Value-Based Software Engineering: Overview and Agenda

Barry Boehm
USC-CSE-2005-504, February 2005

Abstract: Much of current software engineering practice and research is done in a value-neutral setting, in which every requirement, use case, object, test case, and defect is equally important. However most studies of the critical success factors distinguishing successful from failed software projects find that the primary critical success factors lie in the value domain.

The value-based software engineering (VBSE) agenda discussed in this chapter and exemplified in the other chapters involves integrating value considerations into the full range of existing and emerging software engineering principles and practices. The chapter then summarized the primary components of the agenda: value-based requirements engineering, architecting, design and development, verification and validation, planning and control, risk management, quality management, people management, and an underlying theory of VBSE. It concludes by approaches for going toward VBSE at the project, organization, national, or global level.

Keywords: benefits realization, business case analysis, cost-benefit analysis, investment analysis, return on investment, risk management, stakeholder values, software economics, value-based software engineering.

21.1 Overview and Rationale

Much of current software engineering practice and research is done in a value-neutral setting, in which:

- Every requirement, use case, object, test case and defect is treated as equally important;
- Methods are presented and practiced as largely logical activities involving mappings and transformations (e.g., object-oriented development);
- "Earned value" systems track project cost and schedule, not stakeholder or business value;

- A "separation of concerns" is practiced, in which the responsibility of software engineers is confined to turning software requirements into verified code.

In earlier times, when software decisions had relatively minor influences on a system's cost, schedule, and value, the value-neutral approach was reasonably workable. But today and increasingly in the future, software has a major influence on most systems' cost, schedule, and value; and software decisions are inextricably intertwined with system-level decisions.

Also, value-neutral software engineering principles and practices are unable to deal with most of the sources of software project failure. Major studies such as the Standish Group's CHAOS reports (Standish, 1995; Standish 2001) find that most software project failures are caused by value-oriented shortfalls such as lack of user input, incomplete requirements, changing requirements, lack of resources, unrealistic expectations, unclear objectives, and unrealistic time frames.

Further, value-neutral methods are insufficient as a basis of an engineering discipline. The definition of "engineering" in (Webster, 2002) is "the application of science and mathematics by which the properties of matter and sources of energy in nature are made useful to people." Most concerns expressed about the adequacy of software engineering focus on the shortfalls in its underlying science. But it is also hard for a value-neutral approach to provide guidance for making its products useful to people, as this involves dealing with different people's utility functions or value propositions.
It is also hard to make financially responsible decisions using value-neutral methods. Let us illustrate this with an example.

Example: Automated Test Generation (ATG)

Suppose you are the manager of a $2 million software project to develop a large customer billing system. A vendor of an automated test generation (ATG) tool comes to you with the following proposition:

> "Our tool has been shown to cut software test costs in half. Your test costs typically consume 50% of your total development costs, or $1 million for your current project. We'll provide you the use of the tool for 30% of your test costs, or $300K. After you've used the tool and saved 50% of your test costs, or $500K, you'll be $200K ahead".

How would you react to this proposition?

The usual response for traditionally educated software engineers is to evaluate it from a technical and project management standpoint. An excellent literature review and experience paper (Persson and Yilmazturk, 2004) compiled 34 good technical and project management reasons why an ATG tool might not save you 50% of your test costs. The reasons included unrepresentative test coverage; too much output data; lack of test validity criteria; poor test design; instability due to rapid feature changes; lack of management commitment; and lack of preparation and experience ("automated chaos yields faster chaos").

Often, though, a more serious concern lies outside traditional software engineering technology and management considerations. It is that ATGs, like most current software engineering methods and tools, are value-neutral. They assume that every requirement, test case, and defect are equally important.

However, the more usual situation is a Pareto distribution in which 80% of the mission value comes from 20% of the software components. The data in Figure 21.1 are a good illustration of this phenomenon. The Pareto curve in Figure 21.1 comes from an experience report (Bullock, 2000) in which each customer billing type tested led to improved initial billing revenues from 75% to 90% and much lower customer complaint rates; and that one of the 15 customer types accounted for 50% of the billing revenues. The straight-line curve in Figure 21.1 is the usual result of ATG-driven testing, in which the next test is equally likely to have low or high business value.

