

Progress toward an Engineering Discipline of Software

Publisher: IEEE[Cite This](#)[Mary Shaw](#) [All Authors](#)**1**
Cites in
Paper**475**
Full
Text Views**Abstract**[Authors](#)[Citations](#)[Keywords](#)[Metrics](#)[More Like This](#)**Abstract:**

Is "software engineering" really engineering? The term was coined in 1968 to call attention to problems with software production. Both theory and practice for software have evolved since then, but do we yet have a true engineering discipline? This keynote sketches the evolution of software engineering, drawing on civil engineering and software architecture for examples that show the progressive codification of informal knowledge toward rigorous models and tools. This provides the basis for assessing the maturity of the field and identifying our next challenges.

Published in: [2016 IEEE/ACM 38th International Conference on Software Engineering Companion \(ICSE-C\)](#)**Date of Conference:** 14-22 May 2016**Publisher:** IEEE**Date Added to IEEE Xplore:** 23 March 2017**Conference Location:** Austin, TX, USA**► ISBN Information:**

Engineering is the discipline of creating cost-effective solutions to practical problems by applying codified knowledge, building things in the service of mankind. Engineering enables ordinary people to do things that would otherwise require virtuosity. Classical engineering disciplines have emerged from craft practice and commercialization through the infusion of codified knowledge and science. This is evident, for example, in the evolution of arch bridges in civil engineering.

Important characteristics of engineering practice include

- Limited time, knowledge, and resources force decisions on tradeoffs
- Well-codified knowledge, preferentially scientifically-founded, shapes design decisions
- Reference materials make knowledge and experience available
- Analysis of a design predicts properties of its implementation

Software engineering has also followed this emergence pattern since the 1968 NATO conference. Systematic software development methods stabilized commercial software development, and fundamental ideas from computer science provided a scientific basis for codified knowledge. Progress can be seen in the increasing scale of abstraction – the conceptual size of the elements in the software development vocabulary.

Software architecture provides a good example of this pattern. There was a craft practice by the 1980s: software developers often had explicit descriptions of the organization of their systems, but the vocabulary was idiosyncratic, and the descriptions were ad hoc and largely system-specific. Beginning in the late 1980s, catalogs of common idioms and architectural styles began to support a common vocabulary, and product line architectures for specific domains supported systematic development of practical systems. Since then, the scientific basis has been growing through architecture description languages, refined taxonomies and product lines, formal analysis of certain architectural patterns, and evaluation techniques for quality attributes. Several specific styles have their own established ecosystems and communities of practice. However, the field still lacks systematic theories or guidance on the deliberate selection of an architecture to fit a task. Where, then, do we stand on developing a fully mature engineering discipline for software? It's emerging, but still very spotty.

- Certainly, we create systems in which limited time, knowledge, and resources force decisions on tradeoffs
- We have some guidance for design decisions, but not nearly enough nor systematic enough

- Reference materials and documentation are widely neglected. We have scientific papers, textbooks, narratives for practitioners, online discussion groups, and searchable APIs for specific systems – but well-curated reference are sorely lacking.
- We have a rich set of analysis techniques, but most focus on the code rather than the design. We have rich simulation systems in certain areas. But we still lack a widespread culture of exploring design alternatives before implementation.

Recent advances in computing technology and access to computing will change the character of further engineering progress. These do not invalidate the underlying principles, but they will affect the realizations – in some cases by changing the approach, in some cases by adding complexity.

First, Moore's Law increases in computing power, together with algorithms that exploit the improved technology, have shifted the balance between indexing and search. Finding information by looking it up in a well-curated edited collection has long been the norm, but the cost of creating the reference material is large and the editing process introduces significant delays. With respect to finding the sort of material that was once organized in handbooks and reference manuals, the current state of search is largely search for specific strings in code bases and search for topics in discussion forums. Neither of these is particularly satisfactory, but the promising path is now improvement in search rather than lamenting the cost and lag of authoritative reference material.

Second, the increasing abstraction and scale of available components and frameworks is shifting the software development task from writing code for components to composing existing components and shepherding the integration and evolution of the composed systems. However, analysis methods for the architecture and system level lag behind analysis methods for code, and tool support for these compositions tends to be specialized for specific architectures or product families. Reducing the resulting friction in selecting and evolving system architectures is a promising path.

Third, nearly-universal Internet coverage enables continuous and automatic deployment of software as well as access to real-time data. This simplifies software distribution and important (e.g., security, reliability) updates, and a new model of development and maintenance, “devops”, supports this. When the deployment is at extremely large scale, it allows the distributor of the software to explore design variants in the live system. However, the associated changes complicate reasoning about correctness and quality properties of the resulting systems, and in the worst case the systems are in “perpetual beta” and testing is outsourced to the users. Finding a balance between the benefits of continuous deployment and the very real need to provide assurances about the resulting systems is not simply a promising, but an essential path.

Fourth, software is increasingly embedded in physical systems, including networked systems. These cyber-physical systems are not deterministic, because of uncertainties that arise from the physical setting. Their software must therefore actively monitor and adapt their behavior in addition to its primary function. Further, the ability of the physical system to act on its environment increases the potential consequence of either accidental or malicious failure. Promising paths toward engineering include (a) bringing better understanding of control theory principles into adaptive systems, (b) improving support for privacy, security, reliability, and data validity, and (c) developing theories that determine the level of engineering discipline and assurance needed for a system based on the consequences of failure and the degree of human oversight that may prevent that failure.

Finally, in many modern systems the active participation of human users is integral to the function of the system. These cyber-social systems include social networks, crowdsourcing systems, open source development, and applications that support the “gig economy”. The number of highly trained computer professionals involved in cyber-social systems is dwarfed by the number of other people who are contributing code, data, or other resources to the systems. The latter are often not aware of nuances about security, privacy, maintainability, or even complete case analysis. Computing gurus no longer stand between users and the computers, and this disintermediation is not reversible. It is essential to civilize this electronic frontier; promising paths include improved user models (and the commitment of professional developers to respect them), technology to make development less dependent on arcane knowledge, and policy that aligns software more closely with the usual expectations for products.

Authors	▼
Citations	▼
Keywords	▼
Metrics	▼

IEEE Personal Account

CHANGE
USERNAME/PASSWORD

Purchase Details

PAYMENT OPTIONS
VIEW PURCHASED
DOCUMENTS

Profile Information


COMMUNICATIONS
PREFERENCES
PROFESSION AND
EDUCATION
TECHNICAL INTERESTS

Need Help?

US & CANADA: +1 800
678 4333
WORLDWIDE: +1 732
981 0060
CONTACT & SUPPORT

Follow



About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | IEEE Ethics Reporting  | Sitemap | IEEE Privacy Policy
A public charity, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2025 IEEE - All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies.