NPS-CS-20-001

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THE MONTEREY WORKSHOP SERIES (1992-2020)

Luqi

May 2020

Prepared for: Naval Postgraduate School Research Sponsored Programs Office.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From-To)*
05-31-2020	Final Report	1992-2020

4. TITLE AND SUBTITLE

The Monterey Workshop Series (1992-2020)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Luqi, Professor of Computer Science &
Cyber Systems and Operations

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AND ADDRESS(ES)
Naval Postgraduate School
1411 Cunningham Road Bldg 305
Monterey, CA 94943

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-CS-20-001

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
The views presented in this report are those of the authors and do not necessarily represent the views of DoD or its components.

14. ABSTRACT
This report describes the history of the Monterey Workshop series from 1992 to 2020. The objective of the Monterey Workshop series since its inception has been to increase the practical impact of scientific methods in computer science and engineering. This report also describes the creation of the new Monterey workshop website and planning for the 21st Monterey Workshop.

15. SUBJECT TERMS
Cyber, Security, Unmanned Systems, Architectures, Hardware, Software, Machine Learning, Insider Threats, Cyber Education, Command, Control, Communication, Computer, Intelligence

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Luqi
UNCLASS	UNCLASS	UNCLASS		126	**19b. TELEPHONE NUMBER** ***(include area code)*** 831-656-2735

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Ann E. Rondeau	Robert Dell
President	Acting Provost

The report entitled "The Monterey Workshop Series (1992-2020)" was prepared for and supported by the Naval Postgraduate School Research Sponsored Programs Office.

Further distribution of all or part of this report is authorized.

This report was prepared by:

_____
Luqi, Professor
Computer Science & Cyber Systems and Operations

Reviewed by:

_____
Peter Denning, Chairman
Department of Computer Science

Released by:

_____
Jeffrey D. Paduan
Dean of Research

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The Monterey Workshops have been an influential forum for aligning scientific research and innovations among academia, industry, and government agencies since 1992. The objective of the Monterey Workshop series since then has been to increase the practical impact of scientific methods in computer science and engineering. Many of the world's leading researchers have participated, and topics have included computer science, software and system engineering requirements and tools, innovation on systems, big data and cyber, and many other technical disciplines. The Workshops seek to improve software practice via the application of engineering theory and to encourage foundational scientific results using formal methods and sound system models.

The workshop series has developed productive research directions that have been adopted by various sponsors, advanced the capabilities of various researchers, and established professional and personal links between the worlds' leading researchers and academics. While much has been accomplished by the workshops outlined in this report, this effort will have to be continued to ensure that research and academia are aligned with the needs of industry and government stakeholders. Material from various past proceedings and websites have been collected into a new website hosted by NPS. Additionally, planning for the 21st Monterey Workshop is underway, and this upcoming workshop will focus on intelligent systems, machine learning, and artificial intelligence.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Figure 1 - Monterey, CA

The objective of the Monterey Workshop series since its inception has been to increase the practical impact of scientific methods in computer science and engineering. Many of the world's leading researchers have participated, and topics have included computer science, software and system engineering requirements and tools, innovation on systems, big data and cyber, and many other technical disciplines. The Workshop seeks to improve software practice via the application of engineering theory and to encourage foundational scientific results using formal methods and sound system models.

The Monterey Workshops have been an influential forum for aligning scientific research and innovations among academia, industry, and government agencies since 1992. The Workshops have developed productive research directions that have been adopted by various sponsors, and advanced the capabilities of various researchers. The titles of the past Monterey Workshops are:

0th 1992 Concurrent and Real-Time Systems [1]
1st 1993 Software Slicing, Merging and Integration [2]
2nd 1994 Software Evolution [3]
3rd 1995 Specification-Based Software Architecture [4]
4th 1996 CAPSTAG – Computer Aided Prototyping [5]
5th 1997 Requirements Targeting Software and Systems Engineering [6]
6th 1998 Engineering Automation for Computer Based Systems [7]
7th 2000 Modeling Software System Structures in a Fastly Moving Scenario [8]
8th 2001 Engineering Automation for Software Intensive Systems Integration [9]
9th 2002 Radical Innovations of Software and Systems Engineering in the Future [10]
10th 2003 Software Engineering for Embedded Systems: From Requirements to Implementation [11]
11th 2004 Software Engineering Tools: Compatibility and Integration [12]
12th 2005 Networked Systems: Realization of Reliable Systems on Top of Unreliable Networked Platforms [13]

13th 2006 Composition of Embedded Systems: Scientific and Industrial Issues [14]
14th 2007 Workshop on Innovations for Requirement Analysis: From Stakeholders Needs to Formal Designs [15]
15th 2008 Foundations of Computer Software, Future Trends and Techniques for Development [16]
16th 2010 Modeling, Development and Verification of Adaptive Computer Systems: the Grand Challenge for Robust Software [17]
17th 2012 Development, Operation and Management of Large-Scale Complex IT Systems [18]
18th 2016/Feb. Integrity of Industrial Control System & Future Command & Control [19]
19th 2016/Oct. Challenges and Opportunity with Big Data [20]
20th 2018 20th Monterey Workshop on Cyber [21]

The Monterey Workshops have always focused on areas at the edge of the state of the art with potential for improvements that will shift the entire paradigm. Suggestions regarding the most important next step forward are always welcome. For further information on the workshop series, see the Monterey Workshop websites [22] [23].

This report summarizes the topics covered and findings of each Monterey Workshop in the series, based on currently available information. The early workshops in the series predate widespread use of the Internet, and online records are limited.

II. WORKSHOP SUMMARIES

A. 0TH MONTEREY WORKSHOP: CONCURRENT AND REAL-TIME SYSTEMS (1992)

A physical copy of the proceedings of the initial workshop in the series is held by the Office of Naval Research.

B. 1ST MONTEREY WORKSHOP: SOFTWARE SLICING, MERGING AND INTEGRATION (1993)

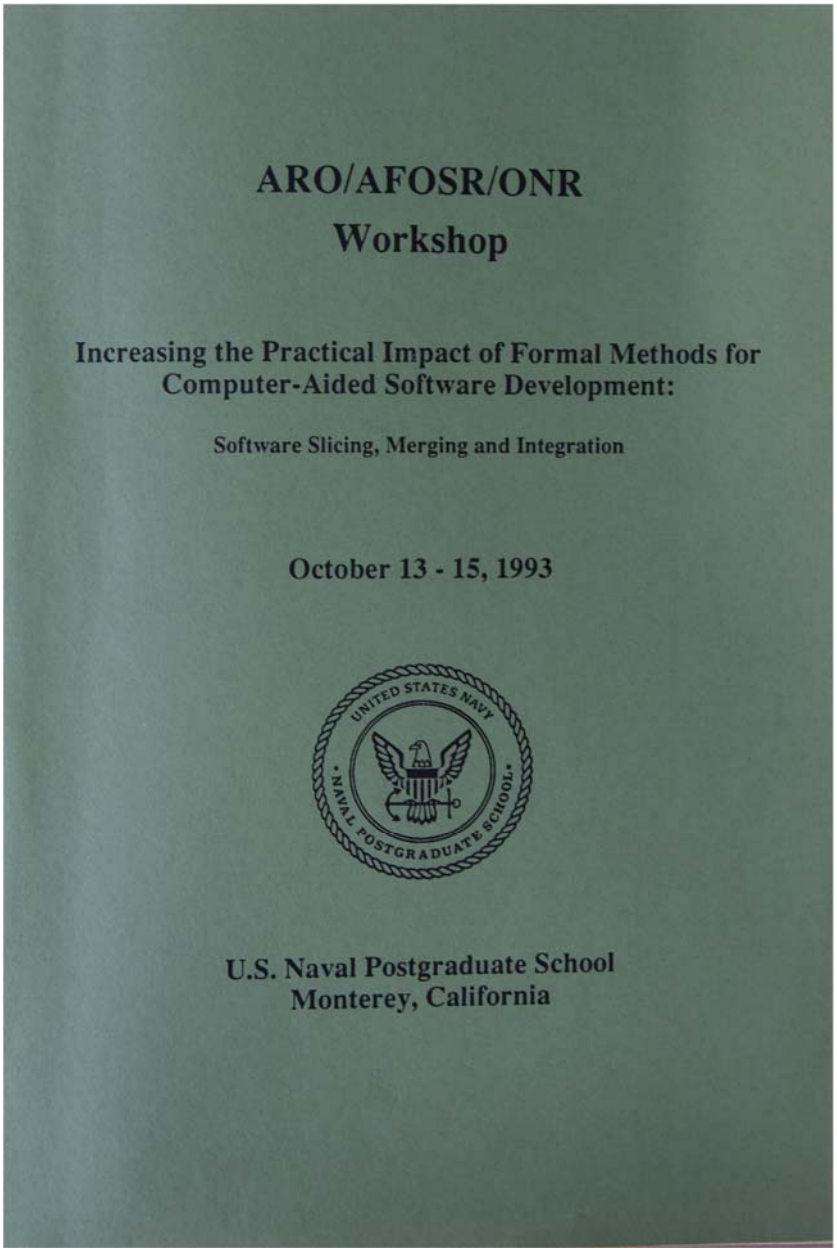


Figure 2 – Cover Page, 1st Monterey Workshop

Preface

1993 ARO/AFOSR/ONR Workshop on Increasing the Impact of Formal Methods for Computer-Aided Software Engineering

by

Valdis Berzins

Computer Science Department, U.S. Naval Postgraduate School, Monterey, California

The U.S. spends billions of dollars per year on software, much of it for software modifications and maintenance. Computer aid should give software designers better control over their products, with resulting improvements in software usefulness and reliability and reductions in time and cost for large scale changes. Our basic premise is that appropriate formal methods supported by appropriate software tools can be very beneficial for practical software development. We believe that it is possible and necessary to validate this premise and to put it into common practice. As part of this effort, we would like to ask for your help to establish and support an important research direction, computer aided software evolution.

1. Computer Aided Software Evolution

Aspects of computer aided software evolution have been studied for many years but the field has not yet gained wide recognition and support, partially because the capabilities of existing theoretical solutions are completely dwarfed by the complex and demanding needs of real software development projects. Much early work in the area addressed relevant but simplified subproblems, ignoring significant parts of real practical problems. These partial solutions have not been integrated to solve the real problem because the long term goals in this area have not been not well understood by researchers and funding agencies.

This is not the case in other disciplines such as operations research, physics, and applied mathematics. The difference is that we have not educated ourselves or the funding agencies about the connections among the different narrow areas of research, and the relationships to pressing national problems that span many of those areas. Everyone is spread thin. There are many aspects to the subject of software evolution and it is not easy for any single individual to recognize all of the connections among the different aspects. We need a forum to bring us together and strengthen these connections. We are very happy that funding agencies and researchers are willing to listen, even if they are primarily concerned with (apparently) different areas.

For computer support to provide enough practical benefit to justify the necessary investment in tools and training, we need a better theoretical understanding of software modification and effective formal methods for solving various subproblems associated with software evolution. We also need an effective framework for integrating these methods and their associated tools, and we need to validate our theoretical models against real software development efforts to make sure we are addressing the right questions.

Problems and solution procedures must be well understood before they can be successfully automated. Much effort has been spent on building big software tools with nice interactive human interfaces but with very little real power inside the wrappings. This has led to an increasingly widespread perception that investment in computer tools for software development does not pay off. Such a practice is wasteful on a large scale, and the perception it creates is very dangerous because it will block all hope of real progress if it is left unchecked.

Our field of study should develop the theoretical understanding and algorithmic solutions that will enable us to put powerful engines inside the pretty interfaces. It is our responsibility to make the capabilities of these new engines match the most urgent practical needs in software development and evolution.

Figure 3 - Page 1, 1st Monterey Workshop

2. Workshop Goals and Procedures

The goals of this workshop are to:

(1) Identify the key technical problems in computer aided software evolution.

(2) Clarify the relations to other parts of Computer Science.

(3) Examine plausible technical approaches and assess advantages and disadvantages.

(4) Identify directions for future work that can have a practical impact.

To provide the background for the workshop, we briefly survey some previous work and indicate its relevance to computer aided software evolution. This introduction will be followed by presentations on different aspects of the subject, interleaved with discussions to bring out implicit assumptions, clarify relationships between different points of view, and make assessments. The conclusions of each session will be summarized by the reporter for the session, and the results will be integrated into a workshop report.

We have a great deal to learn from each other, and this workshop can only start the process. We should get together again next year to assess progress and to take the next steps towards practical application of formal methods in software development.

3. Survey of Technical Problems

Software evolution has both global and local problems. The global problems concern modifying software systems with many variants and coordinating concurrent changes to the same system. The local problems concern a single engineer working on a single change. This workshop is focused on the global problems to keep the scope manageable. We hope that the local problems can be addressed in a future workshop.

To support the process of combining (and recombining) efforts of different people, we are interested in formal methods and algorithms for checking whether a set of changes is compatible, for combining a set of compatible changes, and for reconciling incompatible changes. A related issue is coordinating a set of changes being developed concurrently by a team of engineers. This raises problems of preventing inconsistencies between concurrent changes or detecting and reconciling inconsistencies, based on the partial and uncertain information available while a change is in progress.

The basic assumption of change merging is that the behavior of a system can be separated into a set of independent parts that can be recombined. Two changes are compatible (can be consistently combined) if the parts of the system behavior affected by each change are independent. A change affects a part of system behavior if the original version of the part is not equivalent to the changed version of that part. Behavior can be considered at different levels: computation traces, functions computed by programs, requirements satisfied by program behavior, etc. Each of these levels is associated with different notions of part, independence and equivalence.

3.1. Software Slicing

A program slice is a subset of a program whose computation trace is independent of the rest of the program[1, 12]. A slice includes the parts of a program that can influence the part of its behavior visible from a particular point of view, such as the value of a set of variables or output streams. Slices can thus be "independent parts" supporting change merging at the level of computation traces.

Previous approaches to slicing have mostly been based on data flow analysis of single-thread imperative programs. Some current problems include increasing the resolution of representations and methods for computing slices, developing slicing methods for wider classes of programming languages, and developing analogs at the specification and requirements levels. More detailed semantic models of program slicing can have the advantage of better resolution and the disadvantages of longer running times and possible divergence because exact slicing is not a computable function.

Figure 4 - Page 2, 1st Monterey Workshop

3.2. Software Change Merging

Software change merging is the process of combining modifications to software system behavior. We need methods that guarantee semantically correct results in all cases where they do not report conflicts. Some of the problems in this area are finding better models of changes to software, safe merging methods with fewer spurious conflict reports, methods for automatically resolving conflicts, accommodating changes to data types, module interfaces, optimizing changes that introduce different algorithms, proving safety of change merging procedures, etc.

One approach to semantically based change merging for programs is based on slicing [8]. Slicing has the advantage of efficiency and the disadvantages of being unable to combine changes that can reach the same output or changes that use different algorithms to compute the same function.

Another approach to change merging is based on meaning functions [4]. Meaning functions are the functions computed by programs. The images of individual input values can be considered to be independent parts, leading to high resolution change merging methods. Meaning functions can merge changes to the same output if both changes can not take effect for the same input and the same initial state, and they can accommodate algorithm changes such as speedup transformations. Meaning functions introduce the possibility of recognizing some equivalences between programs denoting different execution sequences that compute the same function. Methods based on meaning functions can involve large amounts of computation and can fail to terminate if not suitably constrained, because exact solutions to these problems are also not computable.

Change merging has also been investigated for specifications and requirements [5], although this work has not yet been carried down to the code level. Responses to different stimuli can be considered to be different components of a system's behavior, and the response to each stimulus can be specified by postconditions. This raises large scale issues such as merging changes to the set of stimuli recognized by a system, provides a means for modeling possible dependencies between responses to distinct system inputs, and introduces a looser interpretation for the equivalence aspect of change merging. In the context of meaning functions, two outputs are considered equivalent only if they are equal, because there is no weaker criterion for equivalence that is safe. In the context of specifications that are not completely tight, two distinct output values can be equivalent if they both satisfy the same postcondition.

3.3. Representing Software Design Decisions

Improved techniques for representing and reasoning about software design decisions are important for supporting change merging as well as for providing intelligent assistance for software development. The plan calculus was introduced in the programmer's apprentice project to help the system analyze the programmer's design rationale and to fill in implied details of partial designs [11]. Similar approaches are relevant to advances in change merging because more accurate techniques depend on the relations between programs and meaning functions or specifications.

Improved change merging methods will have to process specifications or meaning functions to combine changes, and then to transform these changes back into code. Although the general problem of synthesizing code from specifications is not likely to be solved soon, the type of code synthesis required for change merging is easier because pieces of code realizing all the parts of each version are already available. The code synthesis required is to properly recombine these parts, possibly with some adaptations of details. This process must ensure that putting the parts together in new ways still results in valid design justifications, which will require some reasoning support. Although this is a big problem, the process does not have to be "creative". In particular, it can be aided by knowledge of common patterns of program design, such as those captured via the plan calculus and a library of cliches.

Figure 5 - Page 3, 1st Monterey Workshop

3.4. Transformations

Monotonic transformations are another path to change merging. Monotonic transformations produce results that are compatible with the starting point, but which may be further constrained [6]. Thus these transformations preserve meaning, and may add refinements[2]. Monotonic transformations are relevant to change merging because a series of such transformations is one way to represent the dependence of lower level information on higher level information.

Past work on transformations has considered changes with a desire to replay the derivation of an implementation after a change to some of the earlier decisions. This capability is relevant to the problem of turning a specification change into the corresponding program change, which is one aspect of change merging. Direct consideration of the change merging problem in the context of transformations may also be fruitful: given a base version and two enhanced versions of a transformational implementation, can we determine which parts of the derivations correspond to the enhancements, and automatically construct a derivation that incorporated both enhancements.

4. Conclusions

Change merging is an attractive context for research and development related to computer-aided software construction because

(1) it is easier than unrestricted code synthesis, and

(2) much of the effort in software development is spent on modifications.

We have sketched some opportunities in this area and hope that the workshop will clarify the problems, identify new relationships to existing work, and suggest additional directions for progress.

5. Overview of the Position Papers

The position papers for the workshop fall into five main areas: software maintenance and evolution, software specification methods, software merging, slicing, and restructuring, software verification, testing, and synthesis.

The papers on software evolution focus on computer assistance. Luqi and Goguen describe some directions for making progress in software development using formal methods. Mittermeir and Kienzl discuss the use of intra-object schemas to automate certain kinds of changes to object classes. Srinivas and Smith propose a model of evolution in which an aspect of a system is changed and then the change is automatically propagated by constructing a minimal set of other changes that restore consistency.

The papers on specifications discuss the construction and evolution of specifications as well as some of the processes that can be supported by specifications. Feather surveys some efforts at incremental development of specifications. Mili and Mili describe a lattice structure for specifications that can support checking whether specifications of different aspects of a system can be consistently combined and materializing the combination if one exists. Berztiss evaluates the applicability of formal methods to different kinds of software systems, and in a companion paper describes an approach for re-engineering organizations by systematically identifying activities that can be incrementally automated. Shaw explores the adequacy of communicating real-time state machines for specifying large real-time systems. Stemple proposes an approach for extracting specification information from operational prototypes via derivation and proof techniques.

The papers on slicing and merging examine different aspects of program slicing related to automated synthesis and analysis of programs. Agrawal examines how to determine which gotos to include in a program slice. Dampier and Berzins examine slicing and change merging for a prototyping language with concurrency and real-time constraints. Griswold examines the connection between slicing and program

Figure 6 - Page 4, 1st Monterey Workshop

restructuring. Huang examines the application of path decomposition, program slicing, and symbolic execution to program understanding and simplification. Sterling describes a method for merging simple PROLOG programs to automatically construct a complex one.

The papers on software verification, testing and synthesis examine various ways to use formal approaches to achieve reliable software. Antoy and Hamlet describe a method for computing test inputs that force a program down a specified path using the narrowing technique for solving symbolic equations. Cleaveland examines the utility of finite-state approaches to software verification. Kapur presents some recent advances in automated reasoning technology and assesses the potential for application to software development. Salasin and Waugh propose annotation of software architectures with obligations to satisfy non-functional requirements as a way to assess whether or not those requirements will be met in a complex system. Smith describes the transformational development of a class of transportation scheduling algorithms.

1. H. Agrawal and J. Horgan, "Dynamic Program Slicing", *SIGPLAN Notices 25*, 6 (June 1990), 246-256.
2. V. Berzins, "On Merging Software Extensions", *Acta Informatica 23*, Fasc. 6 (Nov. 1986), 607-619.
3. V. Berzins, "Software Merge: Semantics of Combining Changes to Programs", Technical Report NPS 52-91-4, Computer Science Department, Naval Postgraduate School, 1990. Revised for ACM Transactions on Programming Languages and Systems.
4. V. Berzins, "Software Merge: Models and Methods", *International Journal on Systems Integration 1*, 2 (Aug. 1991), 121-141.
5. V. Berzins and Luqi, *Software Engineering with Abstractions*, Addison-Wesley, 1991.
6. V. Berzins, Luqi and A. Yehudai, "Using Transformations in Specification-Based Prototyping ', *IEEE Transactions on Software Engineering* 19, 5 (May 1993), 436-452.
7. D. Dampier, Luqi and V. Berzins, "Automated Merging of Software Prototypes", *Journal of Systems Integration*, to appear.
8. S. Horwitz, J. Prins and T. Reps, "Integrating Non-Interfering Versions of Programs", *Transactions Programming Languages and Systems 11*, 3 (July 1989), 345-387.
9. G. Ramalingam and T. Reps, "A Theory of Program Modifications", *Proceedings of the Colloquium on Combining Paradigms for Software Development*, vol. LNCS 494, Springer-Verlag, Apr. 1991, 137-152.
10. T. Reps, "Algebraic Properties of Program Integration", *Science of Computer Programming 17*, 1-3 (Dec. 1991), 139-215.
11. C. Rich and R. Waters, "Knowledge Intensive Software Tools", *IEEE Transactions on Knowledge and Data Engineering 4*, 5 (Oct. 1992), 424-430.
12. M. Weiser, "Program Slicing", *IEEE Transactions on Software Engineering SE-10*, 4 (July 1984), 352-357.

Figure 7 - Page 5, 1st Monterey Workshop

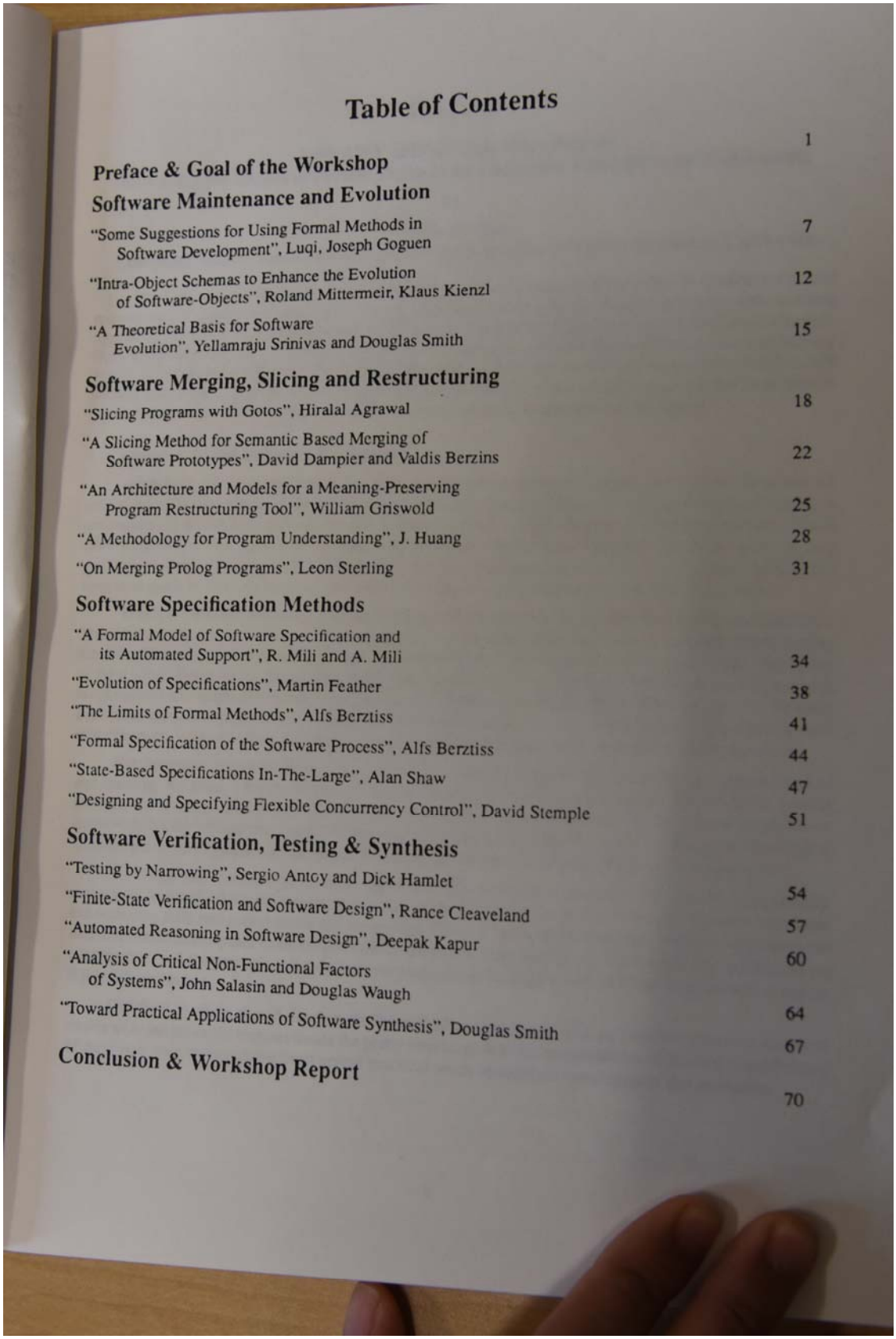


Table of Contents

Figure 8 - Table of Contents, 1st Monterey Workshop

C. 2ND MONTEREY WORKSHOP: SOFTWARE EVOLUTION (1994)

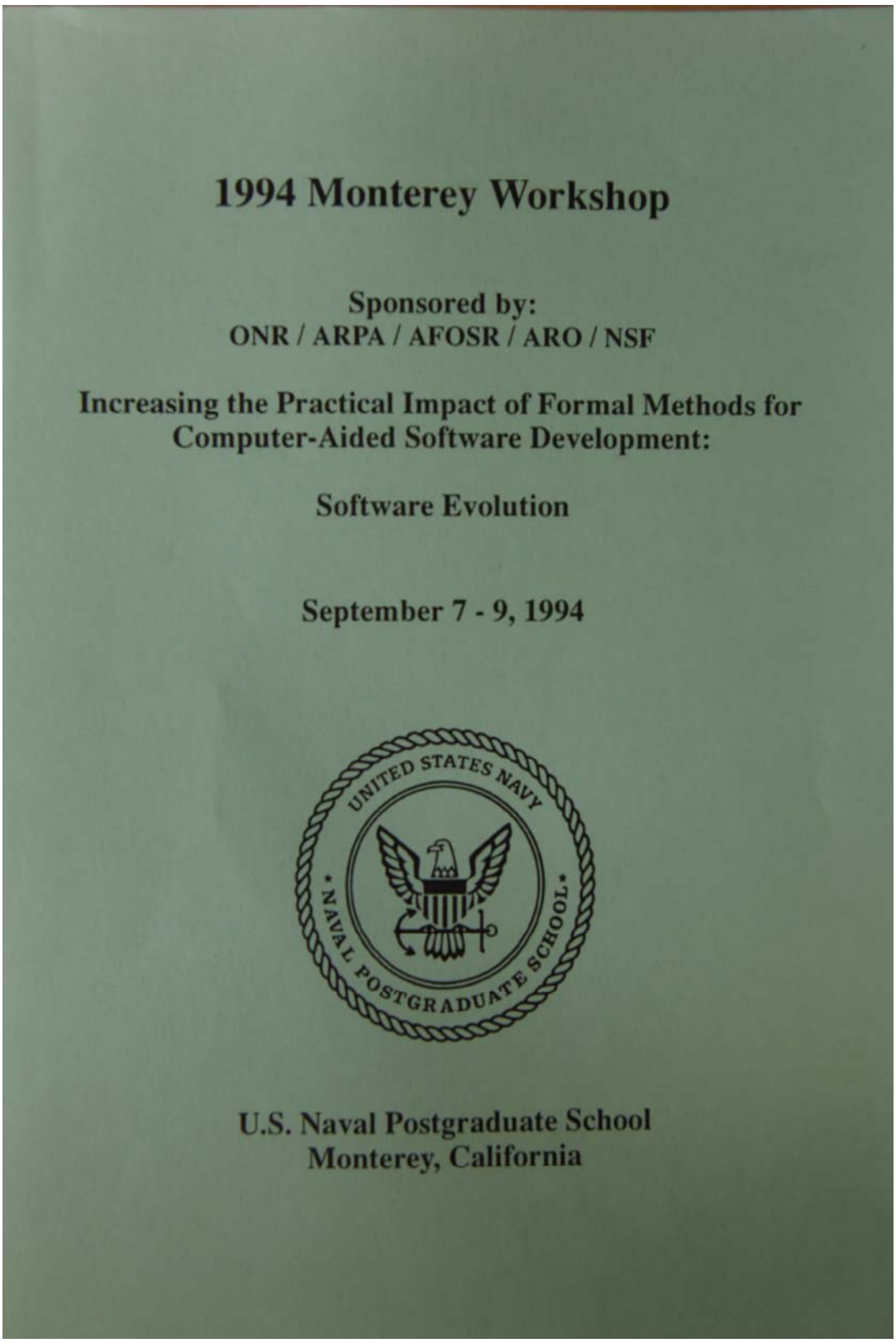


Figure 9 - Cover Page, 2nd Monterey Workshop

Monterey Workshop '94: Software Evolution — Increasing the Practical Impact of Formal Methods for Computer-Aided Software Development

Luqi
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

1. Software and Formal Methods

We need effective and reliable ways to develop software systems that meet user needs. This will have a great impact on society; for example, DoD spends billions of dollars each year on software, but many software systems in use do not satisfy users' needs. An article entitled "Software Chronic Crisis" by W. Gibbs in the September 1994 issue of *Scientific American*, states "despite 50 years of progress, the software industry remains years — perhaps decades – short of the mature engineering discipline needed to meet the demands of an information-age society".

