Value-Based Software Engineering

Barry Boehm
University of Southern California
boehm@sunset.usc.edu

Abstract

Much of current software engineering practice and research is done in a value-neutral setting, in which every requirement, use case, object, and defect is treated as equally important; methods are presented and practiced as largely logical activities; and 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 value-neutral software decisions can seriously degrade project outcomes.

This paper presents an agenda for a discipline of Value-Based Software Engineering. It accepts the challenge of integrating value considerations into all of the existing and emerging software engineering principles and practices, and of developing an overall framework in which they compatibly reinforce each other. Example elements of this agenda include value-based requirements engineering, architecting, design and development, verification and validation, planning and control, risk management, quality management, and people management. It presents seven key elements that provide candidate foundations for value-based software engineering: Benefits Realization Analysis; 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.

Keywords: value, risk, software economics, software life cycle, software management, requirements, architecting, software metrics

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, 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 report [33] 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 [36] 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.

This situation creates a challenge to the software engineering field to integrate value considerations into its principles and practices.

A Value-Based Software Engineering Agenda

Progress has been made over the years to integrate some value-oriented perspectives into software engineering. These include such approaches as participatory design, user engineering, cost estimation, software economics, software investment analysis, and software engineering ethics. However, these have been generally treated as individual extensions to baseline software engineering principles and practices. The Value-Based Software Engineering agenda below accepts the challenge of integrating value considerations into all of the existing and emerging software engineering principles and practices, and of developing an overall framework in which they compatibly reinforce each other. Example elements of this agenda include:

- Value-based requirements engineering, including 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.
- Value-based architecting, involving the further reconciliation of the system objectives with achievable architectural solutions.
- Value-based design and development, involving techniques for ensuring that the system's objectives and value considerations are inherited by the software's design and development.
- Value-based verification and validation, involving techniques for verifying and validating that a software solution satisfies its value objectives; and processes for sequencing and prioritizing V&V tasks to operate as an investment activity.
- Value-based planning and control, including 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.
- Value-based risk management, including principles and practices for risk identification, analysis, prioritization, and mitigation.
- Value-based quality management, including the prioritization of desired quality factors with respect to stakeholders' value propositions.
- Value-based people management, including 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.
- Value-based principles and practices addressing emerging software engineering challenge areas: COTS-based systems, rapid development, agile methods, high dependability systems, systems of systems, and ethics.

The remainder of this paper focuses on seven key elements which provide a starting point for realizing the value-based software engineering agenda.

2. Value-Based Software Engineering: Seven Key Elements

Here are seven key elements that provide candidate foundations for value-based software engineering:

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

2.1 Benefits Realization Analysis

Benefits Realized

Many software projects fail by succumbing to the "Field of Dreams" syndrome. This refers to the American movie in which a Midwestern farmer has a dream that if he builds a baseball field on his farm, the legendary players of the past will appear and play on it ("Build the field and the players will come").

In *The Information Paradox* [35], John Thorp discusses the paradox that organizations' success in profitability or market capitalization do not correlate with their level of investment in information technology (IT). He traces this paradox to an IT and software analogy of the "Field of Dreams" syndrome: "Build the software and the benefits will come".

To counter this syndrome, Thorp and his company, the DMR Consulting Group, have developed a Benefits Realization Approach (BRA) for determining and coordinating the other initiatives besides software and IT system development that are needed in order for the organization to realize the potential IT system benefits. The most significant of these features, the Results Chain, is discussed next.

Results Chain

Figure 1 shows a simple Results Chain provided as an example in *The Information Paradox*. It establishes a framework linking Initiatives that consume resources (e.g., implement a new order entry system for sales) to Contributions (not delivered systems, but their effects on existing operations) and Outcomes, which may lead either to further contributions or to added value (e.g., increased sales). A particularly important contribution of the Results Chain is the link to Assumptions, which condition the realization of the Outcomes. Thus, in Figure 1, if order-to-delivery time turns out not to be an important buying criterion for the product being sold, (e.g., for stockable commodities such as soap and pencils), the reduced time to deliver the product will not result in increased sales.

The Results Chain is a valuable framework by which software project members can work with their clients to identify additional non-software initiatives that may be needed to realize the potential benefits enabled by the software/IT system initiative. These may also identify some additional success-critical stakeholders who need to be represented and "bought into" the shared vision.

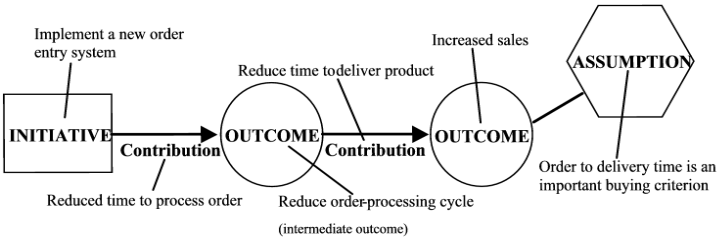

Figure 1. Benefits Realization Approach Results Chain

For example, the initiative to implement a new order entry system may reduce the time required to process orders only if some additional initiatives or system features are pursued to convince the sales people that the new system will be good for their careers and to train them in how to use the system effectively. For example, if the order entry-system is so efficiency-optimized that it doesn't keep track of sales credits, the sales people will fight using it.

Further, the reduced order processing cycle will reduce the time to deliver products only if additional initiatives are pursued to coordinate the order entry system with the order fulfillment system. Some classic cases where this didn't happen were the late deliveries of Hershey's Halloween candy and Toys'R'Us' Christmas toys.

Such additional initiatives need to be added to the Results Chain. Besides increasing its realism, this also identifies additional success-critical stakeholders (sales people and order fulfillment people) who need to be involved in the system definition and development process. The expanded Results Chain involves these stakeholders not just in a stovepipe software project to satisfy some requirements, but in a program of related software and non-software initiatives focused on value-producing end results.

