Software Functionality: A Game Theoretic Analysis

KAI LUNG HUI AND KAR YAN TAM

KAI LUNG HUI is assistant professor in the School of Computing at the National University of Singapore. He received his Ph.D. in Information Systems from the Hong Kong University of Science and Technology. His research interests include consumer choice of IT products, digital product characteristics and competitions, and adoption of information technology. He has papers published or forthcoming in *Communications of the ACM, IEEE Transactions on Engineering Management, Information and Management, Journal of Management Information Systems,* and *Journal of Organizational Computing and Electronic Commerce.*

KAR YAN TAM is currently Professor of Information & Systems Management and Director of the Center for Electronic Commerce at the Hong Kong University of Science & Technology. His research interests include adoption of information technology, electronic commerce, and information technology applications. He has published extensively on these topics in major management science and information systems journals. He serves on the editorial board of a number of information systems journals and has extensive consulting experience with major local and overseas companies including HSBC, IBM Hong Kong, Sun Microsystems, and HK Telecom.

ABSTRACT: Digital products are now widely traded over the Internet. Many researchers have started to investigate the optimal competitive strategies and market environments for such products. This paper studies the competitive decisions made about software, a major class of digital products that can be easily sold through computer networks. Instead of focusing on traditional competitive dimensions, such as price or quantity, we study the number of functions that should be incorporated into the software. Using game theoretic analysis, we show that there is no fixed strategy that is optimal for software developers in a duopoly market with one-stage simultaneous moves. This happens because, given one developer's decision, there is always an incentive for the other developer to deviate and achieve higher payoffs. Nevertheless, a unique reactive equilibrium does emerge if we consider the two-stage variation of the model, where the two developers both enjoy substantial profits by serving different segments of the market. Essentially, the first mover commits himself to a certain functionality level that induces a rational follower to target his software to the (previously) unattended segment. We discuss our results in light of scale economies in the software development process and market segmentation.

KEY WORDS AND PHRASES: development cost, digital products, functionality, game theory, software.

Journal of Management Information Systems / Summer 2002, Vol. 19, No. 1, pp. 151–184.

0742–1222 / 2002 $9.50 + 0.00.

THE INCREASING POPULARITY OF THE INTERNET has brought about in time developments in electronic business. Although the Internet cannot actually carry physical products, it does transmit electronic bits, which is the single and most important constituent of digital goods. Owing to its fast transmission speed and extensive connectivity, the Internet is capable of delivering electronic information to massive populations in minutes, or even seconds. Such an efficient delivery scheme is almost impossible with traditional channels, such as manual deliveries or postage. This partly explains why more and more digital products are being sold over the "information superhighway."

Among different types of digital products, software is probably one of the most widely traded products over the Internet. According to Gurbaxani and Mendelson [23], corporate software expenditures grow almost at the same rate as hardware products (around 8 percent per year) and this growth follows an exponential pattern. This trend will likely escalate given the growing popularity of the Internet. Small to medium-sized software vendors can now easily set up Web presences and sell their products online. Today it is very common to find linkages to over thousands of software vendors on popular shareware sites like TUCOWS (www.tucows.com) or CNET Download.com (download.cnet.com). Given this staggering growth and the appearance of new marketing concepts like limited period free evaluations or limited functionality evaluation copies, the competitive conditions of software products deserve higher attention from information systems (IS)/information technology (IT) researchers.

The present study develops a game theoretic model that looks at the functionality decisions of software vendors. It is well recognized that the functionality of a software product plays a decisive role in its reception by customers. On many occasions, customers are often advised to compile a list of functional specifications and to check them against the market offerings before they select the appropriate software packages or systems [6, 20, 29, 45]. Since software is indeed sets of instructions that perform specific tasks, different combinations of such instructions may appeal to different customer segments. A movie creator may need powerful 3-D modeling software that can handle millions of polygons to generate dazzling animation, whereas an individual designer may only require modeling software that can create simple scenes and graphics. Therefore, different customers may have different functional needs, and it is vital for software vendors to position their software products appropriately.

Using a parsimonious competitive model, we seek to address the following research questions in this study:

> What are the optimal functional levels that software developers should set for their software programs? Are they consistent across different market structures?

Obviously, more powerful software implies higher development costs from the software developers, which may directly translate into higher product prices. At the same time, more powerful programs may appeal to wider customer groups, since they can fulfill the task requirements of more demanding users. This may eventually lead to substantial increments in product sales. How the cost and sales factors affect the final functionality settings is a major issue that we shall explore in this paper.

Literature Review

Research on Digital Goods and Software Products

IT IS WIDELY AGREED BY MANY RESEARCHERS that the onetime development cost of digital product dominates its marginal cost. The reason is fairly simple. It takes literally no effort to reproduce a digitized product. Almost all expenses associated with making a digitized good are incurred during the development stages, and they are paid only once. As soon as the final product is ready, it is nearly costless to produce additional copies.

This diminishing marginal production cost, coupled with the low distribution cost brought about by the ubiquity of the Internet, necessitates new competitive models to characterize the market behaviors of digital products. In particular, traditional cost-based pricing (like Bertrand price competition) may not be readily applicable since the marginal cost of digital goods is close to zero. To charge based on marginal cost would imply almost free products, which is not what we typically observe in today's digital product markets.

This low marginal cost property has brought about radically new implications and practices for the software industry. Because of the negligible replication cost, it is now very common for software vendors to promote free tryout, which is a very effective mechanism in signaling software quality. As first demonstrated by Ackerlof [1], quality uncertainty and asymmetric information between sellers and buyers can lead to severe adverse selection problems. This could sometimes lead to a complete market breakdown, which occurs when good products are completely forced out of the market. Free product tryout, on the other hand, allows consumers to gain hands-on experience with the actual software. Consumers who can easily download the evaluation copies from the Internet can then thoroughly test the various facets (like functionality, quality, or interface) of the software.

