6th ICSE Workshop on Component-Based Software Engineering: *Automated Reasoning and Prediction*

Ivica Crnkovic[1], Heinz Schmidt[2], Judith Stafford[3], Kurt Wallnau[4]

[1]Mälardalen University, Department of Computer Engineering, Sweden, ivica.crnkovic@mdh.se
[2]Monash University, Australia, Heinz.Schmidt@csse.monash.edu.au
[3]Tufts University, Department of Computer Science, USA, jas@cs.tufts.edu
[4]Software Engineering Institute, Carnegie Mellon University, USA, kcw@sei.cmu.edu

Abstract

This report gives an overview of the 6th ICSE Workshop on Component-Based Software Engineering held at 25th International Conference on Software Engineering. The workshop brought together researchers and practitioners from three communities: component technology, software architecture, and software certification. The primary goal of the workshop was to continue clarifying the concepts, identifying the main challenges and findings of predictable assembly of certifiable software components. This report gives a comprehensive summary of the position papers, of the workshop, its findings, and its results.

1 Introduction

The sixth CBSE (CBSE6) workshop held at the 25th International Conference of Software Engineering (ICSE) is a direct continuation of the previous CBSE workshops [6].

CBSE4 focused on reasoning about properties of assemblies from properties of components and their interactions. 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 objectives of CBSE5 were defined at the SEI workshop.

The aim of CBSE5 was 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. 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.

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

2 The Aim of the Workshop

The premise of the CBSE workshop series is that the long-term success of component-based development depends on the ability to predict the quality of component-based systems; however, developers are currently unable to make such predictions. Further research is needed in the area of predictable assembly to develop a component composition theory for reasoning about both the functional and extra-functional properties of component assemblies based on the properties of components. Issues related to developing a composition theory include determining what properties are of interest to developers and users of components, how to predict the properties of assemblies, how to measure properties of components, how to verify the measurements, and how to communicate the property values to component users. Resolving these issues requires collaborative work of researchers in several domains including compositional reasoning, composition languages, component trust and certification, software architecture, and software components.

2.1 Workshop Objectives

The primary goal of CBSE6 is to achieve better understanding of the state of the art in automated compositional reasoning and prediction. While emphasizing state of the art, the workshop aims at bridging theory and practice.

Issues of particular interest included:

- generation and adaptation of component-based systems;
- automatic verification, testing and checking of component systems;
- automated management of software architectures, product-lines, variation and configuration;
- algorithms for automated component-based software engineering;
- compositional reasoning techniques for component models;
- aspect-oriented models and automated weaving of component software;
- 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.

3 Participating in the Workshop and Workshop Organization

Attendance at the workshop was, in large part, by invitation based on acceptance of position papers. Papers submitted to the workshop addressed to following:

- provided an overview of the domain by way of background

- described a family of components associated with the problem
- crisply state properties that developers want answered about components, assemblies, architectures or product lines
- discuss a critical issue or describe a novel approach, method or tool, for automated reasoning or prediction
- discuss the plausibility and benefits by way of example or evaluation

Thirty six papers were received, of which eighteen were accepted. Full papers were reviewed by at least two, most by three, independent reviewers. Forty-two persons attended the workshop.

4 Workshop Sessions and presented papers

The workshop was divided into eight sequential sessions; five working sessions held between a welcome session and a closing session. The opening session included a review of progress made at CBSE5 and with respect to the JSS special issue, second in special edition produced by this community. The closing included reviews of the six working sessions and discussion of directions for follow-on research.

The following sessions were organized:

- Measurement and Prediction of Extra-Functional Properties
- Specification and Runtime Verification
- Analysis, Design and Patterns
- Compositionality Issues for Extra-Functional Properties
- Generative Modeling and Synthesis

The abstracts of the papers presented in the sessions are listed below.

4.1 Session I: Measurement and Prediction of Extra-Functional Properties

A fundamental problem faced when attempting to predict properties of assemblies is determining what contribution properties of a component contribute to assembly behavior and how to identify and package that information for use in assembly prediction. The papers presented in the first session of the workshop addressed these issues in the area of performance analysis, reliability analysis, and run-time resource consumption.

