I. Introduction

Rapid, sustained advances in computing and communications are now enabling the incorporation of high-speed, low-cost, distributed information processing capabilities into technology components and system of all kinds, at all scales. This trend promises to provide enormous benefits by enabling new functions and improving the performance of existing functions. The potential for value creation is seen to be so great that it is driving information machinery into essentially all serious social, business, and military human-machine systems: appliances, homes, communities, industries, research activities, armies. The results will transform society.

Although hardware production is the catalyst, it is the software that embodies new value-added functions. Software thus takes on a critical new level of economic and social importance. This role is reflected in demand that far outstrips our production capacity [PITAC, 1999], in worldwide expenditures on software now estimated at US$800 billion annually [Boehm & Sullivan, 1999], and in many other aspects of the modern economy.

Yet, as the importance of and our dependence on software grows, there is a new awareness that its production and use are among the most complex and problematical aspects of modern technology development. The symptoms are clear. Large projects fail at an alarming rate, for example. The resources invested in failed projects has been estimated at $85 billion for U.S. business in 1998 alone [Business Week, 1999].

Project, program, and business failures are inevitable, even desirable, in an efficient marketplace. However, problems in the development and use of software destroy value and create risks of losses with unacceptable unpredictability and at an unacceptable rate. Doomed projects consume considerable value before being cancelled. Software costs jump in ways inconsistent with expected risk, as exemplified by the experience of Garlan et al., in what appeared to be a straightforward integration task [Garlan et al., 1995]. Delays lead to lost profits and missed opportunities. Unexpected absences of critical properties make costly systems unusable. Our inability to effectively manage the risk-return characteristics of software is a serious and difficult problem.

In this paper we trace many difficulties in software development to our failure to understand adequately the economics of software development and use, and thus to make software and systems design decisions for products, processes, projects, portfolios that maximize the value created for the resources invested. We discuss how a sophisticated economic perspective on software design promises to significantly improve the productivity of investments in software-intensive systems. We review the state of the art in software economics. We identify important shortcomings in the existing work in view of conditions in the software and information technology sectors today. We then provide a roadmap for future research, and we discuss several current activities in that context.
Software Engineering as a Value-Creation Activity

The core competency of software engineers is in making technical software product and process design decisions. Today, however, there is a disconnect between the decision criteria that tend to guide software engineers and overarching value creation criteria. It is not that technical criteria such as information hiding, abstraction, and the need for mathematical precision are incorrect. On average, they are enormously better than no sound decision criteria. However, software engineers are usually not involved in or often do not understand enterprise-level value creation objectives. The connections between technical parameters and value creation are understood vaguely, if at all. There is rarely any real measurement or analysis of how software engineering investments contribute to value creation. And senior management often does not understand success criteria for software development or how investments at the technical level can contribute fundamentally to value creation. As a result, technical criteria tend to be applied in ways that in general are not connected to, and thus are probably not optimal for, value creation.

Software designers, engineers, and managers must begin to understand and reason systematically and effectively about the connections between software design decisions and value maximization objectives. Understanding the connections will drive decision-makers at all levels to new and better technical criteria, and to making better choices. One important adjustment is that decision-makers will begin to think more strategically. Getting to this point requires that software specialists step out of “Flatland” and away from purely technical criteria that are not linked to outcomes in terms of value added. The first step is to understand that the mismatch between the criteria that are used today and ones more aligned with value creation has several identifiable and remediable causes.

Sources of Mismatch Between Software Decisions and Value Creation

First, we lack adequate frameworks for modeling, measuring and analyzing the connections between technical decisions and value creation. Sullivan et al. have argued, for example, that central concepts in software engineering, such as information hiding and its manifestation in modern software architecture [Parnas 72, Shaw & Garlan 95], the spiral model [Boehm 88], and heuristics on the timing of software design decisions, have yet to be linked adequately to the creation of business value, but that such linkages can and should be made. We are left making decisions today on mostly technical grounds that are not necessarily right for value.

Failing to cancel projects quickly that new information shows are unlikely to succeed is an example of not making a value-optimizing decision. Another of the many consequences of inadequate foundations of software design theory in the realm of value is that conflicts among decision-makers are common, often taking the form of arguments over whose technical criterion is better. E.g., should we use a composition or generator approach? Absent clear connections from the technical domain to value, there is little hope that such debates will converge, or that the decisions best for value will be taken.

