Report of the International Symposium on Component-Based Software Engineering

Ivica Crnkovic[1], Ralf Reussner[2], Heinz Schmidt[3],
Kevin Simons[4], Judith Stafford[5], Kurt Wallnau[6]

[1]Mälardalen University, Department of Computer Engineering, Sweden, ivica.crnkovic@mdh.se
[2]Universität Oldenburg, Germany, reussner@informatik.uni-oldenburg.de
[3]Monash University, Australia, Heinz.Schmidt@csse.monash.edu.au
[4]Tufts University, Department of Computer Science, USA, kevin.simons@tufts.edu
[5]Tufts University, Department of Computer Science, USA, jas@cs.tufts.edu
[6]Software Engineering Institute, Carnegie Mellon University, USA, kcw@sei.cmu.edu

Abstract

The International Symposium on Component-Based Software Engineering (CBSE7) was held at 28th International Conference on Software Engineering in Edinburgh, Scotland, May 24-25, 2004. The Symposium brought together researchers and practitioners from several communities: component technology, composition languages, compositional analysis, software architecture, software certification and scientific computing. The primary goal of the symposium was to continue clarifying the concepts, identifying the main challenges and findings of predictable assembly of software components. This report gives a comprehensive summary of the position papers, of the symposium, its findings, and its results.

1 Introduction

The International Symposium on Component-Based Software Engineering was held at the International Conference of Software Engineering 2004 (ICSE 2004) is a direct continuation of the previous CBSE workshops [5].

The first in the ICSE CBSE series of workshops to focus on predictable assembly, CBSE4, was held in Toronto, Canada May 2001. CBSE4 focused on reasoning about properties of assemblies based on properties of components and their interactions [1]. Researchers from three communities: component technology, software architecture, and software certification, joined the workshop, resulting in lively discussion and increased understanding of how the domains can be mutually informing. The need for a model problem, to be utilized for further research of different aspects of predictable assembly, was identified.

The specification of model problems was discussed at a follow-up workshop held at the Carnegie Mellon University's Software Engineering Institute in Pittsburgh, U.S.A. The aims of CBSE5 were defined at the SEI workshop: to more deeply study the problem of predictable assembly, focusing on the sub-problem of compositional reasoning, and benchmarks of the effectiveness of compositional reasoning [2]. Submitters were asked to address the community model problem, either directly or indirectly by adopting the vocabulary of its specification. Much of the discussion during CBSE5 revolved around the nature of compositional reasoning, resulting in a decision to focus CBSE6 on this topic.

CBSE6 was held in 2003 with the primary goal of achieving better understanding of the state of the art in automated compositional reasoning and prediction. While emphasizing state of the art, the workshop aimed at bridging theory and practice [3]. Attendees of CBSE6 represented a growing community of committed researchers and practitioners focusing on the problem of predictable assembly and it was decided to during the final session of the workshop to mature the CBSE series of workshops into a Symposium and thus the first International Symposium on Component-Based Software Engineering was held in Edinburgh in 2004.

This rest of this report is organized as follows: Section 2 gives an overview of the symposium purpose and goal. Section 3 describes symposium participation and Section 4 describes the four tracks of the symposium. The paper concludes with description of future plans.

2 The Aim of the Symposium

Component-based Software Engineering (CBSE) is concerned with the development of software intensive systems from reusable parts (components), the development of such reusable parts, and with the maintenance and improvement of systems by means of component replacement and customization. Although it

holds considerable promise, there are still many challenges facing both researchers and practitioners in establishing CBSE as an efficient and proven engineering discipline. CBSE has been the focus of six workshops, which have been held consecutively at the most recent six International Conferences on Software Engineering. The premise of the last three CBSE workshops was that the long-term success of component-based development depends on the viability of an established science and technology foundation for achieving predictable quality in component-based systems. The intent of this symposium was to build on this premise, and to provide a forum for more in-depth and substantive treatment of topics pertaining to predictability. This symposium brought together researchers and practitioners from a variety of disciplines related to CBSE to help establish cross-discipline insights, to provide a forum for presenting and discussing innovative approaches to CBSE, and to improve cooperation and mutual understanding. As a result of growing interest in CBSE from different communities and increased impact of the research results the contributions to CBSE7 have achieved a maturity and relevance level which called for their publications in conference proceedings. Indeed the accepted papers have been published in the symposium proceedings [4].