Other than cost considerations, past research on digital products and software decision-making has primarily aimed at identifying optimal marketing conditions or strategies. For instance, in view of the inappropriateness of cost-based pricing, many researchers suggest that product differentiation, price discrimination, and quality discrimination were possible means to achieve higher returns for digital product sellers [14, 40, 42]. By examining parsimonious economic models, they show that these strategies can effectively serve more customers and bring in additional benefits to both sellers and buyers.

Setting aside pricing issues, Bakos and Brynjolfsson [3] consider the interesting phenomenon of bundling. They show that bundling a large number of digital products, even if some of them are totally irrelevant to particular customers, can sometimes be profitable to sellers. This is an intriguing and yet surprising result since it is commonly held that the Internet will facilitate micropayments, which will probably lead to "smaller" and more customized products (for example [12, 37]). Along a similar vein, Bakos et al. [4] present another interesting result when they show that product sharing among customers can indeed be beneficial to sellers. This is again

counter-intuitive since, in general, sharing implies lower sales volume and hence lower revenue.

Another stream of research focuses specifically on computer software. Choudhary et al. [15] set up an economic model to study the feasibility of renting software. Since renting encompasses different revenue schemes and legal considerations compared to traditional selling or licensing approaches, it opens up new opportunities for software vendors to capture higher market shares and profits. Padmanabhan et al. [36] study new software development and upgrade decisions when demand externality information is asymmetric. Their results show that, when consumers have limited information on the true externality, an initial less-than-full quality product can serve as a credible signal of a high network externality. Ellison and Fudenberg [18] show that, even with positive network externalities, monopoly software vendors might still want to introduce upgrades that are backward- but not forward-compatible. Finally, from the point of view of a monopoly vendor, Raghunathan [38] shows that under high cannibalization, introducing multiple editions of the same software simultaneously is optimal in many general conditions, whereas a single edition with full features is preferable when cannibalization is low.

All these studies significantly contribute to understanding the competitive decisions on digital products, and they provide valuable guidance toward the marketing and pricing of computer software. However, on the consumer side, the adoption of computer software often involves complicated review procedures and selection criteria, which may largely determine the final choice outcomes. Therefore, when studying software development decisions, it is useful to consider the evaluation criteria of consumers. We next provide a brief overview of the software selection literature.

Software Selection and Evaluation

In the past, many studies have suggested different frameworks, strategies, and processes in selecting various types of software technologies or products. Broadly defined, software evaluation exercises can be classified into two groups: product-oriented and process-oriented [9]. Product-oriented evaluations focus on choosing software products that can fulfill the functional requirements of the users, whereas process-oriented evaluations aim at finding software that can improve existing organization processes and practices.

The process approach, in general, tends to be more extensive and situation-specific, and its outcome depends largely on existing organization customs and practices. For individual users or small businesses, software choices are based more on descriptive variables, such as those related to the software, vendors, or peer opinions [13]. Therefore, product-oriented evaluation methodologies might be more readily applied for software decisions faced by individual or small business users, who could only devote limited resources toward evaluating the alternatives. Besides, for more generic tasks such as word processing or e-mail management, the application processes are normally well defined and the selection among alternative software offerings often boil down to evaluating product (or vendor) related attributes.

Many software evaluation frameworks that place heavy emphasis on product features have been developed for various application domains. For instance, the technology delta framework, proposed by Brown and Wallnau [9], focuses on understanding the differences in features between competitive software technologies. Collier et al. [16] suggest an evaluation framework that focuses on performance, functionality, usability, and ancillary task support as four major categories of criteria that dictate the selection of data mining software. With a particular focus on commercial-off-the-shelf (COTS) software packages, several studies identify, among many others, functionality, quality, and interoperability as important selection attributes [28, 35, 43]. Finally, to illustrate the IusWare software selection methodology, Morisio and Tsoukias [33] apply functionality, usability, portability, and maturity in their case study as the quality attributes in evaluating competitive CASE products.

A common thread in these evaluation frameworks is that functionality was almost unanimously identified as an important attribute toward the final selections of software. As indicated by Goodwin [22, p. 231], "in a sense, functionality itself can determine usability; if the functions provided do not match task requirements, a system will not be usable." Hence, functionality is of paramount importance in software choices, and possessing suitable levels of functionality represents a necessary condition for the success of software packages.[1] Empirically, by studying the impact of software preannouncements, Hoxmeier [25] also shows that fulfilling commitments to software functionality is the most important element that affects vendor reputations.

Because of the dominant role of functionality in software choice decisions, we consider functionality as a key attribute that determines software sales. This is a more delicate approach compared to past studies that lumped all relevant attributes of software into overall measures like "quality" or "value" (for example [18, 36]). Although these measures are theoretically sound and can substantially aid in deriving tractable solutions, they do not readily lend themselves to providing answers to practical, development-related questions such as how many functions should software vendors build into their products, or what should be the final complexity of the software. By explicitly focusing on functionality, we could derive useful implications and strategies that could be more directly applied by practitioners in the software industry.[2] In his study of software editions and upgrades, Raghunathan [38] also considers number of features as the key characteristic that differentiates the high- and low-end editions of the software, and he uses it as the primary variable that shapes the value (consumer utility) of the software.[3]

Note that the aforementioned studies almost exclusively focus on the post-development stages of the software. They assume the software was already in place, and do not consider the phases prior to design and programming. Since computer software is largely made up by functional routines, it is a major decision of software developers to determine the number of routines to be programmed and incorporated into the final commercial product. This is an important step that precedes any marketing or pricing decisions. Overly complex software that includes too many advanced functions may appeal to professional users, but general consumers who demand simple software may opt for a low-value version that costs them less money. Therefore, the functionality of

the software may determine, to a large extent, which consumer segment it is going to serve and the final market size and performance of its vendor.