Second, the design space within which software designers operate today is inadequate. By the design space we mean the set of technologies with which, and the organizational, regulatory, tax, market and other structures within which, software is developed and used. Designers are unable to make decisions that could, if available,
produce based on their current configurations; but also for options that they have to reconfigure to exploit opportunities that might arise in the future or to exploit synergies between concurrent investments or between current and future investments [Trigeorgis, 1997]. Good strategists understand that maximizing the business value of an enterprise often depends on investing to create real options and synergies. Not for selling books at a profit has Jeff Bezos been named Time Magazine’s 1999 Man of the Year.

The extraordinary valuations that the market currently assigns to some internet companies reflects an assessment of the present value of uncertain future gains, including gains made possible by options that these companies have to exploit future conditions if it becomes favorable to do so. The investment by Amazon.com in an infrastructure and an initial foray into books, for example, created not only a cash flow stream from book sales, but real options to enter other markets. The capability to exercise those options quickly depends in part on the ability to change the software on which the company runs. That ability is supported or not by architecture and other technical properties of their systems.

The “leap-frogging” of Intel and AMD in the race to keep the world’s fastest-clocked microprocessor reflects the value of time in that market. The design processes have to be organized in part to enable effective competition. At a grander scale, when firms are driven to compete in this way the resulting increase in velocity of innovation and product range that occurs has a tremendous impact on technology and the economy.

Microsoft’s independent-feature-based architectural style for many applications, combined with their synchronize-and-stabilize process, creates real options to abandon features late in development to meet time-to-market requirements [Cusumano & Selby, 1995]. In selecting and integrating product and process models, they are clearly doing so in a way that is meant to create value in part in the form of valuable decision flexibility.

Reasoning about how software designers can create value with strategic design, and understanding how and the extent to which our technical design decision criteria embody good value-based strategies, are important topics in software economics. As the Amazon.com example illustrates, in competitive markets one can create value by creating options. In the software realm, architectural and process design are critical in that regard. Investing intelligently in product and process design to create options is thus a dimension of this issue. Sullivan et al. suggest that the value of such options can sometimes be made tangible in economic terms, with varying degrees of confidence, using techniques related to option pricing [Sullivan et al., 1999]. The broader point is that design succeeds only when it responds to sometimes complex economic drivers in ways that create value.

One goal of software economics is thus simply to make these connections clear: to describe important relationships between the technical and the economic dimensions. Just understanding that value is the goal and that it is created by intelligent application of technical means is a start. The links between value and software design are tight but still not understood well enough today. A more aggressive but important goal is to develop a basis for prescribing software product and process decisions based on economic analysis. Beyond mere questions of cost, it is becoming important to be able to address, in a sound and systematic way in technical decision-making, such questions as whether the value of an option is more than the cost of the investment in architecture or process to obtain it?
such decision flexibility. Teisberg presents perhaps the best available analysis of three key valuation techniques: options pricing, utility theory and dynamic discounted cash flow analysis. She explains the assumptions that each of these approaches requires as a condition of applicability, and the advantages and disadvantages of each [Teisberg].

The options pricing approach has two major advantages. First, it relieves the decision-maker of having to forecast cash flows and predict the probabilities of future states of nature. Second, it provides valuations that are based not on these subjective, questionable parameter values, but rather on data from the financial markets. The details are beyond the scope of this paper. In a nutshell, the decision-maker provides the current value of the asset under consideration and the variance in that value over time. That is enough to determine the "cone of uncertainty" in the future value of the asset, rooted at its current value and extending out over time as a function of the volatility.

The variance is obtained by identifying assets in the financial markets that are subject to the same risks as the one in question. A requirement for using this method for valuing decision flexibility is that the risk (variance) in the asset being considered be in the span of the market, i.e., be a function of the risk in identifiable traded assets. The option to port a software system to a platform with an uncertain future might be valued this way, because the risk in the platform is arguably reflected in the behavior of the stock price of the company selling the platform. Because the market has already priced that risk, it has implicitly priced the risk in the asset under consideration, even if it is not traded. We get a market-calibrated price, rather than one based on subjective guesses. Much of the literature is vague on the need for spanning to hold. Amram and Kulatilaka provide a very good introduction to this complex field [Amram & Kulatilaka, 1999].

