White Paper for

Workshop on New Visions for Software Design and Productivity

Title: Value Based Software Engineering

Barry Boehm and Dan Port, University of Southern California
Kevin Sullivan, University of Virginia

Abstract

The primary thesis of Value-Based Software Engineering (VBSE) is that the integration of a software system's stakeholder value propositions into the system's definition, design, development, deployment, and evolution is critical to the system's success. This white paper:

- Analyzes the sources of software project failure in the Standish Report, and shows that many of the failed projects were caught in the vise of value-insensitive software engineering.
- Discusses promising research ideas for improving our capability to perform VBSE.
- Presents a roadmap for making progress toward VBSE and its resulting benefits.

The white paper concludes with a summary of the relations between VBSE and other software research and applications areas. It is unavoidably involved with software and information system product and process technology, and their interaction with human values. It is strongly empirical, but includes new concepts in need of stronger theory. It uses risk considerations to balance software discipline and flexibility, and to answer other key "how much is enough?" questions. And it helps illuminate information technology policy decisions by identifying the quantitative and qualitative sources of cost and value associated with candidate decisions.

Failed Software Projects: Sources and Remedies

The 1995 CHAOS Report [Standish, 1995] surveyed several hundred software projects and found that only 16% of them were completed within their planned budget and schedule. The report analyzed the major sources of failure for the other projects, and found that eight problem sources accounted for 80% of the failures. Each of these sources is discussed below in terms of their relation to value-based approaches to software engineering.

1. Incomplete Requirements (13.1% of project failures)

   A major source of project overruns due to incomplete requirements is the neglect of critical off-nominal requirements. Empirical studies have shown that 80% of the software rework comes from 20% of the problems, and that many of these critical problems involve neglect of off-nominal requirements [Boehm-Basili, 2001]. An effective value-based technique for addressing these problems is risk management, particularly the assessment of the relative risk exposure (probability of loss times size of loss) of the various off-nominal operational scenarios [Boehm, 1989]. Other effective techniques involve goal-oriented approaches to software requirements [Dardenne et al., 1991; Young, 2001], in which goal elaboration provides a value oriented approach to identifying critical off-nominal or missing requirements. Further research in each of these areas would strengthen projects' ability to avoid such overruns.

2. Lack of User Involvement (12.4%)

   We would generalize this to "lack of critical-stakeholder involvement," as many system failures also result from lack of higher management, customer, maintainer, and interoperator, as well as user involvement. Here again, several approaches involving stakeholder value-based considerations have been developed and successfully applied, such as Participatory Design and Joint Application Development [Carmel et al., 1993], Quality Function Deployment [Eureka-Ryan, 1998], and stakeholder win-win [Boehm-Ross, 1989]. Further research has considerable potential in areas such as client-developer mutual requirements learning [Majchrzak-Beath, 2001] and multi-attribute tradeoff analysis [Keeney-Raiffa, 1993].

3. Lack of Resources (10.6%)

   Software cost and schedule estimation models have saved many projects from overruns. But the pace of software engineering reinvention (COTS, open source, rapid development, agile development, iterative software process models) requires the software estimation field to be continually reinventing itself as well [Boehm et al., 2000, Chapter 6]. Other promising areas involve innovative critical-resource driven approaches such as the theory of constraints [Friedman-Leondes, 1969; Goldratt, 1990], timeboxing [McConnell, 1996] and the schedule/cost/quality as an independent variable process [Boehm-Brown, 2001].

4. Unrealistic Expectations (9.9%)

   This can be considered the as the opposite side of the coin from "Lack of Resources". Some additional value-based approaches for creating realistic expectations are business case analysis [Reifer, 2002], the DMR Group's Benefits Realization Approach and its Results Chain method linking initiatives to contributions, assumptions, and outcomes [Thorp, 1998], and techniques for expectations management [Karten, 1994].

5. Lack of Executive Support (9.3%)

   As discussed in point 2 above, lack of executive stakeholder support is often highly correlated with lack of executive involvement in formulating an initiative. Techniques such as Results Chain and business case analysis can also help here.

6. Changing Requirements (8.7%)

A highly promising value-based approach here is Sullivan's Strategic Design initiative, which uses techniques from real options theory [Amram-Kalutilaka, 1998] and the economic theory of modular design [Baldwin-Clark, 1999] to analyze the economic value of techniques for modular software design such as information hiding [Parnas, 1972] in facilitating adaptation to changing requirements [Sullivan et al., 2001].

7. Lack of Planning (7.5%)
   This has a straightforward good-practices component, but value based planning approaches such as DMR's Benefits Realization Approach avoid additional failures by identifying complementary initiatives (e.g., data collection, conversion, training, logistics) that must also be pursued to make the software application a success.

