10

Software Design as an Investment Activity: A Real Options Perspective[*]

Kevin Sullivan, Prasad Chalasani, Somesh Jha and Vibha Sazawal

University of Virginia; HBK Investments
Carnegie Mellon University; University of Washington

Many important software design guidelines remain without adequate theoretical or conceptual foundations. Important but inadequately formulated concepts include information hiding (Parnas, 1972 and 1985) and its modern version in software architecture (Shaw and Garlan, 1996); risk-reducing spiral processes models and other phased project structures (Boehm, 1988); and guidelines on the timing of commitments to product and process design decisions (Habermann, Flon and Cooprider, 1976). These ideas, which are so central to software design and engineering, remain in their current formulations more akin to folk wisdom than science.

In practice, the lack of scientific or mathematical foundations for software product and process design significantly impedes software development. First, it is hard to teach and understand software design. The body of knowledge of the field appears as informal, unrelated rules of thumb, rather than as principles based on an underlying theory. What is the connection between information hiding and spiral development models, for example? Second, it is hard to justify trade-offs and other key decisions in design. Debates about economically important design issues end up as arguments about whose heuristics are better, with little or no theory to help resolve such debates. Third, the lack of an adequate conceptual model makes it hard to recognise flaws in or boundaries of applicability of informal heuristics, and thus to correct or extend them.

In this chapter, we outline an integrated view of software design as being largely a process of decision-making under uncertainty and incomplete knowledge, including threats of competitive entry. We then appeal to

*This work was supported by the National Science Foundation (USA) under grants to Kevin Sullivan numbered CCR-9502029, CCR-9506779 and CCR-9804078 decision theory and finance for mathematical foundations for key design guidelines. We take utility maximisation, generally, and wealth creation, in particular, as the basic objective of the software design decision-making process in a business context, and we try to explain how we might develop an account of the value-enhancing aspects of a number of software design guidelines in terms of mathematical utility and finance theories.

In particular, we outline a theory of software design based on the idea that part of the value of a typical software product, process or project is in the form of embedded options. These real options provide design decision-makers with valuable flexibility to change products and plans as uncertainties are resolved over time. We apply some basic ideas emerging from the study of real options to try to establish the plausibility of using such concepts to develop a better account of important software design ideas in three areas:

- ❑ timing of important design decisions;
- ❑ designing the modular structure of a system (information hiding); and
- ❑ the use of phased or iterative project structures (the spiral software development model).

We are less concerned here with the use of arbitrage techniques to develop market-based valuations for real options than in developing a case for an options-based rethinking of key ideas in software design and engineering. In some cases, arbitrage pricing is possible, but it depends on knowledge of current asset values and complete markets: that there are traded assets whose risks are closely related to those in the given options. The first step for software engineering is to understand the nature, role and value of options. The second is to develop pricing tools. Here we take the first step.

ECONOMICS-DRIVEN SOFTWARE DESIGN AND ENGINEERING

Before plunging into our discussion of the creation and management of options in software design, we take a step back to address the notion of software design and engineering as a value-driven activity. We put this view in context with traditional software engineering thought, which focuses more on structure and technical perfection than value added. Then we give a brief overview of current software engineering economics.

We take the view that the software design and engineering activity is one of investing valuable resources under uncertainty with the goal of maximising value added (Sullivan, 1996). It is possible to adopt a complex view of value. One could characterise software design as a multi-objective optimisation activity in which one trades safety for performance, or in which one satisfies multiple stakeholders (Boehm, 1989) for example. In this chapter we take a narrow view: that value is measured in terms of market value added to the design firm.

Although this view might have shortcomings, it is a plausible basis for developing a theory of software design for modern business environments.

The view approximates to much industrial practice, embodying the core axiom of corporate finance theory: the objective of the management of a publicly held firm is to make decisions that maximise owners' wealth (Brealey and Myers, 1996). Second, seeing engineering design as a value-seeking activity is not new. Baldwin and Clark state that: "Designers see and seek value in new designs" (Baldwin and Clark, 1999). Third, some of the most successful software companies make wealth the explicit goal. "Everybody in a business unit has exactly the same ... job description, and that is to ship products. Your job is not to write code, your job is not to test, your job is not to write specs. Your job is to ship products ... You're trying not to write code. If we could make all this money by not writing code, we'd do it" (Cusumano and Selby, 1995, p. 45). Fourth, some prominent managers, such as Strassmann, now bemoan software design that fails to focus on value added:

> The proper measurement of the success of software investments requires more than compliance with models of technical perfection. Technical excellence of programming code is highly desirable, but clearly insufficient. Nowadays, the most profitable and popular software in the world is notorious for its bugs and glitches. Economic utility and independent measures of customer satisfaction must be the ultimate arbiters of all judgement about the utility of software (Strassmann, 1997, p. 147).

From structure to value added as desideratum

Our critique of current software design thinking is that no strong links are made from the use of technical design concepts to the goal of economic value added. The design concepts that we discuss are formulated, not in terms of value added, but in purely structural terms. How should a designer go about structuring a product, project or development process?

Consider information hiding. The guideline promotes designs in which aspects of a system that are likely to vary over time or across a product line are "hidden" within modules that can be accessed only through stable interfaces (Parnas, 1972, 1976 and 1985). The concern with software architecture largely addresses the same issue, with a clear emphasis on structure. The important book of Shaw and Garlan on software architecture begins, "As the size and complexity of software systems increase, the design and specification of overall system structure become more significant issues than the choice of algorithms and data structures ..." (Shaw and Garlan, 1996, p. 1). This statement is true, without a doubt. The problem in the field is that no serious attempt is made to characterise the link between structural design decisions and value added.

Process criteria tend to be formulated in structural terms, as well. They define stages of design, their ordering and transition criteria (Boehm, 1988). The waterfall model, in which design begins with requirements analysis and then proceeds mechanically through specification, modular decomposition, module implementation, module integration, system testing and deployment, pays little regard to value added. The spiral model is at least implicitly value-based in that a project "get[s] started by a hypothesis that a particular operational mission (or set of missions) could be improved by a software effort" (Boehm, 1988). We see this as a hypothesis that such an effort is expected to add value. Nevertheless, value remains implicit. No clear link is made in the formulation of the spiral model to a clear notion of value maximisation.

In timing of design investments, we also rely on informal rules of thumb. One such rule says that a designer should delay decisions for as long as possible to maximise the information available when the decision is made. Later in this chapter we show that this heuristic is not correct, in general. We need a principled basis for reasoning about the timing of design investment decisions in order to have a better founded approach to managing complex software projects. More generally, we need a sound basis for reasoning about the conditions under which important heuristics are valid.

Structure and flexibility in software design

The structure of a software product, process or project influences its economic value in critical and often decisive ways. The reason is that structure largely determines the flexibility that the designer has to make changes in a product or development process over time; and in the face of changing and uncertain future conditions, flexibility can have great value. It can provide both protection against downside risk and exposure to upside opportunity. The ability to cancel a project early in the face of unfavourable new information provides downside protection. The ability to make such decisions more effectively promises substantial economic payoffs (Boehm and Sullivan, 1999). The flexibility to adapt a product to a new market or to select the best of a set of compatible components for use in a given design, by contrast, provides upside potential.

By contrast, the "year 2000 problem" illustrates the enormous cost incurred when the flexibility to make needed changes is missing due to an inadequate design structure. The problem is not a result of designers using only two digits to represent dates, as commonly believed. Rather, it is that, in each case, designers failed to *hide* this design decision within a module that is accessed using a single date manipulation interface. If that had been done, the two-digit representation used privately within the module could be changed at low cost, without impact on the remainder of the system. Unfortunately, the lack of modularity in date management code in many programs led to a situation in which many parts of a system came to depend on the decision to use a two-digit representation. When that representation had to change, entire systems had to be scrubbed by hand to find far-flung dependencies on the assumption of two-digit dates.

Uncertainties in software design

Our goal, then, is to develop a better economic account of the value of structure and its exploitation in software products and process. We seek to provide software designers with intellectual tools, and ultimately prescriptive guidance, for understanding when to invest in flexibility and how to exploit it effectively.

Good software designers and software design concepts recognise that uncertainty and incomplete knowledge are critical issues. Uncertainties that have been addressed by researchers and that are reflected in well-known decision-making guidelines include the cost and schedule required to develop a product; aspects of a system that are likely to change in the future or across a product line; the presence of faults in a product and thus the risk of operational failures, and technical risks in building complex software systems. Other uncertainties have been addressed less well, including those about future technology and standards developments; user needs and the value that users are likely to ascribe to products and features; the impact of process changes on cost and scheduling estimates, and others.

A good designer sizes up relevant uncertainties; designs products and projects to provide flexibility to handle them, to the extent that such investments are expected to maximise the value of the project; and continually evaluates the flow of information into the project and takes decisions, within the bounds of flexibility afforded by the product or project, to track the value maximisation goal. Exogenous uncertainties, such as the risk that a standard on which a project is betting will not be accepted by the market, are resolved only by waiting for external processes to reveal themselves. Endogenous uncertainties, such as the feasibility of building a solution on a given platform, are resolved through active inquiry, eg, experimental prototypes.

A key phrase is *within the bounds of flexibility afforded by the project or product*. Here is the link back to structure. The freedom to take actions at a reasonable cost is often missing unless a product or process was designed for it from the outset. It is thus not surprising that much of software design thinking today is about how to design for flexibility. Information hiding design criteria and software architecture to a considerable extent focus on the use of modularity to provide flexibility in design. Similarly, the spiral software development process model provides flexibility in at least two important dimensions. First, it imposes a phased structure on a project, where the goal of each phase is to reduce a key uncertainty facing the project, with decisions about whether or how to invest in subsequent phases based on information from earlier phases. Second, within each phase it stresses the development of alternatives, creating an option to pick the most promising one.

Having designed for flexibility, the designer is in the position of having to manage it. A key issue is knowing when to take an action that is available, but not mandatory, as a result of the project structure. Under threat of competitive entry, for example, when should a designer decide to take a product to market even though it still has bugs due to incomplete testing?

In the face of increasing upgrade costs because of a decaying modular structure, when should a designer invest in restructuring the system to reduce costs? Does it make sense always to wait until forced? When a designer has flexibility to invest or wait, the question is how to time the decision to maximise the project value.

Traditional software engineering economics

The software designer clearly faces two difficult questions concerning the economics of flexibility. First, when should the designer invest to create flexibility that can be exploited later by means of a follow-on investment, if necessary (eg, by designing a module interface to hide a data representation)? Second, given a degree of flexibility to make such a follow-on investment, what strategy should the designer use to decide whether – and if so when – to make it (eg, in changing the data representation hidden within a module).

In deciding whether or not to invest in flexibility, the designer has to weigh its cost against its value to make the value-maximising decision. The problem that the designer faces is that the cost is usually tangible, but the value hard to grasp. For example, hiding a data format in a module incurs an up-front design cost; but, on the other hand, the *present* value of the flexibility so obtained is elusive. The value is in the ability to respond to uncertain future conditions, if necessary. The payoff occurs in the future, contingent on uncertain future conditions. For example, the flexibility obtained by hiding two-digit date codes pays off only if the system is used after 2000.

How can the designer make the intangible present value of flexibility tangible so that the real value can be compared with the real, tangible cost? Software designers generally lack the means to do this. It is thus perhaps not surprising that designers tend to overlook the value of flexibility. Without a sound basis for reasoning about its value, tangible cost often trumps a hard to quantify but real economic benefit.

Before turning to the options approach for making the value of flexibility tangible, we discuss current work in software engineering economics and the difficulties that it would have with this issue. Can we use existing work to get a handle on the value of flexibility? We believe that the basic answer is *no*.

Boehm (1981) pioneered the use of economics to reason about software. The most influential work was on cost and schedule estimation. Boehm also developed the idea of using finance concepts to reason about software design decisions. He developed the idea of using statistical decision theory, in particular, to reason about the value of investments in risk-reducing software prototypes. However, in most early work and the best work since, economic approaches to software design appeal to the concept of *static* net present value (NPV) as a mechanism for estimating value (Boehm, 1981 and Favaro, 1996). A fundamental problem is that static NPV does not capture the value of flexibility under uncertainty.

The static NPV of an investment opportunity is computed by discounting a projected cashflow stream for time and risk. Uncertainty can be factored in by taking a probability-weighted average over a set of projections for a range of scenarios. The static NPV of a possible project is taken to be the amount by which the value of the firm would be changed if the investment were make. The associated investment decision rule is that the investment should be made if the NPV is positive, otherwise not, since making such an investment increases the expected value of the firm by the NPV amount (Brealey & Myers, 1996).

It is now widely accepted in the finance community that static NPV does not capture the value of managerial or design flexibility and that it thus tends to underestimate the value of flexible assets when flexibility is important because of prevailing uncertainty. The problem is that the NPV technique values assets as if they were passively held, having terms that cannot be changed by the owner no matter how well or poorly the future turns out. Real assets are not as passively held. To the extent that an asset is flexible, its owner can intervene actively to optimise value as the uncertain future is revealed. Averaging cashflow streams over all possible futures ignores the value of the flexibility to avoid negative outcomes and to leverage positive ones, eg, by abandoning a project if markets dwindle (Brealey and Myers, 1996; Dixit and Pindyck, 1994; Kester, 1984; Trigeorgis, 1995; Trigeorgis, 1996). The bottom line is that static NPV is generally not a good measure for valuing software because it tends to overlook one of the key sources of value in software assets: the flexibility to adapt to newly revealed and changing information, markets and circumstances.