The work of Baldwin and Clark is especially relevant. They view Parnas's information hiding modules [Parnas 1972] as creating options, which they then value as options (without depending on spanning). On the basis of this insight, they develop a theory of how modularity in design influenced the evolution of the industry structure for computers over the last forty years [Baldwin & Clark, 1999]. Sullivan et al., draw on this material and the material discussed above to sketch of a unified, options-based account of the value in software available through modularity, phased investment process models, and either delaying or accelerating key design decisions [Sullivan et al., 1999].

The Need for Multi-Stakeholder Satisficing

The question of valuation is clearly difficult. Another complicating factor is who is doing the valuing? Is it a company, a philanthropic foundation, a school or university, a government research funding agency? What does that person or entity value? Is it the likelihood of financial gain, the solution of major societal problems, the continuation of a valued culture, or the pleasure of learning or of designing things that work?

Even the question who is often not simple. Any major software design activity involves many participants, each with its own goals and measures of value. Even if they agree on metrics—as in a coalition of profit-making companies cooperating to make a profit—they have conflicting interests in the distribution of gains. Reconciling economic conflicts of this kind is a key success factor in software development. Reconciliation has to be built into software processes, in particular.
Value creation in this world depends on whether you're likely to encounter the other player for another round in the future. If not, defecting is a reasonable strategy. If so, cooperating is reasonable because cooperation is the only strategy likely to produce consistent positive returns over time. However, cooperating naively with a consistent defector, e.g., one who takes your money but provides an unreliable product, is clearly not optimal. Over time, limited retaliatory defection—i.e., tit-for-tat—appears to be a productive strategy. It punishes defections in a limited way to deter future defections but is otherwise willing to cooperate [Axelrod, 1985]. Software design in a world of dynamically assembled profit-making virtual enterprises might well be subject to such economic considerations.

Future Trends Create Additional Challenges

Future trends will continue to exacerbate this situation. The world is changing rapidly in ways that make the situation ever more challenging. While ever smaller, less costly devices penetrate into the technology fabric, the World-Wide Web and Internet have the potential to connect everything with everything. Autonomous agents making deals in cyberspace will create a potential for chaos. Systems of systems, networks of networks, and agents of agents will create huge intellectual control problems.

Further, the economics of software development leave system designers with no choice but to use large commercial-off-the-shelf (COTS) components in their systems. Developers have no way of knowing precisely what is inside of these COTS components, and they usually have very limited or no influence over their evolutionary paths. The PITAC Report accurately states (page 8) that "The IT industry expends the bulk of its resources, both financial and human, in rapidly bringing products to market." The dizzying pace of change continues to increase. Software architecture and COTS decisions are made in great haste. If you marry an IT architecture in haste, you no longer have the opportunity to repent at leisure. Commercial companies with minimal electronic commerce capabilities must now adapt to e-commerce or die.

Of course, these trends also make this a time of fantastic opportunity. The PITAC Report is "right on" in emphasizing that IT offers us the potential to significantly improve our quality of life by transforming the ways we learn, work, communicate, and carry out commerce, health care, research, and government. Value creation opportunities abound, but the path "from concept to cash" [Thorp, 1998] is becoming ever more treacherous.

A new focus on software economics is needed. We now discuss the history and the current status of software economics, with the goal of understanding how it should evolve to be better positioned to address important emerging issues in software design.

C. History and Current Status of Software Economics

Software economics can be considered as a branch of information economics, a subfield of economics which began to receive serious treatment in the 1960's. Its original subjects were such topics as the economics of advertising and search for best prices [Stigler, 1961], the economics of investments in research and development [Arrow,
functions as articulated in Simon’s The Sciences of the Artificial [Simon, 1988] have been incorporated in software engineering approaches such as Participatory Design, Joint Application Design, and stakeholder win-win requirements engineering [Boehm & Ross, 1989; Carmel et al., 1993].

D. Shortcomings that Need to be Addressed

Currently, our ability to reason about software cost is considerably stronger than our ability to reason about software benefits, about such benefit sources as development cycle time, delivered quality, synergies among concurrent and sequential projects, and real options, including strategic opportunities. The trends toward software-based systems discussed above make it clear that the ability to reason about both costs and benefits, sometimes in sophisticated terms, and under such difficulties as uncertainty, incomplete information, and competition, will be a critical success factor for future enterprises.