2.1 Symposium Objectives

The primary goal of CBSE7 is to achieve better understanding of the state of the art in automated compositional reasoning and prediction. While emphasizing state of the art, the symposium aims at bridging theory and practice.

Issues of particular interest included:

- Measurement and Prediction of Extra-Functional Properties
- Generation and adaptation of component-based systems
- Verification, testing and checking of component systems
- Compositional reasoning techniques for component models
- Measurement and prediction models for component assemblies
- Patterns and frameworks for component-based systems
- Extra-functional system properties of components and component-based systems
- Static and execution-based measurement of system properties
- Assurance and certification of components and component-based systems
- Component specifications
- Development environment and tools for building component-based systems
- Components for real-time, secure, safety critical and/or embedded systems

Eighty-two papers were received, of which twelve long and 13 short papers were accepted. All papers were reviewed by at least three, independent reviewers. Papers of PC members were reviewed by four non-conflicted reviewers. Approximately sixty persons attended the Symposium.

3 Symposium Sessions and Presented Papers

After a brief welcome to the Symposium, Oscar Nierstrasz of the Software Composition Group, University of Bern, Switzerland presented the first keynote talk in which he discussed how software components can help reduce the negative effects of change on long-lived software systems. Next, attendees selected to attend one of two tracks for the next two sessions of the symposium. Each track consisted of a set of short paper presentations, which was followed after lunch by a session devoted to discussion of issues raised during the talks. The final session of the day brought all participants together to review results from the tracks and discuss common themes. Day two began with a keynote talk by Hans Jonker from Philips Research Laboratories Eindhoven, The Netherlands on the topic in which he discussed required characteristics of component interfaces to truly support composition. Reports from the four tracks follow.

3.1 Track I –Generation and Adaptation of Component-Based Systems

The session started with a short presentation of five papers followed by a working session on the topic.

3.1.1 Paper Presentations

The work on “An Open Component Model and Its Support in Java” by E. Bruneton, T. Coupaye, M. Leclercq, V. Quema and J-B. Stefani was presented by Thierry Coupaye. The paper introduces a novel flexible component system developed at France Telecom, including aspect-oriented component adapters such as controllers and interceptors for hierarchical and dynamic component composition. Marija Mikic-Rakic presented her project on “Software Architectural Support for Disconnected Operations” co-authored by N. Medvidovic. The project aims at a compositional approach to designing and monitoring component-based mobile distributed systems for achieving predictable availability. Soo Dong Kim presented on “Using Smart Connectors to Resolve Partial Matching Problems in COTS Component

Acquisition" by H.G. Min, S.W. Choi and himself. His generated connectors bridge between a given specification of required interfaces and mismatching provided interfaces of partially suitable components implementing these interfaces. Common adaptations in include parameter value range adaptations, signature mismatch and protocol or workflow adaptations. "Correctness of Component-Based Adaptation" by S. S. Kulkarni and K.N. Biyani was presented by the first author with a focus on identifying invariants to be preserved by correct dynamic adaptations of components. Finally, Sascha Alda presented "Strategies for a Component-Based Self-adaptability Model in Peer-to-Peer Architectures" co-authored with A.B. Cremers. Self-adaptable components were identified and components reacting with adaptations to exceptional conditions. Consequently an architecture was introduced for managing and detecting peer-to-peer failures and interaction constraint violations as a common platform for self-adaptable components.

3.1.2 Discussion

The initial discussion circled around the difference between adapters and components.

On the one hand, clearly, adapters are not components but are connector-like. Unlike regular connectors however, it was agreed, adapters deal with and fix mismatches in component interfaces or ports. Adaptation requires mismatch. Adapters and adaptation are based on a suitable definition of 'match' and 'mismatch' in the given context. A boundary was drawn between adaptation and the twins, customization and configuration. For example internationalizations such as switches between languages were put into the customization/configuration box, while protocol converters were considered examples of adapters.

