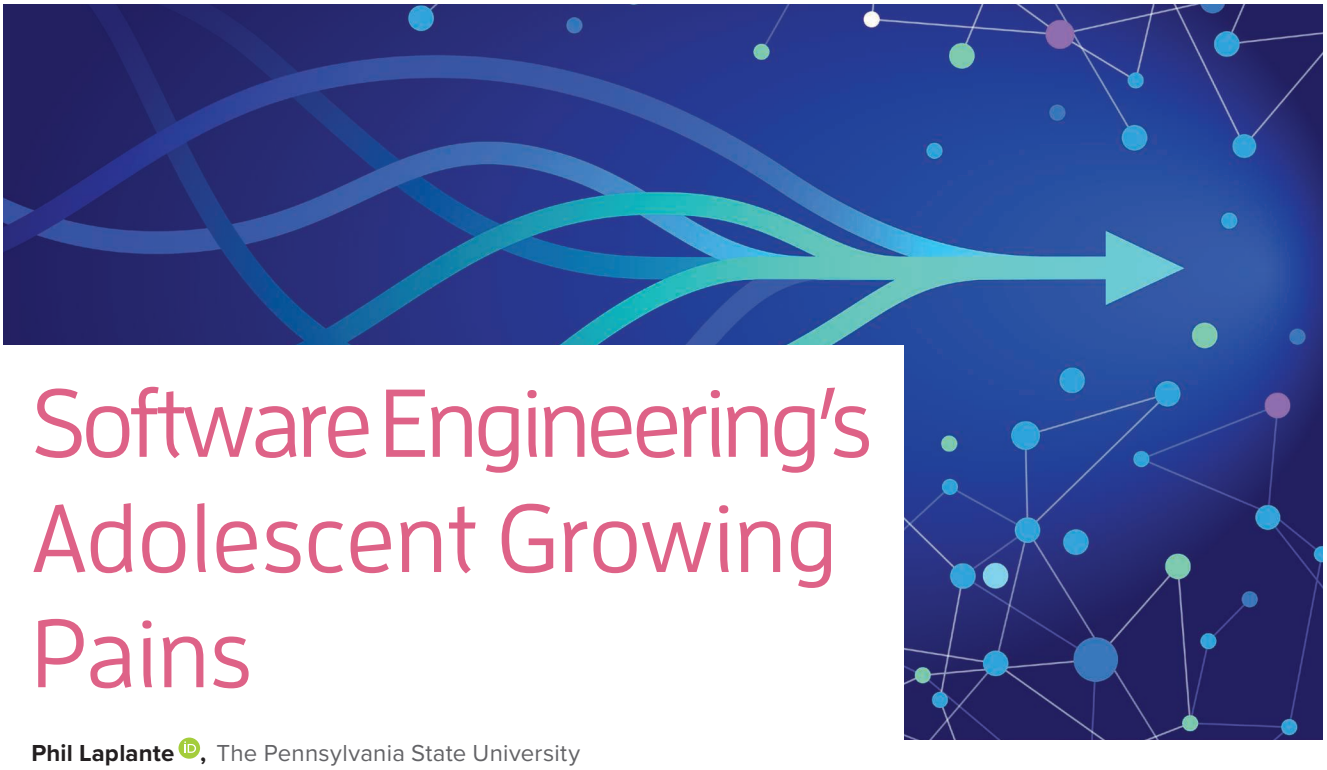

Software Engineering's Adolescent Growing Pains

Phil Laplante, The Pennsylvania State University

Software engineering has a long way to go to reach the stature of other mature scientific or engineering disciplines. Based on the history of these professions, software engineering may not even be halfway from "infancy" to a mature profession.