To remedy this situation, formal software models that can be mechanically processed can provide a sound basis for building and integrating tools that produce software faster, cheaper, and more reliably. Formal methods can also increase automation and decrease inconsistency in software development. For this purpose, researchers in formal methods need to better understand where the developers of large software systems need help. Also software developers need to better understand the benefits of using formal models to construct tools that can solve practical problems.

This year's workshop focuses on software evolution. This refers to all the activities that change a software system, as well as the relationships between those activities. This includes responses to requirements changes, improvements to performance or clarity, repairs of bugs, and the overall organization of the development process. Evolution is not just another name for maintenance; evolution occurs throughout the life cycle, and includes specification-based development, incremental/phased development, requirements prototyping, version and configuration control, on-line documentation, testing, code generation, etc. It captures the dynamic aspects of software development.

This workshop helps clarify what good formal methods are and what are their limits. According to Webster's Dictionary, *formal* means definite, orderly, and methodical; it does not necessarily entail logic or proofs or correctness. Everything that computers do is formal in the sense that syntactic structures are manipulated according to definite rules. Formal methods are syntactic in essence and semantic in purpose. Given the motivations of the workshop, we believe this is the most appropriate sense for the word "formal" in the phrase "formal methods."

The prototypical example of a formal system is first order logic. This system encodes first order model theory with certain formal rules of deduction that are provably sound and complete. Unfortunately, theorem provers for this system can be difficult to work with. Formal systems can also capture higher levels of meaning, e.g., for expressing requirements, but these systems are harder to work with and have fewer nice properties. Fine-grained

Figure 10 - Page 1, 2nd Monterey Workshop

formal methods have a mathematical basis at the level of individual statements and small programs, but rapidly hit a complexity barrier when programs get large. Formal support for large-grain aspects of software development is necessary to extend research results to large problems.

Software research cannot rely solely on the weak validation of proving theorems and checking logical consistency. The value of a contribution has to be measured by its impact on practical software development, rather than on how well it fits with existing mathematics or currently dominant trends in theoretical computer science. At present, we have no accurate formal models of software evolution and its relation to internal software structures. Existing models either focus on narrow aspects of the process or cover the entire life cycle through informal approaches that cannot be automated.

Formal methods can be very effective if the right method is applied to the right problem. The excessive optimism that everything important is provable helps to explain the excessive pessimism that nothing important is provable. Formal methods are often considered useful for proving that programs satisfy certain mathematical properties, but are also often considered too expensive to be practical. This view is narrow and limited because formal methods can reduce time to market, provide better documentation, improve communication, facilitate maintenance, and organize activities throughout the life cycle. We seek to clarify the conditions under which these benefits can best be realized.

Although there are formal models for several aspects of the software development process, each aspect has been considered in isolation, and effects that span the entire process are missing from the models. Such global dependencies are significant in software development, and the difficulty of maintaining these dependencies as software evolves is a limiting factor for large systems. We need to better understand the interactions among different parts of the process, and develop compatible models so that solutions to different parts of the problem can be combined into systems that span the entire process.

2. Increasing the Practical Impact of Formal Methods

Formal methods and tools do not yet adequately support the development of large and complex systems. The Monterey Workshop has sought to make progress by focusing on a specific topic related to formal methods each year. The first (1992) Monterey Workshop focused on real-time and concurrent systems. The second (1993) Monterey Workshop focused on software slicing and merging. This year's Workshop focuses on software evolution because it is important, even though the subject is difficult and not well explored.

This workshop assesses the practical impact of formal methods and tools, identifies gaps between the capabilities of formal methods and the practical needs of software development, and defines appropriate research directions. The workshop provides an opportunity to share recent advances in formal methods and their integration in software development environments.

We need groups of people working together on different aspects of the software problem, on a long-term basis. Users need to learn the capabilities of formal methods and researchers need to learn the real issues in practical software development. It would be desirable to achieve a consensus on what are the most important aspects, and how they might fit together, supported by an on-going discussion as our understanding grows.

Figure 11 - Page 2, 2nd Monterey Workshop

3. Software Evolution and Prototyping

Software evolution has been addressed from many viewpoints. These include management issues such as which parts of a system should be rebuilt, configuration issues such as keeping track of many versions of the same system, testing issues such as determining which test cases could be influenced by a given change, requirements issues such as determining when the assumptions underlying a system specification are no longer valid, code restructuring issues, performance improvement issues, and much more. Many of these problems do not have accepted or validated formal models, and the application of formal methods is less obvious than in more mature areas, such as proving program properties.

Software evolution includes managing all dynamic structures and activities involved in software development. Much of the work in software process modeling does not emphasize automation. This is because this area failed to separate management issues from computer support issues. Both aspects are needed to help managers make better decisions about software evolution with less effort. Software development processes cannot be totally automated, because human judgement is required for good management.

A traditional view is that software evolution only occurs after initial development is completed. For example, software evolution has been defined to consist of the activities required to keep a software system operational and responsive after it is accepted and placed into production. This term previously referred to maintenance activities, to avoid the deadly negative connotation of that word. Evolution has the connotation of life, and it captures the dynamic aspects of software development if used in the context of an alternative software life cycle like prototyping that includes all activities from requirements specification and system construction to updating operational systems.

Alan Perlis took a broad view of software evolution, emphasising the parallel with biological evolution[1]. Here software systems are analogous to biological species, which adapt themselves to survive in changing ecological niches. Perlis was also interested in the analogy with Maturana's notion of autopoietic systems, which continually construct themselves from their own parts in order to adapt.

In the design and development of our Computer Aided Prototyping System (CAPS), we learned the importance of automated support for software evolution. This project worked in parallel on formal modeling, developing software tools based on the models, and applying the tools to practical problems. We found significant synergy among these activities, changing our approach to all three. Many versions of models and tools have been developed and modified during this process. Our work[2] on formal modeling and experimentation for software evolution is summarized in my joint paper with Drs. Goguen and Berzins and the paper by Drs. Dampier and Berzins in the proceedings of this workshop.

4. Goals of the Workshop

This workshop involves three groups of people coorperating to understand and solve software evolution problems. These are software developers, theoreticians, and sponsors; they come

[1]This paragraph is based on the recollections of Joseph Goguen of some conversations with Perlis late in his life.

[2]This work has been supported in part by the National Science Foundation and the Army Research Office.

Figure 12 - Page 3, 2nd Monterey Workshop

from industry, government, and academia. We have an opportunity to learn and to exchange technical information, including some that would not normally be published for economic and security reasons. For software researchers, nothing can be more important than a better understanding of what is needed for real world software development. This help to avoid research results that are only good for toy problems. Industry people can help this process by clearly explaining what their most important problems are, separating essential aspects from the incidental complications of particular situations, and separating technical problems from political and management difficulties.

We have people with strong backgrounds in many branches of mathematics. Hopefully we can narrow some of the problems so that appropriate theoretical models can be applied to important aspects of practical software development. We may not have the luck to find off-the-shelf mathematics solving our problems. Real problems often have scary complexity. We need help in dividing real problems into subproblems to reduce the complexity. It may also be necessary to approach problems in completely different ways, e.g., probabilistic algorithms can handle some problems of practical size that are intractable using deterministic methods. The software evolution problem is interesting and difficult. Hopefully this workshop will be the start of progress in the formal modeling of software evolution, in a way that will have significant impact on practical software development.

The workshop schedule was organized as follows:

Day 1: Introduction to software evolution & formal methods in software development

Jim Brockett, Naval Postgraduate School, Welcome and Introduction.
Luqi, Naval Postgraduate School, Monterey Workshop '94: Software Evolution.
Daniel Berry, Technion, Israel, Whither Formal Methods?
Mantak Shing, Jim Brockett, NPS, Computer-Aided Prototyping System (CAPS) Demonstration.
Barbara Meyers, Senior Programmer, IBM, Richard Schwartz, VP of R&D, Borland International, Industry Perspectives.
Valdis Berzins, Naval Postgraduate School, An Evolution Control Model.

Day 2: Evaluations of successful formal methods

David Dampier, Army Research Laboratory, Software Change-Merging in Dynamic Evolution.
Richard Waldinger, SRI International, Michael Lowry, NASA Ames Research Center, AMPHION: Towards Kinder, Gentler Formal Methods.
Jacob Schwartz, New York University, Design of Languages for Multimedia Applications Development.
Gio Wiederhold, Stanford University, An Algebra for Ontology Composition.

Day 3: Applications of formal methods and summary of the workshop

Steve Vestal, Honeywell Technology Center, Formal Methods for Complex Evolving Systems.
Dan Craigen and Ted Ralston, ORA, Canada, Formal Methods Technology Transfer Impediments.
Dave Robertson, University of Edinburgh, UK, Making Specification Design More Accountable.
Luqi, Naval Postgraduate School, Wrap-up Discussions.

5. Summary and Conclusions

This section summarizes and synthesizes the conclusions reached in discussions at the workshop, especially the final session. This is necessarily a creative task, because of divergent viewpoints among the participants. The presentations by Berry, Craigen and Ralston were

Figure 13 - Page 4, 2nd Monterey Workshop

particularly effective in stirring up controversy. Much of the discussion was about formal methods in general, rather than about their direct application to software evolution, although it always hovered in the background as a central motivation. Subsection 5.4 below focuses specifically on evolution.

5.1 A Vision for Formal Methods

The workshop generally agreed that there is now much more to formal methods than suggested by the themes dominant in the past, namely correctness proofs for algorithms and program synthesis. Although both of these remain interesting topics for theoretical research, workshop participants felt that their direct impact on the practice of large scale software development was limited.

Papers in the workshop suggest a vision for the future that is less ambitious and more realistic than that of the past. This new vision calls for using formal models and algorithms as a basis for computer tools that can help to solve practically significant problems; these problems are more limited and well defined than in the past. This approach gives up the unrealistic artificial intelligence goal of fully automating software development, by recognizing that human understanding and creativity must play an important role. It also recognizes that user requirements changes are a dominant aspect of practical software development, and gives up the ambitions of general-purpose program verification and synthesis. Past research in formal methods contributes to this new vision by providing the basic concepts and algorithms needed for building more practical models and tools.

All participants in the workshop seemed to agree that formal methods could be very useful, and indeed were crucial for making software engineering into a discipline that is as well understood and organized as other engineering disciplines, each of which relies on sound and well-tested mathematical models.

5.2 An Emerging Paradigm?

A number of successful applications of formal methods reported at the workshop seem to form a cluster suggesting an emerging paradigm for applying formal methods. These applications involve a tool having all or most of the following attributes:

1. They address a narrow, well defined, and well understood problem domain; there may even be an existing, successfully used library of program modules for the domain.

2. There is a coherent user community interested in the problem domain; users have some understanding of the domain, good communication among themselves, and potential financial resources.

3. The tool has a graphical user interface that is intuitive to the user community, embodying their own language and conventions.

4. The tool takes a large grain approach; rather than synthesizing procedures out of statements, it synthesizes systems out of modules; it may use a library of components and synthesize code for putting them together.

Figure 14 - Page 5, 2nd Monterey Workshop

5. Inside the tool is a powerful engine that encapsulates formal methods concepts and/or algorithms; it may be a theorem prover or a code generator; users do not have to know how it works, or even that it is there.

Examples of systems described in the proceedings that fit this discussion are: CAPS (see the paper by Luqi, Goguen and Berzins); ControlH and MetaH (see the paper by Vestal); AMPHION (see the paper by Waldinger and Lowry); the Panel system (see the paper by Schwartz and Snyder); the work of Robertson and Hesketh; and the DSDL translator of Kieburtz. If we were to suggest a name for this emerging paradigm, it might be *Domain Specific Formal Methods*, in recognition of the role played by the user community and their specific domain. Generic formal architectures and tools for a variety of software application domains are promising for future research.

5.3 Problems and Limits of Formal Methods

There was a great deal of discussion in the workshop about problems with the current state of formal methods. It is not surprising that some of this discussion repeated ideas that have become common in the literature, but some of it seemed fresh and original.

Attendees noted that formal notation is alien to most practicing programmers; most have little training or skill in formal mathematics. This motivates points 3. and 5. in Section 5.2. Discussion suggested that this problem is worse in the U.S. than in Europe. For example, set theoretic notation is better accepted in Europe.

Another problem is that some advocates of formal methods take a highly dogmatic position, that absolutely everything must be proved, to the highest possible degree of mathematical rigor; it must at least be machine checked by a program that will not allow any errors or gaps, and preferably it should be produced by a machine. In discussion, it was noted that mathematicians hardly ever achieve, or even strive for, such rigor; published proofs in mathematics are highly informal, and often have small errors; they never explicitly call upon rules of inference from logic (unless they are proving something *about* such rules). There are various degrees of formality, and the most rigorous are very expensive; such efforts are only warranted for critical aspects of systems. Practitioners would like to be able to combine informal development of the bulk of an application with formal methods for the critical parts.

Another problem raised in the discussion is that formal methods tend to be rigid and inflexible; in particular, it is difficult to adapt a formal proof of one statement to prove another, slightly different statement. Unfortunately, in the world of practical programming, requirements and specifications are constantly changing, so that such adaptations are frequently necessary. But classical formal methods have difficulty in dealing with the changes that are such an important part of the real world.

Another problem is that formal methods papers and training often deal only with toy examples, and often these examples have been previously treated in other formal methods papers. Although it may not be possible to give a detailed treatment of a realistic example in a research paper or classroom, it is still necessary that such examples exist for a method to have credibility. To be effective, training in formal methods should treat some parts of a realistic (difficult) application. A related problem is that much of the research in formal methods is not applications oriented.

Figure 15 - Page 6, 2nd Monterey Workshop

There are also technical difficulties with some formal methods. For example, the Z language is hard to use for proving properties of either the specification itself or of associated code. This is because the specification has to be flattened before it can be used, since the structure of its specifications is usually very different from the structure of associated proofs and code. In particular, its schemas do not encapsulate their content, and only support re-use at the text level. Proofs about Z specifications must use the axioms of set theory, and this can be very difficult. For example, proofs using algebra are easier, because the axioms and rules of inference are much simpler, as well as easier to automate. It is hard to convince software developers to use a formal method with technical deficiencies and esoteric symbols.

It is useful to distinguish among small, large, and huge grain methods. This distinction refers to the size of the atomic components that are used, rather than the size of the system itself. The "classic" formal methods fall into the small category. In particular, pre- and post- conditions, Hoare Axioms, weakest preconditions, predicate transformers and transformational programming all have small size atomic units, and fail to scale up. In general, these methods have difficulty handling changes to specifications and requirements, and thus have difficulty with maintenance and fit poorly with software evolution. Transformational programming is less sensitive to errors than the other methods, but has the particular problem that there is no bound to the number of transformations that may be needed; this restricts its use to relatively small and well understood domains.

Some of these methods also have technical deficiencies. For example, the original first order formulation of weakest preconditions is inadequate for expressing loop invariants, which are fundamental for software specification. Without loops and equivalent constructs, programming languages would be nearly useless. This problem has been largely ignored by the formal methods community. However, a second order formulation is adequate, and serves as the foundation of the specification language SPEC used in teaching and research on software engineering at the Naval Postgraduate School for many years.

5.4 Formal Methods for Software Evolution

This subsection discusses issues that relate specifically to software evolution. Software evolution is poorly understood at present, and therefore more in need of help from formal models and methods than many other areas of software engineering. It can be very difficult to formalize a new area, especially without understanding from sponsoring agencies.

The difficulties in this area are not purely technical; social, political and cultural factors are also important, and can dominate the cost of software development. Tools based on formal models can help with both technical and management tasks. They can maintain the integrity of a software development project by scheduling project tasks, monitoring deadlines, assigning tasks to programmers, keeping on-line documentation, maintaining relations among system components, tracking versions, variations and dependencies of components, and merging changes to programs. These problems are important when a large group of programmers work concurrently on a large complex system.

Several workshop participants mentioned requirements capture as an important problem: it is necessary to know what to build, but in fact, this is always a moving target for large complex systems. Constantly changing requirements are a major cause of the difficulty of building such systems; this phenomenon has been called *requirements drift*. An extreme

Figure 16 - Page 7, 2nd Monterey Workshop

form of this is *requirements inflation*, where the requirements grow so much that the system development effort collapses.

A related problem is *traceability*, which is the problem of tracing design decisions, or fragments of specification or code text, back to the requirements that they are supposed to meet. The core of this difficulty is maintaining a complex network of links against the constantly changing requirements, which in turn imply constantly changing specifications and code. Real development projects for large complex systems rarely even attempt this, and those that do find it excessively burdensome, since the current state of practice requires manual entry and update of all dependencies. Since the benefits of adequate tool support for traceability would be enormous, effective formal models for this aspect are of interest. Particular subproblems in this area are formalizing dependencies and developing methods for calculating dependencies and propagating the implications of a change through a dependency network.

Another aspect of this problem is the difficulty of maintaining the dependencies among components in a large software system development effort. Often the components are not adequately defined, e.g., module boundaries may be incorrectly drawn, or not even explicitly declared; also, interfaces may be poorly drawn or badly documented. Without formal models of the dependencies and tool support for managing them, it is impossible to know what effect a change to a component will have, and in particular, to know what other components may have to be changed to maintain consistency. Methods for supporting changes to module boundaries would be useful.

An important practical problem for industry is to deal with so-called "legacy code," that is, old code that is poorly structured and poorly documented; often, it is written in an obsolete or obscure language, and nearly always the programmers who wrote it are long gone. For example, many banks depend upon huge COBOL programs for the success of their enterprise, but find it extremely difficult to modify these programs when business conditions change. As pointed out by Jim Baker from Lockheed, Barbara Meyers from IBM, Richard Schwartz from Borland and Raymond Paul and Ace Roberts from the U.S. Army, software researchers have to accept the realities found in industry and the DoD, as this is the source of their scientific and economic impact.

5.5 Promising Directions

The papers and discussions in the workshop, and the summaries given above, suggest several directions that may be promising for future research.

The first of these was called Domain Specific Formal Methods in Section 5.1. Recall that this involves encapsulating formal models or algorithms, such as an inference engine or program generator, into a tool with an intuitive graphical user interface for writing programs for a specific application domain. The animation of formal languages can be a useful complement to this approach in practice, e.g., for debugging.

Several participants, including Profs. Ramamoorthy, Goguen, Kieburtz, and Shing, raised interesting points about teaching formal methods. The negative impact of teaching a formal method and ignoring the social, political and cultural problems that necessarily arise in real projects was mentioned. For example, students may be taught programming from formal specifications, but not that specifications come from requirements, and that requirements

Figure 17 - Page 8, 2nd Monterey Workshop

are always changing. As a result, they are not prepared for the rapid pace of evolution found in real industrial work. A related problem is that many students feel that formal methods turn programming from a creative activity into a boring formal exercise. This can cause them to leave the field. Students need to know how to deal with real programs having thousands or even millions of lines of code, and carefully crafted correctness proofs for simple algorithms give an entirely misleading impression of what real programming is like. Most of the techniques in textbooks and the classroom are small grain and do not scale up to large complex problems.

Reliable formal method based tools can let students do problems that would be impossible by hand; this should increase their confidence. Teachers should also present methods and tools that work on large grain units, that is, on modules, rather than on small grain units like statements, functions and procedures, because these scale up, whereas the small grain methods do not. It is desirable to develop suites of sample problems that systematically show how and when to apply formal methods, and how to combine them with informal approaches. These goals will require refining and extending existing formal methods and tools, developing more natural user interfaces, rethinking process models, revising curricula, retraining teachers, and experimentally validating the resulting methods in practical situations.

Some participants pointed out that many successful applications of formal methods have occurred in the hardware area. Hence this is a good demonstration of the value of formal methods. It also continues to be a good area for further automation and research.

The discussion outlined above emphasized the importance of applications for formal methods, suggesting a need-driven approach, as opposed to a topic-driven approach. Basic research in computational logic still provides the foundation for many practical applications of formal methods. It is important to avoid a short term view of what technology needs, and also to avoid overselling formal methods, either as a general field or as an approach to particular applications.

Taking a long term integrated view, formal methods are beginning to make a real impact on practical software development, and this impact is likely to increase. It seems unlikely that general purpose tools, such as theorem provers for first order logic, will have an immediate impact on users, although they can be useful inside more narrowly focused tools. Software evolution is a challenging but rewarding application area, where formal models are likely to have a large impact if they can be formulated for the right problems. This can be a major step in turning software engineering into a true engineering discipline, with a solid mathematical foundation on which to base its practice.

Acknowledgements

I would like to thank: the participants in the workshop; the co-chair Jim Brockett; the members of the program and local arrangements committees; and the sponsors, the Army, Navy and Air Force Research Offices, the National Science Foundation, and the Advanced Research Projects Office.

Figure 18 - Page 9, 2nd Monterey Workshop

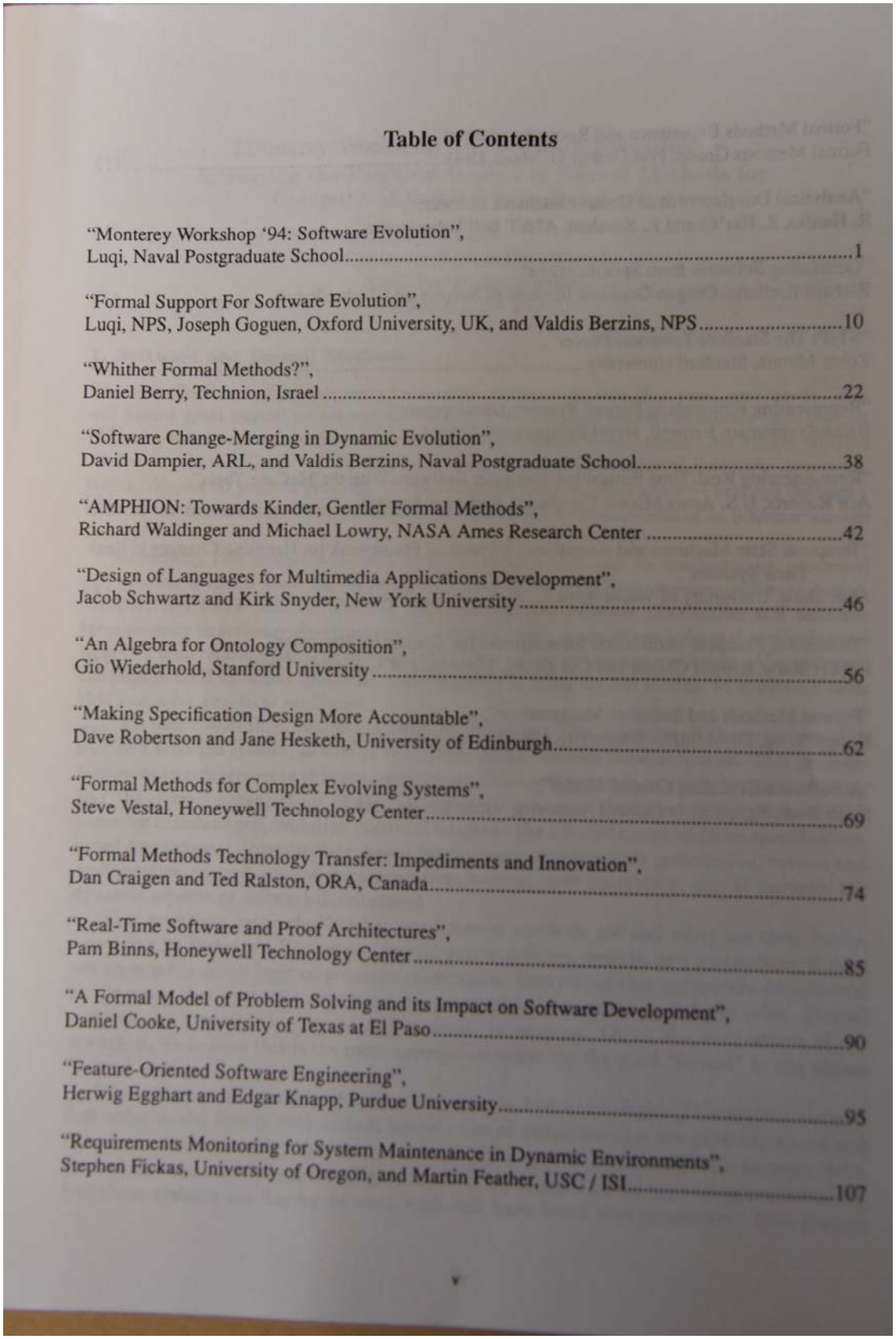


Table of Contents

Figure 19 - Table of Contents, 2nd Monterey Workshop

Figure 20 - Table of Contents, 2nd Monterey Workshop (cont.)

