Panel: Perspectives on Software Engineering

David Notkin (chair),[1] Marc Donner,[2] Michael D. Ernst,[3] Michael Gorlick,[4] E. James Whitehead, Jr.[5]

[1]Department of Computer Science & Engineering
University of Washington
Box 352350
Seattle WA 99195-2350
notkin@cs.washington.edu

[2]Morgan Stanley
750 Seventh Avenue
34th Floor
New York, NY 10019
donner@panix.com

[3]MIT Laboratory for Computer Science
545 Technology Square
Cambridge MA 02139
mernst@lcs.mit.edu

[4]Endeavors Technology, Inc.
19700 Fairchild, Suite 200
Irvine CA 92612
mgorlick@endtech.com

[5]Jack Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz CA 95064
ejw@cse.ucsc.edu

Abstract

This panel gives a non-standard view of the future of software engineering. Two of the speakers are recent Ph.D. graduates in computer science, with expertise in software engineering, who have taken academic positions; as people who will educate the next generation of software engineering practitioners and researchers, they provide a key vision of the future. The other two speakers are senior, having moved from the research community into a world in which they face the problems of engineering software on a daily basis. Collectively, along with interactions from the audience, these two often underrepresented perspectives provide a sense of the key directions in which software engineering— practice, research, and education — should and must go.

1. Introduction

Alan Kay is famous for his observation that "the best way to predict the future is to invent it." As attractive as this notion is, it seems that there are numerous pressures on software engineering that make it difficult to control our own destiny. As perhaps the most obvious example, the broad changes that have accompanied the introduction of the World Wide Web — perhaps an excellent instance of Kay's philosophy — have introduced a set of software engineering problems (such as Internet-time development cycles) that we must understand and address.

The intent of this panel is to introduce and discuss a range of problems and ideas that software engineering practitioners, researchers, and educators may need to appreciate and pursue in the forthcoming years. The panelists represent groups that are often underrepresented at panels like these; because of this, the hope is that they can and will broaden the set of problems and ideas on our community's table.

The panelists have been chosen to represent two distinct but complementary groups. The first group comprises two recent Ph.D.s in computer science who have recently entered academia: as members of the third generation of researchers in software engineering, they provide an important perspective because they have been educated in a world of software significantly different from the earlier generations; furthermore, they will be educating and mentoring the next generations and will thus have enormous influence on the future. The second group comprises a pair of senior people who have established research careers and have moved into the commercial world in which their ability to engineer software is of critical importance to their companies. The combination of research experience with an understanding of the pressures of the commercial world give them a perspective on software engineering research and practice that is unusual and of tremendous value.

This brief report contains the position statements of the panelists, as well as their biographical information.

2. Gorlick: Geek Chic and the Future of Space Systems

Pervasive computing embraces a vision of information that is situated, continuous, and registered. Situated information is appropriate to the current context and the state of the environment — in other words, it is information that may be sensitive to any and all of your personal preferences (who you are), your immediate associates (who you are with), your activity (what you are doing), the time of day (when it is done), the locale (where it is done), and the action (how it is done). *Continuous information* is always available irrespective of location or circumstances. *Registered information* overlays the virtual atop the physical, allowing the virtual world to inform physical objects, events, and persons, and the physical world to act as anchor points or structure for virtual representations. Wearable computing is the intimate apparel of pervasive computing — body-worn sensors, devices, computing engines, and software that interlink personal and public space.

What does this have to do with space systems? The answer — everything. Hidden away in automobile air bags, Game Boys, and cell phones are the technologies that will revolutionize the design, construction, deployment, and management of space systems. By systematically exploiting three basic principles:

- Replace physical structure with information
- Build small and think big
- Transport energy and information, not mass

and repeating the mantra of pervasive and wearable computing — tune in (situated information), turn on (continuous information), drop out (registered information) — we can create space systems of extraordinary grace, beauty, and utility that look, behave, and operate like nothing on earth.

In this context I will offer selected grand challenge problems whose solutions will profoundly impact both earthbound and spaceborne systems.

3. Donner: How to Succeed in Software?

Software development methodologies are effective but somehow unsatisfying. Our intuition about how *we* program and how really good programmers program is at odds with widely recognized best practices. This dissonance is disturbing and results in a failure to accept the validity of software engineering methods on one hand and to consider alternative models and techniques on the other. I will discuss my observations over the past fifteen or twenty years of the software development community, with some particular opinions about the ways in which financially successful software is developed.

4. Ernst: Static and Dynamic Analysis as Complementary Approaches

Dynamic detection and static verification are two essential and complementary techniques for program analysis, in particular for manipulating program invariants: dynamic detection can propose likely invariants based on program executions, but the resulting properties are not guaranteed to be true over all possible executions. Static verification can check that properties are always true, but it can be difficult for people or programs to select a goal and to annotate programs for input to a static checker. Combining these techniques overcomes the weaknesses of each: dynamically detected invariants can annotate a program or provide goals for static verification, and static verification can confirm properties proposed by a dynamic tool.

