Real options and competitive dynamics in software product release

Tom Cottrell
University of Calgary
Faculty of Management
Finance Area*

June, 2000

Abstract

We examine the software product release decision under monopoly and duopoly market structures. The paper begins with the empirical observation that firms occasionally release software riddled with performance difficulties, commonly called 'bugs.' We next develop models of optimal product release that accommodate the real-world concerns of software developers, especially the trade-offs between early release to capture the benefits of a larger installed base and the deferral of product release to improve product quality. For the monopolist, we consider product release under two regimes: where the network benefit is constant, and where it evolves stochastically. Under constant network benefits, we find that it is in the monopolist's interest to release the product with significant quality. The paper then considers the case where a monopolist faces network benefits that evolve stochastically. We derive the trigger prices for optimal timing of product release. When the network benefit evolves stochastically, we demonstrate that lower-quality products may be released.

The paper then considers the duopoly market structure. We find that the decision to release buggy software turns sharply on the effect

*The author would like to thank Gordon Sick and Mark Cassano for helpful conversations on pricing real options.

of preemption. If the first entrant into the market captures the entire network benefit, in a case something like Bertrand competition, then firms release software immediately upon completion of development and before significant debugging efforts. If the network effect is muted, in something like Cournot competition, we find that firms delay their release. In either case, duopoly product quality is lower than monopoly product quality, although in the case of muted competition, product quality is not dramatically lower than the monopoly case.

The product release problem

During the first decade of development, the microcomputer software industry was characterized by rapid technological progress. Applications that were state of the art in one month might easily be worth half their value in the next if a dramatically improved competitor entered the market. For example, the most popular spreadsheet software in 1980 was Visicalc. Competitive entry ensued, but Visicalc remained the dominant product until 1982, when Lotus 1-2-3 made most Visicalc functionality and performance obsolete. But other software products, graphics or games, were virtually worthless until complementary development provided sound or printing capability. In this environment, consumer product valuation was a difficult task, with significant up- or down- side potential. Since software products are long-lived, consumers faced an important problem in determining the timing of a new application purchase. Consumers balance the immediate benefits of a new piece of software with the possibility that a later, incompatible product will arrive with even greater benefits, or against the more amiable prospect that newer hardware or software technology would complement, and thus increase the value or usability, of the software. Other types of product complementary products include nicely written manuals for learning to use the software, and product training and support from third party providers.

In addition, the number of other adopters of compatible software plays a significant role in this valuation for most kinds of software. First, because the availability of complements depends critically on the installed base of the primary product. But also because users of the primary product may exchange files, insights or provide other network benefits to users of compatible software. Thus, network benefits might evolve erratically depending upon the sales and installed base of the primary product and the development of complementary products.

In this paper, we begin with the empirical observation that from time to time, firm's appear to release software riddled with performance difficulties, commonly called 'bugs.' In a market well known for rapid technological progress, why wouldn't the firm delay release until it was satisfied with product quality? The central question is why firms would release early, risking damage to reputation and significant product support costs, when they might delay product release until the product satisfies some optimal characteristics.

Rapid technological progress in durable goods confronts producers with a classic problem in dynamic programming. Product release today means

that the firm foregoes potential future product innovations. Producers of these goods face the problem that innovation in product or process may significantly improve profits, either through greater functionality (enhancing demand) or through reduced production costs (improving the costs), and so may be reluctant to release a product prior to reaching this optimal value.

Overview

To investigate the question of product quality, we turn next to a review of the relevant economic literature. This is followed by a descriptive section detailing the difficulties inherent in software development, providing anecdotal support for the characteristics required for a more formal model. We then present a formal model of software development, built on recognition of the characteristics presented in the descriptive section and other empirical work on the software development process. The paper concludes with a consideration of the implications for empirical research, and refinements expected in the current model.

The literature on product quality in durable goods

Recent research in the theory of economic obsolescence has examined the implications of product development under rapid technological improvement (Dhebar (1994)). More recent work demonstrates that this problem can be investigated in a variety of contexts, including when products are characterized by network externalities, or when there is the possibility of offering product upgrades (Waldman (1993), Lee and Lee (1998),Grenadier and Weiss (1997)). Under these circumstances, product developers may face additional incentives for rapid market entry with lower product quality.

Monopolist choice of quality

We have known for some time that a monopolist may find it advantageous to provide a less-than-socially-optimal level of product quality, in this case measured as product durability. Early papers on quality argued that monopoly would produce lower quality products than competitive markets, owing primarily to convex costs and the difference between the average and marginal

consumer's willingness-to-pay (Kleiman and Ophir (1966), Levhari and Srinivasan (1969), Schmalensee (1970))

Bulow (1982) and Bulow (1986) give the conditions under which a durable goods monopolist chooses to provide quality below the socially optimal level. Bulow's insight is that in a durable goods setting, with the possibility that second-hand goods from first period sales may affect second period demand, the monopolist would supply lower durability to avoid second-period competition in the used market. In Bulow (1986), the monopolist produces a durable good and is concerned with a time-inconsistency problem: How can the monopolist make a period 1 commitment to maintaining the current price instead of dropping it in period 2?

This so-called time-consistency problem is important. Stokey (1979) identifies the necessary conditions for intertemporal price discrimination, otherwise the firm is subject to the risk that consumers will delay purchase. Dhebar (1994) discusses these implications to identify that adopters will wait to purchase if the present value of expected quality improvements is increasing faster than expected price. Ignoring the network effects for the moment, the customer adopts when valuation exceeds current and expected future expenditures, as price plus the product adoption effort required: $w_t > p_t + a_t$ and $w_t > \delta(p^e_{t+1} + a^e_{t+1})$. If consumers participate in a period t market, we conclude that the quality-price differential is not expected to improve in present-value terms. In this case, the discounted value satisfies $\delta W(q_{t+1}, u_{t+1}) < W(q_t, u_t)$ for some value of the index u. In the model below, we will assume that this condition[1] continues to hold.

