Financial Decisions, Spring 2005, Article 2

Some Evidence Concerning the Economic Value of Software Portability: A Real Options Approach

Dean L. Johnson, Brent J. Lekvin and James E. Northey*
Michigan Technological University, Michigan Technological University, and
Jordan & Jordan, respectively

Abstract

Software development typically involves a large capital outlay for an asset with a highly uncertain useful life. A reduction in the degree of uncertainty is likely to have a significant impact upon the expected value of an application. One method for reducing uncertainty is to incorporate modularity (e.g. a portability layer) when the application is first developed. The inclusion of such a layer involves an additional development cost. Using a real options approach we estimate the value of the flexibility that such portability confers. We use sensitivity analysis to examine the relationship between value of portability and changes in factors such as the probability that the application will have to be rewritten, expected application life, and the volatility of future redevelopment costs. Evidence is presented regarding the fundamental requirements necessary to make the additional investment in software design a positive net present value project.

I. Introduction and Background

The cost of writing and rewriting software applications typically represents a major capital expense. This is true whether the application is custom developed within the company, custom developed using external resources, or purchased from an independent software vendor (ISV). Worldwide expenditures on software are estimated at approximately USD $800 billion as of 1999.[1] To put this into context, expenditures on software represented 17.8% of all private fixed investment in equipment and software in the United States during 1999, versus 17.1% for industrial equipment, and 19.5% for transportation equipment during the same year.[2] Moreover, such applications often have a highly uncertain useful life, and software failure can be costly. The cost in terms of lost productivity relating to software failure has been estimated at $85 billion in the United States alone during 1998.[3] One specific example is Gateway's $45 million write-off for the abandonment of a software project in 1997. Another example is the London Stock Exchange's £80 million write-off relating to the abandonment of the Taurus software project in 1993. Consequently, these software investments can be considered a high-risk use of

* We have benefited greatly from comments made by participants at the 2003 Midwest Finance Association Conference. Additional comments by Andrew Jansma were also helpful. Finally, we thank Heath S. Johnson for assistance on this project.

[1] Boehm and Sullivan (1999).
[2] U.S. Department of Commerce, Bureau of Economic Analysis.
[3] Business Week (1999)

capital.[4] The objective of this research is to show the value of a real-options approach in evaluating methods that may reduce the failure rate of software projects.

In order to place this research into context, we begin with the work of Parnas (1972) relating to the information hiding approach to software modularity. Information hiding refers to design features hidden in the software to make it more flexible should changes be required during the life of the software. In this paper we refer to this type of design as a portability layer. Parnas' work has been influential in computer science and software design. Success in this realm, however, is typically defined in terms of functionality of the software rather than based upon some metric for estimating the economic value of the project. Most entities that procure software have traditionally not employed financial analysis. Instead, biases based upon past experience of the application architect or those responsible for project management are used in decision-making.

Boehm and Sullivan (1999) make a strong case concerning the need for economics-based thinking with respect to software development. They refer to this as *software economics*, and assert that "most software designers today make design decisions in an economics-independent 'Flatland'."[5] At least three reasons are given for this lack of financial analysis during the development process. First is a lack of explicit links between design, cost, and value. Second is a lack of training among software engineers with the relevant tools necessary to understand how the software parameters might be manipulated for value creation. Third is that software development was once largely driven by the requirements of government, hence the concomitant lack of incentive for value-creation. Boehm and Sullivan point out that since the majority of software applications are now created under the auspices of private (profit-driven) entities, it is likely that more attention will be placed on the relative value created (or destroyed) by such development activities.

Previous research by Mooney (1993, 1994) has focused on the various component costs of application development, and the cost ramifications resulting from the incorporation of portability at the time the application is written.[6] Baldwin and Clark (2000) appear to have been the first to recognize that the value of software modularity could be modeled with a real options approach. Sullivan, Chalasani, Jha, and Sazawal (1999) have sought to provide a theoretical rationale for the use of real options methodology as a means of placing a value on software design. Beck (2000) also recognizes the various options embedded in software development and implores software developers to incorporate an economic-based decision process.

[4] Further exacerbating a project's risk is the fact that these applications are usually developed using services - such as database management, communications, user interface, operating system - that may require replacement at any given time. These services upon which the application relies shall be referred to as *service components* herein.

[5] A history of software economics is provided in this paper.

[6] Mooney makes a clear distinction between portability and reuse, noting that the term portability implies the "reuse of complete applications", while reuse by itself implies that only a subset of an entire application may be salvageable (with the remainder being rewritten). In this paper we define portability as the ability of a software unit to be ported to a given (new) environment at a cost that is significantly below the expected cost of full redevelopment. The portability layer is a (small) subset of the overall application that permits this porting to a new environment.

Our research takes this line of reasoning to its next logical step: we employ real options methodology as a tool with which software modularity (specifically software portability) can be valued, and present a logical framework for capital budgeting decisions concerning this aspect of software projects. Specifically, we are concerned with the necessity of application redevelopment, and the financial implications of using various strategies to contend with this risk.[7] To this end, we present a model using real options that can assist in providing a quantitative justification for incurring development costs relating to application architecture decisions that attempt to minimize this risk. We are hopeful that this is just a first step in the application of financial evaluation tools to the analysis of capital expenditures on software development projects.

In Section II, we discuss the factors that can lead to the necessity of application redevelopment, specifically focusing on an example relating to the loss of a service component and the various strategies to contend with this occurrence. In Section III, we describe real options, the factors that determine the value of a real option, and their application to software development. In Section IV, we describe our real option model. Section V explores the relationship between the value of portability and factors such as the theoretical and expected practical life of the application and the volatility of redevelopment costs using sensitivity analysis. Section VI concludes the paper.

II. Application Development and Redevelopment Strategies

Software structure largely determines the degree of flexibility, and flexibility is valuable. The inclusion of software portability normally falls within the domain of application architecture. Application architecture is the discipline of developing the overall model and software design, and specifies the major characteristics of a system.

A software rewrite can occur for a variety of reasons, among which is the loss of a service component upon which the application relies. The loss of a service component can result from numerous factors, such as the application becoming obsolete or a change to new hardware that is incompatible with existing software.[8] The actual cause of application failure is not critical to our analysis. We focus on the loss of a service component as one of these causes because it is relatively easy to model, and is intuitive and tractable.