D. 3RD MONTEREY WORKSHOP: SPECIFICATION-BASED SOFTWARE ARCHITECTURE (1995)

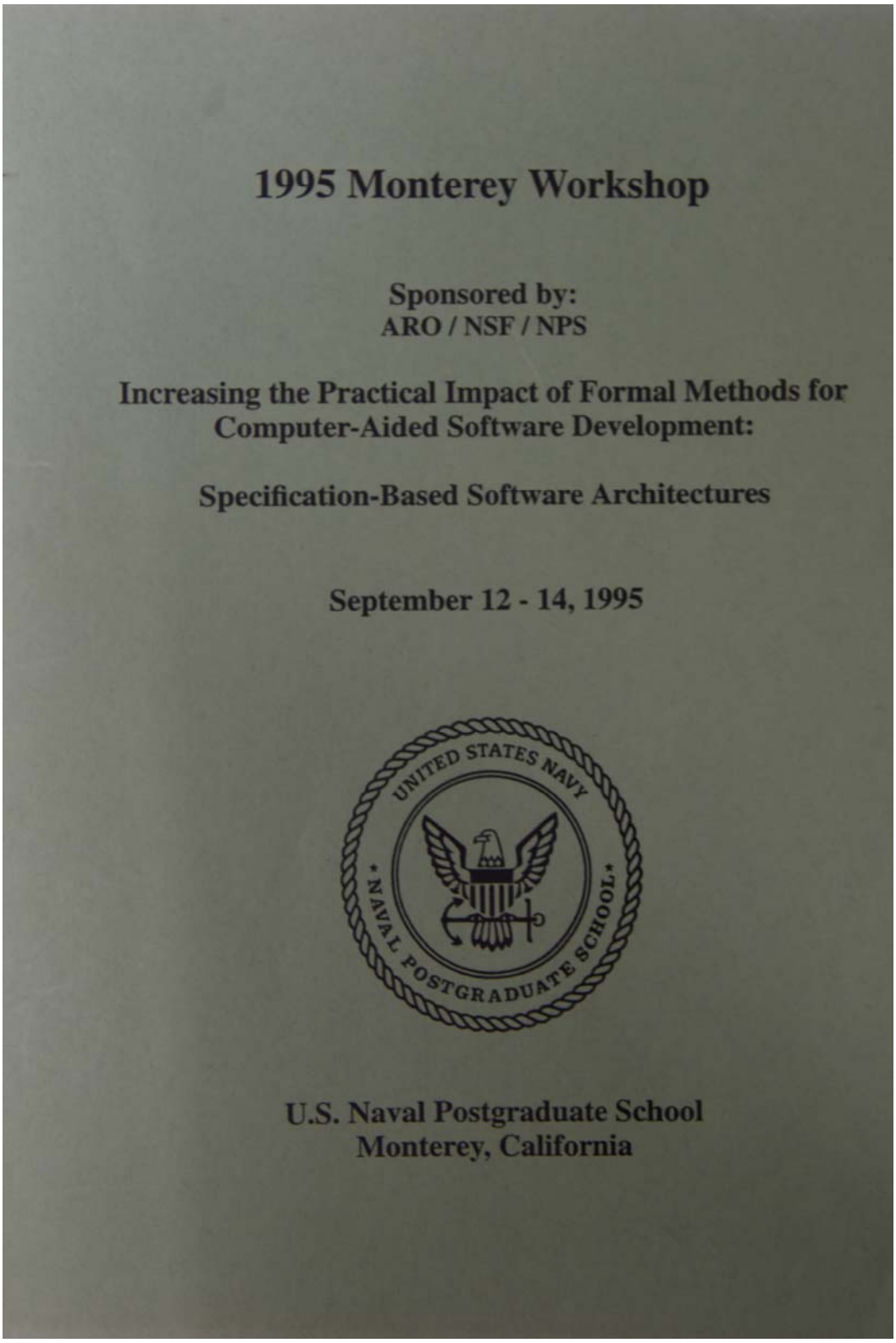


Figure 21 - Cover Page, 3rd Monterey Workshop

Monterey Workshop '95: Specification-based Software Architectures – Increasing the Practical Impact of Formal Methods for Computer-Aided Software Development

Luqi
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

1. Introduction

Current software development capabilities need improvement to effectively produce software that meets users' needs. Formal software models that can be mechanically processed can provide a sound basis for building and integrating tools that produce software faster, cheaper, and more reliably. Formal methods can also increase automation and decrease inconsistency in software development.

The goal of the Monterey Workshop series is to help increase the practical impact of formal methods for software development so that these potential benefits can be realized in actual practice. Each year we focus in depth at one aspect of software development. In 1992, the focus was real-time and concurrent systems; in 1993, software slicing and merging; and in 1994, software evolution. This year's focus is specification-based software architectures.

This workshop helps clarify what good formal methods are and what are their limits. According to Webster's Dictionary, *formal* means definite, orderly, and methodical; it does not necessarily entail logic or proofs or correctness. Everything that computers do is formal in the sense that syntactic structures are manipulated according to definite rules. Formal methods are syntactic in essence and semantic in purpose. Given the motivations of the workshop, we believe this is the most appropriate sense for the word "formal" in the phrase "formal methods." We expect the ultimate main benefits of formal methods to be decision support for and partial automation of the software development process.

2. Scope

Specification-based software architectures address families of software systems with a common problem domain and common solution structures. These domains can be general, and can have many different specializations that are amenable to the same solution structure. The problem domains can have a wide variety, including scientific computing, business information systems, computer-aided design environments, real-time control systems, distributed information systems, and military applications.

We take a specification to be a formal description of the behavior visible at the external interface of a component. A specification typically describes what must happen (liveness constraints), what may not happen (safety constraints), and timing constraints.

A software architecture defines the common structure of a family of systems by specifying the components that comprise systems in the family, the relations and interactions between the components, and the rationale for the design decisions embodied in the structure. The components are subsystems that are used as building blocks in the architecture. They can have a wide variety of scales, and can themselves be defined using lower level software

Figure 22 - Page 1, 3rd Monterey Workshop

architectures, resulting in hierarchical descriptions of system structure. The components are encapsulated black boxes. The component slots in an architecture are abstract in the sense that they can be filled by a variety of concrete components that satisfy the requirements of the slots. These requirements are given via the component specifications associated with the architecture.

3. The Significance of Software Architecture

Software architectures are relevant to many aspects of computer-aided software development, including automatic program generation, reuse, evolution, and systems integration.

Connecting components. Any meaningful interaction between subsystems requires a shared conceptual model that can capture the meaning of the interaction. This model serves to enable design using black-box components, because it provides designers a characterization of the behavior of the component that is independent of its realization. The most effective models are abstract in this sense. The models can also only partially constrain the behavior of a component, with the result that a slot in the architecture can be filled with a variety of components that agree on certain aspects of their behavior and differ in others. This enables a single design to provide a controlled spectrum of possible system behaviors.

The conceptual models of component behavior also enable bridging between different concrete realizations of interfaces with the same abstract meaning. Support for software architectures should include automatic means for generating bridging transformations that enable connections between components with common conceptual models but different physical realizations of the same abstract interaction, including differences in data representations and control conventions.

Relation to languages. A software architecture is based on common models of computations. The most useful of these are abstract ones, which can serve to unify a variety of concrete realizations of the same conceptual model.

Practical applications require descriptions of particular architectures. For effective automation support, these descriptions should be expressed in a formal language that embodies the computational models underlying the architecture. Different choices of models lead to different kinds of languages. Overly detailed models of computation can quickly lead to very complicated languages and architecture descriptions. A suitably abstract model is essential for simplicity and practical usefulness.

Relation to program generation. Generating programs from specifications is very difficult, and is probably not tractable in an unconstrained context. A given problem domain and a given set of general solution structures specified by a software architecture can make the problem tractable in practice. In such a situation, the program generator is not required to create new solutions to problems, but only to tailor the general solutions given by the architecture to particular instances of the problem domain addressed by the architecture. Thus the architecture defines the range of problems that can be handled by a given solution generator.

Relation to composition and reuse. One of the difficulties in creating large systems by connecting (composing) smaller systems is compatibility between the conceptual models underlying the subsystems. A software architecture defines a common conceptual model. Components designed or generated to fit the same architecture will have compatible con-

Figure 23 - Page 2, 3rd Monterey Workshop

ceptual models, and thus consistent connections can be created, possibly via some bridging code to transform between concrete representation conventions. This approach can prevent severe integration problems that could in the worst case be solvable only by redesigning one or more of the components.

Reuse is subject to a similar difficulty, because independently developed components are unlikely to have completely consistent conceptual models. However, components designed to fit a given architecture can be reused without modification in the scope of that architecture. This is significant because economically effective reuse depends on reuse of components without modification.

Relation to decomposition and analysis. Complex systems are understood by people via hierarchical decomposition. An architecture contains the information about the interface conventions and the requirements associated with the component slots in the architecture that are needed to make this work on a large scale.

Relation to evolution and merging. The constraints and conventions associated with a composite design are essential to determine what can be changed without damaging a design. This is precisely the information recorded in the specification part of a specification-based software architecture. Much of the difficulties with evolution of legacy software stem from the loss of this information.

Software change merging is the process of automatically combining several changes to the same version of a software system. An essential requirement for this process is to detect all potential conflicts between changes, and to guarantee semantic integrity of the results of no conflicts are reported. The behavioral requirement information contained in a software architecture can enhance this process and enable more discriminating results. Recent results show that change merging cannot be done via a divide and conquer approach, which implies that the computational cost of change merging increases faster than linearly with the size of the system. Change merging at the architectural level has been shown to be feasible, and this approach is most promising for large systems because it appears to be computationally tractable.

Relation to networks Networks are the physical means for realizing connections between remote nodes in distributed architectures. Knowledge of the constraints and conventions associated with a connection in a software architecture can in principle be exploited by the network protocols to provide better service.

Relation to hybrid systems Some of the components slots in a software architecture can in principle be filled by components realized in hardware rather than by software. The information in a software architecture can be used to support hardware/software codesign, and can eventually lead to automatic realizations in hardware for some classes of components.

Relation to heterogeneous architectures Different components in a software architecture can in principle be implemented using different programming languages, operating systems, or hardware platforms. A carefully structured description of the software architecture that provides annotations to refine abstract architectures with physical realization attributes can effectively support generation of connections between nodes with different (and hence incompatible) physical realizations, by automatically constructing the required transformations and inserting them in the connections.

Figure 24 - Page 3, 3rd Monterey Workshop

4. Workshop Summary

The workshop consists of formal presentations on the subject, interleaved with discussions to bring out implicit assumptions, clarify relationships between different points of view, and make assessments. The conclusions of each session are summarized by the reporters for the session, and the results are integrated into the article "Summary of the '95 Monterey Workshop" on pages 107-112 of this proceedings.

Figure 25 - Page 4, 3rd Monterey Workshop

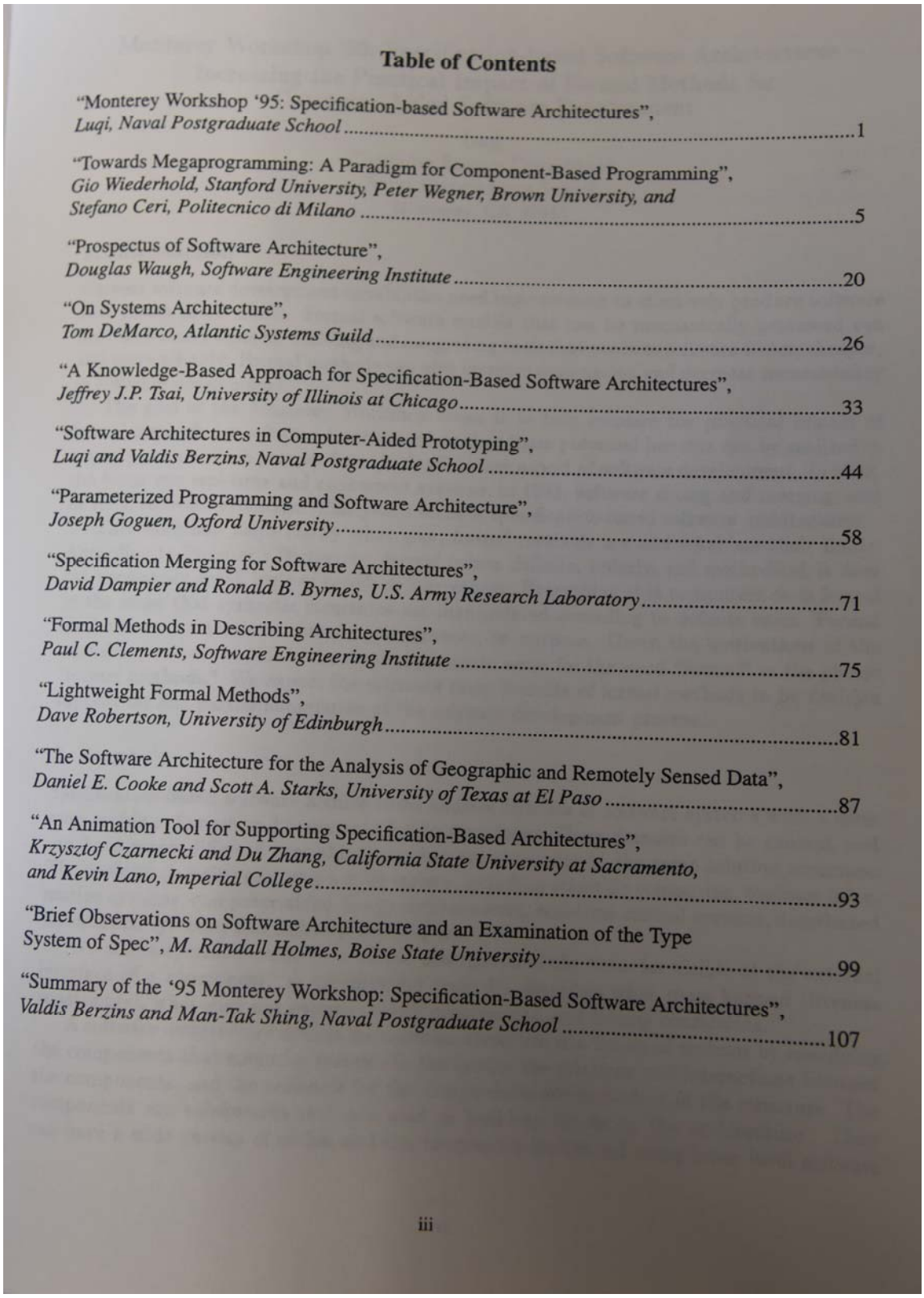


Table of Contents

Figure 26 - Table of Contents, 3rd Monterey Workshop

E. 4TH MONTEREY WORKSHOP: CAPSTAG-COMPUTER AIDED PROTOTYPING (1996)

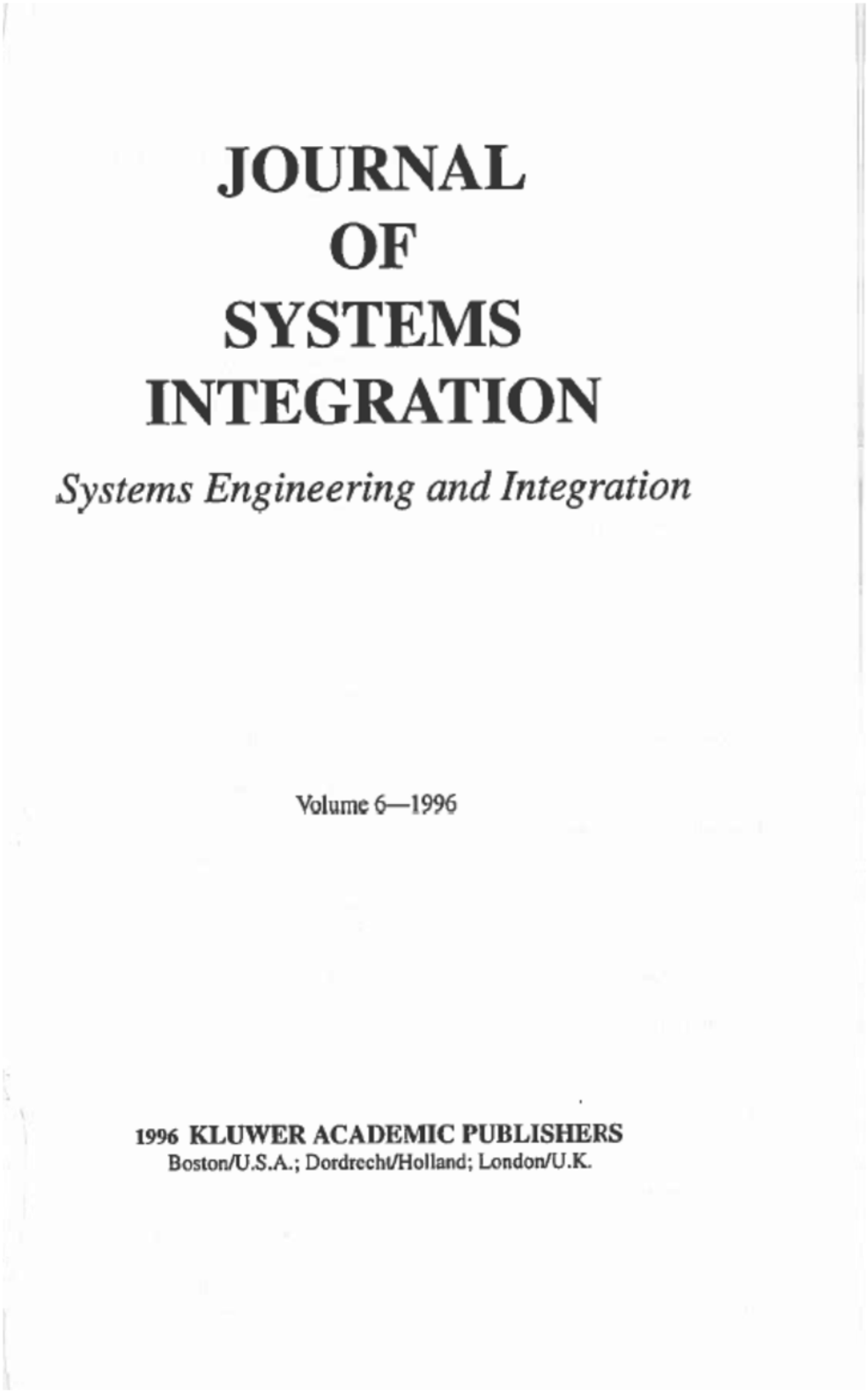


Figure 27 - Cover Page, 4th Monterey Workshop

JOURNAL OF SYSTEMS INTEGRATION

Systems Engineering and Integration

Volume 6, Nos. 1/2, March 1996

CAPSTAG PROCEEDINGS

Special Issue: Computer-Aided Prototyping

Guest Editors: Luqi

Figure 28 - Table of Contents, 4th Monterey Workshop

Journal of Systems Integration, 6, 5–7 (1996)

Foreword: Software Engineering

Only after the appearance of the first large scale relay computers was the problem of programming really faced, although the old accounting machines required "programming" in enormous detail. Stibitz, Aiken, and Zuse all originally came from engineering backgrounds. Of the electronic computer people, Mauchly, Eckert, von Neuman, and Turing, to name a few of the more noted, none seemed to have understood the difficulties of programming. Only Zuse, of the above names, did anything significant in software, while both von Neumann and Turing, even with mathematical backgrounds, never acknowledged the difficulties of programming, though they both used program invariants.

Very early RAND Corporation produced a neat floating point simulator for the IBM CPC and showed what could be done in spite of the computer designers. Wilkes produced the first published book on programming, for the EDSAC. The SHARE organization, mainly involving industrial sites, was created to cope with the IBM machines. Now we have few, if any, similar cooperative organizations with adequate financial industrial backing to attack the problems. Another great contribution was by Backus at IBM who produced Fortran, which has outlasted most of the languages devised by theoreticians. The delay in the broad attack on software problems should not be attributed to ignorance so much as to the simple fact at that time storage was expensive and small, von Neumann saying that 2000 words should be enough for a high speed machine!

By the mid 50's the use of engineering methods for software was being to be mentioned frequently. However, the forty years since then has not produced what was hoped for, and since by then the germs of all the present problems were known, including parallel processing, it bears looking into why we still have not resolved many of them.

First, and perhaps most importantly, those in charge of the early computer installations were either engineers or from the hard sciences like physics, with a few from classical mathematics. The following generation of people did not have such backgrounds to draw on, though they still preached software engineering. At the lower levels we soon found that women from home economics, music, and other disciplined fields often made better programmers than mathematicians with only an undergraduate degree, or the early computer scientists for that matter. The habit of self discipline seems to be a badly needed trait in the software area, rather than amusing speculations, cute programs, new ways of crashing the system, and ingenious devices. As sports coaches know, a group of prima donnas does not make a winning team!

Second, once storage grew as it has, the software problems became orders of magnitude larger, and this profoundly changed the very nature of the problem. Software suddenly became much more difficult than any of the early people ever thought it could be.

Third, the range of applications widened greatly—in the early days only people with hard number crunching problems from engineering, payroll, and the sciences could afford the

Figure 29 - Page 5, 4th Monterey Workshop

cost of the machines, while now the range is enormous. But there are another differences! No physicist trained, say, in acoustics would take on a problem in solid state physics without first acquiring an adequate background of knowledge in the field of application. Fortunately engineers and scientists typically have a deep training in problem solving, which is lacking in most Computer Science programs. The present day programmers are arrogant; shifted from data base management to payroll few, if any, would bother to even consider taking night courses in accounting, payroll practice, and the legal aspects of payroll, so they would know what they were trying to do.

Fourth, the size of the resulting programs is greatly inflated. Writing in high level languages, and descending stage by stage to the machine level produces most of the inflation because it is done by machine tools and the corresponding programs have not had the attention to efficiency that even Backus and his group put into Fortran. The resulting programs are needlessly large and hence hard to test and maintain. Can anyone really believe current software packages need anywhere near the megabyte storage they require?

Fifth, in software there has always been a great willingness to make changes in the specifications, and this makes the job tenuous; hardware people have the habit of freezing a design and not letting a large number of new things be incorporated into it. When you allow changes then you get errors, delays, and cost overruns.

Sixth, software changes are so easily made (in theory) that programmers are seldom restrained, or the changes tracked with the care that hardware changes get. There is a pervasive attitude among software people that they can always fix it up later if it is necessary. The rule in software seems to still apply; "There is never time to do it right but there is always time to fix it later." Experience shows where programs are fixed is where to look for errors.

Seventh, responsibility has not been put on the software people, as the hardware people traditionally have done. The formal act of "signing off" (at many different stages and many levels) with your name in ink on documents produces a responsibility when things are found to go wrong. In software the errors are generally shrugged off by the programmer. In engineering we do not let this happen so easily, we pin it down squarely—witness the Hubble telescope goof where with time the actual details were finally extracted and the guilt apportioned to the individuals. In building an early software system I told the programmers I would get them time on machines for testing and once the machine arrived locally they had exactly one week to fix things, after that they would personally own any errors that appeared—I was not going to issue an endless sequence of field changes. Amazingly, they rose to the challenge (threat?). The extent to which commercial software houses pin the responsibility on the individual and make them feel the consequences of errors, is at present unknown.

Eighth, one may wonder if indeed software can be "engineered", or if the goal is a holy grail. What is thought to be the reliability of hardware (and mathematics) is often greater than it is in fact. Can the adoption of the traditions of knowing the background of the job before you start, the disciplined engineering approach to the problem, and assignment of responsibilities firmly and unambiguously, bring us to where we want to be?

This volume contains a number of papers grappling with the legacy that Computer Science and the practitioneers of the early days left undone, and did not understand the magnitude

Figure 30 - Page 6, 4th Monterey Workshop

of the task. That we are often unable to specify the problem is not unique to software; it occurs in many new engineering projects so pilot models are built and tested. In software rapid prototyping is the equivalent. Certainly, the goal that humans deal most of the time with things at a high level, and all the final instructions be written by machines instead of humans, is essential. The idea that algorithms be automatically incorporated into the writing of the software is proper. Software appears at last to be headed in the proper directions which some earlier people recognized but could not enforce due to the lack of appropriate software tools, which are non-trivial to write. The task is great, and one knows that the final result will be an amelioration of the worst, but it will not be perfection. This outside observer is encouraged by reading the papers, but sees much to be done in the future. The path they are on seems to be the right one, but carelessness persists. The famous collapse of the telephone system and the Hubble telescope failure both occurred because built in safeguards were omitted—which is a tendency all too strong in the software field and must be watched carefully. Rapid prototyping is not a fad that will fade away, it is the only path we know to the future for large, ill-specified software projects.

R. W. Hamming
Naval Postgraduate School
Monterey Cal. 93943

Figure 31 - Page 7, 4th Monterey Workshop

Journal of Systems Integration, 6, 9–14 (1996)

Foreword: Importance of Software Prototyping

Modern software development demands the use of Computer Aided Prototyping, because of its effectiveness in gaining understanding of the requirements, reducing the complexity of the problem and providing an early validation of the system design. For every dollar invested in prototyping one can expect a $1.40 return within the life cycle of the system development.

Barry Boehm's experiments showed that prototyping reduces program size and programmer effort by 40%. It is the technology that is the foundation for his Spiral development method. Prototyping is being used successfully throughout AT&T Bell Laboratories to gain an early understanding of sytstem requirements, to simplify software designs, to evaluate use interfaces and to test complex algorithms. The theme of this issue is Computer Aided Prototyping. It is a best-in-class software approach.

Fully 30–40% of system requirements will change without prototyping. Computer Aided Prototyping offers the hope of looking at the dynamic states of the system before we build it, whereas most other software engineering focuses on the source code. The special problems of reliability, throughput and response time as well as system features are addressed in the best prototypes. A new field of study, Software Dynamics, will emerge once Computer Aided Prototyping is widely practiced. It will focus on quantitative analysis on how software performs under various loads and include a set of design constraints which will make it possible for us to build components which can be hooked together without exhaustive coverage testing.

Software is hard because it has a weak theoretical foundation. Most of the theory that does exist focuses on the static behavior of the software—analysis of the source listing. There is little theory on its dynamic behavior—how it performs under load. To avoid serious network problems software systems are over-engineered with plenty of bandwidth for two or three times the expected load. Without analysis of its dynamic applications have no idea of the resources they will need once they are working. Software has the awful propensity to fail with no warning. One manager of my acquaintance issued a memo stating, "There will be no more software bugs!" The trouble was he meant it; no joke. Even after we find and fix a bug, how do we restore the software to a known state, one where we have tested its operation? For most systems, this is impossible except with lots of custom design that is itself error-prone. Software prototyping has proven its metal in helping designers avoid these problems in their production systems.