Integrating a tool (such as Daikon[1]) for dynamically detecting likely program invariants with tools (such as ESC/Java[2]) for statically verifying program properties is promising. Use of a static verifier to augment dynamic invariant detection overcomes a potential objection about possibly unsound output, classifies the output to permit programmers to use it more effectively, permits proven invariants to be used in contexts that demand sound input, and may improve the performance or precision of dynamic invariant detection. Use of dynamically detected invariants to bootstrap static verification, by providing goals or intermediate assertions or by annotating programs, lessens the burden of using static checkers (both for novices and experienced users) and indicates properties programmers might otherwise have overlooked.

The direct result of this integration will be increased, and more effective, use of both static and dynamic tools, leading to fewer bugs (by enabling programmers to introduce fewer and detect more), lowered costs (due to fewer errors and by detecting errors earlier in the software development process), better documentation, less time spent on program understanding (and more time left for performing other tasks), better test suites, more effective validation of program changes, and more efficient programs. The indirect effect will be the production of more robust, reliable, and correct computer systems. Another indirect effect is leading more working programmers to think about program invariants and

[1] http://sdg.lcs.mit.edu/~mernst/daikon/
[2] http://research.compaq.com/SRC/esc/

formal specification by introducing them to tools that operate in terms of those abstractions.

5. Whitehead: Co-opting the Advantages of the Open Source Development Model

The strength of the Open Source movement derives from its ability to leverage the Internet. The Open Source development model uses the Internet as a technology to support remote software development by loosely coordinated development teams, rapid-fire releases of software to quickly field new features and bug fixes, and integration of technically sophisticated end-users into the development process by submitting bugs, creating plug-ins, or contributing their experience and perspective to the main code release. The challenge for Software Engineering is to leverage the qualities of Open Source software in more traditional development contexts.

Rapid product delivery. Open Source projects can be extremely flexible, employing a process of continual product releases on timeframes of 2-4 weeks (or less). This is a significant departure from longer traditional release processes, and therefore raises several research challenges. Software architectures need to easily accommodate the addition of new product features without major architectural rework. Those changes that do require significant architectural change will need to be phased-in over several releases, or developed independently, and then re-integrated with the rest of the product. Automated regression testing will be increasingly important for detecting errors prior to release, and the test suite needs to accommodate rapid change. Documentation, too, will need to be rapidly synchronized. Tool support is a necessity for rapid product cycles, with important areas being software configuration management, and explicit representation of dependencies between artifacts, so the full extent of changes can rapidly be identified.

Integration of end users. Open mailing lists, bug report forms, and searchable bug report databases allow Open Source projects to provide end users with direct, immediate feedback. Additionally, open code and interfaces allows third-party developers, and technically sophisticated end users to contribute directly to the code. In all, it fosters a more immediate and direct interaction between end users and the development team, resulting in much higher user satisfaction. However, it is unclear how this model can be scaled up for applications with large numbers of users. The research challenge is how to reduce the gap between end users closer and the development team, without this causing loss of productivity and focus.

Support for geographically dispersed development teams. Open Source projects have been very successful at using the Web, email, and CVS for coordinating software development among geographically dispersed and organizationally heterogeneous teams. Sites such as SourceForge (www.sourceforge.net) now require only a few minutes of form filling before a complete project site is ready. There is a clear trend towards centralized, Internet-accessible source code repositories, and Web-based access to many classes of tools in the development environment. One research challenge is to determine what kinds of software environment tools can reasonably be Web-based. For example, text editors will likely not be Web-based, but regression test suites and code analysis engines seem like good candidates for Web hosting. Another challenge is to make changes to the Web's infrastructure so it can better support software development. One example here is the DeltaV project (www.webdav.org/deltav/), which is adding configuration management support to the HTTP protocol.

6. Panelist Biographies

David Notkin is the Boeing Professor and Associate Chair of Computer Science & Engineering at the University of Washington. Before joining the faculty in 1984, he received his Ph.D. degree at Carnegie Mellon University in 1984 and his Sc.B. degree at Brown University in 1977.

Notkin received the National Science Foundation Presidential Young Investigator Award in 1988; served as the program chair of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering; served as program co-chair of the 17th International Conference on Software Engineering; chaired the Steering committee of the International Conference on Software Engineering (1994-1996); served as charter associate editor of both *ACM Transactions on Software Engineering and Methodology* and the *Journal of Programming Languages*; served as an Associate Editor of the *IEEE Transactions on Software Engineering*; was named as an ACM Fellow in 1998; serves as the chair of ACM SIGSOFT; and received the 2000 University of Washington Distinguished Graduate Mentor Award. His research interests are in software engineering in general and in software evolution in particular. Notkin is a senior member of the IEEE.