Regardless of the factor underlying the loss of the service component, there are three fundamental approaches considered here that might be applied to the installation of a new service component. Each of these approaches can have differing financial ramifications, as well as potential effects on the overall functionality of the application relying on the component.

The first approach is to *rewrite the application* to interface directly with the new service component. This approach is assumed to have no cost at time zero beyond the standard

[7] The specific instance triggering the redevelopment is the loss of a service component, but any event making redevelopment a necessity is generally consistent with the analysis provided.

[8] Some other factors leading to the loss of a service component include: vendor going out of business, discontinuation of software, cost of an input to the application changing, inability to scale up/down, and a merger necessitating that software applications be rationalized. This list is not exhaustive.

developmental costs associated with writing the original software, but it effectively entails a full rewrite upon subsequent loss of the service component.[9] Consequently, the expected cost of future redevelopment under this approach is the highest of the three approaches considered.

The second approach is to insert a *software portability layer* between the application software and the service component interface. A software portability layer can be explained as a set of software (and/or hardware) that is adaptable to a replacement service component, thus avoiding modification to the application software. The software portability layer is necessarily a small subset of the overall software application. The application program (the majority of the software) is written to interface to the software portability layer, instead of being written to interface directly to a specific service component. The software portability layer is written to access the service component, thus encapsulating the service component and hiding its direct interface from the application software. This approach is assumed to have a higher initial cost at time zero (i.e., when the application is originally developed), but also results in the largest reduction in expected future costs relating to future service component loss.

The third approach is to create an *emulation layer* in the event of a service component requiring replacement. An emulation layer involves the insertion of an interface that behaves nearly identically to that of a specific service component that is being replaced. In other words, it makes the new service component look exactly the same as the old service component from the perspective of the primary application, thus eliminating the need to rewrite the primary application. This approach is assumed to have no cost at time zero beyond the standard developmental costs associated with writing the original software, since the costs associated with an emulation layer occur in the future. In the event of a service component replacement event this approach has a lower expected cost than a complete rewrite, but a higher expected cost than would be the case if a portability layer is present. We have also allowed for the possibility that emulation may lead to degradation of the software's performance. This degradation is represented as a cost in our model. If the magnitude of the degradation is believed to be extreme (in effect rendering the software unusable) an infinitely large cost can be assigned to emulation. The practical result is that emulation would never be an economically feasible solution under such a scenario.

A summary of the relationship of initial costs and future redevelopment costs that we expect to occur most frequently in practice is presented in the table below, where C_0 is the base application development cost at time t = 0 and C_P is the cost of adding a portability layer for a given service component. The analytical results presented in Section V are based upon these relationships. We note, however, that our real options model is robust for virtually any expected cost relationship.

Strategy	Initial Cost	Expected Redevelopment Costs
Full Rewrite	C_0	High
Portability Layer	$C_0 + C_p$	Low
Emulation Layer	C_0	Medium

[9] The assumption of a full rewrite is simply a base case assumption. Our model is capable of handling an infinite range of values for the cost of rewriting the software.

III. Financial Options, Real Options, and Software Portability and Emulation

Generically, an option is a right to take an action. For example, a call option on a stock provides the option holder the right to buy a particular number of shares of a particular stock at a particular exercise price on (or before) a particular maturity date. Options that can only be exercised on the maturity date are said to be European-style, whereas American-style options can be exercised on or before the maturity date. Given that the option represents a right, but not an obligation, the action will only be taken if it benefits the option holder. The call option holder will only exercise the call option if it allows him/her to purchase the underlying stock at a price (the exercise price) that is less than the current stock price.

Coinciding with the trading of stock options on organized exchanges, Black and Scholes (1973) published their seminal paper on option valuation, leading to the development of a family of option pricing models (OPMs) in subsequent years. Whereas the Black-Scholes OPM provides a closed-form solution for the valuation of European-style options, binomial OPMs have become the norm for analyzing American-style options. [See Cox, Ross, & Rubinstein (1979).]

Relatively recently, a new branch of research has emerged whereby the financial option framework is applied to real assets. This area is typically described as a real options valuation approach. [See Amram and Kulatilaka (1999) or Copeland and Antikarov (2001) for an overview of real options.] Within the capital budgeting, area a large number of option-like features have been identified: the option to abandon a project, the option to expand a project, the option to delay a project, the option to restart a project, the option to switch production inputs, et cetera. [See Trigeorgis (1993) or Dixit & Pindyck (1994) for an overview of real options and capital budgeting.]

Initially it was quite natural for financial researchers to look at familiar applications (e.g. capital budgeting) that normally fall under the purview of the finance profession. However, researchers now find themselves applying real options analysis to new problems that reach across traditional disciplinary boundaries. For example, Benaroch & Kauffman (1999) employ a real options approach to evaluate investments in information technology hardware. It is within this latter branch of research that this paper resides.

Analogous to an option, software portability provides the firm with flexibility to take actions that it would otherwise not possess. For example, software that incorporates portability to work with a variety of databases can simply switch to a new database if the current database becomes obsolete. Alternatively, if the software does not contain a portability layer, the firm is faced with writing an emulation layer, if possible, or re-writing the entire application.[10]

While a stock call option gives the holder the right to acquire a share of stock by paying the exercise price, the portability or emulation option give the firm the ability to restore the use of software from a loss of a service component by paying either the cost to port or the cost to

[10] We are assuming that the software supports critical functions within the firm. As such, the software will always be restored. In other words, abandoning the software is equivalent to abandonment of the entire firm, which is assumed never to be an optimal decision. Modifying the model to incorporate abandonment of the software is possible; however, the case examined in the paper is more relevant.

emulate the software. As such, the portability and emulation options are call options. The value of a call option on a stock depends on the current stock price, the risk-free interest rate in the economy, the option's maturity date, the option's exercise price, and the volatility of the underlying stock price. Next, we draw analogies between these five variables for a stock call option and the portability/emulation option.

Definitions of Variables

Value of the underlying asset. This variable is the foundation from which the option derives its value. For a stock option, it is the current stock price. In the case of real options on software it is the cost to develop (or redevelop) the software. Given that the call option holder has the right to purchase the share for the fixed exercise price, the higher the current stock price, the more valuable the call stock option. In our case, the higher the cost to redevelop the software when the loss of a service component occurs, the more valuable the portability or emulation option.