A good example is Rapid Application Development (RAD). As discussed above, the US PITAC Report [PITAC, 1999] accurately states that the information technology (IT) industry focuses on rapidly bringing products to market. However, most software cost and schedule estimation models are calibrated to a minimal cost strategy. Each has an estimation model similar to a classic schedule estimation rule of thumb:

\[ \text{Calendar Months} = 3 \times \sqrt[3]{\text{Person-Months}}. \]

Thus, if one has a 27 person-month project, the most cost-efficient schedule would be \(3 \times \sqrt[3]{27} = 9\) months, with an average staff size of 3 people. However, this model captures only the direct cost of the resources required to develop the project. It completely fails to account for the opportunity cost of delay in shipping a product into a competitive marketplace, which, today, is often decisive.

Sullivan et al. [Sullivan et al., 1991] have explained that a product can be viewed as a real option on a market, like an option on a stock. Shipping the product to market is the analog of the decision the exercise the option. The entry of a competitor into a market, taking away a share of the cash flow stream that could otherwise be exploited, is the analog of a sharp, discrete drop in the stock price, i.e., a dividend. It is known that for stocks that do not pay dividends, waiting is optimal; but waiting to own a stock that pays dividends (or to enter a market that is subject to competition) incurs an opportunity cost: only the owner of the stock (market) gets the dividend. Thus, dividends (threats of competitive entry) create incentives to exercise early. Here we have a rigorous economic explanation for time-to-market pressure. Understanding such issues is critical to optimal software design decision making, where design decisions include such decisions as that to “ship code.”

If time-to-market is critical, a solution more attractive than that suggested by the rule of thumb above would involve an average of 5.2 people for 5.2 months, or even 6 people for 4.5 months. The earlier work assumes a non-competitive environment, reflecting its orientation to government contracts and classical batch-processing business systems. The recent COCOMO II model [Boehm et al., 2000] has an emerging extension called CORADMO to support reasoning about rapid schedules for smaller projects.
value creation is the goal, value itself can be a complex and subtle quantity. In particular, it is not equivalent to benefits realized, or even benefits realized net of cost, in general.

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

A. Making Decisions that are Better for Value Creation

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

Design decision-making depends in turn on a set of other advances. First, the design space within which designers operate needs to be sufficiently rich. To some extent, the design space is determined by the technology market structure: what firms exist and what they produce. That structure, in turn, 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 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, both to enable better education, and to enable better decision-making in any given situation. An improved understanding of these links depends in turn on developing better models of the sources of value that are available to be exploited by software designers in the first place (e.g., that in uncertain markets, real options can acquire significant value).

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

Fourth, dynamic monitoring and control mechanisms are needed to better guide decision-makers through the design space in search of value added over time. These mechanisms have to be based on models of links between technical design and value and on system-specific models and databases that capture system status, valuation, risk, and so on: not solely as functions of endogenous parameters, such as software development cost drivers, but also of any relevant exogenous parameters, such as the price of memory, competitor behavior, macroeconomic conditions, etc. These system-specific models are based on better cost and payoff models and estimation and tracking capabilities, at the center of which is a business-case model for a given project, program or portfolio. We now discuss some of the central elements of this roadmap in more detail.
Subsequent experience has shown that hypothesis to have been wildly incorrect. In particular, it has turned out to be possible to create tremendous value without formal methods. Some early advocates have admitted that and have posed interesting questions about why things turned out this way. One answer is that the assumed links were based on a view of software products as relatively unchanging.

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

D. Links Between Software Economics and Strategic Policy

Understanding technology-to-value links is critical to making smart choices, not only at the tactical project level, but also in strategic policy-making: e.g., in deciding whether to promote certain results as having demonstrated value creation capabilities today, and in selecting research activities having significant potential to achieve long-term, strategic value creation objectives. Whereas software engineering is about making smart choices about the use of software product and process technologies to create value, software engineering research policy is about making smart choices about how to change the software engineering design space so as to enable greater value creation over time.

The question of who decides precisely what form of value research is to seek, and what value the public is getting for its investment in research and development, is a deep question in public policy. Without trying to answer it here we make several observations.

First, that the prevailing definition of the value to be created by public investment in research has changed in significant ways over the last decade. That change is one of the factors that demands that greater attention now be paid to software economics. During the Cold War and prior to the globalization of commerce and the explosion of advanced technology development in the commercial sector, the nation’s R&D investments were driven largely by national security concerns. Value creation meant contributing to that mission. Today, many argue, the concern has shifted to economic competitiveness. R&D investments in pre-competitive technologies are meant to pay off in design space changes that enable industries to create and societies to capture greater economic value.