Actual business value

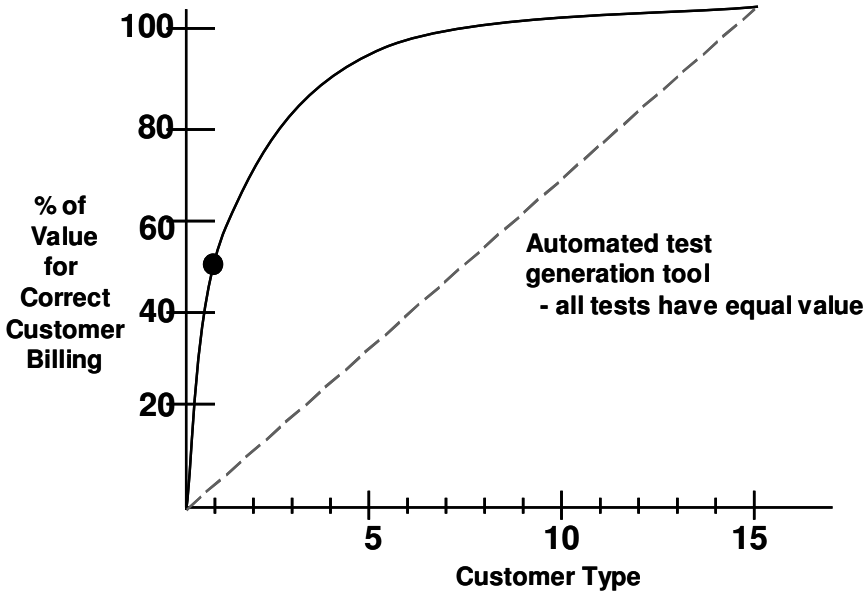

Figure 21.1 Pareto 80-20 distribution of test case value

Table 21.1 shows the relative levels of investment costs, business benefits, and returns on investment ROI = (benefits – costs) / costs, for the value-neutral ATG testing and value-based Pareto testing strategies. Figure 21.2 provides a graphic comparison of the resulting ROIs. The analysis is based on the following assumptions.

- $1M of the development costs have been invested in the customer billing system by the beginning of testing.
- The ATG tool will cost $300K and will reduce test costs by 50% as promised.
- The business case for the system will produce $4M in business value in return for the $2M investment cost.
- The business case will provide a similar 80:20 distribution for the remaining 14 customer types.

Table 21.1 Comparative Business Cases: ATG and Pareto Testing

% of Tests Run	ATG Testing				Pareto Testing			
	Cost	Value	Net Value	ROI	Cost	Value	Net Value	ROI
0	1300	0	-1300	-1.0	1000	0	-1000	-1.0
10	1350	400	-950	-.70	1100	2560	1460	+1.33
20	1400	800	-600	-.43	1200	3200	2000	1.67
40	1500	1600	100	+.07	1400	3840	2440	1.74

60	1600	2400	800	.50	1600	3968	2368	1.48
80	1700	3200	1500	.88	1800	3994	2194	1.21
100	1800	4000	2200	1.22	2000	4000	2000	1.0

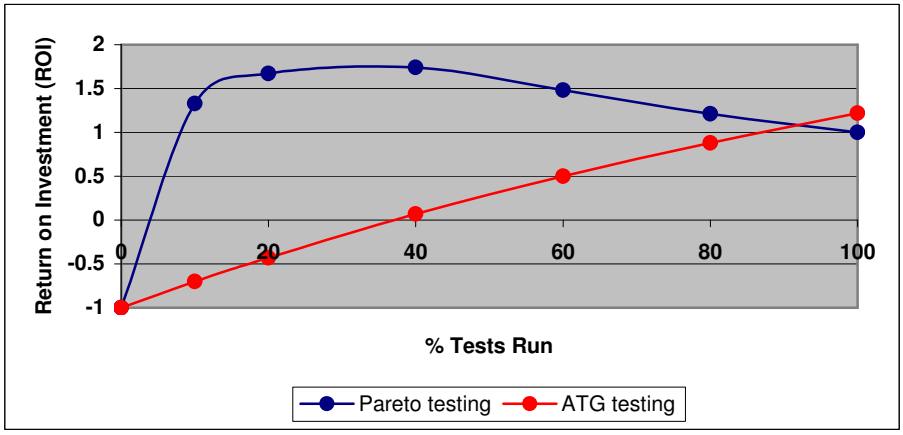

Figure 21.2 ROI: Value-Neutral ATG vs. Pareto Analysis

As seen in Table 21.1 and more graphically in Figure 21.2, the value-neutral ATG approach does achieve a cost reduction and a higher ROI of 1.22 at the 100% tested point. But the value-based Pareto testing approach determines that a much higher ROI of 1.74 can be achieved by running only about 40% of the most valuable tests. Beyond that point, the remaining $600K of test investment will generate only $160K in business value, and is a poor investment of scarce resources. Some further considerations are:

- There may be qualitative reasons to test all 15 of the customer types. Frequently, however, the lessons learned in testing type 1 will identify a more cost-effective subset of tests to be run on the remaining 14 customer types.
- A pure focus on cost reduction can produce a poor ROI profile.
- From a cost-of-money standpoint, much of the benefit from the ATG strategy comes in less-valuable future cash flows.
- However, appropriate combinations of Pareto-based and ATG-based testing may produce even higher ROIs.

From a global standpoint, it appears that finding out which 60% of an organization's testing budget actually produces a negative ROI would be highly worthwhile. Estimates of global software costs are approaching $1 trillion per year. If

half of this is spent on testing, and 60% of the test effort can be profitably eliminated, this creates a $300 billion per year cost savings potential for such value-based testing investments.

21.2 Value-Based Software Engineering (VBSE) Background and Agenda

The treatment of information as having economic value (Marschak, 1974) and the economics of computers, software, and information technology have been topics of study for some time (Sharpe, 1969; Phister, 1979; Kleijnen, 1980; Boehm, 1981). A community of interest in Economics-Driven Software Engineering Research (EDSER) has been holding annual workshops since 1999 [Sullivan et. al., 1999-2004]. Special issues on Return on Investment have appeared in journals such as IEEE Software (Erdogmus et. al., 2004), and books have been increasingly appearing on topics such as software business case analysis (Reifer, 2002), customer value-oriented agile methods (Cockburn, 2002; Highsmith, 2002; Schwaber and Beedle, 2002) and investment-oriented software feature prioritization and analysis (Denne and Cleland-Huang, 2004; Tockey, 2004).

A resulting value-based software engineering (VBSE) agenda has emerged, with the objective of integrating value considerations into the full range of existing and emerging software engineering principles and practices, and of developing an overall framework in which they compatibly reinforce each other. Some of the major elements of this agenda and example results in this book's chapters and elsewhere are discussed next.

Value-based requirements engineering includes principles and practices for identifying a system's success-critical stakeholders; eliciting their value propositions with respect to the system; and reconciling these value propositions into a mutually satisfactory set of objectives for the system. Example results include the release prioritization techniques in Chapter 22 and in (Denne and Cleland-Huang, 2004); the requirements prioritization techniques in Chapter 23 and (Karlsson and Ryan, 1997); the business case analysis techniques in (Reifer, 2002) and Chapter 15; and the stakeholder identification and requirements negotiation techniques in Chapters 15, 35 and 36.

Value-based architecting involves the further reconciliation of the system objectives with achievable architectural solutions. Example results include the multi-attribute decision support and negotiation techniques in Chapters 13 and 36; the Software Engineering Institute's Architecture Tradeoff Analysis work in (Kazman et. al., 2002) and (Clements et. al., 2002); the concurrent system and software engineering approach in Chapter 15; the software traceability techniques in Chapter 31; and the value-based software product line analyses in (Favaro, 1996) and (Faulk et. al., 2000).

Value-based design and development involves techniques for ensuring that the system's objectives and value considerations are inherited by the software's design and development practices. Example results include the software traceability techniques in Chapter 31; the development tool analysis methods in Chapter 44; the process improvement ROI analysis in (van Solingen, 2004); and the customer-oriented design and development techniques in agile methods.

Value-based verification and validation involves techniques for verifying and validating that a software solution satisfies its value objectives; and processes for sequencing and prioritizing V&V tasks operating as an investing activity. Example results include the the value-based testing techniques and tool investments in Chapters 32, 35, and 44; and the risk-based testing techniques in (Gerrard and Thompson, 2002).

