Panel

Software Tools Research: A Matter of Scale and Scope – or Commoditization?

Steven Fraser
Director, Cisco Research Center
Cisco Systems, San Jose
sdfraser@acm.org

Kendra Cooper
Associate Professor
UT Dallas
kendra.m.cooper@gmail.com

Jim Coplien
Software Architecture and Agile Consultant
Gertrude & Cope
jcoplien@gmail.com

Ruth Lennon
Lecturer, Letterkenny
Institute of Technology
ruth.lennon@lyit.ie

Ramya Ravichandar
Software Engineer
Cisco Systems, San Jose
raravich@cisco.com

Diomidis Spinellis
Professor, Athens University of
Economics and Business
dds@aueb.gr

Giancarlo Succi
Dean and Professor
Free University Bolzano-Bozen
Giancarlo.Succi@unibz.it

Abstract

Tools emerge as the result of necessity – a job needs to be done, automated, and scaled. In the "early days" – compilers, code management, bug tracking, and the like – resulted in mostly local home-grown tools – and when broadly successful - spawn (from either industry or university origins) independent tools companies – for example Klocwork from Nortel and Coverity from Stanford University. This panel will bring together academics and industry professionals to discuss challenges in tools research.

Categories and Subject Descriptors
K.0 Computing Milieux

General Terms Management, Design, Economics, Experimentation, Standardization.

Keywords Tools, Process, Research

1. Steven Fraser

STEVEN FRASER joined the Cisco Research Center as Director in July 2007 with responsibilities for fostering university research collaborations, managing PhD recruiting, and nurturing technology transfer. Prior to joining Cisco Research, Steven was a Senior Staff member of Qualcomm's Learning Center in San Diego, leading software learning programs and creating the corporation's internal technical conference (the QTech Forum). Steven held a variety of technology strategy roles at BNR and Nortel including: Process Architect, Senior Manager (Disruptive Technology and Global External Research), and Advisor (Design Process Engineering). In 1994 he spent a year as a Visiting Scientist at the Software Engineering Institute (SEI) collaborating with the "Application of Software Models" project on the development of team-based domain analysis (software reuse) techniques. Fraser is the Panels Chair for XP2013 and the Publicity Chair for ESEC 2013. He was the Corporate Support Chair for OOPSLA'08 and OOPSLA'09. He was the Tutorial Chair for XP2008 and the Tutorial Co-Chair for ICSE'09. Fraser holds a doctorate in EE from McGill University in Montréal – and is a senior member of the ACM and the IEEE.

Based on the SPLASH 2012 theme of "New Tools for Software" topics for panel discussion are likely to include:

- What are the principal challenges in tools R&D?
- What are the challenges assessing and improving user satisfaction (e.g. small sample populations, or populations lacking requisite variety)?
- What are the appropriate measures of tool value (usability, configurability, support, cost, stability, etc.).
- How quickly can tools be commoditized – and does this tend to improve (or reduce) tool capability?

2. Kendra Cooper

KENDRA COOPER is an Associate Professor in the Department of Computer Science at The University of Texas at Dallas; she has an Adjunct faculty position at the University of Calgary. She received her Ph.D. in Electrical and Computer Engineering from The University of British Columbia. She has published extensively in journals, conferences, symposia, and workshops and serves on numerous program committees and editorial boards. Cooper has worked in the early phases of the software and systems engineering of large-scale, complex systems in industrial and academic settings. Her research interests center on modularization and re-use issues in requirements engineering/architecture for adaptive systems and engineering education. She has investigated these issues using a variety of paradigms including component-, aspect-, agent-, and recently cloud-based engineering. Software tools research is a matter of scale and scope (in other words, what

SPLASH'12, October 19–26, 2012, Tucson, Arizona, USA.
ACM 978-1-4503-1563-0/12/10.

is the tool research supposed to accomplish?). It is also a matter of the development environment – where and by whom is the tool research being conducted?

Software tools research is conducted in a variety of settings, including academia, research labs, industry, and open source projects. In academia tools are often developed by research students as proof of concept prototypes; they are used in the validation effort for their proposed solution to a problem. The students need the tool as part of their research and are motivated to develop them: the tool is needed for validation; validation is needed to demonstrate the value of their solution; demonstrated value is needed for a successful defense, graduation, and then "getting a life". The scope and scale of these prototypes are often quite narrow - they are usually intended to provide specific capabilities; they are unlikely to be extensible, scalable, easy to use, and so on. Tool teams in industry or research labs are staffed with experienced, paid professionals, who create larger, more general tools in a team-based, managed environment. Open-source tools are developed by volunteers – enthusiasts seeking to extend and contribute their expertise to the broader community.

