FOSE
The Future of Software Engineering Symposium
22–23 November 2010

Eiffel at 25
FOSE Special Event
24 November 2010

ETH Zurich, Switzerland

Program
Contents

Welcome to FOSE 3
Welcome to Eiffel at 25 3
Venue 4
Meals 4
Internet Access 4
Eiffel at 25 Reception 4
FOSE Dinner 5
FOSE: Program 6
Eiffel at 25: Program 8
FOSE: Abstracts and Speaker Biographies 9
Eiffel at 25: Abstracts 20
About ETH Zurich 24
About Zurich 25
Addresses 26
Welcome to FOSE

The Future of Software Engineering (FOSE) Symposium focuses on defining the achievements of software engineering in the past decades and showcasing visions for the future.

The symposium features invited talks and panel discussions involving some of the most prominent researchers and technologists who have shaped the field. The list of speakers includes:

- Barry Boehm
- Manfred Broy
- Patrick Cousot
- Erich Gamma
- Yuri Gurevich
- Michael A. Jackson
- Rustan Leino
- Bertrand Meyer
- David Lorge Parnas
- Dieter Rombach
- Joseph Sifakis
- Niklaus Wirth
- Pamela Zave
- Andreas Zeller

The invited talks cover a broad range of topics, determined by the views of the individual speakers on what constitutes the most important issues facing software development. They include both research- and technology-oriented talks.

Welcome to Eiffel at 25

The original version of Eiffel was designed in 1985. Since then the Eiffel method, language and supporting tools and libraries have continuously evolved; the current version of the language (ISO standard, 2006) includes a number of radically new concepts such as void safety, agents and conversion, and progress is continuing towards verification techniques and concurrency.

The basic concepts have remained the same, including the central role of Design by Contract, the use of a powerful object-oriented model without compromises, the emphasis on language consistency and simplicity ("one good way to do anything"), the enforcement of strong software engineering principles, and the application of a single notation and set of concepts throughout the lifecycle, from analysis and design to implementation and maintenance ("seamless development").

The special one-day "Eiffel at 25" event is devoted to an examination of the contributions of Eiffel to the progress of software engineering, important applications of the Eiffel approach, its influence on other languages and tools, and future developments of Eiffel.
Venue

FOSE and Eiffel at 25 take place at ETH Zurich. All rooms are located in the main building (“Hauptgebäude”, HG), Rämistrasse 101, Zurich. To get to the venue rooms, enter through the main entrance on Rämistrasse and follow the signs to room F 30 (Audi Max), where the registration desk is located. FOSE will take place in room F 30, Eiffel at 25 will take place in room F 33.1.

The registration desk will be open every day between 22 November and 24 November, from 8:15 to 9:30, in front of room F30. If you need to register outside of these times, or need any other assistance, please talk to one of the helpers.

Meals

Lunch will be served every day between 13:15 and 14:30 in the Dozentenfoyer (faculty club) on the top of the ETH main building (HG). To get there, use the elevators and go to the top floor (J). Vegetarian meals will be available. You will receive vouchers for the lunches as part of the registration package.

Every day there will be two coffee breaks, one in the morning (10:30-11:00) and one in the afternoon (16:00-16:30). The coffee breaks will be held in the area outside room F 30.

Internet Access

Wireless access should be available everywhere in the building. Network configuration uses DHCP. Access to the web pages of the ETH Zurich is possible without any validation. To access resources outside the ETH, connect to the wireless network “public”, then validate at

http://enter.ethz.ch/

Please use the following user name and password combination:

User name:
Password:

This connection does not block ports or filter traffic. You should have full access to the Internet, including Web, Telnet, IMAP, IMAPS and SSH. Be advised that the wireless traffic is not encrypted.

Eiffel at 25 Reception

A stand-up meal will be offered to participants of Eiffel at 25 on Wednesday, 24 November, starting at 18:00 in front of room F 30.
FOSE Dinner

The FOSE dinner (for those who have registered for it) will be held on **Monday, 22 November** (apéro from **19:30**, dinner at **20:00**), at the restaurant **Zunfthaus zur Waag**, Münsterhof 8, Zurich. The closest tram stop is **Paradeplatz** (trams No. 2, 6, 7, 8, 9, 11, 13).

To get to the restaurant from the ETH main building, you can take **tram No. 9** towards **Triemli**, which leaves directly in front of the main entrance (station **ETH/Universitätsspital**). Exit the tram at station **Paradeplatz** (the ride takes about 10 minutes) and walk 200 meters to the restaurant. You can buy the tram ticket for **Kurzstrecke** (2.50 CHF) directly at the tram stop.

Alternatively, you can **walk** to the restaurant in approximately 20 minutes (1.2 kilometers) through Zurich's old town. We recommend to leave the main building via the rear entrance and to follow the route shown on the map.
FOSE: Program

Monday, 22 November 2010
Room F 30 (Audi Max)

9:00–10:30 Session 1
Session chair: Joseph Sifakis

- Welcome
  Bertrand Meyer
- Engineering and Software Engineering
  Michael A. Jackson
- Seamless Method- and Model-based Software and Systems Engineering
  Manfred Broy

10:30–11:00 Coffee break

11:00–13:15 Session 2
Session chair: Andreas Zeller

- Precise Documentation: The Key to Better Software
  David Lorge Parnas
- Design Patterns – Past, Present & Future
  Erich Gamma
- Internet Evolution and the Role of Software Engineering
  Pamela Zave

13:15–14:30 Lunch (Dozentenfoyer)

14:30–16:00 Session 3
Session chair: Yuri Gurevich

- Some Future Software Engineering Opportunities and Challenges
  Barry Boehm
- Empirically Driven Software Engineering Research
  Dieter Rombach

16:00–16:30 Coffee break

16:30–18:30 Session 4 & Panel discussion I
Session chair: Rustan Leino

- Computer Science: A Historical Perspective and a Current Assessment
  Niklaus Wirth
- Panel discussion I
  Panelists: Barry Boehm, Manfred Broy, Erich Gamma, Michael A. Jackson, David Lorge Parnas, Niklaus Wirth, Pamela Zave

