The Options Approach to Software Prototyping Decisions[1]

Prasad Chalasani Somesh Jha Kevin Sullivan

July 1997

CMU-CS-97-161

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Prototyping is often used to predict, or reduce the uncertainty over, the future profitability of a software design choice. Boehm [1] pioneered the use of techniques from statistical decision theory to provide a basis for making prototyping decisions. However his approach does not apply to situations where the software engineer has the flexibility of waiting for more information before making a prototyping decision. Also, his framework considers only uncertainty over one time period, and assumes a design-choice must be made immediately after prototyping. We propose a more general multi-period approach that takes into account the flexibility of being able to postpone the prototyping and design decisions. In particular, we argue that this flexibility is analogous to the flexibility of exercise of certain financial instruments called options, and that the value of the flexibility is the value of the corresponding financial option. The field of real option theory in finance provides a rigorous framework to analyze the optimal exercise of such options, and this can be applied to the prototyping decision problem. Our approach integrates the timing of prototype decisions and design decisions within a single framework.

Author affiliations: **Prasad Chalasani**, Carnegie Mellon University, chal@cs.cmu.edu, **Somesh Jha**, Carnegie Mellon University, sjha@cs.cmu.edu, **Kevin Sullivan**, University of Virginia, sullivan@cs.uva.edu

[1]This material is based upon work supported in part by ARPA Grant #F33615-93-1-1330, by ONR Grant #N00014-96-1-1222, and by NSF Grant #IRI-9508191.

Keywords: Software engineering, economics, prototyping, real options, sequential investing, financial options

1 Introduction

Although much progress has been made in the identification of key software design paradigms and concepts (such as information-hiding, prototyping, reuse, and program-families), the *criteria* for making decisions about how to use these concepts have largely been ad-hoc. For instance, consider a software engineer faced with the decision of whether to change the implementation of a cache-coherence protocol in a certain distributed database management system. The economic profitability of the new protocol depends on which of two possible future scenarios will occur. In one scenario, the new protocol would work extremely well, whereas in the other, it might degrade performance. The two scenarios are perhaps estimated to be equally likely, but we do not know which one will occur. Prototyping can be a way to predict (or reduce our uncertainty over) which scenario will occur. Based on the prototype the software engineer can perhaps revise his estimate of the probability of the favorable scenario to 70%, and then decide whether or not to change the protocol. Of course constructing and evaluating the prototype involves an expenditure of limited resources such as programmer time, CPU time and memory space. The software engineer must therefore weigh the cost of prototyping against the benefit of prototyping. The benefit of prototyping (as we show in an example later) is the increased expected payoff from a more informed decision based on prototyping, compared to a decision without prototyping.

Boehm [1] was among the first to propose an appealing economics-based approach to providing a sound basis for software decisions in general and prototyping decisions in particular. His approach views software decisions as problems of investing in the face of uncertainty over the future payoffs from the investment. In his model, the effect of prototyping is to change the conditional probabilities of adopting each design alternative given that the future is one of a certain set of possible scenarios. He applies techniques from Bayesian decision theory to calculate the expected payoff from a prototyping decision.

Although Boehm's approach recognizes the importance of uncertainty in modeling prototyping decisions, we see some shortcomings in his approach. To begin with, his approach is based on the traditional net present value (NPV) analysis found in business textbooks: First calculate the present value of the future benefits that the investment (i.e. prototyping and the consequent design choice) will generate. If the future benefits are uncertain, compute the expectation of this present value. Next calculate the (expectation of the) present value of the prototyping and design cost. The NPV is the first quantity minus the second, i.e., benefit minus cost. The software engineer then applies the NPV rule traditionally found in business textbooks (e.g., [2]): if the NPV is positive then proceed with the investment, otherwise don't. However, this simple rule is often suboptimal, since it is founded on a faulty assumption. It views the investor's (or software engineer's) decision as a *now-or-never* decision, in the sense that if the prototyping is not done now, the opportunity is lost for ever. Thus if the only possibilities available to the investor are to invest now or invest never, then he or she should invest if and only if the NPV is positive. However, most investment decisions, including those in the software area, can be *delayed*. This is especially true when a certain implementation of the software is already in place, and a design *change* is being considered. This brings into play several other possible strategies that must be considered when picking the optimal one. Specifically, the investor's decision can be *contingent* on what kind of future unfolds. For instance the investor can wait for a month, and decide to not invest if the next month reveals information that indicates that there would be no net positive payoff from the investment. Or perhaps after a month there is less uncertainty about the profitability of the investment. This type of wait-and-see strategy can have a larger expected payoff (even when discounted to the present time) than investing right away, so that even when the above NPV rule indicates an investment, it would be suboptimal to invest right away. Of course, if the expenditures can somehow be recovered when conditions turn sour, then the decision to invest is effectively reversible, and in this case it would suffice to consider this as a now-or-never decision. However, the expense of undertaking a software design decision cannot usually be recovered. In this sense software decisions, including prototyping decisions, are *irreversible* but capable of being delayed.

Another limitation of Boehm's framework for prototyping is that he only considers one period of uncertainty, and assumes that both the prototyping and design decisions must be made in this period. However in reality, once the prototyping is done, the software engineer can wait several time periods to obtain more information before making his design choice (especially if a design *change* is being considered).

In this paper we view the prototyping and design decisions not merely as investments under uncertainty (as previous work has done), but more specifically as *irreversible but delayable investments under uncertainty*. In a growing body of research [4, 5, 6, 7, 10], this viewpoint is being increasingly accepted as the appropriate one (as opposed to the traditional NPV rule) for analyzing capital investments by companies in the face of an uncertain future. Rather than viewing investments as being either now-or-never propositions or as being reversible, this approach stresses the fact that companies have *opportunities* to invest, and that they must decide how best to exploit these opportunities. At any time, the company has the *option* to make the investment: it has the opportunity, but is not obligated to, make the investment. The success of this new research is based on the important observation that investment opportunities are analogous to financial *call options* (see the Appendix for details), and that when a company makes an irreversible investment, it in effect "exercises" its call option. In fact, to make the analogy explicit, the investment opportunity available to the company has been called a "real option". A real option is the flexibility a manager has for making decisions about *real* assets (in contrast to shares of stock) [9]. Thus the company's investment problem is essentially one of optimally exercising its call option. The valuation and optimal exercise of financial options has been an active area of research in finance over the past two decades. Thus researchers have been applying results from the well-established area of options to capital investment problems, and this has lead to the new field of *real option theory.*

In this paper we are advocating a similar use of the options approach to study software prototyping and design decision problems. This has the benefit that we can draw upon the research done in the area of real options. A preliminary paper outlining this approach was presented by one of the authors at the 1996 Software Architectures Workshop [11]. A more detailed paper that applies our ideas to software design decisions appears in [12]. Withey [13] also applies real options ideas to analyze strategies for building reusable software modules. The present paper focusses on developing models for analyzing prototyping strategies as well as design strategies that are influenced by prototyping. Since prototyping can affect our estimates of the probabilities of future decisions, or our estimates of future payoffs, the model presented in [12] cannot be used to study prototyping decisions. An additional complication in the current paper is that we study the optimal timing of both the prototyping decision *and* the resulting design decision in the same framework. (Our earlier paper [12] alludes informally to the possibility of modeling multiple decisions.) Thus in the present paper we need a significantly more complicated model.

Our proposal is that formulating software prototyping criteria in terms of real options addresses the limitations of earlier work on prototyping. Moreover, this approach can lead to previously unrecognized design principles. Our goal is to show that real options theory can help describe, and perhaps even prescribe, software prototyping and design decision-making behaviors.

