperability of components is the main goal to achieve since the component's interactions are evaluated as a way of integrate component's functionality in different contexts; and
- Its role as part of a component composition: reusability, maintainability, and efficiency are the main quality goals, considering that a component is created to be easily reused and updated, dealing with volatile requirements modelling.

Users/acquirers are concerned with selecting the right component. This goal changes the focus of evaluations to include COTS components as well as traditional components evaluations. For COTS components, a different situation arises:

- As a component itself: selecting components and defining selection criteria are the main issues. Some methods and techniques such as the OTSO (Off-The-Shelf-Option) method [3], or the CAP method (COTS Acquisition Process) [4] include these issues. Figure 2 explicitly shows this dimension value.
- Its interactions: adaptability of components is now the main goal to achieve. Users/acquirers see interfaces as the connection used to "plug in" the selected component; hence adaptable provided/required interfaces are focused on. Proposals in [7][9][15] clearly address incompatibilities detection.
- Its role as part of a component composition: COTS components, along with traditional components, contributes to build a composition that should provide quality in use characteristics such as performance or usability to the final product; hence evaluation focuses on individual COTS contributions to safety or satisfaction in a specified context.

Finally, *managers/software quality staffs look* at components as artefacts to be developed and/or acquired under reasonable costs and time-to-market, and compliance with a quality standard. We focus on:

- As a component itself: selection effort and development effort should be traded-off. Several proposed methods address the importance of defining a selection effort calculation. They provide some hints [1][2][3][4][14][15], even though much research must still be done to validate most of the proposals.
- Its interactions: here a balance among reusability, maintainability, and generality must be reached. Managers should aware of potential problems derived from developing/selecting components whose interfaces supply more functionality than what is actually needed for a particular application. In general, developing or buying too much, trying to get a "general component for everything" might increase interface complexity in such a way that application maintainability (and other quality properties) becomes an impossible goal to reach.
- Its role as part of a component composition: integration effort calculation would facilitate a faster time-to-market as well as get more accurate cost and time features. The managerial/quality view focuses on those issues looking at a component as a building block for compositions. A component contributes to the composition functionality by directly supplying part of the functionality or indirectly supporting it.

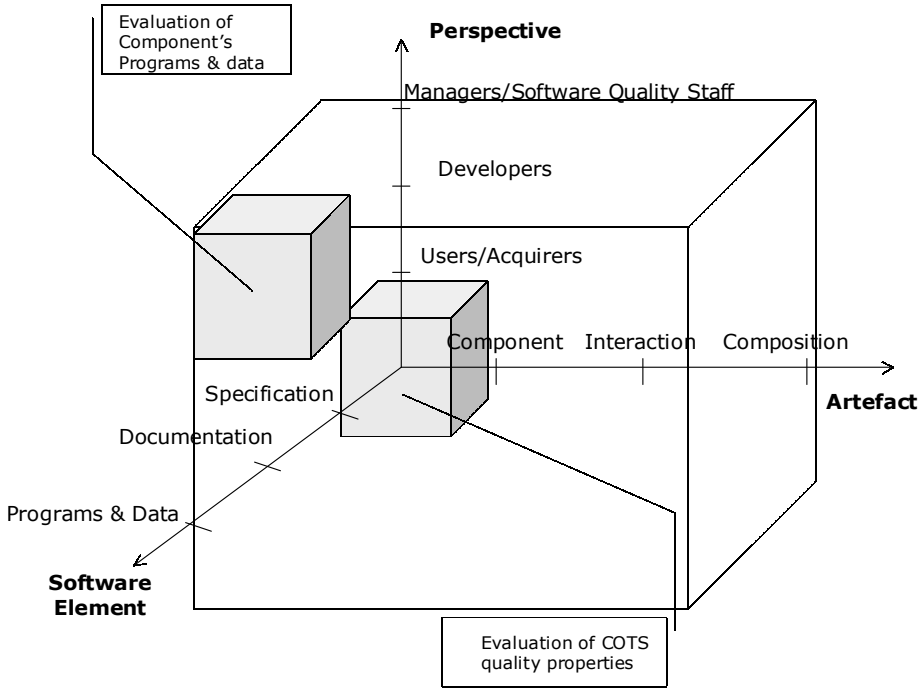

Figure 2: Insights from the classification scheme

We have analysed the component value on the artefact dimension along two combined dimensions (specification on the software element dimension, and all possible value on perspective dimension). Lets now briefly go inside the *documentation* and *programs & data* values on the software element dimension. We again consider the *component* value on the artefact dimension and some possible values on the perspective dimension.

As ISO/WD-Std 12119 emphasizes, *documentation* is another core element to be evaluated. Documentation should contain the information necessary to understand all the functions stated in the specification description, and the user's documentation should completely describe all user-callable functions in the program. During the evaluation it is necessary to gather information about functional features, and other characteristics of the package; for example, description of the system environment, previous product versions, customer's support strategies, etc. Hence, documentation's quality is crucial for a correct understanding and evaluation of COTS, which sometimes demand a translation of several types of vocabulary. Even some approaches suggest the use of particular notations such as use cases or scenarios, including essential customer requirements and customer standards, or the extension of UML diagrams with ADLs (Architecture Description Languages) [5][9], few of them particularly address documentation quality as a main element to understand COTS components.