In the United States, a major strategic emphasis has been put on new public investment in R&D in information technology, with software, broadly construed, a top priority. This emphasis is justified on several grounds. First, society is coming to rely on systems that in turn increasingly depend on unreliable, insecure software. Second, our ability to produce software that is both powerful and easy enough to use to create value added is inadequate. Third, our capacity to produce the amount of software needed by industry is inadequate. There are thus two basic dimensions of value in the calls for new public investments in software R&D: public welfare, and economic prosperity. Realizing value in these dimensions is as much a concern of the public as profit is for shareholders.
E. Better Monitoring & Control for Dynamic Investment Management

Software-intensive systems design generally occurs in a situation of uncertainty and limited knowledge. Designers are confronted with uncertainties and incomplete knowledge of competitor behavior, technology development, properties of products, macro-economic conditions, the status of larger projects within which a given activity is embedded. Conditions change and new information is gained continuously. The benefits that were envisioned at the beginning of such a project often turn out to be not the ones that are ultimately realized, nor are the paths by which such activities progress the ones that were planned. Rather, complex projects take complex, unforeseen paths. The search for value in spaces that are at best only partially known are necessary dynamic if the are to be most effective.

Beyond a better understanding of software design as a decision-making process, a better design space in which to operate, a better understanding of the context-dependent linkages between technical properties and value creation, and better educated decision-makers, software designers need mechanisms to help them navigate complex situations in a manner dynamically responsive to new information and changing conditions. We need models for both the systems being developed and for sophisticated decision processes that support dynamic monitoring and control of complex software development activities. Dynamic management of investment activities in the face of significant uncertainties and gaps in knowledge is critical at levels from the single project to corporate and national software R&D investment policy.

Multiple models of several kinds will be used at once in any complex program. Models will be needed to guide and to support monitoring and control in the areas of product (e.g., architecture, verification), process (e.g., overall lifecycle, property (e.g., dependability), costs (e.g., for staff, materials, overhead), risk (e.g., lawsuits, liability judgements, failure due to technical or managerial difficulties), opportunities (e.g., to improve a product, to extend new markets or other sources of value, or to follow with a synergistic new function), major programs (e.g., the dependencies among projects that determine ultimate success), corporate or national portfolios (constituent projects and how they support strategic objectives), uncertainty (e.g., project risks within programs and co-variance properties), markets (resources, needs, competition), etc. Models at all of these levels are relevant to technical software design decision-making. Product architectural design decisions, for example, are critical to determining strategic opportunities and in mitigating technical and other risks. Such models and associated dynamic decision processes should be developed, integrated into software design activities, and related to our existing software design decision criteria. To enable the use of such models in practice, tool and environment support will often be needed.

IV. Improving Software Economics Within an Enterprise

The lower portion of the roadmap in Figure 1 summarizes a closed-loop feedback process for improving software economics within an enterprise. It involves using better data to produce better estimates of the likely costs and benefits involved in creating, sustaining, and employing a portfolio of software and information technology assets. These estimates can be used to initiate a dynamic management process in which progress
The Elusive Nature of Software Estimation Accuracy

In principle, one would expect that an organization could converge uniformly toward perfection in understanding its software applications and accurately estimating their cost, schedule, and quality. However, as the organization better understands its applications, it is also able to develop better software development methods and technology. This is good for productivity and quality, but it makes the previous estimation models somewhat obsolete. This phenomenon is summarized in Figure 2.

As the organization's applications become more precedented, its productivity increases and its estimation error decreases. However, at some point, its domain knowledge will be sufficient to develop and apply reusable components. These will enable a significant new boost in productivity, but will also increase estimation error until the estimation models have enough data to be recalibrated to the new situation. As indicated in Figure 2, a similar scenario plays itself out as increased domain understanding enables the use of commercial-off-the-shelf (COTS) components and very high level languages (VHLL). A further estimation challenge arises when the organization becomes sufficiently mature to develop systems of systems which may have evolved within different domains.

Modeling Benefits and Value

We were careful not to put any units on the “productivity” scale in Figure 2. Measuring software productivity has been a difficult and controversial topic for a long time. Great debates have been held on whether source lines of code or function points are better for measuring productivity per person-month. Basically, if your organization

Figure 2. Productivity and Estimation Accuracy Trends
Order to delivery time is an important buying criterion

**ASSUMPTION**

**INITIATIVE**
- Implement a new order entry system
- Reduce time to process order

**OUTCOME**
- Reduced order processing cycle (intermediate outcome)
- Increased sales
- Reduce time to deliver product

This framework is valuable not only for evaluating the net value or return on investment of alternative initiatives, but also in tracking the progress both in delivering systems and contributions, and in satisfying the assumptions and realizing desired value. We will return to the Benefits Realization Approach in Section C below.