The rest of this paper is organized as follows. Section 2 describes the basic idea of real option theory and how to apply it to software design decisions. Section 3 and 4 present two alternative models of prototyping and design-decisions based on the options viewpoint. In the first model, the effect of prototyping is modeled as changing our estimates of the probabilities of future events that affect the profitability of our design choices. In the second model, the effect of prototyping is to change the future payoffs from design choices. This model is related to Boehm's treatment in [1], and we recast his example in detail in terms of our model. Section 5 contains a conclusion. For easy reference, the necessary mathematical terminology and notation is described in Appendix A.1. A brief introduction to financial options appears in Appendix A.2.

2 Real options

We now describe some basic ideas in the area of real options theory. See Appendix A.2 for a brief introduction to financial options.

An investor holding an American call option is faced with a decision problem: when to exercise his option. This situation is very similar to the problem faced by managers making irreversible capital investment decisions. Suppose the manager of a computer peripheral manufacturing firm is contemplating whether to invest in a large factory for making a new kind of widget. The investment is irreversible, since the factory can only be used to make these widgets, and if market conditions turn out to be unfavorable, the firm cannot regain its lost investment. For simplicity let us say that the factory can be built instantaneously, at a cost L, and that it can produce widgets forever at zero cost. Once the factory is built, say at time k, the widgets can be sold at the prevailing market price for such widgets. The future profits from widget sales depend on how the market price evolves, which is uncertain. Let S_k be the expected value of these future profits, discounted to time k, under a suitable market price model, probability measure, and discounting factor. Thus S_k represents the value of the asset that the firm can acquire by exercising its option to invest L at time k. Alternatively, one can think of S_k as the "benefit" from making the investment at time k, and L as the cost of the investment. In this respect, S_k is analogous to the price of a stock underlying an American call option, and L represents the strike price. Clearly the firm will not invest in the factory if $S_k < L$.

On the other hand, should the firm invest simply because S_k exceeds L? The traditional Net Present Value (NPV) rule recommends investing in this situation. However, this rule can often be suboptimal since it treats the decision problem as a now-or-never proposition. That is, if there is no possibility of delaying the decision, then the rule is indeed reasonable. However, if the decision to invest can be postponed, then the NPV rule ignores the value inherent in waiting for better information before making the investment. The option viewpoint is the natural framework in which to quantify the worth of the flexibility of being able to choose between investing now and any future time. Thus the value V_k represents the "value of the option to invest", or the opportunity cost of investing, at time k. The reason we think of V_k as an opportunity cost is that when we exercise the option, we lose the opportunity of being able to decide when to invest. In analogy with the above rule for an American call option, the optimal rule for the firm is (given suitable definitions of S_k and V_k):

> *Invest if the asset value (or benefit) S_k exceeds the direct cost of the asset L plus the opportunity cost V_k.*

This idea is at the heart of the theory of real options [5, 10].

The above approach applies equally well to any investment situation where (a) there is an expenditure of limited resources, (b) there is uncertainty over the future profitability of the investment, (c) the decision to invest is irreverrsible, and (d) the decision can be delayed. As we argued before, many software engineering design decisions, and prototyping decisions in particular, satisfy these criteria. In general suppose a software engineer is contemplating when (if at all) to invest in a prototype. In terms of the variables introduced above, the direct cost L is the cost of implementing the prototype. The future profit stream from prototyping is uncertain, and may depend on several factors, such as changes in requirements, hardware, usage-patterns, etc. This uncertainty can be modeled in terms of an event tree (see the Appendix for definitions of this and other terms). At any discrete time k, the *asset value* or *benefit* S_k is the expected value, discounted to time k, of the future (possibly negative) profit stream that would result if the prototype were already in place by time k. The option value V_k is then the opportunity cost of implementing the prototype or design-change – the value of the flexibility (which is lost when the prototype is implemented) of being able to decide when to implement the design. Just as in the capital investment scenario above, the optimal decision strategy for the software engineer is:

> *Invest in implementing the prototype when the benefit S_k is at least equal to the direct cost L plus the opportunity cost V_k.*

We have thus a rigorous way of quantifying when it is beneficial to delay our decision to invest in a prototype. In the next two sections we will present two different models for the effect of prototypes. In both sections we show how to compute the optimal timing of both the prototyping decision *and* the resulting design-change decision in the same event tree. As we will see, the above simple rule, which works when a prototyping (or design-change) decision is considered in isolation, no longer applies when both decisions are considered.

3 Prototyping as improving probability estimates

When a software designer is contemplating a major, expensive design change, he or she may be concerned about the wisdom of going ahead with the change, since the benefits of making the change may depend on uncertain future events. We consider a situation where only estimates of the probabilities of these future events are known. In such situations, as Boehm has argued, it may be possible to improve our estimate of these probabilities by building a *prototype* or *simulation* of the contemplated software design (or design change). In this section we will present a model based on the assumption that the principal effect of prototyping is in improving our probability estimates. In the next section we will present a different approach.

Although we are modeling prototypes as improving our estimates of probabilities of *future* events, essentially the same model applies to prototypes that are used to reveal information about the *current state*. In particular, we can view the prototype as being done "just before", the current state unfolds, and the same mathematics holds. For instance in Boehm's [1] example the prototype reveals information about whether or not the current state is favorable to a certain design change. Our models are more general and cover scenarios such as those in Boehm's paper.

We will henceforth use the word prototype to cover simulation as well. Although less expensive than changing the software, prototyping can still have a significant cost. Moreover, the expense incurred in prototyping cannot be recovered, [1] making a prototyping decision an irreversible one. However, the designer has the option to *wait* for better information before building a prototype. Thus a designer is faced with a two-stage decision:

1. When (if at all) to invest in a prototype, and
2. When (if at all) to *switch* to the new software (based on the results of prototyping), it being understood that this cannot occur *before* prototyping (if it is done).

We present our model through a concrete example that is easy to generalize. Suppose a database company X is contemplating whether to change the cache-coherence protocol on their top-of-the-line distributed DBMS. Since the future is unknown, it is unclear how this change will impact the company. It may be that in some scenarios the usage of the DBMS is such that the change is unfavorable. The company therefore wants to invest in prototying and simulation. It is faced with two decisions:

- Decision P: When (if at all) to invest in prototypes and simulation software? For simplicity, we assume that that prototypes and simulation software are built together and instantaneously. To allow for different levels of prototyping, we assume that the set $A = \{a_1, a_2, \ldots, a_r\}$ represents the possible amounts of money that can be invested in prototyping at any time.
- Decision D: when (if at all) to switch to the new cache-coherence protocol? We of course impose the restriction that prototyping can not occur after switching to the new protocol.

We formulate our problem in terms of an event tree of depth N (See Appendix A.1 for basic mathematical definitions). The branches of this event tree represent various future events that might have an impact on how much benefit the new cache coherence protocol would bring. Note

[1] Unless the prototype can be a part of the system being built. In the case of simulations (which are also covered by the term "prototyping" in this paper) the expense is always unrecoverable.

that for any path ω in the tree, $\omega^{(k)}$ represents the prefix of ω consisting of the first k branches. We use the more general symbol $\omega^{(i,k)}$ to denote the sub-path of ω consisting of its branches from time i and time k. The probability $\mathbf{P}(\omega^{(i,k)})$ is defined similarly: it is the product of the probabilities on the branches in $\omega^{(i,k)}$.

We assume that the branch probabilities are estimates of their true probabilities, and view prototyping as improving on them. For instance, it might be possible to estimate the probability that the usage patterns of the multiprocessor are such that the benefits of using the new protocol are minimal – we refer to this briefly as the "bad" scenario. However, in the absence of the actual implementation of the protocol, these probability estimates are crude at best. We propose that the effect of prototyping is to improve these estimates. In particular, if we decide to invest a_i dollars in prototyping at time k, on path $\omega^{(k)}$, then this changes the probabilities on all branches descending from $\omega^{(k)}$. We assume that these new probabilities are known. For instance in the absence of prototyping, we might consider the bad scenario as occuring with 50% probability. After implementing a small prototype, however, we may revise our estimate of this probability to, say, 30%.

