Challenges for Software Engineers

Adrian Kosmaczewski

2008-08-03

Software Engineering is the youngest of all the professions, being born around 50 years ago, but since then it has been continually improved. Practicers have fiercely debated upon it through the years, given the extremely fast pace of the innovations in the field, and the extremely difficult and inherently dynamic nature of software. Many trends have appeared and vanished, and many others will come.

In this article I will provide a short overview of two kinds of challenges that I consider that software engineers will have to confront in the next 20 years: the human and the technical.

The Human Factor

A quick look at the agenda of the 29th Int. Conference on Software Engineering (held in Minneapolis last year, from the 20th to the 26th May 2007) shows the key themes considered by the software engineering research community as the major challenges today:

- "Improving Software Practice through Education: Challenges and Future Trends"
- "Research Collaborations between Industry and Academia"
- "Model-driven Development of Complex Systems: A Research Roadmap"
- "Source Code Analysis: A Road Map"
- "Software Reliability Engineering: A Roadmap"
- "Global Software Engineering: The Future of Socio-technical Coordination"
- "Collaboration in Software Engineering: A Roadmap"
- "Self-Managed Systems: An Architectural Challenge"
- "Software Project Economics: A Road Map"

(Source: ICSE 2007)

Mixed up with technical concerns, some presentations highlighted core problems that appears in the current state of software engineering: communication, collaboration and human issues.

The core substance of software deserves more eyes and more minds, thinking ways to describe not only the big picture (something that you can do with fancy diagrams) but also to give solutions to the problems that developers find daily while building systems up. Software is a process, but not any kind of process: a human one, maybe the most intangible of all processes; and as such, it is filled with all human brightnesses and failures.

(Myself, in 2006)

I have the deep, strong conviction that software development cannot and must not be separated from the human-side problems of forming, keeping and training teams, enhancing the internal and external communications, improving and enhancing the individual creativity as well as the ways of reaching team consensus. As a powerful example, the seminal Peopleware book by DeMarco and Lister showed that many of the most successful software companies have been those that excelled in creating human-centric environments:

In 1982, (Mitchell Kapor) founded Lotus Development Corporation, for which he is most noted. While there, he revolutionized corporate workplace culture by making diversity and inclusivity top priorities in his goal for creating an environment that attracted and retained employees. There were many "firsts" for Lotus, including being the first company to sponsor an AIDS Walk event in the mid-80's and refusing to do business with South Africa due to Apartheid.

(Sterling-Hoffman)

Thanks to a sharp hiring process, a series of innovations in their flagship spreadsheet product, and a progressive corporate culture, Lotus dominated the software landscape of the 80s. Today, Google follows very closely Lotus' steps (Google, 2007a), and their brilliant results in the last few years seem to confirm this trend. Google for example allows their employees to use 20% of their time in their own projects (Google, 2007b). This is resulting in an incredible amount of code, used internally and also released as open-source projects:

Google is a fantastic company to work for. I could cite numerous reasons why. Take the concept of "20 percent time." Google engineers are encouraged to spend 20 percent of their time pursuing projects they're passionate about. I started one such exciting project some time back, and I'm pleased to announce that Google is releasing the fruits of this project as an open source contribution to the Macintosh community. That project is MacFUSE, a Mac OS X version of the popular FUSE (File System in User Space) mechanism, which was created for Linux and subsequently ported to FreeBSD.

(Google Mac Blog, 2007)

The empowerment of both the individual and the team (the emphasis is important here) is key for a successful software project.

Parallelization

Herb Sutter has put it very clearly: technically speaking, since the beginning of the decade, there is no way for getting more processing power without jumping to multicore architectures:

The key question is: When will it end? After all, Moore's Law predicts exponential growth, and clearly exponential growth can't continue forever before we reach hard physical limits; light isn't getting any faster.(…) If you're a software developer, chances are that you have already been riding the "free lunch" wave of desktop computer performance.(…) Right enough, in the past. But dead wrong for the foreseeable future.

(Sutter, 2005)

The problem is that more cores do not necessarily mean more computing power, because the jump done by chip manufacturers has not (yet) been completely followed by the software community. Of course there is the concept of "threads", and multi-threaded applications can benefit of performance boosts when running on multicore hardware platforms; however, a number of myths have to be debunked, as the common "2 x 3GHz = 6GHz" (as explained by Sutter here: http://www.ddj.com/showArticle.jhtml?documentID=ddj0503a&pgno=3), and even more importantly, creating multithreaded applications is not easy. At all.

A couple of months ago, in the "Questions and Answers" of LinkedIn.com I answered an interesting question about parallelization; the following excerpt of my answer pretty much summarizes my opinions about the current state of multithreading, as well as some challenges that are raised for the future:

The problem is simply that the "streamline" programming languages languages do not provide good ways to code multithreaded applications. (…) Not at all. The problem is real, since multithreading applications are extremely complicated to think of, let alone develop properly. A line of code in a high-level language could mean several hundred instructions in a processor; and depending on the sharing algorithm used at the CPU level, each one of these instructions might be executed separately, sharing resources with other processes. So what happens when? (…) What I mean is that the fact that the JVM and the CLR support threads does not make good .NET or Java developers good multithreading developers by default. It's a different mindset; who is accessing your resources? (…) I think that as long as programming languages do not take multitasking and multithreading as base features (and not as mere library or API add-ons) we will continue struggling with single-threaded applications that collide with each other.