At first glance, our reliance on functionality to characterize computer software may appear to closely correspond to the function point (FP) method of software size measurement. However, recent research has shown that, provided the features of the software are well isolated, the FP method can also be extended to new software engineering approaches like the objected-oriented or component-based frameworks. A good example is the object-oriented function point system, which maps logical files in traditional FP frameworks to object classes and transactions to object methods [10, 34]. Hence, our research model on functionality is general and its application to modern software development practices is straightforward.

We apply game theory to examine the noncooperative outcomes of software vendors who are selling the same category of computer software, with functionality and price as key decision variables. Game theory has been widely applied to study multiagent decision problems in many different contexts (see, for example [21]). It is an important tool for studying strategies involving multiple players, each attempting to maximize their own payoff. The technique is appropriate in the present context because computer software vendors often need to compete against each other to achieve higher market shares and customer bases (such as, Microsoft has been competing against Lotus in the microcomputer spreadsheet market; against Netscape in the Internet browser market; and against Oracle in the database market). Sarvary and Parker [39] also apply game theory to study equilibrium conditions of information products, although in their study only pure informational goods like reports or articles are considered. Since computer software differs substantially from pure informational goods,[4] we propose a new competitive model that centers on the functionality of the software. The detailed specification of the model is outlined in the next section.

Model Specifications

WE CONSIDER A COMPETITIVE MODEL where the functionality of each software ranges from Q_L to Q_H. Q_L denotes the minimum functionality that consumers demand if they are to consider buying the software. For example, in the case of word processors, the minimum set of functions that consumers demand may include the ability to insert and delete text, save and retrieve files, and print typed documents. If the software lacks any one of these functions, it may not be attractive to anyone. Q_H, on the other hand, denotes the upper bound on functionality. A developer simply cannot design and incorporate an unlimited number of functions into a single software product. Hence, we use Q_H to represent the highest possible number of functions that can appear in a software package. Without loss of generality, we further set $Q_L = 0$ and $Q_H = 1$.

Consumer Preference

Different consumers have different tastes or preferences toward the functionality of the software. We use a parameter θ to represent the type of the consumer, where type

here denotes the minimum set of functions required by the particular consumer. If the functionality of the software is higher than θ_i, then consumer i is a potential customer of the software and vice versa. For simplicity and expositional purpose, we further assume θ is drawn from a uniform distribution between Q_L and Q_H (that is, 0 and 1). That is,

$$f(\theta) = 1 \quad \text{and} \quad F(\theta) = \theta,$$

where f denotes the probability density function and F denotes the cumulative density function up to θ. Later in this paper we shall modify this distribution assumption and examine the changes on some of the model results and implications.

If the software vendor charges p for its software (with functionality Q), then the net surplus of a particular consumer is

$$U = \begin{cases} \theta - p, & \theta \le Q \\ 0, & \theta > Q. \end{cases}$$

We assume that consumers are "perfectly" price sensitive. That is, if there is more than one choice of software, consumers will always opt for the alternative that charges the lower price, provided its functionality is higher than θ (that is, provided it can match the task requirements of consumers).

Note that even though we focus exclusively on functionality, this setup of consumer preference is sufficiently general, which can allow for the inclusion of other software selection variables. For instance, if we need to consider fixed, vendor-specific factors such as reputation or complementary product lines, we may impose an additive component δ to the overall functionality that shifts the attractiveness of the software. For function-related concerns such as the programming skill or quality of the vendor, or market specific factors such as the weights placed by consumers on each software function, we may utilize a multiplicative discount factor λ that scales the corresponding functionality. In other words, by choosing suitable scale and discount parameters λ and δ, the expression $g(Q) = \lambda Q + \delta$ can be used in place of Q (as in the above surplus function) to address additional software evaluation criteria.[5] Nevertheless, despite this extended generality, since the objective of this study is to analyze optimal functionality choices, we shall continue to assume $\lambda = 1$ and $\delta = 0$ in the remaining parts of the paper. That is, none of the vendors is enjoying ex ante advantage (or suffering from disadvantage) compared to others.[6]

Demand

The demand for the software (with functionality level Q) consists of those consumers whose net surplus is higher than zero. That is,

$$d(Q, p) = \Pr(U > 0 \text{ and } \theta \le Q) = F(Q) - F(p) = Q - p.$$

To examine the relationship between overall market sizes and the vendors' strategic considerations, we denote M to be the total market potential served by the software, which is a domain-specific parameter. That is, there are possibly different Ms for different categories of software. For instance, it is likely that the number of potential users for word processing software is larger than the one for database software. In that case, M for the former category is substantially larger (in terms of magnitude) than the latter one. Also note that M merely states the maximum. The actual number of consumers who are willing to purchase a firm's software depends on its functionality, price, and also the functionality of competitive software packages that exist in the same market.

The overall demand across the entire market with size M is therefore

$$D(Q, p) = M(Q - p).$$

With more than one choice in the same software market, we need to separately examine different segments within the Q-space. Consider the case of two software products with functionality Q_1 and Q_2. Without loss of generality, assume $Q_1 < Q_2$. We need to consider three separate regions as shown in Figure 1.