Towards Component-Based Software Performance Engineering, ***Antonia Bertolino and Raffaela Mirandola***

Early and rigorous performance analysis of component-based systems is a crucial issue in software engineering to guarantee that the developed components and their assemblies will satisfy their quality requirements. We propose an original approach, called the CB-SPE, for component-based software performance engineering. CB-SPE relies on, and adapts to a CB framework, the concepts and steps of the SPE technology and uses for modeling the standard RT-UML profile, reshaped according to the CB principles. The approach is compositional in that it is applied first at the component layer for achieving parametric performance evaluation of the component in isolation, and then at the application layer for predicting the performance of the assembled components on the actual platform. We also outline the architecture of a tool supporting the automation of the proposed approach, and overview related work.

Measuring Component Reliability, ***John D. McGregor, Judith A. Stafford and Il-Hyung Cho***

Much of the research on component-based software engineering assumes that each component has a single service. This simplifies analyses but it is a significant departure from actual components. This paper reports on an investigation of the feasibility of using design constructs as a means of treating several methods as a single unit. Grouping the services of a component into a few sets satisfies the goal of simplicity while still providing the designer with a more usable model of component reliability.

Component Based Performance Prediction, ***Xiuping Wu, David McMullan and Murray Woodside***

Component Based Software Engineering (CBSE) exploits re-usability of configurable components to generate software products more quickly, and with higher quality. CBSE offers potential advantages for performance engineering. If most of a new system consists of existing software components, it should be possible to predict properties like performance more easily, than if all of the software is new. The performance-sensitive properties of the components can be extracted and stored in a library, and used to build a predictive model for the performance of a proposed product. This paper describes an approach based on performance submodels for each component, and a system assembly model to describe the binding together of library components and new components into a product. In this work a component can be arbitrarily complex, including a subsystem of concurrent processes. The description pays particular attention to identifying the information that must be provided with the components, with the bindings, and to providing for parameterization to describe different configurations and workloads.

Scenario-Based Prediction of Run-time Resource-Consumption in Component-Based Software Systems, ***Merijn de Jonge, Johan Muskens and Michel Chaudron***

Resources of embedded systems, such as memory size and CPU power, are expensive and (usually) not extensible during the lifetime of a system. It is therefore desirable to be able to determine the resource consumption of an application as early as possible in the design phase. Only then, a designer is able to guarantee that an application will fit on a target device. Resource prediction is a technique to estimate the amount of consumed resources by analyzing the design and/or implementation of an application. In this paper we concentrate on predicting memory consumption in component-based applications. Component-based applications complicate resource predictions because resource consumption is spread across individual components. The challenge is to express resource consumption per component, and to combine them to do predictions over compositions of components. To that end, we propose a model in which individual resource estimations of components can be combined. These composed resource estimations are then used in scenarios (which model run-time behavior) to predict memory consumption of applications.

4.2 Session II: Specification and Runtime Verification

The five papers presented in the second session of the first day of the workshop presented approaches for verifying the correctness of component-based applications. The papers presented by Hofmeister, Barnet, and Heineman focus on specifying component constraints or contracts and verifying that these are met at run-time. Jia focus on feature-rich applications and the runtime detection and resolution of feature interactions. Dijkman present a method for determining if the behavior of an application conforms with the behavior of the enterprise in order to increase user satisfaction.

Specifying Architectural Constraints on Components ***Wayne DePrince Jr and Christine Hofmeister***

Research to improve component reuse has focused on providing the specification of various behavior properties. In this paper we present our approach to this problem, which focuses not so much on specifying the behavior of the component, but instead on certain architectural constraints. We introduce our research project "lips", a language for formally capturing these usage constraints and a toolset for automatically providing for their enforcement at runtime. Our approach captures the usage constraints that are local to a particular component. In this way we express these restrictions on its reuse independent of an actual client or application. We then embed these constraints within the component's specification. Our toolset verifies that the component conforms to the specification and uses it to generate code which checks if the constraints are obeyed by clients at runtime.

Serious Specification for Composing Components, ***Mike Barnett, Wolfgang Grieskamp, Clemens Kerer, Wolfram Schulte, Clemens Szyperski, Nikolai Tillmann and Arthur Watson***

We discuss the use of an industrial-strength specification language to specify component-level contracts for a product group within Microsoft. We outline how the specification language evolved to meet the needs of the component-based approach followed by that group. The specification language, AsmL, is executable which allows for testing to be done using runtime verification. Runtime verification dynamically monitors the behavior of a component to ensure that it conforms to its specification.