On the other hand, adapter libraries or generators, offer adapters as a kind of components – however not self-contained or independent but always relative to specific interfaces of other components. Adaptation moreover is a more general concept than adapters: components may adapt to changing context of deployment (for example in a mobile environment), in that they change their behavior dynamically and in response to external failure, the presence or absence of certain other components or services. Adapters, in contrast, are more static offering conversion of fixed protocols or interfaces in an anticipated fashion.

The following problem areas were identified under this theme of component-based adapters, adaptation and generation:

- Adaptation as change or evolution aiming to break the stability of a system at one level to achieve stability at another;
- Managing CB adaptation and evolution across several levels and maintaining the integrity and stability of a system including required extra-functional system properties such as availability and other performance properties;
- Lack of a taxonomy of adaptation and adapters, such as restricted to parameter data, signature renaming, interface, assembly, hierarchical architectural composition etc.;
- Interface versus component behavior adaptation;
- First-class adapters and the reification of adaptation and evolution mechanisms;
- Fundamental self-similarity of evolving component-based systems.

3.2 Track II – Tools and Building Frameworks

As with the other sessions, the *Tools and Building Frameworks* session began with brief presentations of the accepted papers and was followed by an in-depth discussion of common themes running throughout the papers. A brief summary of the presentations is offered here, with an overview of the discussion that followed.

3.2.1 Paper Presentations

The work "Correct Components Assembly for a Product Data Management Cooperative System" was presented by Valentina Presutti on behalf of Massimo Tivoli, Paola Inverardi, Alessandro Forghieri and Maurizio Sebastianis. A tool developed by this group automatically generated component connectors that ensure correct coordination policies, based on the component interface specifications given in Microsoft Interface Definition Language (MIDL) and the coordination policies given as Linear Temporal Logic (LTL).

The software management perspective was presented by Louis Taborda in his work, "The Release Matrix for Component-Based Software Systems". This presentation highlighted the challenges associated with managing software releases for component-based systems. The release matrix approach generalized the software release plan, illustrating the complex coordination between releases of a software product, as well as versions of the components that make up the product.

John Hutchinson presented a work he co-authored with Gerald Kontonya entitled "Viewpoints for Specifying Component-Based Systems". Hutchinson's presentation highlighted the difficulties of specifying component-based system using traditional requirements elicitation techniques. This paper outlines a new method called

Component-Oriented Requirements Expression (COREx). The COREx method involves specifying a set of viewpoints, specifying Actor and Stakeholder requirement. A set of filters are applied to these views in order to facilitate component selection. This process of component selection acknowledges that component specifications will not meet all of the requirements, and is therefore a process of compromise.

The session continued with the presentation of Kevin Simons and Judith Stafford's submission "CMEH: Container-Managed Exception Handling for Increased Assembly Robustness". Simons presented the deficiencies with current exception-handling techniques in component-based systems. A framework for handling and recovering from exceptions in COTS-based systems was presented.

"A Framework for Constructing Adaptive Component-Based Applications: Concepts and Experiences" was presented by Humberto Cervantes, a paper co-authored by Richard Hall. This work leverages GRAVITY, a component framework that allows for run-time adaptation of functionality based on the availability of constituent components. The framework manages the application through relationships specified between component interfaces, based on cardinality requirements, policies and filters.

The final presentation was entitled "Testing Framework Components". This work, co-authored by Benjamin Tyler and presented by Neelam Soundarajan, presents a method for testing object-oriented framework components. These frameworks provide templates with "hook" methods that are implemented by application developers to facilitate the creation of new systems. The leveraging of polymorphism in these systems makes it very difficult to test based on functional specifications. This work presents a method for testing these framework components based on interaction specifications (describing the templates' use of the hook methods), which require no modification to the framework source code.

3.2.2 Discussion

Following the presentations, the attendees participated in an open discussion, facilitated by Kurt Wallnau. One of the initial discussions involved the applicability of the release matrix approach of release management of component-based systems to applications built using the GRAVITY framework. It was proposed the perhaps a system with a constantly changing component makeup may pose a challenge to the release matrix, an approach created with a more or less fixed system in mind. However, while the definition of a release is both loose and dynamic in a GRAVITY application, the release matrix scheme of software management may still prove to be useful.

