

Panel

Software Development: Arts & Crafts or Math & Science?

Jim Haungs (Chair)
IBM

Martin Fowler
ThoughtWorks

Ralph Johnson
University of Illinois at Urbana-
Champaign

Steve McConnell
Construx, Inc.

Richard Gabriel
Sun Microsystems

ABSTRACT

We've have been proposing formal mathematical methods of software development for nearly as long as we've been developing software. CASE tools were a bust, and Gödel long ago nullified any hope of building a system that is both complete and consistent. Model-driven development gets trotted out in a new outfit once every decade as the next "silver bullet," but we're still far from drawing pictures that generate code.

Test-driven development almost produces executable specifications, but there's nothing mathematical or provably correct about them. Quality assurance happens during coding, but there's nothing to ensure that the entire system gets built, or that it's what the customer wants.

Agile methods propose "emergent design," where BDF (big design up front) is old-fashioned and limiting. But agile methods work best with small groups of clever people to implement them; they have a hard time scaling.

The open-source movement, in which hobbyist programmers write code simply for the love of programming, has produced some of the best software in the industry. Is it good because people love what they're doing and build it for themselves to use? Or is it good because there are hundreds of eyes on the code?

Is software development a science or an art? Does it depend on mathematicians or on craftspeople? Are these viewpoints reconcilable?

Categories and Subject Descriptors

K.7.0 [The Computing Profession]: General

General Terms

Management, Economics, Experimentation, Legal Aspects

Keywords

Software Engineering, Extreme Programming, Agile Development, Formal Methods

1. Martin Fowler

fowler@acm.org

This topic gets debated all the time, but the only conclusion we can currently reach is that we don't know the answer. Furthermore

we don't know if there is a single answer, or if there are several equally valid ways of looking at things.

I can imagine a world where there are several distinct schools of software development. Each one has a particular community behind it and a network of practices and principles. Each school targets a particular range of software, but there's a lot of overlap between schools. People orient towards the schools depending on their experience and their personality.

Indeed I think this is what we currently have. But I'd like to see more understanding of the characteristics of these schools separated from the usually hopeless attempts to say which is better.

2. Ralph Johnson

johnson@cs.uiuc.edu

Both sides are wrong. Software development is engineering. Engineering is a creative activity based on artifice and craftsmanship as much as it is on math and science. As the Engineering FAQ puts it, engineering is the bridge between art and science: (<http://www.tcnj.edu/~rgraham/whatare.html>)

Real engineers appreciate beauty and craftsmanship, as well as powerful mathematical models. They know that the models only go so far, and that when the models fail, they will have to invent new models. Scientists try to discover what is, engineers create a new reality. Software development is engineering.

We have had a series of keynote speakers at OOPSLA who have talked about engineering. When Paul MacCready talked about the human-powered aircraft, it was obvious that he relied on both artifice and science. Whenever his aircraft crashed, he strengthened the parts that broke, and if a part never broke then he made it thinner and lighter. But he also used the latest materials and studied the latest works in aerodynamics. He didn't rely on computer models, but he did let his intuition be informed by the latest theoretical work.

Henry Petroski spoke most directly to the subject. Engineering is about invention. It is informed by science, but is driven by learning from failure as much as from science. That is true of software, as well. Testing is about finding out how our system fails and fixing it. Because software is soft, we can repeatedly change it until it conforms to requirements.

We need to improve the science of software. It is important to continue research into formal methods, model checking, languages based on new paradigms and the like. But most research never makes it into practice. It is unrealistic to expect

Copyright is held by the author/owner(s).

OOPSLA '04, Oct. 24–28, 2004, Vancouver, British Columbia, Canada.
ACM 58113-833-4/04/0010.

most engineers to pay much attention to scientific research. This research has first to pass through engineering researchers who figure out how to put it into practice.

It is important to have meetings where scientists, engineering researchers, and practitioners come together. Creativity comes from borrowing ideas from others, and the more we have to borrow, the better.

3. Steve McConnell, Steve.McConnell@construx.com

Software development as currently practiced is art, is science, is mathematics, and is craft. That it is all these things contributes to the highly variable results and industry experiences on projects conducted using these various styles.

In the long run, what software development evolves into will be determined by economic forces. Most software is developed to serve business needs and to satisfy economic goals. The development approaches that best satisfy industry's full configuration of economic goals will be the approaches that prevail. In some cases, the need to protect the public's well-being will also be a consideration.

Historically, the discipline that fuses art, science, mathematics and craft toward practical, economic ends is engineering. A view of software development as engineering explains why sometimes the most economical approach to software development consists of Agile practices, sometimes of plan-driven approaches, sometimes of creative-studio approaches. As in other kinds of engineering, the economically efficient ratio of engineering to craft varies depending on the application's size, complexity, safety-criticality, and other factors. Software projects range from software skyscrapers to software doghouses and from software bridge building to software landscaping. Skilled craftspeople can build doghouses and create beautiful landscaping, but formally trained engineers should be involved in safety-critical or economically critical bridges and skyscrapers.

4. Richard Gabriel rpg@dreamsongs.com

Software Engineering is Math!

Mathematics is lovely, and being a mathematician is great fun. I have both a bachelor's and master's degree in mathematics, and came fairly close to getting a PhD in math, too. Mathematics is about axiomatically creating worlds and exploring them. The main activity is proposing and proving theorems. I estimate that during the 8 years I studied mathematics I proved between 5,000 and 10,000 theorems. At least one of them was a major proof—a theorem Dana Scott could not prove.

One way to think of software is that the requirements are like a proposed theorem, and the actual software is its proof. The theorem is of the form: There exists a program that satisfies these requirements; and the proof is constructive: The software is constructed (and shown to satisfy the requirements, usually through testing). This way we can look at constructing software as a mathematical process. There is one small problem with it: Constructing a proof is an art. Being able to prove theorems is something that one can be good or bad at—one gets better with practice, and one gets better the fastest by learning under a master. People can be talented at it, and the talent is only sometimes related to intelligence. Mathematics is a practice in which proving—trying to generate a proof—is a mechanism for fine-tuning proposed theorems: If a theorem seems like it should be true but can't be proved, then perhaps some of the concepts are not right or not formulated properly.

We think that math and science are somehow more disciplined or methodical than art and craft, but they are all human activities, and when examined closely they share more than they differ. Mathematics is invention, and physics has shown that inventions of the human mind can be useful. Hm, just like software.