2.2 Stakeholder Value Proposition Elicitation and Reconciliation

It would be convenient if all the success-critical stakeholders had readily expressible and compatible value propositions that could easily be turned into a set of objectives for each initiative and for the overall program of initiatives. "Readily expressible" is often unachievable because the specifics of stakeholders' value propositions tend to be emergent through experience rather than obtainable through surveys. In such cases, synthetic-experience techniques such as prototypes, scenarios, and stories can accelerate elicitation.

Readily compatible stakeholder value propositions can be achievable in situations of long-term stakeholder mutual understanding and trust. However, in new situations, just considering the most frequent value propositions or success models of the most frequent project stakeholders (users, acquirers, developers, maintainers) shows that these are frequently in conflict and must be reconciled.

Stakeholders' Success Model Clashes

For example, Figure 2 shows a "spider web" of the most frequent "model clashes" among these stakeholders' success models.

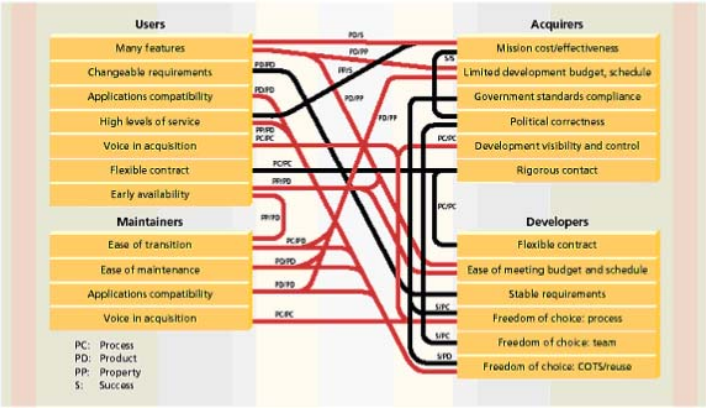

Figure 2. Model-Clash Spiderweb diagram. The red lines show model clashes from the MasterNet system.

The left- and right-hand sides of Figure 2 show these most-frequent success models. For example, users want many features, freedom to redefine the feature set at any time, compatibility between the new system and their existing systems, and so on.

However, the Spiderweb diagram shows that these user success models can clash with other stakeholders' success models. For example, the users' "many features" success model clashes with the acquirers' "limited development budget and schedule" success model, and with the developer's success model, "ease of meeting budget and schedule."

The developer has a success model, "freedom of choice: COTS/reuse" that can often resolve budget and schedule problems. But the developer's choice of COTS or reused components may be incompatible with the users' and maintainers' other applications, causing two further model clashes. Further, the developer's reused software may not be easy to maintain, causing an additional model clash with the maintainers.

The red lines in Figure 2 show the results of one of the analyses performed in constructing and refining the major model clash relationships. It determined the major model clashes in the Bank of America Master Net development, one of several major project failures analyzed. Further explanations are in [8].

Given the goodly number of model clashes in Figure 2 (and there are potentially many more), the task of reconciling them may appear formidable. However, there are several effective approaches for stakeholder value proposition reconciliation, such as:

- Expectations management. Often, just becoming aware of the number of potential stakeholder value proposition conflicts that need to be resolved will cause stakeholders to relax their less-critical levels of desire. Other techniques such as well-calibrated cost models and "simplifier and complicator" lists help stakeholders better understand which of their desired capabilities are infeasible with respect to budget, schedule, and technology constraints.
- Visualization and tradeoff-analysis techniques. Frequently, prototypes, scenarios, and estimation models enable stakeholders to obtain a better mutual understanding of which aspects of an application are most important and achievable.
- Prioritization. Having stakeholders rank-order or categorize the relative priorities of their desired capabilities will help determine which combination of capabilities will best satisfy stakeholders' most critical needs within available resource constraints. Various techniques such as pairwise comparison and scale-of-10 ratings of relative importance and difficulty are helpful aids to prioritization.
- Groupware. Some of those prioritization aids are available in groupware tools, along with collaboration-oriented support for brainstorming, discussion, and win-win negotiation of conflict situations.
- Business case analysis. Determining which capabilities provide the best return-on-investment can help stakeholders prioritize and reconcile their value propositions. Business case analysis is discussed in more detail next.

2.3 Business Case Analysis

In its simplest form, business case analysis involves determining the relative financial costs, benefits, and return on investment (ROI) across a system's life-cycle as:

$$ROI = \frac{Benefits - Costs}{Costs} \qquad (1)$$

Since costs and benefits may occur at different times, the business case analysis will usually discount future cash flows based on likely rates of interest, so that all of cash flows are referenced to a single point in time (usually the present, as in Present Value).

One can then compare two decision options A and B in terms of their ROI profiles versus time. In Figure 3, for example, Option A's ROI becomes positive sooner than Option B's ROI, but its longer-term ROI is lower. The stakeholders `can then decide whether the longer wait for a higher ROI in Option B is preferable to the shorter wait for a lower ROI in Option A. Option Rapid-B illustrates why stakeholders are interested in rapid application development. If Rapid-B can be developed in half the time,

it will be much preferable to either of Options A or original-B.

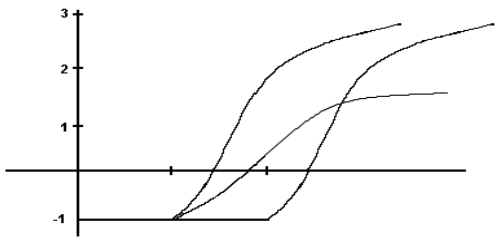

Figure 3. Example of Business Case Analysis Results

Unquantifiable Benefits, Uncertainties, and Risk

Two additional factors may be important in business case analysis. One involves unquantifiable benefits; the other involves uncertainties and risk.