The remainder of the discussion focused on finding a common thread amongst the remaining papers and presentations. There was an immediately connection drawn between the presentation involving the automatic generation of coordinating connectors and the exception handling paper. Both seemed to be based on enforcing some sort of policy decision (be it coordination or the handling of erroneous behavior) outside of the component itself, in some sort of connector or interceptor.

This discussion led (inevitably) to the differences between connectors and interceptors, as well as the difference between connectors in this sense and architectural connectors. Some argued the difference between connectors and interceptors is a matter of transparency; interceptors are a transparent means of cross-cutting method invocations and applying some sort of policy. However, no conclusions were agreed upon in terms of the transparency of connectors. In the example of the coordinating connectors, the connector is itself transparent to the components. The components simply invoke methods on other components, and the connector coordinates the temporal interaction policy.

The GRAVITY container was also examined in order to see if its binding policy enforcement related to the policies enforced by the coordination and exception handling policy.

Despite the lack of consensus on terminology, one finding became very clear through these discussions. There is a definite need for some sort of connectors or interceptors for enforcing policies outside of the components. While there are certain policies that can be self-contained within a component, there are some policies that need to be handled outside of the components. It's not that these policies can't be handled inside components; it's that they shouldn't be handled inside the components. Components are meant to be reusable units of software deployment. The component should not be concerned with the coordination policy of the entire system. It should simply be able to perform its task as specified, and the coordination policies can be enforced by the system assembler via connectors, interceptors, etc.

There are a tremendous number of future research directions involving using interceptors and connectors for enforcing policies external to the components. Currently, there is work being performed in ensuring Quality of Service, security, error and exception handling, coordination and many more, both through automatically generated and hand-coded connectors.

3.3 Track III – Components for Real-Time Embedded systems

This track has shown that there is a clear trend of CBSE adoption in real-time and embedded systems domains.

3.3.1 Paper Presentations

The paper "Industrial Requirements on Component Technologies for Embedded Systems", with co-authors Anders Möller, Joakim Fröberg, Mikael Nolin was presented by Anders Möller. The work has focused on feasibility of component-based technologies in embedded system domain, in particular vehicular systems. Software component technologies have not yet been generally accepted by embedded-systems industries. In order to better understand why this is the case, the work presents a set of requirements, based on industrial needs, that are deemed decisive for introducing a component technology. The requirements present are aimed for evaluation of existing component technologies before introducing them in an industrial context. One of the findings of the paper is that a major source of requirements is non-technical in its nature. For a component technology to become a valuable solution in an industrial context, its impact on the overall development process needs to be addressed. This includes issues like component life-cycle management, and support for the ability to gradually migrate into the new technology.

The paper "Prediction of Run-time Resource Consumption in Multi-task Component-Based Software Systems", authors Johan Muskens and Michel Chaudron was presented by Johan Muskens. The authors start from a permission that embedded systems must be cost-effective. This imposes strict requirements on the resource consumption of their applications. It is therefore desirable to be able to determine the resource consumption of applications as early as possible in its development. This paper discusses a method for predicting run-time resource consumption in multi-task component-based systems based on a design of an application. Previously the authors have presented a scenario based resource prediction technique and showed that it could be applied to non-pre-emptive non-processing resources, like memory. In this paper the authors extend this technique, which enables to handle pre-emptive processing resources and their scheduling policies. Examples of these classes of resources are CPU and network. For component-based software engineering the challenge is to express resource consumption characteristics per component, and to combine them to do predictions over compositions of components. Finally the authors present a model and tools, for combining individual resource estimations of components. These composed resource estimations are then used in scenarios (which model run-time behavior) to predict resource consumption.

Kristian Sandström presented a paper "Introducing a Component Technology for Safety Critical Embedded Real-Time Systems" co-authored by Kristian Sandström, Johan Fredriksson, and Mikael Åkerholm. The work was focused on development of a component model adjusted to the requirements of embedded systems. In the paper the authors show how to use component based software engineering for low footprint systems with very high demands on safe and reliable behaviour. The key concept of the approach is to provide expressive design time models and yet resource effective runtime models by statically resolving resource usage and timing by powerful compile time techniques. This results in a component technology for resource effective and temporally verified mapping of a component model to a commercial real-time operating system.