Region 1: 0 to Q_1

Since $Q_1 < Q_2$, p_1 must be lower than p_2 or otherwise all consumers will switch to software 2 (since software 2 is superior in terms of functionality). Therefore, the constraint $p_1 < p_2$ must always be followed for software 1 to survive in the market. Since consumers are price sensitive and p_1 is lower, all consumers with θ lying in this region are potential customers of firm 1. Therefore, the resulting demand for software 1 is

$$D_1 = M(Q_1 - p_1).$$

Region 2: Q_1 to Q_2

In this region, Q_1 cannot satisfy the task requirements of consumers. All consumers within this segment are therefore potential customers of firm 2. The net surplus of consumers lying within this region is

$$U = \theta - p_2,$$

where

$$Q_1 < \theta \leq Q_2.$$

Since all consumers with type θ less than Q_1 prefer software 1, the corresponding market demand for software 2 is

$$D_2 = M \times \Pr(U > 0 \text{ and } Q_1 < \theta \leq Q_2) = M(Q_2 - \max(Q_1, p_2)).$$

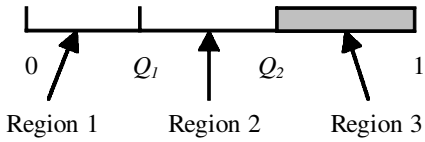

Figure 1. Three Separate Regions in the Q-Space

Region 3: Q_2 to 1

Here consumer requirements on functionality are far too high, and both software 1 and 2 do not possess enough functionality to appeal to these consumers. Therefore, all consumers with type $\theta \in (Q_2,1]$ will prefer to stay out of the market.

If $Q_1 = Q_2$, p_1 must be equal to p_2 (otherwise they are perfect substitutes with unequal prices, and obviously the lower priced software will capture all the demand, which brings us back to the monopoly situation). In this case, the two vendors share the demand, which is

$$D_1 = D_2 = M(Q - p)/2,$$

where

$$Q_1 = Q_2 = Q \quad \text{and} \quad p_1 = p_2 = p.$$

Development Cost

We assume a direct relationship between the development cost and functionality level of the software, and the development cost takes the form VQ^{α}.[7] V is a positive constant that denotes the onetime, per-function development cost that software vendors must pay. The exponent α is a parameter that models the scale economies of the software development process. An α less (bigger) than one indicates diseconomies (economies) of scale, whereas an α equal to one represents constant returns to scale.[8]

It is straightforward to assume that computer software with more functions requires higher development costs, since more designing, programming, and testing efforts are needed to bring in the extra functions. This notion of incremental cost based on functionality is well demonstrated in past theoretical and empirical software cost estimation studies (for example [7, 8, 19, 26, 30]).[9]

On the other hand, research has shown mixed evidence toward the scale economies of the software development process. Early software cost studies applied linear regressions to estimate the growth of cost (for example [2, 27]), which assumes constant returns to scale irrespective of the size or functionality of the software. Latter research, however, has suggested that the log-linear model (which corresponds to our cost formulation) is often more appropriate and it permits direct assessment of the scale economies parameter [5, 30]. Depending on the data sets used for estimation, empirical results on scale economies tend to be inconclusive, where both increasing and decreasing returns to scale were shown to exist for different sets of software

projects.[10] Banker and Kemerer [5] suggest that software size may moderate scale economies. Small software projects tend to show increasing returns to scale, whereas larger projects are likely to exhibit the opposite trend. Finally, by analyzing the minimum software cost model, Hu et al. [26] find that the effect of size on cost diminishes as the size of the software grows (that is, economies of scale).

Our cost formulation can be adapted to model different scale economies by manipulating the constant α. Given that previous research suggests scale economies parameters to be narrowly ranged around one, we begin our analysis by setting $\alpha = 1$ (that is, the development process exhibits constant returns to scale). We shall relax this restriction and examine the model results in light of scale economies in a subsequent section.

Once the software is ready, we consider a scenario where consumers can observe the functionality of the software. This is possibly achieved by detailed product descriptions, performance statistics and comparisons, or product tryouts. Furthermore, we assume software developers are risk neutral, and they know their own (as well as their competitor's) development costs. That is, V and α are common knowledge among software developers.[11]

Having set up the basic model features, we can start to analyze the competitive decisions of software developers. We begin with a monopoly software market in the next section.

Optimal Decisions in a Monopoly Market

IN A MONOPOLY MARKET, there is only one software package available, and consumers can only choose to either buy or not to buy the product. Software developers need to make two decisions: the functionality level Q of the software and its price P. The demand is now $D = M(Q - P)$. And the profit is

$$\text{profit } \pi = \text{demand} \times \text{price} - \text{development cost} = M(Q-P)P - VQ.$$

The first-order conditions (FOCs) over Q and P yield

$$\frac{\partial \pi}{\partial Q} = MP - V, \quad \frac{\partial \pi}{\partial P} = MQ - 2MP,$$
$$\Rightarrow Q^* = \frac{2V}{M}, \quad P^* = \frac{Q^*}{2}.$$

A closer inspection on the FOCs reveals that Q^* is a local minimum. For the profit function to be nonnegative, Q must satisfy the following relationships:

$$Q = 0 \quad \text{or} \quad Q \geq L = \frac{4V}{M},$$

and π is an increasing function in Q after L. Hence, if Q is too low, the demand is too small to compensate the onetime development cost of the software. This result is

intuitive and its implication is straightforward. Occasionally, some consumers may prefer software with very limited functions, but producing specific software for those consumers is not justifiable from the demand and profit point of view.