19:30–23:00 Apéro and Dinner at Zunfthaus zur Waag, Zurich
Tuesday, 23 November 2010
Room F 30 (Audi Max)

9:00–10:30 Session 5
Session chair: Erich Gamma

- Logical Abstract Domains and Interpretations
  Patrick Cousot (with Radhia Cousot and Laurent Mauborgne)
- Tools and Behavioral Abstraction: A Direction for Software Engineering
  Rustan Leino

10:30–11:00 Coffee break

11:00–13:15 Session 6
Session chair: Pamela Zave

- Component-based Construction of Heterogeneous Real-time Systems in BIP
  Joseph Sifakis
- Evidential Authorization
  Yuri Gurevich (with Andreas Blass, Michal Moskal, and Itay Neeman)
- Mining Specifications: A Roadmap
  Andreas Zeller

13:15–14:30 Lunch (Dozentenfoyer)

14:30–16:00 Session 7 & Panel discussion II
Session chair: Michael A. Jackson

- Concurrent Programming Is Easy
  Bertrand Meyer
- Panel discussion II
  Panelists: Patrick Cousot, Yuri Gurevich, Rustan Leino, Bertrand Meyer, Joseph Sifakis, Andreas Zeller

16:00–16:30 Coffee & End of FOSE Symposium
Eiffel at 25: Program

Wednesday, 24 November 2010
Room F 33.1

9:00–10:30 Session 1
Session chair: Bertrand Meyer

- 25 Years of Eiffel IDEs: an Insider's Perspective
  Emmanuel Stapf, Eiffel Software, USA
- CloudStudio: Collaborative, Cloud-based Software Development
  Martin Nordio, ETH Zurich, Switzerland
- A Modular Scheme for Deadlock Prevention in Eiffel's SCOOP
  Scott West, ETH Zurich, Switzerland

10:30–11:00 Coffee break

11:00–13:00 Session 2
Session chair: Carlo A. Furia

- Mathematical Modelling Languages in Eiffel
  Jonathan Ostroff, York University, Canada
- EiffelBase2: Strong Contracts for Design and Verification
  Nadia Polikarpova, ETH Zurich, Switzerland
- Contracts and Testing, the Glamour of Eiffel Programming
  Manuel Oriol, University of York, United Kingdom
- Automated Fixing of Programs with Contracts
  Yi Wei, ETH Zurich, Switzerland

13:00–14:30 Lunch (Dozentenfoyer)

14:30–16:00 Session 3
Session chair: Martin Nordio

- Correctness by Construction: Developing High-integrity Systems
  Piotr Nienaltowski, Altran Praxis, United Kingdom
- A Verification Assistant for Eiffel Programs
  Julian Tschannen, ETH Zurich, Switzerland
- Contracts for Verification – a Personal Perspective
  Carlo A. Furia, ETH Zurich, Switzerland

16:00–16:30 Coffee break

16:30–18:00 Session 4
Session chair: Sebastian Nanz

- Applying Eiffel to Large-scale Developments in Business Information Systems
  Paul-Georges Crismer, Groupe S, Belgium
- Diverse Uses of Eiffel in Industry, and Thoughts for the Future
  Thomas Beale, Ocean Informatics, United Kingdom
- Perspectives for the Evolution of Eiffel
  Bertrand Meyer, ETH Zurich, Switzerland

18:00–19:30 Reception & End of Eiffel at 25
Michael A. Jackson
The Open University, UK

Michael Jackson has worked in software since 1961, focusing first on program and system design method. His program design method, described in Principles of Program Design (1975), was chosen as the standard method for UK government software development. Later work with Pamela Zave at AT&T on specifications and architecture for telecommunication and other systems, is the subject of several patents. More recent work on problem structure, analysis and solution, is described in Software Requirements & Specifications (1995), Problem Frames (2001), and many published papers. He holds visiting posts at The Open University and the University of Newcastle; he participates in research projects there and at other research and academic institutions. He has received several research awards, including the IEE Achievement Medal, The British Computer Society Lovelace Medal, and the ACM Sigsoft Outstanding Research Award.

Engineering and Software Engineering

The phrase 'software engineering' has many meanings. One central meaning is the reliable development of dependable computer-based systems, especially those for critical applications. This is not a solved problem. Failures in software development have played a large part in many fatalities and in huge economic losses. While some of these failures may be attributable to programming errors in the narrowest sense—a program's failure to satisfy a given formal specification—there is good reason to think that most of them have other roots. These roots are located in the problem of software engineering rather than in the problem of program correctness. The famous 1968 conference was motivated by the belief that software development should be based on "the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering." Yet after forty years of currency the phrase 'software engineering' still denotes no more than a vague and largely unfulfilled aspiration. Two major causes of this disappointment are immediately clear. First, too many areas of software development are inadequately specialised, and consequently have not developed the repertoires of normal designs that are the indispensable basis of reliable engineering success. Second, the relationship between structural design and formal analytical techniques for software has rarely been one of fruitful synergy: too often it has defined a boundary between competing dogmas, at which mutual distrust and incomprehension deprive both sides of advantages that should be within their grasp. This paper discusses these causes and their effects. Whether the common practice of software development will eventually satisfy the broad aspiration of 1968 is hard to predict; but an understanding of past failure is surely a prerequisite of future success.

Manfred Broy
Technische Universität München, Germany

Manfred Hans Bertold Broy studied Mathematics and Computer Science at the Technical University of Munich. He graduated in 1976, 1980 he received his Ph.D. and 1982 he completed his Habilitation Thesis at the Faculty of Mathematics at the Technical University of Munich. 1983 till 1989 he worked as a full professor
for computer science and founding dean at the Faculty of Mathematics and Computer Science at the University of Passau. In October he became a full professor for computer science at the Faculty of Computer Science the Technische Universität München (former chair of Professor F.L. Bauer).