DYNAMIC APPROACHES TO ACCOUNTING FOR THE VALUE OF FLEXIBILITY

A number of alternative valuation techniques are now available that can help to make the value of flexibility tangible. Teisberg surveys the conditions for applicability and relative merits and difficulties involved in three approaches: dynamic discounted cashflow analysis, decision analysis (utility theory) and real options (Teisberg, 1995). Each technique is appropriate in some conditions and not others; and each has its strengths and weaknesses.

In this chapter, we focus primarily on the concept of real options. The basic idea is to treat flexibility in capital investment decision-making as an option and to value it as such. An option is an asset that provides its owner the right without a symmetric obligation to make an investment decision under given terms for a period of time into the future ending with an *expiration* date. If conditions favourable to investing arise, the owner can *exercise* the option by investing the *strike price* defined by the option. A *call* option gives the right to acquire an asset of uncertain future value for the strike price. A *put* option provides the right to sell an asset at that price. An American option provides the right to exercise at any time up to expiration.

A European option provides the right to make a decision only on the expiration date. A *financial* option is an option on a financial asset, such as a share of stock. A *real* option is an option on a non-financial (real) asset, such as a parcel of land or a new product design.

The notion that options value must be considered in capital investment is now widely accepted. Speaking on the value of flexibility to defer making investment decisions until more information is available, for example, Ross concludes that,

> ... when evaluating investments, optionality is ubiquitous and unavoidable. If modern finance is to have a practical and salutary impact on investment decision-making, it is now obliged to treat all major investment decisions as option pricing problems ("Uses, Abuses and Alternatives to the Net-Present-Value Rule," 1995, p. 101, quoted by Flatto, 1996).

The contribution of this chapter is to view a software product, process or project as a portfolio of assets, some part of the value of which is in the form of real options, and to interpret some critical software design concepts as having implicit options-based rationales. Real options in software provide the designer with flexibility to respond dynamically to information as it emerges over time by making an ongoing sequence of investment decisions. Investing in flexibility is seen as buying options, and exploiting flexibility as exercising them. This view is not new in the real options literature, but it is quite unusual in software design and engineering.

Deciding when to invest in creating flexibility and then exploiting it are transformed into decisions about both when to buy options and how to exercise them to maximise the expected present value of a project or product. This view helps because it reduces the problem of reasoning about the value of flexibility to one of reasoning about the value of options; and much work has been done over several decades to develop techniques for computing options values, including the groundbreaking work of Black, Merton and Scholes, for which a recent Nobel Prize was awarded.

Our primary concern is with understanding options in software design, and only secondarily with the techniques by which values can be computed. Some work in real options assumes that options are valued using arbitrage-based techniques related to those of Black, Scholes and Merton. Arbitrage techniques can only be used when strong assumptions hold. They will not hold for some, perhaps many, software design decisions. We take a somewhat broader view of option pricing based on event trees and dynamic programming. In practice, a range of techniques will be useful in valuing options embedded in software projects, products and processes.

The rest of this chapter elaborates the idea that it is revealing to think about software design and design guidelines in options terms. Panel 1 introduces financial options concepts. In the next section we move from these concepts to those of real options. Next, we discuss one approach to valuing options in a simple but general setting based on event trees. The five sections thereafter then sketch links between options concepts and software design. In the first, we address the timing of design decisions. In the second, we derive qualitative design guidelines from a sensitivity analysis of option values to changes in key parameters. Third, we discuss an options interpretation of information hiding in modular software architectures. Fourth, we outline an options-based interpretation of the spiral model. Finally, we return briefly to the issue of timing, where we discuss the role of options in decisions about time to market under threat of competitive entry, and the engineering tradeoffs that are appropriate in such circumstances. We touch on related work, discuss practical and theoretical difficulties using an options framework to reason about software design, and finally summarise, outline future work and conclude.

BASIC REAL OPTIONS CONCEPTS

Real options theory addresses the problem that investment valuation based on static discounted cashflow tends to overlook the value of decision flexibility. The person who lacks an understanding of the role, value and characteristics of options is not only unable to justify investments in assets that would add value but where a significant part of the value is in the form of decision flexibility. Perhaps more importantly, that person lacks a critical set of intellectual tools for reasoning about strategy and value creation in an uncertain world.

The real options field opened in 1977 when the economist Myers noted that, "part of the value of a firm is accounted for by the present value of options to make further investments on possibly favourable terms" ("Determinants of Corporate Borrowing", 1977, p. 148). Myers saw that, all else equal, a firm that is in a position to exploit potentially lucrative opportunities, eg, by dint of early strategic investments, is worth more than the firm that is not. He also saw that that such assets took the form of *options*, and that options pricing techniques might be used to measure their value, thereby making them tangible.

Today the options Myers saw are called growth options (Kester, 1984, and Myers, 1977). Many other real options are now recognised. Types of real options that have been analysed include the following, Dixit and Pindyck (1994) and Trigeorgis (1996):

❑ Option to defer decisions to invest until optimal to do so; Dixit and Pindyck (1994), Madj and Pindyck (1987) and Myers (1977).

❑ Option to default early on a project that is structured in phases or abandon a project for its salvage value; Geske (1979), McDonald and Siegel (1986) and Myers and Madj (1990).

❑ Option to expand or contract production if favourable or unfavourable conditions emerge; Pindyck (1988).

PANEL 1

BASIC FINANCIAL OPTIONS CONCEPTS

The first advantage of formulating uncertainty-related software design concepts in terms of options is that the theory provides a framework for reasoning about the value of decision flexibility under uncertainty. Understanding even the basic behaviours of options under varying conditions provides intellectual tools that we believe can help designers to reason about the value of flexibility in their products and processes. In this panel, we provide very basic background to financial options.

Underlying random process: The potential payoff of an option is largely determined at each point in time by the value, at that time, of an underlying random process. For example, the payoff of a call option on a stock is largely determined by the price of the stock, which can be modelled as a random process. If the stock price is higher than the strike price, the payoff is the price of the stock minus the strike price paid to buy it. When the underlying value is greater than the strike price, we say that the option is *in the money*. When the value is less than the strike price, the option is *out of the money*.

Non-linear payoff: A fundamental characteristic of an option is that its potential payoff is non-linear in the value of the underlying random process. Consider a stock option. As the stock price rises above the option strike price, the option payoff rises with the stock price. However, no rational investor would ever choose to exercise an out-of-the-money option, since doing so would require paying more for the stock than it sold for in the market. Thus, the potential payoff for all values of the underlying process less than the strike price is zero. The graph of the function is flat up to the strike price and then it rises directly with the asset value thereafter.

Options have value: An option has value because it gives its owner the flexibility to decide in the future whether or not to pay the strike price for an asset whose future value is not known today. An option provides a right to decide *after* finding out how things turn out. Such a right has value because owning the option not only has no downside risk (because its minimum payoff is zero), but it also provides exposure to any upside potential. Thus, for example, even an out-of-the-money option has value if it is possible that the asset value will exceed the strike price before the expiration. Thus, for example, the right to buy a stock for US$100 any time in the next three months has value, even if the current stock price is US$70, provided that there is enough variance in the stock price over time and enough time left before expiration to create the possibility that the stock price will surpass US$100 before the option expires.

Factors influencing value: The precise value of an option depends on a number of factors. First, it depends on the current value of the underlying asset, since a payoff is that value minus the strike price. Second, for the same reason it depends on the strike price. Any drop in the value of the asset would be reflected in the value of the option on that asset. That point is important for understanding the impact of dividend-like phenomena, in which the asset value drops in a sharp step. Third, the option value depends on the underlying uncertainty, with the value increasing with the variance or risk in the underlying random process. Greater variance exposes the option owner to a greater upside but leaves the payoff in the unfavourable case unchanged, making the option more valuable. Fourth, the value of an option depends on the time left to expiration. The value decreases as the time dwindles, because the option provides less flexibility. Another way to view it is that less time to go means less likelihood of a significant upside move in the asset value. Option values also depend to a degree on macro-economic factors, such as the interest rate at which money can be borrowed.

Optimal exercise policy: Although an option being out of the money means that it should not be exercised at the moment, being in the money does not necessarily mean that it should. The reason is that in addition to requiring payment of the strike price, exercising kills the option, which, in general, has additional value. Exercising is an irreversible decision that incurs as a cost the strike plus the additional opportunity cost of not being able to wait any longer to decide whether to invest. It is worthwhile to exercise an option only if the asset obtained is worth at least the strike price plus the option value (Dixit and Pindyck, 1994). It can be optimal to hold rather than exercise an option, even one that's in the money.

The role of dividends: The best time to exercise an option is not always obvious. Options theory addresses optimal policies for timing these investment decisions. A complicating factor is that of dividend payments. A dividend payment on a stock is a cash distribution of part of the asset value represented by the stock. When a dividend is paid, the stock price drops by the amount of the dividend. For a call option on an asset that is not subject to dividend payments, it is known that it is best to wait as long as possible – until just before expiration – to decide whether or not to invest (Dixit and Pindyck, 1994; Hull, 1993). Nothing is lost by waiting, and valuable information might be gained. However, if the asset is subject to dividends, then early exercise is sometimes justified. The reason is that dividends go to the asset owner; holding an option alone forgoes the benefits of owing the asset. For such assets, there is an opportunity cost not only to investing early, but also to waiting. With real options, any discrete drop in the asset value can be thought of as a dividend. A competitor's entry into a market, grabbing a part of it, is an example. Options thinking provides a way to reason about timing in the face of countervailing incentives to hurry or delay investment decisions (Dixit and Pindyck, 1994).

- ❑ Option to switch materials used in a product, or switch to producing another product as supply and demand conditions change; Kulatilaka, 1993.
- ❑ Compound options, or options on options, which can be used to model phased investments, in which investing in one phase buys an option to invest in the next; Triantis and Hodder, 1990.
- ❑ Option to pick one of several risky assets; Stulz, 1982.
- ❑ Interactions among real options have also been explored; Brennan and Schwartz, 1985; Trigeorgis, 1993.

To help make the real option idea more concrete, we consider a simplified version of an example from the literature involving the flexibility to switch the materials used in an electricity generating plant. Suppose a manager has to decide between investing in a less expensive but inflexible power plant that burns only oil, or a more expensive one that has the flexibility to switch between oil and coal (Kulatilaka, 1993). The price of coal is stable, but the future price of oil is uncertain. Is it worthwhile to pay more for flexibility?

Clearly there is a price at which the flexible plant is too costly, despite the risk of a price rise in oil. The value-maximising decision is to invest in the inflexible plant. On the other hand, flexibility has value today because it confers the ability to save future expenses contingent on a more or less probable price rise. So the manager should be willing to pay somewhat more for the flexible plant. How much more? What are the relevant factors that must be considered in making the decision? And on what data can the decision be based?

The manager should assess the expected NPV of each of the two choices and select the better one, factoring in the cost and value of the option created by the flexibility to change fuels for a defined switching cost. The expected value V_I of the inflexible plant is its expected benefits B_I minus its expected costs C_I adjusted for time and risk: $V_I = B_I - C_I$. For the flexible plant the value is its benefits B_F plus the value of the option O_F to switch to coal should future oil prices be unfavourable, minus its costs: $V_F = B_F + O_F - C_F$. The best choice is the flexible plant if $V_F > V_I$. Assuming that the benefits are the same when the two plants burn oil, ie, $B_F = B_I$, then we find that the flexible plant is best when the value option exceeds its cost, $O_F > C_F - C_I$. The flexible plant is best when the option is worth more than its cost.

That is pretty simple. The difficulty is in evaluating O_F. What is flexibility worth? What factors influence its value and how are they related? How do you reason properly quantitatively, or even just in a rigorous qualitative way, about the value of flexibility?

One must clearly give up static NPV as a valuation measure and use a dynamic technique, such as decision analysis, dynamic NPV, or options pricing. The first insight behind real options theory is that flexibility in real assets is analogous to a financial option, such as a call option on a stock, in that it gives the decision-maker the right, without a corresponding obligation, to make an investment decision in the future. The second insight was that techniques related to those for pricing financial options might be used to estimate real options values quantitatively.

In the preceding example, the owner of the flexible plant has an option to pay a certain amount (a strike price) to reconfigure the plant from oil to coal. In return, the manager acquires the cashflow stream comprising the cost savings accrued by the decision to switch. When the price of oil is high, that asset is valuable. Even if the price of oil is low today, there is a risk that it will go higher, so the right to claim the cost savings in that case has value. The flexible plant thus comprises a portfolio of valuable assets including an embedded option. If the price of oil goes high enough, then it will be worthwhile to exercise the option. The dynamic nature of decision-making becomes clear. At first the decision-maker has an option to buy an option, then, having bought it, to exercise it.

Having formulated the flexibility as an options problem, a major challenge is to find an appropriate technique for pricing the option, so that the first decision (whether to buy it or not at a given price) can be made rationally. One starting point is the Black–Scholes equations (Black and Scholes, 1973), the seminal result in the field. The approach is based on the concept of arbitrage. Arbitrage-based techniques require knowledge of the current value of the asset in question, and on the option on that asset being *in the span of the market*. This phrase means that market-traded, and thus also market-priced assets, are available that could, in principle, be assembled into a portfolio that tracks the option payoff closely as the underlying random process evolves. Because the portfolio payoff tracks that of the option, the market should value both assets equally. The market has already priced the portfolio; so a market-calibrated option price is obtained. The key benefit of such techniques is that market-based valuations of real options are derived without the subjective probability estimates required in decision analysis (Amram and Kulatilaka, 1999).