As long as Q is higher than L, π is always positive and increasing. Since Q_H is the upper bound on Q, optimality requires $Q^* = Q_H = 1$. Hence, a monopoly software developer will choose $Q^* = 1$ and $P^* = 0.5$, which give the resulting profit

$$\pi_{(\text{monopoly})} = \frac{M}{4} - V.$$

To conclude, a monopoly software developer will choose to implement every possible function into its software package. This result is interesting and it partly explains why, empirically, some consumers indicated that they do not use a majority number of functions in certain software packages [11, 31]. Since the optimal price P^* is set according to functional level (which is chosen to be the highest possible value), some consumers (those who demand low functionality software) are unfortunately priced out of the market. The final market demand is $D = M/2$, which means that only one-half of the market is served by the software.

Note that for the monopoly developer to make a positive profit, $\pi_{(\text{monopoly})}$ must be nonnegative, which translates into $V \leq M/4$. That is, either V (the per-function, one-time development cost) must not be too high or M (the market potential) must not be too small. If the market lacks any one of these, it is not profitable for any software developer.

Optimal Decisions in a Duopoly Market

IN THIS SECTION, WE CONSIDER the situation when two firms are developing and selling the same category of computer software. For instance, Microsoft and Lotus are the two dominant players in the spreadsheet market, whereas Microsoft and Netscape have almost taken up the whole Internet browser market.

Following the demand schedules in the "Demand" subsection, the demand and the corresponding profit functions for the firms are:

$$D_1 = \begin{cases} M(Q_1 - P_1) \\ 1/2\, M(Q_1 - P_1) \\ M(Q_1 - \max(P_1, Q_2)) \end{cases} \qquad \pi_1 \begin{cases} M(Q_1 - P_1)P_1 - VQ_1 & \text{if } Q_1 < Q_2 \\ 1/2\, M(Q_1 - P_1)P_1 - VQ_1 & \text{if } Q_1 = Q_2 \\ M(Q_1 - \max(P_1, Q_2))P_1 - VQ_1 & \text{if } Q_1 > Q_2 \end{cases}.$$

Equilibrium Considerations

We first consider a simplified version of this game, where we restrict the strategy set of each developer to be $S_i = \{C, D\}$, $i = 1$ or 2, and $0 < C < D \leq 1$. Also, assume C and

D are both on the high end of the Q-space, and the gap between them is very small. Since $P_i = Q_i/2$ is the optimal monopoly pricing strategy (and indeed also when Q_i is lower than Q_j, $i \neq j$), we tentatively assume the two developers price their software according to this expression. The payoff matrix of this restricted setting is shown in Figure 2.

Provided V is not close to $M/8$ (which is a necessary condition for any positive profit in the duopoly market), it is easy to verify that if $Q_1 \neq Q_2$, the "low Q" firm (strategy C) would enjoy a higher profit than the "high Q" firm (strategy D). Even if the "high Q" player anticipates this outcome and sets a higher price in an attempt to capture the remaining surplus (the payoffs shown in parentheses in Figure 2),[12] both players still have incentives to select $Q = C$. Therefore, no matter what the opponent chooses, both players would always prefer C instead of D. In other words, the unique Nash equilibrium (NE) of the game is $Q_1 = Q_2 = C$, with payoffs $MC^2/8 - VC$ to both players.

Nevertheless, the two players can do better than this NE. It can be shown that the payoffs follow the relationship $MC^2/8 - VC < MD^2/8 - VD$. That is, both firms can do better by simultaneously choosing $Q = D$ (building higher functionality software). This resembles the well-known prisoners' dilemma, where both players have incentives to deviate from cooperative moves, which, however, lead to suboptimal payoffs. From the consumers' point of view, this outcome is also undesirable since some consumers (those demanding software with Q between C and D) are not served by the market.

Indeed, if we extend the strategy space and allow any $Q \in [0, 1]$, provided the opponent is developing software with sufficiently high Q, there is always an incentive for one firm to construct software with fewer functions and capture substantial demand (and hence profit). This is similar to a Bertrand competition, where players have an incentive to bid down the final price. However, there is one subtle difference here: the two firms would not bid Q all the way down to zero. If one firm is building software with very few functions, the other firm may decide to switch and develop the most powerful software ($Q = 1$). This is because, in this situation, the "gap" in market demand is big enough so that the firm producing the high functionality software could still attract enough consumers to generate a positive profit. The indifference point between undercutting Q and going for $Q = 1$ is given by[13]

$$\frac{MQ^2}{4} - VQ = M(1-Q)\frac{1}{2} - V$$

$$\Rightarrow Q' = \frac{2}{M}\left[\left(V - \frac{M}{2}\right) + \sqrt{\left(V - \frac{M}{2}\right)\left(V - \frac{3M}{2}\right)}\right],$$

and the payoff corresponding to Q' is

$$\pi' = -2\left(V - \frac{M}{2}\right) - \sqrt{\left(V - \frac{M}{2}\right)\left(V - \frac{3M}{2}\right)}.$$

Q_1 \ Q_2	C	D
C	$\frac{MC^2}{8}-VC,$ $\frac{MC^2}{8}-VC$	$\frac{MC^2}{4}-VC,$ $\frac{M(D-C)D}{2}-VD$ $[M(D-C)C-VD]^{**}$
D	$\frac{M(D-C)D}{2}-VD$ $[M(D-C)C-VD]^{**},$ $\frac{MC^2}{4}-VC$	$\frac{MD^2}{8}-VD,$ $\frac{MD^2}{8}-VD$

Figure 2. Payoff Matrix When Players Can Choose Either C or D, with $C < D$. Upper entries refer to 1's payoff, whereas lower entries refer to 2's. ** Payoff when the "high Q" firm raises its price to C.

