

1 Value-Based Software Engineering: Overview and Agenda

Barry Boehm

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 summarizes 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 with a global road map for realizing the benefits of VBSE.

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

1.1 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 and will have 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 reports¹ 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 is equally important.

¹<http://www.standishgroup.com>

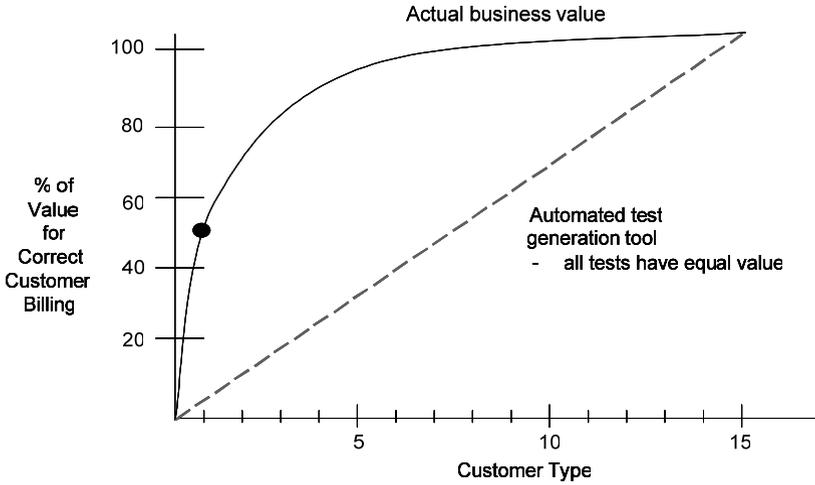


Fig. 1. Pareto 80:20 distribution of test case value

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 1 are a good illustration of this phenomenon. The Pareto curve in Figure 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 one of the 15 customer types accounted for 50% of the billing revenues. The straight line curve in Figure 1 is the usual result of ATG-driven testing, in which the next test is equally likely to have low or high business value.

Table 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	1,300	0	-1,300	-1.00	1,000	0	-1,000	-1.00
10	1,350	400	-950	-0.70	1,100	2,560	1,460	+1.33
20	1,400	800	-600	-0.43	1,200	3,200	2,000	1.67
40	1,500	1,600	100	0.07	1,400	3,840	2,440	1.74
60	1,600	2,400	800	0.50	1,600	3,968	2,368	1.48
80	1,700	3,200	1,500	0.88	1,800	3,994	2,194	1.21
100	1,800	4,000	2,200	1.22	2,000	4,000	2,000	1.00

Table 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 2 provides a graphical

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.

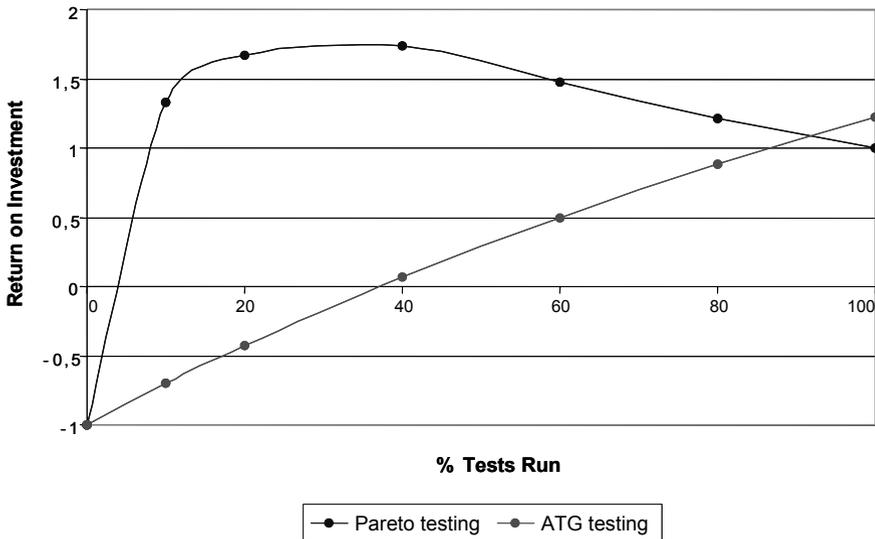


Fig. 2. ROI: Value-neutral ATG vs. Pareto Analysis

As seen in Table 1 and graphically in Figure 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.