In some cases, Option A might be preferred to Option B or even Rapid-B if it provided additional benefits that may be difficult to quantify, such as controllability, political benefits, or stakeholder good will. These can sometimes be addressed by such techniques as multiple-criterion decision-making or utility functions involving stakeholders' preferences for financial or non-financial returns.

In other cases, the benefit flows in Figure 3 may be predicated on uncertain assumptions. They might assume, for example, that the Option B product will be the first of its kind to enter the marketplace and will capture a large market share. However, if two similar products enter the marketplace first, then the payoff for Option B may be even less than that for Option A.

If the profitability of early competitor marketplace entry can be quantified, it can then be used to determine the relative value of the rapid development Option Rapid-B. This value can then be used to determine the advisability of adopting practices that shorten schedule at some additional cost. An example is pair programming: empirical studies indicate that paired programmers will develop software in 60-70% of the calendar time required for an individual programmer, but thereby requiring 120-140% of the cost of the individual programmer.

If the profitability of early competitor marketplace entry is unknown, this means that making a decision between the cheaper Option B and the faster Option Rapid-B involves considerable uncertainty and risk. It also means that there is a value in performing competitor analysis to determine the probability of early competitor marketplace entry, or of buying information to reduce risk. This kind of value-of-information analysis can be performed via statistical decision theory; a discussion and examples of its applicability to software decision making are provided in [6 Chapters 19-20]. An excellent overall introduction to software business case analysis is [28].

2.4 Continuous Risk and Opportunity Management

Risk analysis and risk management are not just early business case analysis techniques; they pervade the entire information system life cycle. Risk analysis also reintroduces the people factor into economic decision-making. Different people may be more or less risk-averse, and will make different decisions in similar situations, particularly when confronted with an uncertain mix of positive and negative outcomes.

For example, consider a programmer who is given 4 weeks to complete the development of a software module. The programmer is given two choices. One is to develop a new version of the module, which he is sure he can do in 4 weeks. The other is to reuse a previously-developed module, for which there is an 80% chance of finishing in 1 week and a 20% chance of finishing in 6 weeks. The expected duration of this option is (.8)(1) + (.2)(6) = 2 weeks. This represents an expected time savings of 2 weeks and a corresponding savings in expected effort or cost.

Understanding and Addressing People's Utility Functions

In this situation, though, many risk-averse programmers would reject the reuse option. They don't want to be known as people who overrun schedules. Their utility function would assign a much larger negative utility to overrunning the 4-week schedule than the positive utility of finishing ahead of schedule. In terms of expected utility, then, they would prefer the assured 4-week develop-a-new-module approach.

However, their boss may have preferred the reuse option, particularly if she had invested resources in creating the reusable components, and if she could organize the project to compensate for the uncertainties in module delivery schedules (e.g., via modular architectures and daily builds rather than a pre-planned module integration schedule). If so, she could revise the programmers' incentive structure (rewarding reuse independent of actual completion time) in a way that realigned their utility functions and success models to be consistent with hers.

Thus, understanding and addressing people's utility functions becomes a powerful tool in reducing the risk of the overall project's failure—or, from a complementary perspective, in improving the opportunity for the overall project's success. It means that value-based software engineering is not a dry "manage by the numbers" approach, but a highly people-oriented set of practices. And its treatment of uncertainty balances negative risk considerations with positive opportunity considerations. Reconciling stakeholders' utility functions involves essentially the same approaches for stakeholder value proposition elicitation and reconciliation as we discussed in Section 2.2.

Using Risk to Determine "How Much Is Enough"

A current highly-debated issue is the use of plan-driven methods versus use of agile methods such as Extreme Programming, Crystal Methods, Adaptive Software Development, and Scrum [19]. Recent workshop results involving plan-driven and agile methods experts have indicated that hybrid plan-driven/methods are feasible, and that risk analysis can be used to determine how much planning or agility is enough for a given situation.

A central concept in risk management is the Risk Exposure (RE) involved in a given course of action. It is determined by accessing the probability of loss P(L) involved in a course of action and the corresponding size of loss S(L), and computing the

risk exposure as the expected loss: RE=P(L)*S(L). "Loss" can include profits, reputation, quality of life, or other value-related attributes.

Figure 4 shows risk exposure profiles for an example e-services company with a sizable installed base and desire for high assurance; a rapidly changing marketplace and desire for agility and rapid value; and an internationally distributed development team with mix of skill levels and a need for some level of documented plans.

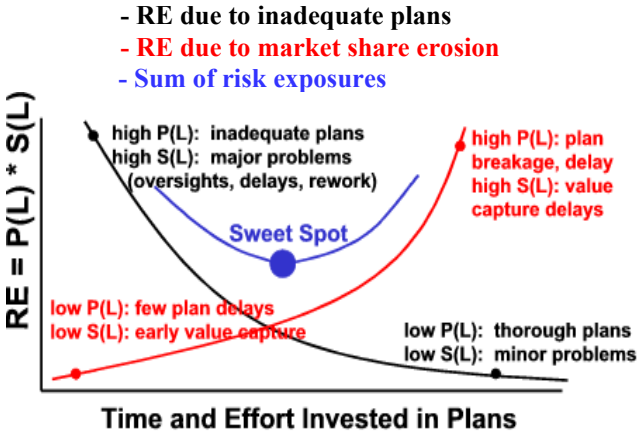

Figure 4. Risk Exposure (RE) Profile: Planning Detail

The black curve in Figure 4 shows the variation in risk exposure (RE) due to inadequate plans, as a function of the level of investment the company puts into its projects' process and product plans. At the left, a minimal investment corresponds to a high probability P(L) that the plans will have loss-causing gaps, ambiguities, and inconsistencies. It also corresponds to a high S(L) that these deficiencies will cause major project oversights, delays, and rework costs. At the right, the more thorough the plans, the less P(L) that plan inadequacies will cause problems, and the smaller the size S(L) of the associated losses.