In general, we assume that at any time there is a set A of possible *levels* of prototyping, i.e., the possible amounts of money we could spend prototyping. A specific prototyping strategy can then be described by a generalized stopping time τ_p which is a function from Ω to $\{1, 2, \ldots, N, \infty\} \times A$. That is, for any path $\omega \in \Omega$, $\tau_p(\omega)$ is of the form (i, a) where i denotes when to invest in a prototype along path ω, and $a \in A$ denotes the level of prototyping. We write $\tau_p^{(1)}$ to refer to the first component of τ_p, and $\tau_p^{(2)}$ to refer to the second component of τ_p, i.e., if $\tau_p(\omega) = (i, x)$, then $\tau_p^{(1)}(\omega) = i$ and $\tau_p^{(2)}(\omega) = x$. Since the prototyping decision is non-clairvoyant, we restrict τ_p so that for all $k = 0, 1, \ldots, N$, and all $x \in A$, the indicator random variable $I_{\{\tau_p=(k,x)\}}$ is $\mathcal{F}_k$-measurable. In other words, both the timing and level of prototyping are based only on what we have seen so far. We let S_p denote the set of all functions τ_p that satisfy this restriction. As mentioned before, investing in prototyping according to a generalized stopping time τ_p results in a new probability measure $\mathbf{P}_{\tau_p}$, i.e., a path ω has probability $\mathbf{P}_{\tau_p}(\omega)$. We write $\mathbf{E}_{\tau_p}$ to denote the expectation under this new measure.

We let τ denote the stopping time that describes the timing of the decision to invest in the new cache-coherence protocol. Since prototyping (if any) can never occur after switching to the new protocol, we impose the restriction:

$$\text{If } \tau_p^{(1)}(\omega) < \infty, \text{ then } \tau_p^{(1)}(\omega) \leq \tau(\omega) \tag{1}$$

In other words, whenever the prototyping time $\tau_p^{(1)}(\omega)$ is finite, this must be no greater than the switching time $\tau(\omega)$. Note that this restriction allows for the situation where prototyping is never undertaken ($\tau_p^{(1)}(\omega) = \infty$) but the new cache-coherence protocol is implemented ($\tau(\omega) < \infty$). Thus for a given execution τ_p of decision P, there is a set $S(\tau_p)$ of stopping times τ for decision D that satisfy the constraint (1).

If the decision D is made at time k, we define the **benefit** S_k of switching at time k to be the present (i.e. time-k) value of the future stream of benefits from switching at time k. Suppose the total **direct cost** of switching to the new implementation is L (including any scrapping costs). Since one would never switch if S_k is less than the direct cost, we can say that the **payoff** from switching at time k is

$$G_k = (S_k - L)^+,$$

which has exactly the same form as the payoff (18) from an American call option. If the decision D is made according to stopping time τ, then the payoff from this decision is G_τ and its present (time 0) value is G_τ / R^τ. (Note that when we say that the payoff is G_τ we mean that on any event tree path ω, if $\tau(\omega) = k$, the payoff is $G_k(\omega)$.) Similarly, if the prototyping decision is given by τ_p, the cost of prototyping is $\tau_p^{(2)}$, and its present value is $\tau_p^{(2)}/R^{\tau_p^{(1)}}$. Thus, given that we invest in prototyping according to the function τ_p, and invest in the new coherence protocol according to $\tau \in S(\tau_p)$, the present value of these decisions is

$$V_0(\tau_p, \tau) = \mathbf{E}_{\tau_p}\left[\frac{G_\tau}{R^\tau} - \frac{\tau_p^{(2)}}{R^{\tau_p^{(1)}}}\right]. \tag{2}$$

Therefore the present value of our investment opportunity is the maximum of this expression over allowable stopping times:

$$V_0 = \max_{\tau_p \in S_p} \max_{\tau \in S(\tau_p)} V_0(\tau_p, \tau). \tag{3}$$

Since S_p and $S(\tau_p)$ are finite sets, both $\max$'es above exist. A modification of the dynamic programming algorithm of Section A.2 can be used to compute V_0, as well as the optimal times τ_p^* and τ^* for decisions P and D respectively.

A reasonable question at this point is to ask, "what do the optimal strategies τ_p^* and τ^* look like"? For instance, the optimal exercise strategy in the case of a simple call option is easy to characterize (see Appendix A.2):

Exercise when the option value V_k equals the payoff from immediate exercise.

Is there an analogous rule when we are considering two strategies τ_p^* and τ^*? To answer this, we must first define what "the payoff from immediate exercise" means, and what the value of our "option", or investment opportunity (i.e., the opportunity to do prototyping and switching to a new implementation) means. To this end, we define, at any time k, for any allowable prototyping strategy $\tau_p \in S_p$ with $\tau_p^{(1)} \geq k$ and any allowable switching strategy $\tau \in S(\tau_p)$ with $\tau \geq k$, the value of these strategies at time k as

$$V_k(\tau_p, \tau) = R^k \mathbf{E}_{\tau_p}\left[\frac{G_\tau}{R^\tau} - \frac{\tau_p^{(2)}}{R^{\tau_p^{(1)}}}\middle| \mathcal{F}_k\right]. \tag{4}$$

Therefore, in analogy with definition (19) in Appendix A.2, we can define the value of our investment opportunity at time k as

$$V_k = \max_{\substack{\tau_p \in S_p \\ \tau_p^{(1)} \geq k}} \max_{\substack{\tau \in S(\tau_p) \\ \tau \geq k}} V_k(\tau_p, \tau). \tag{5}$$

Suppose we decide to exercise our option to do prototyping at time k at level $x \in A$. What is the payoff from this decision? This is simply the maximum expected payoff, discounted to time k, over all possible switching strategies $\tau \geq k$:

$$G'_k(x) = \max_{\tau \geq k} \mathbf{E}_{\tau_p} \left[G_\tau R^{k-\tau} \middle| \mathcal{F}_k \right] - x. \tag{6}$$

Note that this payoff itself is the value of an option, so in this sense when we are considering the two strategies τ_p and τ, we have an *option on an option.* Another point to note is that the first term in the payoff represents the "benefit" from prototyping at time k at level x, whereas the second term, x, of course represents the cost of prototyping. In analogy with our characterization of the optimal exercise strategy for a simple call option, we can say (and this can be proved rigorously) that the optimal prototyping strategy is to prototype at time $k \leq N$ at level $x \in A$ if and only if the payoff $G'_k(x)$ equals the option value V_k:

$$G'_k(x) = V_k.$$

As mentioned before, we can view V_k as the "opportunity cost" of prototyping at time k since this represents the option value that is lost when we decide to prototype at time k. Therefore in analogy with call options, the optimal prototyping strategy is:

> *Implement a prototype at time k at level x only if the benefit exceeds the prototyping cost x plus the opportunity cost V_k.*

To re-emphasize a point we have been making throughout this paper, previous work on prototyping that was based on the traditional NPV rule has ignored the opportunity cost V_k. The above rule shows that it may be better to postpone prototyping even when the traditional NPV rule recommends it.

Now, once prototyping has been done at some level x at time k, we must still decide when to switch to the new implementation. This consists of finding which τ maximizes the payoff $G'_k(x)$ defined in 6. At this point the problem is very similar to that of deciding the optimal exercise strategy of a call option, and this analogy has been explored at great length in our earlier paper [12]. We only note here that it may often be optimal to wait for a while after prototyping before switching to the new implementation (as we see in the example in the next subsection).