1.2 Background and Agenda

Some VBSE Definitions and Background: Some of the dictionary definitions of "value" (Webster, 2002) are in purely financial terms, such as "the monetary worth of something: marketable price." However, in the context of this book, we use the broader dictionary definition of "value" as "relative worth, utility, or importance." This adds complications in requiring VBSE to address less rigorously analyzable situations, but enables it to provide help in addressing software engineering decisions involving personal, interpersonal, or ethical considerations, as discussed in many of the chapters.

Given that the definition of "engineering" above includes the goal of making things useful to people, it would seem that value considerations are already built into the definition of "software engineering." But since so much of current software engineering is done in a value-neutral context, we offer the following definition of VBSE: "the explicit concern with value concerns in the application of science and mathematics by which the properties of computer software are made useful to people." The resulting science includes the social sciences as well as the physical sciences, and the mathematics includes utility theory, game theory, statistical decision theory, and real options theory as well as logic, complexity theory, and category theory.

The book treats the terms "value proposition," utility function," and "win condition" basically as synonyms. "Win condition" is primarily used in the context of stakeholders negotiating mutually satisfactory or win-win agreements, as in Chapters 2 and 7. "Utility function" is primarily used in trying to characterize the nature of a function relating a stakeholder's degree of preference for alternative (often multidimensional) outcomes. "Value proposition" is primarily used as a generic term encompassing both win conditions and utility functions.

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 Re-

search (EDSER) has been holding annual workshops since 1999². 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 VBSE Agenda: 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 12 and in (Denne and Cleland-Huang, 2004); the requirements prioritization techniques in Chapter 9 and (Karlsson and Ryan, 1997); the business case analysis techniques in (Reifer, 2002) and Chapter 6; and the stakeholder identification and requirements negotiation techniques in Chapters 6, 7, and 10.

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 4 and 7; the Software Engineering Institute's Architecture Trade-off Analysis work in (Kazman et al., 2002) and (Clements et al., 2002); the concurrent system and software engineering approach in Chapter 6 and (Grünbacher et al., 2004); the software traceability techniques in Chapter 14; 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 14; the development tool analysis methods in Chapter 16; 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 value-based testing techniques and tool investments in Chapters 10, 11, and 16; 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 in-

²<http://www.cs.virginia.edu/~sullivan/EDSER-7/>

clude planning and control of the value delivered to stakeholders. Example results include the value-based planning and control techniques in Chapters 6 and 8, and in (Boehm and Huang, 2003); the multi-attribute planning and decision support techniques in Chapter 4; and the release planning techniques in Chapter 12 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; DeMarco-Lister, 2003); the risk-based “how much is enough” techniques in Chapter 6; the risk-based analysis of the value of project predictability in Chapter 5; the risk-based simulation profiles in Chapter 13; the risk-based testing techniques in (Gerard 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 17, 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 4, 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 team building 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 15, 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 6.

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 2.

Chapter 6 summarizes seven key elements which provide further context and starting points 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

1.3 A Global Road Map for Realizing VBSE Benefits

Figure 3 shows a road map 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 backward from the end objective, the road map in Figure 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.