The paper "A Hierarchical Framework for Component-Based Real-Time Systems", co-authored by Giuseppe Lipari, Paolo Gai, Michael Trimarchi, Giacomo Guidi and Paolo Ancilotti, was presented by Giuseppe Lipari. This paper describes a methodology for the design and the development of component-based real-time systems. In the presented model, a component consists of a set of concurrent real-time threads that can communicate by means of synchronized operations. In addition, each component can specify its own local scheduling algorithm. The paper also discusses the support that must be provided at the operating system level, and present an implementation in the shark operating system.

Finally, the paper "Design Accompanying Analysis of Component-Based Embedded Software" was presented by its author Walter Maydl. The paper presents a design accompanying analysis techniques for component-based embedded systems based on the dataflow paradigm. Components are modeled as functions on streams of signal data which allows describing the behavior of dataflow components precisely by constraints. Static constraints, e.g., equality of sampling periods, may be as complex as multivariate polynomials and are enforced by a new interface type system. Dynamic constraints, e.g., describing communication protocols, are checked using a novel model checking technique based on fifo automata. The objective of these mathematically well-founded analysis techniques is to detect as many program errors as possible during design. Moreover, the component model is compositional resulting in well-defined hierarchical abstraction. Altogether, the aim of this approach is to achieve a more reliable development of complex applications in a shorter design time.

3.3.2 Discussion

The first part of the discussion was related to question *whether component-based approach has advantages or not in development of real-time and embedded systems*. Component-based approach addresses business concerns (for example more efficient development, reuse), but not the characteristic requirements of real-time and embedded systems. Further some concepts of CBSE, such as higher abstraction may hurt analyzability of the system which is very important for RT systems. So the main question was *whether component-based approach can help in development and maintenance of such RT system?*. The experience from the PECOS project, which aim was providing support for development of small embedded systems, has shown that specification of component model was a crucial part for a successful performance and results of the project. This component model was designed for satisfying time, space, energy constraints

Another question discussed was *should the component-based approach include predicting qualities of systems, and support for building systems with predictable qualities*. Related to these questions is a question *to what extent are specific properties, such as real time properties, a necessary part of a component model?* In real-time analysis the concern is for interaction among tasks (or objects, or primitives of "time"). Should we keep these concerns outside component models or should they be an incorporated part of the component models? The experience in use of general-purpose component models has shown that different problems may appear if these concerns are not built in a component model. For this reason it is important that component models explicitly address them and that the components include specifications of these properties. It was noticed that components are "componentized" by separation of a certain kind of concern, and for RT systems there exist particular separation criteria related to RT properties which should be incorporated into RT component models.

Embedded systems are usually resource-restricted and for this reason it is important that component models do not consume additional resources. For example composition should have a "zero effect" at the run-time. This opens a question to which extent should component model be enriched with capabilities for specification of different properties and providing support for composability of these properties. The answer is that a tradeoff between implicitly, efficiency and composability must be found and the result will be existence of different component models considering different concerns.

3.4 Track IV – Extra-functional properties of components and component-based systems

As with the other sessions, this session began with the paper presentations including discussion of paper related questions. Then we moved on to a general discussion on quality of service for components and component based systems

3.4.1 Paper Presentations

The session started with the presentation Olivier Defour, Jean-Marc Jézéquel, and Noel Plouzeau's paper, "Extra-functional Contract Support in Components". In this presentation, a language for specifying QoS contracts was introduced and, through example, it was shown how to use this language for validating components and component assemblies. In addition, it was shown how to derive from the contracts QoS constraints of the CLP (constraint logic programming language).

Paola Inverardi gave a presentation for Antonia Bertolino and Raffaela Mirandola on "CB-SPE: Putting Component-Based Performance Engineering into Practice". In this presentation an extension of the Software Performance Engineering approach (SPE) by C. Smith with reusable component performance specifications was introduced. In addition to this conceptual extension, a novel tool using the real-time UML performance profile for supporting component-based SPE was presented.