Value-based planning and control includes principles and practices for extending traditional cost, schedule, and product planning and control techniques to include planning and control of the value delivered to stakeholders. Example results include the value-based planning and control techniques in Chapters 12 and 15, and in (Boehm and Huang, 2003); the multi-attribute planning and decision support techniques in Chapter 13; and the release planning techniques in Chapter 22 and (Denne and Cleland-Huang, 2004).

Value-based risk management includes principles and practices for risk identification, analysis, prioritization, and mitigation. Example results include the software risk management techniques in (Boehm, 1989; Charette, 1989; De Marco-Lister, 2003); the risk-based "how much is enough" techniques in Chapter 15; the risk-based analysis of the value of project predictability in Chapter 24; the risk-based simulation profiles in Chapter 33; the risk-based testing techniques in (Gerrard and Thompson, 2002); the insurance approach to risk management in (Raz and Shaw, 2001); and the real-options analyses of intellectual property protection in Chapter 41, of modular design in (Sullivan et al., 2001), and of agile methods in (Erdogmus and Favaro, 2002).

Value-based quality management includes the prioritization of desired quality factors with respect to stakeholders' value propositions. Example results include the multi-attribute decision support techniques in Chapter 13, the quality as stakeholder value approach in (Boehm and In, 1996), the Soft Goal approach in (Chung et. al., 1999), and the value-based approach to computer security in (Butler, 2002).

Value-based people management includes stakeholder teambuilding and expectations management; managing the project's accommodation of all stakeholders'

value propositions throughout the life cycle; and integrating ethical considerations into daily project practice. Example results include the value-based personal preferences work in Chapter 34, the approaches to developing shared tacit knowledge in agile methods, and the use of Rawls' Theory of Justice (Rawls, 1971) as a stakeholder value-based approach to software engineering ethics in (Collins et. al., 1994) and Chapter 15.

A theory of value-based software engineering, that connects software engineering's value-neutral computer science theory with major value-based theories such as utility theory, decision theory, dependency theory, and control theory; and that provides a process framework for guiding VBSE activities. An initial formulation of such a theory is provided in Chapter 14.
The companion Chapter 15 summarizes seven key elements which provide a starting point for realizing the value-based software engineering agenda. They are:

1. Benefits Realization Analysis
2. Stakeholder Value Proposition Elicitation and Reconciliation
3. Business Case Analysis
4. Continuous Risk and Opportunity Management
5. Concurrent System and Software Engineering
6. Value-Based Monitoring and Control
7. Change as Opportunity

21.3 Getting Started Toward Value-Based Software Engineering

21.3.1 Going Toward VBSE At the Project or Organization Level

The material in Section 21.3.1 assumes that you have read Chapter 15 on the seven key VBSE elements.

At the individual project or organization level, there are individual steps you can take for each of the seven key elements of value-based software engineering. They are fairly compatible, and can be pursued in various combinations. As with most changes, it is best to start small with a receptive pilot project with good chances of demonstrating early value.

1. Benefits-Realization Analysis. Write down the name of your software initiative and its specific deliverables as its contribution as the left-hand end of a Results Chain, and your stakeholders' desired outcome(s) as the right-hand end. Then try to fill out the Results Chain with any success-critical assumptions, intermediate outcomes and contributions, and additional initiatives needed to fully realize the desired outcome(s). There usually will be some added initiatives, and they will often identify some missing

success-critical stakeholders, such as operators, maintainers, owners of complementary systems, and additional classes of users.

2. Stakeholder Value Proposition Elicitation and Reconciliation. Use the Results Chain to interview your success-critical stakeholders to validate it and identify their additional high-priority assumptions, initiatives, and outcomes. Use the Model Clash Spiderweb as a top-level checklist, and as a source for identifying model clashes that need to be reconciled among the stakeholders into a mutually satisfactory or win-win set of agreements. Summarize the results and coordinate them with the stakeholders via a Shared Vision document or its equivalent. A simple Shared Vision document would include an "elevator description" of the project and its desired outcome(s), the corresponding Results Chain, a list of the success-critical stakeholders and their roles, a System Block Diagram indicating the desired scope and boundary of the system to be developed and a list of the major project constraints. More detailed guidelines are in Section 2 of the MBASE Operational Concept Description Guidelines at http://sunset.usc.edu/research/MBASE.