Modeling Value: Relating Benefits to Costs

In some cases, where benefits are measured in terms of cost avoidance and the situation is not highly dynamic, one can effectively apply net present value techniques. A good example in the software domain deals with investments in software product lines and reusable components. Several useful models of software reuse economics have been developed, including effects of present value [Cruickshank & Gaffney, 1993] and also reusable component half-life [Malan & Wentzel, 1993]. An excellent compendium of economic factors in software reuse is [Poulin, 1997].

Even with software reuse, however, the primary value realized may not be in cost avoidance but rather in reduced time to market, in which case the value model must account for the differential benefit flows of earlier or later market penetration. Some organizations in established competitive marketplaces (e.g., telecommunications products) have quite sophisticated (and generally proprietary) models of the sensitivity of market share to time of market introduction. In other domains, such as entrepreneurial...
needs are additionally exemplified such as functionality, reliability, maintainability, usability, cycle time, predictability, timeliness, and accuracy. It also emphasizes traceability to not only to requirements but also to business objectives, customer discussions, and market surveys.

Again, these are quite advanced in their focus on customer needs and business objectives, but their primary focus remains on tracking and managing the execution of the project rather than on the value it will presumably deliver. Concepts such as a business case which validates the economic feasibility of the project, and which serves as a basis for tracking the continuing validity of assumptions underlying the project’s business case are not explicitly mentioned. In practice, the usual "earned value" system used to track progress vs. expenditures uses the budgets for the project tasks to be performed as the value to be earned, rather than the expected business value associated with the product’s operational implementation.

In the context of our previous discussions of value creation via information technology, the current normative tracking and managing practices as exemplified by the CMM’s leave open a number of opportunities for improvement. These include improvements in the nature of the achievements to be monitored and controlled, and improvements in the nature of the corrective actions to be performed in case of shortfalls in the projected achievements.

Improvements in the nature of the achievements to be monitored and controlled have been discussed in the context of dynamic investment management in Section III.E, and will be discussed further in Section V. A particularly attractive initial improvement to address is the application of business-value concepts to traditional earned value systems. One approach would be to use the project's business case analysis as the basis of accumulating projected business value, rather than the current measure of success in terms of task-achievement based value.

Improvements in the nature of corrective actions can involve reorganization of the project's process and the system's architecture to best support adaptation to dynamic business objectives. If an entrepreneurial startup's primary objective, for example, is to demonstrate a competitive agent-based electronic commerce system at COMDEX in 9 months, the driving constraint is the inflexible date of COMDEX in 9 months.

An appropriate reorganization of the process and architecture involves using a Schedule as Independent Variable (SAIV) process model, in which product features are dropped in order to meet the 9-month schedule. This requires two additional key steps. One is to continuously update the priorities of the features, so that no time is lost in deciding what to drop. The other is to organize the architecture to make it easy to drop the lower-priority features. Attempting to succeed at SAIV without having provided these necessary preconditions for successful corrective action has compromised the success of a number of entrepreneurial startups.

C. Design for Lifecycle Value

Developing a software system or portfolio of systems is an ongoing activity of design decision-making that extends across multiple organizational and product granularity levels and through time. The software economics viewpoint on this activity
viewpoint leads to the consideration of complex sources of value, and the means by which value creation can be improved. By exploiting modern finance concepts, software engineers can develop a better understanding of such issues as the following:

- the present value of future payoffs that might or might not be attained
- the value of new information that reduces uncertainty about unknown states of nature
- the value of risk reduction with all other factors, including expected payoffs, the same
- the present value of options whose payoffs depend on how exogenous uncertainties are resolved, including options to enter new markets if market conditions become favorable; to make follow-on investments if exploratory phases produce favorable results; to abandon money-losing investments for abandonment payoffs; to ship a product early or just to be able to make a credible threat of doing so
- how desired risk-return characteristics can be attained through diversified portfolios of assets, provided that the assets have somewhat independent returns
- the non-linear-in-size value of networks [Shapiro & Varian, 1999]
- the opportunity cost of investing early in the face of uncertainty, and of investing late in the face of possible drops in asset values, as might result from competitive entry into a market