A second approach to the durable goods problem is that the monopolist cannot commit to quality improvements in later periods. Shepard (1987) provides an model where the monopolist commits to competition in the second period in order to assure the on-going development of quality products in both periods. Shepard's model assumes the timing of product introductions has the effect of higher quality.

The model developed in this paper differs from the prior work in two ways. Like Bulow, we are concerned with the monopolist's incentives to release lower quality products. Unlike that work on durability, however, there is less concern over second-hand market owing to the rapid technological

[1]In the case under certainty, this assumption is innocuous, because early adopters receive the product below the price offered to later adoptions. Under uncertainty, there is greater scope for debate over the applicability of this assumption.

improvements in the market. Like Shepard, we consider the importance of product quality, and the implications for the timing of new product introductions. Unlike that work, however, we examine these effects in the presence of network effects, so that quality enhancements also occur exogenously. Overall, the paper attempts to contribute by both its empirical grounding in the software market and by the introduction of options pricing methodology to understanding the product release decision.

Software quality in product development

In this section we examine the rationale behind uncertainty in software development. We first establish an empirical argument that quality is costly to provide and difficult to forecast. We emphasize two aspects of this: the process of selecting the appropriate functionality of the product, and the problem of choosing optimal release based on the level of functionality. Based upon the discussion in this section, we will later characterize the uncertainty in functionality as asset volatility, and the debugging process as drift in asset value.

Quality uncertainty in software development

The production of computer software is an inherently uncertain process. A 1979 GAO study of software development contracts, described in Boehm (1981), highlights the significant uncertainties involved in production. Over 50% of the contracts in the study had cost overruns, 60% overran the appointed schedule, 45% contracted for could not be used, and in 29% of the contracts, the software was never delivered. Although there are likely incentive compatibility problems in these contracts, Boehm attributes the significant lack of compliance with the complexity of the software development process itself. Even with excellent programmers, the software development process more generally can be difficult to monitor and manage, and this is particularly true in American firms (see OECD Committee for Information, Computer and Communication Policy (1985), Baumert (1991)). Even after a software program is developed, the process of testing and debugging can significantly extend time to product release.

No software firm would decline the opportunity to release high-quality, bug-free code cheaply. But software development is a highly uncertain enter-

prise, and its successful construction is complex and error-prone. Microsoft's Chris Peters, the program manager for Microsoft Excel, compares software development to traditional manufacturing environments. When manufacturing a physical object, a physical model with locational clues aids design and manufacturing. Peters uses the Boeing aircraft as an example: despite the literally millions of parts in a jet aircraft, placement of the instrument console in the baggage compartment is unheard of. In the development of Microsoft Excel, a product with about the same number of production components, namely 1.5 million lines of code, any single line can affect any other line in difficult-to-forecast ways. The possible interactions of 1.5 million factorial are mind-boggling when a software engineer seeks to find and exterminate a software bug; fixing a problem at one line may produce unanticipated problems in other parts of the code.

And, bugs in software can easily emerge when the software is ported to different computing environments (say MS-DOS to Macintosh) or in the presence of different hardware subsystems, such as the display technology or memory architecture. These complexities are an important impediment to rapid and successful software development on one platform, much less across a variety of hardware platforms. Software engineers model the process of finding and killing bugs with an exponential time distribution: bug discovery and correction is rapid in the early part of product development, but it is a longer and more protracted process as development continues. The implications for a potential entrant are important, especially when we consider the importance of firm reputation in user adoption. A *de novo* entrant simply cannot afford the number of bugs that an established firm can. Customers will allow established firms more errors than an unknown entrant under the assumption that the established firm will eventually be able to fix the problem. If there is a constant dollar expenditure per unit of time, the marginal cost to find and fix the last allowable bug will thus be much higher for de novo entrants. Thus, entrants must spend significantly more to enter the industry.

The quality of product design is much more subjectively evaluated than software reliability. In design, concern is over how well the software accomplishes a specific task. While users care about reliability, functionality means that the interface is intuitive or that the charts are pretty. Functionality concerns have subtle implications for consumer demand.

Further, users cannot anticipate all functional aspects of the software that will be needed over the lifetime of the product. Since software is a durable

good, comparative product quality varies over time with changes in what is available in the market. Users sink the investment in understanding the application interface (e.g. command structure or functionality) with the hope that there will later be upgraded versions that are of similar relative quality, and that the supplier firm will survive to provide those improvements. That is, users not only worry about product quality statically (what the product currently offers), but also dynamically, as a function of the survival of the firm and improvement of the product. In a transaction between a single buyer and seller, it is common to insist that software source code be placed in escrow to reduce the loss should the producing firm exit the industry. In a retail market, users are not sophisticated enough to benefit from escrow code nor have they invested enough to warrant redevelopment, so they instead seek signals of producer survival. But survival is not enough, users would like to know that the investment sunk in learning the product today will benefit them in future periods, including when competing products are upgraded. Often consumers respond to these fundamental uncertainties by choosing "industry standard" applications, so anointed by a consensus of industry experts. This structure may be entirely efficient in any period, but it does impose a cost on potential entrants who must compete away an incumbents installed base.

Interface standards example

As an example of the difficulty in evaluating the quality of software functionality, we turn to the development of interface standards. As firms in this industry emerged from garages to office suites, one key technology development was a standardization of the interface to applications software. The evolution of the user interface for most software applications has occurred slowly over the prior decade despite significant investment in user interface research. Today's graphical user interface standard (GUI) gained acceptance only as computing power came more cheaply. Key aspects, one might argue all, of GUIs benefits were available during the early 1980s in character-based application interfaces such as GEM, TOPS, Desqview, or Mondrian . However, the Graphical iconographic interface (MS-Windows) that eventually came to dominate the PC market was hampered throughout the 1980's by lack of computing power, and effectively unavailable to non-Mac users. User interface clearly affects productivity, yet with only an emergent understanding of the potential interface advantages, a user's choice over interfaces was path-dependent. By 1990, users appeared to prefer GUI with the explosion

of Windows 3.0 product development. It is difficult to evaluate why these users would not have chosen the Macintosh interfaces well before that. We offer no solution to this puzzle; it merely illustrates that the evolution of user tastes made it difficult both for users and for software developers to measure quality in such an uncertain environment.