The red curve in Figure 4 shows the variation in RE due to market share erosion through delays in product introduction. Spending little time in planning will get at least a demo product into the marketplace early, enabling early value capture. Spending too much time in planning will have a high P(L) due both to the planning time spent, and to rapid changes causing delays via plan breakage. It will also cause a high S(L), as the delays will enable others to capture most of the market share.

The blue curve in Figure 4 shows the sum of the risk exposures due to inadequate plans and market share erosion. It shows that very low and very high investments in plans have high overall risk exposures, and that there is a "Sweet Spot" in the middle where overall risk exposure is minimized, indicating "how much planning is enough?" for this company's operating profile.

With the example company situation as a reference point, we can run comparative risk exposure profiles of companies having different risk profiles. For example, Figure 5 shows the comparative RE profile for an e-services company with a small installed base and less need for high assurance, a rapidly changing marketplace, and a collocated team of highly capable and collaborative developers and customers. With this profile, as shown in Figure 5, the major change in risk exposure from Figure 4 is that the size of rework loss from minimal plans is much smaller due to the ability of the team to rapidly replan and refactor, and thus the company's Sweet Spot moves to the left toward agile methods.

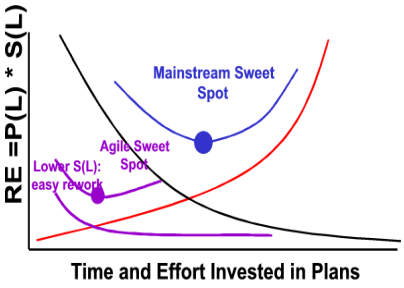

Figure 5. Comparative RE Profile: Agile Home Ground

Figure 6. Comparative RE Profile: Plan-Driven Home Ground

Figure 6 shows the corresponding RE profile for a company in the plan-driven home ground, with a more stable product line of larger, more safety-critical systems. Here, the major difference from Figure 4 is a much higher size of rework loss from minimal plans, and a resulting shift of the company's Sweet Spot toward higher investments in plans. Further discussion of these issues is provided in [7].

Similar analyses have shown that such risk analysis techniques can be used to determine "how much is enough" for other key software engineering levels of activity, such as testing, specification, prototyping, COTS evaluation, formal methods, or documentation.

2.5 Concurrent System and Software Engineering

As we discussed in Section 1, the increasing pace of change in the information technology marketplace is driving organizations toward increasing levels of agility in their software development methods, while their products and services are concurrently becoming more and more software-intensive. These trends also mean that the traditional sequential approach to software development, in which systems engineers determined software requirements and passed them to software engineers for development, is increasingly risky to use.

Increasingly, then, it is much more preferable to have systems engineers and software engineers concurrently engineering the product's or service's operational concept, requirements, architecture, life cycle plans and key sections of code. Concurrent engineering is also preferable when system requirements are more emergent from usage or prototyping than prespecifiable. It is further preferable when the relative costs, benefits, and risks of commercial-off-the-shelf (COTS) software or outsourcing decisions will simultaneously affect requirements, architectures, code, plans, costs, and schedules. It is also essential in determining cost-value tradeoff relationships in developing software product lines [16].

Relevant Process Models

For the future, then, concurrent spiral-type process models will increasingly be preferred over sequential "waterfall"-type

process models. Several are available, such as the Evolutionary Spiral Process [32], the Rational Unified Process (RUP) [29, 21, 24], and the MBASE/CBASE models [12, 9]. Some agile process models such as Lean Software Development and Adaptive Software Development [19] also emphasize concurrent system and software engineering.

An important feature of concurrent process models is that their milestone pass-fail criteria involve demonstrations of consistency and feasibility across a set of concurrently-developed artifacts. For example, Table 1 shows the pass-fail criteria for the anchor point milestones used in MBASE and RUP: Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and Initial Operational Capability (IOC) [12]

LCO	LCA	IOC
For at least one architecture, a system built to that architecture will: • Support the core operational concept • Satisfy the core requirements • Be faithful to the prototype(s) • Be buildable within the budgets and schedules in the plan • Show a viable business case • Have its key stakeholders committed to support the Elaboration Phase (to LCA)	For a specific detailed architecture, a system built to that architecture will: • Support the elaborated operational concept • Satisfy the elaborated requirements • Be faithful to the prototype(s) • Be buildable within the budgets and schedules in the plan • Show a viable business case • Have all major risks resolved or covered by a risk management plan • Have its key stakeholders committed to support the full life cycle	An implemented architecture, an operational system that has: • Realized the operational concept • Implemented the initial operational requirements • Prepared a system operation and support plan • Prepared the initial site(s) in which the system will be deployed for transition • Prepared the users, operators, and maintainers to assume their operational roles

Table 1. LCO, LCA, and IOC Pass/Fail Criteria

These milestones work well as common commitment points across a variety of process model variants because they reflect similar commitment points during one's lifetime. The LCO milestone is the equivalent of getting engaged, and the LCA milestone is the equivalent of getting married. As in life, if you marry your architecture in haste, you and your stakeholders will repent at leisure (if, in Internet time, any leisure time is available). The third anchor point milestone, the Initial Operational Capability (IOC), constitutes an even larger commitment: It is the equivalent of having your first child, with all the associated commitments of care and feeding of a legacy system.

Another important development in this area is the Capability Maturity Model-Integrated (CMMI) [31, 1]. It integrates the previous software-centric Software CMM [25] with CMM's for System Engineering and for Integrated Product and Process Development. The CMMI (and its predecessor iCMM [15]) provides a process maturity assessment and improvement framework, which organizations can use to evolve from sequential to concurrent systems and software engineering approaches, in ways, which emphasize integrated stakeholders teaming and reconciliation of stakeholder value propositions.