Risk-free interest rate. This variable is the return to holding the risk-free asset, and remains identical in both financial options and real options framework. An increase in the risk-free interest increases the value of the call option.

Maturity date. This variable is the time remaining until the option expires, and after which the option ceases to exist. In our framework, the portability/emulation option can be exercised during the life span of the software and ceases to exist once the software no longer exists. In general, a longer term to maturity increases the value of the call option, because the option provides its flexibility over a longer period of time.

Exercise price. For a stock call option, this is the fixed price at which the call option holder can purchase the stock. In the case of real options on software, this represents the cost to port the software or emulate the software when a replacement event occurs. It should be noted that the exercise price is typically fixed throughout a financial option's life. However our real option model allows the exercise price (the cost to port or emulate) to vary through time. A lower exercise price increases the value of a call option.

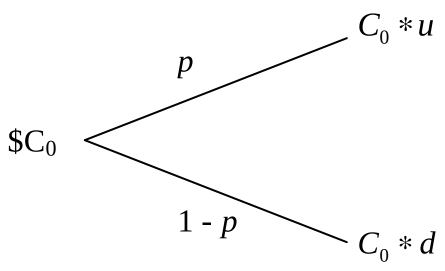
Volatility of the underlying asset value. Generically, an option's value is a function of the volatility of the value of the underlying asset. This is because the greater the volatility, the greater the likelihood that the option will be "in the money" (have positive intrinsic value) at expiration. For stock options, volatility is a function of changes in the stock's price, while in the case of software portability, volatility represents uncertainty regarding changes in the cost to redevelop the software in the future.

In our real options model an additional factor is present—that of the probability of a replacement event occurring. In other words, not only does the cost to redevelop (the stochastic value of the underlying asset) change, but there is also uncertainty as to whether a replacement event will occur at all. Software portability would be more valuable if replacement events were frequent. On the other hand, a software portability layer would yield little benefits if replacement events were the exception. Accordingly, the volatility of redevelopment cost and the probability of a replacement event interact to determine the option value.

A final consideration differentiates the portability option and emulation option from the standard stock option. Whereas a call option can only be exercised once, the portability option and emulation option can be exercised each time a replacement event occurs. Hence, they are analogous to a portfolio of options.

IV. Model Development

We use a binomial option pricing model to value the portability and emulation option. In the binomial model, the value of the underlying asset is allowed to take on one of two values over a very small period of time. In the case of our real options model, the cost of software development, C_0, can increase over the next period by an upward multiplier, u, or decrease over the next period by a downward multiplier, d. The probability of an upward move is *p*, and the probability of a downward move is (1- *p*). The figure below depicts a single binomial tree node.



Cox, Ross, and Rubinstein (1979) established the following relationship between the annual instantaneous standard deviation of the rate of return on the underlying asset and the up and down movements in the binomial tree, given by the set of equations in 1, where T represents the life of the option in years, and n represents the number of branches in the binomial tree over the life of the option.

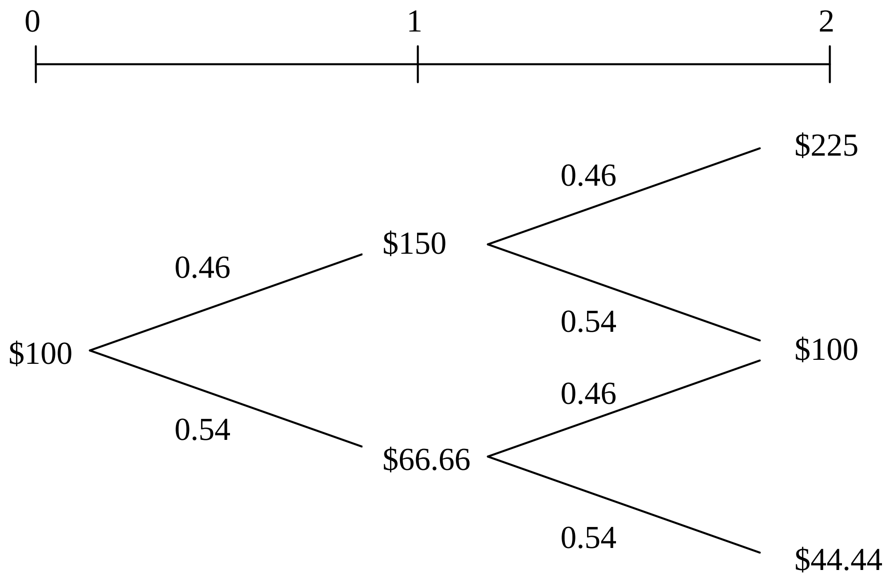
$$u = e^{s\sqrt{T/n}} \qquad d = e^{-s\sqrt{T/n}} \tag{1}$$

In practice, questions like "What is the largest increase or decrease in cost development costs over the next year?" can be used to help determine appropriate values for u, d, and the standard deviation.

A key feature of the binomial option model is the replacement of the actual probabilities of the upstate/downstate occurring with pseudo-probabilities, p and (1-p). As will be demonstrated in the following example, the pseudo-probabilities served to adjust the expected cash flows, such that the appropriate discount rate is the risk-free interest rate.[11]

[11] A requirement of the binomial option pricing model is the existence of a twin security that can be used to construct a risk-free portfolio; that is, a portfolio whose payoffs in the upstate and the downstate are identical. Accordingly, the risk-free rate of return is the required return on this portfolio. Copeland and Antikarov (2001) make a strong argument that the logical twin security is the underlying project without the embedded option(s). They further demonstrate that the assumption of a twin asset is actually analogous to the assumption found in the traditional Net Present Value (NPV) valuation method. The requirement of a twin security is frequently used as a criticism of the binomial model, when in actuality this assumption is found in the most common of valuation methods used everyday without question.