3.1 An example

We now illustrate the above ideas by means of the event tree shown in Fig 1. For this example, we assume the cost L of switching to the new software design is 100. We will assume a discount factor $R = 1.1$. We will refer to specific (partial) paths in the tree by enumerating the sequence of "up" (U) and "down" (D) branches to be followed in the tree, starting at the leftmost node. We will also use the partial-path notation to refer to the node at the end of the path. The benefit S_k is shown in italics next to each node in the tree. For example on the highlighted path UDU, $S_0 = 100, S_1 = 150, S_2 = 90, S_3 = 70$.

In the figure we show a specific prototyping strategy described by the generalized stopping time τ_p. The dark squares show the places where prototyping is done. In particular, on paths ω

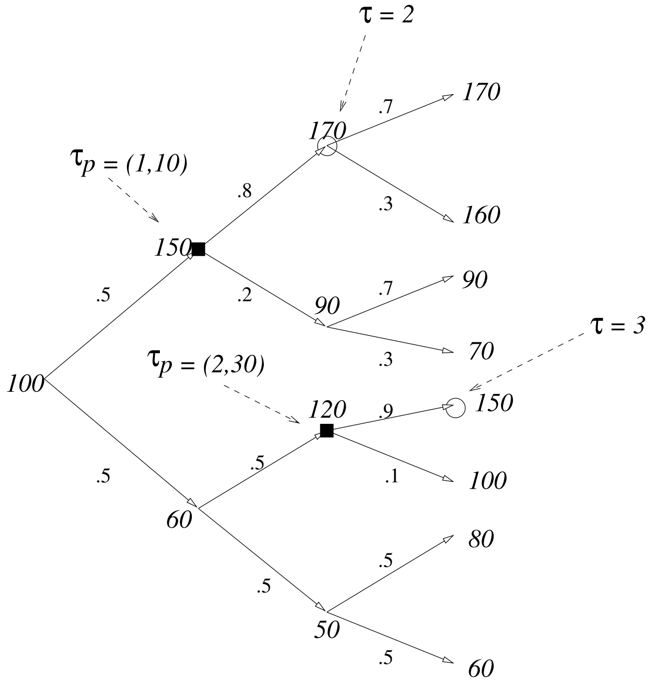

Figure 1: *An event tree showing the effect of prototyping decisions. The nodes of the tree are various states of nature. The dark boxes indicate prototyping decisions described by the generalized stopping time τ_p, while the circles indicate switching decisions described by the stopping time τ. When the value of a (generalized) stopping time is indicated at a specific node, this means that all paths that go through that node have that value of the stopping time.*

starting with U, $\tau_p(\omega) = (1, 10)$, meaning that the prototyping is done at time 1, and at "level 10", in the sense that 10 monetary units are spent in prototyping. (One unit could represent, say, one thousand dollars). Similarly, on paths starting with DU, $\tau_p = (2, 30)$, meaning that prototyping is done at time 2 and at level 30. Note that there is no prototyping on paths starting with DD. Recall that we have modeled prototyping as inducing a new probability $\mathbf{P}_{\tau_p}$ measure on the paths. The branch probabilities are shown alongside each branch. We have shown all branch probabilities to be 0.5 up to the time prototyping is done. This indicates our lack of knowledge about the relative likelihood of events in the absence of prototyping. However after prototyping, the probabilities are different from 0.5, indicating that we have a better idea of which outcomes are more likely to occur.

The figure also shows a specific switching strategy described by the stopping time τ. The circles show the points where switching is done. For instance on paths starting with UU, switching occurs at time 2. The switching strategy τ satisfies the constraint mentioned in the previous subsection: on paths where prototyping is done, the switching cannot occur before prototyping. Also, the payoff from switching at time k on any path is $G_k = (S_k - L)^+$, so there is no positive payoff from switching when $S_k \leq L$. Given these two observations, the only places where we need to consider switching are at nodes DUU, U, UU, UUU, and UUD.

If we consider various switching strategies (i.e., stopping times) τ, we see that the optimal strategy is given by the τ shown in the figure. This examples illustrates an important point, namely, that it may pay to wait for a while after prototyping. In fact in this example, it is optimal to wait one time period after prototyping, on paths starting with UU and DUU. We now illustrate the calculation of the expected discounted payoff (2) for the specific prototyping strategy τ_p and switching strategy τ:

$$\begin{aligned}
\mathbf{E}_{\tau_p}\left(\frac{G_\tau}{R^\tau}\right) &= \frac{(170-100)(0.5)(0.8)}{(1.1)^2} + \frac{(150-100)(0.5)(0.5)(0.9)}{(1.1)^3} \\
&= 23.14 + 8.45 \\
&= 31.59, \\
\mathbf{E}_{\tau_p}\left(\frac{\tau_p^{(2)}}{R^{\tau_p^{(1)}}}\right) &= \frac{10(0.5)}{1.1} + \frac{30(0.5)(0.5)}{1.1^2} \\
&= 4.54 + 6.20 \\
&= 10.74.
\end{aligned}$$

The expected present value of these specific prototyping and switching strategies is therefore $31.59 - 10.74 = 20.85$.

4 Prototypes as predictors

We consider a view of prototyping which is a multi-period extension of Boehm's single-period treatment [1].

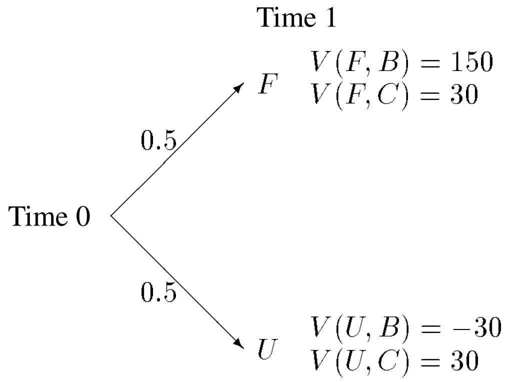

Figure 2: *Payoffs in Boehm's example*

4.1 Boehm's example

Let us begin by reviewing Boehm's example. In that example, we are required to choose between a bold (B) and conservative (C) approach in developing a certain special-purpose operating system. Since there is no operating system to start with, a choice between these two approaches must be made at the present time, which we call time 0. However, the profitability of the operating system will depend on the "state of nature" at time 1. There are two possible states of nature at time 1: "favorable" (F) and "unfavorable" (U), each occuring with probability 0.5:

$$\mathbf{P}(U) = \mathbf{P}(F) = 0.5.$$

In general we denote the payoff of approach a in state s by $V(s, a)$. In Boehm's example (see Figure 2),

$$V(F, B) = 150,\ V(U, B) = -30,\ V(F, C) = V(U, C) = 30.$$

In the absence of any information about the future state at time 1, we would choose either approach B or approach C at time 0. In the bold case, the expected payoff would be

$$P(F)V(F, B) - P(U)V(U, B) = (0.5)(150) - (0.5)(30) = 60,$$

and in the conservative case the expected payoff is 30. Can we do better if we have an idea of what the next state at time 1 will be? Boehm considers the idea of building a rough prototype of the key portions of the bold approach, in order to get information about the time-1 state. First consider a *perfect* prototype, namely one that will predict exactly what the next state will be. Let us say the cost of building this prototype is $C_p = 10$. If the prototype tells us that the next state will be favorable, we would decide on approach B, and otherwise we would pick approach C. The expected net payoff in this case would be

$$\mathbf{P}(F)V(F, B) + \mathbf{P}(U)V(U, C) - C_p = (0.5)(150) + (0.5)(30) - 10 = 80,$$

which is better than the payoffs without prototyping.