2.6 Value-Based Monitoring and Control

A technique often used to implement project monitoring and control functions in the software CMM or the CMMI is Earned Value Management. It works as follows:

1. The project develops a set of tasks necessary for completion, and associated budgets and schedules for each.
2. Each task is assigned an earned value (EV) for its completion, usually its task budget.
3. As the project proceeds, three primary quantities are revised at selected times T:
    a. The Budgeted Cost of Work Scheduled (BCWS): the sum of the EV's of all tasks schedules to be completed by time T.
    b. The Budgeted Cost of Work Performed (BCWP), or project level earned value: the sum of the EV's of all tasks actually completed by time T.
    c. The actual cost of the project through time T.
4. If the BCWP (budgeted cost of work performed) is equal to or greater than the BCWS (budgeted cost of work scheduled), then the project is on or ahead of schedule.
5. If the BCWP is equal to or greater than the project cost, then the project is on or ahead of budget.
6. If the BCWP is significantly less than the BCWS and/or the project cost at the time T, then the project is significantly overrunning its schedule and/or its budget, and corrective action needs to be performed.

The six steps are summarized in the earned value feedback process shown in Figure 7.

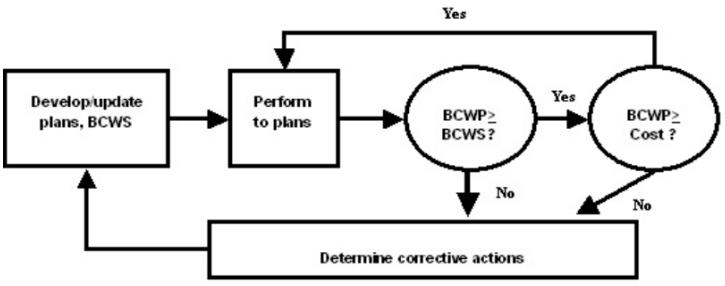

Figure 7. "Earned Value" Feedback Process

The Earned Value Management process is generally good for tracking whether the project is meeting its original plan. However, it becomes difficult to administer if the project's plan changes rapidly. More significantly, it has absolutely nothing to say about the actual value being earned for the organization by the project's results. A project can be tremendously successful with respect to its cost-oriented "earned value," but an absolute disaster in terms of actual organizational value earned. This frequently happens when the resulting product has flaws with respect to user acceptability, operational cost-effectiveness, or timely market entry. Thus, it would be preferable to have techniques which support monitoring and control of the actual value to be earned by the

project's results.

Business-Case and Benefits-Realized Monitoring and Control

A first step is to use the project's business case (discussed in Section 2.3) as a means of monitoring the actual business value of the capabilities to be delivered by the project. This involves continuing update of the business case to reflect changes in business model assumptions, market conditions, organizational priorities, and progress with respect to enabling initiatives. Monitoring the delivered value of undelivered capabilities is difficult; therefore, this approach works best when the project is organized to produce relatively frequent increments of delivered capability.

A related next step is to monitor assumptions and progress with respect to all of the Initiatives and Outcomes involved in the project's Results Chain discussed in Section 2.1 and shown in Figure 1. The suggested monitoring approach in [35] involves coloring in the degree to which Initiatives and Outcomes have been realized. This can be extended to monitor Contributions and validity of Assumptions as well. For example, monitoring the Contribution, "Reduce time to deliver product" in Figure 1 could uncover the problem that speeding up order entry will create more order fulfillment delays unless a complementary order-fulfillment Initiative is established.

The resulting value realization feedback process is shown in Figure 8. With respect to the order-entry example just above, finding out that value was not being realized via reduced delivery times would lead to some corrective action, most likely the establishment of an order-fulfillment speedup Initiative. This would require updates of the overall plans and business case, and new time-phased cost and benefit flows to monitor.

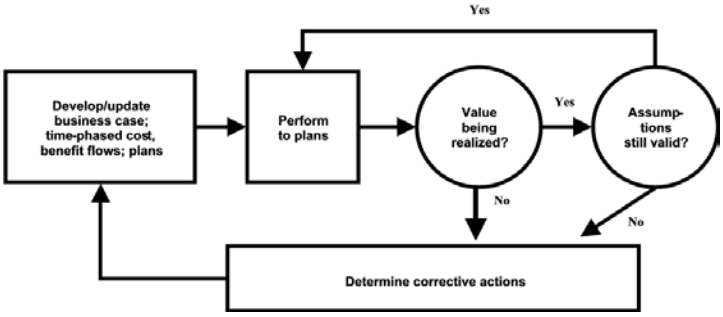

Figure 8. Value Realization Feedback Process

A further option in the value realization feedback process involves adjusting the value function to reflect progress with respect to the product's production function as illustrated in Figure 9. The usual economic production function is an S-shaped curve in which the early "Investment" segment involves development of infrastructure which does not directly generate benefits, but which is necessary for realization of the benefits in the High-payoff and Diminishing-returns segment of the curve. This means that tracking direct benefits realized usually produces pessimistic results during the Investment segment of the curve. One can either manage stakeholders' expectations to accept low early benefit flows (as with the ROI profiles in Figure 3), or use an alternative value function (the dotted line in Figure 9), which ascribes additional indirect value to the early investments in infrastructure. The preferred option will depend on the project's stakeholders and their expectations.

Of course, the actual and potential benefit values realized by each increment of capability need to be monitored and adjusted for changes. For example, a low-cost and user-friendly animated graphics package may increase the net value of animated graphics for certain classes of applications (e.g. education and training).