Denote $\pi_{(equal)}$ as the maximum payoff to both firms when they develop equal software. Appendix A shows that $\pi' > \pi_{(equal)}$. Therefore, the two developers could benefit each other if one of them builds software at Q' and the other builds it at one. This is equivalent to dividing up the market with each of them serving a different consumer segment.

Note that

$$\lim_{V/M \to 0} Q' = \sqrt{3} - 1 \approx 0.732.$$

That is, as $V \to 0$ or $M \to \infty$,[14] Q' can readily be calculated, which is well above 0.5 but lower than 1. In fact, Q' is always larger than 0.5. In a duopoly market, the firm building the mediocre software always appeals to a larger consumer group than the one that builds the most powerful software.

So far, we have been assuming the developers follow the monopoly pricing strategy. Such a strategy is optimal when they select the same functional level. In cases when the developers choose different Qs, the strategy remains the best for the "low Q" player. But for the "high Q" player, provided his opponent's Q is not too low, he could actually do better by raising his price. If the optimal P_2 is lower than Q_1, player 2 (the "high Q" player) could gain extra profits by raising his price to Q_1. Therefore, if the restriction on pricing behavior is relaxed, the new indifference point between producing the highest functionality software (and raising the price according to the opponent's Q) and undercutting Q is given by[15]

$$\frac{MQ^2}{4} - VQ = M(1-Q)Q - V$$

$$\Rightarrow Q'' = \frac{2}{5M}\left[(V+M) + \sqrt{(V+M)^2 - 5VM}\right],$$

and the payoff corresponding to Q'' is:

$$\pi'' = \frac{2M - 8V}{25M}\left[(V + M) + \sqrt{(V + M)^2 - 5VM}\right] - \frac{V}{5}.$$

Although the result is intuitive, in Appendix B we formally show that both $Q'' > Q'$ and $\pi'' > \pi'$. If the developer who builds the most powerful software can manipulate and raise his price, both players can adjust their strategies and achieve a better outcome.

It is easy to verify that

$$\lim_{V/M \to 0} Q'' = 0.8.$$

When $V \to 0$ or $M \to \infty$, the indifference point lies around 0.8, and the segment size served by the lower functionality software is approximately four times the size served by the full functionality alternative. In any case, as $Q'' > Q'$, Q'' also lies above 0.5. That is, the mediocre software has a larger segment of potential consumers. This appears to be congruent to empirical observations. For smaller scale, COTS software markets, the majority of customers of high-power software come from business sectors, whereas, in most cases, individual consumers prefer cheaper software with adequate functionality for home or leisure usage.

The profit π'' enjoyed by both developers is better than either π' or $\pi_{(equal)}$, but it is still considerably less than the monopoly payoff. For consumers, those who lie within $(Q'', 1]$ are served by the most powerful software, whereas only one-half of the consumers in $[0, Q'']$ are served by the mediocre software (the one with $Q = Q''$).

π'' is by far the best payoff for the developers in the duopoly market, which could be achieved when one of them chooses $Q = Q''$ and $P = Q''/2$ and the other chooses $Q = 1$ and $P = Q''$. However, this outcome is not sustainable, since the developer who chooses Q'' has an incentive to deviate from this strategy and switch to $Q = 1 - e$ (e being a small number). The best reactions of the two players are:

If $Q_2 > Q''$, player 1 should choose $Q_1 = Q_2 - e$ and $P_1 = Q_1/2$.
If $0.5 < Q_2 \leq Q''$, player 1 should respond with $Q_1 = 1$ and $P_1 = Q_2$.
If $Q_2 \leq 0.5$, player 1 should set $Q_1 = 1$ and $P_1 = 0.5$.

As a simple example, suppose one developer holds an a priori belief that the other software is going to be very powerful. In order to avoid serving only consumers in the very high end of the market and selling only a few copies of his software, he might decide to produce a slightly less powerful alternative and charge a lower price in order to sell to the lower-end consumers. This could possibly result in much higher sales and hence a better profit. On the other hand, if the developer anticipates his opponent will sell low functionality software, he might want to push himself to the limit and build every possible function into his software. This is because now there are enough high-end consumers available to justify the need of a highly capable software package. Finally, if the developer expects the other software to possess very few functions, he would follow the monopoly strategy stated in the previous section. That

is, he would completely ignore his competitor's software (and its consumers, who are not willing to pay for his product anyway) and focus his attention on the "upper half" of the market.

Irrespective of the final choice made by the two developers, there is always an incentive for one of the players to deviate, and no strategic pairs by the two developers are stable. In other words, there is no stable pure strategy NE in this game setting. This result is easily depicted by the best response correspondence curves shown in Figure 3.

Clearly, the two best response correspondences do not intersect each other, which indicates the absence of pure strategy NE wherein no firm is willing to deviate.

Mixed Strategy Nash Equilibrium?

Having shown that there is no pure strategy NE, we now turn to consider the possibility of a mixed strategy NE. Before we proceed, it would be useful to consider the following propositions.

> **Proposition 1:** There cannot be any mixed strategy NE that forces either player to achieve a constant payoff if we allow that player to vary his software price.

The reason is fairly obvious. Since the payoff of each player depends on his own software price (but not the opponent's), he can freely vary his price level and achieve different profits. This is largely unaffected by his opponent's decisions on functionality or price. Therefore, if one has free choice governing his own price level, by all means his opponent cannot force him to stay at a constant payoff, and hence there cannot be any mixed strategy NE that allows players to vary P freely. That is, price must be preset exogenously by the software developers for any equilibrium to exist.

In view of this proposition, we modify the situation slightly and consider the case when the developers mix only over the variable Q, but not P. In other words, we assume they choose the fixed pricing strategy $P = aQ$, where a is a preset, positive constant.[16] The next proposition proves to be very useful toward searching for mixed strategy NE.