3. Business Case Analysis. Do a simple (e.g. analogy-based) estimate of the costs of developing, installing and operating your proposed system over your chosen benefits period. Do a similarly simple estimate of the resulting benefits across the benefits period. For an order processing system, for example, these could be both cost savings and increased sales and profits. Construct a chart similar to Figure 15.3 in Chapter 15 showing the cumulative return on investment, ROI= (benefits-costs)/costs. Also list the qualitative benefits, such as improved order fulfillment predictability and control and improved customer satisfaction. Iterate the business case with your stakeholders and ensure that they agree that it is worthwhile to proceed. Don Reifer's book, Making the Software Business Case (Reifer, 2002) provides further guidelines and case study examples. An example order processing case study is provided in (Boehm and Huang, 2003) and Chapter 12.

4. Continuous Risk and Opportunity Management. Any uncertainties in your business case analysis, or in your ability to realize the outcomes in your Results Chain, are sources of risk that you should eliminate early (via prototyping, user surveys, COTS evaluation, etc.), or develop plans and fallbacks for managing their future elimination. Also identify a focal point person for doing technology watch or marketplace watch activities to identify potential new risks or opportunities.

5. Concurrent System and Software Engineering. Rather than sequentially developing operational concepts, software requirements, prototypes, COTS and platform choices, architectures, and life cycle plans, perform these concurrently. Use the equivalent of the MBASE and Rational Unified Process Life Cycle Objectives (LCO) and Life Cycle Architecture (LCA) milestones discussed in Section 15.5 in Chapter 15 as stakeholder review and commitment points.

6. Value-Based Monitoring and Control. Use the Results Chain in step 1 to monitor the validity of assumptions, actual vs. expected contributions, and outcomes. Similarly, monitor the actual vs. estimated costs and benefits in the business case, and

update the estimates at major milestones such as LCO and LCA. Also, continuously monitor the status of project risks and opportunities, and balanced-scorecard results such as customer satisfaction. Determine appropriate corrective actions for any progress/plan/goal mismatches. Set up a simple pilot experience base for accumulating lessons learned and key metrics data (software productivity and quality metrics; balanced scorecard results) at the organizational level.

7. Change as Opportunity. For small, non-critical projects with rapidly changing or highly emergent requirements, experiment with using one of the agile methods, enhanced where appropriate by the value-based steps above. For larger, more critical projects, determine the most likely sources of requirements change and modularize the system to accommodate these sources of change. Again, continuously monitor technology and the marketplace to identify and reorient the project to address unanticipated risks and opportunities. Where these are rapidly changing, experiment with hybrid plan-driven and agile methods within an architectural framework addressing the most critical and stable requirements. Process frameworks for medium and large-size hybrid plan-driven and agile methods are provided in (Boehm and Turner, 2004).

21.3.2 Going Toward VBSE at the National or Global Level

Figure 21.3 shows a roadmap for making progress toward Value-Based Software Engineering and its benefits on a national or global level (Boehm and Sullivan, 2000). In the spirit of concurrent software and system engineering, it focuses its initiatives, contributions, and outcomes at the combined software and information technology (SW/IT) level. Its overall goals are to develop fundamental knowledge and practical techniques that will enable 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, the roadmap in Figure 21.3 identifies a network of important intermediate outcomes. It 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 and benefit 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, and using the results to improve software engineering and information technology policy, research, and education.