His research interests are software and systems engineering comprising both theoretical and applied aspects including system models, specification and refinement of system components, specification techniques, development methods and verification. Professor Broy is a member of the European Academy of Sciences and a Member of the Deutsche Akademie der Naturforscher "Leopoldina". In 1994 he received the Leibniz Award by the Deutsche Forschungsgemeinschaft and in 2007 the Konrad Zuse Medal by the Gesellschaft für Informatik.

**Seamless Method- and Model-based Software and Systems Engineering**

Today engineering software intensive systems is still more or less handicraft or at most at the level of manufacturing. Many steps are done ad-hoc and not in a fully systematic way. Applied methods, if any, are not scientifically justified, not justified by empirical data and as a result carrying out large software projects still is an adventure. However, there is no reason why the development of software intensive systems cannot be done in the future with the same precision and scientific rigor as in established engineering disciplines. To do that, however, a number of scientific and engineering challenges have to be mastered. The first one aims at a deep understanding of the essentials of carrying out such projects, which includes appropriate models and effective management methods. What is needed is a portfolio of models and methods coming together with a comprehensive support by tools as well as deep insights into the obstacles of developing software intensive systems and a portfolio of established and proven techniques and methods with clear profiles and rules that indicate when which method is ready for application. In the following we argue that there is scientific evidence and enough research results so far to be confident that solid engineering of software intensive systems can be achieved in the future. However, yet quite a number of scientific research problems have to be solved.

**David Lorge Parnas**

Middle Road Software, Inc.

Dr. David Lorge Parnas has been studying industrial software development since 1969. Many of his papers have been found to have lasting value. For example, a paper written 25 years ago, based on a study of avionics software, was recently awarded a SIGSOFT IMPACT award.

Parnas has won more than 20 awards for his contributions. In 2007, Parnas was proud to share the IEEE Computer Society's one-time sixtieth anniversary award with computer pioneer Professor Maurice Wilkes of Cambridge University.

Parnas received his B.S., M.S. and Ph.D. in Electrical Engineering from Carnegie Mellon University and honorary doctorates from the ETH in Zurich (Switzerland), the Catholic University of Louvain (Belgium), and the University of Italian Switzerland (Lugano). He is licensed as a Professional Engineer in Ontario.

Parnas is a Fellow of the Royal Society of Canada (RSC), the Association for Computing Machinery (ACM), the Canadian Academy of Engineering (CAE), the Gesellschaft für Informatik (GI) in Germany and the IEEE. He is a Member of the Royal Irish Academy.
Parnas is the author of more than 270 papers and reports. Many have been repeatedly republished and are considered classics. A collection of his papers can be found in:


Dr. Parnas is Professor Emeritus at McMaster University in Hamilton Canada, and at the University of Limerick Ireland and an Honorary Professor at Ji Lin University in China. He is President of Middle Road Software.

**Precise Documentation: The Key to Better Software**

The prime cause of the sorry "state of the art" in software development is our failure to produce good design documentation. Poor documentation is the cause of many errors and reduces efficiency in every phase of a software product's development and use. Most software developers believe that "documentation" refers to a collection of wordy, unstructured, introductory descriptions, thousands of pages that nobody wanted to write and nobody trusts. In contrast, Engineers in more traditional disciplines think of precise blueprints, circuit diagrams, and mathematical specifications of component properties. Software developers do not know how to produce precise documents for software. Software developments also think that documentation is something written after the software has been developed. In other fields of Engineering much of the documentation is written before and during the development. It represents forethought not afterthought. Among the benefits of better documentation would be: easier reuse of old designs, better communication about requirements, more useful design reviews, easier integration of separately written modules, more effective code inspection, more effective testing, and more efficient corrections and improvements. This paper explains how to produce and use precise software documentation and illustrate the methods with several examples.

**Erich Gamma**

IBM Rational Zurich Research Lab, Switzerland

Erich Gamma is a Distinguished Engineer at IBM Rational Software's Zurich lab. He is one of the leaders of the Jazz project. He was the original lead of the Eclipse's Java development environment (JDT) and is on the Project Management Committee for the Eclipse project. Erich is also a member of the Gang of Four, which is known for its classic book, Design Patterns - Elements of Reusable Object-Oriented Software. Erich has collaborated with Kent Beck on developing JUnit, the de facto standard testing tool for Java.

**Design Patterns – Past, Present & Future**

Design Patterns are now a 15 year old thought experiment. And today, for many, Design Patterns have become part of the standard development lexicon. This talk looks back to the origin of Design Patterns and how they evolved since their initial description. I will then show patterns in action in the context of the Eclipse and Jazz platforms. Finally, I will discuss how the Design Patterns from the book can be refactored towards a Design Pattern 2.0 version.
Pamela Zave
AT&T Laboratories – Research, USA

Pamela Zave received an A.B. degree in English from Cornell University, Ithaca, New York, and a Ph.D. degree in computer sciences from the University of Wisconsin–Madison.

Dr. Zave has been with AT&T Research since 1981. Currently she works with a group of other researchers building tools for IP-based multimedia services using the Distributed Feature Composition architecture, invented by her and Michael A. Jackson. Her other research interests include Internet architecture, modeling and verification of network protocols, feature interaction, requirements engineering, and multiparadigm specification.

Dr. Zave is an ACM Fellow and an AT&T Fellow. She has received two Ten-Year Most Influential Paper awards, four Best Paper awards, and the AT&T Strategic Patent Award. She holds 14 patents in the telecommunication area. Dr. Zave is currently chair of IFIP Working Group 2.3 on Programming Methodology.

