How Important Is Evidence, Really?

Hakan Erdogmus

I can’t promise this will be my last commentary on the importance of evidence in software engineering—to be more precise, I mean the role that evidence plays in the adoption of software engineering ideas. The topic just creeps up on me time and time again. It can’t be helped.

My essay “Essentials of Software Process” (IEEE Software, July/August 2008, pp. 5–7) made a case for empiricism. I wrote,

Empiricism at its heart is supporting decisions through evidence based on data, both observations and measurements. By observations I mean occurrences that we can simply record. By measurements I mean things we can count, calculate, or quantify. Measurements have values, whereas observations have descriptions, possibly including contextual information. Observations provide deeper insight in areas in which measurements serve only as proxies for other constructs. Each kind of data has a place, and empiricism entails collection and use of both kinds.

A later column, “Must Software Research Stand Divided?” (September/October 2008, pp. 4–6), was an attempt to bust the myths surrounding empirical software engineering and moderate some of the strong claims made by die-hard empiricists. To recall those myths:

- empirical research is boring, too soft, conducted in artificial settings, and dangerously interpreted;
- empirical research takes too long;
- empirical evidence isn’t needed;
- empirical evidence can’t possibly address all the contextual factors or keep up with a fast-changing industry;
- empirical researchers are biased; and
- empirical research uses evidence models from other disciplines that have nothing to do with software development.

Now I feel obliged to expand upon my latest, and rather anticlimactic, mention of the subject in “Regress or Progress? Seeing Good Software Engineering Ideas Through” (March/April 2010, pp. 4–7). In that column, I stated that the maturation, acceptance, and adoption of good software engineering ideas depend on many factors. I counted the availability of evidence among those factors, further qualifying that the value of evidence itself depends on a variety of underlying subfactors. And I left it there. How convenient that the suspense gives me the opportunity to complete the circle.

Let me rewind again momentarily. In “Must Software Research Stand Divided?” I also implied that empiricists sometimes overemphasize evidence. Let’s pick up that thread and weave it together with the factors affecting the usefulness of evidence.

Types of Evidence

The type of evidence available depends on an idea’s maturity and the extent to which the idea lends it-
FROM THE EDITOR

Software Boards

With the exception of the chair, our advisory board members serve a maximum of two consecutive terms, or a total of four years. Their function, ranging from routine strategic planning to infusion of new ideas to execution of special projects, is essential to the operation and success of the magazine. Outgoing members Gargi Keeni, David J. Blaine, and Simon Helsen have each contributed to Software in important capacities: as advisors, reviewers, and guest editors. Gargi has additionally been instrumental in increasing the magazine’s prominence as our South Asian ambassador. I thank them for their many years of dedicated service.

New members Kevin Henney, Jan Bosch, and Taku Fujii have joined Software’s advisory board recently to fill the void left by Gargi, David, and Simon. Kevin Henney is an acclaimed author, consultant, and trainer based in the United Kingdom. His work focuses on programming practices, languages, patterns, software architecture, and the development process. He coauthored two volumes in Wiley’s popular Pattern-Oriented Software Architecture series and is the editor of 97 Things Every Programmer Should Know (O’Reilly Media, 2010).

Taku Fujii is a manager in the Software Engineering Center at the Osaka Gas Information System Research Institute. Over the last two decades, he has served as a project manager, architect, designer, programmer, and technical consultant in the Japanese software industry. With diverse interests in software processes, modeling, and service orientation, he continues to serve the Japanese software community in crucial roles.

Jan Bosch leads a double life as vice president of engineering processes at Intuit and professor of software engineering at the University of Groningen, The Netherlands. His past credentials include leading the Software and Application Technologies Laboratory at Nokia Research Center. A recognized expert in software architecture and variability management, Jan is the author of Design and Use of Software Architectures: Adapting and Evolving a Product Line Approach (Addison-Wesley Professional, 2000). I welcome the incoming members to IEEE Software’s advisory board.

self to that type of evidence. In turn, the type of evidence dictates the strength of evidence.

Feasibility Check

The weakest form of evidence is the feasibility check, which can be established early in the idea’s maturation life cycle, even at its conception. The purpose of the feasible check is to quickly assess major arrounds start to tell a coherent story. This type of evidence isn’t available until the idea is well into its testing state. Anecdotes are helpful, but they might not be powerful enough to push a decision-maker over the adoption barrier. Anecdotes tend to be susceptible to positive reporting bias and the infamous halo effect (the tendency of a few, often positive, attributions overshadowing other, often negative, attributions).