Consider a two-year example, where the cost to write the original software is \$100, and the annualized volatility of this cost is estimated at 40.55%. Let the annual risk free rate be 5% per year. From the Cox, Ross, and Rubinstein relationship, this corresponds to a 50% increase or a 33.33% decrease per period in software costs. The tree for this software development cost is depicted below, where the pseudo-probability of the upstate (downstate) move is 46% (54%) as determined by (2).



$$p = \frac{(1+r)-d}{u-d} \tag{2}$$

$$= \frac{(1+.05)-0.6666}{1.5-0.6666} = 46\%$$

To confirm that the risk-free interest rate is the appropriate discount rate, consider the present value calculation of the software development cost in the first period. The expected time-one costs using the pseudo-probabilities and discounting at the risk-free rate are determined by (3).

$$C_0 = \frac{p * C_u + (1-p) * C_d}{1+r} \tag{3}$$

$$= \frac{0.46 * 150 + 0.54 * 66.66}{1.05}$$

$$= 100$$

This matches the actual software development cost of \$100 at time zero, confirming our result. The use of pseudo-probabilities combined with discounting at the risk-free interest rate will be employed to find the value of the portability option and the emulation option.

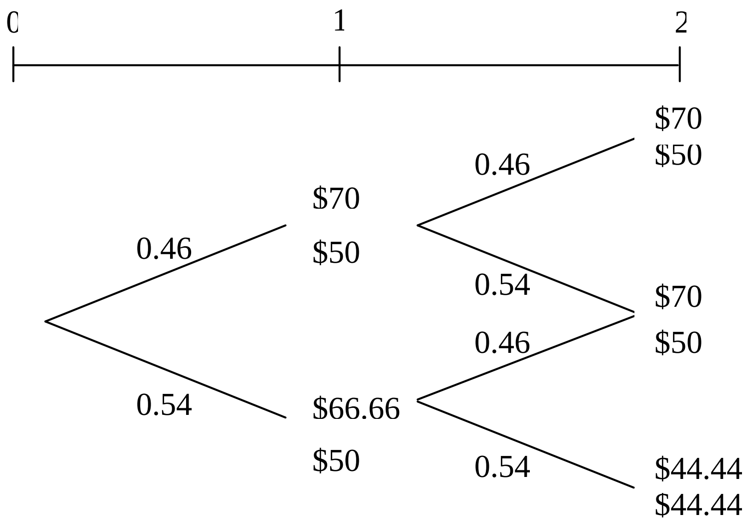
To continue the example, assume for simplicity that the cost to emulate is constant at \$70, while the cost to port is constant at \$50. Finally, assume there is a 20% (30%) probability of a replacement event at time 1 (2). To begin, consider the expected cost of software development without the flexibility given by emulation or portability, as determined by (4).

$$50 = 0.2\overset{\text{First Period Expected Cost}}{\left[\frac{0.46*150+0.54*66.66}{1.05}\right]}+ \tag{4}$$

$$+0.3\overset{\text{Second Period Expected Cost}}{\left[\frac{(0.46)^2*225+2*0.46*0.54*100+(0.54)^2*44.44}{(1.05)^2}\right]}$$

Given the 20% chance of a replacement event in the first period, there is a 46% pseudo-probability of the upstate occurring (\$150 in redevelopment costs) and a 54% pseudo-probability of the downstate occurring (\$66.66 in redevelopment costs). In the second period, there is a 30% chance of a replacement event. In order to reach the top node in period two, we would need to move up twice, which occurs with a 21.16% pseudo-probability (46%*46%). In order to reach the middle node, we need to move up once and down once. However, realize there are two paths to reach the middle node (up first and then down, or down first and then up). Accordingly, the pseudo-probability for the middle node is 49.68% (2*46%*54%). To reach the lower node, we need to move down twice, which occurs with a 29.16% pseudo-probability. Finally, note that the pseudo-probabilities in each period sum to one. Given that these cash flows occur in the second periods, we discount them back two periods at the risk free interest rate.

From the above calculations, we have determined that the expected development cost without the portability or emulation option is \$50. We can now repeat the expected development cost tree for the portability option and then the emulation option. To conserve space, we present a single tree to show both cases. At each node in the tree, the decision rule is that the firm will select the least costly alternative available, i.e. $\min\{C_{t,full-rewrite}, C_{t,emulate}\}$. This tree is shown below, where



the top number represents the development cost under the emulation option and the bottom number represents the development cost under the portability option.

In states of nature where the cost of development exceeds \$70 (\$50), the option to emulate (port) the software will be employed to reduce the expected development costs. If the cost to develop is below the cost to emulate (\$70) or port (\$50), then the optimal action is to simply rewrite the software. Using the same present value approach as in (4), the expected development cost with emulation and with portability is calculated by (5) and (6), respectively.

$$\begin{aligned}30.01 = 0.2&\left[\overset{\text{First Period Expected Cost}}{\frac{0.46*70+0.54*66.66}{1.05}}\right]+\\ +0.3&\left[\overset{\text{Second Period Expected Cost}}{\frac{(0.46)^2*70+2*0.46*0.54*70+(0.54)^2*44.44}{(1.05)^2}}\right]\end{aligned} \tag{5}$$

$$\begin{aligned}22.69 = 0.2&\left[\overset{\text{First Period Expected Cost}}{\frac{0.46*50+0.54*50}{1.05}}\right]+\\ +0.3&\left[\overset{\text{Second Period Expected Cost}}{\frac{(0.46)^2*50+2*0.46*0.54*50+(0.54)^2*44.44}{(1.05)^2}}\right]\end{aligned} \tag{6}$$

The expected savings in development costs represents the value of the emulation option and the portability option:

Value of Emulation Option = Expected Development Cost Without Flexibility –
– Expected Development Cost With Emulation
= \$50 – \$30.01 = \$19.99

Value of Portability Option = Expected Development Cost Without Flexibility –
– Expected Development Costs With Portability
= \$50 - \$22.69 = \$27.31

Notice that the additional value of the portability option versus the emulation option comes from two components. First, in states of nature where the cost of rewriting the software is high, the portability option reduces costs by an additional \$20 over the emulation option. Secondly, the portability option is exercised in additional states of nature. For example, if a replacement event occurs in the downstate of the first year the portability option reduces costs by \$16.66, whereas the emulation option is not exercised and no cost savings are realized.

Up to this point, we have considered the emulation option and the portability option separately. As described in Section II however, the emulation option requires no action today and as such is assumed to be always available. Accordingly, the true value of the portability option is the additional cost savings beyond the emulation option.