The presentation on "Component Technology and QoS Management" by Goerge T. Heineman, Joseph Loyall, and Richard Schantz discussed current approaches to QoS management, such as static analysis and prediction techniques, run-time enforcement of QoS and QoS middleware extensions. Thereafter, "quoskets", a packaged unit of reusable QoS related behavior and policy were introduced. Similar to components, qoskets can be composed to larger qoskets and can be decomposed. This approach supports reasoning on system QoS requirements in terms of component QoS requirements.

Rob Armstrong reported on "Computational Quality of Service for Scientific Components", a paper written by Rob, Boyana Norris, Jaideep Ray, Lois C. McInnes, David E. Bernholdt, Wael R. Elwasif, Allen D. Malony, and Sameer Shende. When components are used in scientific computing, performance is of utmost importance. This fact and the usual massively parallel environment lead to very specific requirements to component models. The talk discussed how these requirements shaped the CCA (common component architecture) for computational sciences. In addition, the talk discussed was reported on the QoS of such CCA components, namely performance and accuracy of scientific simulations.

The paper of Rakesh Shukla, Paul Strooper, and David Carrington, entitled "A Framework for Reliability Assessment of Software Components" describes a

conceptual framework for measuring the reliability of software components. The framework combines statistical testing, as known for example from the cleanroom approach and oracle tests, for self checking of components. The framework supports the execution of test cases and the evaluation of their output. Given a usage-profile, the framework also supports the generation of test cases.

The presentation of Markus Meyerhöfer and Christoph Neumann introduced "TESTEJB -- A Measurement Framework for EJBs". This framework uses interceptors to measure the QoS of Enterprise Java Beans. After a discussion on the usage of the interceptor-pattern and measurements for the open-source EJB server JBOSS, the presentation looked at the details of a specific interceptor for response time measurements. However, the framework itself supports measurements of many other QoS properties by using different interceptors.

Last, but not least Sten Loecher gave a presentation on "Model-Based Transaction Service Configuration for Component-Based Development. The talk summarized the current situation of transaction management services and discussed issues related to attribute-based approaches to transaction service specification, such as restricted attribute sets, unclear responsibilities and missing support for requires-interfaces. Thereafter, a model-based procedure for transaction service configuration was presented. This approach has the benefit of early detection of configuration errors in designs and an increased predictability of the design through analysis.

3.4.2 Discussion

As containers and interceptors were mentioned in several talks, we started the discussion on containers and their impact on component run-time behavior. However, we soon realized that containers or, more generally interceptors, are a solution technique. The problem they solve is how to add aspects to a system and how to weave extra-functional attributes to a system (e.g., superimposing transactional services to a system). In discussions related to quality attributes, we agreed that a quality of service (QoS) is not a constant property of a component, but highly depends on the component's environment. It was recognized that this is not a new insight as it has been previously discussed in other Component-Based Software Engineering forums including ECBS 2002 [7], and WCOP 2003 [8], but this phenomenon has not been solved and is sufficiently important to be worthy of further discussion. We concluded that component QoS must be modeled as a function of three parameter groups:

- **QoS of run-time environment**: for example, the performance of a component is influenced by the virtual-machine it runs on, the operating system (and its resource scheduling policy) and the underlying hardware. Similar arguments hold for reliability.
- **QoS of external services**: as a component service most often relies on the results of external services called, the QoS of such external service influences the "end-to-end" QoS the component service has. (For example, for reliability this fact is demonstrated by models and measurements).
- **The usage profile**: The way a component is used, influences its QoS. This is true per definition for reliability, as software reliability is modeled as a function of the usage profile. However, even QoS attributes which are not always modeled as a function of the usage profile (such as the various performance metrics), usually depend on the parameters given by the service call.