8. Absence of Need (7.5%)
   Here again, business case analysis, improved developer-client mutual learning, and stakeholder requirements prioritization capabilities can help avoid software "gold plating" phenomena.

Software Economics Roadmap

Value-based software engineering is a major component of the overall field of software economics. Our roadmap [Boehm-Sullivan, 2000] for the next major phase of research in software economics begins with the goal of developing fundamental knowledge that will enable the end objective of significant, measurable increase in the value created over time by software and information technology projects, products, portfolios and the industry.

Working backwards from the end objective, we identify a network of important intermediate outcomes. The roadmap in Figure 1 illustrates these intermediate outcomes, dependence relationships among them, and important feedback paths by which models and analysis methods will be improved over time. The lower left part of the diagram captures tactical concerns, such as improving cost estimation for software projects, while the upper part captures strategic concerns, such as reasoning about real options and synergies between project and program elements of larger portfolios.

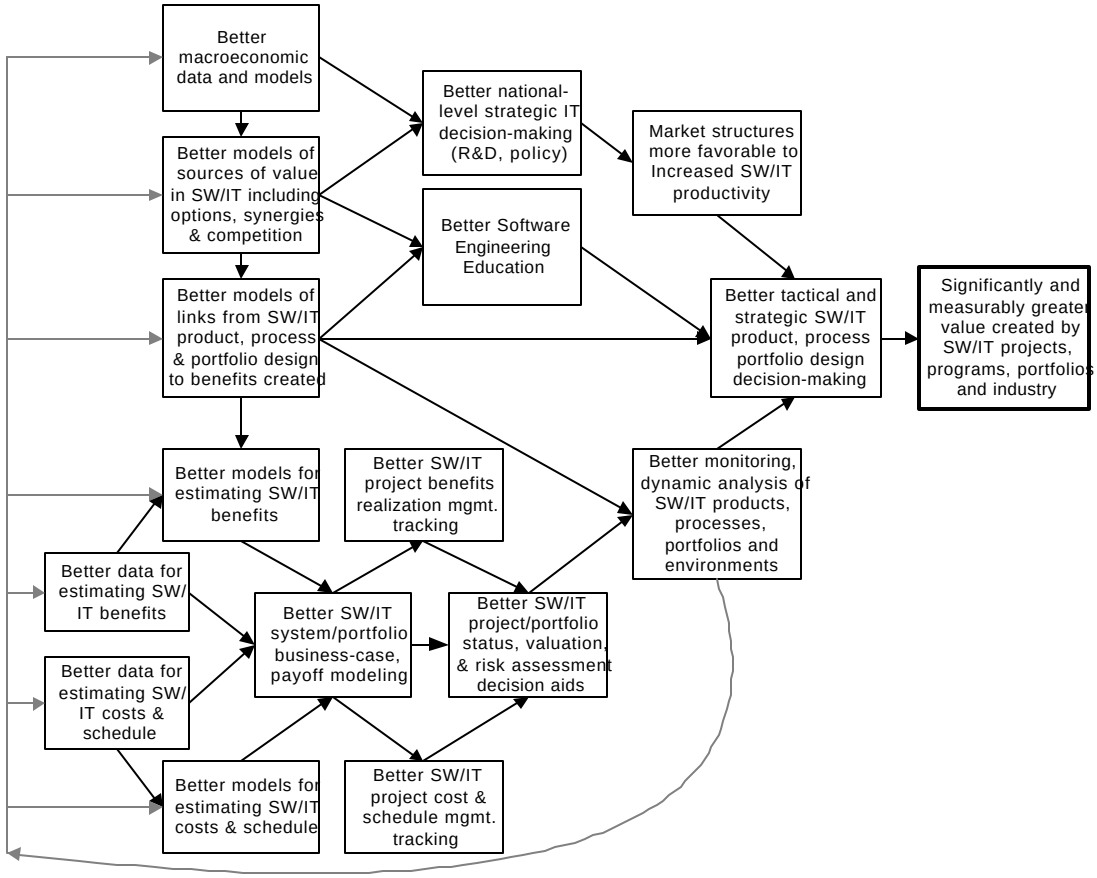

Figure 1: Roadmap for research in software engineering economics.

Making Decisions that are Better for Value Creation

The goal of our roadmap is supported by a key intermediate outcome: designers at all levels must make design decisions that are better for value added than those they make today. Design decisions are of the essence in product and process design, the structure and dynamic management of larger programs, the distribution of programs in a portfolio of strategic initiatives, and to national software policy. Better decision-making is the key enabler of greater value added.