Much has been written about the best way to develop software applications. But there is no "best way." Both prototyping and requirements are necessary. The tried-and-true process of synthesis and analysis is used to solve software engineering problems. Bottom-up is the synthesis. Top down is the analysis. Bottom up is prototyping. Top down is developing requirements. Prototyping is the best way to encourage synthesis. Prototyping also eases

Figure 32 - Page 9, 4th Monterey Workshop

communication with the customer and with the designer. Formal written requirements are needed to establish a clear definition of the job, to control changes and to communicate the system capabilities between the customer and the developer.

So where does this leave us? Start with an English language written statement of a problem and broadly outline its solution. Now build a prototype for the elements where you need insight. Analyze the prototype either by refining the prototype or building a new one. Once you and the customer agree on the workings of the prototype, write requirements which include features, performance goals, product costs, product quality, development costs and schedule estimates.

What do I mean by prototype? Prototyping is the use of approximately 30% of the ultimate staff to build one or two working versions of various aspects of a system. It is not production code but it may eventually become pre-production code or it may be completely discarded. In the prototyping effort, we normally aren't concerned with the maintainability of the code nor are we concerned with formally documenting it; in computer aided prototyping these issues are addressed by automated tools. Code resulting from prototyping is often used to train the programmers. Only after we have written specifications resulting from the experience with the prototype should we start the formal development process. If we are fortunate enough that some of the code that was developed for the prototypes can be carried forward, that's great, if not, there is no loss.

A prototype produces "running" software and the production development produces "working" software.

Recent project experience has led to the widespread acceptance of the concept that early prototyping is fundamental to the success of operations support software products. The reasons why prototyping is fundamental include:

1) The prototype provides a vehicle for systems engineers to better understand the environment and the requirements problem being addressed.

2) A prototype is a demonstration of what's actually feasible with existing technology, and where the technical weak spots still exist.

3) A prototype is an efficient mechanism for the transfer of design intent from system engineer to the developer.

4) A prototype lets the developer meet earlier schedules for the production version.

5) A prototype allows for early customer interaction.

6) A prototype demonstrates to the customers what is functionally feasible and stretches their imagination, leading to more creative inputs and a more forward looking system.

7) The prototype provides an analysis testbed and a vehicle to validate and evolve system requirements.

Now, what is wrong with prototyping? If the initial prototype is too far off the mark, we can get some disastrous results such as fielding unresponsive systems that really turn the user off, or we could concentrate on short term needs or develop sub-optimal systems. Because

Figure 33 - Page 10, 4th Monterey Workshop

of that, we should write requirements to force us to do a careful analysis of the users overall problem before plunging into the code. It is difficult to manage and schedule prototyping and hard to get people off the prototype into the real system. Specifically, getting them to deal with size, performance, and the build constraints and practicalities of a production system too can be a management problem. Computer aided prototyping can help avoid these pitfalls.

In one project we tried to use structured system analysis which is an excellent analysis tool but a horrible way to communicate with the customer and with the software designers. The problem is, the customer just doesn't want to learn the language of structured system analysis and that the System Engineers tend to jump from the general to the detailed. Whereas, English text feature memos provide a convenient way of communicating across both boundaries. Nevertheless, structured system analysis makes sure that our thinking is clear. Unfortunately, system analysis implies an architecture which often clashes with the software architecture and makes it difficult to understand what the system must do. What the developer tends to do is talk with the System Engineer rather than read the formal written requirements. The prototype makes these discussions effective.

Here are several effective real-world uses of prototypes:

1. Project: Order Reading and Analysis Software

Size of Prototyping Effort: 12K lines of C Code (10% of final system module)

Purpose:

1. Find a method for order reading and analysis, applicable to variable formats.

Experience:

1. Final requirements written based on prototype results.
2. Trained developers for the possibility of a tunable system.
3. Early evaluation of functional decomposition and performance.
4. Elimination of usable code alternatives as not feasible.
5. Prototype was thrown away due to decomposition and performance problem.

Duration and Staff of Prototype:

Four people for eight months.

2. Project: Store and Forward Message Switch

Size of Effort: 2% of total System of 500K lines of code

Purpose:

To evaluate a new scheduling algorithm for an existing system.

Figure 34 - Page 11, 4th Monterey Workshop

Experience:

In a store and forward message switching system, we found the buffer overlcad strategy was unstable. After the system went into overload and returned to normal processing, it would immediately poll for more traffic. Polling had a higher priority than distributing the messages already queued in the mistaken belief that polling must be the highest priority task to meet the response time requirement. This exhausted even more buffers, drove the system into overload again and caused it to stay in overload longer than before. To convince the users that lowering the priority of polling would solve the problem, a prototype system was put together in the test lab. It demonstrated stable over-load response with an imperceptible increase in response time. With the prototype in hand, schedules for a system release with improved over-load response were adopted. Demonstration of the prototype avoided an emotionally charged battle with the users over response time. The prototype became pre-production code which took one year to field after it was working in the lab.

Duration of Prototype

Four month and one person. The production code required one calendar year and three staff years.

3. Project: Order entry

Size of prototyping effort: 10%

Purpose of Prototype:

1. Evaluate Human Interface by the user.

2. Validate economic assumptions.

3. Train software developers before requirements are available.

Experience:

The Human Interface was changed to put more data on a single screen as the user preferred to see those transactions that could be completed with a single action and therefore the screens became denser.

The table structures were changed to make them easier to maintain and more flexible to change.

The economics proved favorable for the system. High sensitivity of the economics to response time was established.

An earlier version was rushed to production without adequate analysis resulting in project

Figure 35 - Page 12, 4th Monterey Workshop

termination due to difficulties in operating the system and lack of capacity growth even though the user was thrilled with the system.

Duration and Staff:

The prototype took nine months to build and required approximately seven people for nine months.

4. Project: Outside Plant Data Base System

Size of Effort: 5% of 500K lines of source code.

Purpose:

To evaluate data base structures for an outside plant data base implemented in the Facility Assignment and Control System and to experiment with approaches to handling multiple future states of equipment usage.

Experience:

A new data base structure using hyper graph theory was invented and an algorithm for explaining why the heuristic approach being used worked was developed.

The prototype became the production code. UNIXTM was used to model loop plant by way of a directed graph.

The prototype was ported from the PDP 11/70 to Sperry to demonstrate the transportability of the code.

Duration:

Three people for 15 months.

These prototyping experiences show that even when the prototype is intended to be thrown away it may turn out to be deployable software, the prototype is a step toward final system definition captured in a set of formal requirements, and the prototype is the ideal vehicle to get system engineer, developer and user to deal with the problem statement and potencial solution in concrete terms.

Today's software technology cannot support scalability, robustness and reliability. In their article on scalable software libraries, Don Batory and his colleagues at the University of Texas at Austin argue that a large feature-rich collection of software components is inherently unscalable. To go further, we cannot be certain that a small change in its software will result in a small change in system performance. We test and retest every time we make the smallest change or we suffer system crashes and cranky customers. University research focus on these and similar issues will lead to a technology for software design that can be taught to aspiring software engineers and move us to a disciplined approach to software development.

Small changes in the software or in the input data often result in unstable but predictable system performance, remarkably similar to chaos theory. The April 25, 1994 Forbes points

Figure 36 - Page 13, 4th Monterey Workshop

out that a three-line change to a 2-million line program caused multiple failures due to a single fault. Instabilities can arise when:

1. Computations cannot be completed before new data arrives.
2. Roundoff errors or buffer usage builds and eventually dominates system performance.
3. The algorithm embodied in the software is inherently flawed.
4. Memory leaks cause overlapping memory use.

When we finally get a system to work, how do we know it will continue to work? As its load grows, can we be sure that it will respond properly to out-of-range data, that it will handle data that arrives when it shouldn't, and that its performance degrades predictably? These are just three of the eighteen items that Robyn Lutz proposed in his safety checklist. In fact I hung my laptop computer as I wrote this introduction and lost some text.

Using the technologies captured in the philosophy of Computer Aided Prototyping we can begin to get at these issues and questions in a systematic way and build the foundation for the study of Software Dynamics, which is so sorely lacking. Once we have this body of knowledge we can reliably predict system performance for a range of offered loads and define the point where the software will fail. It will move the software industry to quantitative analysis and to component modeling and specifications. It will mark the maturation of the software industry.

Larry Bernstein
Executive Director AT&T Bell Laboratories
ACM, Ball State, and IEEE Fellow

Figure 37 - Page 14, 4th Monterey Workshop

Journal of Systems Integration, 6, 15–17 (1996)

Guest Editor's Introduction

The goal of research on computer-aided prototyping is to provide tools for developing reliable software systematically with high productivity. This area has strong connections with many other areas of software engineering, because it must face many of the same problems as other software development methods. For this reason, our goal of building a Computer-Aided Prototyping System (CAPS) required us to study software in its context of system engineering, and to consider software engineering issues ranging from requirements and domain modeling to software synthesis, analysis, reuse, and evolution, and also required us to integrate diverse systems addressing those issues.

Often, difficulties with requirements for new systems only appear when clients actually use the system. Rather than throwing away an initial full-scale implementation, it is better to develop requirements incrementally through inexpensive prototypes. This reduces the cost and increases the value of the envisioned system to the people it serves. Computer aid for rapidly and inexpensively constructing and modifying prototypes makes this feasible.

Evolution, which is the growth and change of systems, plays a key role in both computer-aided prototyping and in system engineering. Computers can support evolution by retrieving suitable reusable components from a software base, by merging independent changes to software components, by tracing dependencies to determine which parts of a software system are afffected by a requirements change, by using constraints to prevent errors by completing partial designs, and by generating programs, e.g., executable schedules for meeting real-time constraints or for gluing components together. Furthermore, methods and tools developed to support prototyping can often be applied to the ongoing evolution of mature software systems.

The CAPS project has focused on computer support for designing, building and modifying large real-time systems, because requirements for such systems can be especially troublesome. Begun more than a decade ago, this effort has addressed all the areas mentioned above, as well as prototyping language design, engineering databases, and project coordination based on computer-aided design and manufacturing.

The Papers

The papers in this special issue illustrate the diversity of the connections between prototyping and other areas of software engineering and some of the research progress in these subareas of system engineering:

1. *A Model-Based Computer-Aided Prototyping System*, by Li and Ketabchi, considers how domain models can be used to rapidly construct and exercise prototypes in an

Figure 38 - Page 15, 4th Monterey Workshop

object oriented framework. The MB-CAPS system has a graphical interface designed for problem domain experts who may not be programming experts.

2. *Real-Time Scheduling for Software Prototyping*, by Luqi and Shing, explains how hard real-time constraints can be realized for prototyping systems. This paper presents some practical scheduling algorithms for the single and multiple processor cases, and assesses their effectiveness.

3. *Translating EQL Programs for Bounded-Time Execution*, by Mok and Wangs, shows how to realize rule-based programs that must complete execution within fixed bounds on computation time. The paper gives both a method for translating a rule based program into programing code and a method for determining a bound on the execution time of the resulting target code.

4. *Software Component Search*, by Goguen *et al.*, suggests improved methods for finding reusable software components for a given design. The user's query is a syntax declaration and a set of test cases. Incrementally ranked multi-level filtering finds a small set of most promising components. Users do not need specialized skills in module semantics.

5. *Software Merge: Combining Changes to Decompositions*, by Berzins and Dampier, addresses a subproblem of software evolution, providing a method for combining changes to a software design represented by a hierarchy of annotated dataflow diagrams. This method addresses changes to the structure of a design, as well as changes to the specified system behavior.

Progress to Date

The prototyping language of the CAPS system, called PSDL, has been designed, formally specified, and partially implemented. Its semantics covers single and multiple processor implementations in a possibly distributed environment. Implementation techniques include program generation, automated scheduling, and reuse. One reuse method takes the design used for constructing a prototype as the basis for a query. Software evolution models have helped in developing automated methods for configuration management, team coordination, and semantics based merging of changes to prototypes. Summaries and references to papers describing these results are accessible over the world wide web at

```
http://www.cs.nps.navy.mil/research/caps
```

and

```
http://caps.airmics.gatech.edu/caps.html
```

Release 1 of the CAPS system is available free of charge from DISA (703-681-2364 or email to dsrscao@ssed1.ims.disa.mil), and from the Ada Joint Program Office (800-ADA-ICI1).

CAPS has been used to develop a variety of prototypes, including a robot controller, a fish farm controller, a simple autopilot, a controller for a hyperthermia cancer therapy system, a secure communications link, a generic C3I system, a land to air missile, and a cruise missile guidance system.

Figure 39 - Page 16, 4th Monterey Workshop

Ongoing and Future Research

The CAPS project is currently working to improve support for software evolution, software reuse, program generation, real-time scheduling, and configuration management. One interesting topic for future research is the transition from prototypes to final products. Progress here could remove barriers between prototyping and product development, and success could revolutionize the way the software industry does business, by introducing much higher levels of automation. Research issues here include the following: (1) optimization of prototype implementations, (2) transforming prototypes to run on hardware and operating systems different from those of the prototyping environment, (3) switching from simulated external systems to actual ones by generating appropriate concrete interface programs, (4) introducing data persistence and scaling up to large volumes of data, (5) realizing fault tolerance, (6) realizing security constraints, (7) certifying the integrity of designs, and (8) checking that implementations realize designs.

The advances in computer-aided prototyping, partially reported in this issue, have been made possible by the contributions of a few hundred researchers and graduate students, and reflect Raymond Yeh's vision of twelve years ago. It is due to the trust, encouragement and strong support from Jack Schwartz, Dan Berry, KC Tai, CV Ramamoorthy, Sartaj Sahni, JB Rosen, David Hislop and many others, from the time when the CAPS approach to software development was a little "ugly duckling". I am grateful to all these people.

Luqi
Computer Science Department,
Naval Postgraduate School,
Monterey, CA 93943

Figure 40 - Page 17, 4th Monterey Workshop

F. 5TH MONTEREY WORKSHOP: REQUIREMENTS TARGETING SOFTWARE AND SYSTEMS ENGINEERING (1997)

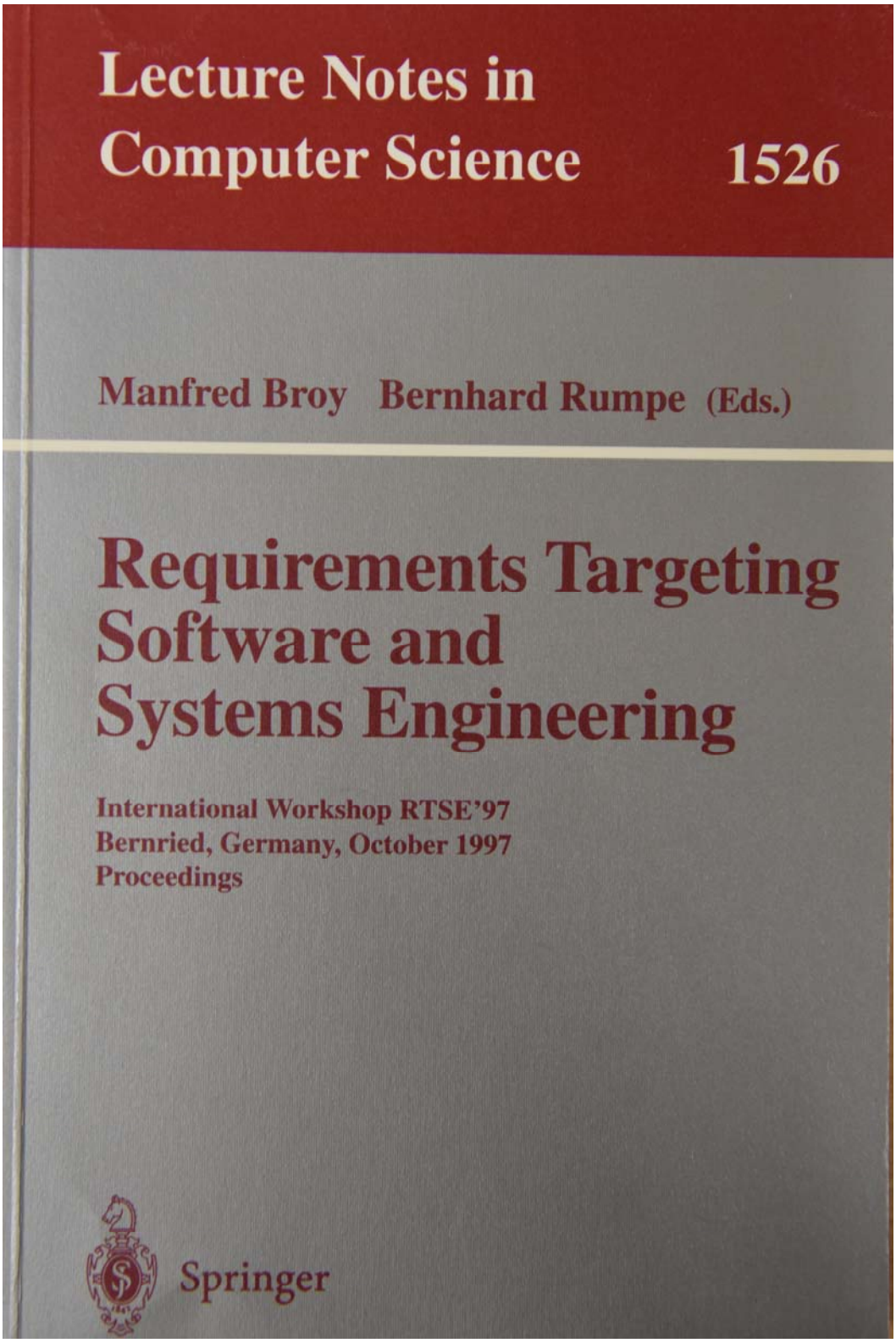


Figure 41 - Cover Page, 5th Monterey Workshop

Preface

Software engineering research has different profiles in Europe and North America. While in North America there is a lot of knowhow in the practical, technical, and organizational aspects of software engineering, in Europe the work concentrates more on foundations and formal modeling of software engineering issues. Both approaches have their individual strengths and weaknesses. Research driven solely by practice in software engineering runs the danger of developing into a shallow field that fails to find a solid scientific basis or to contribute substantially to the progress in software engineering. Work concentrating on formal aspects alone is in the danger of becoming too theoretical and isolated from practice, so that any transfer into practical application will fail.

Substantial progress in software engineering can be achieved, however, by bringing together pragmatic and foundational work in software engineering research. This can provide a step toward a common scientific basis for software engineering that allows us to integrate the various research results, leading to fruitful synergetic effects. It will also help to identify critical research paths and to develop an appropriate paradigm for the scientific discipline of software engineering.

In software and systems engineering it is necessary to distinguish the enormous difference between the dynamics in development we refer to and the limited scope assumed by many of today's software managers who still use outdated techniques. Many of the unsolved problems associated with the old techniques are symptoms of a lack of formalization and a lack of automation support.

It was the goal of this workshop to bring together experts from science and practice in software and systems engineering from North America and Europe. The workshop focused on unified sets of formal models and associated methods suitable for automation for many aspects of software development, in particular those that address change and those that apply on a large scale. Some of the aspects of software evolution are

- modifiable software architectures,
- resource changes,
- context changes,
- requirements changes,
- changes to decomposition structures, and
- changes in plans.

These issues are closely related to formal representations of the version history, and formal representations of the activities that produced existing versions or have been proposed to produce future versions. The essence of the software engineering product model is to establish and maintain consistency among various kinds of software artifacts throughout the development and evolution process,

Figure 42 - Page v, 5th Monterey Workshop

including consistency between requirements, architectures, and programs. Automated support is needed to determine dependencies and to use this dependency information to provide decision aid for software synthesis, analysis, and evolution. Many versions of each artifact are produced as the software evolves, and changes in the dependency structure must be recognized and reacted to. The challenge is to formalize the problems in this area better, and to develop some of the badly needed technical solutions.

If we as a community can succeed in doing this, the results will provide convincing evidence that formal methods can have strong practical value, and help reverse the trend of weakening support for the subject from both industry and governments. It seems that previous work on formal methods can be applied to problems related to these topics, but it may require non-traditional approaches. This challenge helped to trigger new ideas at the workshop, and perhaps opened new opportunities for progress.

It is well recognized nowadays that software and systems engineering is an important issue in technical systems that still lacks a proper scientific basis. Numerous initiatives in academia, especially under the heading of formal methods, toward such a scientific basis have produced many valuable and interesting scientific results; but still a lot of work lies ahead of us to actually integrate this with the practice of software engineering. Nevertheless, we can observe that a beginning has been made to bring together practical and scientific approaches. A good example for this is the Unified Modeling Language, which was designed only recently and will evolve further. The fact that a proper semantic basis is needed for proper methodological support is much more widely recognized than before. Further efforts are now needed to give scientific research the right focus on the questions that are important in practice and to stimulate a transfer of ideas between academia and application. It was the goal of the workshop to contribute to this process.

The workshop took place in early October 1997 in Bernried, Germany. It was a highly successful event and an encouraging step toward the unification of the various aspects and techniques of software and systems engineering. It is our pleasure to thank Luqi for excellent cooperation in preparing and implementing the workshop and Sascha Molterer for his distinguished help in organizing the workshop. We also thank the Army Research Office and in particular Dave Hislop for financial support.

August 1998

Manfred Broy, Bernhard Rumpe

Figure 43 - Page vi, 5th Monterey Workshop

Table of Contents

Foundations of Software Engineering

Methodology

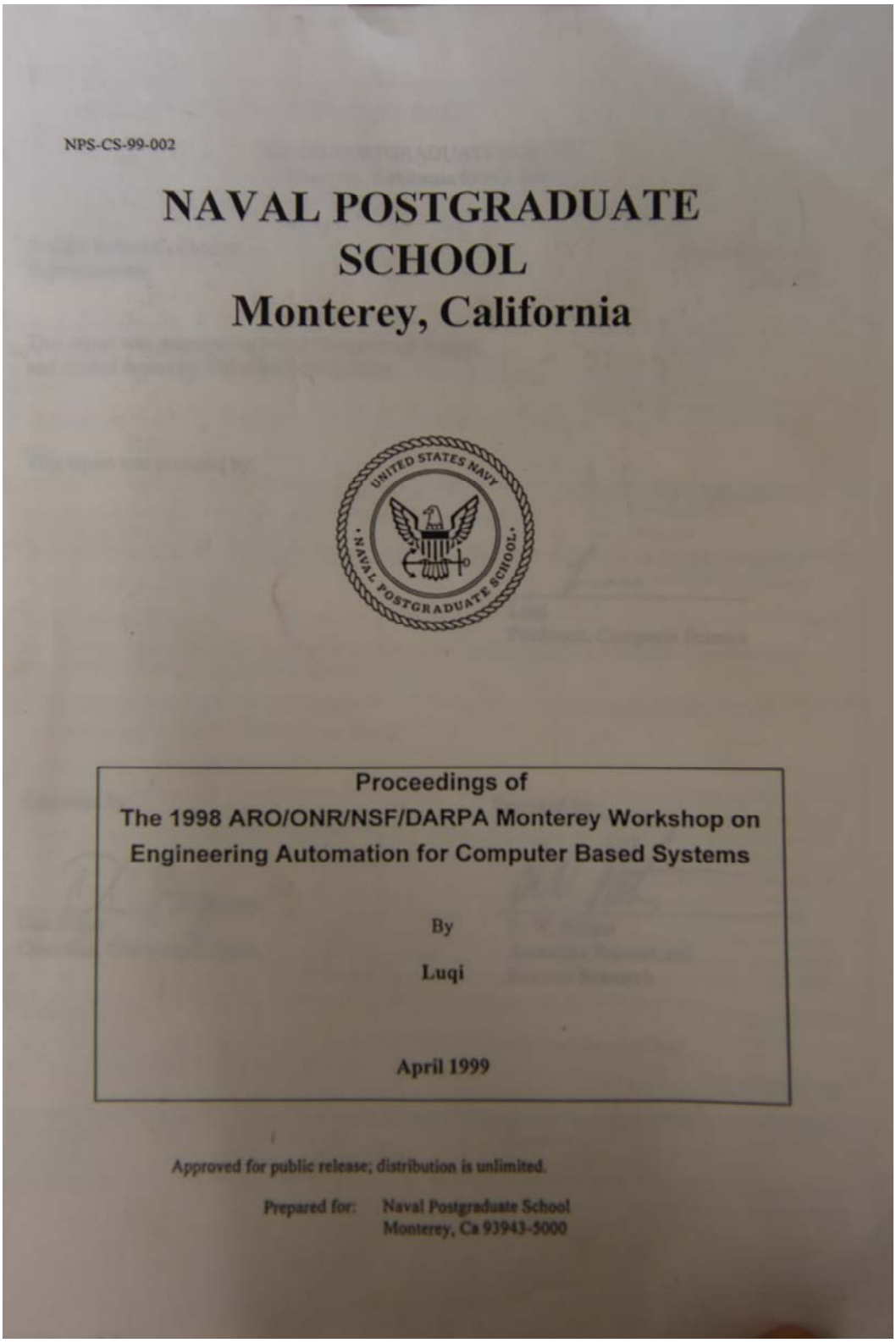
Evaluation and Case Studies

Figure 44 - Table of Contents, 5th Monterey Workshop

Tool Support and Prototyping

Figure 45 - Table of Contents, 5th Monterey Workshop (cont.)

G. 6TH MONTEREY WORKSHOP: ENGINEERING AUTOMATION FOR COMPUTER BASED SYSTEMS (1998)



NPS-CS-99-002

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Proceedings of
The 1998 ARO/ONR/NSF/DARPA Monterey Workshop on
Engineering Automation for Computer Based Systems

By

Luqi

April 1999

Approved for public release; distribution is unlimited.

Prepared for: Naval Postgraduate School
Monterey, Ca 93943-5000

Figure 46 - Cover Page, 6th Monterey Workshop

Proceedings of the
1998 ARO/ONR/NSF/DARPA Monterey Workshop on
Engineering Automation for Computer Based Systems

Luqi
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5118

Abstract

The "Engineering Automation for Computer Based Systems" Workshop is the 6th in a series of Software Engineering workshops for formulating and advancing software engineering models and techniques, with the fundamental theme of increasing the practical impact of formal methods. Previous workshops have been devoted to "Real-time & Concurrent Systems", "Software Merging and Slicing", "Software Evolution", "Software Architecture", and "Requirements Targeting Software". A major goal for this series of workshops is to help focus the software engineering community on issues that are vital to improving the state of software engineering practice. This focus promotes consistency among diverse research directions that address different aspects of the same problem to facilitate future integration efforts.

The workshop represents a bridge between industry and academia. The material in these proceedings presents a balanced view of academic and industrial developments. Formalization is fundamental to the development of software engineering as an engineering discipline. The critical importance of formal models and formal methods is painfully clear when one considers the escalating demands for larger, more complex, reliable software systems.