As such, we will define the true value of the portability option as follows:

Value of Portability Option = Expected Development Cost With Emulation –
– Expected Development Costs With Portability
= \$30.01 - \$22.69 = \$7.32

(or, in terms of the savings, \$27.31 - \$19.99 = \$7.32).

This is the definition of portability option value that is used in Section V of the paper. Accordingly, if the cost to write a portability layer is below \$7.32, we should incorporate a portability layer into the software. If the cost to write a portability layer exceeds \$7.32, we should forego the portability layer in the software.

If the emulation option is not available (for example, a real-time application that would suffer an unacceptable level of performance degradation with the insertion of an emulation layer), this can be modeled by simply assigning an arbitrarily large value as the cost to emulate. Returning to the prior example, if the \$70 emulation cost were replaced with a \$300 emulation cost, the emulation option would have higher redevelopment costs at each node than a full rewrite and would provide no cost savings at any node. As such the expected development cost with emulation would equal the expected development cost without flexibility since emulation would never be economically rational.

In the above example, the two-year life of the software was divided into two periods. As a result, the cost of software development could only take on one of two values over the next year. However, the time periods are completely arbitrary in the model. In other words, we can divide the life of the software project into as many periods as we desire. For example, we can model the two-year software project using four six-month periods, twelve two-month periods, etc. While the number of computations increases dramatically as the number of periods increases, the approach remains identical.

To model the probability of a replacement event in our model, we use the gamma distribution. This distribution was selected to allow us to vary the expected life of the application while holding the theoretical "maximum" life of the application constant. We define the theoretical life of an application as the point at which 95% (i.e. the cumulative replacement probability) of similar applications require replacement. The gamma distribution allows us to maintain a constant theoretical life (i.e. cumulative replacement probability value at time T) while altering the expected life of the application, defined as the mean expected replacement date as a percentage of the theoretical life. We refer to the expected life variable as the "persistence" of the application in Tables 1 and 2. Figure 1 shows the cumulative probability of replacement for 3 hypothetical applications with differing levels of persistence. The highest curve represents applications that have a life expectancy of 25% of their feasible useful life (25% persistence), while the lowest curve represents applications that have a life expectancy of 75% of their useful life (75% persistence). In practice, our model would allow for the inclusion of virtually any probability specification as a proxy for the frequency of occurrence of a replacement event.

In the prior section, we described the impact the six main determining factors (cost of software redevelopment, the risk-free interest rate, the software's life span, the cost of emulation/portability, the volatility of redevelopment costs, and the probability of a replacement event) on the value of a *single* option, the emulation option *or* the portability option, in *isolation*. However, the true value of the portability option, as defined in our model, depends upon the *difference* between the portability option and the emulation option. As we shall see in the next section, the interaction of these value-determining factors can lead to results that appear counter-intuitive at first glance.

V. Results and Analysis

We value the portability option by assuming that there are three states of nature. First, we assume that there are situations in which neither emulation nor portability is possible. For example, certain real-time applications may be sufficiently sensitive to performance degradation that it is not possible (or desirable) to insert either an emulation or portability layer. While we believe that there are relatively few such cases, this is posited as our base case and the value to emulation and portability are a function of the cost reduction available relative to a full rewrite necessitated under the base case. The cost incurred under the base case when a service component fails is simply the expected cost of a full rewrite of the application.

Second, we assume that in most situations it will be possible to emulate to a new service component. We have arbitrarily assumed that the cost of such emulation is 20% of the cost of the initial application development cost. However, it would be possible to use any cost of emulation from 0% (i.e. the transition was costless) to 100% (i.e. the transition involved effectively replacing the entire application), without invalidating our model. The requirement for the portability option to have positive value is simply that the inclusion of a portability layer confers some expected benefit relative to emulation in the face of a replacement event. This is the emulation case.

Third, we assume that it is possible to include a portability layer at the time the application is originally developed. If so, it is assumed that the inclusion of such a layer will necessarily provide some expected cost benefit relative to the expected (but not necessarily actual) cost of rewriting, as well as a somewhat smaller but positive cost benefit (either immediately or in the future) relative to the cost of emulation. This is the portability case.

There are two hypothetical portability cost-benefit scenarios examined in our analysis, and these are sufficiently general that they likely have applications beyond the software application development scenarios described in the paper. The initial application cost is assumed to be $1 million under both scenarios, but this is simply a scalar. Scenario One is that in which the cost to port is less than the cost to emulate, but that the two costs are increasing at the same rate over the theoretical life of the applications. In specific, the porting cost is initially 10% of the initial application cost (relative to 20% to emulate), and both the cost to port and the cost to emulate are growing at a rate of 5% per annum.

Table 1 shows the net value to portability under Scenario One for various expected theoretical maximum application lives, persistence levels (meaning expected life as a percentage of the theoretical life of the application), and levels of volatility in the cost of software application development. It can be seen that when volatility of application development costs is low (in Panel A where this is 10%), the net present value of owning the portability layer is approximately $95,000 in all cases. In other words, you would pay up to this amount to put a portability layer in place, since this is the expected present value cost advantage (over emulation) of the option to port. This makes intuitive sense, as the original built-in cost advantage is $100,000, and we have, by construction assumed that 95% of all applications are rewritten during their theoretical lives.

In Panels C, D, and E, we begin to see some variation in the value of the portability option. In all cases, the value of the option is decreasing in both the volatility of the underlying process, and in the theoretical life of the application. For example, under a volatility assumption of 40%, the value of portability is estimated at $94,793 for an application with a theoretical life of 5 years, and a persistence level of 25% (i.e. it is expected to last 5 years $\times$ 25% = 1.25 years after it is put into service). An option with the same volatility assumption, but a theoretical life of 30 years, and a persistence level of 75% has an estimated value of $51,179. In other words, you would be willing to pay about $40,000 less for the portability layer for applications typified by the latter the case than by the former.

Scenario Two is that in which the cost to port is initially the same as the cost to emulate, but there is an anticipated future cost advantage in that the cost to port is growing more slowly than the cost to emulate over the theoretical life of the application. In specific, the porting cost is initially 20% of the initial application cost (relative to 20% to emulate), and the cost to port is growing at a rate of 5% per annum, while the cost to emulate is growing at 7.5% per annum.