> **Proposition 2:** If a mixed strategy NE exists, it must involve the two players mixing uniformly over $Q \in [0, 1]$. That is, they must mix perfectly according to a constant probability density function $f(Q) = 1$.

Proposition 2 can be established by contradictions. Suppose a mixed strategy NE exists involving nonuniform probability distributions. Without loss of generality, assume player 1 assigns a disproportionally large probability density F_A over the range Q'' and 1, and the remaining density F_B over 0 and Q'' in his equilibrium strategy. Denote this strategy S_E. Since this game is symmetric, if S_E constitutes a NE, player 2 should also follow the same strategy. That is, the player should also assign F_A to the range $[Q'', 1]$ and F_B to the range $[0, Q'')$. However, if player 2 assigns F_A as a fixed mass point on Q'' and F_B as a fixed mass point on 1 (and charges $P = Q''$ when he

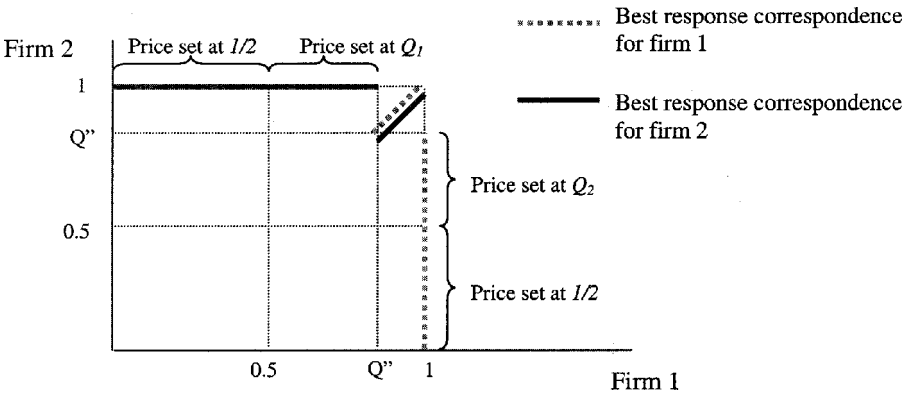

Figure 3. Best Response Correspondences of the Two Software Developers

chooses $Q = 1$), his payoff will improve over S_E. Therefore, S_E is not a best reply to itself and hence cannot be an equilibrium strategy.

Proposition 2 is very useful in that it eliminates almost all candidates for equilibrium strategy. The only candidate that remains is the one mixing uniformly over [0, 1]. Is it an NE? Unfortunately, our results in Appendix C show that it is also not an equilibrium strategy. Hence, there is no mixed strategy NE in this setting.

Up to this point, we have shown that there is no NE in the one-stage, simultaneous game setting. This is quite a discouraging result to computer software developers because it implies that if two developers enter the market simultaneously, there is no promising guiding principle that can secure them positive payoffs. Even if they are willing to vary the functionality levels of their software according to fixed probability distributions, there is still no guarantee that they can extract positive profits on expectations. Despite the fact that Q'' is a good separating point for the two developers to divide the market and share good profits, it is almost self-enforcing that one of them (the one choosing $Q = Q''$) would adjust his level of Q and invade the segment of his opponent. Therefore, setting appropriate functionality is not an easy decision for software developers if they decide to enter the duopoly market.

Nonetheless, if we allow either software developer (together with consumers) to be patient and to wait until his opponent's software appears on the market before he makes his functionality decision, there can be a stable NE that involves pure strategies from both developers. This is the topic of the next section.

A Two-Stage Variation of the Model

WE NOW CONSIDER A SLIGHTLY DIFFERENT MODEL in which the two developers will set their functionality decisions in sequence. In stage 1, one of the two developers would make his functionality and price decisions. The other developer observes this move and then makes his own decision regarding how many functions should be

included in stage 2. Payoffs are then determined by their choices of functionality and prices.

This setting corresponds to cases when the idea behind the software is rather innovative and some of the consumers are skeptical about adopting the software (that is, they are patient and prefer to wait until the arrival of more alternatives before making purchases). A majority of leisure software such as computer games, or new application domains such as Customer Relationship Management (CRM) software or e-commerce applications, fall within this framework. By analyzing this setting, we shall offer useful strategic implications to software developers who are uncertain about locating the functionality of their new, innovative software.

To study the two developers' behaviors in the two-stage model, we first analyze the decision at stage 2. Without loss of generality, we assume that firm 1 is the first moving firm. If firm 1 chooses to develop software at $Q_1 > Q''$ in stage 1, firm 2 would react by setting $Q_2 = Q_1 - \varepsilon$ in stage 2, which would result in a very bad payoff to firm 1. If firm 1 opts for $Q_1 < Q''$ in stage 1, then firm 2's best response in stage 2 would be $Q_2 = 1$ (coupled with $P_2 = Q''$ or 0.5, depending on 1's choice). Hence, firm 1 is better off by developing software as close to Q'' as possible. If firm 1 produces software exactly at Q'', then firm 2 would be indifferent between choosing $Q_2 = 1$ or $Q_2 = Q_1 - \varepsilon$. Since it is Pareto optimal for firm 2 to go for $Q_2 = 1$ in this situation, we assume firm 2 would go for $Q_2 = 1$ when $Q_1 = Q''$.