Run-Time Management of Feature Interactions, ***Yinghua Jia and Joanne M. Atlee***

There is a push to develop feature-rich applications as collections of interconnected feature modules. The problem is that these modules are conceived as independent features, but when strung together, they may interfere with each other because they modify the same shared data (e.g., two features may inconsistently update variables that are encapsulated in a third module). We are studying how to support modular feature development via a framework that interconnects features and that automatically detects and resolves feature interactions. In this paper, we propose a component model for coordinating features and we describe a prototype framework that implements this model.

Verifying the Correctness of Component-Based Applications that Support Business Processes, ***Remco M. Dijkman, Joao Paulo Andrade Almeida and Dick A.C. Quartel***

Developing applications that properly support the enterprise is a difficult task. Failing to perform this task results in applications that are not accepted by the end-users and that frustrate daily conduct of business. In this paper we introduce a formal yet practical method that helps to design component based applications that properly support the enterprise. The method can be used to verify whether the behavior of an application conforms to the behavior of the enterprise, where the behavior of the enterprise is specified in the form of business processes. The method helps to avoid applications being designed that support the enterprise in an incorrect manner.

Integrating Interface Assertion Checkers into Component Models, ***George T. Heineman***

Run-time enforcement of behavioral contracts has been studied extensively in procedural and object-oriented languages. This research has led to a better understanding of specific techniques, including pre-processing compilers or wrappers. However, component-based software engineering (CBSE) imposes additional restrictions and it is appropriate to consider how to extend these techniques when the software is decomposed into independently-developed third-party components. In this paper we identify some requirements for integrating run-time enforcement of behavioral contracts into the component model and illustrate a solution using a scaled-down component model and example. The primary result is that a standardized service should be added to component model implementations to enable application assemblers to enforce local properties as specified by the components in the application as well as global properties as specified by the application.

4.3 Session III: Analysis, Design and Patterns

Session three focused on how to design quality into component assemblies. The four papers presented during this session discussed mechanisms to support design and implementation of component-based systems for which certain guaranteed can be made about predicted assembly behavior. Mehlitz and Penix describe a system, D4V, that uses a combination of design patterns and component verification to select the sets of components that can be assembled to produce systems that meet specific quality attribute goals. Sridhar and Hallstrom propose that component containers be viewed as parameterized components in order to leverage existing support for reasoning about properties of parameterized components. Baresi et al. describe the use of known properties of architectural styles and graph transformations as a means for providing continued assurance of system quality when an assembly is reconfigured at run-time. Vecellio and Thomas extend component infrastructure to support policy enforcement mechanisms.

Design for Verification: Using Design Patterns to Build Reliable Systems, ***Peter C. Mehlitz and John J. Penix***

In commercial software development, components are mainly used to reduce time to market. While some effort has been spent on formal aspects of components, most of this was done in the context of integration into programming languages or operating system frameworks. As a consequence, increased reliability of composed systems is merely regarded as a side effect of a more rigid testing of pre-fabricated components. In contrast to this, Design for Verification (D4V) puts the focus on component-specific property guarantees, which are used to design systems with high reliability requirements. D4V components are domain specific design pattern instances with well-defined property guarantees and usage rules, which are suitable for automatic verification. The guaranteed properties are explicitly used to select components according to key system requirements. The D4V hypothesis is that the same general architecture and design principles leading to good modularity, extensibility and complexity/functionality ratio can be adapted to overcome some of the limitations of conventional reliability assurance methods, such as too large a state space or too many execution paths.

Generating Configurable Containers for Component-Based Software, ***Nigamanth Sridhar and Jason O. Hallstrom***

Existing container-based development strategies provide solutions to the problem of encapsulating cross-cutting concerns in component-based software systems. These approaches fall short, however, in enabling tractable reasoning. To extend existing work in reasoning about parameterized components to container-based approaches, we view containers as parameterized components. We present a model of component containers based on Service Facilities (Serfs) [18] — a design pattern framework that supports the construction of parameterized components that supports dynamic binding. To ease the transition to this new approach, we present the design of a tool that automatically generates Serf containers for existing component libraries.

Modeling and Analysis of Architectural Styles Based on Graph Transformation, ***Luciano Baresi, Reiko Heckel, Sebastian Thöne and Dániel Varró***

Modern architectural styles, like the service-oriented style underlying web services, are highly dynamic. This complicates not only their practical application, but also the modeling and prediction of their behavior. To account for this problem, we propose to model architectures as graphs, represented as instances of UML class diagrams, and to describe their reconfigurations by graph transformation rules. Based on a sample model for service-oriented architectures, we discuss what properties are interesting to be analyzed and how such analysis could be performed.