(Myself, this time on LinkedIn Answers, 2007)

I think that the challenge of parallelization is not only an extremely tough one, requiring what Thomas Kuhn calls a "paradigm shift", but also an extremely

huge business opportunity; after all, while the top of the Chinese ideogram for "Crisis" means "Danger", the bottom part means "Opportunity" (Mary R. Bast, 1999).

Very Large Systems

I also think that software systems will invariably get bigger and bigger. And given the historically high risk of failure of software projects, the dependency on software of the modern society, the pervasiveness of the Internet, the low prices of connectivity and the overall globalization, it is more important than ever to get ready for those challenges.

In July 2006, the well known Software Engineering Institute of the Carnegie Mellon University published an impressive report (freely downloadable) called "Ultra-Large-Scale (ULS) Systems: The Software Challenge of the Future":

The study brought together experts in software and other fields to answer a question posed by the U.S. Army Office of the Assistant Secretary of the U.S. Army (Acquisition, Logistics & Technology): "Given the issues with today's software engineering, how can we build the systems of the future that are likely to have billions of lines of code?" Increased code size brings with it increased scale in many dimensions, posing challenges that strain current software foundations. The report details a broad, multi-disciplinary research agenda for developing the ultra-large-scale systems of the future.

(SEI, CMU, 2006)

The 150-page long report gives an extremely detailed vision of the challenges raised by complex systems, in the following areas:

- Design
- Monitoring
- Human interaction
- Computational Engineering
- Deployment
- Legal issues

The report provides interesting conclusions, highlighting the methodologies and techniques that will required to tackle these systems efficiently, among them the role of the W3C, the forthcoming trends of grid computing and parallelization, the Model-Driven Architecture (MDA) initiative of the OMG, and finally the development of larger Service-Oriented Architectures (SOA) platforms, such as .NET or J2EE (page 41 of the report).

The report also places a strong emphasis in the concept of socio-technical ecosystems and I think it's worth a read by everyone interested in software engineering.

Conclusion

Given its youth, we have yet to see the most important developments in software engineering. However, it is extremely difficult to predict the future in this industry: Bill Gates himself published a book in 1995, "The Road Ahead", where he only slightly talks about the World Wide Web:

"The Road Ahead" appeared in December 1995, just as Gates was unveiling Microsoft's master plan to "embrace and extend" the Internet. Yet the book's first edition, with its clunky accompanying CD-ROM, mentioned the Web a mere seven times in nearly 300 pages. Though later editions tried to correct this gaffe, "The Road Ahead" remains a landmark of bad techno-punditry – and a time-capsule illustration of just how easily captains of industry can miss a tidal wave that's about to engulf them.

(Salon.com, 2000)

In any case, I think that there are important challenges in our industry: the need for better human management, the jump to multicore architectures and multiprocessing, and the ever-growing size of software projects. These three elements will without any doubt change the shape of the industry in the years to come, and raise new challenges in turn.

References

Adrian Kosmaczewski on LinkedIn Answers, "For the software architects out there, do you feel there is an impending paradigm shift in the software development model, towards"parallel computing" models?", January 2007, [Internet] http://www.linkedin.com/answers?viewQuestion=&questionID=7804&askerID=4194838 (Accessed June 3rd, 2007)

Adrian Kosmaczewski on Kosmaczewski.net, "What will the Software Architecture discipline look like in 10 years' time?", March 16th, 2006 [Internet] http://kosmaczewski.net/2006/03/16/software-architecture-future/ (Accessed June 3rd, 2007)

Bast, Mary R; "Crisis: Danger & Opportunity", 1999 [Internet], http://www.breakoutofthebox.com/crisis.htm (Accessed June 3rd, 2007)

DeMarco, Tom & Lister, Timothy, "Peopleware - Productive Projects and Teams, 2nd Edition", 1999, Dorset House Publishing, ISBN 0-932633-43-9

Google, "Top 10 Reasons to Work at Google", 2007a [Internet] http://www.google.com/jobs/reasons.html (Accessed June 3rd, 2007)

Google, "What's it like to work in Engineering, Operations, & IT?", 2007b, [Internet] http://www.google.com/support/jobs/bin/static.py?page=about.html (Accessed June 3rd, 2007)

Google Mac Blog, "Taming Mac OS X File Systems", January 11th, 2007,

[Internet] http://googlemac.blogspot.com/2007/01/taming-mac-os-x-file-systems.html (Accessed June 3rd, 2007)

ICSE, "Future of Software Engineering", 2007, [Internet] http://web4.cs.ucl.ac.uk/icse07/index.php?id=104 (Accessed June 3rd, 2007)

Software Engineering Institute, Carnegie-Mellon University, "Ultra-Large-Scale (ULS) Systems - The Report", July 2006, [Internet] http://www.sei.cmu.edu/uls/ (Accessed June 3rd, 2007)

Salon.com, 2000 [Internet], "Why Bill Gates still doesn't get the Net", [Internet] http://archive.salon.com/21st/books/1999/03/cov_30books.html (Accessed June 3rd, 2007)

Sutter, Herb; "A Fundamental Turn Toward Concurrency in Software", 2005, [Internet] http://www.ddj.com/dept/architect/184405990 (Accessed June 3rd, 2007)

Sterling-Hoffman, "Opening Doors To Higher Education", [Internet] http://www.sterlinghoffman.com/newsletter/articles/article140.html (Accessed June 3rd, 2007)

Wikipedia, "Thomas Kuhn" [Internet], http://en.wikipedia.org/wiki/Thomas_Kuhn (Accessed June 3rd, 2007)