Because these factors are not part of the component, but rather of the component's environment, one needs parametric component QoS descriptions. However, parametric component description will only allow prediction of the quality of provided component services in dependency of the above mentioned parameters. A harder problem is to compute QoS requirements a component requests from its environment and the externally called services in dependency of requested QoS of the component. This is because, there is a functional dependency from external QoS factors to the QoS of provided services, but this function is not injective. Many different combinations of values of the above-mentioned influence factor will result in the same QoS of a provided service. For example, one may think about a slower run-time environment but (some) fast external services may be as good as a fast run-time environment combined with some slow external services. This fact hinders the functional description of the dependency of external requested QoS on the QoS requested from a component by its users. Instead, one can use constraints; given a QoS requirement to a component and a component to fulfill this requirement, one can state a set of constraints on the QoS of the run-time environment and the external services having a given usage-profile.

After discussing these problems of QoS specification, the group questioned what CBSE QoS researchers can learn from prior work in similar issues of hardware. Obviously, in hardware QoS is also of concern. However, we soon saw that there are also fundamental differences between hardware and software QoS models, even is the actual QoS property has the same name. For example, hardware reliability is usually modeled as a function of time (the older a product is, the more likely it fails). As there is no aging process for software (taking aside issues like architectural drift, etc which occur by manipulations of software over the time), it makes no sense to model software reliability as a function of the age of the software (at least if the software is not changed). Hence, there is an

urgent need to come up with specific software reliability models. Unfortunately, the ones we have (taking the usage profile into account) still do not reflect important influence factors (such as code complexity, programmer's experience or quality assurance measures, such as reviews, etc). Similar arguments also hold for other quality attributes, as software and hardware differ in many fundamental properties. For example, side-effects between software units are different in nature to the various physical side effects between hardware units.

Given these issues for good QoS models it was asked whether there is at least one quality attribute that we can claim to understand. We went through a list of possible answers and reliability immediately was ruled not to be a good candidate, for the reasons discussed above. Memory consumption was also not considered as such. Although it might look simple at the first glance, it is not clear how to model indirectly used memory or shared memory. Power consumption also was considered as difficult, because of its intriguing dependencies to hardware and software usage. Finally, we concluded that time is understood in a sense that we at least know how to measure it. As we realized in the later discussion, the term "understood" can be interpreted quite differently, such as "we can measure it", which is true for time and power consumption, or "we have models for it with somewhat precise predictions", as we have for some timing properties, or we have "compositional models", which is currently a topic for research. As models cannot be validated (falsified) without knowing how to measure the property, we agreed that “we can measure it” is the minimal requirement on "understanding" a property.

When comparing internal software properties, such as maintainability, one has several problems (a) how to define it formally, (b) how to measure it (which depends on the definition of course) and (c) how to find simple measures that can be measured early in the software development process and that correlate with the internal quality property. Therefore, these properties are probably least understood. Among other problems, it is unclear how to formally define these properties.

The session concluded, leaving open the question of how to model the influence of usage profiles and measurable quality attributes on the quality of component services.

4 Closing Session

The closing session was organized as a large-group facilitated discussion. The question “what is the key result of this workshop?” was offered as an opening gambit. As expected, there was no shortage of opinions. The result of the discussions, however, was not an answer to the opening question, but rather a series of related, and more detailed questions.

The first question that emerged was: “what value is added to system design by using components?” The participants were searching for answers that went beyond the usual vague generalities of “better, faster, and cheaper” Eventually, the discussion began to focus on the software components and software component technology as exhibiting a kind of architectural style or design pattern that, in effect, restricts developers of components and their assemblies in certain pre-ordained ways. But restricted to what end? On this question a consensus arose that these restrictions serve to make the development of certain classes, or families of systems systematic and with predictable quality.

This merely begged the further question, however: “What is the nature of analytic models for predicting the behavior of component-based systems? Are such models different in any way from models of non-component-based systems?” This led, inevitably, to a discussion of compositionality, and to the relation between the syntactic composition of software components (on their interfaces) and the analytic composition of behavior of those same components. Unlike previous workshops, the discussion did not founder on the topic of compositionality. Instead, the participants discussed those behaviors for which compositional analysis is reasonably well understood (e.g., resource consumption, such as time, power, and memory) and those which are not well understood (e.g., reliability).

