Editor in Chief: Hakan Erdogmus ■ National Research Council Canada
hakan.erdogmus@computer.org

Must Software Research Stand Divided?

Hakan Erdogmus

A few years ago at lunch, I was sitting next to a prominent software engineering researcher. The topic was the antagonism between empirical software engineering and the rest of software engineering research. My colleague was explaining how physicists seem to have figured it all out. Experimental physics and theoretical physics have a symbiotic relationship. Theoretical physicists try to understand and predict natural phenomena by constructing abstract models. Experimental physicists observe natural phenomena and collect data in the laboratory, either to validate existing models or inspire new ones. Thus, beyond supplying the models that explain experimental physics' data and observations, theoretical physics also conceptualizes new models to be validated, in turn feeding back to experimental physics. Theoretical and experimental physicists generally seem to value each other's work.

Sounds like a beneficial, collegial division of labor, right? In the real world, things might not always be that smooth: the enduring human traits of self-righteousness and self-preservation tend to intervene and perturb peace. When I asked my physicist friend, "Is it really a love-in between experimental and theoretical physics?" he replied, "Yes and no, leaning more on the yes side."

Two sides of software research

Enough of physics. Let's go back to the topic: software engineering research. (From here on, for brevity's sake, I'll omit "engineering.") By "research" I do not imply only academic research. Rather, I refer to research in the broader sense—the pursuit to create the new and the better, and to understand the old, the current, and the unknown. So, I categorically include software practitioners' efforts to advance the state of the art. Although most wouldn't call themselves researchers, the grassroots contributions in this context are not only undeniable but also primary. Regardless of whether they're carried out in laboratory-like settings, within organizational boundaries, or across broad communities of interest, research efforts by both practitioners and academics entail technological and process innovations as well as experiential contributions to the existing body of knowledge.

Empirical software research operates in the latter realm: that of the experiential, the world of trial, observation, experience, and evidence. Empiricists try out proposed solutions to gauge how they work to solve a problem. They observe others practicing software development or study properties of the resulting artifacts. They accumulate experience and evidence related to a concept, approach, or technique. They collect and organize findings of their investigations, analyze them to validate theories, make inferences, and generate insights.

Software research isn't dominated by empirical work. So what other type of software research is there? I don't want to call the counterpart theoretical software research, for that has a connotation for being contrasted with "practical," and this, I wish to avoid. So borrowing loosely from learning theory, I'll call it *constructionist software research*. Constructionist learning advocates learning by actively building or

Mission Statement: To be the best source of reliable, useful, peer-reviewed information for leading software practitioners—the developers and managers who want to keep up with rapid technology change.

 0740-7459/08/$25.00 © 2008 IEEE

manipulating the objects of learning. Then, constructionist software research entails creating artifacts and concepts to address the challenges of software development. Constructionists propose tools, techniques, theories, methods, and processes. The experience gained through engineering solutions and related artifacts forms the basis of the constructionist's understanding of the world.

Flames, excuses, and myths

Most software research explicitly favors the constructionist side of the continuum, although empirical research has been gaining momentum as its practical value has been increasingly acknowledged. Alas, hostile divisions do exist. The die-hard constructionists' arsenal against empirical research is manifold. Let's question common objections to empiricism.

Empirical research is boring.

Not to the empiricist. Empirical research requires a different set of skills. People who possess those skills are happy to apply them to do what they do best. Empirical work is necessary, no matter how many excuses we make to avoid it. But not everybody has to do it. If you're a software researcher and find empirical work boring, you should be happy that someone else is willing to do it.

Empirical research takes too long.

Generating a deep understanding about a technique or technology might take years. Practitioners often can't wait that long in a competitive environment: they must take calculated risks, trying out new approaches on their own to make up their minds, ultimately exercising a form of small-scale empiricism. Giving up on knowledge because it takes too long to acquire is short sighted. And what's the alternative? Just keep churning out new techniques and technologies without ever attempting to objectively study them?

Empirical research is too soft. It's more like social or management science, and it doesn't produce any technology.

Some empirical research, by the nature of the questions it addresses, deliberately employs methods borrowed from the social sciences. Others employ hard, quantitative methods. Even though empirical research often doesn't produce technology directly, software innovation is by no means the exclusive domain of constructionists. Insights gained through empirical research do drive innovation: understanding a tool's deficiencies leads to better tools; understanding the factors that make a software team tick leads to processes better tailored for specific needs.

We don't need empirical evidence to know whether something works. If it's widely used, it works. If nobody uses it, it doesn't work.