One way to view the effect of the perfect prototype in this example is that it gives us a way of making our decision at time 0 be *contingent* on the next state. To see this, we let $\mathbf{P}(B|U)$ denote the conditional probability that we pick the bold approach given that the next state is U. We define $\mathbf{P}(C|U), \mathbf{P}(B|F), \mathbf{P}(C|F)$ similarly. In the absence of a prototype, our decision cannot be contingent on the next state, so we would either have

$$\mathbf{P}(C|U) = \mathbf{P}(C|F) = 1, \;\; \mathbf{P}(B|U) = \mathbf{P}(B|F) = 0, \tag{7}$$

which corresponds to picking choice C, or

$$\mathbf{P}(C|U) = \mathbf{P}(C|F) = 0, \;\; \mathbf{P}(B|U) = \mathbf{P}(B|F) = 1, \tag{8}$$

which corresponds to picking choice B. Now with a perfect prototype, we will pick approach B if the state is favorable, and approach C otherwise (i.e. our decision is contingent on the next state), so that

$$\mathbf{P}(C|F) = 0, \; \mathbf{P}(C|U) = 1, \;\; \mathbf{P}(B|F) = 1, \; \mathbf{P}(B|U) = 0.$$

Note that for both prototypes we have

$$\mathbf{P}(B|F) + \mathbf{P}(C|F) = 1, \;\; \mathbf{P}(B|U) + \mathbf{P}(C|U) = 1. \tag{9}$$

Thus our expected payoff with the perfect prototype can be written

$$\begin{aligned} &\mathbf{P}(U)\left[\mathbf{P}(B|U)V(U,B) + \mathbf{P}(C|U)V(U,C)\right] \\ &\qquad\qquad + \; \mathbf{P}(F)\left[\mathbf{P}(B|F)V(F,B) + \mathbf{P}(C|F)V(F,C)\right] - C_p, \end{aligned} \tag{10}$$

which is, as computed before,

$$(0.5)[1(150) + 0(30)] + (0.5)[0(-30) + 1(30)] - 10 \; = \; 80.$$

With this notation we can consider more generally an *imperfect prototype,* one where the only restriction is that (9) be satisfied. In particular, *not* using a prototype can be viewed as using a completely uninformative prototype, in which case the conditional probabilities would be given by either (7) or (8), and the corresponding payoffs would again be given by (10), as can be easily verified. Let us consider the imperfect prototype (costing 10) in Boehm's example, characterized by the conditional probabilities

$$\mathbf{P}(C|F) = 0.1, \; \mathbf{P}(C|U) = 0.8, \;\; \mathbf{P}(B|F) = 0.9, \; \mathbf{P}(B|U) = 0.2. \tag{11}$$

Note that condition (9) is satisfied. Now the expected payoff, from (10), is

$$\begin{aligned} &(0.5)[(0.9)(150) + (0.1)(30)] + (0.5)[(0.2)(-30) + (0.8)(30)] - 10 \\ &\qquad\qquad = 78 - 10 = 68. \end{aligned} \tag{12}$$

Thus the effect of any prototype can be modeled by the conditional probabilities of choosing various alternatives, in each of the possible *next-states.* We do not make explicit in what form we obtain the results of prototyping (e.g, state-prediction, etc) are, nor how they lead to our decision.

4.2 A multi-period framework

As we just saw above, Boehm's example (as well as his generalization) considers only one time-period: the prototyping decision and the choice between bold/conservative must be done at the present time, since there is no software to start with.

We now extend this in two ways: (a) we consider multiple time periods, in our event-tree framework, and (b) allow the prototyping and implementation decisions to take place at any time, so that both the decision to prototype and the implementation decision can be delayed. In particular the implementation-change does not have to take place immediately after prototyping – the software engineer can wait until conditions (the "state of the world") are appropriate for this. We assume that some implementation is already in place, and we are considering when (if at all) to *switch* to a specific, different one (and in subsection 4.4 we generalize this to switching to one of several alternatives). We therefore refer to the implementation decision as the *switching decision*. The ability to delay both the prototyping and switching decisions can lead to increased payoffs for the software engineer.

We will be able to use much of the notation introduced in Section 3. In fact it turns out that just as prototyping was viewed in the previous section as changing (or determining) the future probability *measure*, in this section we can view prototyping as changing the future *payoffs*. As before we let τ_p denote the generalized stopping time for the prototyping decision, so that $\tau_p^{(1)}$ is the (ordinary) stopping time for the prototyping decision, and $\tau_p^{(2)}$ is the level (or cost) of prototyping used. We describe the timing of the switching decision by the stopping time τ, although as we will point out next there is an important difference in what it means to make a decision.

In the multi-period setting we assume that after doing a certain level of prototyping at time m, we can observe our path on the event tree, and make the switching decision at some time $k \geq m$. When making this decision, we are using the prototype to give us some indication of what the *next* (i.e. time-$(k+1)$) state will be. This indication may be in the form of an explicit state-prediction, but not necessarily so. In any case, as in the numerical example of the previous subsection, we model the effect of the prototype as follows. For each time-$(k+1)$ state α that succeeds the current state, we have a *conditional probability* of having switched to the new implementation (at time m) given that the *next* state is α. Thus for any given prototyping decision τ_p, for $k > \tau_p^{(1)}$, we can define the random variable $\mathbf{P}_k(\tau_p)$ on any path $\omega^{(k)}$ to be the conditional probability of having switched to the new implementation at time $k-1$ given that we are on the path (or state) $\omega^{(k)}$ at time k. If $k \leq \tau_p^{(1)}$, we define $\mathbf{P}_k(\tau_p) = 1$: this represents the fact that, since there has been no prototype strictly before time k, we are unable to use any state-prediction information to have conditional probabilities other than 1. (This corresponds to the uninformative prototypes described by (8), (7).)

Note that (in contrast to Section 3) when we speak of making a "switching decision" at time k, we are not necessarily switching since, depending on the probabilities $\mathbf{P}_{k+1}(\tau_p)$, there is some chance that we do not switch. In fact we are deciding whether to switch or not to switch, and if we happen to decide *not* to switch, we are committed to this decision and cannot undo it subsequently. As before, for a given prototyping decision τ_p, there is a set $S(\tau_p)$ of allowable stopping times τ, defined by the restriction that if prototyping is done at all then it must precede the switching decision.

For $k \geq 1$, S_k will denote the benefit at time k (and in state $\mathcal{F}_k$) from the new implementation, if the switching decision was made at time $k-1$. Of course, as before S_k represents the present (i.e. time-k) value of all future benefits from the new implementation. We continue to use L to denote the direct cost of switching. We will assume a discount factor of R per time period.

Noting that a switching decision made at time k translates into a probabilistic outcome at time $k+1$, we define, for a given τ_p, the **payoff** $G_k(\tau_p)$ at time k from this decision to be the expected payoff from time $k+1$, given the current state (described by $\mathcal{F}_k$):

$$G_k(\tau_p) = \left[\mathbf{E}\left(S_{k+1}\mathbf{P}_{k+1}(\tau_p)/R|\mathcal{F}_k\right) - L\right]^+, \quad k \geq 0. \tag{13}$$

(The switching decision, and hence the expenditure of L, is made at time k, whereas the benefit S_{k+1} applies at time $k+1$, which is why we are discounting S_{k+1} to time k by the factor R).