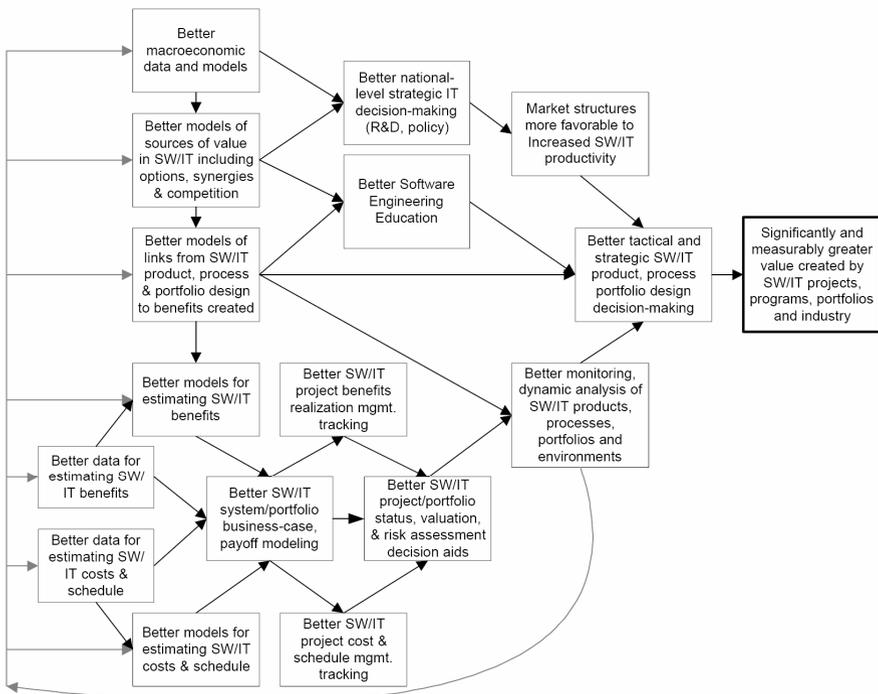


Fig. 3. Road map for realizing benefits of value-based software engineering

Making Decisions That Are Better for Value Creation

The goal of the road map 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: which 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 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 managers and 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 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 6.6.

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 road map are discussed in more detail in (Boehm and Sullivan, 2000).

1.4 Summary and Conclusions

As indicated in the automated test generator (ATG) example in Section 1.1, the use of value-neutral software engineering methods often causes software projects to expend significant amounts of scarce resources on activities with negative re-

turns 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 6 – 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 2. Evolutionary approaches for going toward VBSE practices at project, organization, national, and global levels are provided in Section 1.3.

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.

Acknowledgements

This paper is based on research supported by the National Science Foundation, the DoD Software Intensive Systems Directorate, and the affiliates of the University of Southern California's Center for Software Engineering (USC-CSE). 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 1.3 was co-authored by Kevin Sullivan in (Boehm and Sullivan, 2000).

References

- (Boehm, 1981) Boehm, B. W.: *Software Engineering Economics* (Prentice Hall, 1981)
- (Boehm, 1989) Boehm, B. W.: *Software Risk Management* (IEEE-CS Press, 1989)
- (Boehm and In, 1996) Boehm, B. W. and In H.: Identifying Quality-Requirement Conflicts. *IEEE Software* (March 1996), pp 25–35
- (Boehm and Huang, 2003) Boehm, B. W. and Huang, L.: Value-Based Software Engineering: A Case Study. *IEEE Computer* (March 2003), pp 33–41

