Progress, Obstacles, and Opportunities in Software Engineering Economics

Pressing demands from industry suggest more attention should be focused on this critical area.

Chris F. Kemerer

Nearly a decade of Workshops on Information Systems and Economics (WISE) have examined a wide range of research topics. These include a number of what could be termed "demand-side" topics, which focus on the effective use of information technology, such as IT-induced changes to markets, and "supply-side" topics, which focus on the efficient provision of IT resources. *Software engineering economics* (SEE) spans both groups, but it has generally been the most dominant topic within the supply area. While the economics of hardware have been relatively ignored in the management research literature, given industry's ability to continually provide faster, smaller hardware at ever-decreasing prices, software has been an entirely different matter. Software costs continue to be significant, and industry's understanding of how to reduce them has improved only very slowly in contrast to hardware. This dilemma has merited a significant amount of research attention, and the consequent research findings tend to have important practical applications.

Software engineering focuses on the production of software, which is by its very nature a relatively intangible good. Objectifying and measuring its many dimensions are often challenging to researchers.

Also in contrast to some of the demand-related topics, SEE research commonly has a significant technological component to it, since much of the research is focused on evaluating promising new technologies or approaches to software development. This diverges with much demand-side work, which tends to be closer to traditional economics research. Therefore it requires some skills and knowledge of both the software engineering and economics disciplines. Accordingly, it has also attracted some researchers from the software engineering area within computer science, whose work will also be reflected in this review, and in the suggestions for future work. This combination of skill requirements also tends to make research in this area comparatively difficult, and suggests one reason why we don't see more research in this important area.

Practitioners are most concerned about understanding what aspects of software engineering innovations have worked best and whether they are applicable to their particular situation. This motivation also lies at the heart of most SEE research. In addition, with the huge growth in the packaged software industry, more attention has been focused on software as an economic good.

Obstacles to Progress

A number of factors militate against observing progress in software engineering in general, and software engineering economics in particular. Software engineering focuses on the production of software, which is by its very nature a relatively intangible good. Objectifying and measuring its many dimensions are often challenging to researchers. Even in situations where we have relatively good software-related artifacts to examine, there may be little evidence to examine concerning the process by which they were constructed.

Even if there were good access to project data, there is another significant confounding factor. The number and complexity of applications for software have grown very rapidly, fueled, in large measure, by the dramatic improvements in hardware technology. As hardware decreases in size and increases in performance per dollar, more applications can be justified. This then creates demand for software. While this tremendous demand for software has fueled the fortunes of many software developers and their organizations, it provides significant challenges to researchers as we attempt to assess progress in the delivery of software. Since the applications have changed as well as the tools, it becomes very difficult to assess the impacts of these new tools. It may well be the case that the effect of a new tool is to create a difference in kind (systems are completed that would not otherwise be attempted) rather than a difference in the degree of efficiency with which certain classes of systems are created.

Much software engineering research has been focused specifically on assisting teams of software developers to plan, design, implement, and control software projects. The success of software engineering projects needs to be evaluated in terms of the delivery of a set of functionality assessed on a number of dimensions, for instance, the application's cost, reliability, ease of use, maintainability, etc. These dimensions have proven notoriously difficult to measure, even when functionality is relatively constant. Instead, it is widely believed that applications have grown in terms of their complexity, and that what would otherwise be seen as gains are essentially obscured by the greater complexity. For example, it might be believed that a measure such as the "percentage of failed projects" should decline as progress is made in developing better software processes. However, this number could easily remain constant, or even increase if such processes were simply used to attempt more ambitious projects.

Applying SEE to Practice. Although concrete evidence of progress is difficult to document, given the previously referenced difficulties, there are strong beliefs in the field about some key progress areas over the last decade. These are areas in which research subjects have evolved to become more widely accepted practical approaches, and topics that formerly merited notice as best practice have now become merely the professional standard.