If we want to go beyond proof of concept, prototype tools for individual research projects in academia, for example, to establish their place in the community as The School That Developed Tool X, is it possible to move towards team-based, managed development or an open-source approach? I've been trying the former on the R&D of a tool. The students (M.Sc. in computer science) work on the project as team members to gain practical development experience and improve their resumes. The students are not paid with salaries; they sign up for the course as an elective. Over the last five terms, a number of challenges have become clear when trying to adopt a more industry-like model in academia. For example:

Some of the challenges seem relatively straightforward to address. For example, graded homework assignments to build technical and soft skills have been included in the course; research assistants have been hired to reduce the impact of the turnover by becoming a repository of knowledge about the project. Overall, however, the productivity of the industry-like teams has not been strong. Recently, I've been trying a more open-source like approach, where an individual student works on one specific problem.

3. Jim Coplien

JIM COPLIEN is an old C++ shark who now does world-wide consulting on Agile software development methods and architecture. He is one of the founders of the software pattern discipline, and his organizational patterns work is one of the foundations of both Scrum and XP. He is a Certified Scrum Trainer and a Member Emeritus of the Hillside Group.

He currently works for Gertrud & Cope in Denmark, is a partner in the Scrum Foundation, a Director of Scrum Tide and the Product Owner for Scrum Knowsy®. Together with Gertrud Bjørnvig, he has written a book on Lean Software Architecture and Agile Software Development.

When I visited the temples in Nara several years ago I learned that the craftsmen who built them, and those who still maintain them, start their work by building their own tools. I have found that the same has been true in the most memorable of my software experiences over the years, because general-purpose tools are often only suitable for general-purpose work and are never ideal for any job in particular. In Bell Labs we wrote our own compilers because there were no commercial compilers that could scale to our needs or that were of high enough quality to meet the stringent engineering constraints of continuous-running software. We all have frequently built our own tools for custom tasks. These are the tools worthy of dialog and focus; commodity tools form an important but uninteresting backdrop. And most tools are commodities.

Most Agile tools suffer generation rot. Many of the tools that are the darlings of upstart Scrum teams are based on pre-Agile building blocks and pre-Agile thought. Few of the household name Agile support tools truly support what Agile development needs. While trumpeting support for communication they in fact get in its way. The tools are often used in under-engineered environments whose sluggish performance frustrates users until they are lulled into submission and resignation to poor performance. Few of them are able to capture the complex requirement relationships discovered during the elaboration of user stories. Capturing user stories is almost useless; capturing their detailing is important; structuring requirements in a way that drives high-ROI backlogs is crucial; mapping them onto a tool ideologically suited to capturing old-fashioned requirements is just stupid; believing that one metaphor and organization fits all or that it easily can be parameterized is naïve. Every Agile shop is compromised by the lack of an ideal fit between its goals and its tools, but instead of working to improve the tools, Agilists seem to celebrate their quirkiness. The great tools are home-grown.

In the Toyota Production System traditions at the roots of Scrum, high-order tools are used only to automate proven manual processes. Few Agile tools used in contemporary software shops emerge from optimized manual processes; they more often replace earlier dysfunctional processes. What's worse is that because of organizational politics or "formal envy," these tools often replace perfectly good manual processes. We're an industry that uses tools for tools' sake, in large part because almost all of us build tools. But a fool with a tool is still a fool.

A commodity tool bought off a shelf and brought into a development context is either de-contextualized or primitive by necessity. We in fact need commodity tools: the compilers, editors, and operating systems are the saws and hammers of our trade. Most of what was called a "tool" in the past generation is a commodity tool today: True power tools are generally done in-house because they rely heavily on business expertise. It's interesting to note that commodity tools are becoming increasingly invisible and that our need for tools as such is decreasing. As a user of XCode and it storyboarding facility, I haven't directly used a compiler or configuration management tool in years. XCode, while a tool, probably doesn't fit the category of being a commodity; such animals in the burgeoning tool zoo are particularly worthy of discussion. Beyond that, I fail to appreciate why SPLASH would have a panel that revolves around commodity items. I rarely see organizations get in trouble for what are tool problems.

There are some "tools" that transcend groups and organizations, and among them are games. A game is a tool that can support reflection, growth, and the proper *esprit de corps* in an Agile organization. We have recently launched a tool called Scrum Knowsy® designed to assess and build alignment within a Scrum team and to allow each team to assess its collective alignment to community norms. If you've got to have a tool, it might as well be fun and support the Agile agenda.