The Mythical Man-month

In Brooks (1975), a model of software development is described with the surprising result that adding more people to a software project actually *delays* progress. Based on experience in the development of software for the IBM 360, Brooks relays the importance of understanding the development project from the start, and advises against addition of additional people to the product development team unless additional development time is expected.

By the time software developers were creating products for microcomputers, the importance of these lessons was widely understood. Of the most successful products, the majority were designed and implemented by either a single individual or a small team of programmers. Examples of these success stories are found in Lammers (1986). In this product development environment, initial team-size placed an important constraint on the ability of the team to make progress in development and debugging. We apply Brooks' insight in the later development of a model.

Summary

We develop a model of software based on these 'stylized facts.' First, that the value of software follows a diffusion process. Second, that at the time that the project is undertaken, there is a significant delay until product release. We note that there have been a number of products demonstrated well ahead of product release, a phenomenon known as 'vaporware', and that ultimate release of the product requires a significant debugging period. Third, we assume that the debugging process is complex, time-consuming, and follows the process described in the software development texts. While for modelling purposes it may be convenient to take an alternative approach, we will impose restrictions on functional forms based upon these 'stylized facts.'

Formal models of software development

Based on the above discussion, we describe a formal model of the software debug and release process. We emphasize two aspects of this. First, the network benefits interaction with the timing of product release. Second, the option value that the firm derives from the opportunity to delay product release.

Monopolist software development under certainty of valuation

We now turn to a software development model where the network benefits are well understood and do not evolve with uncertainty.

Modeling the debugging process

Software developers are well aware of the problems of debugging software code. Bugs are usually easy to find and fix at first, but as development proceeds, identifying the source of problems, much less determining the solution, becomes an increasingly difficult task.

A monopoly software developer maximizes profit with respect to a one-time development cost D followed by debugging costs prior to product release date T:

$$\max_T \Pi = -D - \int_0^T ce^{-\rho t}dt + \int_T^\infty \theta^* \lambda \nu (t-T)e^{-\rho t}dt$$

Define terms as:

D =product development (design and coding) costs.

c =instantaneous cost of debugging, measured as programmer effort.

ρ =discount rate.

T =product release date.

r =debug rate, determines rate at which product improves if $r > 0$.

θ^* =optimal choice of product quality. Quality at any instant lies in $[0, 1]$ as: $\theta = 1 - \frac{l_0}{l_0 rt+1}$. This functional form is chosen from an empirical study of the debugging process. The equation of motion for product quality improvement sets $\frac{l_0}{l_0 rt+1}$ as the instantaneous failure rate for a program with initial failure rate l_0. (cf Pressman (1992) page 637). The model is a logarithmic Poisson execution-time model which takes the form: $f(t) = \frac{1}{r}\ln(l_0 rt + 1)$. In

this case, f is the cumulative number of expected failures once the software has been tested for execution time t. Normalizing $l_0 = 1$, at any point in time $\theta(t) = \frac{rt}{rt+1}$, and at the time of product release is equal to $\theta^* = \frac{rT}{rT+1}$.

λ =customer arrival rate. Homogeneous consumers arrive with utility function $U(\theta, N, p) = \theta\nu(N) - p$. The monopolist prices at consumer willingness to pay $p = \theta\nu(N)$.

ν =willingness-to-pay function, is increasing in installed base. This models consumers valuation of the product according to evolving size of network. For simplicity, we use a linear network benefit function $\nu(t-T) = \nu*(t-T)$.

$\delta = e^{-\rho t}$ is the discount factor.

Under these conditions, the constraint for profit maximization is the following:

$$\frac{d\Pi}{dt} = \delta\left(-c + \frac{r}{rt+1}\frac{\lambda\nu}{\rho^2} - \frac{r^2t}{(rt+1)^2}\frac{\lambda\nu}{\rho^2} - \frac{rt}{(rt+1)}\frac{\lambda\nu}{\rho}\right)$$

Applying this first order condition, where $\theta' = \frac{r}{(rt+1)^2}$ for product release is:

$$c = \frac{\lambda\nu}{\rho^2}(\theta' - \rho\theta)$$

This is the condition where marginal revenue is equal to marginal cost. Rearranging, the left hand side is the cost of waiting an additional instant dt: the cost of debugging effort c plus the deferral of revenues at interest rate ρ. The right hand side is the marginal benefit of this deferral, which is the present value of the benefit of the increase in marginal quality.

$$c + \rho\theta\frac{\lambda\nu}{\rho^2} = \theta'\frac{\lambda\nu}{\rho^2}$$

Since the value of the revenue stream is $R(t) = \theta\frac{\lambda\nu}{\rho^2}$ we get

$$R'(t) = \rho R(t) + c$$

If we set the instantaneous debugging cost c is 0, then the ratio of improvement to current value is the discount rate:

$$\frac{R'(t)}{R(t)} = \rho$$

When $c > 0$, the solution is a function of customer arrival rates, cost, the discount rate and valuation of the network benefits. Solving the first order

conditions yields a quadratic in t, and since these are wait times, only $T^* > 0$ are admissable. Thus

$$T_\nu^* = -1/2\,\lambda\,\nu\rho - c\rho^2 + 1/2\,\frac{\sqrt{\lambda\,\nu\,(\lambda\,\nu\rho^2 + 4\,(c\rho^2 + \lambda\,\nu\rho)\,r)}}{(c\rho^2 + \lambda\,\nu\rho)\,r}$$

When $c = 0$, the optimal release time is

$$T^* = -\frac{1}{2}\,\frac{\rho - \sqrt{\rho\,(\rho + 4\,r)}}{\rho\,r}$$

Which demonstrates that the optimal release is a function of the discount rate ρ (the cost of delaying revenue) and the rate of quality improvement r (the benefit in future revenue from increasing customer willingness to pay).