Design decision-making depends in turn on a set of other advances. First, the design space within which designers operate needs to be sufficiently rich. To some extent, the design space is determined by the technology market structure: what firms exist and what they produce. That structure is influenced, in turn, by a number of factors, including but not limited to national-level strategic decision-making, e.g., on long-term materials that are produced that designers can then employ, and their properties.

Second, as a field we need to understand better the links between technical design mechanism (e.g. architecture), context, and value creation, to enable both better education and decision-making in any given situation. An improved understanding of these links depends on developing better models of sources of value that are available to be exploited by software designers in the first place (e.g., real options).

Third, people involved in decision-making have to be educated in how to employ technical means more effectively to create value. In particular, they personally need to have a better understanding of the sources of value to be exploited and the links between technical decisions and the capture of value.

Fourth, dynamic monitoring and control mechanisms are needed to better guide decision-makers through the design space in search of value added over time. These mechanisms have to be based on models of links between technical design and value and on system-specific models and databases that capture system status, valuation, risk, and so on: not solely as functions of endogenous parameters, such as software development cost drivers, but also of any relevant exogenous parameters, such as the price of memory, competitor behavior, macroeconomic conditions, etc.

These system specific models are based on better cost and payoff models and estimation and tracking capabilities, at the center of which is a business-case model for a given project, program or portfolio. We now discuss some of the central elements of this roadmap in more detail.

Richer Design Spaces

The space in which software designers operate today is inadequate. One of the important reasons for this is that the market structures within which software development occurs are still primitive in comparison to those supporting other industries. We are less able to build systems from specialized, efficiently produced, volume-priced third-party components than is possible in many other fields. We are also less able to use markets to manage risk through warranties, liability insurance, etc., than is common in most fields. The inability to manage risk by the use of market mechanisms is a major hindrance to efficient production.

Links Between Technical Parameters and Value

Software design involves both technical and managerial decisions. The use of formal methods or the shape of an architecture are technical issues. The continuation or reorientation of a program in light of new information is managerial. The two are not entirely separable. The selection of a life–cycle model is a technical decision about the managerial framework for a system. Moreover, even where software engineering is concerned with technical issues, the connection to value creation is what matters. The promotion of Parnas' concept of information hiding modules, for example, is based on the following rationale: most of the life-cycle cost of a software system is expended change [Lientz-Swanson, 1980]. For a system to create value, the cost of an increment should be proportional to the benefits delivered; but if a system has not been designed for change, the cost will be disproportionate to the benefits [Parnas, 1972]. Information hiding modularity is a key to design for change.

Design for change is thus promoted as a value-maximizing strategy provided one could anticipate changes correctly. While this is a powerful heuristic, we lack adequate models of the connections between this technical concept and value creation under given circumstances. What is the relationship between information hiding modularity and design interval? Should one design for change if doing so take any additional time in an extremely competitive marketplace in which speed it market is a make-or-break issue? Is information hiding obligatory if the opportunity cost of delay might be enormous? What is performance if of the essence? How does the payoff from changing the system relate to the cost of enabling the change?

What role does the timing of the change play? What if it is not likely to occur until far in the future? What if the change cannot be anticipated with certainty, but only with some degree of likelihood? What if the change is somewhat unlikely to be needed but in the case that it is needed, the payoff would be great [Sullivan et. al., 1999]? Va lue-optimal technical design choices depend on many such factors.

Similarly, early advocates of the aggressive use of formal methods promoted them on the grounds that software could not be made adequately reliable using only informal and ad hoc methods, but only through the use of formal methods. Some thought that systems that could not be proven right should not be built. The implied hypothesis (all too often promoted as fact) was that using formal methods was optimal for value, if only because value simply could not be created, not of cost and risk, otherwise.

Subsequent experience has shown that hypothesis to have been wildly incorrect. In particular, it has turned out to be possible to create tremendous value without formal methods. Some early advocates have admitted that and pose interesting questions about why things turned out this way. One answer is that the assumed links were based on a view of software products as relatively unchanging that turned out not to be an accurate view.

We are not saying that formal methods cannot add value. They obviously can in some circumstances: e.g., for high-volume, unchanging artifacts, such as automotive steering-gear firmware. We still do not understand adequately the economic parameters under which investments in the use of formal methods create value. Recent work, e.g., of Praxis, Inc., is improving our understanding. Serious attempts to support limited but significant formal methods in industrial, object-oriented design modeling frameworks, such as the Catalysis variant of UML [D'Souza-Wills, 1999], should provide additional information over time.

Summary

To summarize, value-based software engineering is unavoidably involved with software and information system product and process technology, and their interaction with human values. It is strongly empirical, but includes new concepts in need of stronger theory. It uses risk considerations to balance software discipline and flexibility, and to answer other key "how much is enough?" questions. And it helps illuminate information technology policy by identifying the quantitative and qualitative sources of cost and value.