4. Ramya Ravichandar

RAMYA RAVICHANDAR is currently leading the *Agile Lean at Cisco* program at Cisco Systems, Inc. She is focused on evangel-

izing emerging software practices, researching on enterprise-level initiatives, and fostering an Agile Lean community in Cisco. Related to this she has presented in Agile 2010, and XP 2012 conferences. Her other interests span software quality, processes, change-tolerant systems, and requirements engineering. She has a Ph.D. in Computer Science from Virginia Tech and is a member of IEEE Computer Society.

Oh so appealing is the nifty blazingly fast in-house tool! Compare it with the magnificent full-featured enterprise-level solution. What you choose, depends on where you sit in the food chain. Therein lies the crux of the argument about tooling in our industry.

In this distributed environment where teams are connected through tools, there is a danger of the tools becoming the sole connectors. They are often used in lieu, and not as an aid to good software engineering. And can you blame the engineer, for using the aggressively marketed silver bullet? A tool's success should be driven by its effectiveness and not measured by a fixed deployment schedule; a challenge in this metrics-driven environment of instant gratification.

Tools are bound to influence the process. It is a possibility that when both become so intertwined we are in the danger of overlooking other innovations. Perhaps what we need is a tool litmus test. How do we decide that a tool is right? Is it a trade-off between: hidden costs vs. productivity gains, or resolving singular issues vs. one-size-fits-all approach? The answer lies somewhere in between.

5. Ruth Lennon

RUTH LENNON has worked for Letterkenny Institute of Technology for over 14 years in lecturing and research activities. Prior to joining LYIT, Ruth worked with a software company developing software in Delphi. Ruth is currently working on research on *Software As A Service* and BYOD. Ruth is one of the Directors of InfoSecurity Ireland. Ruth has been an active participant in the ACM's OOPSLA workshops and BOF's since 2005. Ruth is a member of the ACM and IEEE and actively encourages women to work in all aspects of engineering.

I represent an important part of the "software tools research" community -- a "user" of the tools. In the work I describe in my BYOD and Cloud paper, my institution is trying to use a wide variety of commercial and open source tools in a challenging environment. We are trying to keep costs down, we are dealing with wide variations in experience levels among learners and staff, and many of the existing tools create some difficult and unnecessary obstacles to deployment. The range of specifications of PCs in our institute leads to the need to consider issues such as ease of deployment, deployment in a cloud environment, deployment on a low end PCs and legacy operating systems, interoperability with other tools, documentation, training, ability to export useful data in a standard data format, etc. I represent a unique body of system users in that some are computer experts while others simply want to use the tools with the greatest level of simplicity. The software and hardware tools requirements for users and systems administrators can be quite disparate and this can prove challenging. Take as one example the need to provide access to Fire Modelling Software which has not changed much in the last 10 years. Some software still restricts licenses via dongles and may be significantly processor heavy. The Fire laboratory equipment software exports data onto a Windows 95 PC and has not yet be upgraded to keep with modern systems.

One problem with industry-academic research partnerships: they are often narrowly focused on research areas that are less likely to produce a real paradigm shift. There are some new innovative ideas that are "easy to launch" as a new product, such as web apps, open-source toolkits, and other products that are centered around new innovative algorithms but in relatively conventional packaging. The most creative and different new ideas are the most difficult for established industry product groups to embrace. It is already difficult for most product development groups to accept an "escaped from the lab" product idea from an internal corporate research organization, it is doubly difficult to build on the work of graduate students in a corporate environment. On the other hand, if a partnership is done right, there are plenty of benefits going in both directions. In particular, both sides of an industry-academic partnership will usually bring in useful ideas about how to convert an innovation into a marketable product.

6. Diomidis Spinellis

DIOMIDIS SPINELLIS is a Professor in the Department of Management Science and Technology at the Athens University of Economics and Business, Greece. From 2009 to 2011 he served as the Secretary General for Information Systems at the Greek Ministry of Finance. His research interests include software engineering, computer security, and programming languages. He has written two award-winning, widely-translated books: *Code Reading* and *Code Quality: The Open Source Perspective*. He is a member of the *IEEE Software* editorial board, authoring the regular "Tools of the Trade" column. Spinellis is the author of many open-source software tools, packages, and libraries. Some tools he has developed include the *UMLGraph* declarative UML drawing engine, the *CScout* refactoring editor for huge complex systems written in C, and the *ckjm* tool, which efficiently calculates Chidamber and Kemerer object-oriented metrics in Java programs. His implementation of the Unix *sed* stream editor is part of Apple's Mac OS X and all BSD Unix distributions. Spinellis holds a M. Eng. in Software Engineering and a PhD in Computer Science, both from Imperial College London. He is senior member of the ACM and the IEEE.