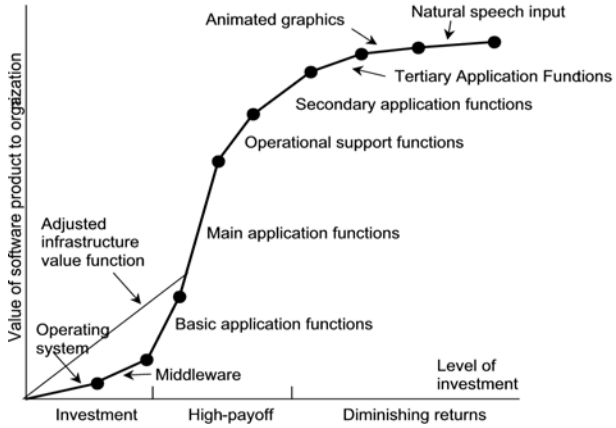

Figure 9. Example Production Function for Software Product Features

Value-Based Monitoring and Control at the Organization Level

The preceding discussion focused on value-based monitoring and control at the individual project level. At least as important is value-based monitoring and control at the organization level. A particularly good approach for integrating both organization-level and project-level value-based monitoring and control is the Experience Factory (EF) and its associated Goal-Question-Metric (GQM) approach [4, 5].

A value-based version of the EF-GQM approach is shown in Figure 10 [9]. Using the benefits realization, stakeholder value proposition elicitation and reconciliation, and business case analysis activities discussed in Sections 2.1 through 2.3, the organization establishes a shared vision, goals, and strategies for its improvement initiatives (the upper left box in Figure 10).

For example, the organization may establish more rapid software development as a high-priority competitive strategy, and set a goal to reduce its projects' software development time by 50 percent. The implementing initiative may then set goals and plans to have each project activity reduce its calendar time by 50 percent.

Once an initial pilot project is selected based on stakeholder commitment and business-case value, its progress is monitored for progress/plan/goal mismatches, as shown at the bottom of Figure 10. While design, code, and test planning may finish in 50 percent less time, integration and test may start showing a 50 percent increase rather than decrease in duration. Analyzing this progress/plan/goal mismatch would determine the root cause to be delays due to inadequate test planning and preparation of test tools, test drivers, and test data. Further, shortening the test plan activity had produced no cycle time savings, as test planning was not on the project's critical path.

The results of this analysis would be fed into the organization's experience base: Future cycle time reduction strategies should focus on reducing the duration of critical path activities, and options for doing this include increasing the thoroughness and duration of noncritical-path activities. Overall then, as shown in the center of Figure 10, the EF analyzes and synthesizes such kinds of experiences, acts as a repository for the experiences, and supplies relevant experience to the organization on demand. The EF packages experience by building informal and formal models and measures of various processes, products, and other forms of knowledge via people, documents, and automated support.

Figure 10. Value-Based Experience Factory Framework

Several useful techniques are available for organizing and managing multi-dimensional improvement strategies. The Balanced Scorecard technique [22] organizes goals, strategies, and initiatives into four perspectives: financial; customer; internal business process; and learning and growth. The BTOPP business system [30, 35] uses five perspectives: business, technology, organization, process, and people. Both are similar; organizations can choose the one that best fits or develop an alternative as appropriate.

2.7 Change as Opportunity

Expending resources to adapt to change is frequently treated as a negative factor to avoid. Software change tracking systems often treat changes as defects in the original requirements. Quality cost systems often treat change adaptations as a quality cost to be minimized. These criteria tend to push projects and organizations toward change-aversion.

Nowadays, changes are continually going on in technology, in the marketplace, in organizations, and in stakeholders' value propositions and priorities. And the rate of change is increasing. Organizations that can adapt to change more rapidly than their competition will succeed better at their mission or in the marketplace. Thus the ability to adapt to change has business value.

And software is the premier technology for adaptation to change. It can be organized to make the cost of changes small as compared to hardware. It can be updated electronically, in ways that preserve continuity of service as the change is being made. Thus, change as opportunity for competitive success is a key economic and architectural driver for software projects and organizations.

Examples of Change as Opportunity

The main sources of change as opportunity come from changes in technology or in the marketplace that open up new opportunities to create value. These are of course other opportunity sources such as changes in legislation, organizational alignments, and international relations.

An excellent example of technology change as opportunity has been the Internet and the Web and their effect on electronic commerce. Organizations that learned early how to capitalize on his technology made significant competitive gains. Other good examples of technology change as opportunity have been agent technology, mobile computing, and the Global Positioning System (GPS).

A good example of marketplace change as opportunity is the existence of GPS and mobile computing in automobiles as an opportunity to provide mobile location-based services. Another is the opportunity to add mobile electronic collect-on-delivery billing and collection systems at the delivery point of rapid-delivery services such as Federal Express and United Parcel Service.

Techniques for Enhancing Adaptability to Change

As documented in Microsoft Secrets [14], the world's leading software business uses a number of techniques for enhancing its adaptability to change. Its synchronize-and-stabilize approach focuses on concurrent evolutionary development, in which each feature team has the flexibility to adapt to change, while buffer periods are built into each increment to enable the teams to synchronize their results. Nightly build techniques also accommodate flexibility in the integration schedule and adaptability to change. Also, Microsoft uses a number of techniques to enhance organizational learning and adaptation, such as customer feedback analysis, project postmortems, and technology-watch and marketplace-watch activities.

Project techniques for enhancing adaptability to change tend to fall into two categories: architecture-based and refactoring-based. Architecture-based techniques focus on identifying the product's most likely sources of change, or evolution requirements, and using information-hiding modularity techniques to hide the sources of change within architectural modules [26]. Then, when the changes come, they can be accommodated within modules rather than causing ripple effects across the entire product. A related technique is schedule-as-independent-variable (SAIV), which uses prioritized requirements as potential sources of change to ensure delivery of the highest-priority requirements within a fixed schedule [10]

Refactoring-based change focuses on keeping the product as simple as possible, and reorganizing the product to accommodate the next set of desired changes. A number of the agile methods discussed in Section 2.4 rely on refactoring to accommodate change, while the plan-driven methods rely on architecture. Which one is more likely to succeed for a given project is largely based on the validity of the Extreme Programming slogan, "You Aren't Going to Need It (YAGNI)." If the evolution requirements are knowable in advance and stable, the architecture-based approach will easily accommodate them, while the YAGNI approach

will incur a steady stream of excess refactorings. On the other hand, if the requirements changes are frequent and highly unpredictable, pre-architected frameworks will continually break, and refactoring simpler designs will be preferable.