Figure 47 - Abstract, 6th Monterey Workshop

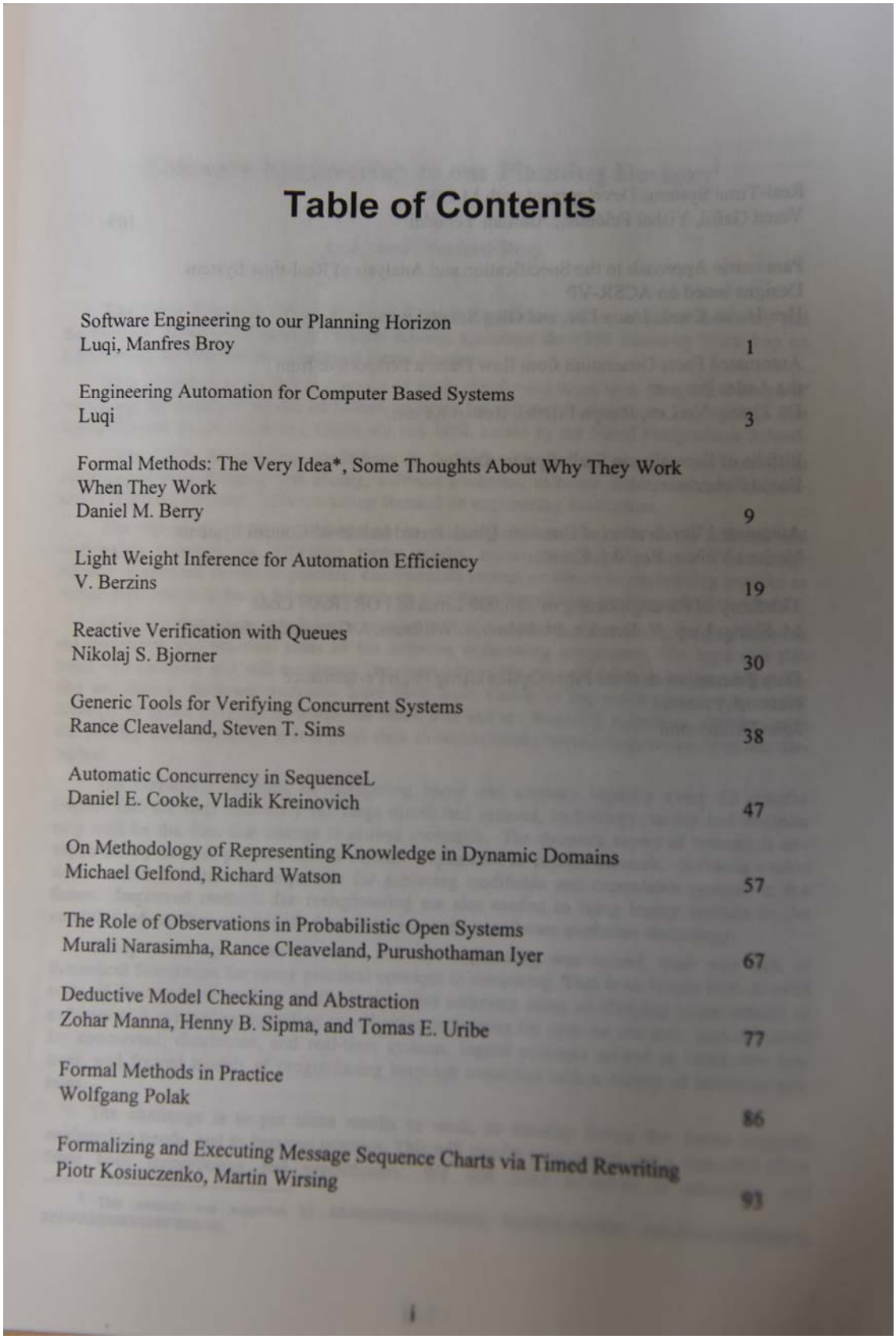


Table of Contents

Figure 48 - Table of Contents, 6th Monterey Workshop

Figure 49 - Table of Contents, 6th Monterey Workshop (cont.)

H. 7TH MONTEREY WORKSHOP: MODELING SOFTWARE SYSTEM STRUCTURES IN A FASTLY MOVING SCENARIO (2000)

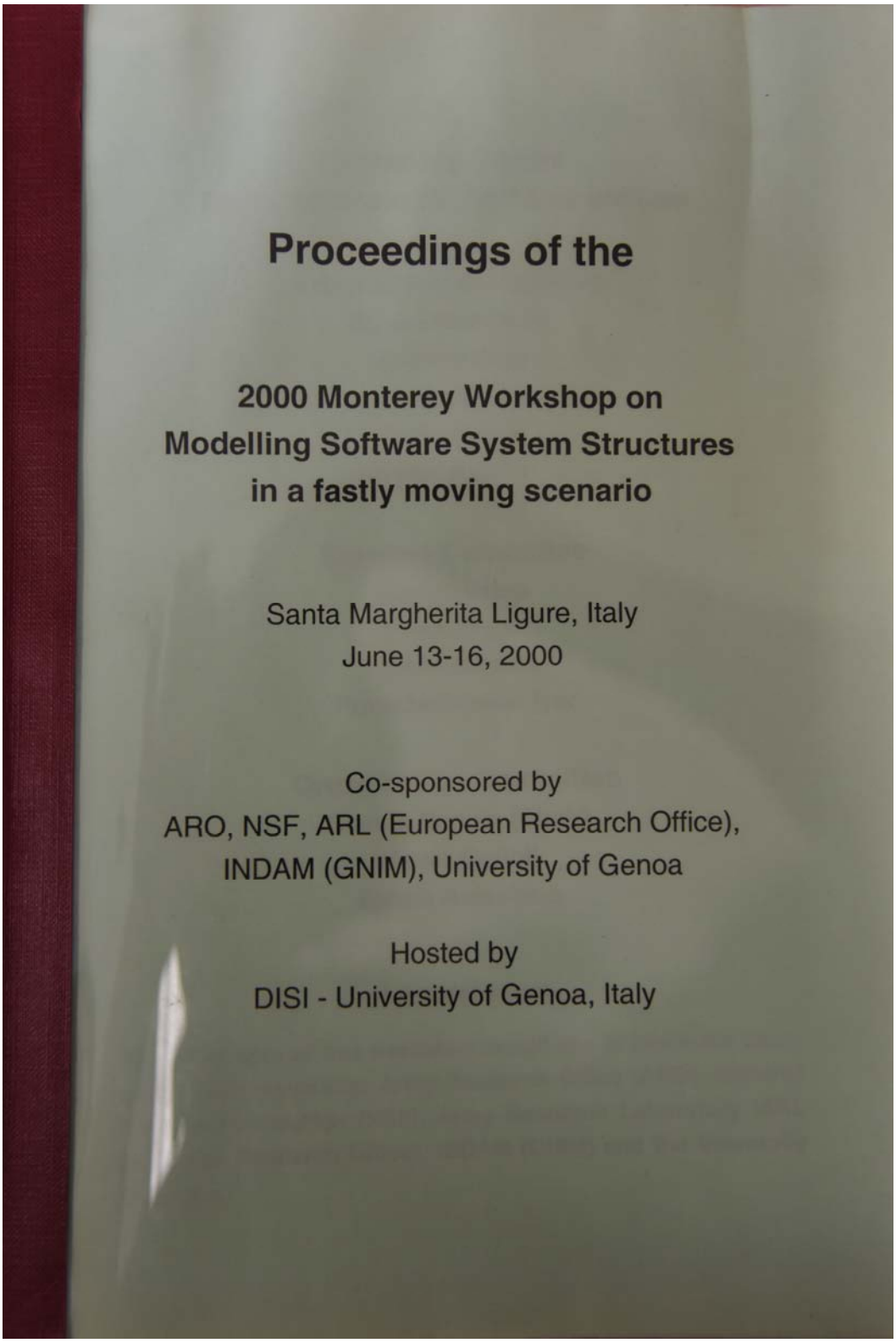


Figure 50 - Cover Page, 7th Monterey Workshop

The foreword of the proceedings for this workshop is reproduced below.

1. Foreword

The Workshop on Modelling Software System Structures in a fastly moving scenario was sponsored by the Army Research Laboratory European Research Office, U.S. Army Research Office, National Science Foundation, Istituto Nazionale di Alta Matematica/GNIM and Università di Genova. This workshop is the 7th in a series of Software Engineering workshops, called "Monterey Workshops" from the Monterey Naval Postgraduate School, where they originated under the initiative of Prof. Luqi. The general aim of these workshops is formulating and advancing software engineering models and techniques, with the fundamental theme of increasing the practical impact of scientifically well-founded techniques such as formal methods. Previous workshops have been devoted to "Real-time & Concurrent Systems", "Software Merging and Slicing", "Software Evolution", "Software Architecture", "Requirements Targeting Software", and "Engineering Automation for Computer Based Systems". A major goal for this series of workshops is to help to focus the software engineering community on issues that are vital to improving the state of software engineering practice, bringing together American and European leading scientists actively engaged in the area.

The context for the workshop initiative is nicely set up in the words from the PITAC (the USA President's Information Technology Advisory Committee) 1998 Interim Report.

> "The demand for software has grown far faster than the resources we have to produce it. The result is that desperately needed software is not being developed. Furthermore, the nation needs software that is far more usable, reliable, and powerful than what is being produced today."
>
> "...it has become clear that the processes of developing, testing, and maintaining software must change. We need scientifically sound approaches to software development that will enable meaningful and practical testing for consistency of specifications and implementations."
>
> Unfortunately, as the same interim report emphasizes, "current support (for research) is taking a short-term focus, looking for immediate returns, rather than investigating high-risk long-term technologies".

As a consequence, there is a danger of even widening the gap between fundamental research and current (not always best-) practice. Indeed, together with long

standing problems, such as the quest for software reliability, we are facing the need and partly the emergence of radically different ways of producing software.

The 7th Workshop, continuing the effort to bring together pragmatic and foundational research in software engineering, has primarily focused the attention on the major issues characterizing the new and rapidly evolving scenario of software development, such as the emphasis on high-level architectural aspects and the component-based and web-based software development. Together with proposing new concepts and techniques, another major achievement of the workshop has been the demonstration that the wealth of past foundational research in SE can be uplifted to handle some, if not all, of the new problems posed, among others, by the different level of component and system granularity, the heterogeneity of components, the use of distribution and communication and the request for appropriate human-interface support.

The participation was well balanced, considering that the event took place in Italy: we had 38 participants, 15 from USA, 15 from Europe, 7 from Italy (including 3 local people) and 1 from Canada. Altogether there have been 29 talks and two panels, each with five participants. There has been a nice mix of technical talks and talks surveying/proposing hot topics. The discussion was quite alive and reached high peaks, especially in the discussions centered around Component based SE and the emergence of UML.

To mark the importance of the event, on Wednesday, at the official banquet, we have been honored by the presence of the President of the University of Genova, who welcomed us, also reacting positively to a nice dinner speech by Manfred Broy, who presented the motivation for the Workshop within the worldwide fastly moving scenario shaped by the IT explosion. Dr. John Zavada also spoke at the banquet presenting the goals of the research support provided by his office and expressing the opinion that this kind of meeting USA/EUROPE should be more frequent, because they offer the opportunity of merging different cultures. The difference in cultures was indeed clearly visible at the workshop, which however was already showing some remarkable convergence in attitude.

I think it is fair to summarize the overall feeling, saying that, as result of the workshop, everybody really got a picture of a fastly moving scenario and the many

problems we have to face rapidly to cope with the pace in software development, as it was summarized by Luqi and Manfred Broy in the closing session and discussion and in the words of a postworkshop message by Dr. John Zavada (ARL/ERO): “I enjoyed the workshop and the discussions that we had. I think that I now have a better understanding of the issues facing software development.”

Egidio Astesiano

DISI - University of Genova

Via Dodecaneso, 35, 16146 Genova

ITALY

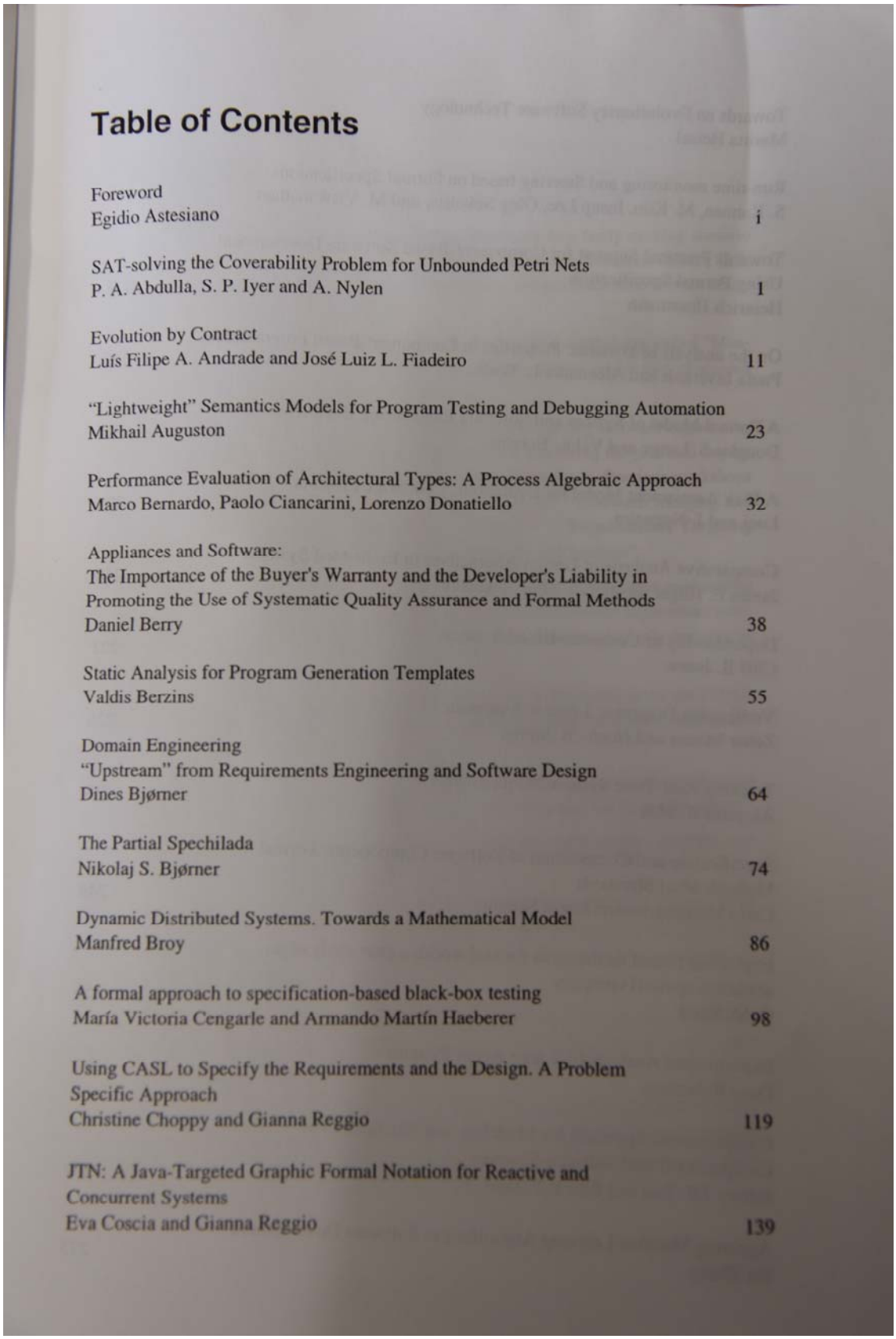


Table of Contents

Figure 51 - Table of Contents, 7th Monterey Workshop

Figure 52 - Table of Contents, 7th Monterey Workshop (cont.)

I. 8TH MONTEREY WORKSHOP: ENGINEERING AUTOMATION FOR SOFTWARE INTENSIVE SYSTEMS INTEGRATION (2001)

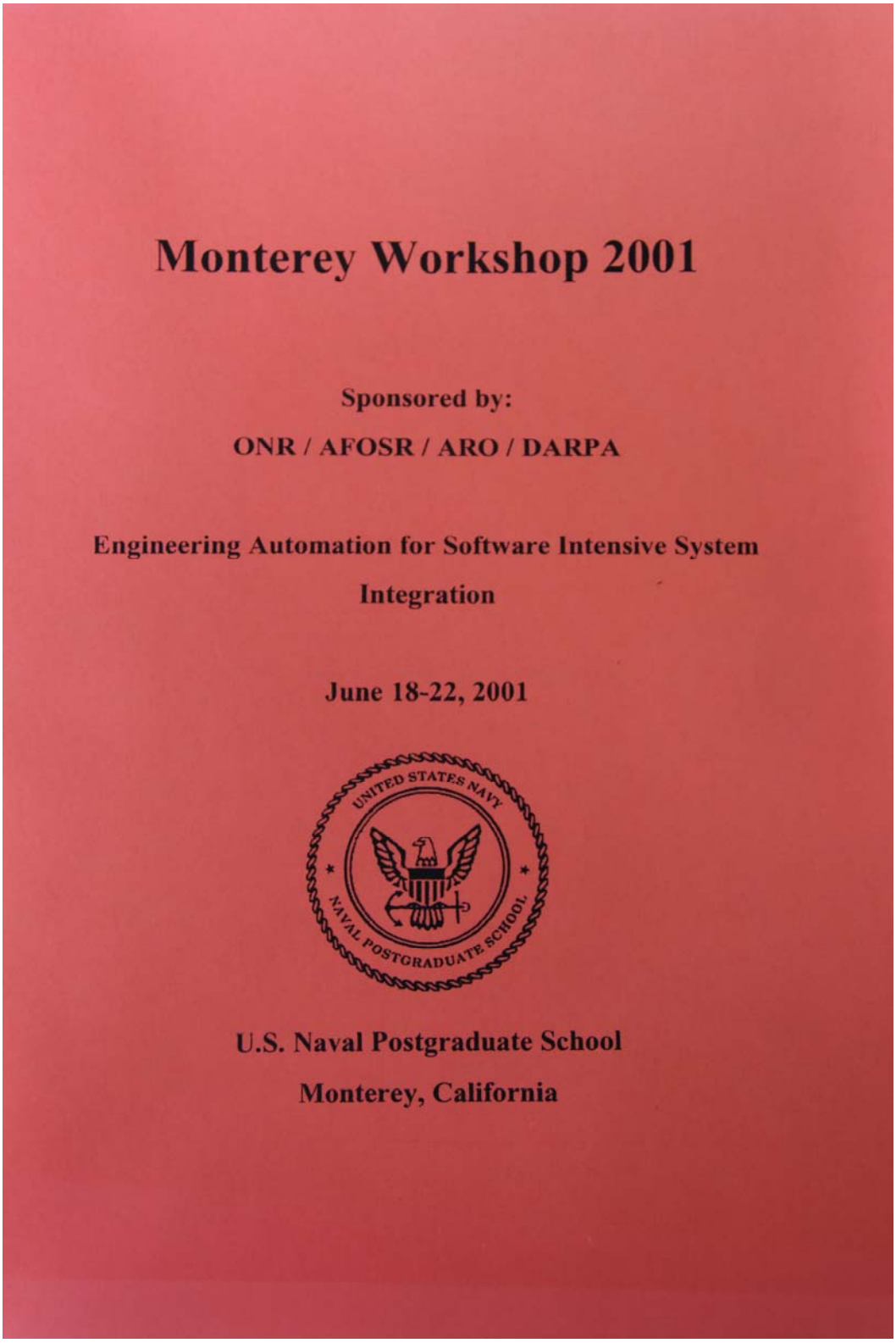


Figure 53 - Cover Page, 8th Monterey Workshop

Preface

Luqi

The 2001 Monterey Workshop on Engineering Automation for Software Intensive System Integration was sponsored by the Office of Naval Research, Air Force Office of Scientific Research, Army Research Office and the Defense Advance Research Projects Agency. It is our pleasure to thank the workshop advisory and sponsors for their vision of a principled engineering solution for software and for their many-year tireless effort in supporting a series of workshops to bring everyone together.

This workshop is the 8th in a series of International workshops. The workshop was held in Monterey Beach Hotel, Monterey, California during June 18-22, 2001. The general theme of the workshop has been to present and discuss research works that aims at increasing the practical impact of formal methods for software and systems engineering. The particular focus of this workshop was "Engineering Automation for Software Intensive System Integration". Previous workshops have been focused on issues including, "Real-time & Concurrent Systems", "Software Merging and Slicing", "Software Evolution", "Software Architecture", "Requirements Targeting Software" and "Modeling Software System Structures in a fastly moving scenario".

A major goal for this series of workshops is to encourage the software engineering community in general to improve interaction between researchers and engineering practitioners. The workshop has long established itself as a summit where researchers from academics and industries can exchange recent results, assess their significance and earn motivation for transferring the relevant results to practice. This indeed is a forum where software engineers may communicate current problems in engineering practice to researchers and help focus to bridge the gap between the theoretical and practical sides of the subject.

It is no longer the case that theoretical foundations for computing are lacking. However, keeping in mind the challenge to put these results to work, the formal aspects of computing cannot be studied in isolation in the context of software engineering. The need to ensure that the assumptions on which formal models are based are consistent with the situations encountered in practical applications puts interdisciplinary requirements on researchers and lends importance to interactions between experts from heterogeneous backgrounds.

This year, apart from the distinguished panel of invited speakers, we have accepted contributed papers mainly to encourage the emerging researchers in software engineering. This has widened the scope of discussion and the sessions were highly interactive and rich with intellectual frictions in opinion from a broad range of experts. Members of the academic, government, military and commercial world exchanged their vision, insight and concerns on many important issues. I hope that the workshop has made another step to reduce the gap between theory and practice of software engineering.

Figure 54 - Page vi, 8th Monterey Workshop

Content

Figure 55 - Table of Contents, 8th Monterey Workshop

Figure 56 - Table of Contents, 8th Monterey Workshop (cont. i)

Figure 57 - Table of Contents, 8th Monterey Workshop (cont. ii)

J. 9TH MONTEREY WORKSHOP: RADICAL INNOVATIONS OF SOFTWARE AND SYSTEMS ENGINEERING IN THE FUTURE (2002)

1. Introduction

During the last decade object-orientation was the driving factor for new system solutions in many areas ranging from e-commerce to embedded systems. New modeling languages such as UML, new programming languages such as Java and CASE tools have considerably influenced the system development techniques of today and will remain key techniques for the near future. However, actual practice shows many deficiencies of these new approaches:

- There is no proof and no evidence that software productivity has increased with the new methods;
- UML has no clean scientific foundations which inhibit the construction of powerful analysis and development tools;
- support for mobile distributed system development is missing;
- for many applications, object-oriented design is not suited to produce clean well-structured code as many applications show.

As a consequence, there is an urgent need for new "post object-oriented" software engineering and programming techniques.

This workshop was the 9th in a series of Software Engineering workshops for formulating and advancing software engineering models and techniques, with the fundamental theme of increasing the practical impact of formal methods. Previous workshops have been devoted to "Real-time & Concurrent Systems", "Software Merging and Slicing", "Software Evolution", "Software Architecture", "Requirements Targeting Software", "Engineering Automation for Computer Based Systems", "Modeling Software System Structures in a Fastly Moving Scenario" and "Engineering Automation for Software Intensive System Integration". This workshop was held in Venice, Italy, and an article summarizing the workshop is reproduced in APPENDIX B: “Meeting in the Rain” – Monterey Workshop 2002 in Venice [24].

A major goal for this series of workshops is to help focus the software engineering community on issues that are vital to improving the state of software engineering practice, bringing together American and European leading scientists

actively engaged in the area. The aims of the Workshop, continuing the effort to bring together pragmatic and foundational research in software engineering, were threefold:

- to discuss the actual problems and short-comings in Software and Systems Engineering, to evaluate potential or partial solutions that have been proposed, and to analyze why some ideas were or were not successful;
- to propose and discuss in a pro-active way radically new innovations in Software and Systems Engineering and to present visionary and explorative perspectives and bold ideas for the modeling language, the programming language, the system development method and the system development process of tomorrow;
- to show how the wealth of past foundational research in Software Engineering can be uplifted to handle the new problems posed, among others, by the different level of component and system granularity, the heterogeneity of components, the use of distribution and communication and the request for appropriate human-interface support.

The workshop encouraged joint work leading to joint publications by researchers from different institutions by including a session focused on identifying opportunities for future collaboration and integration of complementary advances.

2. Topics Addressed

The workshop provided a bridge between industry and academia. The program provided a balanced view of academic research and industrial visions, developments and proposals. Contributions were sought in but not limited to the following areas:

- Analysis of actual problems in Systems and Software Engineering and their existing solutions
- New Paradigms for System Development Methods and Processes
- New Paradigms for Modeling and Specification Languages
- Post-Object-Oriented Programming Concepts
- Scientific and technological foundations of System Development
- Innovative Tool Support for System Development

Scientific development has a large amount of inertia and it takes effort sustained for a long time to produce changes in direction. Over the years, the Monterey Workshop has succeeded in changing the attitude of the top researchers in the field towards practical relevance, and has initiated technology transfer by encouraging researchers to apply revolutionary ideas and methods developed at other research centers. The workshop has

created stronger cooperation between U.S. and European researchers and fostered cooperation and collaboration among researchers with disparate points of view. Collaboration between researchers from different backgrounds requires a long time for people from different schools of thought to understand each other's work and to find common ground for integration and fruitful collaboration. Some examples new collaborations between Prof. Manna's group at Stanford and Prof. Broy's group at Technical University of Munich, as well as between Prof. Auguston at New Mexico State University and Prof. Luqi's Software Engineering Group at NPS.

3. Committees

The steering committee was as follows:

- Egidio Astesiano, University of Genova, Italy
- Manfred Broy, Technical University of Munich, Germany
- David Hislop, US Army Research Office, USA
- Luqi, US Naval Postgraduate School, USA
- Zohar Manna, Stanford University, USA

The program committee was as follows:

- Martin Wirsing (Chair), University of Munich, Germany
- Mikhail Auguston, New Mexico State University, USA
- Simonetta Balsamo, Università Ca' Foscari di Venezia, Italy
- Dan Berry, University of Waterloo, Canada
- Valdis Berzins, US Naval Postgraduate School, USA
- Swapan Bhattacharya, Jadavpur University, India
- Barrett Bryant, University of Alabama at Birmingham, USA
- Rance Cleveland, SUNY at Stony Brook, USA
- Peter Freeman, Georgia Institute of Technology, USA
- Marie-Claude Gaudel, University of Paris-Orsay, France
- Carlo Ghezzi, Politecnico di Milano, Italy
- Purush Iyer, North Carolina State University, USA
- Oscar Nierstrasz, Bern University, Switzerland
- Axel van Lamsweerde, Université Catholique de Louvain, Belgium
- Jeanette Wing, Carnegie Mellon University, USA

K. 10TH MONTEREY WORKSHOP: SOFTWARE ENGINEERING FOR EMBEDDED SYSTEMS: FROM REQUIREMENTS TO IMPLEMENTATION (2003)

1. Introduction

The theme for this workshop was "Software Engineering for Embedded Systems: From Requirements to Implementation." The workshop was the 10th in a series of workshops, initiated in 1993 and devoted to exploring the critical problems associated with cost-effective development of high-quality software systems. The previous workshops were focused on the following themes:

- Real-Time and Concurrent Systems
- Software Merging and Slicing
- Software Evolution
- Software Architecture
- Requirements Targeting Software
- Engineering Automation for Computer Based Systems
- Modeling Software System Structures in a Fastly Moving Scenario
- Engineering Automation for Software Intensive System Integration
- Radical Innovations of Software and Systems Engineering in the Future

This workshop brought together researchers and practitioners working on all aspects of design, implementation, and evaluation of software for embedded systems to exchange ideas on creating fully integrated software engineering solutions, from requirements capture to implementation. The goal was two-fold: first, to assess the state-of-the-research in current "point solution" specific problems in this domain; and second, to define a roadmap for future research aimed at bridging the gaps between current tools and methodologies and attempting to articulate a complete set of technology requirements. The first objective was addressed through presentations by leaders in a variety of relevant areas. The second objective was addressed through presentations of position papers and extensive open discussion periods.