Internet Evolution and the Role of Software Engineering

The classic Internet architecture is a victim of its own success. Having succeeded so well at empowering users and encouraging innovation, it has been made obsolete by explosive growth in users, traffic, applications, and threats. For the past decade, the networking community has been focused on the many deficiencies of the current Internet and the possible paths toward a better future Internet. This paper explains why the Internet is likely to evolve toward multiple application-specific architectures running on multiple virtual networks, rather than having a single architecture. In this context, there is an urgent need for research that starts from the requirements of Internet applications and works downward toward network resources, in addition to the predominantly bottom-up work of the networking community. This paper aims to encourage the software-engineering community to participate in this research by providing a starting point and a broad program of research questions and projects.

Barry Boehm
University of Southern California, USA

Barry W. Boehm is TRW Professor of Software Engineering and Director of the Center for Software Engineering at the University of Southern California.

Barry Boehm received his B.A. degree from Harvard in 1957, and his M.S. and Ph.D. degrees from UCLA in 1961 and 1964, all in Mathematics. He also received an honorary Sc.D. in Computer Science from the U. of Massachusetts in 2000.


His current research interests focus on value-based software engineering, including a method for integrating a software system's process models, product models, property models, and success models called Model-Based (System) Architecting and Software
Engineering (MBASE). His contributions to the field include the Constructive Cost Model (COCOMO), the Spiral Model of the software process, the Theory W (win-win) approach to software management and requirements determination, the foundations for the areas of software risk management and software quality factor analysis, and two advanced software engineering environments: the TRW Software Productivity System and Quantum Leap Environment.

He has served on the boards of several scientific journals, including the IEEE Transactions on Software Engineering, IEEE Computer, IEEE Software, ACM Computing Reviews, Automated Software Engineering, Software Process, and Information and Software Technology. He has served as Chair of the AIAA Technical Committee on Computer Systems, Chair of the IEEE Technical Committee on Software Engineering, and as a member of the Governing Board of the IEEE Computer Society. He has also served as Chair of the Air Force Scientific Advisory Board's Information Technology Panel, Chair of the NASA Research and Technology Advisory Committee for Guidance, Control, and Information Processing, and Chair of the Board of Visitors for the CMU Software Engineering Institute.

His honors and awards include Guest Lecturer of the USSR Academy of Sciences (1970), the AIAA Information Systems Award (1979), the J.D. Warnier Prize for Excellence in Information Sciences (1984), the ISPA Freiman Award for Parametric Analysis (1988), the NSIA Grace Murray Hopper Award (1989), the Office of the Secretary of Defense Award for Excellence (1992), the ASQC Lifetime Achievement Award (1994), the ACM Distinguished Research Award in Software Engineering (1997), and the IEEE Harlan D. Mills Award (2000). He is a Fellow of the primary professional societies in computing (ACM), aerospace (AIAA), electronics (IEEE), and systems engineering (INCOSE), and a member of the National Academy of Engineering.

Some Future Software Engineering Opportunities and Challenges

This paper provides an update and extension of a 2006 paper, "Some Future Trends and Implications for Systems and Software Engineering Processes," Systems Engineering, Spring 2006. Some of its challenges and opportunities are similar, such as the need to simultaneously achieve high levels of both agility and assurance. Others have emerged as increasingly important, such as the challenges of dealing with ultralarge volumes of data, with multicore chips, and with software as a service. The paper is organized around eight relatively surprise-free trends and two "wild cards" whose trends and implications are harder to foresee. The eight surprise-free trends are:

1. Increasing emphasis on rapid development and adaptability;
2. Increasing software criticality and need for assurance;
3. Increased complexity, global systems of systems, and need for scalability and interoperability;
4. Increased needs to accommodate COTS, software services, and legacy systems;
5. Increasingly large volumes of data and ways to learn from them;
6. Increased emphasis on users and end value;
7. Computational plenty and multicore chips;
8. Increasing integration of software and systems engineering;

The two wild-card trends are:

9. Increasing software autonomy; and
Dieter Rombach  
Fraunhofer Institute for Experimental Software Engineering, Germany  
University of Kaiserslautern, Germany  

Prof. Dr. Dieter Rombach studied mathematics and computer science at the universities of Karlsruhe and Kaiserslautern. Since 1992, he has held the chair for Software Engineering at the University of Kaiserslautern, and since 1996, he has been Executive Director of the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern (FhG ISE). From 1984-1991, he was Professor of Computer Science at the University of Maryland, and from 1986-1991 he was Project Manager at NASA/Goddard Space Flight Center.

Prof. Rombach's main research topics can be found in the area of "Software Engineering", especially in methods for developing software with predictable quality, in quantitative methods for measuring and assessing software products and software processes, in languages and methods for developing and using explicit software process models, in process-based software development environments as well as in software reuse. As a consultant, he works for various U.S. American and European companies.

In 1990, Prof. Rombach received the "Presidential Young Investigator Award" from the National Science Foundation (USA) "in recognition of his successful work in the area of Experimental Software Engineering". For his exceptional services rendered to the state of Rhineland-Palatinate and his exemplary commitment to the good of society, he received the Order of Merit of the state of Rhineland-Palatinate in late 2000. Effective 1 January 2003, Prof. Rombach was appointed Fellow of the American Institute of Electrical and Electronics Engineers (IEEE) in recognition of his achievements in the area of Experimental Software Engineering.

Empirically Driven Software Engineering Research

Software engineering is a design discipline. As such, its engineering methods are based on cognitive instead of physical laws, and their effectiveness depends highly on context. Empirical methods can be used to observe the effects of software engineering methods in vivo and in vitro, to identify improvement potentials, and to validate new research results. This paper summarizes both the current body of knowledge and further challenges wrt. empirical methods in software engineering as well as empirically derived evidence regarding software typical engineering methods. Finally, future challenges wrt. education, research, and technology transfer will be outlined.