Overall, perhaps, the most significant area of progress is in the notion of *software process*. While in a sense much or all of software engineering research can be seen as illuminating the subject of process, there has been a tremendous awakening on the part of practice to this research topic. Practitioners have made significant changes in how they view software development. Early focus on coding, and then later on whole projects, has broadened toward thinking about portfolios of projects and the processes by which these are developed and maintained. For example, even the market leader, Microsoft, realized that its strategy of employing small teams of star developers did not scale up when confronted with the market realities of developing, marketing, and maintaining integrated suites of mass-market applications [5].

Much of the credit for this evolution must go to the Software Engineering Institute's Capability Maturity Model, which, despite some controversy in its details and in the policy decisions surrounding its implementation, has effected a change in the language by which people describe software development. By providing a shared framework, much more effective communication has taken place regarding the appropriate role for process in software development. In addition, empirical evidence is emerging surrounding the economic value added when advancing to a higher level of process maturity [8].

A second area is the greater acceptance and use of *measurement* in software development practice [11]. Partly in response to market pressures to evaluate tools and to benchmark against competitive alternatives, and, more recently, as the key component of process improvement efforts, there has been a noticeable increase in the measurement of software products and processes. This greater use of measurement can also be attributed, at least in some part, to greater awareness of, and confidence in, software metrics stemming from software engineering economics research. Some specific WISE-related metrics work includes the development of object points [2] and extensions of function points [9].

Beyond these two general categories of process and measurement, progress has been made in substantiating the value of a number of specific approaches to improving software engineering. For example, the economics of software reuse have been well elaborated and validated over the last decade [10]. The development and use of models for cost estimation, perhaps the single greatest focus of the earliest work in software engineering economics, have progressed greatly and now are standard tools in well-managed software development environments [3, 12]. Risk management is another area in which a significant amount of progress has been made [6].

Another area that has received a considerable amount of attention from the WISE community is *software evolution and maintenance*. Long recognized as the major element of cost in any true software development life cycle, the economics of software maintenance have been addressed both in theory and in practice [1, 7]. SEE research has focused on estimating the costs and benefits of investments in reducing controllable complexity and in promoting initial software quality.

While the vast majority of the work in software engineering economics focuses on the supply side, a new area has taken increased notice of the significant packaged software market and has begun to look more carefully at software as an economic good [4]. This work brings to bear traditional econometric tools to the questions of the value of individual software features and of adherence to standards; it also examines trends in software demand and the quality-adjusted price of software.

In summary, while its effects are difficult to measure concretely, research on software engineering economics has greatly expanded over the past decade and its likely consequences are manifest in multiple areas of software development practice.

Challenges and Opportunities

Work in this area will continue to face the same set of challenges that have confronted it in the past. The rapid rate of technological change is at variance with the pace of most research efforts. Practitioners and researchers alike are often concerned with this disparity. However, some of this concern may be misplaced. While there is a significant amount of real change, there is also a tremendous amount of surface change that should not be allowed to affect the progress of research in this area.

Motivated in part by practitioners' desires for solutions, many of the efforts in software engineering tool creation come equipped with new labels for what may be relatively minor differences. This nomenclature tends to cloud the discussion about the degree to which results obtained in one area are relevant to another. For example, while there has been a tremendous amount of apparent change in the development of programs, there is a core set of fundamental ideas, for example, information hiding, that has remained important.

Within these fundamental ideas lie some concepts that have had considerable longevity, for example, within modularity the notions of coupling and cohesion. It is important for both researchers and practitioners to keep this in mind when examining new software engineering technologies. Researchers should be clear about the fundamental ideas within any new technology—technology innovators need to be explicit about why the new technology is believed to work and evaluation research should be pegged to these fundamental ideas, rather than the surface nomenclature. In this way the value of any research study should be much easier to communicate to practitioners who could make use of it. This focus should prevent a certain amount of distraction for researchers who should focus on problems with long-term and significant payoffs.