A recent article in the Forbes Technology Council "an invitation-only community for world-class CIOs, CTOs, and technology executives" entitled "It's Time for Software Engineering to Grow Up,"[1] caught my eye. The author contends that "headcount has always been the primary lever for [software] engineering leaders to substantially increase output." The article further argues that one reason that metric-based management is not used in software engineering is over "fear of alienating a volatile and rare resource—the software engineer. Software engineering is a creative craft. Some operational metrics may be 'big brotherly' and would stifle the creativity that leads to innovation."

These negative views of software engineers and software engineering are not based on reality. For example, my own recent surveys of practitioners showed that almost 50% of software engineers were using metrics-based requirements management tools (75% for agile projects) and of software testing position ads 56% asked for testing-tool skills as a requirement or as a preference.[2,3] The software engineering profession embraces tools. In fairness, the author of the curious article is mostly promoting a "software project visibility tool," but the assertion that software engineering "needs to grow up" got me thinking—if software engineering were a human, where would it be on the development spectrum? I think adolescence.

SOFTWARE ENGINEERING VERSUS MEDICINE OR ELECTRICAL ENGINEERING

For almost 60 years, software engineering (or whatever name the discipline has gone under over the years: coding,

Digital Object Identifier 10.1109/MC.2022.3200281
Date of current version: 24 October 2022

0018-9162/22©2022IEEE

EDITOR **PHIL LAPLANTE**
The Pennsylvania State University;
plaplante@psu.edu

programming, systems analysis, and so on) has been trying to define itself and be regarded with the same respect as other engineering disciplines and the medical profession.[4] Anyone can write software, so what is a "software engineer?" Software "engineers" are almost never licensed in any country in the same way that other engineers are licensed. Why? I think that even though software engineering's history includes curriculum standardization, certifications, and attempts at licensure, it is not yet mathematically mature. But other science and mathematically based professions took a very long time to reach maturity. For example, fewer than 200 years ago the medical profession was a hodgepodge of practitioners of various approaches to healing, including many untrained and unskilled quacks. It was until the late 20th century that the medical profession became profession we recognize today. What brought the medical doctors together was unification behind science.[5]

Likewise, electrical engineering took longer than 200 years to resemble its modern form. While luminaries such as Leyden, Galvani, Franklin, and Volta were experimenting with electricity almost 300 years ago, there were few foundational equations until the late 1890s. I have an original 1898 copy of Trowbridge's "What is Electricity?" (1896), an early and seminal text on the subject. The 315-page book has almost no equations—it is mostly long narratives and a smattering of diagrams. Yet a modern electrical engineering textbook is densely packed with mathematical theory and important equations, such as Ohm's law, Maxwell's equations, and Kirchoff's laws. How did electrical engineering mature to this "adulthood"?

Frankly, reaching this maturity required intellectual and legal battles among and between experimentalists and theorists. The era from the 1850s through early 1900s featured battles over the theory, application, and patent superiority of telegraph, telephone, light bulb, and phonograph technologies, of ac versus dc for the power grid, batteries, motion pictures, radio, and more. Along the way, significant theory was developed. In his fantastic book on Edison, Morris relates the battles of this skilled experimentalist with many important and practical inventions, versus the theorists with few practical achievements during that transformative period of the discipline.[6] These battles forced theory to harmonize with experiment and practice—yielding the modern, scientific, and mathematically rigorous discipline of electrical engineering.

THE ENGINEERING PART OF SOFTWARE ENGINEERING

Rigor in software engineering requires the use of mathematical techniques. Of course, any kind of scientific programming, embedded control systems, and so on use mathematical algorithms. But real engineering of software further requires that there be a rigorous mathematically based approach to the specification, design, coding, and documentation of the software. Formal methods might fit this characterization, but its use is so limited that stories of its successful implementation in practical systems are article worthy.

Software engineering has a problem, and it relates to the lack of grand theories. Can you name five foundational theories or formula of software engineering, based on mathematics? I can only think of a few. To me, the most practical is McCabe's Cyclomatic Complexity Theorem (which demonstrates the maximum number of linearly independent code paths in a program) and is used by software engineers for software test planning, code complexity analysis, and more. Other theories have only theoretical applications of what is possible or impossible in programming. For example, the Boem–Jacopini Theorem, that is, that all programs can be constructed using only sequential and goto statements. I can't think of any others.