Systematic Evidence

Beyond anecdotes is the most powerful but elusive type of evidence: the systematic kind. Researchers gather systematic evidence in a deliberate and methodical
Launching a New Kind of Article Series, Edited by Linda Rising

Software’s ongoing efforts to beef up the relevance, timeliness, and practicality of its content are redoubling once again. Our latest exciting initiative is a new article series entitled Insights and edited by Linda Rising, our newest associate editor in chief. Employing a shorter format and a separate, more intimate review process than that of our regular peer-reviewed content, Insights will feature experience-based nuggets from thought leaders and practitioners who have braved the software development trend wars, visited its frontiers, guarded its trenches, and emerged with new wisdom. Linda calls Insights an “experiment” in progress. I believe it will prove to be something much more. But don’t let me spoil the party. Read Linda’s introduction on page 6 to find out how the new series will work and what you can do to contribute. Be sure to check future issues for new Insights.

FROM THE EDITOR

attempt to isolate and reveal common, situational effects on the basis of credible, rich, consolidated data. Alas, this type of evidence might not be available until well into an idea’s streamed state. As such, systematic evidence in software engineering is rare. When it exists, it tends to materialize after the fact, too late for risk-loving early adopters and of limited use to risk-averse late adopters. This latter point brings us to the remaining factors that are the attributes not of the evidence but of the idea’s receptors and the idea itself.

Risk Preferences and Tolerance of Receptors

Adoption decisions are largely affected by the attitudes of the organizations and people who make the decisions. Those attitudes are often dictated by the business environment in which organizations, and decision makers acting on behalf of organizations, find themselves. A fast-changing environment with narrow windows of opportunity and low entry barriers might warrant a risk-taking, or risk-loving, attitude. A stable and rigid environment in which organizations, find themselves. A fast-changing environment with narrow windows of opportunity and low entry barriers might warrant a risk-taking, or risk-loving, attitude. A stable and rigid environment in which organizations, find themselves. A fast-changing environment with narrow windows of opportunity and low entry barriers might warrant a risk-taking, or risk-loving, attitude. A stable and rigid environment with large, messy bundles are difficult to obtain and becomes available substantial and largely irreversible. Therefore they warrant the strongest form of evidence, which unfortunately is also the most difficult to obtain and becomes available late in the game.

Size Matters

Size, the number of concepts and dependencies among those concepts that together make an idea whole, affects the utility of the evidence regarding the idea’s effectiveness. Small ideas, such as pair programming, in small bundles require the least and the weakest form of evidence. Some small ideas are viral in that they’re instantly and obviously recognized as valuable. They can be wrapped in tidy bundles and easily sandboxed with a low cost of learning and application. They pose little adoption risk, and the decision to adopt tends to be reversible.

Medium-sized ideas, such as in-process unit testing, with larger bundles are composed of multiple concepts with interdependencies. They are more difficult to sandbox and incur modest learning and application costs. Part of the adoption costs might be irreversible. Such ideas pose an adoption risk commensurate with the irreversible portion of the underlying adoption costs.

Large ideas, such as model-driven development, with large, messy bundles are most risky. Adoption costs might be substantial and largely irreversible. Therefore they warrant the strongest form of evidence, which unfortunately is also the most difficult to obtain and becomes available late in the game.
Adoption Context

The adoption context is a strategic factor that can be partially controlled depending on other factors, which are intrinsic. Adoption context has two dimensions: scale and rate. The scale of adoption is the number of instances in which a new idea will be applied in a specific organizational situation. It answers the question: how widespread will the idea’s adoption be?

The rate of adoption is the speed at which the idea will be spread and applied in that situation. Gradual adoption might hedge high adoption risks if the underlying evidence is weak, the recurring application costs are high, and the components of the idea’s bundle can be incrementally applied. This would allow the existing solution to remain in effect while more experience is being gained with the replacement solution and until any major uncertainties have been resolved. Gradual adoption provides a chance to preempt an idea’s spread if problems materialize early. Rapid adoption might make more sense if evidence is strong, one-time learning costs dominate, and recurrent application costs are relatively low.

Let’s put this all together. Table 1 summarizes the roles of different types of evidence under various situations. The risk-taking attitudes of the decision makers shift the rows left (as risky behavior is increasingly prevalent) or right (as risky behavior is increasingly avoided).

In software engineering, evidence tends to be over- or underemphasized, emphasized too early and too indiscriminately or too late and too sparingly. The value of evidence must be gauged carefully in each situation.

For other and complementary points of view, check the January/February 2005 issue of IEEE Software. In “Evidence-Based Software Engineering for Practitioners,” Tore Dybå, Barbara A. Kitchenham, and Magne Jørgensen make a case for systematic evidence and how practitioners can leverage it for higher-quality adoption decisions. In “Soup or Art? The Role of Evidential Force in Empirical Software Engineering,” Shari Lawrence Pfleeger lashes out at common wisdom and vendor hype when they substitute for solid evidence, and gives examples of usage from other disciplines regarding different types of evidence. Finally, for the latest on evidence in software engineering, don’t forget to check Greg Wilson’s forthcoming book What Really Works (O’Reilly, 2010).