This idea also applies to software engineering economics research aside from technology evaluation. For example, a current practitioner problem may be stated as the shortage of Java programmers. Clearly, this is a problem that will be solved by market forces. What we ought to focus on is the fundamental and recurring problem of organizations' long-term strategies for training and retaining staff members in emerging technologies. How can organizations that wish to adopt new technologies best structure themselves to provide an environment for individuals with scarce skills that will always be in high demand? What lessons can be drawn from the economics contracting literature that would aid these organizations?

Another continuing challenge in the area of software engineering economics is its fundamentally interdisciplinary nature. Because of the significant role played by people, software engineering is already one of the computer science disciplines that is closest to the social sciences. By focusing on the economic aspects of software development the research moves even more in that direction. Work done in business schools using economics as a discipline needs to have a strong basis in the underlying technology to assure that pertinent questions are being asked and that the models used are appropriate. Therefore, this work tends to be closer to engineering than other research conducted in most business schools.

Despite this confluence of interests, sizable gaps remain in practice. For example, computer scientists too frequently fail to cite relevant work done by business school researchers that is published in business school journals. Similarly, despite current chronic faculty shortages, business schools too rarely hire computer scientists for information systems positions in business schools. These two academic communities could be doing much more to work together on research in this area. Joint work would marry traditional computer science strengths (facility with the technology and understanding of the fundamental concepts at work) with business strengths (measurement, modeling, and access to industrial research partners). There are synergies that could be brought to bear on many, if not all, software engineering economics problems. C

References

1. Banker, R., Davis, G. and Slaughter, S. Software development practices, software complexity, and software maintenance effort: A field study. *Manage. Sci.* (forthcoming)
2. Banker, R.D., Kauffman, R.J. and Kumar, R. An empirical test of object-based output measurement metrics in a computer-aided software engineering (CASE) environment. *J. Manage. Info. Syst. 8*, 3 (1991), 127–150.
3. Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., and Selby, R. Cost model for future software life cycle processes: COCOMO 2.0. *Annals of Software Eng. 1*, 1 (1995), 57–94.
4. Brynjolfsson, E. and Kemerer, C.F. Network externalities in microcomputer software: An econometric analysis of the spreadsheet market. *Manage. Sci. 42*, 12 (1996), 1627–1647.
5. Cusumano, M. and Selby, R. *Microsoft Secrets.* The Free Press, New York, 1995.
6. Dorofee, A.J., Walker, J.A., Alberts, C.J., et al. *Continuous Risk Management Guidebook.* Carnegie Mellon University Software Engineering Institute, Pittsburgh, PA, 1996.
7. Gode, D.K., Barua, A. and Mukhopadhyay, T. On the economics of the software replacement problem. In *Proceedings of the 11th International Conference on Information Systems*, (Copenhagen, Denmark, 1990), pp. 159–170.
8. Humphrey, W.S., Snyder, T.R. and Willis, R.R. Software process improvement at Hughes aircraft. *IEEE Software 8*, 4 (1991), 11–23.
9. Kemerer, C.F. Reliability of function points measurement: A field experiment. *Commun. ACM 36,* 2 (Feb. 1993), 85–97.
10. Lim, W.C. Effects of reuse on quality, productivity, and economics. In Kemerer, C.F., Ed., *Software Project Management: Readings and Cases*, McGraw-Hill (R.D. Irwin), Boston, Mass., 1997.
11. Pfleeger, S.L. Lessons learned in building a corporate metrics program. *IEEE Software 10*, 3 (1993) 67–74.
12. Shepperd, M., Schofield, C., and Kitchenham, B. Effort estimation using analogy. In *Proceedings of the 18th International Conference on Software Engineering* (Berlin, Germany, 1996).

Chris F. Kemerer (ckemerer@katz.business.pitt.edu) is the David M. Roderick Chaired Professor of Information Systems at the University of Pittsburgh.