Of course, we can defer to theorems and formulas from probability and statistics for applications, for example, in artificial intelligence, failure analysis, timing analysis, testing, and so on, but none of these uses are unique to software engineering. And while there are many metrics used in software project management (for example, function or use case points, churn, and so on), these are informal and certainly do not form a comprehensive engineering approach to software.

There are no grand mathematical theorems of writing software, though there are rules and principles of structure, organization, and grammar. We have various principles of object-oriented design, patterns, and so on and these are useful and important. But of the thousands of papers published in various transactions and journals, I can think of only a few important theoretical papers that strongly influence modern, software engineering practices. For example, those that introduced McCabe's metric, the Chidamber–Kemmerer metrics, and Parnas Partitioning, but there are few others.

Electrical engineering has its unifying Maxwell's equations: four vector equations that define the fundamental relationships between electricity and magnetism, which are among the most important formulas in all physical sciences. The original formulation by Maxwell was in 20 quaternion equations with 20 variables, which were comprehensible to only a few individuals and practically useless. These equations were simplified to their modern form (four equations in six variables) by Heaviside, which are practical and ready for use. That is how a discipline grows up.

I was once skeptical that software engineering could become a true engineering discipline in my lifetime. But then I was given hope that this could happen much sooner, I even

helped launch the first software engineering licensing effort in the United States. But this effort fizzled after only a few years for a number of reasons, including that there is not enough engineering in software engineering to persuade other engineering disciplines that we deserve to join their ranks. Now I realize my original opinion was correct—it will be decades for software engineering to be a true engineering discipline.

It took both the medical profession and electrical engineering at least 200 years to move from an informal science to a more rigorous one. Software engineering researchers and practitioners need to focus on the science and mathematics and find more ways to make the theory practical. We need to find grand theorems that unify the many informal practices, that may work, but that need to be mathematically based. Only then, as in the medical profession and electrical engineering, can the discipline come together to agree on a body of knowledge that joins theory and practice.

We are only about 60 years into software engineering as a discipline. We lack unifying principles and a rigorous mathematical basis that influences daily practices. Thus, software engineering is just entering its adolescence.

REFERENCES

1. S. Nabar. "It's time for software engineering to grow up." Forbes Technology Council. Accessed: Jul. 26, 2022. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2022/08/04/its-time-for-software-engineering-to-grow-up/?sh=3d8d18d57c4e
2. M. Kassab and P. Laplante, "The current and evolving landscape of requirements engineering in practice," *IEEE Softw.*, early access, Nov. 2022, doi: 10.1109/MS.2022.3147692.
3. M. Kassab, P. Laplante, J. Defranco, V. V. G. Neto, and G. Destefanis, "Exploring the profiles of software testing jobs in the United States," *IEEE Access*, vol. 9, pp. 68,905–68,916, May 2021, doi: 10.1109/ACCESS.2021.3077755.
4. P. Laplante, "A brief history of software professionalism and the way forward," *Computer*, vol. 53, no. 9, pp. 97–100, Sep. 2020, doi: 10.1109/MC.2020.3004017.
5. P. A. Laplante, "Professional licensing and the social transformation of software engineers," *IEEE Technol. Soc. Mag.*, vol. 24, no. 2, pp. 40–45, Jun. 2005, doi: 10.1109/MTAS.2005.1442380.
6. E. Morris, *Edison*. Munich: Random House Trade Paperbacks, 2020.

PHIL LAPLANTE is a professor of software and systems engineering at The Pennsylvania State University, State College, PA 16801 USA, a Fellow of IEEE, and an associate editor in chief of *Computer*. Contact him at plaplante@psu.edu.