As seen in the discussion of software project failure sources above, the value-based approaches cited can be applied to avoid many current project failures. They also provide starting points for further high-leverage research to broaden and strengthen the techniques, and to extend them to address emerging future software development approaches involving COTS, open source, rapid development, agile methods, and global distributed development. Figure 1 provides a candidate roadmap for pursuing such a research program.

References

[Amram-Kalutilaka, 1999]. M. Amram and N. Kalutilaka, *Real Options*, Harvard Business School Press, Cambridge, Mass., 1999.

[Baldwin-Clark, 1999]. Baldwin, C. and K. Clark, *Design Rules: The Power of Modularity*, MIT Press, 1999.

[Boehm, 1989]. Boehm, B., *Software Risk Management*, IEEE-CS Press, 1989.

[Boehm-Basili, 2001]. Boehm, B. and V. Basili, "Software Defect Reduction Top 10 List," IEEE *Computer*, January 2001.

[Boehm-Brown, 2001]. Boehm, B. and A.W. Brown, "Mastering Rapid Delivery and Change with the SAIV Process Model," *Proceedings, ESCOM 2001*, April 2001, pp. 147-156.

[Boehm-Ross, 1989]. Boehm, B. and R. Ross, "Theory-W software project management: principles and examples," *IEEE Transactions on Software Engineering*, 15(7):902-916, July 1989.

[Boehm-Sullivan, 2000]. Boehm, B. and K. Sullivan, "Software Economics: A Roadmap," *The Future of Software Economics*, IEEE/ACM, 2000.

[Carmel et al., 1993]. Carmel, E., R. Whitaker, and J. George, "PD and Joint Application Design: A Transatlantic Comparison," *Communications of the ACM*, June 1993, pp. 40-48.

[D'Souza-Wills, 1999]. D'Souza, D.F. and A.C. Wills, *Objects, Components and Frameworks with UML: The Catalysis Approach*, Addison Wesley, Reading, Mass., 1999.

[Dardenne et al., 1991]. Dardenne, A., S. Fickas, and A. van Lamsweerde, "A Goal directed Concept Acquisition in Requirements Elicitation," *Proceedings, IWSSD* 6, *IEEE*, 1991, pp. 14-21.

[Eureka-Ryan, 1988]. Eureka, W. and N. Ryan, *The Customer-Driven Company: Managerial Perspectives on QFD*, ASI Press, 1988.

[Friedman-Leondes, 1969]. Friedman, G. and Leondes, C., "Constraint Theory," *IEEE Trans. Systems Sciences and Cybernetics*, January 1969, pp. 48-57.

[Goldratt, 1990]. Goldratt, E., *What is this Thing Called Theory of Constraints and How Should It Be Implemented?*, North River Press, 1990.

[Karten, 1994]. Karten, N., *Managing Expectations*, Dorset House, 1994.

[Keeney-Raiffa, 1993]. Keeney, R.L. and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, (Cambridge, England: Cambridge University Press), 1993.

[Lientz-Swanson, 1980]. Lientz, B.P. and E.B. Swanson, *Software Maintenance Management*, Addison-Wesley, Reading, Mass., 1980.

[Majchrzak et al., 2001]. Majchrzak, A. and C. Beath, "Beyond User Participation: A Process Model of Learning and Negotiation During Systems Development." In A. Seagards, J. Sampler, and R. Zmud (Eds), *Refining the Organizational Roles of Information Technology in the Information Age*. University of Minnesota Press, 2001.

[McConnell, 1996]. McConnell, S., *Rapid Development*, Microsoft Press, 1996.

[Parnas, 1972]. Parnas, D.L., "On the criteria to be used in decomposing systems into modules", *Communications of the Association of Computing Machinery*, 1972, 15(12) pp. 1053–58.

[Standish, 1995]. Standish Group, "CHAOS," http://www.standishgroup.com/chaos.htm.

[Sullivan et al., 1999]. Sullivan, K.J., P. Chalasani, S. Jha and V. Sazawal, "Software Design as an Investment Activity: A Real Options Perspective," *Real Options and Business Strategy: Applications to Decision Making*, L. Trigeorgis, ed., (London, England: Risk Books), 1999, pp. 215-261.

[Sullivan et al., 2001]. Sullivan, K., Y. Cai, B. Hallen, W. Griswold, "The Structure and Value of Modularity in Software Design," Proceedings, ESEC/FSE 2001, ACM Press, pp. 99-108.

[Thorp, 1998]. Thorp, J. and DMR's Center for Strategic Leadership, *The Information Paradox: Realizing the Benefits of Information Technology*, McGraw-Hill, 1998.

[Young, 2001]. Young, R., *Effective Requirements Practices*, Addison-Wesley, 2001.