From the above description it is clear that the effect of a prototyping decision τ_p is captured in the payoff function $G_k(\tau_p)$. (This is in contrast to the prototyping model of the previous section, where τ_p determines the probability measure $\mathbf{P}_{\tau_p}$.) Thus, if we invest in prototyping according to the generalized stopping time τ_p, and invest in switching according to $\tau \in S(\tau_p)$, the present value of these decisions is given by an expression analogous to (2):

$$\mathbf{E}\left[\frac{G_\tau(\tau_p)}{R^\tau} - \frac{\tau_p^{(2)}}{R^{\tau_p^{(1)}}}\right]. \tag{14}$$

Analogously, the value of our investment opportunity is the maximum of this expression over allowable stopping times:

$$V_0 = \max_{\tau_p \in S_p} \max_{\tau \in S(\tau_p)} \mathbf{E}\left[\frac{G_\tau(\tau_p)}{R^\tau} - \frac{\tau_p^{(2)}}{R^{\tau_p^{(1)}}}\right]. \tag{15}$$

4.3 Boehm's example revisited

As an illustration, we now analyze Boehm's one-period example of Subsection 4.1 in the model just introduced. We will interpret that example from the viewpoint of switching from the conservative to the bold approach. That is, we assume that the conservative approach (C) is already in place, and we are required to decide whether to switch to approach B. Our event tree is simple: the time-0 state has two successor states at time 1: favorable(F) and unfavorable (U). We want to define S_1, the benefit from swiching at time 0, for each time-1 state. For state F, $S_1 = 150 - 30 = 120$. For state U, $S_1 = -30 - 30 = -60$. The direct cost L of switching is 0. Note that Boehm did not consider the cost of switching. In this simple event tree, we cannot delay the prototyping and switching decision, i.e., we must make them at time 0. We will take the discounting factor R to be 1.

Consider the imperfect prototype at the end of Subsection 4.1, whose conditional probabilities are given by (11). The cost of this prototype is 10. Since we are using this prototype at time 0, our prototyping decision is described by $\tau_p = (0, 10)$. The effect of this prototype is characterized by the probabilities $\mathbf{P}_1(\tau_p) = 0.9$ in state F and $\mathbf{P}_1(\tau_p) = 0.2$ in state U. Thus the payoff G_0 from using this prototype and making the switching decision at time $\tau = 0$ is, from the definition (13):

$$G_0(\tau_p) = \mathbf{E}(S_1\mathbf{P}_1(\tau_p)/R) - L = (0.5)(120)(0.9) - (0.5)(60)(0.2) = 48.$$

The net payoff, taking into account the cost of prototyping is, from (14):

$$\mathbf{E}\left[G_0(\tau_p)/R^0 - \tau_p^{(2)}/R^{\tau_p^{(1)}}\right] = 48 - 10 = 38.$$

Since the benefit of the conservative approach is 30, we can conclude that with the above imperfect prototype at time 0, the payoff from choosing an implementation at time 0 is $30 + 38 = 68$, which matches the value obtained in (12).

4.4 Switching to one of several alternatives

In the previous subsections we have assumed that there is some software implementation already in place, and we considered when (if at all) to switch to a specific different implementation. We now show how this can be easily generalized to the case of switching to one of m possible alternatives. Recall that $\mathbf{P}_k(\tau_p)$ on a path $\omega^{(k)}$ was defined as the conditional probability of having switched to the new implementation at time $k-1$ given that we are on the path $\omega^{(k)}$ at time k. Generalizing this, we define $\mathbf{P}_k^i(\tau_p)$ to be the conditional probability of having switched to the i'th alternative at time $k-1$ given that we are on the path $\omega^{(k)}$ at time k. Next, recall that $G_k(\tau_p)$ was defined as the payoff from making a switching decision at time kn, and was calculated as the expected payoff from time $k+1$, given the current state $\mathcal{F}_k$. When there are m alternatives to choose from, we should generalize this to be the *maximum* of the quantity analogous to $G_k(\tau_p)$ over all alternatives:

$$G_k(\tau_p) = \max_{1\leq i\leq m} \left[\mathbf{E}\left(S_{k+1}\mathbf{P}_{k+1}^i(\tau_p)/R|\mathcal{F}_k\right) - L\right]^+, \quad k \geq 0. \tag{16}$$

With these definitions, the present value of a specific prototyping and switching strategy is given by (14), and the value of our investment opportunity is given by equation (15).

5 Conclusion

We began this paper by pointing out the shortcomings of previous economics approaches to prototyping decisions, focusing in particular on Boehm's approach. Boehm recognizes that prototyping can involve significant expenditure, and that the benefits from prototyping depend on uncertain future events. He therefore applies the classical economics approach to decision-making under uncertainty, namely Net Present Value (NPV) analysis: invest in prototyping only if the benefits (on average) outweigh the costs. One problem with Boehm's approach is that he views prototyping decisions and implemention decisions as occuring simultaneously. It is more realistic to separate these decisions in time. Moreover, he does not consider the possibility of *delaying* the prototyping (or implementation) decision until better information becomes available. As a result, Boehm's approach may lead to sub-optimal prototyping decisions.

In this paper, we presented two different models that address both these limitations: prototyping and implementation decisions that depend on prototyping are separated, and can occur at any time. In allowing the decisions to occur at any time, we can no longer use the classical NPV analysis. We must instead appeal to a more advanced but well-established theory, namely that of financial call options. The basic ideas of this approach to software decisions were introduced by the present authors in a different paper [12]. In a sense the present paper extends that approach to software prototyping decisions. However, prototyping adds two novel twists: Firstly, the prototyping decision can change our estimates of the probabilities of future events, or change the future payoffs (depending on which of the two models we use). Secondly, we are now considering *two* decisions (prototyping and design-change) rather than just one. The models presented in the present paper are therefore significantly more complicated.

The combination of prototyping and design-change decisions is a special kind of *sequential investment* scenario studied in financial contexts [5], since the prototyping cost and the cost of switching to a different implementation are, after all, investments with an uncertain profitability. A firm may invest in a large factory stage by stage, and may want to time its decisions in order maximize the expected payoff. Withey [13] considers a sequential investment model from real options theory to analyze strategies for building reusable software modules. In a future paper we plan to further explore the connection between sequential investment and prototyping.

References

[1] B. W. Boehm. Software engineering economics. In *IEEE Transactions on Software Engineering (SE)*, volume 10, Jan 1984.

[2] R. Brealy and S. Myers. *Principles of Corporate Finance*. McGraw-Hill, 1992.

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

[4] G. Daily. Beyond puts and calls: Option pricing as a powerful tool in decision-making. *The Actuary*, 30(3), 1996.

[5] A. Dixit and R. Pindyck. *Investment under uncertainty*. Princeton Univ. Press, 1994.

[6] A. Dixit and R. Pindyck. The options approach to capital invesment. *Harvard Business Review*, pages 105–115, May-June 1995.

[7] J. Flatto. *The application of real options to the information technology valuation process: A benchmark study*. PhD thesis, Univ. of New Haven, 1996.

[8] J. Hull. *Options, Futures, and Other Derivative Securities*. Prentice Hall, 2nd edition, 1993.

[9] J.Sick. Real options. In *Handbook of Finance*. Elsevier/North-Holland, 1995.

[10] R. McDonald and D. Siegel. The value of waiting to invest. *Quarterly Journal of Economics*, 101:707–727, 1986.

[11] K. Sullivan. Software design: the options approach. In *Proc. Software Architectures Workshop*, 1996.

[12] K. Sullivan, P. Chalasani, and S. Jha. The options approach to software design. Technical report, University of Virginia, Computer Science Dept., 1997. Also submitted to IEEE Trans. Soft. Engg.

[13] J. Withey. Investment analysis of software assets for product lines. Technical Report CMU/SEI-96-TR-010, Carnegie Mellon University – Software Engineering Institute, Nov. 1996.