Niklaus Wirth

Niklaus Wirth has been a Professor of Computer Science at ETH (Federal Institute of Technology) in Zurich, Switzerland, from 1968 to 1999. His principal areas of contribution were programming languages and methodology, software engineering, and design of personal workstations. He has designed the programming languages Algol W (1965), Pascal (1970), Modula-2 (1979), and Oberon (1988), was involved in the methodologies of structured programming and stepwise refinement, and designed and built the workstations Lilith (1980) and Ceres (1986). He has published several text books for courses on programming, algorithms and data structures, and logical design of digital circuits. He has received various prizes and honorary doctorates, including the Turing
Award (1984), the IEEE Computer Pioneer (1988), and the Award for outstanding contributions to Computer Science Education (acm 1987).

**Computer Science: A Historical Perspective and a Current Assessment**

We begin with a brief review of the early years of Computer Science. This period was dominated by large, remote computers and the struggle to master the complex problems of programming. The remedy was found in programming languages providing suitable abstractions and programming models. Outstanding was the language Algol 60, designed by an international committee, and intended as a publication language for algorithms. The early period ends with the advent of the microcomputer in the mid 1970s, bringing computing into homes and schools. The outstanding computer was the Alto, the first personal computer with substantial computing power. It changed the world of computing.

In the second part of this presentation we consider the current state of the field and the shift of activities towards applications. Computing power and storage capacity seem to be available in unprecedented abundance. Yet there are applications that ask for even faster computers and larger stores. This calls for a new focus on multiprocessing, on systems with hundreds of processes proceeding concurrently. The invention of programmable hardware opens new possibilities for experimentation and exploring ways to design new architectures. Codesign of hardware and software becomes mandatory. The days of the general purpose von Neumann architecture seem to come to an end.

Finally, we will look at the present state of the art, the problems and the tools with which the engineer finds himself confronted today. Not only have computers become faster and storage bigger, but the tasks have become accordingly more demanding. The mass of tools has grown, programming is based on huge libraries, and the environments appear monstrously complicated and obscure. The most basic instruments, programming languages, have not improved. On the contrary, outdated languages dominate. They, and their tools, belong to the apparently irreplaceable legacy, and sometimes it seems that we have learnt little from the past. What can we do?

**Patrick Cousot**
École Normale Supérieure, France
New York University, USA

Patrick Cousot is Professor of Computer Science. He was appointed, in France at the École Normale Supérieure (ENS, 1991), Paris, France where he is Director of the Computer Science educational activities and leads the research on abstract interpretation and semantics (ENS-CNRS-INRIA joint project team "Abstraction"); the École Polytechnique (X, 1984-1997), where he developed the educational activities in Computer Science and created and headed the research laboratory (LIX) and the University of Metz (1979-1984). He was appointed, in the US, at New York University (NYU, 2008); the Massachusetts Institute of Technology (MIT, 2005) as J. C. Hunsaker Distinguished Visiting Professor. Before, Patrick Cousot was Research Scientist at the French National Center for Scientific Research (CNRS, 1974-1979). Patrick Cousot is Engineer from École des Mines of Nancy (1971), Doctor Engineer (PhD) in Computer Science (1974) and Doctor ès Sciences in Mathematics (1978) from the University of Grenoble. Patrick Cousot is the inventor with Radhia Cousot, of Abstract Interpretation. Abstract interpretation is a theory of sound approximation of mathematical structures, in particular those involved in the behavior of computer systems. It allows the systematic derivation of sound methods and algorithms for approximating undecidable or highly complex problems in various
areas of computer science (semantics, verification and proof, model-checking, static analysis, program transformation and optimization, typing, software steganography, etc.). Its current main industrial application is on the safety and security of complex hardware and software computer systems. Patrick Cousot was named Chevalier (knight) in the Ordre des Palmes académiques (1990) and in the Ordre National du Mérite (1993). He was awarded the Silver Medal of the CNRS (1999), an honorary doctorate from the Fakultät Mathematik und Informatik of the Universität des Saarlandes (2001), the Grand Prix of Computer Science and its Applications of the EADS Corporate Research Foundation attributed by the French Academy of Sciences (2006) and a Humboldt Research Award (2008). He is Member of the Academia Europaea (2006).

Logical Abstract Domains and Interpretations
We give semantic foundations to abstract domains consisting in first order logic formulae in a theory, as used in verification tools or methods using SMT-solvers or theorem provers. We exhibit conditions for a sound usage of such methods with respect to multi-interpreted semantics and extend their usage to automatic invariant generation by abstract interpretation.

Rustan Leino
Microsoft Research, Redmond, USA

Rustan Leino is a Principal Researcher in the Research in Software Engineering (RISE) group at Microsoft Research. He is known for his work on programming methods and program verification tools. At Microsoft Research, he has led the Spec# project, which includes enforced pre- and post-conditions on the .NET platform, and is the architect of the Boogie program verification framework, which underlies several program verifiers for Spec#, C, Eiffel, and other languages. Previously, Leino led the ESC/Java project at Compaq SRC, and worked on specifications on the pioneering ESC/Modula-3 project at DEC SRC. Before getting his PhD (Caltech, 1995), Leino wrote and designed object-oriented software as a technical lead in the Windows NT group at Microsoft. Leino collects thinking puzzles on a popular web page and has recently started the Verification Corner video show on channel9.msdn.com. In his spare time, he plays music and teaches cardio exercise classes.

Tools and Behavioral Abstraction: A Direction for Software Engineering
As in other engineering professions, software engineers rely on tools. Such tools can analyze program texts and design specifications more automatically and in more detail than ever before. While many tools today are applied to find new defects in old code, I predict that more software-engineering tools of the future will be available to software authors at the time of authoring. If such analysis tools can be made to be fast enough and easy enough to use, they can help software engineers better produce and evolve programs.