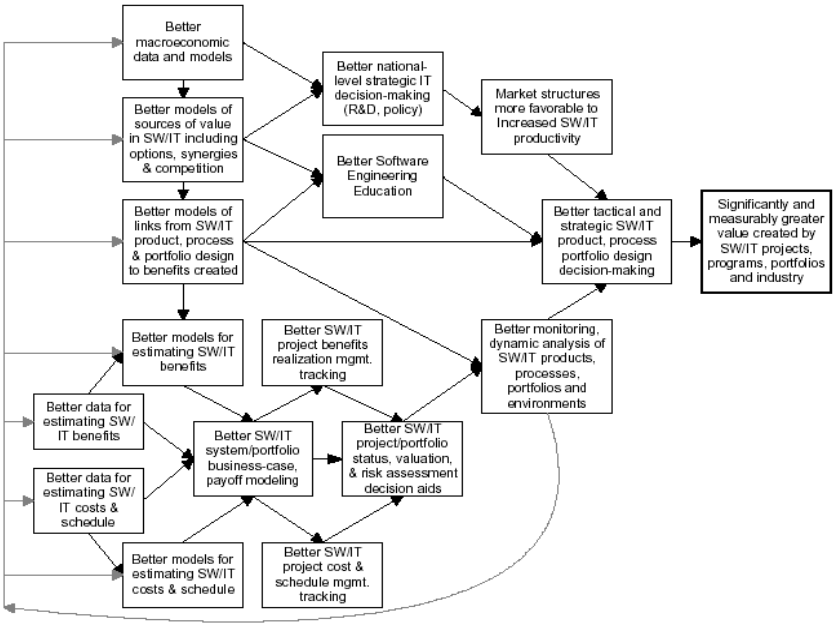

Figure 21.3: Roadmap for realizing benefits of value-based software engineering

Making Decisions That Are Better for Value Creation

The goal of the roadmap is supported by a key intermediate outcome: designers and managers at all levels must make decisions that are better for value added than those they make today. Value-based 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 national software policy. Better decision making is the key enabler of greater value added.

Value-based decision making depends in turn on a set of other advances. First, the option space within which managers and designers operate needs to be sufficiently rich. To some extent, the option 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 R&D investment policy, on anti-trust, and so forth. The market structure determines the materials that are produced that managers and designers can then employ, and their properties.

Second, as a field we need to understand better the links between technical design mechanisms (e.g., architecture), context, and value creation, to enable both better education and decisionmaking 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 managers and designers in the first place (e.g., real options).

Third, people involved in decisionmaking 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 decisionmakers through the option 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 software engineering parameters, such as software development cost drivers, but also of any relevant external parameters, such as the price of memory, competitor behavior, macroeconomic conditions, etc., as discussed in Section 15.6 in Chapter 15.

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. Further elements of this roadmap are discussed in more detail in (Boehm and Sullivan, 2000).

21.4 Summary and Conclusions

As indicated in the automated test generator (ATG) example in Section 21.1, the use of value-neutral software engineering methods often causes software projects to expend significant amounts of scarce resources on activities with negative returns on investment. Just in the area of software testing, the magnitude of this wasted effort could be as high as $300 billion per year worldwide.

As indicated by the chapters in this book and related literature in the References, substantial progress is being made towards realizing the value-based software engineering (VBSE) agenda of integrating value considerations into the full range of existing and emerging software engineering practices, and of developing an overall framework in which they compatibly reinforce each other.

The seven key VBSE elements in Chapter 15 – benefits realization; stakeholder value proposition elicitation and reconciliation; business case analysis; continuous risk and opportunity management; concurrent system and software engineering; value-based monitoring and control; and change as opportunity – provide a starting point for realizing the VBSE agenda, along with the initial theory and VBSE process framework presented in Chapter 14. Evolutionary approaches for going toward VBSE practices at project, organization, national, and global levels are provided in Section 21.3.

The transition to value-based software engineering is necessarily evolutionary because it hasn't all been invented yet. There are no mature packages available on the shelf for performing software benefits analysis or value-based earned value tracking. As with everything else in information technology, VBSE is undergoing considerable change. And those who embrace this source of change as opportunity will be the first and fastest to reap its rewards.

21.5 Acknowledgements

This paper is based on research supported by the National Science Foundation, the DoD Software Intensive Systems Directorate, and the affiliates of the USC Center for Software Engineering. It owes a great deal to discussions with the USC-CSE principals, with participants in the Economics-Driven Software Engineering Research (EDSER) workshops, and with participants in the International Software Engineering Research Network (ISERN) workshops, and with the authors of the other chapters in this book. Much of section 21.3.2 was co-authored by Kevin Sullivan in (Boehm and Sullivan, 2000).

21.6 References

B. Boehm, Software Engineering Economics, Prentice Hall, 1981.

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

B. Boehm and H. In, "Identifying Quality-Requirement Conflicts", IEEE Software, March 1996, pp. 25-35.

B. Boehm and L. Huang, "Value-Based Software Engineering: A Case Study", Computer, March 2003, pp. 33-41.

B. Boehm and K. Sullivan, "Software Economics: A Roadmap," The Future of Software Economics, A. Finkelstein (ed.), ACM Press, 2000, pp. 319-343.