Table 2 shows the net value to portability under Scenario Two for various theoretical application lives, persistence levels, and levels of volatility in the cost of software development. It can be seen that when volatility of application development costs is low (Panel A where this is 10%), the net present value varies significantly as a function of both theoretical and practical application life. Specifically, it can be seen that the value of portability is relatively modest for applications with both short theoretical and practical lives. For example, for a project with a 5-year theoretical life and a persistence level of 25%, the value of a portability layer is only $4,897. This increases to $140,000 for a project with a 30-year theoretical and 75% persistence level. Hence, the value of the portability option is increasing in theoretical and expected practical project life.

In Panels B, C, D, and E we see results that are similar in that the value of the option is increasing in both the theoretical and expected practical application life. See Figure 2 for a graphic depiction of how these variables affect the value of the portability option for an application with a theoretical life of 20 years. However, we also note that, as in the prior case, the results seem to be puzzling insofar as the value of the portability option is seen to be decreasing in volatility. Here we illustrate that these results, while somewhat surprising, are, in fact, consistent with the theory underlying the valuation model.

With regard to the results displayed in Table 2, we note that while option prices are generally increasing in volatility, the degree of change is dependent upon the intrinsic value of the option. In specific, options that are at-the-money will experience a greater change in value for a given change in the volatility of the value of the underlying asset. Since the emulation option is closer to being at-the-money than the portability option, its value will increase more rapidly as the volatility increases. Given that the value of the portability option has been defined incrementally as the difference between the value of the portability option and the value of the emulation option, it then makes sense that this difference is actually decreasing in volatility.

With regard to the results displayed in Table 2 and Figure 2, we note that as the volatility increases the dispersion of future application development costs increases – both upward and downward. Hence, while the expected cost of application development is also increasing by 5% per year, there are some states that occur with positive probability in which the cost is actually less than the certain cost to port the original application. Such cases may occur, for example, when Firm A owns a viable application that it can replicate. Suppose that Firm B has an application that includes a portability layer, and that the application is presently in need of a replacement. If Firm A acquires Firm B, A can simply replace B's application at a cost that is close to zero, thereby rendering the value of B's portability layer worthless. Hence, the greater the volatility of the development cost, the more such states exist, and the concomitant decrease in the value of the portability option. This is reflected in decreasing option values as the volatility of the value of the underlying asset increases.

VI. Summary and Conclusions

Software application development represents a significant use of capital for firms (and other entities) in virtually all industries and in all parts of the world. Yet, by comparison with other comparably-sized capital projects, relatively little economic analysis is employed in the evaluation of these expenditures. In addition, this type of investment often appears to represent a relatively risky use of capital since these applications have highly uncertain useful lives as a result of the pace of technological change in the area. Consequently, features that can be incorporated into software applications that reduce this risk should have economic value.

Other researchers have argued that it may be possible to reduce the risk of software obsolescence by building flexibility into the application at the time it is developed. In this paper we refer to this type of flexibility as software portability. Such flexibility may permit the application to adapt to technological change, thereby reducing the probability that the application will need to be rewritten prematurely. This research applies financial theory to the valuation of flexibility in the form of real options analysis, and it attempts to infer the value of the portability option that can be incorporated into software applications when these are developed. The outcome is a quantitative model within which such decisions to include or exclude portability can be made based upon a rigorous analytical framework. The result is to put the evaluation of the software architecture decision on the same footing as other capital budgeting decisions wherein the cost, benefit, and risk can be assessed in a systematic manner (typically NPV).

The sensitivity analysis presented indicates that incorporating portability can be a positive net present value (NPV) project. This value is shown to be a function of the theoretical and expected practical life the application as well as the volatility of the underlying cost of application development. Moreover, for the scenarios considered here, this value is shown to be non-trivial in an economic sense. The valuation examples presented result in option values ranging from almost zero to as much as 14% of the application value. Using an average real option value of 7%, and the estimated total worldwide software development expenditures of USD $800 billion per year, this implies that somewhere in the neighborhood of 7% x $800 billion = $56 billion in real options values may be available each year to software engineers and their firms.

By demonstrating how this approach works with a stylized, base case set of assumptions we hope to have provided a theoretical foundation for the further use of real options analysis in this area. In particular, we believe that the next logical step is to apply this type of analysis to specific circumstances that are encountered in practice. Further research involving the application of the model to actual case study data is a part of our ongoing research in the area.

References

Amram, M. and N. Kulatilaka (1999), “Real Options,” Harvard University Press.

Benaroch, M. and R. Kauffman (1999), “A Case for Using Real Options Pricing Analysis to Evaluate Information Technology Project Investments,” *Information Systems Research,* 10:1, 70-86.

Baldwin, C. Y. and K. B. Clark (2000), *Design Rules: The Power of Modularity* (MIT Press).

Beck, K. (2000), *Extreme Programming Explained: Embrace Change,* (Boston: Addison-Wesley).

Black, F. and M. Scholes (1973), “The Pricing of Options and Corporate Liabilities,” *Journal of Political Economy*, 81, 637-659.

Boehm, B. and K. Sullivan (1999), “Software Economics: Status and Prospects,” *Information and Software Technology,* 41, 937-46.

Copeland, T. and V. Antikarov (2001), *Real Options*, (New York: TEXERE).

Cox, J., S. Ross, and M. Rubinstein (1979), “Option Pricing: A Simplified Approach,” *Journal of Financial Economics*, 7, 229-264.

Dixit, A. and R. Pindyck (1994), *Investment Under Uncertainty* (Princeton: Princeton University Press).

Gross, Neil, Marcia Stepanek, Otis Port and John Carey (12/6/99), “Software Hell,” *Business Week*, 3658, 104-118.

Mooney, J. (1993), “Issues in the Specification and Measurement of Software Portability,” Working Paper, West Virginia University.

Mooney, J. (1994), “Portability and Reusability: Common Issues and Differences,” Working Paper, West Virginia University.

Parnas, C. L. (1972), “On the Criteria to be Used in Decomposing Systems into Modules,” *Communications of the ACM*, 15:12, 1053-1058.