2. Topics Addressed

Workshop topics included, but were not limited to:

- Exploiting domain-information for scalable design analysis

- Requirements capture (from informal to formal)
- Model-based design
- Refinement checking of design-implementation correspondence
- Special features of embedded systems
- Synthesis of (partial) implementations from designs/specifications
- Verification and validation
- Technology for producing requirements and design documentation
- Technology for semi-automated translation of legacy/domain languages to design/analysis platform needs
- Engineering user interfaces

A sample of focus-questions for the workshop are the following:

- How can embedded systems models be exploited throughout the software development process?
- How can view-specific models of embedded systems be integrated?
- How can requirements be carried effectively through the entire design life cycle?
- How can engineering judgments and insight be married with formal methods to solve practical large-scale problems?
- How can model-based methods work for the implementation since there's a tendency to change code, not models, when systems are being debugged?

3. Committees

The steering committee was as follows:

- Egidio Astesiano, University of Genova, Italy
- Manfred Broy, Technical University of Munich, Germany
- David Hislop, US Army Research Office, USA
- Luqi, US Naval Postgraduate School, USA
- Zohar Manna, Stanford University, USA

The program committee was as follows:

- Matthew Dwyer (Co-Chair), Kansas State University
- Bruce Krogh (Co-Chair), Carnegie Mellon University
- Insup Lee (Co-Chair), University of Pennsylvania
- Daniel M. Berry, University of Waterloo
- Valdis Berzins, Naval Postgraduate School
- Lori Clarke, University of Massachusetts

- Elsa Gunter, New Jersey Institute of Technology
- Fabrice Kordon, University Pierre & Marie Curie France
- Martha Palmer, University of Pennsylvania
- Anna Philippou, University of Cyprus
- Shaz Qadeer, Microsoft Research
- Owen Rambow, Columbia University
- Steve Ray, National Institute of Standard and Technology
- Scott Smolka, SUNY at Stony Brook
- Paul Young, United States Navy Academy

L. 11TH MONTEREY WORKSHOP: SOFTWARE ENGINEERING TOOLS: COMPATIBILITY AND INTEGRATION (2004)

1. Introduction

Software is the new physical infrastructure of the information age. It is fundamental to economic success, scientific and technical research and national security. Our current ability to construct the large and complex software systems demanded for continued economic and military success are inadequate. A lack of tool integration has been recognized as a major impediment to the construction of complex computer systems, especially in the development of embedded systems, where the design of a single artifact may involve up to 50 different tools, including design, functional analysis, performance analysis, etc. tools. Tool interoperability is an important prerequisite for increasing software productivity and reliability. This workshop sought to address these issues.

2. Topics Addressed

- Open tool integration frameworks / Repositories of interoperable tools;
- Systematic solutions for tools integration;
- Web-based tool integration;
- Tool integration patterns;
- Tool architectures;
- User perspectives on tool integration
- End-user programmability / customization;
- Model-driven architectures;

- Model transformation technologies;
- Domain-driven software development frameworks;
- Generic vs domain-specific solutions;
- Standardized modeling environments;
- Hierarchy of layers of abstractions;
- Software analysis and design tools;
- Component-based systems;
- Interface theories;
- Global information utilities/services

3. Committees

The Monterey Workshop Community has a long history of productive, high-quality workshops, directed by an international steering committee consisting of leaders in the field of software engineering and formal methods under the visionary leadership of David Hislop from the Army Research Office:

- Egidio Astesiano, University of Genova, Italy
- Manfred Broy, Technical University Munich, Germany
- David Hislop, Army Research Office
- Hermann Kopetz, Vienna University of Technology
- Fabrice Kordon, University of Pierre & Marie Curie, Paris, France
- Luqi, Naval Postgraduate School, Monterey, California
- Zohar Manna, Stanford University
- Janos Sztipanovits, Vanderbilt University

The program chairs were assisted by an international program committee consisting of the following researchers from the United States and Europe:

- Lori Clarke, University of Massachusetts
- Carlos Delgado Kloos, University Carlos III of Madrid, Spain
- Matt Dwyer, Kansas State University
- Carlo Ghezzi, Polytechnic of Milan, Italy
- Kane Kim, University of California at Irvine
- Bruce Krogh, Carnegie Mellon University
- Insup Lee, University of Pennsylvania

- Tom Maibaum, King's College, London, UK
- Ugo Montanari, University of Pisa, Italy
- Olaf Owe, University of Oslo, Norway
- Wolfgang Pree, University of Salzburg, Austria
- Henny Sipma, Stanford University
- Martin Wirsing, Ludwig Maximilians University Munich, Germany

M. 12TH MONTEREY WORKSHOP: WORKSHOP ON NETWORKED SYSTEMS: REALIZATION OF RELIABLE SYSTEMS ON TOP OF UNRELIABLE NETWORKED PLATFORMS (2005)

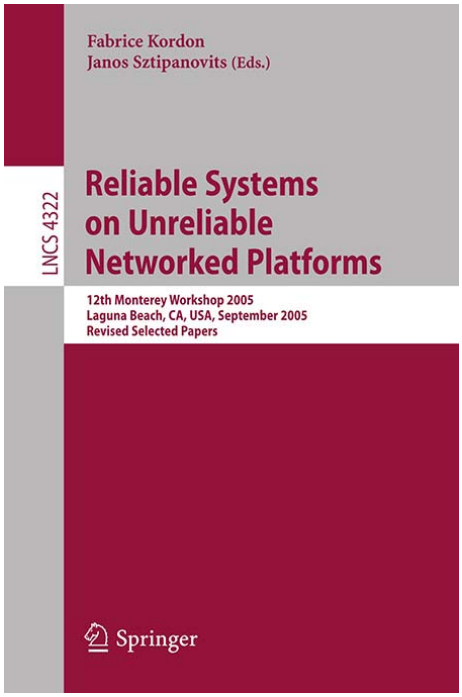


Figure 58 - Lectures Notes on Computer Science, vol. 4322

1. Introduction

Networked computing is increasingly becoming the universal integrator for large-scale systems in defense and industry. In addition, new generation of wireless networked embedded systems rapidly create new technological environments that imply complex interdependencies amongst all layers of societal-scale critical infrastructure, such as transportation, energy distribution and telecommunication.

This trend makes reliability and safety of networked computing a crucial issue and a technical precondition for building software intensive systems that are robust, fault

tolerant, and highly available. The workshop focused on new, promising directions in achieving high software and system reliability in networked systems.

In the years preceding this workshop was held alternately in Europe and the United States. This workshop was held in Laguna Beach, CA during September 22 - 24. The workshop was chaired by Prof. Fabrice Kordon of the Université Pierre & Marie Curie, France and Prof. Janos Sztipanovits of Vanderbilt University, USA.

2. Topics Addressed

Software is the new infrastructure of the information age. It is fundamental to economic success, scientific and technical research and national security. Our current ability to construct the large and complex software systems demanded for continued economic progress is inadequate. The workshop discussed a range of challenges in networked systems that require further major advances in software and systems technology:

- **System Integration and Dynamic Adaptation.** A new challenge in networked systems is that stable application performance needs to be maintained in spite of the dynamically changing communication and computing platforms. Consequently, the run-time architecture must include active control mechanisms for adapting the Please purchase PDF Split-Merge on www.verypdf.com to remove this watermark. Preface VII system/software components to changing conditions. Global system characteristics need to be achieved by increased run-time use of reflection (systems that utilize their own models), advanced interface modeling, self-adaptation, and self-optimization.
- **Effects of Dynamic Structure.** The structure of networked systems is complex and highly dynamic. Because systems are formed by ad hoc networks of nodes and connections, they lack fine-grain determinism for end-to-end behaviors that span subsystem and network boundaries. In addition, there are end-to-end system qualities such as timeliness and security that can only be evaluated in this dynamically integrated context.
- **Effects of Faults.** Faults and disruptions in the underlying communication and computing infrastructure are the normal events. Since well-understood techniques for fault-tolerant computing, such as n-modular redundancy, are not applicable in the dynamically changing networked architecture, new technology is required for building safe and reliable applications on dynamic, distributed platforms.
- **Design for Reliability.** Although there are varieties of metrics and established practices for characterizing the expected failure behavior of a system after it is fielded and there are established practices for specifying the desired reliability of a system, the evaluation of system or software reliability prior to fielding is a significant problem.

- **System Certification.** The process for certifying that a system meets specified reliability goals under the range of conditions expected in actual use currently involves exhaustive analysis of a system, including its development history and extensive testing. Current methods do not give systems engineers the confidence they would like to have in concluding that a system will have particular reliability characteristics.
- **Effects of Scale.** Another risk that overlays all proposed solutions is scale. Scale also addresses both run-time and design-time concerns. Typically, demonstrations are the convincing drivers to technology adoption. Demonstrations of new technologies however are usually small-scale, focused efforts. It is an open problem how to scale up a demonstration that addresses the number of nodes and connections, and the number of software developers, analysts, and integrators to provide enough proof to justify technology transition.

3. Committees

David Hislop continued his visionary leadership of the steering committee. The membership of the steering committee was as follows:

- Egidio Astesiano, University of Genova, Italy
- Manfred Broy, Technical University Munich, Germany
- David Hislop, Army Research Office
- Hermann Kopetz, Vienna University of Technology
- Fabrice Kordon, University of Pierre & Marie Curie, Paris, France
- Luqi, Naval Postgraduate School, Monterey, California
- Zohar Manna, Stanford University
- Janos Sztipanovits, Vanderbilt University

The program chairs were assisted by an international program committee consisting of the following researchers from the United States and Europe.

- Carlos Delgado, Kloos University Carlos III of Madrid, Spain
- Bertil Folliot, University of Pierre & Marie Curie, Paris, France
- Tom Henzinger, Ecole Polytechnique Federale de Lausanne, Switzerland
- Kane Kim, University of California at Irvine
- Insup Lee, University of Pennsylvania
- Chenyang Lu, Washington University
- Tom Maibaum, King's College, London, UK
- Ugo Montanari, University of Pisa, Italy
- Laurent Pautet, Telecom Paris

- Wolfgang Pree, Univ. Salzbourg
- Doug Schmidt, Vanderbilt University - ISIS, USA

N. 13TH MONTEREY WORKSHOP: COMPOSITION OF EMBEDDED SYSTEMS: SCIENTIFIC AND INDUSTRIAL ISSUES (2006)

Figure 59 - Lecture Notes on Computer Science, vol. 4888

1. Introduction

Embedded Systems are becoming more and more present in our day-to-day life. In particular, they are the foundation of numerous critical functions, such as in automotive, satellite or aircraft systems. Such systems are difficult to build since reliability must cope with numerous constraints such as memory footprint or strict time schedule. Moreover, embedded systems are now combined together to build larger ones also as networked systems (such as sensor networks). We are now in a process where the area of DRE (Distributed Real-time Embedded Systems) is coming out from very specialized areas to become natural in normal applications.

However, such systems require even more specific techniques to be developed. These techniques come from both the real-time/embedded and the distributed communities. In fact, numerous new problems must be addressed: integration of components, managing concurrency, new definition of the "real-time" concept (to support

distribution), new runtime infrastructures, development methodologies, safe and deterministic behavior, etc.

This workshop addressed scientific and practical aspects of the development of distributed real-time embedded systems. The workshop was the 13th in the Monterey workshop series, and was held in Paris, France on October 16-18, 2006. The workshop was chaired by Prof. Fabrice Kordon of the Université Pierre & Marie Curie, France and Prof. Oleg Sokolsky of the University of Pennsylvania, USA. Prof. Kordon was also responsible for the local arrangements.

2. Topics Addressed

- **Model-driven development for DRE systems.** Modeling and model-driven development (MDD) are of particular importance for DRE systems, because of their dependence on continuously evolving environments and strict requirements that need to be specified precisely for testing and verification.
- **Balancing cost and assurance in DRE systems.** High assurance comes at a high cost. System developers need to balance development costs and assurance levels depending on the criticality of particular system aspects. This area has not received enough attention from the research community and system developers lack proper tools to reason about such trade-offs.
- **Domain-specific languages for DRE systems.** Domain-specific languages (DSL) allow designers to represent systems directly using concepts from their application domains. Because of this, models and designs are easier to understand and validate, increasing confidence in the system. Research on DSL has been very active recently, yet many open questions remain, including semantic definitions for DSLs and correctness of model transformations with respect to the language semantics.
- **Composition of real-time components.** Timing and resource constraints, prevalent in DRE system development, make composition much more difficult. Component interfaces that are the basis for system integration now have to expose not only the input and output behaviors of the component, but also its resource demands. Formalisms that are used to reason about composition need to be able to capture the notion of resources and resource scheduling.
- **Fault tolerance for DRE.** Dealing with emergency situations is a major part of the DRE operation. The handling of faults and other abnormal events consumes a major portion of the system development efforts and represents the vast majority of code in a system implementation. At the same time, most model-driven approaches concentrate on the functional aspects of system behavior and the nominal environment.

3. Committees

The membership of the steering committee was as follows:

- David Hislop, Army Research Office
- Luqi, Naval Postgraduate School, Monterey, California
- Zohar Manna, Stanford University
- Manfred Broy, Technical University Munich, Germany
- Egidio Astesiano, University of Genova, Italy
- Fabrice Kordon, University of Pierre & Marie Curie, Paris, France
- Janos Sztipanovits, Vanderbilt University
- Hermann Kopetz, Vienna University of Technology

The program chairs were assisted by an international program committee consisting of the following researchers from the United States and Europe:

- Beatrice Berard, Universite Dauphine, France
- Valdis Berzins, Naval Postgraduate School, USA
- Juan de la Puente, Universidad Politecnica de Madrid, Spain
- Gabor Karsai, Vanderbilt Universiy, USA
- Insup Lee, Pennsylvania University, USA
- Edward Lee, University of California at Berkeley, USA
- Tom Maibaum, King's College, London, UK
- Joseph Sifakis, Verimag, France
- Henny Sipma, Stanford, USA
- Francois Terrier, CEA-LIST, France

O. 14TH MONTEREY WORKSHOP: WORKSHOP ON INNOVATIONS FOR REQUIREMENTS ANALYSIS: FROM STAKEHOLDERS NEEDS TO FORMAL DESIGNS (2007)

1. Introduction

Errors or failures of software-based systems are due to a variety of causes, e.g. misunderstanding of the real world, erroneous conceptualization, or problems in representing concepts via the specification or modeling notations. Precise specification is a key success factor as are communication and the deliberation about whether the specification is right and whether it has been properly implemented. Not all stakeholders are familiar with the formal models and notations employed. Some important requirements might be difficult to quantify and/or express using formal languages, such

as the desire that a system should be user-friendly or easily maintainable. Better technologies for requirements analysis should be thus considered.

The majority of requirements are given in natural language, either written or orally expressed. Other requirements might also be visually expressed in terms of figures, diagrams, images or even gestures. Artificial-intelligence approaches might be used to develop prototypes, which can then be re-engineered using more conventional requirements technologies and safety assurance techniques. For example, we might employ large amounts of semantic and statistical data, knowledge bases and theorem provers to infer as much contextual information as possible from the (vague) textual or visual requirements. The automatic analysis of natural language expressions has not yet been fully achieved, and interdisciplinary methodologies and tools are needed to successfully go from natural language to accurate formal specifications. Conformance of a system implementation to its requirements requires dynamic and efficient communication and iteration among system stakeholders. It is in supporting this process, and not in supplanting it, that innovative approaches to requirements analysis need to find their proper role.

We wanted to gain a better understanding about how to deal with natural language as the vehicle from which we derive system/software requirements, how to use intelligent agents as entities to facilitate semi-automatic requirements-documentation analysis, and how to build automatic systems to aid in requirements/specifications elicitation.

A good case study for these issues was to consider how to extract a conceptual model of the goals and requirements of the software needs discussed in a blog. As blogs are unstructured natural language, it represents one of the most difficult challenges for natural language processing.

In this workshop we brought together experienced researchers who have been involved in the specification, design, development and validation/verification of software-intensive systems, both for software engineering and for natural language processing. The overall aim was to exchange ideas for continued research in the intersection of these two areas and to reduce the gap between theory and practice. The workshop was chaired by Professors Luqi (Naval Postgraduate School, Monterey, US) and Kordon (Université Pierre & Marie Curie, Paris, France).

2. Topics Addressed

Workshop topics included but were not limited to:

- Techniques and tools for rapid/iterative requirements elicitation, refinement, and validation.
- Methods, modeling and cost of AI based systems.
- Application of innovative requirements engineering techniques to large, complex systems.
- Natural language techniques for requirements elicitation.
- From natural language and visual requirements to formal models and languages.
- Semi-automated co-development of various system documents requirements through fielded system.

3. Committees

The membership of the steering committee was as follows:

- David Hislop - Army Research Office, US
- Luqi - Naval Postgraduate School, Monterey, California, US
- Zohar Manna - Stanford University, California, US
- Manfred Broy - Technical University Munich, Germany
- Egidio Astesiano - University of Genova, Italy
- Fabrice Kordon - University of Pierre & Marie Curie, Paris, France
- Janos Sztipanovits - Vanderbilt University, Nashville, Tennessee, US
- Hermann - Kopetz Vienna University of Technology, Vienna, Austria

The program committee was chaired by Craig Martell, Naval Postgraduate School and Barbara Paech, University of Heidelberg. The chairs were assisted by an international program committee consisting of the following researchers from the United States and different countries in Europe:

- D. Berry - University Waterloo, Canada
- C. Choppy - University Paris XIII, France
- S. Clark - Oxford University, UK
- L. Clarke - University of Massachusetts, USA
- R. Cleveland - University of Maryland, USA
- V. Gervasi - University of Pisa, Italy
- A. Joshi - University of Pennsylvania, USA

- K. Kim - University of California, Irvine, USA
- L. Kof -Technical University of Munich, Germany
- F. Kordon - University P. & M. Curie, France
- B. Kraemer, FernUniversitt, Germany
- M. Marcus - University of Pennsylvania, USA
- B. Nuseibeh - The Open University, UK
- M. Rodriguez, National Research Council, USA
- S. Shatz - University of Illinois at Chicago, USA
- P. Sheu, University of California, Irvine, USA

P. 15TH MONTEREY WORKSHOP: FOUNDATIONS OF COMPUTER SOFTWARE, FUTURE TRENDS AND TECHNIQUES FOR DEVELOPMENT (2008)

1. Introduction

Computer systems are required to manage very large numbers of data and pieces of information in a distributed way. While complex computations are performed, there is a need to provide relevant and helpful displays to human actors so as to ease some tasks as interpretation, decision making, etc.

We wanted to explore how the foundations and techniques of computer software should be adapted to comply with such a challenge. This involved the whole software life cycle, starting from specification and analysis, design and the choice of architectures, large scale, real-world software development, code generation and configuration/deployment. In this context, certification and security became critical issues.

Decision making requires such mechanisms as automatic classification or clustering, high level visualization and navigation mechanisms, and may involve also market mechanisms, e.g. economic mechanisms for control of complex systems behavior. The workshop was chaired by Professors Tadeusz Dobrowiecki (Budapest University of Technology and Economics) and Janos Sztipanovits (Vanderbilt University, USA).

2. Topics Addressed

Workshop topics included but were not limited to:

- specification, analysis and design, formal methods

- certification, relationships between code and specification, security
- code generation, configuration, deployment, software product lines, MDA (Model Driven Architectures), SOA (Services Oriented Architectures)

3. Committees

The membership of the steering committee was as follows:

- David Hislop - Army Research Office, US
- Luqi - Naval Postgraduate School, Monterey, California, US
- Zohar Manna - Stanford University, California, US
- Manfred Broy - Technical University Munich, Germany
- Egidio Astesiano - University of Genova, Italy
- Fabrice Kordon - University of Pierre & Marie Curie, Paris, France
- Janos Sztipanovits - Vanderbilt University, Nashville, Tennessee, US
- Hermann Kopetz - Vienna University of Technology, Vienna, Austria

The program committee was chaired by Christine Choppy (Université Paris XII), France). She was assisted by an international program committee consisting of the following researchers from the United States and different countries in Europe:

- Wolfgang Pree (University of Salzburg, Austria)
- Oleg Sokolsky (University of Pennsylvania, USA)
- Kurt C. Wallnau (Carnegie Mellon University, USA)

Q. 16TH MONTEREY WORKSHOP: MODELING, DEVELOPMENT AND VERIFICATION OF ADAPTIVE COMPUTER SYSTEMS: THE GRAND CHALLENGE FOR ROBUST SOFTWARE (2010)

1. Introduction

Figure 60 - 16th Monterey Workshop Poster

Extremophiles are some of Mother Nature's crowning achievements. These microorganisms thrive in the cruelest of places; from acid pits to burning hot abysses to radioactive wastelands. They brilliantly demonstrate the robustness of biological systems even in the harshest conditions. More generally, extremophiles demonstrate just how robust and adaptive a complex system can be. How do our software systems compare to their biological counterparts? There is no competition: our systems lose in terms of robustness, adaptability, and dependability. For example, the desktop computer is an idyllic environment for software systems, yet desktop software fails regularly. Failure is so predictable that most commercial software maintains an "umbilical cord" to receive updates and patches. Unlike biological systems, we can never cut this umbilical cord.

Importantly, software is not limited to the desktop environment, but must persevere in less hospitable places, such as the internet and safety critical systems. Recent

studies have shown that failure in these environments is an omnipresent and growing threat to our economies, governments and societies. In this Monterey Workshop we challenged researchers to formulate a “digital” response to nature’s most robust systems. How can we use modeling and formal methods to architect adaptiveness into distributed and embedded systems, so that the “colony” can survive even though the individual might fail? How can we rethink the foundations of software systems and employ certification to improve dependability and robustness? What case studies illustrate where we succeed and how we go wrong? The workshop was chaired by Professors Luqi (Naval Postgraduate School, USA) and Fabrice Kordon (Université P. & M. Curie, France).

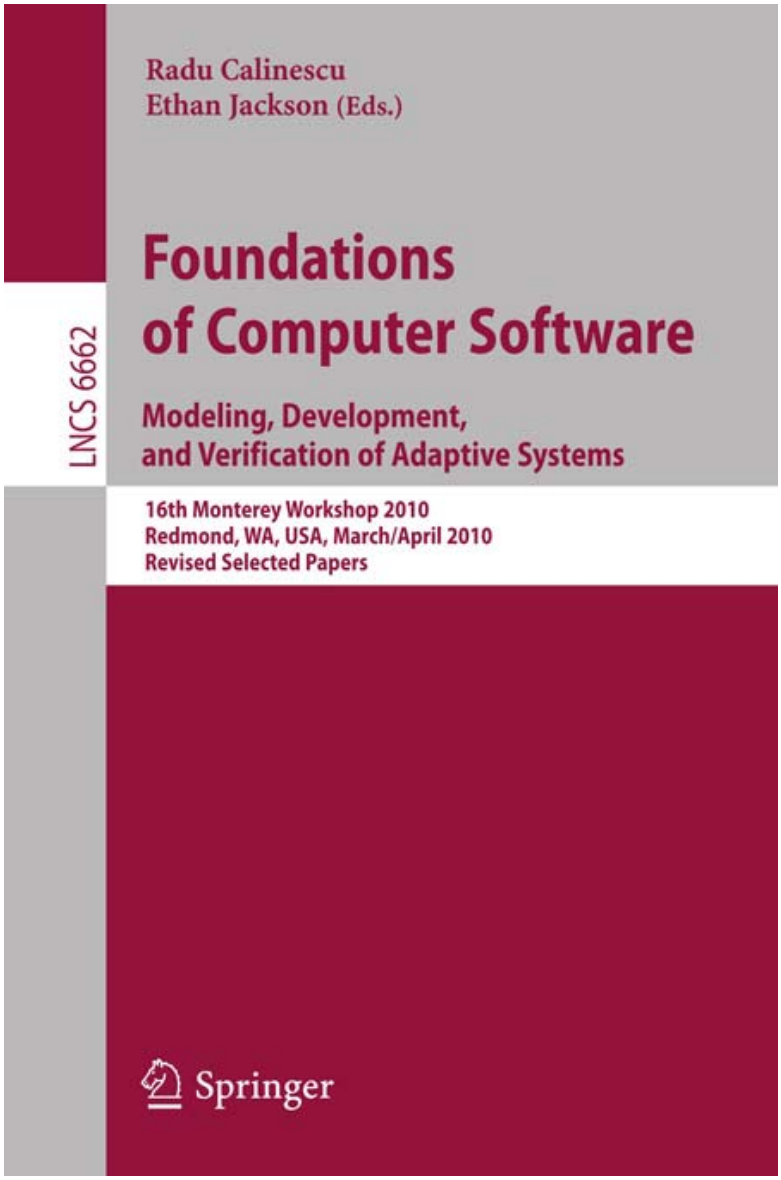


Figure 61 - Lecture Notes on Computer Science, vol. 6662

2. Topics Addressed

Workshop topics included but were not limited to:

- Formal Methods
- Certified Software
- Computational Biology
- Modelling and Architectures
- Distributed and Embedded Systems
- Adaptive Systems
- Systems of Systems

3. Committees

Program chairs were Radu Calinescu, University of Oxford (UK) and Ethan Jackson, Microsoft Research (USA). The steering committee was as follows:

- Egidio Astesiano - University of Genova, Italy
- Manfred Broy - Technical University of Munich, Germany
- Hermann Kopetz - Vienna University of Technology, Vienna, Austria
- Fabrice Kordon - University of Pierre & Marie Curie, Paris, France
- Luqi - Naval Postgraduate School, Monterey, California, USA
- Zohar Manna - Stanford University, California, USA
- Oleg Sokolsky - University of Pensylvania, USA
- Janos Sztipanovits - Vanderbilt University, USA

R. 17TH MONTEREY WORKSHOP: DEVELOPMENT, OPERATION AND MANAGEMENT OF LARGE-SCALE COMPLEX IT SYSTEMS (2012)

1. Introduction

Figure 62 - 17th Monterey Workshop Poster

Key critical applications in our economy and society rely increasingly on the dependable operation of large-scale complex IT systems. Such systems are created and evolved dynamically through the integration of independently built and controlled heterogeneous software components and infrastructure. As a result, traditional techniques, which assume complete control over the parts of a system, are inadequate in supporting assurable engineering of important safety-, security- and business-critical requirements.