The development of production-quality software tools is becoming an increasingly difficult task for individual researchers. First, the application domains are becoming more complex. This includes language specifications, interfaces, and development frameworks. Kernighan and Ritchie's *The C Programming Language* (1988) was 274 pages long, whereas Stroustrup's *The C++ Programming Language* (2000) stands at a hefty 1030 pages. This rising complexity is reflected in the size of the corresponding tools. The 7th Edition Unix C compiler consisted of about 14k lines of code, whereas GCC 4.2, at 1.5 million lines, is two orders of magnitude larger. We can find at least one order of magnitude difference between the original tools and their current ancestors across the board; consider for instance RCS (1982, 25k lines) versus the current version of git (2012, 449k lines). Distribution and support is nowadays also more demanding. Whereas Ken Thompson mailed Unix source code magnetic tapes to his friends, labeling them simply "Love, ken", nowadays researchers developing tools are expected to maintain a project web site, support binary packages for diverse operating system distributions, provide a bug tracking service, answer questions on a forum, and respond to patch pull requests.

Thankfully, a number of countervailing factors help us keep the situation in balance. Modern software frameworks, like Java

and .NET, and libraries, like Boost, make modern code considerably more expressive. Mature open-source offerings can dramatically reduce the implementation effort required for a tool's front-end (e.g. LLVM), processing (MINISAT), and presentation (GraphViz). Robust package management systems enable the painless installation of tools with complex dependencies on external software. Cloud offerings, like GitHub, Google Groups, and StackExchange, simplify collaboration, bug tracking, wiki maintenance, and end-user support. The open source software community often stands ready to embrace a useful tool, offering extensions, bug fixes, and support. Finally, nowadays tool developers can often create plugins for a larger framework, like Eclipse, rather than build a tool from scratch.

The challenge for software tool researchers is to avoid being intimidated and turned-off by the large scope and complexity associated with modern tool development, learn how to harness the available countervailing factors, and benefit from them. Curriculum developers and mentors can help budding researchers by bringing them in contact with modern tool development practices.

7. Giancarlo Succi

GIANCARLO SUCCI is Professor and Dean of the Faculty of Computer Science and Director at the Center for Applied Software Engineering at the Free University of Bozen-Bolzano. Before moving to Bolzano he held several academic appointments around the world and is a global consultant for software companies and public institutions. Giancarlo Succi is a Fulbright Scholar.

The evolution in the software industry has been possible because of the presence of smart software development tools, which enabled the production or more complex systems and of better tools. For instance, it would be unimaginable today to develop the simplest native Android application writing the code using vi and running by hand or via some sort of make the compilation process. Moreover, it would not be conceivable to have an IDE like Eclipse without first having the experience of emacs, which in turn draws from the experience of vi.

Still, the research done on how people use tools is very limited and the design of new tools does follows more personal tastes and fads rather than a systematic empirical process. A major improvement toward a better understanding of how tools are used has been the introduction of AISEMA (Automated In-Process Software Engineering Measurement and Analysis) systems, which track non-invasively the activities of developers, including the tools they use [1]**Error! Reference source not found.**. However, so far it has still remained an open question how to actually analyze data coming from AISEMA systems on tools usage, especially trying to understand the usefulness and the usage of each tool together with the mutual interactions between tools. To this end specific visualization techniques could be employed, like the ones proposed in [2]. Such visualization enables the immediate understanding of the roles that specific tools have in software development and how people then use such tool, evidencing for instance that Pair Programming increases significantly the permanence in tools before doing context switching triggering higher level of attention, which then result in better work [3]. This visualization also evidences in a case study a fact always known in the software engineering community but never rigorously experimented: that only a small fraction of the installed tools are effectively used [4].

Altogether, more systematic, empirical, let me say "scientific" research has to done in this direction, determining empirically the impact of software tools, and also trying to understand how different tools mutually interact and contribute, as a cluster and not just individually, to the production of software systems.

References

[1] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Jelena Vlasenko, Does Pair Programming Increase Developers' Attention? Industrial Track of ESEC/FSE2011, Szeged, Hungary, Sept 2011

[2] Alberto Sillitti, Andrea Janes, Giancarlo Succi, and Tullio Vernazza. Collecting, integrating and analyzing software metrics and personal software process data. In EUROMICRO '03: Proceedings of the 29th Conference on EUROMICRO, page 336, Washington, DC, USA, 2003. IEEE Computer Society.

[3] Alberto Sillitti, Giancarlo Succi, Jelena Vlasenko, Toward a better understanding of tool usage (NIER Track), Proceedings of the ICSE2011 Conference, Honolulu, Hawaii, May 2011.

[4] Alberto Sillitti, Giancarlo Succi, Jelena Vlasenko, Understanding the impact of Pair Programming on Developers Attention, Proceedings of the ICSE2012 Conference, Zurich, Switzerland, June 2012.