A APPENDIX

A.1 Mathematical Background

In this section we introduce some standard mathematical terminology and notation that is used in the paper. Much of this material appears in our earlier paper [12]. For simplicity of exposition, we will model future uncertainty by means of a discrete **event tree** of finite depth N, where N represents the maximum number of future time steps (e.g. months, years, etc) that we wish to model. We will often take N to be so large that we can treat it as essentially infinite. In much of the following, certain tedious technical conditions and definitions will be omitted. Each node in the tree represents a state of the world. The root node is considered to be at depth 0 and represents the present time, i.e., time 0. A node at depth k represents a state of the world at time k; its children are the possible next-states at time $k+1$. A typical path in this tree from the root to a leaf (i.e. from time 0 to time N) is denoted by the letters α or ω. We write $\omega^{(k)}$ to denote the prefix of ω consisting of the states from time 0 to time k; $\omega^{(0)}$ represents the empty path. The collection of tree paths ω is called the **sample space** Ω. For our purposes a **random variable** Z is a mapping (or function) that associates with each $\omega \in \Omega$ a real number $Z(\omega)$. A random **process** is a sequence of random variables such as $\{Z_n\}_{n=0}^{N}$, which we will sometimes denote simply by Z_n.

As an illustrative example, it is useful to have the following simple event tree, called the **binomial tree**, in mind. Imagine we toss a coin N times. Each non-leaf node in this tree has two children. If we imagine the tree branching left to right, each of the 2^N paths represents a particular sequence of coin-toss outcomes. On any path ω, for $k = 1, \ldots, N$, the k'th branch is an up-branch if the k'th coin-toss comes up heads (H), and it is a down-branch if it comes up tails (T). For future reference we define the following random variables on this tree: For $k = 1, 2, \ldots, N$, we define $X_k = 1$, if the k'th coin-toss is a H and $X_k = -1$ otherwise. We refer to X_k as the **random walk process**. In fact, X_k can be viewed as representing a particle that starts at the origin and performs a random walk on the x axis: at time k it moves 1 unit to the right if $X_k = 1$ and 1 unit to the left otherwise. The *position* of the particle at time k is given by the random variable Y_k defined by $Y_0 = 0$, and

$$Y_k = \sum_{i=1}^{k} X_i.$$

With each branch in an event tree, we associate a probability, so that the sum of the probabilities of the branches emanating from a given node is 1. For instance if we have a fair coin in the coin-toss example above, the probability of each branch is 0.5. For any k, and any path ω, the probability of the k-prefix $\omega^{(k)}$, denoted $\mathbf{P}(\omega^{(k)})$, is computed in the obvious way: multiply the probabilities of the k branches in $\omega^{(k)}$. We define the probability $\mathbf{P}(\omega)$ of the entire path ω in the same way. The **expectation** of a random variable X, denoted by $\mathbf{E}(X)$, is defined as

$$\mathbf{E}(X) = \sum_{\omega \in \Omega} X(\omega)\mathbf{P}(\omega). \tag{17}$$

We will need to use the generalized random variable $\mathcal{F}_k$, defined as follows. For any path $\omega \in \Omega$, $\mathcal{F}_k(\omega) = \omega^{(k)}$. In other words, the function $\mathcal{F}_k$ associates with any path $\omega \in \Omega$ its k-prefix $\omega^{(k)}$. It will be useful to think of $\mathcal{F}_k$ as representing the "information up to time k". Thus $\mathcal{F}_k$ represents a "state of the world" at time k. A random variable X is said to be $\mathcal{F}_k$**-measurable** if for any path $\omega \in \Omega$, $X(\omega)$ only depends on $\omega^{(k)}$. More precisely, X is $\mathcal{F}_k$-measurable if for any pair of paths $\omega, \alpha \in \Omega$, $\omega^{(k)} = \alpha^{(k)}$ implies that $X(\omega) = X(\alpha)$. For instance in the coin-toss example, if the random variable H_k represents the number of heads up to time k, then H_k is $\mathcal{F}_k$-measurable, for $k = 1, 2, \ldots, N$. Similarly, the random variable Y_k (the number of heads minus the number of tails by time k) is $\mathcal{F}_k$-measurable. A random process $\{X_k\}_{k=0}^{N}$ is said to be **adapted** if for $k = 0, 1, 2, \ldots, N$, X_k is $\mathcal{F}_k$-measurable.

The concept of conditional expectation is an important one for this paper. Let us imagine we are in a particular state of the world at time k, represented by the value of the random variable $\mathcal{F}_k$. In other words, we are on some path ω and we know $\mathcal{F}_k(\omega) = \omega^{(k)}$. Now suppose we want to compute the expectation of some random variable X *given* that we are in state $\mathcal{F}_k$. Clearly this expectation will in general be different from $\mathbf{E}X$, and will depend on $\omega^{(k)}$. For example, in our coin toss example, suppose X is the random variable Y_n. If $\omega^{(k)}$ consists only of heads then the expectation of Y_n given $\omega^{(k)}$ will be higher than if $\omega^{(k)}$ contained only tails. We compute the expectation of X given $\omega^{(k)}$, in a manner similar to expression (17): the difference is that we take the weighted sum of $X(\alpha)$ only over paths α such that $\alpha^{(k)} = \omega^{(k)}$, and we only weight each term $X(\alpha)$ with the product of the probabilities of the branches of α that are taken *after* time k. This probability-product is simply $\mathbf{P}(\alpha)/\mathbf{P}(\omega^{(k)})$. Then the **conditional expectation** of X given $\mathcal{F}_k$ is denoted $\mathbf{E}(X|\mathcal{F}_k)$ and is defined as the ($\mathcal{F}_k$-measurable) random variable that maps any path $\omega \in \Omega$ to

$$\frac{\sum_{\substack{\alpha \in \Omega \\ \alpha^{(k)} = \omega^{(k)}}} \mathbf{P}(\alpha) X(\alpha)}{\mathbf{P}(\omega^{(k)})},$$

which is just a form of the familiar Baye's rule. The conditional expectation $\mathbf{E}(X|\mathcal{F}_k)$ can be thought of as the expectation of X given that we have all the information up to time k, or given the state of the world at time k.

For a random variable X, it is customary to write $\{X = x\}$ to denote the set of paths ω such that $X(\omega) = x$. For any set of paths A, I_A is a random variable called the **indicator function** for A, and is defined as

$$I_A(\omega) = \begin{cases} 1 & \text{if } \omega \in A, \\ 0 & \text{otherwise.} \end{cases}$$

A **stopping time** τ is a random variable taking integral values in the range $[0, N]$, such that for each $k = 0, 1, \ldots, N$, $I_{\{\tau = k\}}$ is $\mathcal{F}_k$-measurable. In the coin-toss tree, an example of a stopping time is

$$\tau(\omega) = \begin{cases} \min\{k \geq 0 : \omega^{(k)} \text{ contains 3 Heads}\} & \text{if such a } k \text{ exists,} \\ N & \text{otherwise.} \end{cases}$$

This stopping time can be viewed as specifying the following rule: stop when the coin has landed heads 3 times. Note that if we happen to stop at time k on a path ω, i.e., $\tau(\omega) = k$, then for *any* path α with $\alpha^{(k)} = \omega^{(k)}$, we have $\tau(\alpha) = k$. Thus a stopping time is a non-clairvoyant decision rule of when to stop, and in this sense models real-world decisions that must be made in the absence of information about the future. An example of a rule ρ that is clairvoyant (and therefore not a stopping time) is: stop the first time that the *next* coin-toss will land heads. While this rule defines a random variable ρ, the indicator random variable $I_{\{\rho = k\}}$ is not $\mathcal{F}_k$-measurable. We remark that "stopping" is just a convenient metaphor that could represent any action for which we are studying decision rules.