As a final remark, *programs & data* should be analysed considering particular characteristics that are not isolated from other software elements to be evaluated. For example, it should be possible to install the programs successfully by following the information in the documentation; the programs and data should correspond to all statements in the specification description, etc. Of course, also functions must be executed in a correct manner for the programs and data. Some of those features can only be checked on the source code, while others are only run-time verifiable. Therefore, the evaluation of every value on

the artefact dimension has to be specialised to include its particular evaluation requirements and procedures.

4. CONCLUSION

In this paper, we proposed a three-dimensional classification scheme for assessing component-based systems. The classification scheme provides insight into what quality characteristics, managerial features, or assessment methods and techniques might be used for evaluating different component artefacts. We have shown how to use the scheme by discussing some quality aspects along some dimensions, particularly on the component value. Interactions and compositions are intended to be analysed in similar way.

Although the scheme addresses the main components of an evaluation process, some other equally important analysis must be taken into account, such as vendor feasibility analysis, context (execution environment as well as organisational environment) analysis, etc. Therefore, the three-dimensional scheme should be immerse into a wider spectrum to include those other features. Further discussion is needed in this direction.

ACKNOWLEDGEMENT

This work was partially supported by the CYTED (Ciencia y Tecnología para el Desarrollo) project VII-J-RITOS2 (Red Iberoamericana de Tecnologías de Software para la década del 2000).

REFERENCES

[1] Polen S., Rosen L., and Phillips B., Component Evaluation Process, *Software Productivity Consortium, Report SPC-98091-CMC*, Version 01.00.02, May 1999.

[2] Ncube C. and Maiden A., PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm, In *Proceedings of ICSE'99*, available at
http://www.sei.cmu.edu/cbs/icse99/papers/11/11.htm

[3] Ochs M., Pfahl D., Chrobok-Diening G., and Nothhelfer-Kolb, A Method for Efficient Measurement-based COTS Assessment and Selection - Method Description and Evaluation Results, *Fraunhofer Institut Experimentelles Software Engineering, IESE-Report No. 055.00/E*, Version 1.0, 2000.

[4] Kontio J., Caldiera G., and Basili V., Defining Factors, Goals and Criteria for Reusable Component Evaluation, In *Proceedings of CASCON'96 Conference*, Toronto, Canada, 1996.

[5] Kunda D. and Brooks L., Applying Social-Technical Approach for COTS Selection, *Proceedings of 4th. UKAIS Conference,* University of York, McGraw Hill, April 1999, http://www.cs.york.ac.uk/mis/

[6] Cechich A. and Piattini M., Methods for Measurement-Based COTS Assessments and Selection, *4º Argentinean Workshop on Computer Science Research, WICC 2002*, Bahía Blanca, Argentina, 16-17 May 2002, 262-266.

[7] Yakimovich D., Bieman J., and Basili V., Software architecture classification for estimating the cost of COTS integration, In *Proceedings of ICSE 99*, 1999, pp. 296-302, available at
http://www.cs.umd.edu/users/dyak/COTS/class2h.pdf

[8] Wallnau K., Stafford J., Hissam S., and Klein M., On the Relationship of Software Architecture to Software Component Technology, In *Proceedings of the 6th International Workshop on Component-*

Oriented Programming (WCOP6), in conjunction with the *European Conference on Object Oriented Programming (ECOOP),* Budapest, Hungary, 2001, available at http://www.sei.cmu.edu/pacc/Wallnau+-WCOP2001.pdf

[9] Bose P., Scenario-Driven Analysis of Component-Based Software Architecture Models, In *Proceedings of the First Working IFIP Conference on Software Architecture*, San Antonio, TX, USA , 1999, available at http://www.ece.utexas.edu/~perry/prof/wicsa1/final/bose.pdf

[10] Sedigh-Ali S., Ghafoor A., and Paul R., Metrics-Guided Quality Management for Component-Based Software Systems, In *Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC'01)*, 2001, 303-310

[11] ISO/WD 12199 - V4.: Software Engineering - Software product evaluation - Requirements for quality of Commercial Off The Shelf software products (COTS) and instructions for testing, *International Standard ISO/WD 121199*, ISO, November 2001

[12] Heineman G. and Council W., *Component-Based Software Engineering - Putting the Pieces Together*, Addison-Wesley, 2001

[13] Korel B., Black-Box Understanding of COTS Components. *In Proceedings of the Seven International Workshop on Program Comprehension,* IEEE Press, 1998.

[14] C. Abts and B. Bohem, *COTS Software Integration - Cost Modeling Study*, University of Southern California, 1997, http://sunset.usc.edu/research/COCOTS/docs/USAFReport.ps

[15] John Dean and Andrée Gravel, COTS-Based Software Systems, *Proceedings of the First International Conference, ICCBSS 2002*, Orlando, FL, USA, February 2002, Springer-Verlag LNCS 2255, ISBN 3-540-43100-4.