The comparative statics on optimal release demonstrate that the wait time is strictly decreasing in the discount rate ρ and the quality improvement parameter r for all reasonable[2] values of r (the debug rate) and ρ (the discount rate). For optimal choice of T, we take the derivative at the optimal to yield:

$$\frac{dT}{dr} = -\frac{1}{\sqrt{\rho^3}\sqrt{\rho + 4r}}$$

$$\frac{dT}{d\rho} = -\frac{1}{2}\,\frac{\sqrt{\rho^2 + 4\rho r} - (\rho + 2r)}{r^2\sqrt{\rho^2 + 4r}}$$

These are negative for reasonable values of the parameters, with the result that an increase in r or ρ will shorten the time to optimal release.

Network effects

In the monopolist model outlined above, the firm has an incentive to increase product quality with increases in the network benefits. However, these gains are at the expense of foregone revenues and the cost of continuing to debug.

[2]There are sensible constraints on discount rates ρ and r such that the firm will choose to delay release - $r > \rho > 0, \lambda > \rho$. For most products, the rate of quality improvement will initially exceed the discount rate, and becomes equal at a later point, for the optimal solution to be unique. Our quality improvement function is concave, and the optimal release point has a single crossing value.

Since the expression quickly becomes complex[3], we choose to simulate the results for reasonable values of the parameters.

For purposes of simulation, we set the following values:

- $\lambda = 1$: customer arrival rate
- $\rho = 0.1$: discount rate of 0.1.
- $c = 1/2$: instantaneous cost of debugging
- $r = 1.1$: debugging effectiveness ($r > 1$)
- $\nu = 1.2$: network effect on willingness to pay ($\nu > r$)

As long as the monopolist recovers the returns from waiting in the form of higher expected reveneues, it is profit-maximizing to delay product release. With increases in the network effect, there is an expected increase in the time to product release. To illustrate this, figure 1 graphs the optimal release (T^*), and the quality at optimal release ($\theta(T)$), against a range of values for the network effect ν.

Discussion of model results

The optimal release is sensitive to the network effect for lower ranges of the parameter ν, but for values above 1, there is little incentive for additional delay. For this parameterization, the optimal release time is bounded at ≈ 2.595, and the optimal release is already ≈ 2.5 as $\nu = 1$. Network effects above $\nu = 1$ do not increase quality significantly above the level reached as quality is bounded at ≈ 0.8.

At least for the monopolist, the incentives to reduce quality below efficient levels does not seem an important constraint. In part, this is because the monopolist does not face the same constraints as in Bulow (1986), where in subsequent periods the monopolist competes with its own prior period sales.

[3]

$$\frac{dT}{d\nu} = -1/2\,\lambda\,\rho + 1/4\,\frac{\lambda\,\left(\lambda\,\nu\rho^2 + 4\,\left(c\rho^2 + \lambda\,\nu\rho\right)r\right) + \lambda\,\nu\,\left(\lambda\,\rho^2 + 4\,\lambda\,\rho\,r\right)}{\sqrt{\lambda\,\nu\,\left(\lambda\,\nu\rho^2 + 4\,\left(c\rho^2 + \lambda\,\nu\rho\right)r\right)}\,\left(c\rho^2 + \lambda\,\nu\rho\right)r} - 1/2\,\frac{\sqrt{\lambda\,\nu\,\left(\lambda\,\nu\rho^2 + 4\,\left(c\rho^2 + \lambda\,\nu\rho\right)r\right)}\lambda\,\rho}{\left(c\rho^2 + \lambda\,\nu\rho\right)^2 r}$$

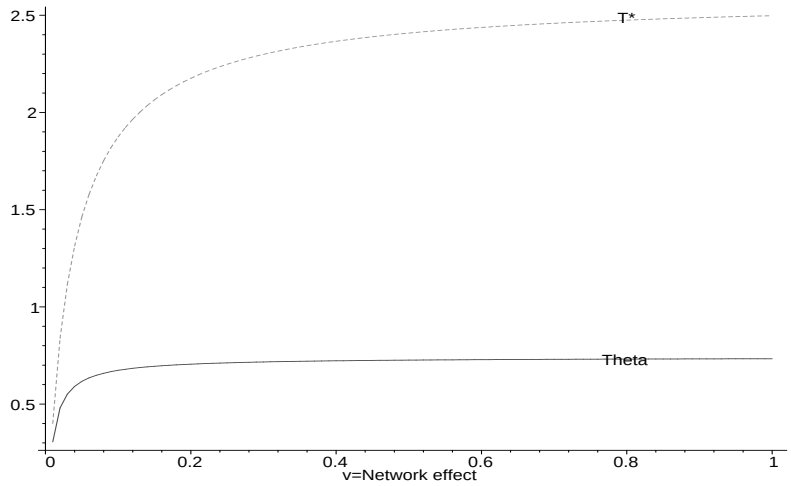

Figure 1: $\theta(T)$ and T versus network effect ν

Monopolist product development under uncertainty

We next examine the firm's incentives for product release when network benefits, and by implication, project value, evolve stochastically.

To review, $\theta(t)$ denotes the state of the quality for software development for the product under development, and ν denotes the customer willingness-to-pay, primarily a network benefit. As described above, the quality of software is measured relative to the evolution of other products in the industry, and there is significant uncertainty over this process. In this model, the firm has the option to delay development costs D and the initiation of debugging costs c until time t_0, and to delay product release until time T. We can model the value of the project at t_0 as

$$\max_{t_0,T} \Pi = -D - \int_{t_0}^{T} ce^{-\rho t} dt + \int_{T}^{\infty} \theta^* \lambda \nu (t-T) e^{-\rho(t+T)} dt$$

For simplicity, we assume that

$$d\nu = \alpha\, \nu\, dt + \sigma\, \nu\, dz$$

where

- α drift rate of ν.
- σ is the proportional variance parameter.
- dz is the increment of a standard Wiener process.