Marc Donner is a Principal in the EBusiness Technology area at Morgan Stanley Dean Witter. He spearheads a number of strategic efforts aimed at helping the Institutional Securities businesses move aggressively into the Internet era. He joined Morgan Stanley in 1992 and has led a number of efforts, all centered around Internet, and UNIX technologies. He formed the EOffice group that built the Morgan Stanley Intranet in 1993-1994, at a time when the term wasn't known. The success of this work helped Donner mobilize support in the business areas for the establishment of the Morgan Stanley Internet web site, the first Web presence from a bulge bracket Wall Street firm. In addition to his Internet and Intranet ventures, Donner has managed hemispheric infrastructure operations for a major bank and led the planning effort for a project to reengineer the back office systems for MSDW's Institutional Securities businesses.

Before moving to Wall Street in 1992, Donner was a basic researcher, earning a Ph.D. in Computer Science from Carnegie-Mellon University with a dissertation on robot walking. After that he developed a juggling robot, used to explore issues of high-performance real time systems programming, at IBM's T. J. Watson Research Center in Yorktown Heights. In addition to his research activities at IBM, Marc introduced UNIX and TCP/IP to IBM, smuggling Sun Microsystems workstations into the research lab in the mid-1980s and ultimately establishing a network of UNIX workstations that numbered more than 100. This led to the establishment of the Agora project at IBM, intended to develop large-scale distributed system management technology and provide high-end workstation services to the Research community. Earlier in his career he worked at the Jet Propulsion Lab in Pasadena, California where he worked on projects for the Digital Telecommunications Research Section, helping gather data from planetary radar experiments. Donner holds a BS in Electrical Engineering from Caltech and a PhD in Computer Science from Carnegie-Mellon University. He is reported to be a reformed practical joker.

Michael D. Ernst is an assistant professor in the Department of Electrical Engineering and Computer Science and in the Lab for Computer Science at MIT. He received the Ph.D. in Computer Science & Engineering from the University of Washington, prior to which he was a lecturer at Rice University and a researcher at Microsoft Research. He holds the S.B. and S.M. degrees from MIT. Ernst's primary technical interest is programmer productivity, encompassing software engineering, program analysis, compilation, and programming language design. However, he has also published in artificial intelligence, theory, and other areas of computer science.

Michael Gorlick is Chief Software Architect for Endeavors Technology, Inc., where he is responsible for the design, construction, and deployment of a secure, Internet-scale, commercial, peer-to-peer infrastructure based upon embedded devices. Prior to joining Endeavors in June 2000 he was a Research Scientist in the Computer Science Laboratory of The Aerospace Corporation, where he still consults regularly. Gorlick is also a co-investigator for SensOS, a DARPA-sponsored investigation into novel techniques for the construction of embedded, adaptive, real-time operating systems undertaken in collaboration with University of California, Irvine, the University of Southern California, and the University of Colorado at Boulder.

During his tenure at Aerospace Gorlick conducted research in software engineering practices for very large-scale systems, with an emphasis on software integration techniques and highly adaptive architectures. He was the principal designer of the DARPA pico-satellites — the world record-holder for the smallest active satellites (3" x 4" x 1" at 255 grams) ever successfully launched and deployed (February 2000). As a member of the development team he was responsible for the pico-satellite conceptual design, mission planning, system integration, and flight software.

In addition, Gorlick was, from 1995 on, an active member of an informal affiliation of Aerospace researchers devoted to the exploitation of MEMS and VLSI digital electronics as foundation technology for extremely small spacecraft. He is known within the space community for his work on operating system and flight software for MEMS-based spacecraft.

Prior to joining Aerospace Gorlick spent five years at TRW Defense & Space Systems where — as Senior Unix Systems Programmer — he was responsible for providing Unix time-sharing services to a campus of 18,000 engineers and support staff. Gorlick received his M.Sc. degree in Computer Science from the University of British Columbia. He has extensive experience in a variety of industrial applications, including commercial operating systems, satellite ground stations, telemetry processing, large-scale networks, mobile computing, electronic commerce, Internet applications, Internet appliances, embedded operating systems, and digital consumer electronics.

Jim Whitehead is an Assistant Professor of Computer Science at the University of California Santa Cruz. Additionally, Whitehead is the founder and Chair of the WebDAV working group of the Internet Engineering Task Force, and led the development of the WebDAV standard for remote collaborative authoring on the Web. This standard is supported in shipping products from such companies as Microsoft, Adobe, Oracle, Apple, Novell, and many others.

In 2000, Whitehead received his Ph.D. in Information and Computer Science from the University of California, Irvine, with a dissertation titled, "An Analysis of the Hypertext Versioning Domain" under the supervision of Richard N. Taylor. Other degrees include a MS in Information in Computer Science from UC Irvine in 1994, and a BS Electrical Engineering from the Rensselaer Polytechnic Institute in 1989. His research interests include configuration management, hypertext versioning, Web protocols, remote collaborative authoring, open hypertext, and the open source development model.