The likelihood that oil prices will rise, for example, has already been "calculated" by the information-integrating operations of the market, and is reflected in the prices of oil futures for some period into the future. Moreover, there is a considerable body of historical data for calibrating models of the random process of oil prices. The availability of such data provides an empirical basis for estimating the parameters of options pricing formulas.

If the conditions for using arbitrage-based techniques (in particular, spanning) do not hold, then other methods for estimating values must be used: dynamic programming, for example (Dixit and Pindyck, 1994). Although there are considerable data on movements in the price of oil over time, there might be few or no market data on which to base estimates of the likelihood that a particular well will turn out dry. So, if one has an option on that risky asset, how should it be valued?

Some real options researchers define the real options approach as requiring arbitrage-based pricing, and would assert that an approach that uses anything other than market data is essentially decision analysis or some other technique. Others see that option-pricing techniques can be used even if spanning conditions do not hold, albeit with the need to use subjective estimates of certain parameters. This approach loses the benefit of objective, market-based valuations, but many of the benefits of "options thinking" (to use Amram's phrase) nevertheless are preserved.

In our view, the critical step is to begin to think in terms of options: to view projects and products as portfolios of assets that can be designed to include valuable options, and then to manage such portfolios dynamically as uncertainties are resolved and new information is acquired over time. The next step is to decide how to model different kinds of options for valuation, and to understand how much confidence can be placed in the reliability of valuations, based on the valuation approach used and the adequacy of any empirical data used for setting the model parameters.

Implicit in our view is the idea that even qualitative reasoning with options thinking is important. The intellectual framework is most critical, because it provides a designer with a sound structure for reasoning about decisions that have to be made anyway. Moreover, back-of-the-envelope calculations based on a powerful model can have real value. For example, a quantitative model permits a systematic analysis of the sensitivity of computed value to variations in subjective assumptions or parameter values estimated from empirical data.

FORMALISING UNCERTAINTY, OPTIONS AND VALUATION

In this section we discuss one approach to reasoning about option values. Although some important options in software design will satisfy the required conditions for the use of arbitrage-based pricing techniques, others will not. There will be options on risky assets for which the risks have not been priced directly or indirectly by the market. In other cases, market assessments of the risks will be aggregated in the market prices of assets exposed to several risks, sometimes making it impossible to separate out the data on the specific risk at issue. Such options are essentially not in the span of the market and so cannot be priced using arbitrage-based techniques. Therefore, we present an account of option pricing that does not depend on spanning conditions or arbitrage, but which can be adapted to exploit market data if they are available.

In particular, in this section, we describe basic concepts in options theory, with a focus on the interplay between the value of an option and the optimal exercise strategy. First we define the mathematical notions and notation that we use. Then we show formally how option pricing is related to optimal stopping times. For more basic information on options the reader is referred to any of several excellent texts, including Hull (1993) or Luenberger (1998). Merton (1990) provides an advanced treatment.

Modelling uncertainty with event trees

Uncertainty and the value of flexibility in the face of uncertainty are at the heart of both software design and finance. Options pricing and related techniques are financial applications of general mathematical techniques used to support optimal decision-making under uncertainty. There are both discrete- and continuous-time formulations of the key issues. In this chapter, we employ a very simple, discrete-time formulation. Our goal is to get at the key concepts, not to present a complete mathematical theory.

We model uncertainty about the future value of a single parameter (random variable) with a discrete event tree of finite depth N, with N the maximum number of future time steps that we wish to model (eg, months, years, etc). The root node, at depth 0, represents the present time. The set of nodes at depth k represent possible states of the world at time k. Given a depth k node v, the children of v are the possible next states at time $k + 1$, given that the state at time k is v. If a node w is a descendant of v, we write $v \rightarrow w$.

For $k = 0, 1, ..., N$, a random variable X is a mapping (or function) that associates with each node v, a real number X(v). A random process is a sequence of random variables $\{X_k\}_{k=0}^{N}$, (often referred to briefly as the "process X_k") where for each k, $X_k(v)$ has non-zero values only for nodes at depth k. When we want to refer to the value of a random process X_k at a specific node v, we will often drop the subscript and just write X(v). We define the special random variable $\delta(v)$ to be the depth of v.

As an example, consider tossing a fair coin N times. For each toss there are two outcomes, each equally likely. We represent the uncertainty in outcomes by a simple event tree, called the binomial tree. Each non-leaf node has two children. Each of the 2^N paths represents a sequence of coin-toss outcomes. On any path, for $k = 1, ..., N$, the kth branch is an up-branch if the kth coin-toss lands heads (H), and it is a down-branch if it lands tails (T).

To each branch in an event tree, we associate a probability. The sum of the probabilities of the branches emanating from a node is 1. In the example above, the probability of each branch is 0.5. For any node v, the probability that state v occurs, denoted $\mathbf{P}(v)$, is the product of the probabilities of the branches from the root node to v. If a node v has a branch to node w, we denote its probability by $\mathbf{P}(w \mid v)$. Clearly,

$$\mathbf{P}(w|v) = \mathbf{P}(w)/\mathbf{P}(v)$$

The expectation of a random variable X is the probability-weighted sum of possible outcomes, denoted by $\mathbf{E}(X)$, and defined precisely as

$$\mathbf{E}(X) = \sum_{v} X(v)\mathbf{P}(v) \tag{1}$$

The concept of conditional expectation is important. Imagine we are in some state at time k, eg, at depth k node v. Then the conditional expectation of a random variable X, given that we are at v, is denoted by E(X | v), and is defined as

$$\mathbf{E}(X|v) = \sum_{w:v\to w} X(w)\frac{\mathbf{P}(w)}{\mathbf{P}(v)} = \sum_{w:v\to w} X(w)\mathbf{P}(w|v) \qquad \textbf{(2)}$$

This is just a form of the familiar Baye's rule. Clearly this conditional expectation will in general be different from **E**X, and will depend on the given depth k node v. For instance in the coin-toss tree above, suppose the random variable H_k is the number of heads up to time k, on the path to a specific node in the tree. Then if v is at depth k and m ≥ k, the conditional expectation $\mathbf{E}(H_m | v)$ will be higher if the path to v consists of more heads.

Decision rules

We will use decision rules in this chapter to model the timing of various investment decisions. For instance we might want to decide *when* (ie, at which nodes in the tree) and *how much* to invest in building a software prototype. In general let us assume that at any node we are allowed c possible levels of investment, numbered 1, 2, ..., c. We consider level 0 to represent no investment. A decision rule τ with respect to an event tree T is a mapping from the set of nodes of T to the set {0, 1, ..., c}, with the restriction that on any path of the tree, there is at most one node v with a non-zero value of τ(v). In other words, a decision rule specifies a rule that we can follow as the state of the world changes along the tree. Whenever we are in a state v for which τ(v) ≠ 0, we invest at level τ(v). In the coin-toss binomial tree, an example of a decision-rule would be: "invest at level 1 when the coin has landed heads three times", or more formally: τ(v) = 1 if H(v) = 3, and τ(v) = 0 otherwise. Those familiar with stochastic processes will recognise that decision rules are closely related to *stopping times*.

American call options

One of the simplest kinds of option is an American call option. As we discussed earlier, an American call option on an asset, say a share of stock, is a contract that gives the holder of the contract the right, but not the obligation, to buy a share of the stock at a predetermined price called the strike price, which we will call L, on or before a certain expiration date, T time units in the future. In other words, the option gives its holder the flexibility to decide whether or not to exercise the contract, ie, buy a share of stock for L after uncertainty about its future value has been resolved. Such flexibility has value. When the option is exercised or if it expires without being exercised, it ceases to exist. Exercising an option is thus an irreversible decision. Conversely, we can think about discretionary but irreversible investments as options.

We model opportunities to make discretionary but irreversible investments in real assets as call options. There are other dimensions of flexibility that are modelled by other kinds of options. Although our approach includes the range of options, we will focus primarily on call options in this chapter.

Uncertainty and payoff

To describe an American call option formally, we need a model of the underlying uncertainty. It is typical to model the price of a stock as a depth-N event tree, with N being the time to expiration, and $\{S_k\}$ a random process modelling the uncertain stock price.

Given such a model, we can reason about when, if ever, to exercise an option. It makes no sense to exercise at any node v where the value of the asset is no more than the strike price $S(v) \leq L$. However, if $S(v) > L$, the option holder can but is not obliged to exercise by investing L to obtain the asset. In the case of a share of stock, the holder could then immediately sell the asset in the market for S(v) making a profit of $S(v) - L$. The profit that can be realised by exercising the American call option at time k is thus $\max(S(v) - L, 0)$, which we refer to as the payoff G(v). It is standard to denote $\max\{x, 0\}$ by x^+, so we can write the payoff as the random variable

$$G_k = (S_k - L)^+ \tag{3}$$

In other words, for any node v, the payoff $G(v) = S(v) - L^+$.

Optimal exercise strategy

What is the best strategy for deciding whether to exercise an American call option held at time k? An exercise strategy can be described by a decision rule τ that maps each node to the set $\{0, 1\}$. If we are in state v, we exercise if and only if $\tau(v) = 1$. An example of an exercise strategy is "exercise when the stock price reaches a threshold λ or when the expiration date is reached", which is formalised by the decision rule τ, where $\tau(v) = 1$ if $S(v) \geq \lambda$ or the depth $\delta(v)$ of the node v is N, and $\tau(v) = 0$ otherwise.

In reasoning about payoffs, it is necessary to account for the time value of money: a dollar tomorrow is worth less than one today. To discount future cashflows to the present time, we have to assume that money can be borrowed or lent at a risk-free interest rate of *r*. Thus a dollar lent or borrowed at time k is worth $R = 1 + r$ dollars at time $k + 1$. It is common to refer to R as a discount factor since a dollar at time k, discounted to the present time (ie, time 0) is worth $1/R^k$.

The optimal exercise strategy is the strategy with the highest expected payoff. At a depth k node v the expected value of a strategy τ discounted to time k is denoted V_k^τ. It is computed as follows. At any node w, a descendant of v, if the option is exercised (ie, $\tau(w) = 1$), the payoff is $G(w) = (S(w) - L)^+$, and if it is not exercised the payoff is 0. Thus in general at any node w we can write the payoff as $G(w)\tau(w)$, which is worth $G(w)\tau(w)R^{k-\delta(w)}$ at time k. Therefore the expected value of the strategy τ, discounted to time k, given that we are at a node v, is

$$V_k^\tau(v) = \mathbf{E}(G\tau R^{k-\delta} | v) \tag{4}$$

The rational option holder wants to use the strategy τ that maximises this expectation. We denote this maximum by the random variable V_k:

$$V_k(v) = \max_\tau V_k^\tau(v) \tag{5}$$

V_k is the greatest expected present value at time k realisable over all possible exercise strategies. We take the option value at time k to be precisely V_k. The reasons for this definition will become clear shortly. The point is that we now have a technique for valuing options provided that assets are modelled in this event tree framework.

Since immediate exercise is a valid strategy at any time, the option value $V(v)$ must be at least as large as $(S(v) - L)^+$. A situation in which immediate exercise is not optimal is one in which $(S(v) - L)^+ < V(v)$. In this case, some other strategy will yield a strictly greater expected present value of payoff, under our assumed asset model, so it is beneficial not to exercise but to wait. On the other hand, if the payoff now is equal to the best expected payoff, $(S(v) - L)^+ = V(v)$, nothing is gained by waiting, and it is optimal to exercise immediately. Indeed it can be shown rigorously that the decision rule τ that achieves the maximum in equation (5) above is given by

$$\tau(v) = \begin{cases} 1 & \text{if } (S(v) - L)^+ = V(v) \text{ or } \delta(v) = N \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

We can think of the strike price L as the cost of an asset obtained by exercising the option, since this is the price one must pay to obtain the asset. Similarly, S_k is the benefit from exercising at time k, since this is the price one could obtain by selling the asset, if obtained. It is useful to view the option value V_k as representing the value of the right to choose whether or not to invest. It is often the case that this right is worth more than the payoff that could be obtained by exercising immediately. That is obviously true if the option with time remaining to expiration is out-of-the-money. One would decline to exercise immediately, for a payoff of zero; yet the option would still have value, because it would provide the potential for a future positive payoff contingent on a rise in value of the asset. The statement also can be true if the payoff that one can obtain by exercising immediately is small in relation to the value of the option to wait for a bigger payoff.

When an option is exercised, the option is killed and the flexibility created by the right to decide is lost. An opportunity cost is incurred in addition to the strike price. V_k, the option value, is that opportunity cost. From the discussion above, the optimal strategy is to exercise when $(S_k - L)^+ = V_k$: *Exercise only when the benefit* S_k *equals the direct cost* L *plus the opportunity cost* V_k. This is the viewpoint most useful in this chapter.

In terms of real options and software design, this analysis suggests that if there is some uncertainty about the future, is might not be optimal to invest in an opportunity (eg, to port a computer program to a new platform or to restructure a legacy system) even if doing so would yield a small expected payoff. Delaying preserves a valuable option to make the decision in the future when the expected payoff might be greater.