In product development under certainty, the developer makes a product release timing decision as soon as $\Pi > 0$. When product development proceeds under uncertainty, the developer makes two timing decisions. The first timing decision is choice of t_0, the time at which to sink development cost D and begin the debugging process. The second timing event is the decision to release the product, denoted T, with quality level $\theta(T - t_0)$. Under the classic Marshallian NPV rule, the initial investment occurs whenever the expected profit is greater than 0: $\mathcal{E}[\Pi] > 0$. Under the optimal option pricing rule, development occurs when Π^* reaches the appropriate boundary condition.

Under this set of assumptions, we apply Itô's lemma to yield

$$d\Pi^*(\nu; t_0, T) = \frac{1}{2}\Pi^*_{\nu\nu}\nu^2\sigma^2\,dt + \Pi_\nu\,d\nu$$

and since ν evolves stochastically,

$$d\Pi^*(\nu; t_0, T) = \frac{1}{2}\Pi^*_{\nu\nu}\nu^2\sigma^2\,dt + \Pi_\nu\,(\alpha\nu\,dt + \sigma\nu dz)$$

It seems unrealistic to assume trading in the underlying asset (the software product under development). It is, therefore, useful to employ a pricing model that does not assume asset liquidity. Following Sick (1995), we apply the consumption capital asset pricing model (C-CAPM) to derive boundary conditions that allow a solution to the partial differential equation.

Using risk-neutral valuation of the partial differential equation, we derive the following conditions:

$$\frac{1}{2}\sigma^2\nu^2\Pi_{vv} + \Pi_t + \hat{\alpha}\nu\Pi_\nu = \rho\Pi + c$$

where: $\hat{\alpha} = (\rho\nu - \delta)$ is the risk-neutral drift. The approach is to equate the required return on the option with the expected capital gains and and dividend cash flow, which in our case is an outflow in the form of the debugging cost. As described elsewhere[4], the components of this equation allow a risk-neutral interpretation. The first term on the left-hand-side is the so-called

[4]see Sick (1995) page 651, Dixit and Pindyck (1994) chapter 6, circa page 180, or Wilmott *et al.* (1995) pg 114

'Itô adjustment', or the change in option price imposed by the random variation in ν and the nonlinear response of the option value. The second term describes the dependence of the option value on time. The third term is the response of the option valuation to a shift in the value of the underlying asset. The right-hand-side is the opportunity cost of the option: a risk-neutral investors required return on the option plus the direct cost c of debugging.

As with the risk-neutral valuation in Sick (1995), ν reflects all of the information about systematic risk necessary for contingent-claims valuation. That is, the valuation ν is sufficient information for determining the value of the project that is contingent on future values of ν.

Boundary conditions

The optimal exercise will be a function of how long the product has been being debugged (the time dependence of θ) and diffusion of the underlying value ν. The boundary condition requires that, at the time of exchange of one risky asset for the other, the value of the option is equal to the value of the asset (Miller (1993), Margrabe (1978). From this, and the smooth- pasting condition, we determine that the choice of optimal release date will depend on two parameters. First, the date at which the firm began debugging (t_0), with product quality $\theta(t - t_0)$ at time t. Second, the evolution of customer willingness to pay for the software product, ν. This dependence on duration leads to two boundary conditions for the optimal release time T:

$$B_T = \Pi_T \tag{1}$$

$$B_\nu = \Pi_\nu \tag{2}$$

The equation for the first is essentially the same (after accounting for the t_0 term) as we earlier derived under certainty. The second is:

$$\frac{r\left(T^* - t_0\right)}{\left[r\left(T^* - t_0\right) + 1\right]} \frac{\lambda\nu}{\rho^2}$$

For illustration, we assume for the moment that the strike price D has been paid, and that the project debugging duration is $t - t_0$, where product quality is $\theta(t - t_0) = 0.8$. Figure 2 shows the value of keeping the project alive, and the smooth-pasting condition where it is optimal to exercise the option and release the product. There are three 'smooth-pasting' results,

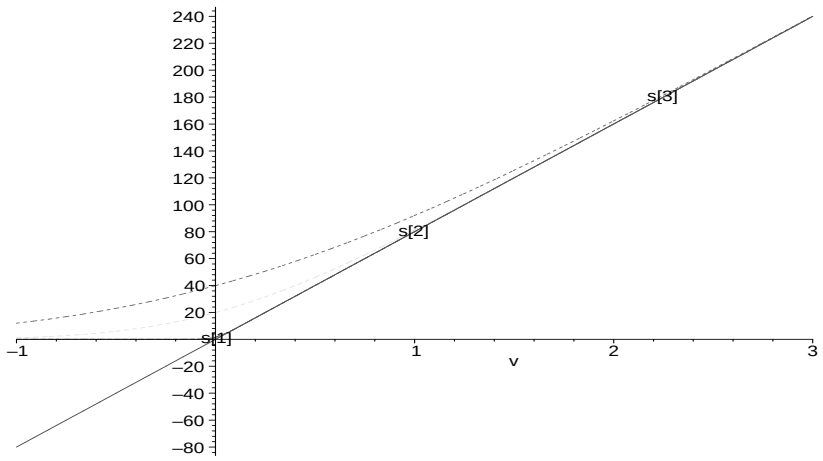

Figure 2: Option value

denoted $s[\cdot]$, for increasing values of volatility. The valuation approach is based upon Rose (1993) and Miller (1993). The graph shows that exercise occurs for values of ν^* that are increasing in the volatility of the underlying process.

Thus, the network benefit has to be relatively more significant before the firm releases the product. As the volatility increases, the smooth-pasting condition requires a higher strike price in the underlying network valuation.

Dynamic programming solution: stage 2

The decision to begin the debugging process requires that the firm form expectations about profits. The firm thus solves a two stage dynamic optimization process. First, it determines optimal stage 2 product release trigger prices ν^* for a given quality level $\theta(T)$. Then, given these solutions, determines the stage 1 expected value of the opportunity.