This workshop explored the challenges associated with the development, operation and management of large-scale complex IT systems. The objectives of the workshop included the formulation of these challenges, the presentation and discussion of novel engineering techniques that have the potential to address them, and the formulation of a research agenda for this important research area. The workshop was chaired by

Professors Luqi (Naval Postgraduate School, USA) and Bill Roscoe (University of Oxford, UK).

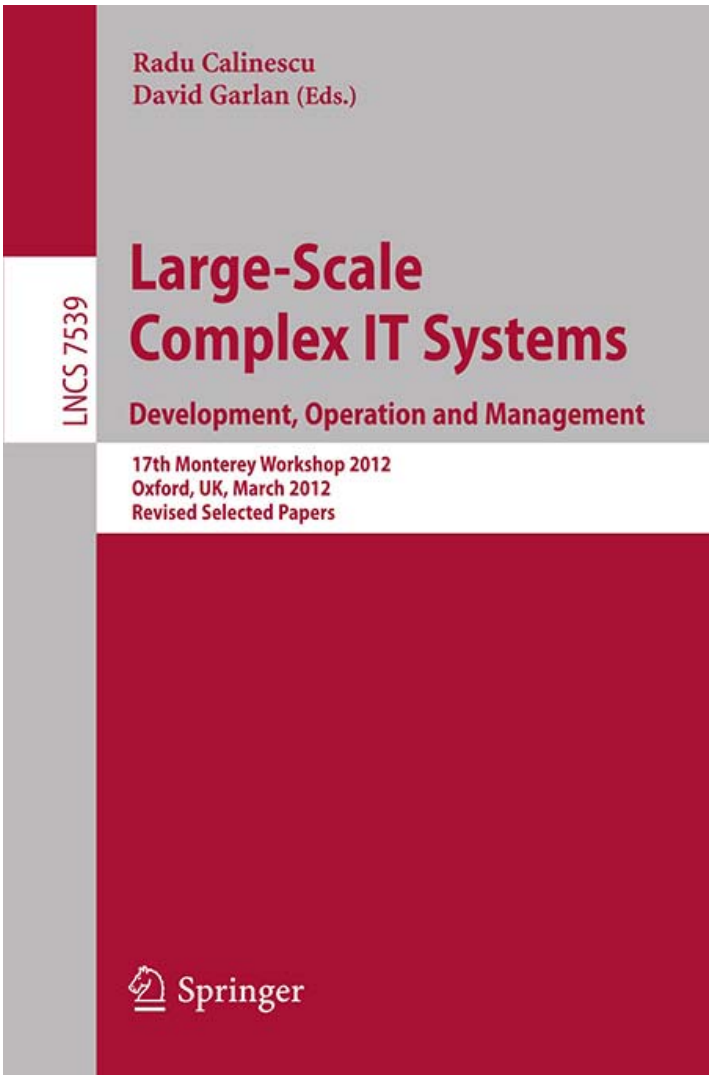


Figure 63 - Lecture Notes on Computer Science, vol. 7539

2. Topics Addressed

Workshop topics included but were not limited to:

- Compositional Specification, Analysis and Verification for Predictability
- Architectural Structuring of Cyber Systems
- Cyber-Security for Complex Systems
- Large-Scale Complex IT Systems in the Cloud
- Engineering High-Integrity Systems
- Socio-Technical Engineering
- Evolution of Long-Lived High Quality Cyber-Physical Systems

3. Committees

Program chairs were Radu Calinescu, University of Oxford (UK) and David Garlan, Carnegie Mellon University (USA). The steering committee was as follows:

- E. Astesiano - University of Genova, Italy
- M. Broy - Technical University Munich, Germany
- M. Griss, Carnegie Mellon University, USA
- F. Kordon - University P.& M. Curie, France
- Luqi - Naval Postgraduate School, USA
- W. Roscoe - Oxford University, UK

S. 18TH MONTEREY WORKSHOP: INTEGRITY OF INDUSTRIAL CONTROL SYSTEM & FUTURE COMMAND & CONTROL (FEB. 2016)

1. Introduction

Figure 64 - 18th Monterey Workshop Poster

Due to the sensitive nature of the material discussed in this workshop, access to the proceedings is limited. The proceedings summarized the progress to date on problems related to industrial control systems and command and control (C2), and implications for future needs of MARFORCYBER. The study addressed integrity of industrial control

systems, connections to networks, and C2, as well as advanced technologies needed for the future of C2 systems. This project addressed automated behavior-based computer network defense and cyber analysis techniques. The study explored impacts on C2 as cyber operations have evolved, and assessed implications for USMC cyber operations policy and doctrine. Project objectives were to examine automated behavior-based computer network defense, and critique cyber analysis techniques.

Mechanical and electrical control systems, and the communications network through which these components operate, are a high priority concern for MFCC leadership. Industrial Control Systems (ICS) represent an area of potential vulnerability. Embedded, microprocessor-controlled systems run mission-critical electromechanical processes for Marine Corps infrastructure. Failure due to cyber vulnerabilities or malware can be catastrophic for deployed forces. Additionally, the Marine Corps has unique requirements driven by the coordination and support provided to deployed Marine units. This extension of Marine networks and the wide variety of industrial applications leaves deployed forces in a highly vulnerable cyber posture. Metrics and evaluations processes are required to assess current Marine Corps ICS infrastructure, and to determine cyber vulnerabilities and potential solutions. Objectives of this project were to answer the following questions:

1. What does a secure posture look like when using industrial control systems and how do we know that this is secure?
2. How do we obtain reliable, effective, and secure measures of the performance of ICS supporting the Marine Corps?
3. What can be done to detect and prevent unauthorized access or influence on Marine Corps ICS?

C2 in cyber warfare requires command decisions under cognitive overload from an overwhelming volume and variety of multi-source data. We are investigating computer assisted methods for identifying the subset of available relevant to making decisions in particular situations, based on analysis of computer models of related cognitive tasks. Cognitive assistants offer computational and human-machine interfacing capabilities based on machine learning. Reasoning chains on large amounts of data provide cognition power that complement, augment, and scale human intelligence.

Human-Machine communication modes that effectively convey the identified relevant information to human decision makers are needed.

The workshop was chaired by Professors Luqi (Naval Postgraduate School, USA).

2. Topics Addressed

- Integrity of industrial control systems (ICS)
- Connections to networks
- Command and Control (C2)
- Future structure of C2 systems

3. Committees

Local chair was Lyla Englehorn. Keynote speaker was Janos Sztipanovitz.

T. 19TH MONTEREY WORKSHOP: CHALLENGES AND OPPORTUNITY WITH BIG DATA (OCT. 2016)

1. Introduction

2016 marked the 23rd anniversary for the Monterey Workshop series which started in 1993. Over the preceding quarter of a century, the Monterey Workshops established themselves as an important international forum to foster, among academia, industry, and government agencies, discussion and exchange of ideas, research results and experience in developing software intensive systems, and have significantly advanced the field. The community of the workshop participants grew to become an influential source of ideas and innovations and its impact on the knowledge economy has been felt worldwide. The workshop was chaired by L. Zhang of Beihang University, China and F. Kordon of University P. & M. Curie, France.

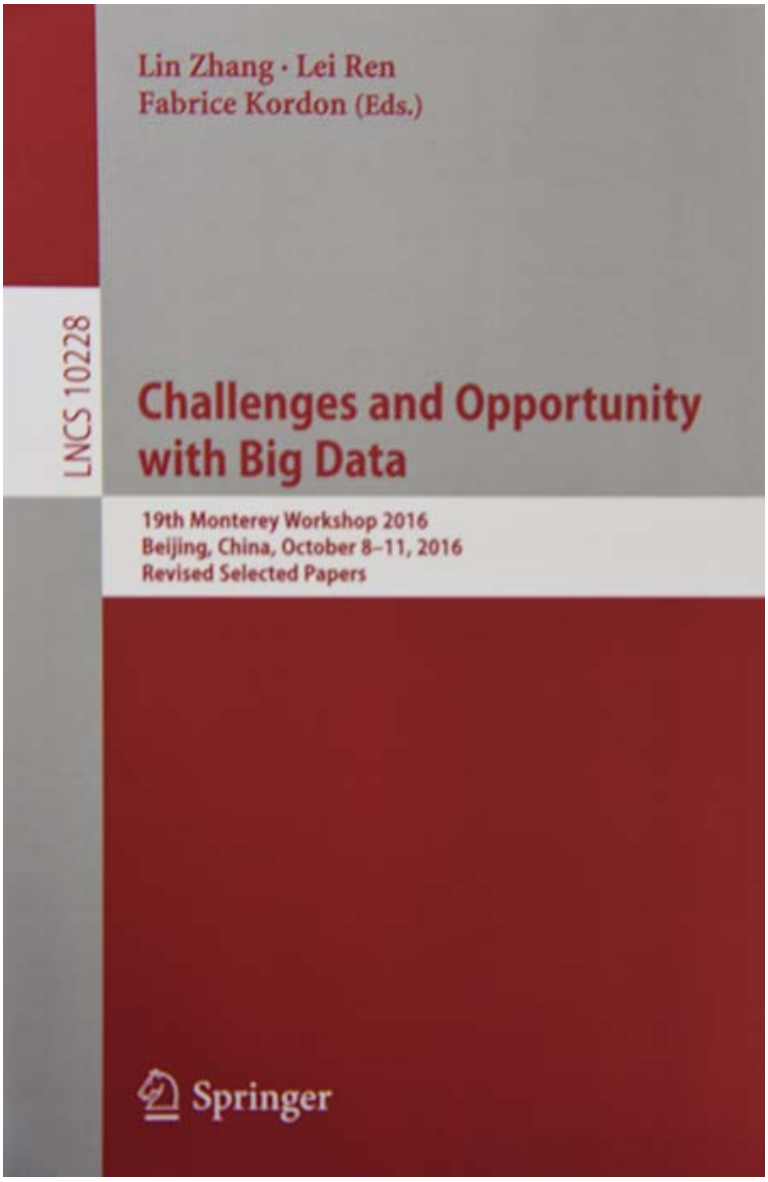


Figure 65 - Lecture Notes on Computer Science, vol. 10228

2. Topics Addressed

The main theme for 2016 Monterey Workshop was Challenges and Opportunities with Big Data. Big Data and Big Data Analytics comprise a multi-dimensional scientific and technological pursuit that is transforming sciences, engineering, healthcare, medicine, manufacturing, business, finance, retail, service industries, education, media, transportation, and government. There are new challenges and opportunities in developing software intensive systems that support Big Data and Big Data Analytics. We solicited high-quality original research papers on any aspect of Big Data enabled software intensive systems. This included but was not limited to the following:

- Theoretical Underpinnings for Big Data
    - Theoretical models for Big Data
    - Computational models for Big Data
    - Machine learning algorithms for Big Data Analytics
    - Time series analysis algorithms
    - Inconsistency theory for data/information/knowledge quality
    - Data uncertainty and inconsistency handling
    - Challenges due to volume, variety, velocity, veracity, distribution
    - Challenges in storage, robustness, curation, search, retrieval, visualization
    - Challenges in privacy, security, provenance
- Software Intensive Systems for Big Data Analytics
    - Cloud/grid/stream computing for Big Data
    - High performance/parallel computing platforms for Big Data
    - Autonomic computing
    - Energy-efficient computing for Big Data
    - Programming models and environments for Cluster, Cloud, and Grid Computing to Support Big Data
    - Software techniques and architectures for Big Data Computing
    - Open Platforms for Big Data computing
    - Models for Big Data beyond Hadoop/MapReduce, STORM
    - Software systems to support life cycle of Big Data Analytics
- Big Data Management
    - Search and mining of data from a variety of sources and problem domains
    - Managing granularity of knowledge content in Big Data
    - Algorithms and systems for Big Data search
    - Distributed, and peer-to-peer Search
    - Software architecture for Big Data search, scalability and efficiency
    - Data acquisition, integration, pre-processing
    - Link and graph mining
    - Semantic-based data mining
    - Mobile, location-centric and unstructured Big Data
    - Multimedia and multi-structured data
    - Organizing and reorganizing Big Data

- Big Data Enabled Knowledge Engineering -Innovative, concurrent, and scalable big data intelligence analysis
    - Big Data enabled knowledge extraction, discovery, analysis, and mining
    - Big Data enabled knowledge integration, merging, migration, and transformation
    - Big Data enabled decision making
    - Big Data enabled perpetual learning
    - Big Data enabled knowledge systems development and tools
    - Context-driven user querying
- Big Data Simulation
    - Big data based modeling
    - Big Data in simulation systems
    - Big Data and simulation model management
    - Big Data enabled intelligent simulation
    - Big Data based embedded simulation
    - Simulation for visual analytics
- Big Data Visualization
    - Visual analytics of high-dimensional and spatio-temporal data
    - Scientific and information visualization
    - Machine learning and data mining for Big Data visual analytics
    - Visual representation and interaction
    - Perception and cognition of visual information
    - Analytical reasoning and sense-making on Big Data
    - Semantics-driven visual analytics
    - Visual analytics algorithms and foundations
    - High performance graphics for visualization
    - Advanced technologies and applications for visual analytics
    - Visual analytics processes, workflows and evaluation methods
- Big Data applications
    - Complex Big Data applications in sciences, engineering, medicine, healthcare, finance, business, education, transportation, retail and services, law, telecommunication
    - Big Data Analytics in small business enterprises
    - Big Data Analytics in government, public sector and society in general

- Large-scale recommendation systems and social media systems
- Intrusion detection based on Big Data Analytics
- Anomaly Detection in very large scale systems
- Threat detection using Big Data Analytics
- Real-life case studies of value creation through Big Data Analytics
- Big Data Analytics as a service
- Big Data industry standards
- Experiences with Big Data project deployments

- **Industrial Track**: This track solicited papers that described successful Big Data applications. The emphasis of the industrial track was on contributions that reported challenges and opportunities for Big Data applications.

3. Committees

Program chairs were D. Zhang of Macau University of Science and Technology, Macau and L. Ren of Beihang University, China. The steering committee was as follows:

- E. Astesiano - University of Genova, Italy
- M. Broy - Technical University Munich, Germany
- M. Griss - Carnegie Mellon University, USA
- F. Kordon - University P. & M. Curie, France
- Luqi - Naval Postgraduate School, USA (chair)
- W. Roscoe - Oxford University, UK
- J. Sztipanovits - Vanderbilt University, USA

U. 20TH MONTEREY WORKSHOP ON CYBER

1. Introduction

From November 27-29, 2018, the Naval Postgraduate School hosted a workshop with attendees from academia, industry, and government to study the topic "Cyber" in Monterey. This workshop was the 20th in the Monterey Workshop series. Each of the previous workshops were focused on a specific, then-relevant topic. The Monterey Workshop on Cyber recognized that there was an urgent national need to provide assured and dependable software-intensive systems that can operate reliably in today's contested network and computing environments. These systems and the personnel that develop, maintain, and use them are "a strategic asset that protects the American people, the

homeland, and the American way of life", as stated in the Presidential Executive Order 13870 reproduced in Appendix A [25].

The workshop explored current capabilities and gaps related to this national need. Potential participants were asked to present unclassified DoD concerns to industry and academic minds. The best papers were selected for presentation in order to familiarize researchers and practitioners with stakeholders' issues, and to help them understand the feasibility and limitations of current and near-future technologies. The workshop poster is shown in Figure 66, and an article summarizing the workshop is reproduced in Appendix C [26].

Figure 66 – 20th Monterey Workshop Poster

2. Topics Addressed

Workshop topics included, but were not limited to:

- unmanned autonomous vehicles
- managing the "internet of things"
- future wireless communications using light (i.e., LIFI)
- auto-tagging XML data based on user behavior and heuristics
- Cybersecurity for military operations and baseline software/hardware
- unmanned/ autonomous systems
- networked industrial control systems
- relation between big data analysis and decision-making

A sample of focus-questions for the workshop are the following:

- How could machine learning, big data analysis, secure software/hardware architectures, AI, and other science methods enable the Cyber workforce to counter insider threats, integrate physical security, and conduct offensive Cyber operations?
- Is industry ready for the challenges and does it have solutions and tools to overcome the obstacles to secure Cyber operations?

3. Committees

General Chairs:	Steve Lerman, Provost
	Dan Boger, Cyber, IS, & C4I Chair
Program Committee Chair:	Jeff Paduan, Dean of Research
Committee Members:	Douglas Fouts, Electrical & Computer Engineering
	George Dinolt, Computer Science
	Robert Dell, Operations Research
	Jessica Piombo, National Security Affairs
Local Arrangement Chair:	Sharon Runde, Associate Cyber Chair
Committee Members:	LCDR Dave Couchman, US Navy
	Andre Xie & Karen Darken, Interns
Monterey Workshop Series Chair:	Luqi, Computer Science & Cyber Group

4. General Chairs' Welcome Remarks

The General Chairs, Dr. Steve Lerman, Provost of the Naval Postgraduate School and Prof. Dan Boger, Chair of the Cyber Academic Group, welcome you to Monterey, California for the 20th Monterey Workshop on Cyber.

There is an urgent national need to provide assured and dependable software-intensive systems that can operate dependably in today's contested network and computing environments. This workshop provides a forum for discussion and communication to align academic efforts with this need, and it solicits innovative solutions to Cyber problems that offer an achievable path to Cybersecurity assurance at all levels of the system. As in previous Monterey Workshops, the leading computer scientists, software and system engineers in government, academia, and industry are invited as well as Cyber agencies from around the nation. There were both classified and unclassified sessions at the workshop.

For a quarter-century, the Monterey Workshops have served as an influential forum for exchange of ideas and experience in software-intensive systems. Given the current state of industry, there are obvious challenges to fully secure functioning Cyber systems. This workshop provides a fantastic opportunity for all of us to work together on innovations for Cyber problem solving.

5. Workshop Program

Day 1, Tuesday 11/27/2018

08:00-08:30 Registration & Check In

08:30-08:45 Workshop Chairs' Remarks

08:45-10:15 Unmanned/Autonomous Vehicles in Cyber

- Autonomous Aerial Swam Robotics Cyber Challenges, Duane Davis, NPS
- Mcity Autonomous Vehicle Testbed, Brian Noble, University of Michigan
- Risks of Integrating UAS into the National Airspace System, Brian Smith, NASA Ames

10:15-10:30 Break

10:30-12:00 Unmanned/Autonomous Vehicles in Cyber - continued

- Engineering Lightweight Autonomy, Nicholas Weaver, UC Berkeley and the International Computer Science Institute
- UAV with Munition, Valdis Berzins, NPS
- Tracking Provenance of Autonomous Decisions, Doug Lange, Space and Naval Warfare Systems Center

12:00-13:00 Lunch

13:00-14:00 Challenges of Naval Operations to Cybersecurity

- LVC Simulation Support for the Marine Corps' CEMOES, Chris Fitzpatrick, NPS
- Zero-Days, One Obligation, Anthony Akil, NPS

14:00-15:00 Cybersecurity Architectures & Hardware

- Continuous Monitoring, Fragility and Trust, Ron Durbin, AFIT
- Cybersecurity Architectures and Hardware Issues, Brian Smith, NASA Ames

15:00-15:15 Break

15:15-16:30 Panel on Monterey Cyber Institute, Jeff Paduan (Chair), Jim Newman, Dan Boger, & John Drummond

Day 2, Wednesday 11/28/18

08:45-10:00 Cyber Fundamentals and Machine Learning

- Fundamentals of Cybersecurity, Dave Dampier, University of Texas, San Antonio
- NetBrane: Detecting and Protecting from DDoS Attacks, Stephen Hayne, Colorado State University

10:00-10:15 Break

10:15-12:00 Cyber Fundamentals and Machine Learning

- Timing Side Channels Due to Human-Computer Interaction, Vinnie Monaco, NPS
- AI and Safety/Security Threats, Brian Smith, NASA Ames
- Securing & Breaking Cyber Using SMT, Nikolaj Bjørner, Microsoft

12:00-13:00 Lunch

13:00-14:30 Insider Threats

- Personal Intelligence (PERSINT), William Roof, WHR
- Self-equity - A Trustworthiness Construct, Ryan Kelly, US Army
- Information Systems and Insider Threats, Sharon Runde, NPS

14:30-15:00 Break

15:00-16:30 Panel on Cyber Education and Cyber Physical Systems, Paul Tortora (Chair), Tracy Emmersen, Duane Davis, & Sharon Runde

Day 3, Thursday 11/29/18

08:30-10:00 Classified Cyber Issues in Military & Civilian Conflict

10:00-10:15 Break

10:15-12:00 Roundtable Discussion, Workshop Summary & 21st Workshop Plan

III. CONCLUSION

The Monterey Workshop series has yielded many fruitful results and conclusions, and professional and personal links between the world's leading researchers and academics have been established and strengthened. While much has been accomplished by the workshops outlined in this report, this effort will have to be continued to ensure that research and academia are aligned with the needs of industry and government stakeholders.

A. WEBSITE

In February 2019 work began on consolidating the material from the various proceedings and websites created over the course of the Monterey Workshop series into one new website, which would be hosted by NPS. We began by collecting and organizing as much digital material as possible, including pictures, posters, and the various proceedings. Due to the workshop series' long history, these materials were in a wide variety of formats and locations, and it took significant effort to make them easily transferrable to the planned website.

Once the material was collected, we worked closely with the NPS webmaster to create a new website for the workshop series using Liferay. This website has a landing page describing the series as a whole, and individual pages describing each workshop, including topics covered, committee memberships, and a link to the full proceedings. Extensive email correspondence was required due to the vast amount of material used.

The new website is publicly available and contains all basic information about the Monterey Workshop series. The earlier workshops, due to their antiquity, have somewhat shorter pages as the content of their proceedings is not easily transferrable to the website. Additionally, posters were not created for the earlier workshops, and the same is true for photographs. This new website can be found at [22].

B. FUTURE WORK

The 21st Monterey Workshop will be held in 2021, date and location is still to be determined as of the publication of this report. The tentative focus of this workshop will be on intelligent systems, machine learning, and trustworthy artificial intelligence. As the

use of drones and other autonomous systems proliferate in civilian and military applications, methods for certifying AI and embedding of artificial intelligence will be crucial to the success of future Navy systems and operations.

The 20th Monterey Workshop on Cyber was very productive, and resulted in a 200-page proceedings report [21], as well as follow-on discussions of potentially productive next steps. The incorporation of Machine Learning and AI has been recognized as a national security priority through the issuing of an Executive Order [27], an Artificial Intelligence Strategy [28], and the creation of the Joint Artificial Intelligence Center (JAIC) under the purview of the Department of Defense's Chief Information Officer (CIO). Implementation and incorporation of Machine Learning (ML) will be necessary to exploit and enhance the opportunities revealed by the results of the 20th Monterey Workshop.

However, implementation of Machine Learning systems is not as simple as it might first appear, as the effectiveness of ML is very domain-specific and requires clearly defined problems. It is often the case that users spend more time manipulating and preparing data than actually doing ML. One problem already identified is that Machine Learning needs large quantities of data that are well aligned to the problem domain, and the different parts of this data have to be orthogonal. Knowing exactly how to represent the data, which visualizations to use, when to transform/extract features, and when/how to aggregate, normalize, and remove outliers is halfway to solving the entire problem. Feature engineering is of paramount importance, as the biggest gains often come from choosing the right features to extract. This reinforces the domain-specific nature of effective Machine Learning. Adapting the words of Dr. Richard Hamming, when confronted with vast amounts of data, "learning to learn" is as vital for ML systems as it is for human analysts.

It should also be noted that as downsizing pressures and the focus on cost control increases, the ability of the acquisitions workforce (AWF) to understand the costs and risks associated with acquiring effective and efficient materiel solutions that support naval missions is of paramount importance. One example of how ML can contribute to streamlining acquisition is its application to managing and applying lessons learned (LL) from previous projects. Analysis and sharing of LL data is critical to a wide variety of

organizations, and the Navy AWF is no exception. Manual approaches to finding and utilizing relevant LL data are slow and labor-intensive. Additionally, in the next decade much of the current AWF is expected to retire. This will negatively impact the institutional knowledge of acquisitions that has been built up by the Navy, but modernization of the AWF data environment through the use of ML should revolutionize continuous learning and allow the AWF to compensate for this and other issues, such as constrained budgets, increasing complexity, limited competition, a shrinking industrial base, and cyber security challenges.

With these complications and objectives in mind, current efforts are directed towards organizing a Monterey Workshop on Machine Learning and Lessons Learned, embedding Artificial Intelligence, and certification of intelligent systems. Details of these themes are subject to change, however, as anyone may propose a workshop topic to the steering committee. The details of the topic focus are usually refined as a result of discussions among the steering committee and experts in the field. This process is currently ongoing, but it has become clear that the inclusion of non-US subject matter experts will be important. These experts will bring a wealth of perspective and experiences to the workshop, and they may also be supported by EU or local funds for their participation in the workshop. NPS and Navy contacts with sponsor organizations will also need to be leveraged to ensure that top researchers in the field are made aware of the opportunities provided by this workshop.

The 21st Monterey Workshop will enable assessment of recent progress, identify profitable directions for future research, and identify aspects of ML and AI where advancements would bring the greatest value to the Navy. The adoption of AI by the Navy has the potential to strengthen national security and transform the speed and agility of Navy operations. It may also help to counteract limiting factors that the Navy is predicted to face in the future, such as constrained budgets, increasing complexity, limited competition, a shrinking industrial base, and various cyber security challenges. Good progress has been made, but much remains to be done to realize the vision laid out in this section of the report.

REFERENCES

[1] "AFOSR/ARO/ONR Research Review on Formal Methods in Software Engineering: Concurrent and Real-Time Systems," in *Monterey Workshop*, Monterey, CA, 1992.

[2] "Proceedings of AFOSR/ARO/ONR Workshop on Increasing the Practical Impact of Formal Methods for Computer-Aided Software Development," in *Monterey Workshop*, Monterey, CA, 1993.

[3] "Proceedings of Monterey Workshop 94," in *Monterey Workshop*, Monterey, CA, 1994.

[4] " Proceedings of 1995 Monterey Workshop on Increasing the Practical Impact of Formal Methods in Computer Aided Software Development: Software Architecture," in *Monterey Workshop*, Monterey, CA, 1995.

[5] "Special Issue on Computer-Aided Prototyping," *Journal of Systems Integration: Systems Engineering and Integration,* vol. 6, no. 1/2, 1996.

[6] "Requirements Targeting Software and Systems Engineering," in *Springer LNCS 1526*, Bernried, Germany, 1997.

[7] "Proceedings of the 1998 ARO/ONR/NSF/DARPA Monterey Workshop on Engineering Automation for Computer Based Systems," in *Monterey Workshop*, Monterey, CA, 1999.

[8] "Proceedings of the 2000 Monterey Workshop on Modeling Software Systems Structures in a Fastly Moving Scenario," in *Monterey Workshop*, Santa Margherita Ligure, Italy, 2000.

[9] "Monterey Workshop 2001: Engineering Automation for Software Intensive System Integration," in *Monterey Workshop*, Monterey, CA, 2001.

[10] "Monterey Workshop 2002 Radical Innovations of Software and Systems Engineering in the Future," in *Monterey Workshop*, Venice, Italy, 2002.