Sullivan, K. J., P. Chalasani, S. Jha, and V. Sazawal (1999), “Software Design as an Investment Activity: A Real Options Perspective,” in *Real Options and Business Strategy: Applications to Decision-Making,* L. Trigeorgis, ed., (London: Risk Books), 215-261.

Trigeorgis, Lenos (1993), “Real Options and Interactions with Financial Flexibility,” *Financial Management*, 202-224.

Table 1: Portability Option Value

Initial Cost Differential, Porting – Emulation = $100,000.
The values in this table represent the net present value of a portability layer. This is defined as the expected present value to rewrite when emulation is possible, minus the expected present value to rewrite when a portability layer is in place. The initial cost to emulate is assumed to be 20% of the application's initial cost of $1,000,000, and is growing at a rate of 5.0% per annum; the cost to port is 10%, and is growing at a rate of 5.0% per annum. Persistence is the mean point during the application's theoretical life at which it will be necessary to rewrite the application.

	Theoretical Maximum Life of Application					
Persistence	**5**	**10**	**15**	**20**	**25**	**30**
Panel A: Volatility = 10%						
25%	$95,017	$95,029	$95,043	$95,056	$95,070	$95,081
35%	$95,028	$95,046	$95,068	$95,090	$95,111	$95,131
45%	$95,030	$95,060	$95,091	$95,121	$95,150	$95,176
55%	$95,034	$95,073	$95,111	$95,149	$95,196	$95,227
65%	$95,026	$95,073	$95,139	$95,199	$95,235	$95,268
75%	$94,911	$95,184	$95,166	$95,183	$95,297	$95,306
Panel B: Volatility = 20%						
25%	$95,017	$95,029	$95,043	$95,056	$95,070	$95,081
35%	$95,028	$95,046	$95,068	$95,090	$95,111	$95,131
45%	$95,030	$95,060	$95,091	$95,121	$95,150	$95,176
55%	$95,034	$95,073	$95,111	$95,149	$95,196	$95,227
65%	$95,026	$95,073	$95,139	$95,199	$95,235	$95,268
75%	$94,911	$95,184	$95,166	$95,183	$95,297	$95,306
Panel C: Volatility = 30%						
25%	$94,996	$94,699	$94,046	$93,165	$92,161	$91,096
35%	$94,989	$94,491	$93,362	$91,829	$90,079	$88,226
45%	$94,977	$94,197	$92,436	$90,076	$87,422	$84,658
55%	$94,953	$93,753	$91,137	$87,756	$84,084	$80,365
65%	$94,896	$93,076	$89,386	$84,924	$80,224	$75,632
75%	$94,693	$92,183	$87,291	$81,703	$76,182	$70,891
Panel D: Volatility = 40%						
25%	$94,793	$93,551	$91,770	$89,833	$87,894	$86,014
35%	$94,649	$92,503	$89,398	$86,038	$82,703	$79,501
45%	$94,448	$91,112	$86,405	$81,452	$76,667	$72,183
55%	$94,139	$89,229	$82,697	$76,154	$70,083	$65,576
65%	$93,652	$86,816	$78,477	$70,640	$63,606	$57,430
75%	$92,833	$84,081	$74,176	$65,311	$57,726	$51,179
Panel E: Volatility = 50%						
25%	$94,189	$91,502	$88,455	$85,507	$82,766	$80,245
35%	$93,610	$88,933	$83,663	$78,644	$74,054	$69,903
45%	$92,826	$85,711	$78,032	$71,002	$64,791	$59,340
55%	$91,711	$81,761	$71,789	$63,144	$55,827	$49,620
65%	$90,178	$77,322	$65,533	$55,896	$47,996	$41,501
75%	$88,154	$72,872	$59,814	$49,575	$41,524	$35,021

Table 2: Portability Option Value

Initial Cost Differential, Porting – Emulation = $0
The values in this table represent the net present value of a portability layer. This is defined as the expected present value to rewrite when emulation is possible, minus the expected present value to rewrite when a portability layer is in place. The initial cost to emulate is assumed to be 20% of the application's initial cost of $1,000,000, and is growing by 7.5% per annum; the cost to port is 20%, and is growing by 5.0% per annum. Persistence is the mean point during the application's theoretical life at which it will be necessary to rewrite the application.

	Theoretical Maximum Life of Application					
Persistence	**5**	**10**	**15**	**20**	**25**	**30**
Panel A: Volatility = 10%						
25%	$4,879	$9,690	$14,808	$20,248	$26,012	$32,080
35%	$8,919	$18,156	$23,782	$38,647	$42,074	$52,014
45%	$10,367	$21,148	$32,697	$45,066	$58,265	$72,248
55%	$13,089	$26,890	$41,785	$57,849	$75,085	$93,446
65%	$15,829	$32,740	$51,159	$71,166	$92,825	$115,980
75%	$18,596	$38,707	$60,864	$85,193	$111,627	$140,000
Panel B: Volatility = 20%						
25%	$4,879	$9,658	$14,574	$19,501	$24,363	$29,122
35%	$8,918	$18,083	$23,399	$36,933	$39,311	$47,028
45%	$10,367	$21,070	$32,109	$43,148	$53,982	$64,508
55%	$13,088	$26,770	$40,880	$54,907	$68,549	$81,693
65%	$15,827	$32,549	$49,736	$66,629	$82,892	$98,369
75%	$18,594	$38,387	$58,594	$78,194	$96,762	$114,236
Panel C: Volatility = 30%						
25%	$4,861	$9,347	$13,544	$17,410	$20,970	$24,258
35%	$8,877	$17,369	$21,644	$32,205	$33,468	$38,633
45%	$10,322	$20,260	$29,381	$37,591	$44,975	$51,640
55%	$13,020	$25,530	$36,773	$46,683	$55,424	$63,189
65%	$15,718	$30,659	$43,726	$54,966	$64,706	$73,210
75%	$18,411	$35,571	$50,144	$62,403	$72,837	$81,801
Panel D: Volatility = 40%						
25%	$4,768	$8,705	$12,046	$14,903	$17,372	$19,528
35%	$8,664	$15,915	$19,059	$26,702	$27,258	$30,489
45%	$10,082	$18,552	$25,414	$31,019	$35,655	$39,526
55%	$12,650	$22,996	$31,061	$37,430	$42,530	$46,656
65%	$15,146	$27,058	$35,956	$42,739	$47,992	$52,089
75%	$17,540	$30,702	$40,133	$47,070	$52,270	$56,143
Panel E: Volatility = 50%						
25%	$4,586	$7,905	$10,470	$12,507	$14,153	$15,499
35%	$7,143	$12,393	$16,334	$19,367	$21,740	$23,613
45%	$9,597	$16,433	$21,330	$24,927	$27,601	$29,591
55%	$11,918	$19,970	$25,418	$29,194	$31,816	$33,606
65%	$14,071	$22,994	$28,654	$32,328	$34,663	$36,054
75%	$16,017	$25,531	$31,170	$34,544	$36,464	$37,341