A programming language shapes how software engineers approach problems. Yet the abstraction level of many popular languages today is not much higher than that of C programs several decades ago. Moreover, the abstraction level is the same throughout the program text, leaving no room for behavioral abstraction where the design of a program is divided up into stages that gradually introduce more details. A stronger arsenal of analysis tools can enable languages and development environments to give good support for behavioral abstraction.
Joseph Sifakis is a CNRS researcher and the founder of Verimag laboratory, in Grenoble, France. He studied Electrical Engineering at the Technical University of Athens and Computer Science at the University of Grenoble. Joseph Sifakis is the founder of Verimag laboratory. His recognized for his pioneering work on both theoretical and practical aspects of Concurrent Systems Specification and Verification. He contributed to emergence of the area of model-checking, currently the most widely-used method for the verification of industrial applications. His current research activities include component-based design, modeling, and analysis of real-time systems with focus on correct-by-construction techniques. Joseph Sifakis has received with Ed Clarke and Allen Emerson for their contribution to Model Checking, the Turing Award for 2007.

Component-based Construction of Heterogeneous Real-time Systems in BIP

We present a framework for the component-based construction of real-time systems. The framework is based on the BIP (Behaviour, Interaction, Priority) semantic model, characterized by a layered representation of components. Compound components are obtained as the composition of atomic components specified by their behaviour and interface, by using connectors and dynamic priorities. Connectors describe structured interactions between atomic components, in terms of two basic protocols: rendezvous and broadcast. Dynamic priorities are used to select amongst possible interactions -- in particular, to express scheduling policies.

The BIP framework has been implemented in a language and a toolset. The BIP language offers primitives and constructs for modelling and composing atomic components described as state machines, extended with data and functions in C. The BIP toolset includes an editor and a compiler for generating from BIP programs, C++ code executable on a dedicated platform. It also allows simulation and verification of BIP programs by using model checking techniques.

BIP supports a model-based design methodology involving three steps:

- The construction of a system model from a set of atomic components composed by progressively adding interactions and priorities.
- The application of incremental verification techniques. These techniques use the fact that the designed system model can be obtained by successive application of property-preserving transformations in a three-dimensional space: Behavior x Interaction x Priority.
- The generation of correct-by-construction distributed implementations from a BIP model. This is achieved by source-to-source transformations which preserve global state semantics.

We present the basic theoretical results about BIP including modelling interactions by using connectors, modelling priorities, incremental verification and expressiveness. We also present two examples illustrating the methodology as well as experimental results obtained by using the BIP toolset. Further information is available at:

http://www-verimag.imag.fr/BIP,196.html
Evidential Authorization

Consider interaction of principals where each principal has its own policy and different principals may not trust each other. In one scenario the principals could be pharmaceutical companies, hospitals, biomedical labs and health related government institutions. In another scenario principals could be navy fleets of different and not necessarily friendly nations. In spite of the complexity of interaction, one may want to ensure that certain properties remain invariant. For example, in the navy scenario, each fleet should have enough information from other fleets to avoid unfortunate incidents. Furthermore, one wants to use automated provers to prove invariance. A natural approach to this and many other important problems is to provide a high-level logic-based language for the principals to communicate. We do just that. Three years ago two of us presented the first incarnation of Distributed Knowledge Authorization Language (DKAL). Here we present a new and much different incarnation of DKAL that we call Evidential DKAL. Statements communicated in Evidential DKAL are supposed to be accompanied with sufficient justifications. In general, we construe the term "authorization" in the acronym "DKAL" rather liberally; DKAL is essentially a general policy language. There is a wide spectrum of potential applications of DKAL. One ambitious goal is to provide a framework for establishing and maintaining invariants.

Mining Specifications: A Roadmap

Recent advances in software validation and verification make it possible to widely automate whether a specification is satisfied. This progress is hampered, though, by the persistent difficulty of writing specifications. Are we facing a "specification crisis"? In this paper, I show how to alleviate the burden of writing specifications by reusing and extending specifications as mined from existing software and give an overview on the state of the art in specification mining, its origins, and its potential.
Bertrand Meyer
ETH Zurich, Switzerland

Bertrand Meyer is Professor of Software Engineering at ETH Zurich (the Swiss Federal Institute of Technology) and Chief Architect of Eiffel Software, based in Santa Barbara (California). He is the author of numerous articles and over ten books on many topics of software engineering, including the bestseller "Object-Oriented Software Construction" (Prentice Hall). He is an ACM Fellow and has received the ACM Software System Award and the Dahl-Nygaard prize for object technology, and is a member of the French academy of technologies. His most recent book, "Touch of Class: An Introduction to Programming Well using Objects and Contracts" (Springer) applies advanced software engineering techniques to the introductory teaching of programming.

Concurrent Programming Is Easy

Software engineering of the future, it is widely agreed, will be concurrent: as Moore's law comes to an end, the only path to continued progress is to use highly parallel architectures. It is also widely held that this change of architecture requires a change of mindset: programmers must learn to "think parallel". The talk departs from this second assumption. Using the advent of concurrent architectures to force a radical departure from highly successful models of programming, in particular object technology, is not possible, not useful, and not desirable. The talk presents an approach to concurrency, based on the SCOOP model of concurrent object-oriented programming, that allows programmers to take advantage of concurrency while retaining the methods and disciplines that have permitted the tremendous success of programming technology over the past decades.
Eiffel at 25: Abstracts

25 Years of Eiffel IDEs: an Insider’s Perspective
Emmanuel Stapf, Eiffel Software, USA

Like the Eiffel language, Eiffel tools have come a long way since the first Eiffel-to-C compiler of 1986. EiffelStudio today is a modern multi-platform IDE providing a wide array of facilities, some of them comparable to what you find in competing tools (Eclipse, Visual Studio, ...), others still off-the-wall (push-button automatic testing, text extraction from failed cases, roundtrip engineering between text and diagrams, support for analysis as well as design and programming, metrics tool etc.).

In unifying the Eiffel offering, EiffelStudio came out of a long line, or rather tree, of developments. From the original "ec" compiler for Eiffel 1 to EiffelBench, EiffelCase, EiffelBuild and many others, some were kept, some were subsumed by later developments, and some were dropped.