Infrastructure Support for Predictable Policy Enforcement, ***Gary Vecelli and William Thomas***

Component and service-based application infrastructures provide mechanisms for efficiently composing a system from a diverse collection of components and services. However, because of the lack of insight into the components and services within the application, integrating changes can be challenging. One class of change that we perceive as being both common and necessary is in the area of policy adherence (i.e., the constraints on a system's behavior that are imposed across the system). Unless the mechanisms that implement the policy are well isolated from the core application logic, any upgrade to the policy can have a ripple effect through the system. For systems that require robust certification, this ripple effect hampers the ability to rapidly deploy changes in policy. In this paper we highlight some patterns for separating policy adherence from application core logic, and discuss how these patterns can be mapped to commercially available infrastructures. By realizing these patterns as common infrastructure extensions, we allow applications to be developed in a manner consistent with the commercial infrastructure, provide the power of policy enforcement mechanisms to the system developers, and separate the policy enforcement logic from core application

4.4 Session IV: Compositionality Issues for Extra-Functional Properties

Both papers in this session discuss the need to provide languages and reasoning frameworks that support usage-context sensitive description of component properties. Hamlet et al. apply prior work, based on identification of usage subdomains for components, in predicting assembly reliability to assembly runtime prediction. Sitaraman et al. argue the need for more expressive languages for expressing properties and for mathematical models that support reasoning in an environment in which the component properties depend on their usage.

Experiments with composing component properties, ***Dick Hamlet, Milan Andric and Zheng Tu***

A detailed, microscopic theory of software component composition into systems was presented in this workshop in 2000 and subsequently at ICSE 2001. The essential idea of this theory is that by decomposing the input domain of a component into appropriate subdomains, its properties can be measured so that a system developer can later use the measurements, factoring in usage and system-structure information when such system-defined information is available. In principle, the theory could be the basis for a CAD tool supporting system design, which would take as input: (1) the black-box components, (2) their developers' subdomains and property measurements, and (3) a proposed system structure. The CAD tool could then calculate the system properties to be expected. This compositional theory was originally proposed for the reliability property, but it was soon recognized that it applies to almost any component/system property that is input- and structure-dependent. In particular, the run-time property is an ideal one for experimentation, because it is easier to measure than reliability, and does not depend on the somewhat dubious background theory of software reliability. While reliability measurements require random testing, run time measurements can be made systematically, since there is no issue of failure correlation. Thus run times can be measured with fewer evenly distributed test points. We report on initial validation experiments for this theory, using a rudimentary CAD tool that does calculations of run times. These experiments address the basic validity of the theory and the efficiency of the system-design calculations.

Expressiveness Issues in Compositional Performance Reasoning, ***Bruce W. Weide, William F. Ogden and Murali Sitaraman***

Compositional reasoning about any behavioral property of a system depends, first, on the ability to express that property for both individual components and systems constructed from them. Expressiveness problems arise when considering compositional reasoning about performance in the presence of complex user-defined types (as opposed to simpler built-in types). There are interesting implications not just for compositional reasoning but for language design and for formal specification.

4.5 Session V: Generative Modeling and Synthesis

In the fifth and final presentation session of the workshop three papers describing technologies that support system generation and adaptation were presented. Cervantes and Hall described a system that is capable of monitoring assembly behavior and managing service dependencies, Inverardi and Tivoli describe connector synthesis to support assembly evolution, and Zhao et al. generate implementations from architectures using components.

Automating Service Dependency Management in a Service-Oriented Component Model, ***Humberto Cervantes and Richard S. Hall***

This paper describes a mechanism to automate service dependency management in a service-oriented component model. The impetus behind this mechanism is not merely to eliminate complex and error-prone code from component-based applications, but also to deal with the phenomena of application building blocks that exhibit dynamic availability, i.e., they may appear or disappear at any time and this is not under the control of the application. This intense focus on dynamic availability of building blocks is the result of the belief that applications of the future will become context aware in order to deal with building block proliferation. Such applications will employ context-aware architectures that use context (e.g., location, environment, user task) as a filter for including/excluding building blocks in/from their compositions. In this vision, automatic handling of dynamically available building blocks and their impact on application composition is critical. The service dependency management mechanism described in this paper is a starting point for such research and is implemented on top of the Open Services Gateway Initiative (OSGi) framework. The concepts and solutions it provides are sufficiently general for application in other service-oriented component models.