Dynamic programming algorithm for valuing options

In our formulation, with its finite time horizon, we can compute V_k for all k by a *dynamic programming* procedure (Corman, Leiserson and Rivest, 1990). First observe that $V_N = (S_N - L)^+$. This is clear both from equation (5) and from observing that, since the option expires at time N, there can be no advantage in waiting to decide whether or not to invest. Stepping backward in time in the decision tree, we compute $V_k(v)$ at any depth k node v by

$$V_k(v) = \max\{(S_k(v) - L)^+, \mathbf{E}(V_{k+1}|v)/R\} \qquad \textbf{(7)}$$

In other words, the option value $V_k(v)$ at a depth k node v is the maximum of the immediate payoff $(S_k(v) - L)^+$ and the expected present value of the option value one time step ahead given that we are at v. It can be shown that this backward-recursive formula for V_k and equation (5) are equivalent, regardless of the specific process that the stock price S_k follows.

OPTIONS IN TIMING SOFTWARE DESIGN INVESTMENTS

We have introduced the idea of applying options-based thinking to software design decision-making, and have presented a simple framework for modelling uncertainty and valuing options. The rest of this chapter is concerned with making concrete, through several examples, the link from real options concepts to some key issues in software design. The examples are highly idealised, so that we can focus on the essential concepts. Options models of actual software design decisions are likely to be much more complicated.

The decision to restructure a legacy system

We first present an example that illustrates how real options concepts might be used to reason about when to reorganise a software system to reduce the costs of enhancing and maintaining it. A central objective of the software designer is to select a modular structure for the system that makes it easy to understand and to change over time as corrections are needed and requirements evolve. The choice of modular structure, or architecture, is the key mechanism for achieving ease of understanding and evolution. The designer will try to modularise a system to separate conceptually independent parts so that they can be understood independently, and to ensure that potential changes will have a module-local (and thus inexpensive) impact on the system, rather than a system-wide (and thus costly) impact.

A major problem in software design is that designers cannot always anticipate how a system will have to change. Thus a modular structure that makes one set of potential changes easy by localising their impact might not localise the impacts of changes that turn out to be needed in practice. Such a change incurs a larger up-front cost relative to the functional increment achieved than a change that is facilitated by the existing design structure.

The additional up-front cost of such a change is, however, just the beginning of a long-term cost stream. Making a change that does not fit well into the existing design structure often ends up compromising that structure to some extent. Two modules that previously could be understood independently might end up more tightly coupled, for example, as a result of the need to incorporate into both, different but interconnected aspects of an unexpected enhancement. Over time, the accumulation of such changes lead to ever-increasing disorder in the structure of the system, and thus ever-increasing costs to achieve given increments in function. At some point, a restructuring of the system to "clean up" its degraded modular structure might be the economical choice (Belady & Lehman, 1976).

The decision about when to restructure a system is one that many designers face. It is often a very costly prospect. Much of the many tens of billions of dollars being spent on software in the telecommunications sector each year, for example, is spent on restructuring legacy systems for a new deregulated, network-centric environment.

Not only are such investments costly. They are often also risky in the sense that the payoffs are uncertain, most likely in multiple dimensions. The ability to enter a new market successfully with a restructured system is not necessarily given. Similarly, if the payoff is simply in the form of a reduced cost to make future changes, the stream of changes that will be needed might not be certain. To illustrate options reasoning about these kinds of issues, we take uncertainty over future changes as an example.

Suppose that restructuring will cost US$1,600.[1] The benefit, if any, will come in the form of reduced future costs. If the benefits were certain, the decision would be easy: the designer would compare the present value of the future benefits, represented as a cashflow stream, against the present value of the cost of US$1,600.

The issue is more complex when the payoff depends on uncertain future conditions. Suppose that the demand for future changes is uncertain. For one reason or another, the system will have to be changed either often or hardly at all. Perhaps the uncertain success of a partial replacement system in development would reduce the need to change the existing system.

As a simplification, let us assume that there are two possible futures: one in which many changes are needed, which is thus favourable to an investment in restructuring; and an unfavourable future, in which so few changes are required that it would have been better not to pay to restructure. We represent the possible payoffs as cashflows. Suppose that the payoff in the favourable scenario is 3,300, and just 1,100 otherwise.

The nature of the uncertainty

We model the uncertainty with a one-level event tree, rooted at the present, with the payoff of 1,100 at one leaf and 3,300 at the other. Next, we assign probabilities to each branch. Suppose that at present it is our expert judgement that each outcome has a probability of occurring of 0.5. Finally, we account for the depreciation in the value of money over time by assuming a discount factor of 1.1 per time period – a 10% discount rate. Thus, for example, a cashflow of US$1,100 one period from now is worth US$1,000 today.

In this example we are using subjective estimates rather than market data based on an arbitrage construction to set the probability parameters of the event tree. Some real options theorists would state that we have moved from a real-options-based approach to one more resembling decision analysis. Our concern is first to recognise the options in any given situation and thus the need to use techniques other than static NPV to reason about value. Our second concern is to select the best valuation technique for the given situation. Satisfaction of spanning conditions that permit the use of market-based data in a no-arbitrage construction rather than subjective probability estimates is an enormous advantage, but, as we have noted, in some cases it won't be possible.

In this chapter, we present the concepts in a setting that does not assume that spanning conditions hold. The same kinds of event trees and dynamic programming algorithms can be used for both arbitrage and non-arbitrage approaches. The interpretations of the probabilities and discount rates of the formulations are vastly different in the two cases, with synthetic *probability-like* numbers derived from market data and the risk-free discount rate being used for the arbitrage approach. Amram (1999) provides an excellent introduction. In this work, we must be satisfied to present the outline of a method in a setting based on subjective estimates. A comprehensive approach to software design investment decision-making under uncertainty would have to incorporate multiple valuation techniques, some requiring subjective data, with others based on market data, when available.

Static net present value

The central point is that the traditional static NPV decision strategy does not produce an investment that is optimal for expected value added. Recall that the traditional rule states that the investment should be made if the static NPV is positive. The decision under this rule is to restructure immediately, because the static NPV of the investment is 400 – for a present value added of 400 – as shown in the following computation.

$$\text{NPV} = 0.5(-1{,}600 + 1{,}100/1.1) + 0.5(-1{,}600 + 3{,}300/1.1) = 400$$

Suppose, however, that the designer can wait to invest until more is known about how much the requirements will have to change (eg, whether the related project succeeds). The flexibility to wait is tangibly valuable. The reason is that investing now risks a loss of $-1{,}600 + 1{,}100/1.1 = -600$ should the unfavourable scenario emerge (the related project succeeds and fewer changes are needed). Waiting allows the designer to invest only in the favourable scenario for a profit of $-1{,}600/1.1 + 3{,}300/1.1 = 1546$. Given that the likelihood of a favourable outcome is estimated at 0.5, the expected value of the dynamic strategy $(0.5)(3{,}300/1.1 - 1{,}600/1.1) = 773$, is significantly greater than that of investing today. It's clearly better to wait.

Making the option explicit

Real options thinking suggests that the software designer should view products and projects as portfolios of assets that include valuable options, that options can have significant value, and that their value should be factored into decision-making. To make the analogy between the restructuring decision and options clear, we recast our analysis into an options formalism.

We formulate the cost to restructure as the strike price $L = 1{,}600$ of an American call option on the expected benefits. This asset is worth S_t, the expected value of the profit stream from restructuring at time t. S_t is a random variable. Letting subscripts denote time and discounting to the present, S_1 is either $3{,}300/1.1 = 3{,}000$ or $1{,}100/1.1 = 1{,}000$, each with probability $p = 0.5$, and so $S_0 = 0.5 * (1{,}100/1.1) + 0.5 * (3{,}300/1.1) = 2{,}000$.

Net of cost, the payoff of exercising now is $S_0 - L = 400$. A month from now it is $\max(S_1 - L, 0)$, since the option is not exercised in the unfavourable future. With even odds this is either $3{,}300/1.1 - 1{,}600/1.1 = 1{,}546$, or 0, for an expectation of 773. Recall that the value V_t of an option at any time t is the expected present value of future payoffs under the optimal exercise strategy. In our example, all uncertainty is resolved in the first period, so there are only two strategies: exercise now or in the next period. The value V_0 of our option is thus 773, since this is the expected payoff from the best exercise strategy of the two available at time $t = 0$.

The software designer is in the position of holding an option to invest, and of having to manage that option carefully. A design decision to commit to restructuring must be viewed as a decision to exercise that option. Restructuring today kills the option, which is worth 773, for an expected value added of only 400. Despite the apparent profitability of investing today, it is clearly suboptimal for expected value added to the firm.

Incorporating cashflow streams

In practice, restructuring a system might yield a benefit (savings) each time a change is made. The benefit is thus a cashflow *stream*. Extending the model and analysis to include cashflow streams in multiple time periods is straightforward. The traditional static NPV is computed as the expected present value S_0 discounted to time 0 of the stream of profits B_n, $n \geq 0$ from the investment, with the cashflow in each period discounted in the standard fashion at a rate compounded by the number of periods:

$$S_0 = \sum_{n=0}^{\infty} \mathbf{E}B_n / R^n \tag{8}$$

As we noted above, even if the NPV is positive, it might be better to wait to find how the underlying uncertainty is resolved. The optimal approach is to decide at any time k whether the value of expected profits discounted to time k (ie, S_k) is sufficiently above the direct cost L to offset the additional cost of losing the option. Thus a designer must compare the value of investing now against that of investing at *all* future times. The question is not just whether to invest, but when, if ever.

At any time k, the designer has the right but not the obligation to invest L (the strike price) to receive a stream of profits (the savings) with an expected present value S_k (the asset price). The random variable S_k is the expected benefit of restructuring at time k, ie, the value of the future profit stream resulting from investing at time k discounted to time k.

We computed S_0 above. The payoff from exercising at time 0 is $G_0 = (S_0 - L)^+$, since one would not invest if $S_0 < L$. This is the same equation (3) for the payoff from an American call option on a stock at time 0. S_0 is analogous to the stock price at time 0, hence our choice of notation.

How do we generalise the equation (8) for S_0 to an arbitrary node v in the tree? To compute the benefit S(v) at node v, we proceed as in equation (8), except that we discount the profits to time corresponding to the depth of node v rather than time 0, and we replace the expectation by the conditional expectation. Finally, we only perform the summation from $\delta(v)$ to ∞. Recall that $\delta(v)$ denotes the depth of node v which also corresponds to the time associated with node v. So S(v) is given by the following expression:

$$S(v) = \sum_{w:v \to w} \mathbf{E}\left[B(w)R^{\delta(v)-\delta(w)} \middle| v\right] \tag{9}$$

The expected benefit of restructuring at node v is then

$$G(v) = (S(v) - L)^+ \tag{10}$$

which is the same as equation (3) for the payoff from a call option. The value V_k of this option represents the value of the investment opportunity, which would be lost if we were to exercise at time k. As described above, V_k can be computed for any k using dynamic programming. Also, as mentioned, it is optimal to exercise when the value V_k equals or exceeds the payoff G_k. Thus it is optimal to restructure when $S_k - L \geq V_k$, or $S_k \geq L + V_k$.

Informally, we should exercise our option to restructure when the benefit S_k is at least as much as the sum of the direct cost L and the opportunity cost V_k. What we end up with is an option-based rule for committing to design decisions under uncertainty:

If at any time k, S_k, *the expected value of future profits discounted to time* k *is at least* V_k *more than the direct costs,* L, *then commit to the design decision, otherwise do not.*

Trading evolvability for market share and an option to restructure

We consider another simplified example involving the design of internet agents (Maes, 1994; Mitchell, Caruana, Freitag and McDermott, 1994; Sycara, Decker, Pannu and Williamson, 1996). Agents are specialised autonomous software entities that provide services to people, such as finding inexpensive airline tickets or filtering news. It often makes sense for one agent to use another to accomplish its task. Indeed, the dynamic composition of such service-providing agents executing in the internet is an emerging model for future software systems. We imagine a system of agents in which each agent has a capability, and in which the capabilities of available agents are stored in some kind of capability directory. For the sake of being concrete, consider a financial portfolio management agent that might want to find another agent to get company reports. The agent-based system is expected to evolve primarily by the addition of new agents over time. The designer is considering two designs:

Distributed directory (D). A copy of the directory is wired into the code of each agent. An agent that wants to find another agent consults its local directory at essentially zero cost. This approach does not follow the information hiding design criterion in that a likely kind of change (adding an agent) will have a costly system-wide impact. Whenever an agent is added to the system, the directory code in each agent has to be changed. We denote the total cost of the changes required when an agent is added at time n by the random variable D_n.

Centralised directory (C). There is a *yellow pages* agent that implements the capability directory. All other agents access directory information through an interface with this agent. We let C denote the total cost of initially hard-coding the centralised directory and its associated interfaces. When a new agent is created, only the yellow-page agent needs to be changed. Also, an agent requiring a capability needs to query the yellow-page agent. We let the random variable C_n denote the total query and update costs at time n.

Which approach should the software engineer choose? If starting from scratch, as in a start-up company, there is no flexibility to delay a decision. The decision must be made now. The trade-off might be increased design and runtime costs for a centralised design that reduces future maintenance costs.

Because there is no flexibility to wait, a static NPV valuation approach might be appropriate. We assume that the costs common to both approaches are 0. Thus, the only relevant costs are C and C_n for choice C, and D_n for choice D. We view the problem as one of deciding whether or not to use choice C, and express the costs and benefits relative to choice D. There are two quantities of interest when choice C is compared with D: the direct cost of choice C, ie, the immediate cost of implementing it, which is L = C; and the monthly profit of choice C relative to D in month n for $n \geq 0$: $B_n = D_n - C_n$.

