Software engineering has reached a stage that more closely resembles quackery than engineering. Everyone extols the virtues of a particular procedure, style, technique, or set of rules that will tame the software monster and lead to an elusive promised land. Subjective opinions are often strongly held, vigorously advocated, and more prevalent than real objective data. Who has not heard claims that “new development environment X improves developer productivity,” that “using life cycle Y will shorten the development process,” and that “new development approach X can build more reliable software”? Unfortunately, most of these claims have no foundation. No evidence exists to demonstrate that using the aforesaid artifacts will really improve such things as productivity, development time, and the number of errors. For example, some claim that object-oriented programming will improve productivity or program quality, but evidence also exists to the contrary (see, for example, Les Hatton’s work “Does OO Sync with How We Think?” in IEEE Software, May 1998).

In software engineering, we distinguish valid ideas from false beliefs by applying the test of time. We judge an idea’s certainty by whether or not people use it. If lots of people use it, it seems to be certain; if not, we assume that it’s false and will be ravaged by time. This modus operandi better suits disciplines such as the fashion industry. But, supposing we accept this natural selection of ideas, what happens to development projects that use ideas later deemed false? How long do we have to wait before we know that an idea works? More importantly, even if the idea is commonly used, are the project settings in which it’s usually employed similar to the settings where we want to use it?

From art to engineering

Confronted by a confusing array of options for producing software, software engineers need proof of a particular approach or technique’s superiority. They must know the clear-cut benefits of one approach versus another. This need to work with facts rather than assumptions is a property of any engineering discipline. As Bruno Latour and Steve Woolgar discuss in their book Laboratory Life: The Construction of Science Facts (Princeton Univ. Press, 1986), knowledge goes through stages:

- Conjectures or speculations
- Disputed statements
- Undisputed statements
- Established and accepted facts

A statement’s ranking depends on its factuality status, which we determine by testing.
or empirical comparison with reality. For software development to really be an engineering discipline and be able to predictably build quality software, we must transition from software development based on speculation to software development based on facts. We must lay aside perceptions, bias, and market-speak to provide fair and impartial analysis and information.

This shift from art to craft and from craft to engineering has taken place in other disciplines that build products, such as in aeronautical engineering recently. Walter Vincenti’s book, What Engineers Know and How They Know It (Johns Hopkins Univ. Press, 1990), discusses the advance of aeronautical engineering between the world wars, when engineers were able to “translate an amorphous qualitative design problem into a quantitative problem susceptible of realistically attainable solutions.”

Software development still has a ways to go before reaching this level of engineering knowledge. However, we shouldn’t lose sight of where we need to go: we must identify possible causative factors so that we can explore and discover the effectiveness of technologies in a quantitative, reproducible way. As Shari Pfleeger explained (see Computer, Oct. 1999), “if we look long enough and hard enough, we will find rational rules that show us the best ways to build the best software.” However, we shouldn’t expect to find a one-size-fits-all approach to building good software; indeed, we’ll likely discover that certain approaches work best in certain situations.

Moving forward

Software developer community stakeholders can contribute to developing our discipline’s engineering knowledge in several ways.

Researchers must test the strengths of the new artifacts they develop before putting them on the market, providing data that demonstrate how they improve product quality and identifying the best application conditions. Researchers must also replicate their colleagues’ studies, either to corroborate the results under the same application conditions or to provide additional information in new contexts. This will let researchers provide practitioners with tested knowledge on the benefits of applying given artifacts and their application conditions. Disciplines such as chemistry, medicine, and psychology apply experimental design and analysis techniques to conduct such empirical studies.


In turn, practitioners must demand and understand the results of the empirical tests the researchers supply for new software artifacts. They can refer to the previously mentioned books for this purpose. Practitioners must also measure and monitor improvements and changes that occur during development and in their software products when applying such software development artifacts or knowledge. Commonly known as process control, this activity is widely used by several production-related disciplines. Developers who intend to use process control in software development must address questions such as

- What type of data or metrics does my organization need to collect (for example, productivity, cost, and reliability measures)?
- How should we collect these data to assure that they are reliable and comparable? What granularity should they have, and who will supply them?
- How will they be stored and interpreted? What analysis techniques or tools are available?

We refer practitioners interested in addressing these questions to William Florac and Anita Carlton’s book Measuring the Software Process: Statistical Process Control for Software Process Improvement (Addison-Wesley, 1999), which addresses global process measurement in detail, for example. We also refer practitioners to real examples of process control in software development: In “Benchmarking Software Development Productivity” (IEEE Software, Jan./Feb. 2000), the authors apply process control to study the productivity of different software development organizations. Additionally, “Our Worst Current Development Practices” (IEEE Software, Mar. 1996) describes an application of this process to improve software project estimation and planning.

Software engineering has the tools to transition from working with conjecture to working with fact. Widely using these tools will bring software engineering an important step closer toward maturity. As other engineering disciplines have demonstrated, this step will help make quality software construction a predictable process.

Natalia Juristo is a full professor in the School of Computer Science at the Universidad Politécnica de Madrid. Contact her at natalia@fi.upm.es.

Ana M. Moreno is an associate professor in the School of Computer Science at the Universidad Politécnica de Madrid. Contact her at ammoreno@fi.upm.es.