This talk presents the (so far) untold story of how Eiffel and EiffelStudio came to fruition. It will walk you through the successful tools, the challenges we had to tackle in building them, the sleepless nights spent wondering whether the newest version would bootstrap, the ideas that were retained and those that had to be dumped. It will explain the tight relationship between the evolution paths of the language, the tools and the supporting hardware and software platforms. It will not hide the mistakes that were made along the way, but will explain how we recovered from them. In the end it will, we hope, provide you with a clear understanding of why and how Eiffel and EiffelStudio came to be what they are, and of their contribution to the IT world.

CloudStudio: Collaborative, Cloud-based Software Development
Martin Nordio, ETH Zurich, Switzerland

Today's software production is increasingly distributed. Gone are the days of one-company, one-site projects; most industry developments involve teams split over locations, countries, cultures. Software tools have not kept up; they provide little if any support for this new reality of software development, relying on paradigms and practices such as traditional configuration management that were developed for earlier modes of operation. The CloudStudio project at ETH Zurich is developing an IDE (integrated software development environment) enabling distributed projects to produce software "on the cloud". The environment enables every developer to work on a common project repository, shared "on the cloud". One of the main differences with traditional IDEs is that configuration management becomes unobtrusive; instead of the explicit update-modify-commit cycle, CloudStudio keeps track of successive versions and maintains the history automatically. Direct modification of a shared repository avoids raising spurious conflict notifications even when two developers are working on the same module; actual conflicts are detected early, and resolved through prevention rather painful post-hoc reconciliation of changes. Managers get an accurate and up-to-date view of the state of the development.

A Modular Scheme for Deadlock Prevention in Eiffel's SCOOP
Scott West, ETH Zurich, Switzerland

Despite the advancements of concurrency theory in the past decades, practical concurrent programming has remained a challenging activity. Fundamental problems such as data races and deadlocks still persist for programmers since available detection and prevention
tools are unsound or have otherwise not been well adopted. In an alternative approach, programming models that exclude certain classes of errors by design can address concurrency problems at a language level. In this paper we review SCOOP, an existing race-free programming model for concurrent object-oriented programming, and extend it with a scheme for deadlock prevention based on locking orders. The scheme facilitates modular reasoning about deadlocks by associating annotations with the interfaces of routines. We prove deadlock-freedom of well-formed programs using a rigorous formalization of the locking semantics of the programming model.

**Mathematical Modelling Languages in Eiffel**

Jonathan Ostroff, York University, Canada

A specification is a precise description of the client's requirements. As such, a specification is a contract between the client and the specifier as well as between the specifier and the implementor. In specification languages such as Z or B, the specification is one thing. The implementation of the specification in an object oriented language such as Java or Eiffel is something else, often leading to a certain amount of impedance mismatch between the specification (in one notation) and the final implementation (in an entirely different notation). Eiffel is designed with the single model principle in mind, i.e. object oriented software is a single product that supports multiple views. One view, suitable for compilation and execution, is the full source code. Another view is a well-defined specification of modules (or classes) amenable both to reuse and ultimately to formal verification. All views work consistently within the same semantic context. This talk describes the notion of model based contracts that extend classic Design by Contract by means of expressive mathematical models of the kind used in Z and B, but with the single model principle in mind. We show how Eiffel's design simplicity and blend of powerful mechanisms such as strong typing, agents, conversion, expanded as well as reference classes, void safety, genericity and inheritance allow for the construction of model specifications based on predicate logic and set theory. Executable model based specifications can be used to document base libraries as well as develop new software that use these libraries by first developing a mathematical model and then refining the model into an implementation. The use of the full Eiffel IDE with powerful debugging and testing capabilities, UML diagramming and documentation facilities allow the implementation to be validated using diverse systematic tests. However, we also discuss work that has been done to verify the implementations using theorem proving.

**EiffelBase2: Strong Contracts for Design and Verification**

Nadia Polikarpova, ETH Zurich, Switzerland

EiffelBase is, without any doubt, the most widely used Eiffel library. Its data structures hierarchy is an excellent example of applying object-oriented techniques and Design by Contract. However, there are known deficiencies in EiffelBase, the most important of which, in our opinion, is insufficient specification.

This talk presents EiffelBase2, a successor of EiffelBase, developed from the start with strong specifications and with the ultimate goal of proving its full functional correctness. We will discuss model-based contracts: a methodology to equip Eiffel classes with expressive and structured contracts, and to evaluate the completeness of these contracts. We will focus on how strong specifications solidify the design of the library, improve its usability and enable more extensive verification.
Contracts and Testing, the Glamour of Eiffel Programming
Manuel Oriol, University of York, United Kingdom

Testing is always the most tedious part of a software project. It is generally time consuming. Worst of all, it generally discovers bugs late in the project which can have a direct impact on deadlines and costs. Eiffel has built-in contracts and seasoned programmers use them to improve readability and help with debugging. This is one of the unique features of Eiffel and it enables automated random testing tools, such as AutoTest, to output meaningful test reports. In this presentation we review some of the techniques we tried that combine testing and contracts. We will also talk about what we believe are the next logical steps in that direction.

Automated Fixing of Programs with Contracts
Yi Wei, ETH Zurich, Switzerland

In program debugging, finding a failing run is only the first step; what about correcting the fault? Can we automate the second task as well as the first? In this presentation, I will talk about a technique that automatically generates and validates fixes for software faults. The key insights are to rely on contracts present in the software to ensure that the proposed fixes are semantically sound, and on state diagrams using an abstract notion of state based on the boolean queries of a class. I will also talk about the limitations of the technique, the difficulties that it faces and possible future research directions.

Correctness by Construction: Developing High-integrity Systems
Piotr Nienaltowski, Altran Praxis, United Kingdom

Correctness by Construction is a radical and effective method of building high-integrity software systems, producing software with extremely low defect rates and achieving good productivity. It brings together four basic principles: rigorous requirements engineering, the use of formal methods, a systematic approach to system architecture, and static analysis.