Economic Value of Adaptability to Change

Developing a change-anticipatory modular design can be considered as an investment in real options which can be exercised in the future to execute changes which enhance the system's value [2, 3]. More specifically, [34] uses the options-pricing approach to analyze the economic value of Parnas' information-hiding technique to modularization around anticipated sources of change. This approach can also be combined with other economic approaches, such as buying information to reduce the risk of anticipating the wrong set of changes (e.g. via prototypes, user surveys, marketplace watch, or technology watch activities).

Another perspective on the value of adaptability to change comes from studies of complex adaptive systems [23, 20]. These studies show that for various "fitness landscapes" or value functions, one can tune a set of adaptation parameters so that a high-value operational solution will emerge over time via the interaction of a set of adaptive agents. A too-rigid set of adaptation parameters will lead to gridlock; a too-flexible set will lead to chaos. [18] shows numerous parallels between software development and complex adaptive systems, including the value of relatively agile over highly rigorous approaches to software development in domains undergoing rapid change.

3. Getting Started Toward Value-Based Software Engineering

3.1 Going Toward VBSE At the Project or Organization Level

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.

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, these could be both cost savings and increased sales and profits. Construct a chart similar to Figure 3 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 [28] provides further guidelines and case study examples.

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 2.5 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.

3.2 Going Toward VBSE at the National or Global Level

Figure 11 shows a roadmap for making progress toward Value-Based Software Engineering and its benefits on a national or global level [13]. 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.

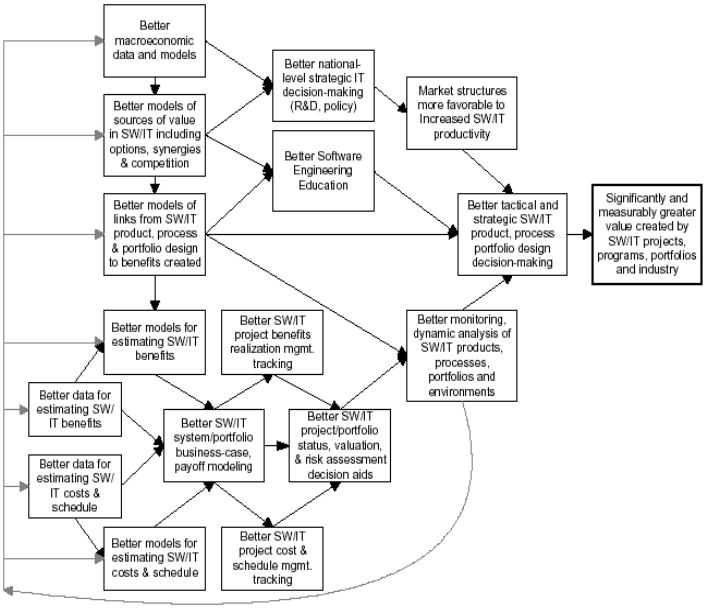

Figure 11. Roadmap for realizing benefits of value-based software engineering

Working backwards from the end objective, the roadmap in Figure 11 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.

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 decisionmaking 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 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 2.6.

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 roadmap are discussed in more detail in [13].

4. Summary and Conclusions

What Are You Getting Paid For?

If you're a professional software engineer, you're getting paid good money for your efforts. What do you think that people ultimately responsible for your paycheck feel that they are paying for? Your immediate bosses may say that you're getting paid to produce designs, code, tests, and so forth. But their sources of support are expecting something different.

The ultimate sponsors of your project are expecting that the project's end result will be to add more value for them than they are paying you and the project team to create it. Their value proposition may be a financial return on investment or an improved public service like health, education, or defense. Or it might be scientific curiosity, a political objective, or pure ego sat-

isfaction.

Why Should You Care?

It used to be that the decisions you made on a software project were pretty much decoupled from the value propositions that established the project. A requirements analysis could establish the requirements for the software, and all you were responsible for was the traceability from your software back to the requirements. But in today's world of rapidly changing information technology, organizations, and marketplaces, the "requirements" tend also to change rapidly, and in ways that require participation of all knowledgeable parties in determining just how a system's definition should change.

As a software engineer, you are both a critically knowledgeable and a critically responsible party in this new system-level process. In particular, just saying, "Oh, I'll do whatever's needed on the same software budget and schedule," is a recipe for disaster both for your career and for your sponsor's value propositions. Thus, traceability to value propositions becomes more important and relevant than traceability to requirements. You need to be able to understand and deal with other stakeholders' value propositions and they with yours. If they want new features, they must be prepared to drop lower-priority features or add budget and schedule.

How Does Value-Based Engineering Help You Do This?

A value-based approach to software engineering helps by providing new perspectives, tools, skills, and success criteria for most of the activities involved in software engineering. In Section 2, we presented seven key elements which are emerging as the foundations for value-based software engineering:

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

These elements fit together into a coherent framework for practicing value-based software engineering as shown in Figure 12.

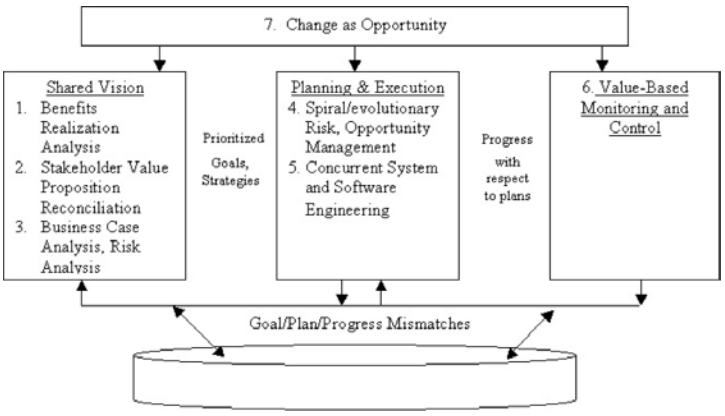

Figure 12. Value-Based Software Engineering Framework

How Can You Get Started Toward Value-Based Software Engineering?