If these concepts are to be exploited by software engineers, then it is important to relate them to terms and decision criteria that software engineers understand. Important software engineering decision-making heuristics include the following:

- information hiding modularity
- architecture first development
- incremental software development
- always having a working system
- risk-based spiral development models
- the value of delaying design decisions
- components and product-line architectures

Modularity and architecture, in particular, have strategic value in establishing valuable options: to improve a system one part at a time as new or revised parts are delivered; to create variants to address new or changed markets; and to develop variants quickly. Phased project structures, as promoted most especially by the spiral development model and its variants, create valuable options in the form of intermediate decision points. It’s far less costly to abandon a relationship before becoming engaged than after, before getting married than after, before having children than after. The value maximizing decision is to cut one’s losses early as soon as it is clear that a project is not going to succeed. A structure within which that is both possible and legitimized in a value-enhancing structure.
Two such efforts are Model-Based (System) Architecting and Software Engineering (MBASE) [Boehm & Port, 1999] and the “benefits realization” approach of Thorp and DMR Consulting [Thorp, 1998]. The focus of each is on achieving the conjunction of conditions necessary for defined benefits to be realized. We discuss each approach in turn. We compare and contrast them with each other. Then we put them in the broader context of active investment management. We then discuss what some of the challenges might be in reorganizing existing enterprises to follow such approaches.

**MBASE/USC**

The Model based System Architecting and Software Engineering approach [Boehm & Port, 1998] is driven by the view that a narrow focus on technical aspects of system architecture is far from enough to promote successful outcomes. Instead, the approach advocates a holistic treatment of all of the key issues, in four basic dimensions, that must be addressed for success to occur.

The *product* dimension addresses issues such as domain model, architecture, requirements, and code. The *process* dimension involves tasks, activities, milestones. The *property* dimension involves cost, schedule, and performance properties. The *success* dimension addresses what each stakeholder needs, e.g., in terms of business cases, satisfaction of legal requirements, or in less formal terms. The basic idea is that in each of the dimensions, activities and expectations are guided by models, and that these models must be mutually consistent for a project to succeed.

The central axiom of the MBASE approach is that success depends on the contributions of multiple self-interested parties—stakeholders—and that for the required set of contributions to materialize, all stakeholders must be satisfied that the process will satisfy their success criteria. It becomes critical to understand the success models of each success-critical stakeholder and to manage the activity to sustain each stakeholder’s expectation of success. Recognizing conflicts among success models, reconciling them, and managing expectations emerge as key challenges.

More generally, the approach recognizes that one of the key sources of problems in complex projects is in the misalignment or incompatibility of models. The approach thus emphasizes the need to bring the models into harmony so that they reinforce each other. Stakeholder success models along with application domain models drive choices of property, process and product models. The approach provides guidance for identifying and resolving conflicts, and heuristics for promoting the coherence of the multiple models. As clashes among success criteria are resolved, for example, the consistent set of criteria are embodied in product models, e.g., in the system specification.

The next important concept is that the elaboration of a harmonious set of models cannot occur either sequentially or in a single “big bang.” Things change and people cannot foresee all issues that will arise in a complex development effort. The MBASE approach thus emphasizes the use of an approach in which models are elaborated and made ever more mutually reinforcing over time in an iterative fashion. The Win-Win spiral model is employed in an attempt to ensure that the ultimate objectives—each stakeholder’s desire for the satisfaction of its success criteria—is continually accounted for in the evolving set of models. As conditions evolve over time, success criteria,
can absorb change at a finite rate. Thorpe also emphasizes that the parameters in these dimensions will themselves change over time.

The Thorpe/DMR benefits realization approach is based on several premises: first, technology alone is insufficient to produce benefits; second, early visions of benefits are rarely realized, but rather the benefits that are ultimately achieved are based on a dynamic, somewhat unpredictable process of benefits pursuit over time; and third, realizing benefits requires a continuous process of envisioning the benefits desired, implementing, and dynamically adjusting course in light of new information.

The first point drives a change in perspective from traditional engineering, based on a cycle of design-develop-test-deliver, to an end-to-end view of technology-intensive development: from concept to cash. In other words, the approach is holistic and focused on realizing value, rather than focusing just on technical issues, per se. It recognizes that the strategic value of information technology is increasing exponentially, but that as its impacts cut deeper and ever more broadly across organizations, and as the costs of information technology continue to drop, the fraction of the cost of IT-enabled change attributable to IT itself continues to dwindle.