J. Bullock, "Calculating the Value of Testing", Software Testing and Quality Engineering, May/June 2000, pp. 56-62.

S. Butler, "Security Attribute Evaluation Method: A Cost-Benefit Approach", Proceedings, ICSE 2002, ACM/IEEE, May 2002, pp.232-240.

R. Charette, Software Engineering Risk Analysis and Management, McGraw Hill, 1989

L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, Non-Functional Requirements in Software Engineering, Kluwer, 1999.

P. Clements, R. Kazman, and M. Klein, Evaluating Software Architecture: Methods and Case Studies, Addison Wesley, 2002.

A. Cockburn, Agile Software Development, Addison Wesley, 2002.

W. Collins, K. Miller, B. Spielman, and J. Wherry, "How Good is Good Enough?", Comm. ACM, January 1994, pp. 81-91.

T. De Marco, T. Lister, Waltzing with Bears, Dorset House, 2003.

M. Denne and J. Cleland-Huang, Software by Numbers, Prentice Hall, 2003.

H. Erdogmus, J. Favaro, and W. Strigel (eds.), "Special Issue: Return on Investment", IEEE Software, May/June 2004.

H. Erdogmus and J. Favaro, "Keep your Options Open: Extreme Programming and the Economics of Flexibility", in G. Succi et. al. (eds.), Extreme Programming Perspectives, Addison Wesley, 2002, pp. 503-552.

J. Favaro, "A Comparision of Approaches to Reuse Investment Analysis", Proceedings, ICSR 4, IEEE, 1996.

S Faulk, D. Harmon, and D. Raffo, "Value-Based Software Engineering (VBSE): A Value-Driven Approach to Product-Line Engineering," Proceedings, First International Conference on Software Product Line Engineering, August 2000.

P. Gerrard and N. Thompson, Risk-Based E-Business Testing, Artech House, 2002.

J. Highsmith, Agile Software Development Ecosystems, Addison Wesley, 2002.

J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements", IEEE Software, September-October, 1997, pp. 67-74.

R. Kazman, J. Asundi, and M. Klein, "Quantifying the Costs and Benefits of Architectural Decisions", Proceedings, ICSE 2001, ACM/IEEE, pp. 297-306.

J. Kleijnen, Computers and Profits: Quantifying Financial Benefits of Information, Addison Wesley, 1980.

J. Marschak, Economic Information, Decision, and Prediction (3 vol.), 1974.

C. Persson and N. Yilmazturk, "Establishment of Automated Regression Testing at ABB: Industrial Experience Report on 'Avoiding the Pitfalls", Proceedings, ISESE 2004, IEEE, August 2004, pp. 112-121.

M. Phister, Data Processing Technology and Economics, Digital Press, 1979.

J. Rawls, A Theory of Justice, Belknap/Harvard U. Press, 1971, 1999.

O. Raz and M. Shaw, "Software Risk Management and Insurance", Proceedings, EDSER-3, IEEE-CS Press, 2001.

D. Reifer, Making the Software Business Case, Addison Wesley, 2002.

K. Schwaber and M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2002.

W. Sharpe, The Economics of Computers, Columbia U. Press, 1969.

The Standish Group, CHAOS Report, 1995, 2001; www.standishgroup.com

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

S. Tockey, Return on Software, Addison Wesley, 2004.

R. van Solingen, "Measuring the ROI of Software Process Improvement", IEEE Software, May/June 2004, pp. 32-38.

Webster's Collegiate Dictionary, Merriam-Webster, 2002.

Author Biography

Barry Boehm is the TRW Professor of Software Engineering and Director of the Center for Software Engineering at USC. His current research interests include software process modeling, software requirements engineering, software architectures, software metrics and cost models, software engineering environments, and value-based software engineering. His contributions to the field include the Constructive Cost Model (COCOMO), the Spiral Model of the software process, and the Theory W (win-win) approach to software management and requirements determination. He is a Fellow of the primary professional societies in computing (ACM), aerospace (AIAA), electronics (IEEE), and systems engineering (INCOSE), and a member of the US National Academy of Engineering.