We view the design problem as an investment decision problem: Should L dollars be invested in choice C? Let us assume a discount factor R. Consider the traditional NPV approach. We first compute the expected present value S_0 (discounted to time 0) of the stream of profits B_n, $n \geq 0$ from the investment:

$$S_0 = \sum_{n=0}^{\infty} EB_n / R^n \tag{11}$$

which we refer to as the expected benefit of choice C relative to D at time 0. The NPV of the investment at time 0 is

$$NPV = S_0 - L = S_0 - C$$

The traditional NPV rule states that if the NPV is positive, then the investment should be made, otherwise not. When there is no flexibility, the NPV rule is correct. Thus a decision rule that might be used when there is no design to start with is the following:

If the expected present value of the future profits S_0 *that would flow from choice C exceeds the direct cost C of implementing it, then go ahead and implement choice C, otherwise implement choice D.*

However, the analysis might not be that easy. If there is competition, the optimal strategy might be to invest in a capability that can be brought to market as quickly as possible. This would mitigate the threat that the market share meant to be acquired by the investment will disappear as the result of entry by a competitor. The best strategy might be to get to market quickly, and, if successful, to make a follow-on investment in restructuring to keep long-term costs manageable and to create additional value through greater design flexibility.

If for certain technical reasons the centralised design is harder and will take longer to get running initially, the value-maximising choice might be to sacrifice a design with long-term maintenance benefits for a shot at market share, augmented with a plan for a possible second phase beginning with a restructuring for long-term success. This is an example of a situation in which a heuristic, such as *always use information hiding design techniques*, might conceivably not be valid. Thus it is an example of where options thinking can inform design strategy.

In practice, we suspect that some start-ups pursue such a strategy initially, but either lack a plan for a follow-on phase for restructuring, or fail to execute such a phase. In these cases, initial successes can run into difficulties in the longer term because their software designs cannot sustain long-term correction and enhancement effectively.

Of course, the decision to invest in that second phase is one that can be delayed. So when, if ever, should it be made? Suppose the existing system has the distributed structure (D), and the designer is deciding whether or not to invest in restructuring to switch to choice C. In addition to the cost C of creating the yellow-pages agent, there might be a cost C^s to scrap the distributed design. Each agent must be changed so that it queries the yellow pages instead of its own local directory. Thus the direct cost of choice C is $L = C + C^s$. It is tempting to propose the following strategy (compare it with the previous decision rule):

If the expected present value of future profits S_0 *that would flow from restructuring exceeds the direct cost of restructuring,* L, *then go ahead and restructure, otherwise do not.*

By now we recognise the flaw in this analysis: the designer has the option to wait in hopes of making a better decision in the future. In addition to the direct cost L, there is an opportunity cost that represents the loss of the value of flexibility. At any time k, the value of the expected profit discounted to time k (ie, S_k) must be sufficiently more than the direct cost L to justify switching. The designer should compare the value of investing now (at time 0) versus investing at *all* possible future times.

Numerical calculations

We make the analysis of this decision concrete with a numerical example. Suppose the cost C to restructure is 9,000, and the cost C^s of scrapping the distributed directory structure is 1,000. Thus the total direct cost L of restructuring is $C + C^s = 10{,}000$.

For simplicity, we assume that building the yellow-page agent and scrapping the distributed directories takes 0 time. Each time step in our model represents 1 month. Time n represents the beginning of the nth month, for n = 0, 1, 2, ... We imagine that during the current month, or month 0, several new agents will be created, and that the associated updating cost under the distributed approach is $D_0 = 2{,}000$. We assume that the total query/update cost under the centralised approach is $C_n = 500$ at all time n. Thus if we move to a centralised directory at the beginning of month 0, the monthly profit for month 0 would be

$$B_0 = D_0 - C_0 = 2000 - 500 = 1500$$

Suppose that if our designer's system beats the competition (probability $p = 0.5$) then several new agents will be added each month, starting in month 1. This outcome is favourable for approach C and we will therefore superscript variables under this scenario with the letter f. Suppose that if this event occurs the total maintenance cost under approach D, is $D_n^f = 3{,}000$ for all $n \geq 1$. On the other hand, if the technology does not achieve dominance in the marketplace, few agents will be created, a situation unfavourable for approach C: the cost to switch to a centralised directory will not be paid back by the cost-savings of the information hiding restructuring. Thus, we superscript variables in this scenario by the letter u, for *unfavourable*. We suppose that the corresponding update cost under choice D in this case is much lower, at $D_n^u = 400$ for all $n \geq 1$.

Thus from month 1 onward, the monthly profit from restructuring for information hiding would be

$$B_n^f = D_n^f - C_n = 3000 - 500 = 2500, \quad n \geq 1$$

in the favourable scenario, and

$$B_n^u = D_n^u - C_n = 400 - 500 = -100, \quad n \geq 1$$

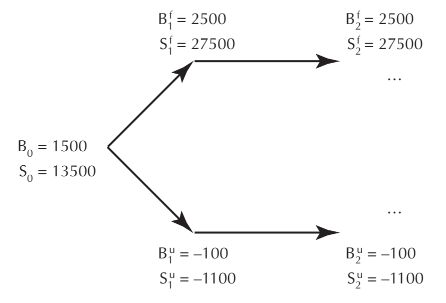
in the unfavourable scenario, each case occurring with probability 0.5. Therefore, for $n \geq 1$,

$$\begin{aligned} \mathbf{E}B_n = \mathbf{E}B_1 &= pB_1^f + (1-p)B_1^u \\ &= 0.5(2500 - 100) = 1200 \end{aligned}$$

Our model is represented by the event tree in Figure 1. There are just two possible paths in this event tree, which we denote by ω_f (favourable) and ω_u (unfavourable). Note that for $k \geq 1$ and $n \geq k$, on the favourable path,

$$\mathbf{E}(B_n|\omega_f) = B_n^f = B_1^f = 2500$$

1. Event tree for the numerical example



and on the unfavourable path,

$$\mathbf{E}(B_n|\omega_u) = B_n^u = B_1^u = -100$$

Assuming a monthly discount factor of R = 1.1, we calculate the benefit of switching at time k. For k = 0, we use equation (8) to compute

$$\begin{aligned} S_0 &= \sum_{n=0}^{\infty} \mathbf{E}(B_n)/R^n \\ &= B_0 + \sum_{n=1}^{\infty} \mathbf{E}(B_1)/R^n \\ &= B_0 + \mathbf{E}(B_1)/(R-1) \\ &= 1500 + 1200/0.1 \\ &= 1500 + 12000 = 13500 \end{aligned} \qquad \textbf{(12)}$$

For k = 1, from equation (9), the benefit from switching at time k in the favourable scenario (or on the favourable path) is

$$\begin{aligned} S_k^f &= \sum_{n=k}^{\infty} \mathbf{E}[B_n|\omega_f)R^{k-n} \\ &= \sum_{n=k}^{\infty} B_1^f R^{k-n} && \textbf{(13)} \\ &= R\sum_{n=0}^{\infty} B_1^f/R^n && \textbf{(14)} \\ &= B_1^f \frac{R}{R-1} && \textbf{(15)} \\ &= 2500(1.1)/0.1 = 27500 \end{aligned}$$

which is bigger than the direct cost L = 10,000. Thus in the favourable scenario the expected benefit of switching is always greater than the cost. Similarly,

$$S_k^u = B_1^u \frac{R}{R-1} = -100(1.1)/0.1 = -1100, \quad k \geq 1 \qquad \textbf{(16)}$$

which is smaller than the direct cost L = 10,000, so in the unfavourable scenario the expected benefit of switching is smaller than the cost. Note that from the definitions of S_0, S_1^f and S_1^u it follows that

$$S_0 = B_0 + (p/R)(S_1^f + S_1^u) \qquad \textbf{(17)}$$

Should the designer invest L = 10,000 and restructure to switch to design C now, or would it be better to wait for a month and invest only if the situation favours a switch? There is no uncertainty after the first month, so these are the only strategies to consider.

We first approach this question by computing the net present value of these two strategies. The NPV of strategy 1 is

$$\mathrm{NPV}(1) = S_0 - L = B_0 + \mathbf{E}(B_1)/(R-1) - L = 13500 - 10000 = 3500 \tag{18}$$

Since the NPV is positive, the NPV rules indicates that the designer should invest. However, let us calculate the NPV of investing under strategy 2: Wait one month, and invest in switching to the yellow-page agent only if the technology succeeds. Since the technology succeeds with probability $p = 0.5$, the net present value of strategy 2 is

$$\begin{aligned}\mathrm{NPV}(2) &= (p/R)(S_1^f - L)\\ &= (p/R)\left(B_1^f \frac{R}{R-1} - L\right) = (0.5/1.1)(27500 - 10000) = 7954\end{aligned} \tag{19}$$

which is significantly greater than the NPV of investing immediately. This shows that it is better to wait.

We now approach the question by computing the value V_k of the investment opportunity at times $k = 0$ and $k = 1$. The payoff if we exercise our option to invest at time k is given by $G_k = (S_k - L)^+$ which is identical to the expression for the payoff from an American call option. Since there is no uncertainty after time 1, it is easy to see that $V_k = G_k = (S_k - L)^+$ for all $k \geq 1$. In particular, if our designer's system succeeds, the option value at time 1 is

$$V_1^f = (S_1^f - L)^+ = (27500 - 10000)^+ = 17500 \tag{20}$$

and if it fails,

$$V_1^u = (S_1^u - L)^+ = (-1100 - 10000)^+ = 0 \tag{21}$$

From the backward recursion in equation (7) we conclude that

$$\begin{aligned}V_0 &= \max\{G_0, (1/R)\mathbf{E}(V_1)\}\\ &= \max\{G_0, (1/R)(pV_1^f + (1-p)V_1^u)\}\\ &= \max\{(S_0 - L)^+, (p/R)(S_1^f - L)^+\} && (22)\\ &= \max\{(B_0 + \mathbf{E}(B_1)/(R-1) - L)^+, (p/R)(B_1^f R/(R-1) - L)^+\} && (23)\\ &= \max\{3500, (1/1.1) \times 0.5 \times (17500 + 0)\}\\ &= \max\{3500, 7954\}\\ &= 7954 && (24)\end{aligned}$$

The values 3,500 and 7,954 in the max above are exactly the NPVs of strategy 1 and strategy 2, respectively. Also, $V_0 > G_0 = 3{,}500$, so it is not optimal to invest right away. However, after 1 month, if the technology succeeds, $V_1 = G_1 = 17{,}500$, so it is then optimal to invest at that time. Thus we have shown in two different ways that strategy 2 is optimal. In general when the uncertainty lasts several periods, computing NPVs for the exponentially many strategies is impractical. The dynamic programming approach from option pricing theory would be a method of choice.

QUALITATIVE DESIGN PRINCIPLES

Software design strategy is based today almost entirely on experience and heuristics. The information hiding principle exhorts the designer to *hide within stable modules design decisions that are likely to change*. The spiral development approach urges the designer to *identify and rank risks and take a phased, iterative approach in which the greatest risks are mitigated first and plans reconsidered after each phase*. One heuristic on timing suggests that all decisions should be delayed for as long as possible.

To the first order, none of these heuristics is based on documented scientific or mathematical models or theories. We see in options theory the potential to begin to understand better why these principle seem to work, to explore conditions under which they are and are not valid, and to develop normative models for software design decision-making based on solid foundations.

The insight is that real options thinking makes key variables explicit in relation to the value of decision flexibility under uncertainty. It thus enables reasoning about the sensitivity of value to the relevant parameters. A designer who understands the mathematical relationships in the theory is better equipped to reason about the critical issues of decision flexibility, uncertainty and value in software design.

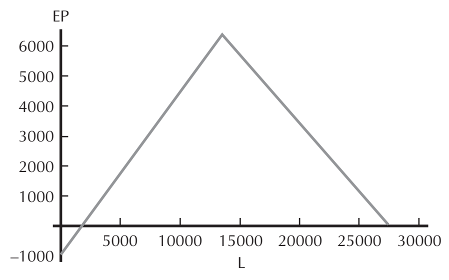
In this section we explore this view, using the example from the previous section to keep the discussion concrete. In particular, we investigate the sensitivity of options values to variations in the cost to invest, uncertainty over benefits, likelihood of a favourable outcome, and uncertainty over cost. As we vary parameters, we continue to assume that the two scenarios retain their favourable/unfavourable status:

$$S_1^f > L > S_1^u$$

Effect of direct cost

From equation (22), we notice that the value V_0 of the option is the maximum of two quantities: the NPV of strategy 1, namely $(S_0 - L)^+$, and the NPV of strategy 2, $(p/R)(S_1^f - L)^+$. Note that since $p/R < 1$, as L decreases, the former NPV increases faster than the latter. Thus, if all other parameters remain the same, there is a critical value for the direct cost L below which it is optimal to restructure immediately, ie, at time 0. Consider the quantity EP = NPV(2) – NPV(1). EP represents the benefit achieved by waiting, or in other words the difference between the value of strategy 2 (where one waits until the next period) versus strategy 1 (where one makes the decision at the initial node). Larger values of EP mean higher benefits of waiting. The graph of EP with respect to the direct cost L is shown in Figure 2.