Figure 1: Cumulative Density Functions

This figure shows the cumulative probability of replacement for three hypothetical applications with differing levels of persistence. The highest curve represents applications that have a life expectancy of 25% of their feasible useful life (25% persistence), while the lowest curve represents applications that have a life expectancy of 75% of their useful life (75% persistence).

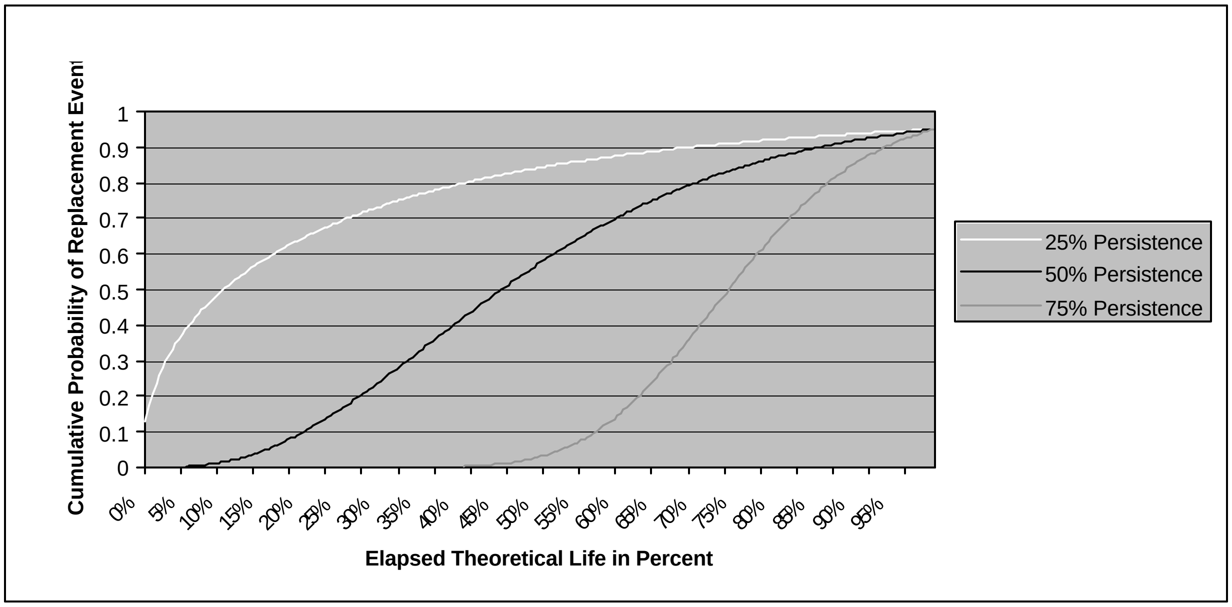
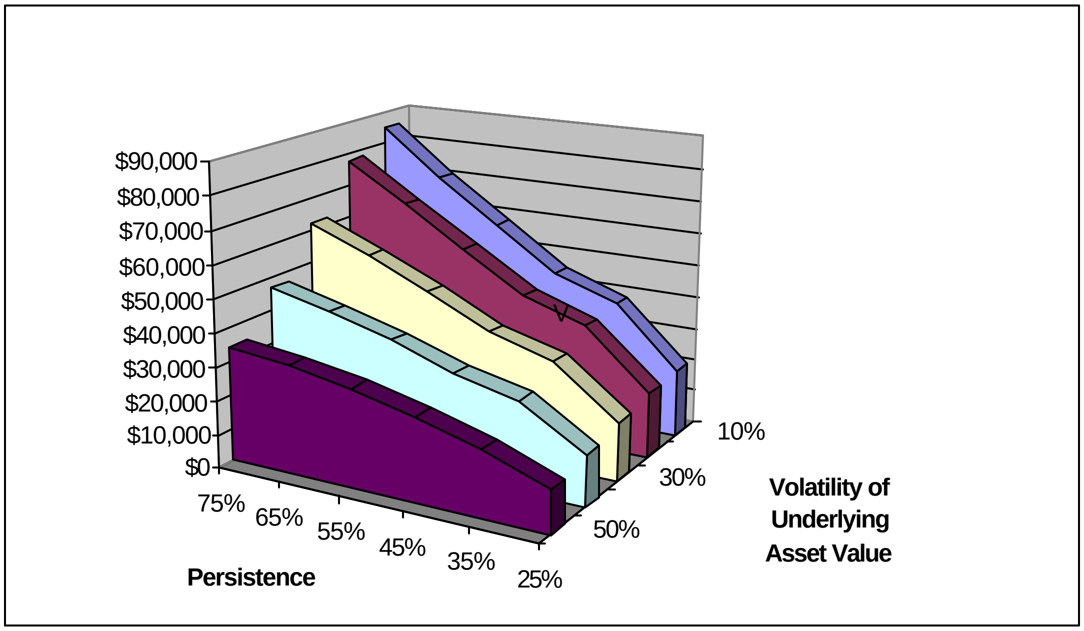


Figure 2: Portability Option Value with a 20-Year Theoretical Application Life

This figure shows a graphic depiction of how persistence and volatility of the underlying asset value affect the value of the portability option for an application with a theoretical life of 20 years. In the case of software portability, volatility represents uncertainty regarding changes in the cost to redevelop the software in the future.


$90,000
$80,000
$70,000
$60,000
$50,000
$40,000
$30,000
$20,000
$10,000
$0
Persistence
Volatility of Underlying Asset Value