The presentation will focus on that last principle: static analysis using SPARK. SPARK is a programming language and set of verification tools designed for the development of high-integrity systems. It relies on Design by Contract, information flow analysis, verification-condition generation and theorem proving to achieve fully static verification of software. The presentation will cover the design goals of SPARK, its major features, the various forms of analysis provided by the tools, and some examples of industrial SPARK projects in aerospace, rail and government security.

The primary goal of the language – static verifiability of software – leads to challenging trade-offs in the design. I will discuss these trade-offs and cover some interesting differences between SPARK and Eiffel, including the design of the contract language and state abstraction.

A Verification Assistant for Eiffel Programs
Julian Tschannen, ETH Zurich, Switzerland

Support for verification – static and dynamic – is improving. The programmer has more and more tools available which help to improve the correctness of the program. Since many of those tools are automatic and don't need any interaction, we can relieve the programmer from invoking them explicitly.

This talk presents the scheme of an automatic verification assistant which orchestrates different verification and inference tools. A demonstration of the current state involving only verification tools will be given.
Contracts for Verification – a Personal Perspective
Carlo A. Furia, ETH Zurich, Switzerland

Whether you are skeptical of formal methods or sanguine about them, contracts can help you write better programs. In this talk, I will illustrate my view on the major strengths of contracts – as available in Eiffel – for formal verification and discuss how they can balance the divergent needs of practitioners and formalists. Based on some current research trends in practical formal verification, I will also suggest some directions to extend the contract mechanism with the goals of widening its scope and boosting its effectiveness.

Applying Eiffel to Large-scale Developments in Business Information Systems
Paul-Georges Crismer, Groupe S, Belgium

A testimony on about 14 years of Eiffel experience developing large-scale applications in a business environment. Three questions shall be addressed:

- Why Eiffel?
- How Eiffel?
- Why Eiffel in the future?

Diverse Uses of Eiffel in Industry, and Thoughts for the Future
Thomas Beale, Ocean Informatics, United Kingdom

A presentation in three parts, from an architect and CTO who has used Eiffel for 20 years. The first section describes the attractions of Eiffel as a language and formal approach. The second talks about its use in finance and health, what features make it indispensable, and how it could be made more attractive to industry. In the last part, a challenge to the modelling orthodoxy of 'classical software engineering' developed in the openEHR project will be shown. The compiler is written in Eiffel, and more importantly many of its design features have been inspired by Eiffel's own clear design. Practical realities of working in a 'Java and .Net world' will be discussed.

Perspectives for the Evolution of Eiffel
Bertrand Meyer, ETH Zurich, Switzerland

The evolution of Eiffel proceeds under the responsibility of the ECMA TC49-TG4 committee, responsible for the ISO standard. It raises interesting questions of how to preserve backward compatibility while continuing to advance language technology. I will discuss some of the ideas on the horizon and how Eiffel is likely to evolve in the next decades.
About ETH Zurich

The Eidgenössische Technische Hochschule Zürich (Swiss Federal Institute of Technology Zurich) is a science and technology university with an outstanding research record.

ETH Zurich forms a community of 20,000 people from 80 nations who study, do research or are employed there. About 360 professors in 15 departments teach mainly in the technical, mathematical and natural sciences areas.

The members of this community have the important task of maintaining and developing its prominent standing among the top universities of the world.

Twenty-one Nobel Prize winners are connected with ETH Zurich. The ETH Zurich was founded in 1855 as the "Eidgenössisches Polytechnikum" (Federal Polytechnic Institute). At that time, as the "child" of the new federal government, it was the only national university of Switzerland. Today, ETH Zurich, EPF Lausanne and four research institutions have been linked together to form the national ETH domain.
Zurich is the largest city in Switzerland and capital of the canton of Zurich. The city is Switzerland's main commercial center, and home to the country's largest airport.

'Züri' – as it is called in Swiss-German – is situated where the river Limmat leaves Lake Zurich, and is surrounded by wooded hills. The biggest of them is the Uetliberg mountain. The modern heart of the city, including the main station and most churches, is on the West side of the river, while the hillier East is home to the little alleys of the Old Town and the hip drinking spots of Niederdorfstrasse.

Zurich is not the capital (this is actually Berne) but the industrial and commercial center of Switzerland. UBS, Credit Suisse, and many private banks have their headquarters in Zurich. Zurich is the world's primary center for offshore banking, mainly thanks to Swiss bank secrecy. The financial sector accounts for about one quarter of the city's economic activities. The Swiss Stock Exchange is also headquartered in Zurich.

Having the ETH in its heart makes Zurich a center of education and research activities in Switzerland. Besides ETH, there is the University of Zurich and several private research locations such as the IBM Zurich Research Laboratory and the ABB research center in Baden.
Addresses

Bertrand Meyer
RZ Building, RZ J22
Clausiusstrasse 59
8092 Zürich
Phone: +41 (0)44 632 04 10
E-Mail: bertrand.meyer@inf.ethz.ch

FOSE Organizing Committee

Sebastian Nanz
RZ Building, RZ J4
Clausiusstrasse 59
8092 Zürich
Phone: +41 (0)44 632 65 74
E-Mail: sebastian.nanz@inf.ethz.ch

Claudia Günthart
RZ Building, RZ J7
Clausiusstrasse 59
8092 Zürich
Phone: +41 (0)44 632 83 46
Fax: +41 (0)44 632 14 35
E-Mail: claudia.guenthart@inf.ethz.ch
The Chair of Software Engineering at ETH Zurich is also organizing **TOOLS Europe 2011**, the 49th International Conference on **Object, Models, Components and Patterns**, Zurich (Switzerland), 28-30 June 2011, [http://tools.ethz.ch/](http://tools.ethz.ch/).

In addition, the Chair of Software Engineering is organizing the **LASER Summer School on Software Engineering**, Elba (Italy), 4-10 September 2011, [http://se.ethz.ch/laser/](http://se.ethz.ch/laser/).