[11] "10th Monterey Workshop Software Engineering for Embedded Systems: From Requirement to Implementation," in *Monterey Workshop*, Chicago, IL, 2003.

[12] "11th Monterey Workshop Software Engineering Tools: Compatability and Integration," in *Monterey Workshop*, Vienna, Austria, 2004.

[13] "Reliable Systems on Unreliable Networked Platforms," in *Springer LNCS 4322*, Laguna Beach, CA, 2005.

[14] "Composition of Embedded Systems: Scientific and Industrial Issues," in *Springer LNCS 4888*, Paris, France, 2006.

[15] "Innovations for Requirements Analysis. From Stakeholders' Needs to Formal Design," in *Springer LNCS 5320*, Monterey, CA, 2007.

[16] "Foundations of Computer Software. Future Trends and Techniques for Development," in *Springer LNCS 6028*, Budapest, Hungary, 2008.

[17] "Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems," in *Springer LNCS 6662*, Redmond, WA, 2010.

[18] "Large-Scale Complex IT Systems. Development, Operation and Management," in *Springer LNCS 7539*, Oxford, United Kingdom, 2012.

[19] "18th Monterey Workshop Integrity of Industrial Control Systems and Command & Control for the New Norm," in *Monterey Workshop*, Monterey, CA, 2016.

[20] "Challenges and Opportunity with Big Data," in *Springer LNCS 10228*, Beijing, China, 2016.

[21] Luqi et. al., "Proceedings of the 20th Monterey Workshop," in *Monterey Workshop*, Monterey, CA, 2019.

[22] Luqi and R. Stuart, "Monterey Workshop Series," NPS, 1 December 2019. [Online]. Available: https://my.nps.edu/web/monterey-workshop/welcome. [Accessed 31 December 2019].

[23] The Monterey Workshops, "The Monterey Workshop Series," 2019. [Online]. Available: http://www.montereyworkshop.org/.

[24] Luqi, ""Meeting in the Rain" - Monterey Workshop 2002 in Venice," *The Software Practitioner,* vol. 13, no. 3, pp. 9-10, May - June 2003.

[25] D. J. Trump, "Executive Order on America's Cybersecurity Workforce," 2 May 2019. [Online]. Available: https://www.whitehouse.gov/presidential-actions/executive-order-americas-cybersecurity-workforce/. [Accessed 10 May 2019].

[26] Luqi, "NPS Hosts Cyber Workshop in Monterey," NPS, 17 December 2018. [Online]. Available: https://www.nps.edu/web/guest/-/nps-hosts-cyber-workshop-in-monterey. [Accessed 27 December 2019].

[27] D. J. Trump, "Maintaining American Leadership in Artificial Intelligence," 14 February 2019. [Online]. Available: https://www.federalregister.gov/documents/2019/02/14/2019-02544/maintaining-american-leadership-in-artificial-intelligence. [Accessed 12 December 2019].

[28] U.S. Department of Defense, "Summary of the 2018 Department of Defense Artificial Intelligence Strategy," 12 February 2019. [Online]. Available: https://media.defense.gov/2019/Feb/12/2002088963/-1/-1/1/SUMMARY-OF-DOD-AI-STRATEGY.PDF. [Accessed 12 December 2019].

APPENDIX A: EXECUTIVE ORDER 5/2/2019

AMERICA'S CYBERSECURITY WORKFORCE

By the authority vested in me as President by the Constitution and the laws of the United States of America, and to better ensure continued American economic prosperity and national security, it is hereby ordered as follows:

Section 1. Policy. (a) America's cybersecurity workforce is a strategic asset that protects the American people, the homeland, and the American way of life. The National Cyber Strategy, the President's 2018 Management Agenda, and Executive Order 13800 of May 11, 2017 (Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure), each emphasize that a superior cybersecurity workforce will promote American prosperity and preserve peace. America's cybersecurity workforce is a diverse group of practitioners who govern, design, defend, analyze, administer, operate, and maintain the data, systems, and networks on which our economy and way of life depend. Whether they are employed in the public or private sectors, they are guardians of our national and economic security.

b) The United States Government must enhance the workforce mobility of America's cybersecurity practitioners to improve America's national cybersecurity. During their careers, America's cybersecurity practitioners will serve in various roles for multiple and diverse entities. United States Government policy must facilitate the seamless movement of cybersecurity practitioners between the public and private sectors, maximizing the contributions made by their diverse skills, experiences, and talents to our Nation.

(c) The United States Government must support the development of cybersecurity skills and encourage ever-greater excellence so that America can maintain its competitive edge in cybersecurity. The United States Government must also recognize and reward the country's highest-performing cybersecurity practitioners and teams.

(d) The United States Government must create the organizational and technological tools required to maximize the cybersecurity talents and capabilities of American workers –-especially when those talents and capabilities can advance our national and economic security. The Nation is experiencing a shortage of cybersecurity

talent and capability, and innovative approaches are required to improve access to training that maximizes individuals' cybersecurity knowledge, skills, and abilities. Training opportunities, such as work-based learning, apprenticeships, and blended learning approaches, must be enhanced for both new workforce entrants and those who are advanced in their careers.

(e) In accordance with Executive Order 13800, the President will continue to hold heads of executive departments and agencies (agencies) accountable for managing cybersecurity risk to their enterprises, which includes ensuring the effectiveness of their cybersecurity workforces.

Sec. 2. Strengthening the Federal Cybersecurity Workforce. (a) To grow the cybersecurity capability of the United States Government, increase integration of the Federal cybersecurity workforce, and strengthen the skills of Federal information technology and cybersecurity practitioners, the Secretary of Homeland Security, in consultation with the Director of the Office of Management and Budget (OMB) and the Director of the Office of Personnel Management (OPM), shall establish a cybersecurity rotational assignment program, which will serve as a mechanism for knowledge transfer and a development program for cybersecurity practitioners. Within 90 days of the date of this order, the Secretary of Homeland Security, in consultation with the Directors of OMB and OPM, shall provide a report to the President that describes the proposed program, identifies its resource implications, and recommends actions required for its implementation. The report shall evaluate how to achieve the following objectives, to the extent permitted by applicable law, as part of the program:

(i) The non-reimbursable detail of information technology and cybersecurity employees, who are nominated by their employing agencies, to serve at the Department of Homeland Security (DHS);

(ii) The non-reimbursable detail of experienced cybersecurity DHS employees to other agencies to assist in improving those agencies' cybersecurity risk management;

(iii) The use of the National Initiative for Cybersecurity Education Cybersecurity Workforce Framework (NICE Framework) as the basis for cybersecurity skill requirements for program participants;

(iv) The provision of training curricula and expansion of learning experiences to develop participants' skill levels; and

(v) Peer mentoring to enhance workforce integration.

(b) Consistent with applicable law and to the maximum extent practicable, the Administrator of General Services, in consultation with the Director of OMB and the Secretary of Commerce, shall:

(i) Incorporate the NICE Framework lexicon and taxonomy into workforce knowledge and skill requirements used in contracts for information technology and cybersecurity services;

(ii) Ensure that contracts for information technology and cybersecurity services include reporting requirements that will enable agencies to evaluate whether personnel have the necessary knowledge and skills to perform the tasks specified in the contract, consistent with the NICE Framework; and

(iii) Provide a report to the President, within 1 year of the date of this order, that describes how the NICE Framework has been incorporated into contracts for information technology and cybersecurity services, evaluates the effectiveness of this approach in improving services provided to the United States Government, and makes recommendations to increase the effective use of the NICE Framework by United States Government contractors.

(c) Within 180 days of the date of this order, the Director of OPM, in consultation with the Secretary of Commerce, the Secretary of Homeland Security, and the heads of other agencies as appropriate, shall identify a list of cybersecurity aptitude assessments for agencies to use in identifying current employees with the potential to acquire cybersecurity skills for placement in reskilling programs to perform cybersecurity work. Agencies shall incorporate one or more of these assessments into their personnel development programs, as appropriate and consistent with applicable law.

(d) Agencies shall ensure that existing awards and decorations for the uniformed services and civilian personnel recognize performance and achievements in the areas of cybersecurity and cyber-operations, including by ensuring the availability of awards and decorations equivalent to citations issued pursuant to Executive Order 10694 of January 10, 1957 (Authorizing the Secretaries of the Army, Navy, and Air Force To Issue

Citations in the Name of the President of the United States to Military and Naval Units for Outstanding Performance in Action), as amended. Where necessary and appropriate, agencies shall establish new awards and decorations to recognize performance and achievements in the areas of cybersecurity and cyber-operations. The Assistant to the President for National Security Affairs may recommend to agencies that any Cyber unified coordination group or similar ad hoc interagency group that has addressed a significant cybersecurity or cyber-operations-related national security crisis, incident, or effort be recognized for appropriate awards and decorations.

(e) The Secretary of Homeland Security, in consultation with the Secretary of Defense, the Director of the Office of Science and Technology Policy, the Director of OMB, and the heads of other appropriate agencies, shall develop a plan for an annual cybersecurity competition (President's Cup Cybersecurity Competition) for Federal civilian and military employees. The goal of the competition shall be to identify, challenge, and reward the United States Government's best cybersecurity practitioners and teams across offensive and defensive cybersecurity disciplines. The plan shall be submitted to the President within 90 days of the date of this order. The first competition shall be held no later than December 31, 2019, and annually thereafter. The plan for the competition shall address the following:

(i) The challenges and benefits of inviting advisers, participants, or observers from non-Federal entities to observe or take part in the competition and recommendations for including them in future competitions, as appropriate;

(ii) How the Department of Energy, through the National Laboratories, in consultation with the Administrator of the United States Digital Service, can provide expert technical advice and assistance to support the competition, as appropriate;

(iii) The parameters for the competition, including the development of multiple individual and team events that test cybersecurity skills related to the NICE Framework and other relevant skills, as appropriate. These parameters should include competition categories involving individual and team events, software reverse engineering and exploitation, network operations, forensics, big

data analysis, Cyber analysis, Cyber defense, Cyber exploitation, secure programming, obfuscated coding, cyber-physical systems, and other disciplines;

(iv) How to encourage agencies to select their best cybersecurity practitioners as individual and team participants. Such practitioners should include Federal employees and uniformed services personnel from Federal civilian agencies, as well as Department of Defense active duty military personnel, civilians, and those serving in a drilling reserve capacity in the Armed Forces Reserves or National Guard;

(v) The extent to which agencies, as well as uniformed services, may develop a President's Cup awards program that is consistent with applicable law and regulations governing awards and that allows for the provision of cash awards of not less than $25,000. Any such program shall require the agency to establish an awards program before allowing its employees to participate in the President's Cup Cybersecurity Competition. In addition, any such program may not preclude agencies from recognizing winning and non-winning participants through other means, including honorary awards, informal recognition awards, rating-based cash awards, time-off awards, Quality Step Increases, or other agency-based compensation flexibilities as appropriate and consistent with applicable law; and

(vi) How the uniformed services, as appropriate and consistent with applicable law, may designate service members who win these competitions as having skills at a time when there is a critical shortage of such skills within the uniformed services. The plan should also address how the uniformed services may provide winning service members with a combination of bonuses, advancements, and meritorious recognition to be determined by the Secretaries of the agencies concerned.

(f) The Director of OMB shall, in consultation with appropriate agencies, develop annually a list of agencies and subdivisions related to cybersecurity that have a primary function of intelligence, counterintelligence, investigative, or national security work, including descriptions of such functions. The Director of OMB shall provide this list to the President, through the Deputy Assistant to the President for Homeland Security and Counterterrorism (DAPHSCT), every year starting September 1, 2019, for consideration

of whether those agencies or subdivisions should be exempted from coverage under the Federal Labor-Management Relations Program, consistent with the requirements of section 7103(b)(1) of title 5, United States Code.

Sec. 3. Strengthening the Nation's Cybersecurity Workforce. (a) The Secretary of Commerce and the Secretary of Homeland Security (Secretaries), in coordination with the Secretary of Education and the heads of other agencies as the Secretaries determine is appropriate, shall execute, consistent with applicable law and to the greatest extent practicable, the recommendations from the report to the President on Supporting the Growth and Sustainment of the Nation's Cybersecurity Workforce (Workforce Report) developed pursuant to Executive Order 13800. The Secretaries shall develop a consultative process that includes Federal, State, territorial, local, and tribal governments, academia, private-sector stakeholders, and other relevant partners to assess and make recommendations to address national cybersecurity workforce needs and to ensure greater mobility in the American cybersecurity workforce. To fulfill the Workforce Report's vision of preparing, growing, and sustaining a national cybersecurity workforce that safeguards and promotes America's national security and economic prosperity, priority consideration will be given to the following imperatives:

(i) To launch a national Call to Action to draw attention to and mobilize public- and private-sector resources to address cybersecurity workforce needs;

(ii) To transform, elevate, and sustain the cybersecurity learning environment to grow a dynamic and diverse cybersecurity workforce;

(iii) To align education and training with employers' cybersecurity workforce needs, improve coordination, and prepare individuals for lifelong careers; and

(iv) To establish and use measures that demonstrate the effectiveness and impact of cybersecurity workforce investments.

(b) To strengthen the ability of the Nation to identify and mitigate cybersecurity vulnerabilities in critical infrastructure and defense systems, particularly cyber-physical systems for which safety and reliability depend on secure control systems, the Secretary of Defense, the Secretary of Transportation, the Secretary of Energy, and the Secretary of Homeland Security, in coordination with the Director of OPM and the Secretary of

Labor, shall provide a report to the President, through the DAPHSCT, within 180 days of the date of this order that:

(i) Identifies and evaluates skills gaps in Federal and non-Federal cybersecurity personnel and training gaps for specific critical infrastructure sectors, defense critical infrastructure, and the Department of Defense's platform information technologies; and

(ii) Recommends curricula for closing the identified skills gaps for Federal personnel and steps the United States Government can take to close such gaps for non-Federal personnel by, for example, supporting the development of similar curricula by education or training providers.

(c) Within 1 year of the date of this order, the Secretary of Education, in consultation with the DAPHSCT and the National Science Foundation, shall develop and implement, consistent with applicable law, an annual Presidential Cybersecurity Education Award to be presented to one elementary and one secondary school educator per year who best instill skills, knowledge, and passion with respect to cybersecurity and cybersecurity-related subjects. In developing and implementing this award, the Secretary of Education shall emphasize demonstrated superior educator accomplishment — without respect to research, scholarship, or technology development — as well as academic achievement by the educator's students.

(d) The Secretary of Commerce, the Secretary of Labor, the Secretary of Education, the Secretary of Homeland Security, and the heads of other appropriate agencies shall encourage the voluntary integration of the NICE Framework into existing education, training, and workforce development efforts undertaken by State, territorial, local, tribal, academic, non-profit, and private-sector entities, consistent with applicable law. The Secretary of Commerce shall provide annual updates to the President regarding effective uses of the NICE Framework by non-Federal entities and make recommendations for improving the application of the NICE Framework in cybersecurity education, training, and workforce development.

Sec. 4. General Provisions. (a) Nothing in this order shall be construed to impair or otherwise affect:

(i) the authority granted by law to an executive department or agency, or the head thereof; or

(ii) the functions of the Director of OMB relating to budgetary, administrative, or legislative proposals.

(b) This order shall be implemented consistent with applicable law and subject to the availability of appropriations.

(c) This order is not intended to, and does not, create any right or benefit, substantive or procedural, enforceable at law or in equity by any party against the United States, its departments, agencies, or entities, its officers, employees, or agents, or any other person.

DONALD J. TRUMP

THE WHITE HOUSE,

May 2, 2019.

APPENDIX B: "MEETING IN THE RAIN" – MONTEREY WORKSHOP 2002 IN VENICE

"Meeting in the Rain" – Monterey Workshop 2002 in Venice

Luqi, Jennifer, Valdis & Lynn

National Research Council
Naval Postgraduate School

The 9th Monterey Workshop had the pleasure of convening in one of the world's most beautiful cities: Venice, Italy. Though one's experience of the historically rich city depends on the weather, the rain did cease long enough to allow workshop attendees to take in the magnificent architecture of Venice. Venice proved to be a perfect setting for the workshop, encouraging the innovations of the great thinkers that have populated the city to inspire the minds of the present.

How could the Monterey Workshop that originated in Monterey, California, USA be held in Venice, Italy? How did it happen to attract top software engineering and formal methods researchers from the US, Europe, and Asia to gather in a raining Venice? To answer these questions, we will have to trace back a decade of history.

During the last decade, object-orientation was the driving factor for new system solutions in many areas ranging from e-commerce to embedded systems. New modeling languages such as UML, new programming languages such as Java, and CASE tools have considerably affected the system development techniques of today and will remain key influences in the near future. However, actual practice shows the deficiencies of these approaches. For example, there is insufficient evidence that software productivity has increased with the new methods; UML does not have a complete mathematical definition of its meaning, which inhibits the construction of powerful analysis and development tools. Support for mobile distributed system development is missing, and for many applications, object-oriented design may not produce clean well-structured code, as much experience shows. Consequently, there is an urgent need to clarify the role of object-orientated and new "post object-oriented" software engineering and programming techniques. That is the aim of the Monterey Workshop, so named because it was initiated in Monterey, California, USA.

The objective of the entire series of Monterey Workshops is to "Increase the Practical Impact of Formal Methods in Computer-Aided Software Development". The approach is to improve software practice via application of engineering theory and to encourage development of engineering theory that is well suited for this purpose. The series was initiated in response to a situation in which much computing research was conducted in a "mathematical vacuum tube", with little or no recognition of or impact on the difficulties encountered in actual software development projects. Since the initial workshop, many special topics have been and will be discussed

- 0th 1992 Concurrent and Real-Time Systems
- 1st 1993 Software Slicing, Merging and Integration
- 2nd 1994 Software Evolution
- 3rd 1995 Specification-Based Software Architecture
- 4th 1996 CAPSTAG - Computer Aided Prototyping
- 5th 1997 Requirements Targeting Software and Systems Engineering
- 6th 1998 Engineering Automation for Computer Based Systems
- 7th 2000 Modeling Software System Structures in a Fastly Moving Scenario
- 8th 2001 Engineering Automation for Software Intensive System Integration
- 9th 2002 Radical Innovations of Software and Systems Engineering in the Future
- 10th 2003 Embedded Systems

The Monterey Workshop has been able to bring the brightest minds in Software Engineering together with the purpose of increasing the practical impact of formal methods for software development so that these potential benefits can be realized in actual practice. In the workshop, attendees and organizers work to clarify what good formal methods are, what are their feasible capabilities, and what are their limits. Overall, the workshop strives to reduce the gap between theory and practice. This has been a slow and difficult process because theoreticians and practitioners do not normally talk to each other, and did not at the beginning of the series of workshops. This gap has gradually been reduced. In particular, researchers have focused on problems that are relevant to practitioners, and have helped to demonstrate how recent theory can be applied to solve current problems in software development practice.

The series of Monterey Workshops was continuously guided and sponsored by the U.S. Army Research Office, National Science Foundation, ONR, AFOSR, DARPA, many

Figure 67 - *The Software Practitioner*, Vol. 13, No. 3, May-June 2003, p. 9

universities and organizations. Dr. David Hislop at ARO served years as the vice president of a large industrial firm in charge of software development. His personal experience and vision on how to improve the state of the software R&D has greatly influenced the software engineering community in the world. He is a firm believer of the objectives of Monterey Workshops and therefore persistently supports workshops for helping attendees to make progress one step at a time to reach the goal. He has greatly inspired hundreds of software engineering researchers in the world to participate in Monterey Workshops.

With her 30 years of experience in foundational software research and practical software tool building, Prof. Luqi (Naval Postgraduate School, USA), the initiator of and chair of most of the Monterey Workshops, has followed her intellectual beliefs to organize the workshops for merging two sides of the community to work together toward a practical impact on software development. She emphasized that software engineering is tangible and is directly connected to the system and its real-world context, and is not simply a batch or mechanical process. This context is complicated, includes groups of people, and is constantly changing. These issues are inescapable sources of difficulties.

The Monterey workshop has indeed won its prestige worldwide. Because of its cutting edge topics and influences on both academy and industrial communities, it has attracted more and more researchers and engineers from all over the world, especially from Europe. The organizers decided that the workshop should alternate between continents, so the 9th Monterey Workshop convened in beautiful – damp – Venice.

The specific theme of the 9th workshop in the series (Venice 2002) was "Radical Innovations of Software and Systems Engineering in the Future". This theme was addressed by papers, position statements and discussions among 40 invited researchers. The focus of the event is a natural outgrowth of the previous workshops in the series. Prof. Martin Wirsing (University of Munich, Germany) served as an outstanding workshop chair. He considered the software development by using *views*. He pointed out that each stakeholder of a software system might express his view of the system from his own viewpoint and may employ the notation most appropriate for this viewpoint. In particular, most viewpoints taken by system stakeholders concentrate on parts of the whole system under construction which may either be fortunately orthogonal and separated by clean interfaces, or may overlap in intricate ways. His philosophy and research results have influenced the interesting program, which included a broad range of topics from abstract data types on top of XML to new design notations for software engineering.

The basic premise of this workshop was that using appropriate formal methods supported by appropriate software tools could greatly benefit practical software development. Computer aid should give software designers better control over their products, with resulting improvements in software usefulness and reliability and reductions in time and cost for large-scale changes. For computer support to provide enough practical benefit to justify the necessary investment tools and training, a better theoretical understanding of software modification and effective formal methods for solving various subproblems associated with software evolution is needed.

Many researchers expressed their thoughts about O-O. Dr. Bertrand Meyer (Eiffel Software) said, "Rather than declaring object technology passé and chasing unspecified breakthroughs, it would be more effective to apply O-O principles seriously."

Prof. Manfred Broy (Technical University of Munich, Germany) pointed out, "For the development of distributed software systems that interact over a wide area network, the object oriented paradigm is not sufficient. It does not support concurrency and it does not support the right patterns of asynchronous interaction." The author thought that "current object-oriented languages do not provide enough support for separating the components from the connections in reconfigurable software architectures."

"All software engineering bullets, even those that contain some silver, are mostly made of lead."

It appears that there is no software engineering silver bullet. Prof. Daniel Berry (University of Waterloo, Canada), who is a regular attendee of Monterey Workshop, pointed out in his presentation that "all software engineering bullets, even those that contain some silver, are made mostly of lead. The situation with software engineering methods is not unlike that stubborn chest of drawers in the old slapstick movies: a shlimazel pushes in one drawer and out pops another one, usually right smack dab on the poor shlimazel's knees and shins. If you find a new method that eliminates an old method's pain, the new method will be found to have its own source of pain... I no longer get excited over any new language, development model, method, tool, or environment that is supposed to improve programming... The most important work is that addressing requirements, changes, and the psychology and sociology of programming." Prof. Berry has worked on both formal program verification and obtaining accurate requirements from users and customers of systems.

Prof. Egidio Astesiano (University of Genova, Italy) also thought that requirement capture and specification had been recognized as a paramount activity in every significant software development process model.

Throughout the workshop, many different views on how this pain could be reduced were presented. Many of these approaches involved computer aid for or automation of software engineering subtasks based on particular theories and various kinds of formal models. A common theme was to hide theoretical results and complex mathematical ideas inside tools with simple interfaces so that practitioners could use them without needing to fully understand the theoretical results behind them. However, there was general agreement that the pain could not be eliminated entirely: no matter what you do, somewhere in the process someone has to think clearly and in detail to determine what problems should be solved by the software that is to be developed.

Prof. Jeannette M. Wing (Carnegie Mellon University, USA) presented a paper titled "What, Who and How of Tomorrow". "Systems of the future will be more complex than they are today. Users will expect greater capability, security, customizability, etc. ..." She said, "Systems of the future will be developed by teams of people. These teams will be cross-cultural, cross-disciplinary, across time lines, and involve different skill sets. Not all team members will be savvy with computers. There will be artists, astrophysicists, linguists, psychologists, policy makers, etc." When talking about how we can build better systems in a better way, she agreed that there was no silver bullet, but cautioned not to let that be an excuse for taking an unprincipled approach to software and system engineering.

The Monterey Workshop has been a great success in its pursuit of sharing and expanding the current outlook and function of Software Engineering. The workshop continues to bring out the best of the Software Engineering field, and helps to guide the directions for future initiatives on software engineering research. The next Monterey Workshop will be led by a strong technical team of four – Prof. Bruce Krogh(CMU), Prof. Insup Lee (UPenn), Prof. Matt Dwyer (Kansas State University), and Prof. Sol Shatz (Uni. of Chicago). It is going to be held in Chicago in September 2003. It is anticipated to follow the spirit and tradition of all Monterey Workshops.

Figure 68 - *The Software Practitioner*, Vol. 13, No. 3, May-June 2003, p. 10

APPENDIX C: ARTICLE ON THE 20TH MONTEREY WORKSHOP

The source of this appendix is a web article published by NPS summarizing the events and findings of the 20th Monterey Workshop on Cyber [26].

NPS HOSTS CYBER WORKSHOP IN MONTEREY

By Dr. Luqi, Professor, NPS Department of Computer Science

The 20th Monterey Workshop on Cyber was held on Nov. 27-29, 2018. NPS Provost Steve Lerman, Cyber Academic Group Chair Dan Boger, and Dean of Research Jeff Paduan chaired the workshop, and led the workshop program committee with members from several departments, including computer science, electrical and computer engineering, information sciences, operations research and national security affairs.

The first of two workshop panels discussed the establishment of a Monterey Cyber Institute in depth. The second panel on Cyber Education exchanged concepts and scopes including large cyber programs with more than 4,500 students in the nation, with panel members from the Naval Academy and University of Texas at San Antonio.

The workshop solicited innovative solutions to cyber problems with an achievable path to cyber security assurance at all levels of systems. All at the workshop worked together on innovations for cyber problem solving in both classified and unclassified sessions. Participants explored how machine learning, big data analysis, secure software/hardware architectures, artificial intelligence (AI), and other science methods enable the cyber workforce to integrate security, counter insider threats, and conduct cyber operations.

The Monterey Workshop series has been an influential forum for aligning scientific research and innovations among academia, industry, and government sponsoring agencies since 1991. The overarching theme is increasing the practical impact of scientific methods in computer science and engineering of software intensive systems. There is an urgent national need to provide software intensive systems that operate dependably in today's contested computing environments. Given the state of industry, there are obvious challenges to fully secure functioning cyber systems. Leading computer scientists, software and system engineers in government, academia, cyber agencies and computer industry participated.

Steering committee member, Stanford Computer Science pioneer Prof. Zohar Manna (1939-2018) was honored at the workshop. He was a long-standing workshop participant along with a joint effort on increasing the practical impact of theoretical computer science that today helps form the basis for artificial intelligence and reliable software, and underpins advances in cyber.

NPS collaborated with his theorem prover and temporal logic for specification-based, real-time rapid prototyping to remove requirement errors and then link to program synthesis to prevent implementation errors. More in depth study is planned for the 21st Monterey Workshop on Artificial Intelligence.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, VA 22060

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California 93943

3. Research Sponsored Programs Office, Code 41
   Naval Postgraduate School
   Monterey, CA 93943

4. Dean of Research
   Naval Postgraduate School
   Monterey, CA 93943

5. Prof. Luqi, Computer Science
   Naval Postgraduate School
   Monterey, CA 93943