- (Boehm and Sullivan, 2000) Boehm, B. W., Sullivan, K., Software Economics: A Roadmap. In: *The Future of Software Economics*, ed by Finkelstein, A. (ACM Press, 2000), pp 319–343
- (Bullock, 2000) Bullock, J.: Calculating the Value of Testing. *Software Testing and Quality Engineering* (May/June 2000), pp 56–62
- (Butler, 2002) Butler, S.: Security Attribute Evaluation Method: A Cost-Benefit Approach. *Proceedings ICSE 2002 (ACM/IEEE, May 2002)*, pp 232–240
- (Charette, 1989) Charette, R.: *Software Engineering Risk Analysis and Management* (McGraw Hill, 1989)
- (Chung et al., 1999) Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: *Non-Functional Requirements in Software Engineering* (Kluwer, 1999)
- (Clements et al., 2003) Clements, P., Kazman, R., Klein, M.: *Evaluating Software Architecture: Methods and Case Studies* (Addison Wesley, 2002)
- (Cockburn, 2002) Cockburn, A.: *Agile Software Development* (Addison Wesley, 2002)
- (Collins et al., 1994) Collins, W., Miller, K., Spielman, B., Wherry, J.: How Good is Good Enough? *Communications of the ACM* (January 1994), pp 81–91
- (DeMarco and Lister, 2003) DeMarco, T., Lister, T.: *Waltzing with Bears* (Dorset House, 2003)
- (Denne and Cleland-Huang, 2003) Denne, M., Cleland-Huang, J.: *Software by Numbers* (Prentice Hall, 2003)
- (Erdogmus et al., 2004) Erdogmus, H., Favaro, J., Strigel, W. (eds.): Special Issue: Return on Investment. *IEEE Software*, (May/June 2004)
- (Erdogmus and Favaro, 2002) Erdogmus, H., Favaro, J.: Keep your Options Open: Extreme Programming and the Economics of Flexibility. In: *Extreme Programming Perspectives*, ed by G. Succi et al. (Addison Wesley, 2002), pp 503–552
- (Favaro, 1996) Favaro, J.: A Comparison of Approaches to Reuse Investment Analysis. *Proceedings, ICSR 4, (IEEE, 1996)*
- (Faulk et al., 2000) Faulk, S., Harmon, D., Raffo, D.: Value-Based Software Engineering (VBSE): A Value-Driven Approach to Product-Line Engineering. *Proceedings, First International Conference on Software Product Line Engineering*, (August 2000)
- (Gerrard and Thompson, 2002) Gerrard, P., Thompson, N.: *Risk-Based E-Business Testing* (Artech House, 2002)
- (Grünbacher et al., 2004) Grünbacher, P., Egyed, A. F., Medvidovic, N.: Reconciling software requirements and architectures with intermediate models. In: *Software and System Modeling (SoSyM)*, Vol. 3, No 3, (Springer, 2004), pp 235–253
- (Highsmith, 2002) Highsmith, J.: *Agile Software Development Ecosystems* (Addison Wesley, 2002)
- (Karlsson and Ryan, 1997) Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. *IEEE Software* (September–October, 1997), pp 67–74

- (Kazman et al., 2001) Kazman, R., Asundi J., Klein, M.: Quantifying the Costs and Benefits of Architectural Decisions. Proceedings, ICSE, (ACM/IEEE, 2001), pp 297–306
- (Kleijnen 1980) Kleijnen, J.: Computers and Profits: Quantifying Financial Benefits of Information (Addison Wesley, 1980)
- (Marschak, 1974) Marschak, J.: Economic Information, Decision, and Prediction (3 volumes), 1974
- (Persson and Yilmazturk, 2004) Persson, C., Yilmazturk, N.: Establishment of Automated Regression Testing at ABB: Industrial Experience Report on ‘Avoiding the Pitfalls.’ Proc. ISESE 2004, IEEE, August 2004, pp 112–121
- (Phister, 1979) Phister, M.: Data Processing Technology and Economics (Digital Press, 1979)
- (Rawls, 1971) Rawls, J.: A Theory of Justice. (Belknap/Harvard U. Press, 1971, 1999)
- (Raz and Shaw, 2001) Raz, O., Shaw, M.: Software Risk Management and Insurance. Proceedings, EDSE-3, IEEE-CS Press, 2001
- (Reifer, 2002) Reifer, D.: Making the Software Business Case (Addison Wesley, 2002)
- (Schwaber and Beedle, 2002) Schwaber, K., Beedle, M.: Agile Software Development with Scrum (Prentice Hall, 2002)
- (Sharpe, 1969) Sharpe, W.: The Economics of Computers (Columbia U. Press, 1969)
- (Sullivan et al., 2001) Sullivan, K., Cai, Y., Hallen B., Griswold, W.: The Structure and Value of Modularity in Software Design. Proceedings, ESEC/FSE, 2001, ACM Press, pp 99–108
- (Tockey, 2004) Tockey, S.: Return on Software (Addison Wesley, 2004)
- (van Solingen, 2004) van Solingen, R.: Measuring the ROI of Software Process Improvement. IEEE Software, May/June 2004, pp 32–38
- (Webster, 2002) 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 the University of Southern California (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.



<http://www.springer.com/978-3-540-25993-0>

Value-Based Software Engineering

Biffi, S.; Aurum, A.; Boehm, B.; Erdogmus, H.;

Grünbacher, P. (Eds.)

2006, XXII, 388 p., Hardcover

ISBN: 978-3-540-25993-0