2. The benefit of waiting, EP, versus the direct cost L



Another way to state this principle is that if the direct cost L is sufficiently low, the cost of waiting (the profit S_0 – L one would forgo) outweighs the value of waiting (the value V_0 of the flexibility to reverse the decision not to invest). Since there is nothing special about time 0, this statement applies at any time. Thus we have an options-based justification for what would otherwise remain an informal software design guideline:

If the cost to effect a software decision is sufficiently low, then the benefit of investing to effect it immediately outweighs the benefit of waiting, so the decision should be made immediately.

Although this design decision-making rule of thumb seems obvious, it contradicts the heuristic, always delay design decisions as long as possible. The plausible reasoning behind that rule is that you should wait until all information is in before investing. The options approach shows that that is wrong in general. This conclusion makes the point that by reducing costs, new technologies, such as restructuring tools (Griswold and Notkin, 1993), can change a situation from one in which it is best to delay to one in which it is optimal to invest.

Effect of uncertainty over benefits B_n

In the numerical example of the previous section, the two possible values of B_n for $n \geq 1$ were $B_n^f = 2{,}500$ in the favourable case and $B_n^u = -100$ in the unfavourable case. Now suppose we keep all parameters the same, except that we change B_n^f to 3,000 and change B_n^u to –600. Notice in particular that the expectation of B_n,

$$\mathbf{E}(B_n) = 0.5 \times (3000 - 600) = 1200, \quad n \geq 1$$

is the same as before, but that the variance of B_n is larger. This new parameterisation models greater uncertainty about the range of future benefits without any change in the net expected benefit, ie, a "higher risk, higher return" project.

Since the expectation remains the same, the NPV of strategy 1, ("restructure at time 0"), given by equation (18), is the same as before, because (see equation (12)) the expected benefit S_0 of switching at time 0 depends only on the expectation of each B_n. On the other hand, if the designer waits for 1 month and switches only if the situation is favourable (strategy 2), the net benefit S_k^f (see equation (15)) is

$$S_k^f = B_1^f R/(R - 1) = 3000 \times 1.1/0.1 = 33000, \quad k \geq 1$$

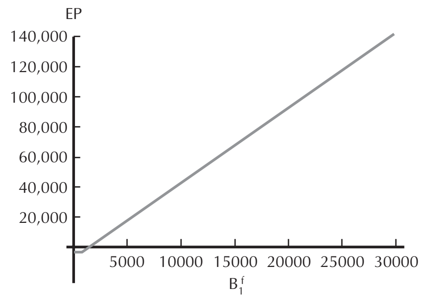
which is bigger than the previous S_k^f value of 27,500. Thus the NPV of strategy 2 (see equation (19)) is bigger than before.

This shows that the incentive to delay the decision to invest in restructuring increases with project risk, manifested as uncertainty over future benefits B_n, as long as all else, including the *expected* benefit, stays the same. Conversely, as uncertainty about future value diminishes, it becomes clearer whether it would pay to invest. The option value preserved by delaying diminishes. In the limiting case, you can decide immediately based on the static NPV.

Consider again the quantity EP = NPV(2) – NPV(1). The graph of EP against B_n^f (such that $E(B_n) = 1{,}200$ or $B_n^u = 2400 - B_n^f$) is shown in Figure 3. The options formulation makes the issue clear. The expected payoff of restructuring immediately is the same as before, since the values $E(B_n)$ are the same. However, if restructuring is delayed, then one of two outcomes occurs. In the unfavourable case (see equation (21)) the payoff V_1^u is still zero, because the design option will not be exercised. However, in the favourable case, the payoff V_1^f (see equation (20)) is greater than before. Thus the option value V_0 given by equation (22) increases.

These conclusions make sense intuitively. Uncertainty over the value of investing in a users' manual, for example creates an incentive to wait for information. If a manual is very likely to be profitable, there is less benefit to waiting. Similarly, if its value is clearly minimal or negative, a decision not to invest can be made immediately. Thus we can conclude with the following qualitative design guideline.

3. The benefit of waiting, EP, versus B_1^f, given that $\mathbf{E}B_1$ remains fixed at 1200



With other factors, including the static NPV, remaining the same, the incentive to wait for better information before effecting a design decision increases with risk – ie, with the variance, or spread, in possible benefits.

Effect of the probability of a favourable outcome

In the example of the previous section, we assumed that at time 0 the likelihood of favourable and unfavourable outcomes was equal, with $p = 0.5$. This probability distribution represents the risk that the favourable outcome will not be actualised. We now examine how the value V_0 of the real option depends on the probability p of a favourable outcome.

Consider the payoff $G_0 = (S_0 - L)$ from immediate exercise, ie, the NPV of strategy 1 (see equation (17)):

$$G_0 = B_0 + (pS_1^f + (1 - p)S_1^u)/R - L = (p/R)(S_1^f - S_1^u) + B_0 - S_1^u/R - L$$

If we plot G_0 against p the slope would be $(S_1^f - S_1^u)/R$. The discounted expected value of the option V_1 is thus (equation (22))

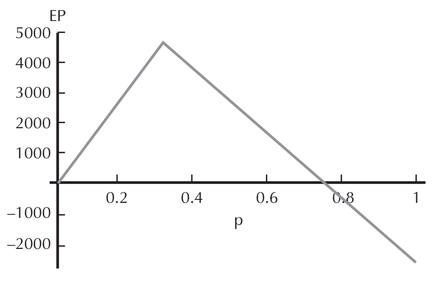
$$\mathbf{E}V_1/R = (p/R)(S_1^f - L)$$

This is the NPV of strategy 2. If we plot this value against p, we find the slope to be $(S_1^f - L)/R$. Since we have assumed $S_1^u < L$, we see that as we increase p, the NPV of strategy 1 grows faster than that of strategy 2. Thus as the probability p of a favourable outcome increases, at some point the strategy of investing right away becomes optimal. To put it differently, as the risk of an unfavourable future decreases, so does the incentive to wait. The graph of EP = NPV(2) – NPV(1) versus p is shown in Figure 4. We have thus given a rigorous basis for the following design decision-making heuristic:

The incentive to wait before investing increases with the likelihood of unfavourable future events occurring.

Notice that this decision-making heuristic addresses uncertainty and risk in a different way from the previous rule. The previous rule addresses the variance in the payoffs under different outcomes. This rule addresses variation in the uncertainty about the likelihood of future events that influence outcomes. These are two important but distinct dimensions of risk.

4. The benefit of waiting, EP, versus the probability p of a favourable outcome p



Effect of uncertainty over direct cost

We now examine the possibility of the cost L being uncertain. This aspect of uncertainty is critical in software engineering, especially if delaying design decisions is an accepted strategy: It goes to the question of estimating project costs, which is widely known to be subject to significant uncertainties. Uncertainty about costs might reflect uncertainty about availability of skilled labour in the future, or about changes in technology, such as the development of automated tools or new methods that could reduce costs.

To simplify matters, let us assume that the monthly profit B_n from restructuring is 1,500 at all times $n \geq 0$ (there is no uncertainty in this regard). Thus the expected benefit of switching at time k, for any $k \geq 0$, is given by an expression analogous to equation (15) (in either scenario):

$$S_k = B_k \frac{R}{R-1} = 1500(1.1)/(0.1) = 16500, \quad k \geq 0$$

However, now assume that the direct cost L_0 at time 0 is known to be 10,000, but that it is uncertain at time 1. Let us assume that L_1 is either L_1^f = 5,000 (a favourable situation) or L_1^u = 20,000 (an unfavourable situation). The NPV of strategy 1, investing now, is

$$NPV(1) = S_0 - L_0 = 16500 - 10000 = 6500$$

which is positive. The traditional NPV rule suggests switching right away. Again, this rule is faulty because it ignores the contingent, option strategy: wait a month, and switch only if the direct cost is L_1^f = 5,000. The NPV of this strategy is

$$NPV(2) = (p/R)(S_1 - L_1^f) = (0.5/1.1)(16500 - 5000) = 11500$$

which is considerably greater than the NPV of the first strategy. Thus it is optimal to wait a month in this case before deciding whether to invest.

Now let us go a step further, and see what happens if we keep L_0 the same and increase the uncertainty (in particular, the variance) of L_1, while keeping its expectation $\mathbf{E}L_1$ the same. This would mean L_1^f is smaller, and NPV(2) larger. In this case, the value of waiting is even greater. This situation is analogous to the one in the section headed "The effect on uncertainty over benefits". When the uncertainty over direct costs is larger, and the expectation remains the same, the potential profit in the favourable scenario increases, while in the unfavourable scenario it remains the same at 0. Thus we have provided a justification for another heuristic:

All else being equal, the value of the option to delay increases with variance in future costs.

AN OPTIONS INTERPRETATION OF INFORMATION HIDING

Having discussed the application of real options to reasoning about the value of being able to decide when to invest in software design decisions, we now turn to the value of modularity. Although real options have had little impact on software engineering to date, it is fascinating that concepts from software engineering have made their way into serious work in the real options literature. Here, we discuss the work of Baldwin and Clark. Based on Parnas's seminal concept of information hiding modules, which hide internal design details that are likely to change (Parnas, 1972 and 1985), Baldwin and Clark interpret modular designs (for computer hardware) as creating real options (Baldwin and Clark, 1999).

They aim to develop a theory of the evolution of the computer industry. Before 1960 or so it was highly centralised. Now it is very decentralised. They see firms as being driven by the options value of modularity to modularise designs, but that published modular architecture enables competitors to compete for slots in the design (eg, disk drives). Thus, modularity in design drives decentralisation of the industry structure.

In more detail, modules create options to select the best implementation from a set of implementations for a given slot in a modular design. Variants are produced by experimentation, whether internal to a firm or in the marketplace. Modules thus create real options for designers to change hidden module details without impacting the rest of a system. The investment in the design of the module thus has a benefit that can be expressed as the value of such an option. (Modularity does have other benefits, as well, which are not necessarily captured in an options formulation, eg, supporting large-scale development through the concurrent efforts of designers working on separate modules.)

More generally, modularising a system involves the selection of a portfolio of options. The choice of modular architecture for a system should be based on the goal of maximising its value, which includes options values. Information hiding concepts having informed real options thinking, it now appears that real options thinking can inform software design theory. Taking the options perspective suggests ways to broaden the traditional view of information hiding.

Recall that the idea of information hiding is to hide within stable module boundaries aspects of a system that are *likely* to change (likelihood is rarely quantified). As a bet on future conditions, it might make sense to hide design aspects that are *unlikely* to change, but where the risk is high. An option created by a module will have some value even if the likelihood of payoff is small, provided that the payoff in the unlikely event is sufficiently high and the cost of the option small. We are thus driven at least to consider using information hiding to speculate on *unlikely* changes. More broadly, the options perspective allows us to integrate the information hiding concept into a more comprehensive understanding of software design as a strategic activity.

Baldwin and Clark model the value of a system as a sum of the values of the modules in the system. They take value maximisation as the goal of design and derive design investment policies under which such maximisation is achieved. They assume that a given system has already selected a module implementation for each slot and that the other implementations (produced by investments in experimentation) have values that are normally distributed around that of a currently selected implementation, with a variance determined by the number of design parameters that are bound within the module. Among other aims, they seek to gain a better understanding of optimal levels of investment in experimentation and testing to produce and evaluate variant module implementations. They conclude that firms generally under-invest in experimentation.

This work is encouraging for software design and engineering. It substantiates our insight that options thinking can help to model, analyse and explain basic issues in software design, including information hiding. Their work leaves many questions unanswered (as does ours). First, although they draw heavily from software design, appealing to information hiding and structured design, they do not try to develop an investment or options-based view of software design principles themselves. Second, their additive model of the value of a system is simple. It might be a reasonable model of systems that are essentially collections of largely independent features. Some shrink-wrapped software packages have roughly this structure (Cusumano and Selby, 1995). However, the model does not capture other systems well, in which the value of the system is not a simple sum of module values. Nor are software module implementations necessarily created by a random walk around the current module, as Baldwin and Clark assume. In future work, we plan to investigate the ways in which real options in real software systems interact, more convincing models of module-based options in *software*, and the extent to which such options might be valued in an arbitrage style using market data.

AN OPTIONS INTERPRETATION OF THE SPIRAL MODEL

The spiral model (Boehm, 1988), is generally regarded as a risk-based process model. We sketch an account of the spiral model as a value-maximising rather than just a risk-minimising model, based on real options concepts.

Risks in the spiral model are addressed by a phased project structure in which, in each phase, important risks are mitigated, often by investing in prototypes, user surveys or other such experimental efforts. In addition, investments are made at the beginning of each phase to develop and evaluate a range of solution approaches for meeting the objective of the phase. At the end of each phase, the project is re-evaluated, and plans are developed for the next phase.

The spiral model is often held up against the waterfall model, historically the most important software development project model and one that is still widely used; and the relative benefits of the spiral model are stressed. The waterfall model begins with a feasibility study, after which there is decision to proceed or not. Once there is a decision to proceed, the project goes forward mechanically through elaboration of requirements and specifications, design of a modular system architecture, parallel implementation and testing of the individual modules, system integration, system testing, release and maintenance. Investments of tens or hundreds of millions of dollars are often handled using this process model.

Today, researchers and many practitioners acknowledge the chief deficit of the waterfall model: it does not recognise risk or uncertainty as critical issues in the development of complex software-intensive products. It includes no serious strategy to address uncertainty. It is thus not surprising that beyond the initial feasibility study, the model does not promote investment in options; it is not designed to have embedded options for the designer to change course; nor does it suggest a dynamic approach to managing projects in uncertain conditions as new information continually arrives.