A compositional synthesis of failure-free connectors for correct components assembly, ***Paola Inverardi and Massimo Tivoli***

Correct automatic assembly of software components is considered an important issue of CBSE (Component-Based Software Engineering). It is related to the ability to establish properties on the assembly code by only assuming a relative knowledge of the single components properties. In our precedent works, we have provided our answer to this problem by discussing a software architecture based approach in which the software architecture imposed on the assembly allows for detection and recovery of COTS (Commercial-Off-The-Shelf) integration anomalies. One of the crucial aspects of our assembly technique is related to the ability to synthesize a specification-satisfying assembly code (i.e. the failures-free connector) in such a way that the synthesis results compositional with respect to system evolutions. That is every time the system evolves, in order to automatically synthesize the failures-free connector for the new version of the specification-satisfying system it is enough to repeat the synthesis only for the part of the system related to its evolution.

A Generative and Model Driven Framework for Automated Software Product Generation, ***Wei Zhao, Barrett R. Bryant, Jeffrey G. Gray, Carol C. Burt, Rajeev R. Raje, Andrew M. Olson and Mikhail Auguston***

Component-based Software Engineering (CBSE) and related technologies have demonstrated their strength in recent years by increasing development productivity and parts reuse. Recently, the Model Driven Architecture (MDA) has raised the abstraction level of programming languages to modeling languages that can be compiled by downward model transformations. Correspondingly, the goal of Generative Programming (GP) is to automate concrete software product generation from a domain-specification and reusable components. This paper describes the UniFrame framework, which is built on the foundation of CBSE while leveraging the capabilities offered by MDA and GP. UniFrame provides theories and implementation for steps of model transformations for a concrete software product based on domain development in various Generative Domain Models.

5 Working Sessions

There were two working sessions during the afternoon of the second day of the workshop, which provided a forum for in depth discussion of issues that had been raised during the paper sessions. These issues covered a wide range of topics related to trusted components, runtime verification, container generation, component generation, and the general state of research in the area. During the first working session a top ten list of questions was created and then reduced to three broad issues that were subject of discussion during the final working session.

1. Abstraction versus encapsulation

2. Trusted predictions—how does abstraction affect our ability to predict properties?

3. Why do we believe that a “compositional theory of ‘X’ is possible (X denotes different properties)? And do we need a theory in order to produce useful predictions?
   a. Does it matter what ‘X’ is? Are some properties more compositional than others?
   b. Are there a standard set of component properties (component “measures of merit”) for each X or for all X ?

4. How is the prediction of system properties from components different in CB/not-CB way?

5. Can the specified/certified values of component properties be acquired independent of a specific use context?

6. How do we “measure” component properties (logical and empirical) is traditional software measurement theory adequate? e.g;. GQM

7. The perennial question: what is a component, what is component- based?

8. Do we understand (what is) the relationship between system, “architecture” and “components”
   a. Where is our abstraction level for components? and
   b. Where does composition occur – at which “level” and is it different in kind at different levels, and what are its mechanisms?

9. Are there common composition principles across levels?

10. What services do component models “add” to components, at runtime, and why isn’t it overt in our takes?

This list was “boiled down” into three broad issues to which the remainder of the working sessions was devoted. During each session there was lively discussion and sometimes heated debate, each resulting in a new list of questions that provide fertile ground for future writing and research in the area. The three topics were:

1. Is it possible to develop a composition theory for some assembly property?

Issues have been raised at this workshop and past workshops that lead one to the ability to predict assembly reliability and performance: Can we define them sufficiently to define composition operators? Are some properties inherently non-compositional while others are? We recognize the need for compositionality for analyzing systems that are open and dynamically growing but perhaps, if we can close the system, then we can do global analysis, and scale through abstraction. This led to the identification of the need to produce or find a technical definition of compositionality upon which the community could agree. This was left as an open issue.

2. What is the relationship among components, architecture, and system?

Software architecture papers clearly define the world in terms of component and connectors, Acme is the meta language of ADLs and looks like a language for describing assemblies of components. Objects/Classes are also clearly defined – and are not components. Architecture is the compositional bridge between the desired properties of a system and the empirical properties of components, and component implementations. Questions arose as to why we are introducing levels of abstraction t? If there are a variety of levels of abstraction, at what level do components live? Where does composition occur – at which "level" and is it different in kind at different levels, and what are its mechanisms? Are there common composition principles across levels? What characteristics of an architecture are required for composability? Different types of components exist at different levels of the hierarchy. Component is of itself 'empty' without the modifier 'software,' 'architectural', 'performance spec,'.