True, a technique's state of adoption often correlates with its inherent effectiveness. But how would you assess the extent of the actual practice? Well, such assessment is also the domain of empirical research. Our industry is as fluid as an industry gets. On one hand, paradigm shifts and technological innovation can affect scale and scope economies that make a previously infeasible approach feasible. On the other, some approaches persist in practice for obscure reasons long after they outlive their usefulness. In these instances, as well, empirical research is vital.

It's impossible to prove a technique effective when there are so many factors, so why bother?

The goal of empirical research is neither proof nor hard quantitative evidence. Rather, the goal is a sufficient, persuasive level of understanding that enables educated decisions in particular contexts. Moreover, a primary goal of empirical research is to explore and expose such factors.

Software practice and technology are changing too fast. There's no point.

It's impossible to study and evaluate every single fad or proposed technology. And not all deserve or require rigorous study. A new framework that lets us automatically generate code stubs from existing requirements artifacts probably doesn't warrant deep investigation. Conversely, a new testing approach that promises superior quality but incurs a large overhead probably does. Resources can be focused on sweeping, promising, or controversial approaches with significant payoffs or uncertain break-even points.

If it worked on a few students with a few toy examples, that doesn't mean it will work for me. Why should I care?

IEEE Software

Empirical research entails a variety of different kinds of studies, from classroom experiments with students to practitioner-conducted case studies involving professionals working on a real project. It also includes practitioner narratives of experiences and lessons learned. Each kind of study serves a different purpose, addressing a different level and strength of evidence. Some are exploratory, some are explanatory, while others are confirmatory.

Empirical research is dangerous. People misunderstand, misrepresent, or overinterpret the findings.

This is a phenomenon crossing disciplinary boundaries. It has more to do with education than the nature of empirical research itself—for example, lay people and medical practitioners alike often misunderstand clinical-research findings. Regular expert interpretations of the literature alleviate confusion. Follow Forrest Shull's Voice of Evidence column to avoid this problem.

Empirical researchers are biased. Why should they be trusted?

Researchers can be biased regardless of their orientation. Research that appears in peer-reviewed publications is less likely to be biased and more likely to declare any biases. Reviewers are notoriously good in detecting and exposing bias.

Empirical research uses evidence models from other disciplines. These models don't apply to software development.

The most commonly advocated models are adopted from medicine and law (see the section on evidence-based software engineering in the Jan./Feb. 2005 issue of *IEEE Software*). These models might not be appropriate for all software research, but until custom models are developed, they still advance our thinking, reveal underlying issues, and provide a first approximation.

The other side of the story

Some hard-core empiricists aren't immune to self-importance, either. Their defenses are somewhat less imaginative, mostly centered on stamping out ideas as unworthy just because they don't have a rigorous empirical basis. A typical argument from their arsenal is this: "The idea looks novel, but there's no proof that it works." The truth is, in all likelihood, research will never "prove" the merit of some of the best, most influential, and most practical software innovations. Empiricism is only one way to demonstrate (some forms of) merit, often after the fact. In many instances, it's not particularly suitable for assessing potentially ground-breaking work with latent implications. Where empiricism falls short, theoretical argumentation and validation, and ultimately common sense, should at least temporarily suffice.

The way forward

Isolationist divisions among overlapping fields have always existed. Dismissive sentiments between software engineering and computer science, theoretical and applied computer science, computer science and information systems, and computer and software engineering are common (see, for example, Robert Glass's column "Never the CS and IS Twain Shall Meet?" in the Sept./Oct. 2005 issue). In the case of software research, however, adversarial attitudes are not only unwarranted but also seriously mutually destructive.

Silos tend to be self-serving. Empirical research often produces open questions that feed further empirical research. Constructionist research often triggers a plethora of improvements, adaptations, or copycats, feeding further constructionist research. To accelerate the advance of the practice and of our understanding of it, we need more collaboration that generates the genuinely productive kind of feedback that crisscrosses the two orientations.

It's not that such collaboration hasn't been happening. Plenty of examples exist: we can find many in the pages of *IEEE Software*. We just need more of it. Much more.

All right, the separation I've been talking about might not be as harsh as I made it seem here. Many software researchers play in both worlds, alternating between creating and studying. However, most of them naturally gravitate toward one side of the spectrum. Only the rare software researcher possesses the mentality and complete set of skills to excel in both types of research. Software research needs both orientations, and researchers of both orientations need each other.

So, really, what's the problem? Write to me at hakan.erdogmus@computer.org.