To determine the correct trigger price, the firm requires a decision rule that matches the optimal strike price and the level of quality. The optimal choice of ν^* will vary depending on the level of quality $\theta(t)$, so that $\nu^*(\theta)$. Each θ has an implied project duration, and figure 3 shows the required

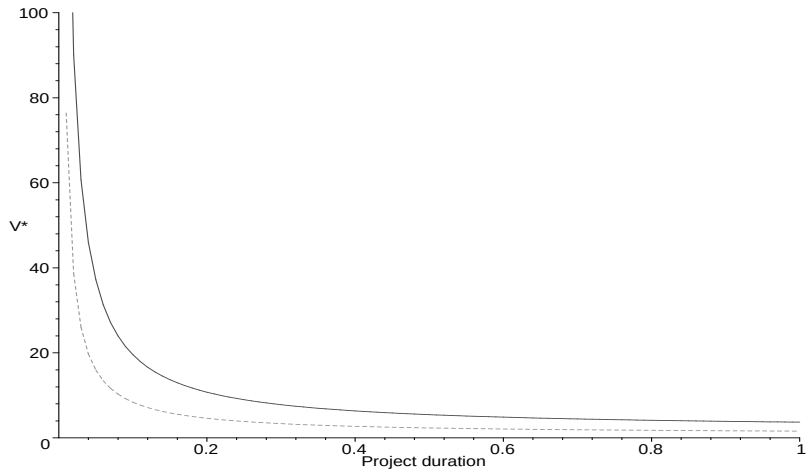

Figure 3: ν^* versus optimal release

trigger price for a project of duration $T - t_0$. For comparison, the optimal ν^*, T are shown for two levels of volatility, with the higher volatility values closer to the origin. This distribution, then, is the solution to the stage 2 decision criteria.

Dynamic programming solution: stage 1

The stage 1 decision rule is dependent on the schedule of stage 2 payoffs for optimal product release. The firms thus require a decision rule to maximize stage 1 given that stage 2 decisions will be optimal. If we denote the network valuation at stage 1 and 2 as ν_1, ν_2, then the payoffs in stage 1 depend on the joint distribution of ν_2^* and T and the current network valuation ν_1. As with the stage 2 decision, there is a smooth-pasting condition where optimal product release ν_1^*. The rationale for this decision is similar to the model presented in Grenadier and Weiss (1997), where the timing of technology adoption is dependent upon the the technology diffusion process.

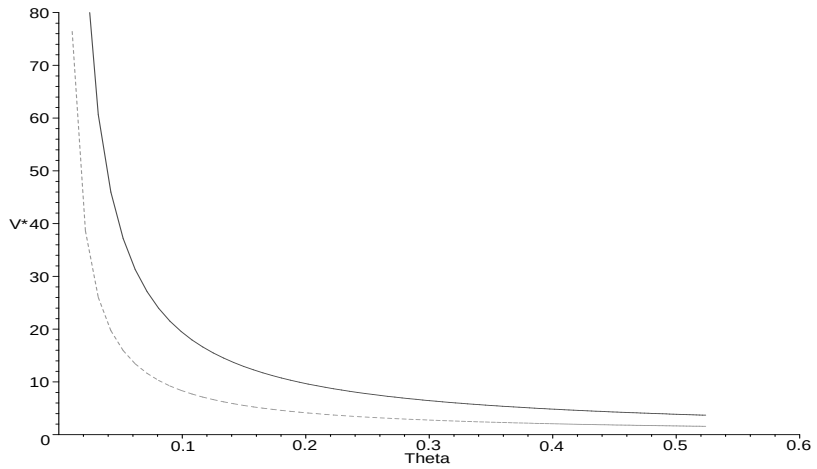

Figure 4: θ versus ν^*

Discussion

The result here is in contrast to product development under certainty. Under certainty of network benefit, product valuation is strictly increasing with time. When network valuation is uncertain, there are levels of network effect significant enough to induce product release, despite the fact that the valuation may decrease over time, and despite the apparently poor quality of the product. In fact, the implied product quality declines as the network benefit increases. This is depicted graphically in figure 4. One interpretation of this graph is that there is a certain window of opportunity available in product release if the network benefit is significant. If the network benefit is high, then the firm should release software of lower reliability. If the network benefit is small, then product quality dominates the gains from early entry to the network.

We turn next to differences in market structure as a rationale for early product release.

Cournot Duopoly

We have explored the monopolist's incentives for product release under various assumptions about the context of competition. The model under certainty shows that, except for very small values of the network effect, there is little incentive for the monopolist to release a product early. However, under uncertainty, conditions can be such that it is in the monopolist's interest to release a product of dubious quality. When there is competition over the installed base, however, there may be opportunity for product release with very low quality in order to capture a larger installed base. Other researchers have considered the effect of entry as a real option to preempt competitive entry or expansion in a competitive industry (Trigeorgis (1991), Pindyck (1993)). Lambrecht and Perraudin (1998) looks more explicitly at the preemption argument in the real option setting, but does not model our differences in customer response or the installed base effect. Recent work by Aguerrevere (1999) shows the effect of capacity expansion in electricity markets in a competitive market. What is interesting about this work is the timing decision that is not dissimilar from that considered here in determining the competitive effect.

In order to model the product release decision under competition between firms A and B, we allow for two customer types, a and b. Consumer's have preferences for products exogeneous to the producing firm. We normalize the population to 1 to allow μ to represent the consumer that is just indifferent between products from A and B. By construction, $\mu\%$ of consumers prefer A. Despite this preference, some measure λ_i of customers will choose $j \neq i$ if the network benefit warrants it. Purchasers, then, have measure $\lambda_i = \mu + (t_i - t_j)\nu$, where ν is a term denoting the persistence of the network benefit in determining the rate at which customers drift away from making their preferred choice.

The strategy choices in this game are product release dates t_i, t_j, and firms have instantaneous costs of debugging of c_i. A Nash equilibrium is a pair of product release times $\{t_i^*, t_j^*\}$ where each t_i in the pair is an optimal response to its t_j counterpart.