By contrast, the central elements of the spiral model have clear interpretations in real options terms. Analysing the model in this way provides a valuable way of seeing this process model as one concerned with value maximisation under uncertainty.

First, the *implicit* message in the model is that decisions should be made that create the most value. Boehm states that the designer begins with a hypothesis that a project will "*improve the operational mission*" of an enterprise. Recasting this point in economic terms leads to the position that the designer begins with a hypothesis that investing will *maximise value added*. This formulation is consistent with Boehm's description and makes the model consistent with standard corporate finance theory. Similarly, Boehm states that the designer should terminate the spiral (cancel a project) when the mission improvement hypothesis is rejected. Our interpretation is that one should decline to invest in the next phase when doing so is not optimal for maximising added value.

Second, the model exploits options extensively. Developing and evaluating variant approaches in each phase creates option to pick the best of n approaches identified. Perhaps most important, the phased project structure creates value in the form of compound options: the investment in each phase creates an option to invest (or not) in the next phase.

What we have done is to interpret software development process models as implicit investment strategies, and we sketched an explanation in options terms for why one model (spiral) is so much more effective than another (waterfall). We elaborate this idea after introducing a concept important for understanding the treatment of uncertainty by the spiral model, but one not based mainly on options concepts. The issue is the value of information that can be obtained by investing.

Value of flexibility versus value of information

One of the most important tactics that Boehm emphasises for mitigating risk is use of prototyping to gain information. In software design it sometimes pays to invest a little in design and implementation early to resolve uncertainties that might affect future outcomes, (Boehm, 1981). There is a significant difference between prototyping and options that needs to be understood. Although prototyping seeks value maximisation under uncertainty, it does not do so by waiting for the resolution of exogenous uncertainty, but rather by investing now in a controlled way. For example, when a designer invests in a prototype to resolve a risk, it is investing itself that resolves the uncertainty. By contrast, you cannot learn how much a stock is going to be worth tomorrow by investing a little today.

The essence of prototyping is thus not in the value of flexibility (VOF) to respond to uncertain exogeneous conditions, but in the value of information (VOI) obtained by investing. The idea is that an investment in a prototype is justified when the information revealed is worth more than it costs – eg, when it helps to avert a costly commitment to an unworkable design. Hillier and Leibermann (1995) provide an introduction and references to the mathematical formulation in terms of statistical decision theory.

The distinction is important. For example, while VOF considerations can drive one to delay investing early, VOI is just the opposite. Because important information won't be obtained until it is "purchased," it can make sense to buy it early. The fact that the two situations are so different suggests that a comprehensive approach to managing uncertainty in software design will require an integrated set of tools, including tools for modelling and analysing both options (using arbitrage and non-arbitrage methods) and the value of information. Ultimately, we need process models that go beyond the spiral in taking a comprehensive approach to the modelling of, analysis of, and decision-making under uncertainty in software design.

Real options in projects based on the spiral development model

A key element of the spiral model is the generation of alternative solution approaches at the beginning of each phase. Generating alternatives is related to the generation of alternative module implementations in Bladwin and Clark's real options analysis of the value of information hiding. Generating a set of n solutions provides the designer with the option to select the best. The options view suggests qualitative guidelines for deciding how much to invest in the exploration that leads to new alternatives: when the risk is high, options are worth more, so spend more on options.

Phased projects as compound options

Phased project structures are an essential part of the spiral model. The goal is to keep investments small until risks are resolved. Such a structure provides an ability to manage risk by embedding options in a project to change course or even to abandon it as uncertainties are resolved over time. One way to think about phased projects is that they contain embedded options to scale back or to abandon the project if conditions turn out to be unfavourable (Trigeorgis, 1996). Such options are *put* as opposed to *call* options, giving holders the right to sell assets at a specified price (eg, to abandon a project in return for the market value of its assets or avoidance of future costs). Embedding such options in the project increases its value relative to one without options.

A second interpretation is that investing in one phase of a project buys the designer an option to make a follow-on investment should the situation be favourable at that time. A designer considering such initial investment thus has an option on an option, called a compound option. Similarly, a second phase can yield an option on the third, and so on. Real options theory has been applied to value such phased projects as compound options (Geske, 1979).

A striking insight emerges from the options view: it can be optimal to invest in the first phase of a project even though the overall project has a negative *static* NPV (Brealey and Myers, 1996). The key idea is that spending a little on one phase can yield an option to invest in the next if conditions turn out to be favourable. If the upside potential of the second phase is high, the option value today can outweigh its cost, even if the entire project currently has a negative NPV. Although this concept is familiar to those who have studied modern corporate finance, including options concepts, it is not the way that software designers are taught to think.

Extending this insight to the spiral model we infer that it can be optimal to begin a spiral not on the hypothesis that the project *will* improve the operational mission of the firm, but that it *might*. In other words, we infer that it can pay to begin a spiral process as a *strategic bet* on favourable future conditions. It might be worthwhile to invest in learning a standard even before it is widely adopted, if doing so makes it possible for the designer to respond quickly if the standard does become widely adopted, for example.

A concrete example

We make the preceding point concrete with a simple example. Suppose that the designer can restructure in two phases. The first phase costs 1,000. With probability 0.5, that is the total cost incurred; but with probability 0.5 serious difficulties will be encountered, and 3,000 more dollars will have to be invested to produce a satisfactory design. Perhaps the selected restructuring tool has unforeseen shortcomings that require some manual restructuring. More generally, a project might contain risks that are resolved by investing in the first place.

Suppose that the profit from restructuring, at each step t, is 200. Summing up the sequence, the present value of this stream, at any time, with the discount rate at 10%, turns out to be 200R/(R – 1) = 2,200. NPV analysis compares the expected cost, 1,000 + (0.5)(3,000) = 2,500, with the present value of profits, 2,200. Owing to the risk, the static NPV is –300. Not investing appears to be advised: the project does not improve the mission of the firm.

However, this analysis ignores the value of the option to cancel the project after the first phase if a second phase costing 3,000 turns out to be necessary. The traditional NPV analysis ignores the contingent strategy enabled by the embedded option to cancel the project. The dynamic NPV is (0.5(2,200) – 1,000 = 100. In other words, for 1,000 the designer can buy an asset – an option on the second phase – that with even odds will be worth either 2,200 or 0. It's a good bet, with an expected value of 100.

Traditional software engineering economics using static NPV overlooks the value of the options – ie, flexibility – embedded in development projects and products. The options approach is superior to static NPV analysis for quantitative reasoning about projects and assets with embedded options. More importantly for our purposes in this chapter, it provides a framework for understanding strategic design, by which value is created with intelligent bets on uncertain future outcomes.

REAL OPTIONS IN TRADE-OFFS RELATING TO TIME TO MARKET

Finally, we discuss an options-oriented approach to reasoning about an issue that troubles many software engineering researchers: the rush to market with software products that are not engineered to the highest standards. In the internet economy, in which technology is evolving at a tremendous pace and lock-in is a serious issue, opportunities to exploit new markets are fleeting, and the threat is always looming that a competitor will take that market away by getting there first. There is thus a powerful heuristic commonly employed by software designers: "be first". The designer often sacrifices some quality to be the first with a new product.

The threat can be understood in options terms. The entry of a competitor can reduce the share of a market that would otherwise be available for a designer to exploit, and thus the value of the cashflow stream from sales that the designer gets by shipping a product. Such a discrete drop in asset value can be seen as a dividend – similar to the dividend that a stock pays, with an immediate, corresponding drop in share value.

It is known in options theory that it is optimal to delay exercising an American call option on a stock that does *not* pay dividends until expiration. In other words, there is no opportunity cost to delaying, and you know most about the value of the stock just before your right to buy it expires. In this case, waiting as long as possible is the right strategy. However, when a stock pays a dividend the share value drops and the option owner is not entitled to the dividend. The payoff from exercising after a dividend is reduced. That is, there is an opportunity cost to waiting. In these cases, it can be optimal to exercise early to capture value that would be lost otherwise.

Consider a software designer developing a product under threat of competitive entry. The designer has an option to ship the product, ie, to exercise an option to capture a share of the market, even if the product is not yet perfect. The entry of a competitor, reducing that share, is a dividend (Trigeorgis, 1996). To capture that benefit the designer might ship early. More generally, dividends create a countervailing incentive: not to delay, but to exercise early to capture benefits that flow from actually owning an asset. We do not advocate designers shipping shoddy products. That is not likely to maximise value over time. However, we can see why in some cases trading some quality for time could be the right *engineering* choice. The purpose of design is value, not perfection.

RELATED WORK

To begin our summation, we address related work in several areas: flexibility in software; options; software engineering economics; software reuse investment analysis; and real options in the analysis of information technology projects. In the following section, we discuss theoretical difficulties in taking an options approach to computer-related issues.

Flexibility in software product and process design

We are obviously not the first to notice that flexibility is critical in software design and that it has both costs and benefits that have to be weighed against each other. Information hiding (Parnas, 1972), extension and contraction (Parnas, 1978), and program family concepts (Parnas, 1976), were seminal. More recently, Fayad and Cline (1996), emphasise that flexibility has costs, that it should be designed in where it is economically the most effective. They see *design patterns* (Gamma, Helm and Vlissides, 1995), as providing "hinges" needed for flexibility in particular dimensions. They even state that such hinges provide "opportunities" to make changes that might be necessary – reflecting an implicit options mode of thinking. However, to the best of our knowledge there have been few attempts to connect such concepts explicitly to precise notions of the value added by flexibility.

Financial options theory

Real options theory is based on financial options theory. The options literature is immense. It is not feasible to cite all relevant work. Financial options have been studied since 1900; however the seminal modern results, which provided long-sought closed-form mathematical formulations for valuing financial options, are due to Scholes, Merton and Black, Black and Scholes (1973) and Merton (1973). Scholes and Merton received the 1997 Nobel Prize in economics for their work on this topic. Many other results, which are now elements of basic finance, have been produced since (Brealey and Myers, 1996; Cox and Rubinstein, 1984 and 1979; and McDonald and Siegel, 1986). For the past 20 years, researchers have been building the theory of real options (Brealey and Myers, 1996; Dixit and Pindyck, 1994; Trigeorgis, 1995).

Software engineering economics

The idea of finance-based decision criteria for software design is not new. Boehm wrote on it in his *Software Engineering Economics* (1981). In general, he emphasised static net present value as the basic concept of value and how it might be taken as the objective of software design. He appealed to statistical decision theory as well to reason about the value of investments in information under uncertainty.

Boehm's emphasis on static NPV is understandable. Static NPV was the standard measure of value added taught in business schools. Moreover, real options were hardly known when Boehm's book was published. Boehm mentions options in the colloquial, but it does not analyse software design concepts in terms of real options.

The software reuse literature reports applications of finance theories to software reuse investment analysis (Lim, 1996). Favaro has argued that much work in this area uses techniques known to have technical problems. He has argued that NPV is the proper basis for such analysis (1996). That view is consistent with the traditional academic view, but it does not consider the value of flexibility.

Withey (1996 – and more recently Favaro *et al*, 1998) considers flexibility in options terms in work on investment analysis of product line architectures. He shows how economies of scope achieved through investments in product-line design can be interpreted as options on uncertain markets. Withey did not invoke real options because of questions about its broader validity for analysing software design decisions (personal communication). We agree with Withey that it is critical not to take real options or other such investment concepts as silver bullets for software design decision-making. It is especially important to understand the conditions under which arbitrage pricing techniques can legitimately be used.

It bears mentioning that Boehm (1981) warns of the pitfalls of adopting financial criteria for software design. Focusing exclusively on *quantifiable* economics can compromise attention to human economic issues that in the long run are critical to success. Our work is meant to provide intellectual tools to help designers think more effectively about important issues in design decision making, and to provide a basis for the eventual development of modelling and analysis tools. But models are prone to incompleteness and inaccuracy. They can improve but not replace human judgement.

Information technology project valuation

Henderson and Kulatilaka are developing an approach to investment analysis of information technology (IT) projects based on real options (Kulatilaka, 1996). The proposed approach appears to be a standard use of real options for investment analysis at the project level. The work does not address software design. Flatto (1996) reports on a survey of whether managers use real option concepts in IT investment analysis, which showed that options concepts are not used.

PRACTICAL AND THEORETICAL DIFFICULTIES

The quantitative use of real options theory to support software design decision-making is an attractive possibility, but one that will clearly require the careful treatment of practical and theoretical problems. In this section, we enumerate some of these problems and discuss possible solution strategies.

Arbitrage-based valuation techniques require knowledge of the current asset values and of the risks of assets determined by appeal to market data. They do not require subjective estimates of cashflows. When arbitrage-based techniques cannot be applied, estimates of cashflow streams and discount rates are needed. Estimating software costs remains a highly imprecise exercise, and little work has been done on benefit estimation. We provide a framework within which to explore the implications of various estimates. Having such a framework in addition to sheer intuition is likely to be better than not. However, the problem of estimating parameters will remain a serious impediment to accurate valuation of some software options.

Second, formulating design issues in options-oriented terms can be hard because each possible product or process design is likely to contain a set of embedded options, each of which should be valued to value the design. Building options-based models is a challenge. Tool support would help. The feasibility of building real options-based models of industrial-scale software projects has yet to be determined, and is the subject of a case study that we are pursuing.

