The future of software engineering

Perdita Stevens

School of Informatics
University of Edinburgh

Plan

This course is designed from my perspective, of course, and you have watched or will watch my inaugural lecture which says where I think SE is going.

The idea today is to play devil's advocate and look at some very different perspectives.

I will not put details on the slides, and there may not be a recording. I will not examine specific facts from what I say, but I could (have) set a question for which having heard it will help.

Is "software engineering" a thing?

Could it be, should it be?

Lots of software is not built by software engineers...

Bug-free software

Could we achieve it? How?

`http://deepspec.org`

"In our interconnected world, software bugs and security vulnerabilities pose enormous costs and risks. The Deep Specification project addresses this problem by showing how to build software that does what it is supposed to do, no less and (just as important) no more: No unintended backdoors that allow hackers in, no bugs that crash your app, or your computer, or your car. What the software is supposed to do is called its specification."

Process, languages, tools?

Which are most important to getting software that works?

How do advances in SE get made?

Do they just happen?

Do universities matter?

Does Informatics as a subject have a future?

or will it simply be taken for granted that every discipline involves information and computation?

What does Informatics do that couldn't be done elsewhere? Is it really a discipline in its own right?

January/February 2016 IEEE Software

This special issue collected a number of papers under the banner "The future of software engineering".

Let's look at the abstracts: you may like to follow up by reading the papers (but this is not required).

Toward Data-Driven Requirements Engineering

Walid Maalej ; Maleknaz Nayebi ; Timo Johann ; Guenther Ruhe

Nowadays, users can easily submit feedback about software products in app stores, social media, or user groups. Moreover, software vendors are collecting massive amounts of implicit feedback in the form of usage data, error logs, and sensor data. These trends suggest a shift toward data-driven user-centered identification, prioritization, and management of software requirements. Developers should be able to adopt the requirements of masses of users when deciding what to develop and when to release. They could systematically use explicit and implicit user data in an aggregated form to support requirements decisions. The goal is data-driven requirements engineering by the masses and for the masses.

Requirements: The Key to Sustainability

Christoph Becker ; Stefanie Betz ; Ruzanna Chitchyan ; Leticia Duboc ; Steve M. Easterbrook ; Birgit Penzenstadler ; Norbet Seyff ; Colin C. Venters

Software's critical role in society demands a paradigm shift in the software engineering mind-set. This shift's focus begins in requirements engineering. This article is part of a special issue on the Future of Software Engineering.

Reducing Friction in Software Development

Paris Avgeriou ; Philippe Kruchten ; Robert L. Nord ; Ipek Ozkaya ; Carolyn Seaman

Software is being produced so fast that its growth hinders its sustainability. Technical debt, which encompasses internal software quality, evolution and maintenance, reengineering, and economics, is growing such that its management is becoming the dominant driver of software engineering progress. It spans the software engineering life cycle, and its management capitalizes on recent advances in fields such as source code analysis, quality measurement, and project management. Managing technical debt will become an investment activity applying economic theories. It will effectively address the architecture level and will offer specific processes and tools employing data science and analytics to support decision making. It will also be an essential part of the software engineering curriculum. Getting ahead of the software quality and innovation curve will inevitably involve establishing technical-debt management as a core software engineering practice. This article is part of a special issue on the Future of Software Engineering.

Crowdsourcing in Software Engineering: Models, Motivations, and Challenges

Thomas D. LaToza ; Andr van der Hoek

Almost surreptitiously, crowdsourcing has entered software engineering practice. In-house development, contracting, and outsourcing still dominate, but many development projects use crowdsourcing-for example, to squash bugs, test software, or gather alternative UI designs. Although the overall impact has been mundane so far, crowdsourcing could lead to fundamental, disruptive changes in how software is developed. Various crowdsourcing models have been applied to software development. Such changes offer exciting opportunities, but several challenges must be met for crowdsourcing software development to reach its potential.

Speed, Data, and Ecosystems: The Future of Software Engineering

Jan Bosch

An evaluation of recent industrial and societal trends revealed three key factors driving software engineering's future: speed, data, and ecosystems. These factors' implications have led to guidelines for companies to evolve their software engineering practices. This article is part of a special issue on the Future of Software Engineering.

Intelligently Transparent Software Ecosystems

James Herbsleb ; Christian Kstner ; Christopher Bogart

Today's social-coding tools foreshadow a transformation of the software industry, as it relies increasingly on open libraries, frameworks, and code fragments. Our vision calls for new intelligently transparent services that support rapid development of innovative products while helping developers manage risk and issuing them early warnings of looming failures. Intelligent transparency is enabled by an infrastructure that applies analytics to data from all phases of the life cycle of open source projects, from development to deployment. Such an infrastructure brings stakeholders the information they need when they need it.