Section 3 provides some initial steps you can take for each of the seven key elements of value-based software engineering. They are fairly compatible, and can be used 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.

Section 3 also provides a roadmap for making progress toward value-based software engineering at a national or global level. Its overall goals are to develop fundamental knowledge and practical techniques that will enable significant increases in the value created by software engineering and its information technology products and services.

Does This Mean That You Need to Reinvent Everything?

Fortunately not. Several approaches have been evolving in this direction in response to the changes in the information technology marketplace. The DMR Group's Benefits Realization Approach and Results Chain have been successfully used in a wide variety of applications [35]. Balanced Scorecard methods have also been successfully applied to software projects and organizations [17]. The Rational Unified Process is organized around the economics of software development, and has emerging extensions addressing business case analysis and business modeling [29, 21, 24].

The Capability Maturity Model-Integrated (CMMI) extends the software CMM to address system-level considerations such as operational concept definition, stakeholder shared vision achievement, and risk management [31, 1]. The spiral model's risk-driven approach has been extended into a value-driven approach called Model-Based (System) Architecting and Software Engineering (MBASE) [11, 12], which is compatible with RUP, CMMI, and organizational approaches such as the Experience Factory and CeBASE Method [9]. Thus, there are ways to evolve from current approaches to increasingly robust value-based approaches.

Also, 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.

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.

References:

[1] D. Ahern, A.Clouse, and R. Turner, CMMI Distilled, Addison Wesley, 2001.

[2] M. Amram and N. Kulatilaka, Real Options, Harvard Business School Press, 1999.

[3] C. Baldwin and K. Clark, Design Rules: The Power of Modularity, MIT Press, 2000.

[4] V. Basili, G. Caldeira, and H. D. Rombach, "The Experience Factory", in J. Marciniak (ed.), Encyclopedia of Software Engineering, Wiley, 1994.

[5] V. Basili, G. Caldeira, and H. D. Rombach, "The Goal Question Metric Approach," in J. Marciniak (ed.), Encyclopedia of Software Engineering, Wiley, 1994.

[6] B. Boehm, Software Engineering Economics, Prentice Hall, 1981.

[7] B. Boehm, "Get Ready for Agile Methods, With Care," Computer, January 2002, pp. 64-69.

[8] B. Boehm, D. Port, and M. Al-Said, "Avoiding the Software Model-Clash Spiderweb," Computer, November 2000, pp. 120-122.

[9] B. Boehm, D. Port, A. Jain, & V. Basili, "Achieving CMMI Level 5 Improvements with MBASE and the CeBASE Method," Cross Talk, May 2002.

[10] B. Boehm, D. Port, L. Huang, and A. W. Brown, "Using the Spiral Model and MBASE to Generate New Acquisition Process Models: SAIV, CAIV, and SCQAIV", Cross Talk, January 2002.

[11] B. Boehm and W. Hansen, "Understanding The Spiral Model as a Tool for Evolutionary Acquisition", Cross Talk, May 2001.

[12] B. Boehm and D. Port, "Balancing Discipline and Flexibility with the Spiral Model and MBASE", Cross Talk, December 2001. See also http://sunset.usc.edu/research/MBASE

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

[14] M. Cusumano and R. Selby, Microsoft Secrets, How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People, The Free Press, 1995.

[15] Federal Aviation Administration, "The Integrated Capability Maturity Model," 1997.

[16] 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.

[17] P. Ferguson et al., "Software Process Improvement Works! (Advanced Information Services, Inc.)," CMU/SEI-99-TR-027, November 1999.

[18] J. Highsmith, Adaptive Software Development, Dorset House, 2000.

[19] J. Highsmith, Agile Software Development Ecosystems, Addison Wesley, 2002.

[20] J. Holland, Emergence: From Chaos to Order, Perseus Books, 1998.

[21] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison Wesley, 1999.

[22] R. Kaplan and D. Norton, The Balanced Scorecard: Translating Strategy into Action, Harvard Business School Press, 1996.

[23] S. Kauffman, At Home in the Universe, Oxford University Press, 1995.

[24] P. Kruchten, The Rational Unified Process, (2nd ed.), Addison Wesley, 2001.

[25] M. Paulk, C. Weber, B. Curtis, and M. Chrissis, The Capability Maturity Model, Addison Wesley, 1994.

[26] D. Parnas, "Designing Software for Ease of Extension and Contraction," IEEE Trans. Software Engr., March 1979, pp. 128-137.

[27] Rational Software Corp., Driving Better Business with Better Software Economics, Cupertino, CA 95014, 2001.

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

[29] W. E. Royce, Software Project Management, Addison-Wesley, 1998.

[30] M. Scott Morton, The Corporation of the 1990s: Information Technology and Organization Transformation, Oxford University Press, 1991.

[31] Software Engineering Institute, Capability Maturity Model Integration (CMMI), Version 1.1., CMU/SEI-2002-TR-012, March 2002.

[32] Software Productivity Consortium, "The Evolutionary Spiral Process," SPC Technical Report, Herndon, VA, 1992.

[33] The Standish Group, CHAOS Report, 1995, www.standishgroup.com

[34] 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.

[35] J. Thorp and DMR, The Information Paradox, McGraw Hill, 1998.

[36] Webster's Collegiate Dictionary, Merriam-Webster, 2002.