At an investment structuring level, the Thorpe/DMR approach is based on several concepts. The first is that the emphasis has to shift from stand-alone projects, for which the goal is the delivery of a discrete capability (e.g., a software system), to multi-project programs, for which the goal is to produce benefits at the organizational level. A project might deliver a computer, but it takes a program to put a person on the moon. The second idea is that an organization should take a disciplined approach to designing portfolios of programs in order to realize given benefits while managing risk and return. The programs in these portfolios, as suggested above, will combine investments in information technology with other initiatives as necessary to achieve defined business objectives. The third core concept is that a full cycle governance approach be taken to managing the portfolio and its constituent programs. This idea combines active, full life cycle management with the notion of a phased and incremental investment approach based on well defined stage gates. Stage gates are major evaluation and decision points for programs that enable reevaluation of changes in the state of a program and its environment, and decisions about whether to change course, e.g., to abandon, redirect, or reinforce a program.

In terms of management, per se, the Thorpe/DMR approach requires what he calls activist accountability, relevant measurement, and proactive management of change. Activist accountability means that a senior business manager owns each program and is accountable for the realization of its benefits. An important corollary is that the information technology group cannot reasonably be held accountable for realizing business benefits of IT-enabled change. It does not have the ability or authority and scope of control to coordinate all of the elements needed to realize benefits at the organizational level; so it must not be given such responsibility, either. The measurement issue stresses that the tracking of performance parameters related not just to costs but to benefits is critical. Costs are tangible; performance measures that reflect the realization of benefits are harder to find. The third issue is proactive management of change. This issue goes to the question of implementing a benefits realization approach or any of the major IT-enabled changes within it, in an organization not already set up for such changes. Senior
and dynamically managed investments in multiple, interrelated areas with specific value creation objectives. In addition to direct payoffs and realized benefits, value is created through the clever design and exploitation of synergies among concurrent and serial investments and through the creation and dynamic management of real options. It is now understood that in business, for example, much of the present value of some enterprises is in the form of options that they hold to profit from possible future opportunities [Myers, 1977], and that a fundamental strategy for increasing value is therefore to invest in both the creation of such options, and in the capabilities that are necessary to exercise them if it becomes favorable to do so. Capabilities extend all the way to culture. An enterprise can do better than its competitors, for example, if its culture accepts project abandonment in light of unfavorable new information as a positive-value action.

Beyond the management activities of any individual enterprise, there is the larger question of how best to improve the design space in which they operate. Traditional software engineering research plays a vital role, of course, in providing new and better technologies and processes; but innovative research on what is necessary to achieve a transformation of the industry structure is also needed. Understanding how to create more liquid markets in software risk is one potentially important step.

Within the enterprise, there are interesting questions about how best to allocate scarce resources at the “micro” level. Many activities contribute to the development of a software product. They all compete for resources. The value added is in part a complex function of the distribution of resources over those activities. Getting a sense of what that function is overall, and for particular life-cycle activity, such as verification, is important. Of course, the function will vary from one context to another, just as cost functions do.

To support all of these activities, new models and associated analysis methods for reasoning about the value of investments in software technical assets, including design aspects of software processes, products, projects, programs and portfolios. Supporting tools and environments will then be needed to make the models useful for engineers in practice.

Finally, understanding how to integrate such advances into industrial software development processes would be an essential to the realization of the benefits enable by investments in such research. It is not enough to produce disembodied models. Recent advances represented in the MBASE and DMR approaches provide indications that we are now at a stage where we can envision achieving such an integration effectively over time.

VII. Education

Finally, we will note, briefly, that the research cannot have the desired value-enhancing impact without a coordinated investment in education. At a minimum, a traditional course in engineering economics would help to give software engineers a rudimentary background in finance.

Other engineering disciplines consider such an engineering economics course to be an essential part of their respective bodies of knowledge. Recent attempts to define bodies of knowledge and related curricula for software engineering, by contrast, under-emphasize the engineering economics of software. For example, it might be embedded in
and for suggesting some of the important issues that we discuss in this paper. This work was supported in part by the National Science Foundation under grant CCR-9804078.

Mary Shaw emphasized the need to consider the issue of markets in risk for software. Somesh Jha has discussed with us his investigations into a portfolio analysis approach to allocating resources to activities in software projects. Elizabeth Teisberg has provided valuable feedback to Sullivan on the topic of real options.

Bibliography