Another challenging problem is to identify and model quantitatively the underlying uncertainties that determine future costs and benefits. When the uncertainties have been priced by the market in cases where traded stocks reflect the market judgement of the uncertainty, then market data can be used as a basis for estimates. In cases where the uncertainty is private to a given project, expert subjective estimates will be needed, and it will be hard to provide reliable and justified estimates.

The distinction between options that are in and not in the "span of the market" might help explain Strassmann's reservations about Henderson's use of options in valuing IT investments (Strassmann, 1997). Strassman objects that in richly traded markets there is information on values and uncertainty. However, he notes that computer projects occur in low volumes, and he argues that therefore getting valid data, treating them consistently, and dealing with non-quantifiable effects makes IT project valuation different. Strassmann then concludes that "... there is no true resemblance between trading in financial options and planning computer projects", (Strassmann, p. 159).

Strassmann is correct that arbitrage-based techniques (which many people define as options pricing techniques) cannot be used for options not in the span of the market. He is also clearly correct that estimating valid inputs for options that are not in the span is difficult. Lack of spanning can be a problem, but options are present whether they can be priced by arbitrage or not. As Dixit and Pindyck state that "whether or not spanning holds, we can obtain a solution to the investment problem, but without spanning, the solution will be subject to an assumed discount rate" (1994, p. 152). Furthermore, the assumed discount rate is not itself subject to theoretical justification. In essence, when markets are incomplete (spanning does not hold), the data required for pricing using dynamic programming will be subjective estimates. Strassmann's valid objection is that getting valid estimates will be hard.

The pessimist will claim that we are back to where we started, with design reasoning being based on no more than expert opinion. In cases where arbitrage-based pricing is possible, market-based valuations of real options are possible without the need for subjective projections and estimates. Even in cases where market-based data are not available, the options perspective appears to be useful, because it highlights the role of flexibility in software design and gives the designer a way to think about that value as being tangible. Software designers have to estimate such quantities as "likelihood of change" today, in any case, in order to apply existing concepts such as information hiding. However, these estimates are treated without the benefit of any kind of well-grounded mathematical model. Options concepts identify the parameters in the underlying investment decisions and tells us how they relate to optimal strategy for value added. At a minimum, it provides a framework for back-of-the-envelope models, with support for sensitivity analysis. We expect real gains as researchers and designers explore the implications of the options view for *strategic software design*.

Software strategy is the term that we use to refer to software design techniques in which uncertainty, incomplete knowledge, competition, and related issues are treated systematically and consciously in a sound manner designed to maximise the expected value of a given product or project. A designer will use techniques such as modularity and phased project structures with a conscious understanding of how they provide hedges, bets, and the flexibility to respond to new information. Ideally, designers will be able to engineer projects for selected levels of exposure to risk and reward. The options framework provides a powerful language in which to develop this idea. We are exploring the idea of managing software projects as portfolios of assets whose value includes the values of embedded real options. A card game might be a good metaphor. As uncertainty is resolved over time, investment decisions are made on the basis of the options held at the time: cards are bought, assessed, played, and discarded in a dynamic and strategic game against both nature and adversaries.

CONCLUSION

Existing software design principles, such as information hiding and spiral processes, can be quite effective. To a considerable degree they operate by serving as (imperfect) proxies for value-maximising decision criteria for investing under uncertainty. We have tried to make this insight precise by relating a range of important software design heuristics to options theory. This approach has enabled us to present a somewhat unifying analysis of a set of previously disconnected design principles. In addition to merely analysing such concepts, we have provided some support for the claim that a value-based perspective that accounts for the value of flexibility in terms of real options can lead to a better understanding of, and new insights into software design. For example, we see the spiral model not merely as a risk-reducing model, but as a value-maximising model, with equal emphasis on profitable reduction of downside risk and upside opportunities.

We are taking several additional steps to develop this work. First, a comprehensive approach to scientific uncertainty management for strategic software design requires an integrated set of modelling and analysis methods appropriate for valuing different kinds of decision flexibility under different conditions: arbitrage or not; value of information or flexibility; etc. We are working to elaborate the models.

Second, we are investigating whether real designers change how they make design decisions as they learn about the nature, role, and value of decision flexibility and its options interpretation.

Third, we are developing new, normative development models based on the idea of scientific uncertainty management and strategic design. The idea is that software development should be driven not by risk reduction, *per se*, but value-maximisation. We see an *active investment management* approach, in which software systems and projects are treated as portfolios of assets with embedded options, as a promising start on such models.

Fourth, we are designing software tools for use in such processes. Among other functions, these tools have to support mathematical modelling and analysis of real options in software and continuous monitoring of technical, market and other issues for new information that might change asset valuations in ways that require dynamic decision-making.

We are also working to identify options in software design for which arbitrage-based valuation techniques can be used. The availability of market-calibrated valuations of decision flexibility structures would be enormously useful. The value of portability to new computing platforms, for example, seems amenable to such an approach. Market data are often available on prospects for platforms, for example Apple Corporation's iMac.

1 Our numbers are fictional, and largely from Dixit and Pindyck, 1994, "Investment Under Uncertainty", Princeton University Press.

BIBLIOGRAPHY

Amram, M. and N. Kulatilaka, 1999, *Real Options: Managing Strategic Investment in an Uncertain World*, Cambridge, Massachusetts: Harvard Business School Press.

Baldwin, C.Y. and K.B. Clark, 1999, *Design Rules: The Power of Modularity*, Cambridge, Massachusetts: MIT Press.

Belady, L.A. and M.M. Lehman, 1976, "A Model of Large Program Development", *IBM Systems Journal*, 15(3), reprinted in Lehman, M.M. and L.A. Belady, 1985, *Program Evolution*, London: Academic Press, pp. 165–200.

Black, F. and M. Scholes, 1973, "The Pricing of Options and Corporate Liabilities", *Journal of Political Economy*, 81(3), pp. 637–55.

Boehm, B.W., 1981, *Software Engineering Economics. Advances in Computing Science and Technology*, Prentice Hall.

Boehm, B.W., 1988, "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, 21(5), pp. 61–73.

Boehm, B.W. and R. Ross, 1989, "Theory-W Software Project Management: Principles and Examples", *IEEE Transactions on Software Engineering*, 15(7).

Boehm, B.W. and K.J. Sullivan, "Software Economics: Status and Prospects", invited paper, *Information and Software Technology*, special millennium issue, forthcoming.

Brealey, R. and S. Myers, 1996, *Principles of Corporate Finance*, New York: McGraw-Hill, fifth edition.

Brennan, M. and E. Schwartz, 1985, "Evaluating Natural Resource Investments", *Journal of Business*, 58(2), pp. 135–57.

Corman, T.H., C.E. Leiserson and R.L. Rivest, 1990, *The Design and Analysis of Computer Algorithms*, Cambridge, Mass.: MIT Press.

Cox, J. and M. Rubinstein, 1984, *Options Markets*, Englewood Cliffs, NJ: Prentice-Hall.

Cox, J.C., S.A. Ross and M. Rubinstein, 1979, "Option Pricing: A Simplified Approach", *Journal of Financial Economics*, 7, pp. 229–63.

Cusumano, M. and R. Selby, 1995, *Microsoft Secrets*, New York: Free Press.

Dixit, A. and R. Pindyck, 1994, *Investment under Uncertainty*, Princeton Univesity Press.

Gamma, R.J.E., R. Helm and J. Vlissides, 1995, *Design Patterns*, Reading, Massachusetts: Addison-Wesley.

Favaro, J., 1996, "A Comparison of Approaches to Reuse Investment Analysis", *Proceedings of the Fourth International Conference on Software Reuse*, 4.

Favaro, J., K.R. Favaro and P.F. Favaro, 1998, "Value based software reuse investment", *Annals of Software Engineering*, 5, pp. 5–52.

Fayad, M. and M. Cline, 1996, "Aspects of Software Adaptability", *Communications of the ACM*, 39(10), October, pp. 58–9.

Flatto, J., 1996, "The Application of Real Options to the Information Technology Valuation Process: A Benchmark Study", PhD thesis, University of New Haven.

Geske, R., 1979, "The Valuation of Compound Options", *Journal of Financial Economics*, 7, pp. 63–81.

Griswold, W. and D. Notkin, 1993, "Automated Assistance for Program Restructuring", *Transactions on Software Engineering and Methodology*, 2(3), July.

Habermann, A., L. Flon and L. Cooprider, 1976, "Modularization and Hierarchy in a Family of Operating Systems", *Communications. of the ACM*, 19(5), May, pp. 266–72.

Hillier, F. and G. Liebermann, 1995, *Introduction to Operations Research*, McGraw-Hill, sixth edition.

Hull, F., 1993, *Options, Futures, and Other Derivative Securities*, Prentice Hall, second edition.

Kester, C., 1984, "Today's Options for Tomorrow's Growth", *Harvard Business Review*, March/April, pp. 153–60.

Kulatilaka, N., 1993, "The Value of Flexibility: The Case of a Dual-Fuel Industrial Steam Boiler", *Financial Management*, 22(3), Autumn, p. 271.

Kulatilaka, N., P. Balasubramanian and J. Storck, 1996, "Managing Information Technology Investments: A Capability-based Real Options Approach", Boston University School of Management Working Paper No. 96-35, June.

Lim, W., 1996, "Reuse Economics: A Comparison of Seventeen Models and Directions for Future Research", *Fourth International Conference on Software Reuse*.

Luenberger, D., 1998, *Investment Science*, New York: Oxford University Press.

Madj, S. and R. Pindyck, 1987, "Time to Build, Option Value, and Investment Decisions", *Journal of Financial Economics*, 18(1), pp. 7–27.

Maes, P., 1994, "Agents that Reduce Work and Information Overload", *Communications. of the ACM*, 37(7), July, pp. 31–40.

McDonald, R. and D. Siegel, 1986, "The Value of Waiting to Invest", *Quarterly Journal of Economics*, 101(4), pp. 707–27.

Merton, R., 1973, "Theory of Rational Option Pricing", *Bell Journal of Economics and Management Science*, pp. 141–83.

Merton, R.C., 1990, *Continuous-Time Finance*, Cambridge, Mass.: Blackwell.

Mitchell, T., R. Curuana, D. Freitag and J. McDermott, July 1994, "Experience With a Learning Personal Assistant", *Communications of the ACM*, 37(7), pp. 81–91.

Myers, S., 1977, "Determinants of Corporate Borrowing", *Journal of Financial Economics*, 5, pp. 147–75.

Myers, S. and S. Majd, 1990, "Abandonment Value and Project Life", *Advances in Futures and Options Research*, 4, pp. 1–21.

Parnas, D., 1976, "Program Families", *IEEE Transactions on Software Engineering*, SE-2(1), March, pp. 1–9.

Parnas, D., 1985, "The Modular Structure of Complex Systems", *Transactions on Software Engineering*, SE-11, March, pp. 259–66.

Parnas, D.L., 1972, "On the Criteria to be Used in Decomposing Systems into Modules", *Communications of the Association of Computing Machinery*, 15(12), December, pp. 1053–8.

Parnas, D.L., 1978, "Designing Software for Ease of Extension and Contraction", *Proceedings of the Third International Conference on Software Engineering*, pp. 264–77, IEEE.

Pindyck, R., 1988, "Irreversible Investment, Capacity Choice, and the Value of the Firm", *American Economic Review*, 78(5), pp. 969–85.

Ross, S., 1995, "Uses, Abuses, and Alternatives to the Net-Present-Value Rule", *Financial Management*, 24(3), Autumn, pp. 96–102.

Shaw, M. and D. Garlan, 1996, Software Architecture: Perspectives on an Emerging Discipline, Upper Saddle River, NJ: Prentice-Hall.

Strassmann, P., 1997, *The Squandered Computer: Evaluating the Business Alignment of Information Technologies*, New Canaan, Connecticut: The Information Economics Press.

Stultz, R., 1982, "Options on the Minimum or the Maximum of Two Risky Assets: Analysis and Applications", *Journal of Financial Economics*, 10, pp. 161–85.

Sullivan, K.J., 1996, "Software Design: the Options Approach", Second International Software Architecture Workshop (ISAW-2), in *Joint Proceedings of the SIGSOFT'96 Workshops*, L. Vidal *et al*, ed., New York, NY: Association for Computing Machinery.

Sycara, K., K. Decker, A. Pannu and M. Williamson, 1996, "Distributed Intelligent Agents", technical report, Carnegie Mellon University.

Teisberg, E., 1995, "Methods for Evaluating Capital Investment Decisions under Uncertainty", in *Real Options in Capital Investment: Models, Strategies, and Applications*, L. Trigeorgis, ed., Westport, Connecticut: Praeger.

Triantis, A. and J. Hodder, 1990, "Valuing Flexibility as a Complex Option", *The Journal of Finance*, XLV(2), June, pp. 549–65.

Trigeorgis, L., 1993, "The Nature of Option Interactions and the Valuation of Investments with Multiple Real Options", *Journal of Financial and Quantitative Analysis*, 28(1), pp. 1–20.

Trigeorgis, L., editor, 1995, *Real Options in Capital Investments: Models, Strategies and Applications*, Westport, Connecticut: Praeger.

Trigeorgis, L., 1996, *Real Options: Managerial Flexibility and Strategy in Resource Allocation*, Cambridge, Massachusetts: MIT Press.

Withey, J., November 1996, "Investment Analysis of Software Assets for Product Lines", Technical Report CMU/SEI-96-TR-010, Carnegie Mellon University – Software Engineering Institute.