Given that there are at least some behaviors for which compositional theories are obtainable, the next question is: “Can we deduce, from these theories, the restrictions that must ultimately be expressed as design restrictions in a component technology?” Several presentations at the workshop suggested a “yes” answer to this question—Mikic-Rakic’s presentation of Prism-MW, Chaudron’s presentation of Robocop, Fioukov’s presentation of APPEAR, Nierstrasz’s keynote references to PECOS, to name just a few, were all evidence in support of the affirmative.

From here the discussion turned to the question: “Are there common or perhaps canonical characteristics of component models lurking within these exemplars of component-based predictability?” The large group made it impractical to undertake an analysis of the various component technologies presented during the workshop. There was, however, a renewed call for some form of cross-project collaborative study of common features of component models for predictability.

The last question addressed to the group was whether it could produce a question that defines the component-based software community (in the sense that Kuhn asserted that a science communities are defined by a tacit agreement on which questions are worth answering)? The

agreed community question(s)—or at least a first approximation—was stated as:

"*What design and engineering qualities can be made observable, measurable, and predictable, in a component-based setting? And can we define theories and mechanisms for composing these qualities, and understand the limits of these theories and mechanisms?*"

With this formulation the workshop adjourned.

5 Workshop Results and Future Plan

5.1 Publication of Results

The proceedings of the symposium are available as Lecture Notes in Computer Science, Volume 3054/2004 [4]. The call for papers and symposium program are available on the web at both the Software Engineering Institute [2] and Monash University [3].

5.2 Future Plans

CBSE 2005, the International Symposium on Component-Based Software Engineering, will be co-located with the International Conference on Software Engineering in St. Louis, Missouri in May 14-15, 2005.

6 Acknowledgement

We would like to thank the program committee who contributed greatly to the success of the workshop.

The program committee:
Uwe Aßmann, Linköping University, Sweden
Jakob Axelsson, Volvo Car Corporation, Sweden
Mike Barnett, Microsoft Research, USA
Judith Bishop, University of Pretoria, South Africa
Jan Bosch, University of Groningen, The Netherlands
Michel Chaudron, University Eindhoven, The Netherlands
Wolfgang Emmerich, University College London, UK
Jacky Estublier, LSR-IMAG, France
Andre van der Hoek, University of California, Irvine, USA
Kathi Fisler, WPI, USA
Dimitra Giannakopoulou, NASA Ames, USA
Richard Hall, Imag/Lsr, France
Bengt Jonsson, Uppsala University, Sweden
Dick Hamlet, Portland State University, USA
George Heineman, WPI, USA
Paola Inverardi, University of L'Aquila, Italy
Shriram Krishnamurthi, Brown University, USA
Jeff Magee, Imperial College University, UK
Nenad Medvidovic, University of Southern California, USA
Magnus Larsson, ABB, Sweden
Rob van Ommering, Philips, The Netherlands
Heinz Schmidt, Monash University, Australia
Judith Stafford, Tufts University
Dave Wile, Teknowledge, Corp., USA
Kurt Wallnau, SEI, Carnegie Mellon University, USA

7 References

[1] I. Crnkovic, H. Schmidt, J. Stafford, K. Wallnau, 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction, Software Engineering Notes, November 2001.

[2] I. Crnkovic, H. Schmidt, J. Stafford, K. Wallnau, 5th ICSE Workshop on Component-Based Software Engineering: Benchmarks for Predictable Assembly, Software Engineering Notes, September 2002.

[3] I. Crnkovic, H. Schmidt, J. Stafford, K. Wallnau, 6th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction, Software Engineering Notes, May 2004.

[4] I. Crnkovic, H. Schmidt, J. Stafford, K. Wallnau , 7th International Symposium, Edinburgh, Scotland, May 24-25, 2004, Proceedings Series: Lecture Notes in Computer Science, Vol. 3054, ISBN: 3-540-21998-6

[5] http://www.csse.monash.edu.au/~hws/cgi-bin/JSS-ACBSE/

[6] http://www.sei.cmu.edu/pacc/events.html

[7] http://www.idt.mdh.se/ecbse/2002/

[8] http://research.microsoft.com/~cszypers/events/WCOP2003/