3. How is system property prediction affected by the black-box nature of components? Or put another way, Can the specified/certified values of component properties be acquired independent of a specific use context?

Dick Hamlet pointed out that components are inherently input dependent; if they are composed they acquire composition-specific dependencies; how can we reason about all the possibilities? It was suggested that invariants imposed by a component model provide bounds on assemblies that will result in assemblies that are 'well formed' about which reasoning is possible. It was suggested that defining a closure on the effects a component behavior can have (impact radius) might produce benefits. The need for more expressive notations and tools was agreed. All non-trivial composition must respect its context, because composition will take place in multiple contexts; therefore, the context must be made explicit. On an operational level, users want a description in a CAD style tool, with parts (boxes) and links among them, this is reminiscent of the Acme development environment and resurfaces the fact that there is a duality between sw architecture – how things can be plugged, their topologies, etc – and how we define components, when a software component is a component in an architecture. This last observation led to the final question of whether 'wiring' is the same as 'composition'?

6 Workshop Results and Future Plan

Participants left the workshop stimulated with new ideas about issues that require attention if predictable assembly is to become a reality. Some old questions were closed, some new ones were raised to add to the only slightly smaller list that existed at the opening of the workshop. It was determined that the system must be designed with reasoning about specific properties in mind and that is important to consider the type of component (specification, COTS, etc.) before reasoning frameworks can be developed. New questions that arose during the workshop include whether we can define the norms for component models that support predictability? Is reflection antithetical to predictable component models? And what are requirements to compositionality in the face of feature-oriented extension such as services provided by component containers? Is feature-oriented extension the "way to go" to attain predictability?

While the workshop opened more issues than it closed there were many interesting points to take away about progress in research and practice in component-based development. We have seen that the notion of component trust is finding its way into industry, the use of design rules has gained the interest of researchers, support for reasoning about performance issues his improved, primary objectives of CBSE still: reuse of components?, productivity, ease of maintenance, the notion of connector is fundamental to the work of this community, and parameterized component models are receiving increasing interest.

Not only did the workshop show us that many positive strides have been made during the last twelve months but also that certain topics that we expect to come up less often in the future including the question of 'how do we define the term component? We expect to see fewer attempts to create generic approaches to reasoning about components and assemblies.

6.1 Publication of Results

The proceedings of the workshop are available on the web at both the Software Engineering Institute [2] and Monash University [3].

6.2 Future Plans

The workshop was deemed a great success and suggestions for future directions were discussed. It was suggested that each workshop should begin with a mini-workshop on what has been accomplished and agreed upon in the past so that new attendees can be brought up to speed and veterans will be reminded. It was decided to that the topic is attracting sufficient attention that we should consider a larger event for 2004, perhaps a federated event with other related workshops. It was also suggested and agreed that the status of the proceedings be improved, perhaps by publishing through the IEEE digital library or perhaps having longer versions of papers published in a special proceedings such as LNCS.

7 Acknowledgement

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

The program committee:
Judith Bishop, University of Pretoria, South Africa
Jan Bosch, University of Groningen, The Netherlands
Ivica Crnkovic, Mälardalen University, Sweden
Jacky Estublier, LSR-IMAG, France
Kathi Fisler, WPI, USA
Dimitra Giannakopoulou, NASA Ames, USA
Richard Hall, Imag/Lsr, France
Dick Hamlet, Portland State University, USA
George Heineman, WPI, USA
Paola Inverardi, University of L'Aquila, Italy
Shriram Krishnamurthi, Brown University, USA
Otto Preiss, ABB/CRC, Switzerland
Heinz Schmidt, Monash University, Australia
Judith Stafford, Tufts University, USA
Clemens Szyperski, Microsoft Research, USA
Kurt Wallnau, Software Engineering Institute, Carnegie Mellon University, USA

Dave Wile, Teknowledge, Corp., USA
Christian Zeidler, ABB Research, Germany

8 References

[1] I. Crnkovic, H. Schmidt, J. Stafford, K. Wallnau, 5th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction, Software Engineering Notes, 2001. Nov

[2] http://www.sei.cmu.edu/pacc/CBSE6

[3] http://www.csse.monash.edu.au/~hws/cgi-bin/CBSE6/Proceedings/proceedings.cgi

[4] http://www.sei.cmu.edu/pacc/CBSE5/CBSE5-Proceedings.html

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

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