We assume that firm i releases first, yielding profits:

$$\max_{t_i} \Pi^i = -D - \int_0^{t_i} c_i e^{-\rho t} dt + \int_{t_i}^{t_j} \lambda \, \nu \, \theta^* e^{-\rho t} dt + \int_{t_j}^{\infty} \lambda_i \, \nu \, \theta^* e^{-\rho t} dt$$

and firm j releases next ($t_i < t_j$), yielding profits:

$$\max_{t_j} \Pi^j = -D - \int_0^{t_j} c_j e^{-\rho t} dt + \int_{t_j}^{\infty} \lambda_j \, \nu \, \theta^* e^{-\rho t} dt$$

To find a Nash equilibrium strategy, we seek a pair of product release times that satisfies the properties of the equilibrium, namely the first order condition that:

$$\frac{\partial \Pi^i}{\partial t_i} \equiv \Pi_i^i \left(t_i^*, t_j^* \right) = 0$$

and the second order condition that $t_i = t_i^*$ is a maximum:

$$\frac{\partial \Pi^i}{\partial t_i \partial t_i} \equiv \Pi_{ii}^i \left(t_i^*, t_j^* \right) < 0$$

The optimal release for firm I is the solution to the first order condition of the profit function. If we denote the revenue function $R_i(t) = \frac{\lambda_i \nu \theta}{\rho}$ and $R(t) = \frac{\lambda \nu \theta}{\rho}$ then we derive a condition where marginal benefit of waiting is equal to the marginal cost of waiting, the same condition as in the monopoly case. To show this, let $\xi = \frac{e^{-\rho t_j}}{e^{-\rho t_i}}$, then

$$c + \rho R(t) = R'(t) \ (1 - \xi) + R_i'(t) \xi$$

$$c + \rho R(t) = \frac{\lambda \nu \theta' \ (1 - \xi)}{\rho} + \frac{\lambda_i \nu \theta' \, \xi}{\rho}$$

The LHS of this is the cost of waiting: the instantaneous cost of effort c plus the instantaneous opportunity cost of delaying revenue. The benefit of waiting is the improvement in product quality that produces revenues from customers $\lambda_i + \lambda_j$ until t_j, and revenues from λ_i thereafter. Under these settings, the optimal release time for the first firm is contingent on the value of ν in relation to the delay.

If $\nu * (t_i - t_j) > \lambda$, then the first firm to release will gain all of the market. Whenever this is the condition, both firms will have an incentive to ϵ-preempt the market, since the follower earns zero profits. This is the analogous to a Bertrand Nash equilibrium in the classical one-shot game. The product quality θ when $\nu * (t_i - t_j) > \lambda$ will be close to zero, as the firm will release immediately. If instead $\nu = 0$, neither firm is worried about the other pre-empting the market. Developing a product with $\theta > 0$ is still possible, as the firm benefits from product improvement without eventual loss

of market share. Product quality will be lower than the monopoly case, however, since the firms will choose to ship the product before the monopolists optimal release date. The justification for early release is two-fold. First, there are, opportunities to gain monopoly profits in the interim where only one firm has entered. Second, since they share the market, marginal product improvements bring lower increases in revenues than in the monopoly case; the marginal benefit of waiting is reduced while marginal cost remains the same.

It will not affect our analysis if we now normalize $\lambda = \lambda_i + \lambda_j = 1$. Then the profit that the duopolist seeks to maximize is:

$$-D + \frac{c\,(e^{-\rho\,t_i} - 1)}{\rho} + \frac{v\theta\,\left(-e^{-\rho\,t_j} + e^{-\rho\,t_i}\right)}{\rho} + \frac{(\mu + (-t_j + t_i)\,\nu)\,v\theta\,e^{-\rho\,t_j}}{\rho}$$

with the following first order condition:

$$c + \rho\frac{v\,\theta}{\rho} = \frac{v\theta'\,(1-\xi)}{\rho} + \frac{\mu\,v\,\theta'\xi}{\rho}$$

Figure 5 shows reaction curves for these firms using the same parameter set as earlier with the monopolist. The Nash equilibrium in this case results in product quality below that of the monopolist as a result of 1) sharing the market since the relative market size is $\frac{1}{2}$ that of the monopoly market, and 2) a strategic interaction in competition for the early entrant monopoly position. Products are released more quickly with lower quality [5] in this market structure than for the monopolist.

Discussion

There are several useful points to consider in these models. The first is that product quality is determined by two factors: market structure in its effect on competition for the installed base, and the evolution of network benefits. The market structure result identifies that when network effects are stable or constant, a monopolist will produce quality software. While the model here allows the monopolist to capture the gains from quality improvements, even capturing a portion of the benefit as the network grows provides the monopolist with an incentive to enhance quality in welfare-improving ways. If,

[5] Products are released at the following times: ($t_i^* = 1.23 < t_m^* = 2.86$) with lower quality ($\theta_i = 0.58 < \theta_i = 0.76$) These results can be characterized analytically.

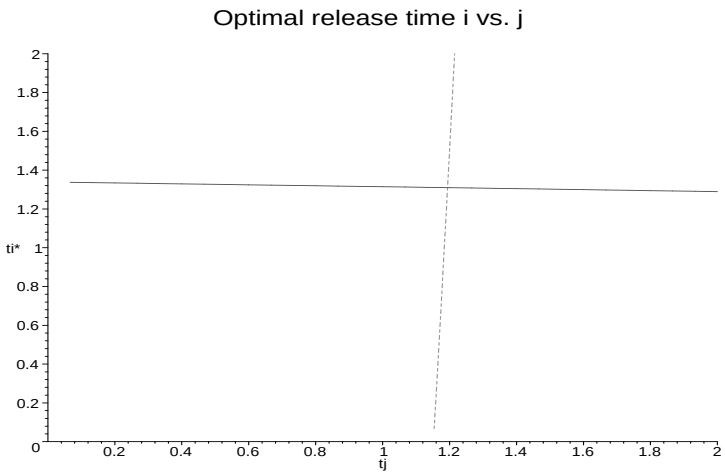

Figure 5: t_i^* versus t_j^*

instead, network effects evolve stochastically, the monopolist may be release lower-quality products in an effort to capture early profits in the product.