A.2 Options

We now describe some basic concepts in option theory. For further details the reader is referred to Hull's [8] excellent introductory text.

The simplest kinds of options are call options. An **American call option** on a certain stock is a financial contract with the following features: it gives the holder of the contract the *right* but not the obligation to buy a share of the stock at a fixed price called the **strike** (or **exercise**) price L from the writer (i.e. seller) of the contract, on or before a certain **expiration** date of T time units. The holder thus has the "option" of deciding whether or not to exercise the contract, i.e., demand a share of stock at the strike price L from the contract writer. This is why the contract is called an option. When the option is exercised or the option expires, the option ceases to exist. Thus option exercise is *irreversible*.

In order to discount future cash flows to the present time, we will need to assume that money can be borrowed or lent (for example, via a bank or government bond) at a risk-free interest rate of r. Thus a dollar lent or borrowed at (discrete) time k is worth $R = 1 + r$ dollar 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 (i.e. time 0) is worth $1/R^k$.

Now let us return to the description of the American call option in terms of the binomial stock-price model. It is clear that the holder should not exercise the option at time $k \leq N$ if $S_k \leq L$. On the other hand, if $S_k > L$, the holder can (but is not obligated to) exercise the option, and if she does, the option writer is obligated to sell her a share of stock at the strike price L. The holder could then immediately sell the share in the market at S_k, and make a profit of $S_k - L$. Thus the profit that can be realized by exercising the American call option at time k is $\max(S_k - L, 0)$, which we refer to as the **payoff** G_k from the option. It is standard notation to write for any real number x, x^+ for $\max\{x, 0\}$, so we can write the payoff as:

$$G_k = (S_k - L)^+. \tag{18}$$

Since the option holder never takes a loss from exercising the option, it has a positive value for the holder, and it is therefore not surprising that she must pay a certain price to the seller for owning the option. Trillions of dollars worth of this and many other kinds of options are traded daily in world financial exchanges [8].

What is the best exercise strategy for the holder of an American call option, if she is still holding it at time k? As mentioned in the previous section, any (non-clairvoyant) exercise strategy can be described by a stopping time τ. In particular when we say that the holder of the option is following an exercise strategy τ, we mean: on any path ω on the event tree, if $\tau(\omega) = k$, then k is the time when the option is exercised. For instance the strategy of exercising immediately (at time k) is defined by the constant stopping time $\tau = k$. As another example, the strategy: "exercise when the stock price exceeds a certain threshold λ", is described by the stopping time

$$\tau = \min\Big(\{N\} \cup \{m \in [k, N] : S_m \geq \lambda\}\Big).$$

If the holder of the option exercises immediately, she will obtain a payoff $(S_k - L)^+$. If she exercises at some future time $m > k$, the payoff would be $(S_m - L)^+$, which is worth $(S_m - L)^+/R^{m-k}$ at time k. Depending on the value of S_m, this could be smaller or bigger than the payoff $(S_k - L)^+$ from immediate exercise. The stock price S_m at the future time m is of course uncertain and cannot be predicted. For a given exercise strategy (i.e. stopping time) $\tau \geq k$, the expected present value $V_k^{(\tau)}$ can be computed as follows. The payoff upon exercise at time τ is $(S_\tau - L)^+$, which is worth $(S_\tau - L)^+ R^{k-\tau}$ at time k. Therefore the expected present value of the payoff from this exercise strategy, given the information up to time k is

$$V_k^{(\tau)} = \mathbf{E}\left((S_\tau - L)^+ R^{k-\tau} \middle| \mathcal{F}_k \right), \quad \tau \geq k.$$

Our option holder would of course want to choose τ so that this expectation is maximized. We denote this maximum by V_k:

$$V_k = \max_{\tau \geq k} V_k^{(\tau)}. \tag{19}$$

Thus the value V_k, which we loosely refer to as the "value of the option at time k", is the best expected present value at time k realizable over all possible exercise strategies. We note in passing that in option pricing theory, the "fair value", or fair trading price, of an option is defined by assuming the absence of arbitrage, i.e., the fair value of the option is the value at which it can be traded so that there are no opportunities for unlimited riskless profit. More specifically, the fair value is defined as the value of a dynamically updated portfolio that replicates the random fluctuation of the option payoff G_k. The replicating portfolio consists of the underlying asset, namely the stock, and risk-free bonds. It turns out that this arbitrage-free fair value of an American call option is defined exactly as V_k above, but under a specific artificial probability measure (called the risk-neutral measure) that is not necessarily the "actual" probability measure. However in the case of real options, as we will see, the underlying "asset", which represents expected future profits, only exists as a result of exercising the option, and is not traded independently. This makes the option-replication approach to valuation less compelling for real options. Also, since real options themselves are not traded either, we are not so much concerned with their fair value, as with how to exercise them optimally, in the sense of maximizing the expected discounted payoff under the actual probability measure. For the purpose of determining the optimal exercise policy, therefore, we can take the option value V_k to be defined as above.

Since immediate exercise is a valid strategy at any time, V_k must be at least as large as $(S_k - L)^+$. In fact, if $(S_k - L)^+ < V_k$, this means that the immediate exercise strategy is *not* optimal, and that some other strategy will yield a strictly greater expected present value of payoff, under our assumed stock price model. Thus in this situation, it is beneficial to not exercise and wait. On the other hand, if $(S_k - L)^+ = V_k$, then there is nothing to be gained in waiting, at least under our assumed stock price model. In this case it *is* optimal to exercise immediately. Indeed it can be shown rigorously that the stopping time τ that acheives the maximum in (19) above is given by

$$\tau = \min\Big(\{N\} \cup \{m \in [k, N] : V_m = (S_m - L)^+\} \Big). \tag{20}$$

Let us look at the optimal exercise rule from a cost-benefit viewpoint. We can think of the strike price L as the "cost" of exercising the option, since this is the price one must pay to obtain a share of stock. Similarly, S_k is the benefit from exercising at time k, since this is the price one would obtain by selling the stock in the market. We just remarked above that it may not be optimal to exercise as soon as the benefit S_k exceeds the cost L. To explain this, it will be useful to view the option value V_k as representing the "value of the choice to exercise". When the option is exercised, the option (and the choice) is killed and this value is lost, so that V_k represents the *opportunity cost* of exercising the option. Thus when the option is exercised, there are two costs: the *direct* cost L, and the opportunity cost V_k. From the discussion above, the optimal exercise strategy is to exercise when $(S_k - L)^+ = V_k$, which in cost-benefit terms can be stated as:

> *Exercise only when the benefit S_k equals (or exceeds) the direct cost L plus the opportunity cost V_k.*

This viewpoint is the one that we will find most useful in this paper.

The value V_k can be computed for all k by a simple *dynamic programming* procedure (see [3]) as follows. First observe that $V_N = (S_N - L)^+$. This is clear both from formula (19) and from observing that the since option expires at time N, there is no advantage to waiting. Now stepping backward in time on the binomial tree, we compute V_k in any state of the world (given by $\mathcal{F}_k$) using the formula

$$V_k = \max\{(S_k - L)^+, \mathbf{E}(V_{k+1}|\mathcal{F}_k)/R\}. \tag{21}$$

In other words, the option value V_k on a path ω is the maximum of the immediate payoff $(S_k(\omega) - L)^+$ and the expected present value of the option value one time step ahead, given that we have seen the prefix $\mathcal{F}_k(\omega) = \omega^{(k)}$ so far. It can be shown that this backward-recursive formula for V_k and formula (19) are equivalent. And this is true regardless of the specific process that the stock price S_k follows – that is, even if S_k does not follow the binomial model assumed here. This fact will be useful in our application to software engineering decisions, since the analog of S_k in those applications does not necessarily follow the binomial model.