While we have only considered the duopoly case in the non-stochastic context, the results are likely to be accentuated in the case where duopolists compete over installed base where network benefits evolve stochastically. In the case of great benefits to pre-emption, this will be reflected in lower strike prices. When there is less benefit to pre-emption, the strike price will shift up towards the monopoly strike-price. In future work, these considerations should be addressed.

It is tempting to apply the implications of this research to recent U.S. antitrust action by the Department of Justice against Microsoft. Careful reading of the judiciary's findings of fact suggest that Microsoft paid very careful attention to the conditions under which it could ignore competitor activity in a market, because its monopoly position was not threatened, and conditions under which early product release was adviseable to protect itself from the installed base effect. In terms of the model here, Microsoft strategy would pay careful attention to ν as the key parameter.

Conclusion

This paper set out to determine the conditions under which product quality was affected by the characteristics of the market structure and the evolution of network benefits. The model shows that where the network effect is constant, the monopolist has a strong incentive to provide high quality software as long as customers are willing to pay for it. If the evolution of the network effect is stochastic, following geometric brownian motion, then the monopolist may find it optimal to release lower-quality software. In either case, the monopolist, has no concern for the strategic effect on installed base, since it is the only game in town,

A duopoly market structure is in strong contrast to this result. The incentive to release a software product early in order to gain installed base has a dramatic effect on the willingness to release bug-riddled products. But the nature of this effect depends critically on the importance of the installed base in consumer purchase decisions. If market competition is fierce, that is, if small differences in installed base make significant differences in the purchase decision, lower quality products are released immediately in order to capture market share. If market competition is muted, that is, if preemption of a competitor's installed base is unlikely, product quality is lower than the monopolist, but not dramatically. Product quality turns carefully on this preemption effect.

Future research should apply the implications of these model to actual product release decisions of firms in order to determine the implications for strategy and public policy.

References

Aguerrevere, F. L. (1999). Time to build, operating options and optimal capacity choice: Implications for equilibrium output prices.

Baumert, J. (November 1991). New SEI maturity model targets key practices. *IEEE Software*, **8**(6), pg 78–9.

Boehm, B. W. (1981). *Software engineering economics.* Prentice-Hall, Englewood Cliffs, NJ.

Brooks, F. P. (1975). *The mythical man-month: Essays on software engineering.* Addison-Wesley, Reading, MA.

Bulow, J. I. (1982). Durable goods monopolists. *Journal of Political Economy*, **90**, 314–332.

Bulow, J. I. (1986). An economic theory of planned obsolescence. *Quarterly Journal of Economics*, pages 729–749.

Dhebar, A. (1994). Durable goods monopolists, rational customers, and improving products. *Marketing Science*, **13**, 100–120.

Dixit, A. K. and Pindyck, R. S. (1994). *Investment under Uncertainty.* Princeton University Press, Princeton, NJ.

Grenadier, S. R. and Weiss, A. W. (1997). Investment in technological innovations: An option pricing approach. *Journal of Financial Economics*, **44**, 397–416.

Kleiman, E. and Ophir, T. (1966). The durability of durable goods. *Review of economic studies*, **33**, 165–178.

Lambrecht, B. and Perraudin, W. (1998). Real options and preemption under incomplete information. *Real Options Conference, Leiden*, pages 1–41.

Lammers, S. (1986). *Programmers at Work: 1st Series.* Microsoft Press, Redmond, WA.

Lee, I. H. and Lee, J. (1998). A theory of economic obsolescence. *The Journal of Industrial Economics*, **XLVI**, 383–401.

Levhari, D. and Srinivasan, T. N. (1969). Durability of consumption goods: Competition versus monopoly. *The American Economic Review*, **59**, 102–107.

Margrabe, W. (1978). The value of an option to exchange one asset for another. *Journal of Finance*, **33**, 177–186.

Miller, R. M. (1993). Option valuation. In H. R. Varian, editor, *Economic and financial modeling with Mathematica*, chapter 12, pages 266–285. TELOS, Santa Clara.

OECD Committee for Information, Computer and Communication Policy (1985). *Software, an emerging industry.* OECD Publications, Paris, France.

Pindyck, R. (1993). A note on competitive investment under uncertainty. *American Economic Review*, **83**, 273–277.

Pressman, R. S. (1992). *Software engineering: A practitioner's approach.* McGraw-Hill, Inc., NY, NY.

Rose, C. (1993). Bounded and unbounded stochastic processes. In H. R. Varian, editor, *Economic and financial modeling with Mathematica*, chapter 11, pages 239–265. TELOS, Santa Clara.

Schmalensee, R. (1970). Regulation and the durability of goods. *Bell Journal of Economics*, **1**, 54–64.

Shepard, A. (Autumn 1987). Licensing to enhance demand for new technologies. *Rand Journal of Economics*, **18**(3), p 360–8.

Sick, G. (1995). Real options. In R. A. Jarrow, V. Maksimovic, and W. T. Ziemba, editors, *Finance*, chapter 21, pages 631–690. Elsevier, Amsterdam.

Stokey, N. L. (1979). Intertemporal price discrimination. *The Quarterly Journal of Economics*, **93**, 355–371.

Trigeorgis, L. (1991). Anticipated competitive entry and early preemptive investment in deferrable projects. *Journal of economics and business*, **43**(2), 143–156.

Waldman, M. (Feb 1993). A new perspective on planned obsolescence. *Quarterly Journal of Economics*, pages 273–283.

Wilmott, P., Howison, S., and Dewynne, J. (1995). *The mathematics of financial derivatives: A student introduction*. Cambridge University Press, Cambridge, UK.