Having resolved the reactions of firm 2 in the second stage, we now turn back to stage 1. Given firm 2's likely reactions, it is apparent that firm 1 should develop his software at $Q_1 = Q''$ in stage 1, since this would yield the highest possible payoff for him. The reaction of firm 2 in the second stage poses a big threat to firm 1. On one hand, Q_1 cannot be set too low or else firm 1 could only get a small demand and hence a lower profit. On the other hand, if Q_1 is set too high, firm 2 may undercut Q_1 by a slight amount, which would lead to a very bad payoff to firm 1. Therefore, the only viable strategy for firm 1 is to strike the balance and go for $Q_1 = Q''$ in stage 1. We shall term the strategy $Q_1 = Q''$ a "secure payoff" strategy for firm 1. Hence, the unique NE consists of firm 1 choosing his secure payoff strategy $Q_1 = Q''$ (and $P_1 = Q''/2$) in stage 1, followed by firm 2 choosing $Q_2 = 1$ (and $P_2 = Q''$) in stage 2. The profit would be π'' for both firms. This is the best and yet sustainable outcome for both players among all the strategies that we have considered so far in this paper.

In fact, this NE belongs to a more general class of equilibrium concepts called reactive equilibrium [24]. The essential spirit of a reactive equilibrium is that if one of the players (call him player *A*) prefers a deviation from his equilibrium strategy, the other player can "punish" *A* by yet another strategy (which is also out of equilibrium) that will lead to a worse outcome for *A* but a better outcome for himself. Furthermore, no further deviation is possible for *A* so that he is strictly worse off if he chooses to deviate at the very beginning. It can be easily checked that the NE that we characterized in this section fulfills the criteria of a reactive equilibrium. If player 1 deviates from Q'' and chooses $Q_1 > Q''$ in stage 1, player 2 can punish him by choosing $Q_1 - \varepsilon$ in stage 2. This would lead to a poor outcome for player 1 but a better outcome for player 2 compared with the original equilibrium strategy. More importantly, player 1

cannot move further (that is, he cannot react by further choosing $Q_2 - \varepsilon$). Therefore, the likely reaction of player 2 forces player 1 to stay at his equilibrium strategy, and the two-stage NE is a reactive equilibrium.[17]

Model Extensions

IN THIS SECTION, WE CONSIDER A FEW EXTENSIONS to our software functionality model. We begin by relaxing the assumption on the scale economies parameter α.

Scale Economies in the Software Development Process

In the monopoly market, If $\alpha \neq 1$, then the FOCs of π over Q and P become

$$\frac{\partial \pi}{\partial Q} = MP - \alpha V^{\alpha-1}, \quad \frac{\partial \pi}{\partial P} = MQ - 2MP,$$

$$\Rightarrow Q^* = \left(\frac{2\alpha V}{M}\right)^{\frac{1}{2-\alpha}}, \quad P^* = \frac{Q^*}{2}.$$

Similar to the case in constant returns to scale, Q^* is a local minimum, and π increases with Q provided $Q \geq (4V/M)^{1/(2-\alpha)}$. The monopoly software developer would continue to produce full functionality software in view of scale economies.[18]

In a duopoly market, the payoff matrix in Figure 2 can be modified slightly to add in α. Again, provided V is not close to $M/8$ and α is near one, both developers have incentives to lower the functionality of the software until a certain indifference point that leads to equal profits. The only difference here lies in the "savings" in cost. In qualitative terms, if $\alpha < 1$ (diseconomies of scale), there is an additional incentive (other than capturing higher demand) to lower functionality since the potential cost saving is now higher than when there is constant returns to scale. If $\alpha > 1$ (economies of scale), the cost saving is comparatively smaller.

If the firms follow the monopoly pricing strategy $P = Q/2$, the condition that characterizes the indifference point in Q is

$$\frac{MQ^2}{4} - VQ^{\alpha} = M(1-Q)\frac{1}{2} - V$$

$$\Rightarrow \frac{MQ^2}{4} + \frac{MQ}{2} - VQ^{\alpha} - \left(\frac{M}{2} - V\right) = 0.$$

In this case, a closed-form solution in Q cannot be solved, but we can always perform a numerical search in the Q-space to locate the new Q'. In the presence of economies of scale ($\alpha > 1$), the new Q' will be bigger than the Q' in the constant returns to scale case, whereas if $\alpha < 1$, the new Q' will be smaller.

If the "high Q" developer raises his price, then the new indifference point is characterized by

$$\frac{MQ^2}{4} - VQ^{\alpha} = M(1-Q)Q - V$$
$$\Rightarrow \frac{5MQ^2}{4} - MQ - VQ^{\alpha} + V = 0.$$

Again, a numerical search is needed to locate the new Q''. If $\alpha > 1$, the new Q'' will be bigger than the old one in the constant returns to scale case, whereas it will be smaller if $\alpha < 1$.

The general results remain intact. In the one-stage simultaneous setting, there will be no NE in either pure or mixed strategy. If the developers make subsequent moves, the reactive equilibrium in the two-stage model continues to apply and the first mover would select the new Q'', whereas the second mover would choose $Q = 1$. Depending on the value of α, payoff π needs to be slightly modified. If $\alpha > 1$, the new equilibrium profit will be higher for both players. If $\alpha < 1$, then both players suffer because of higher development costs.

Market Segmentation

We assumed a uniform distribution of consumer tastes in the previous discussions. This is appropriate when the consumers have uniform tastes toward functionality. Nevertheless, depending on the market under study, research has shown that consumer preferences may be clustered into different segments, and techniques such as cluster analysis, conjoint segmentation, and mixture regression models have been devised to identify these segments [44]. We consider the results and implications of our model in the presence of segmentation in this section.

Consider a segmented software market with two groups of consumers who differ in their desire toward functionality. Consumer types are denoted by θ_l and θ_h, where $\theta_l = r\theta_h$, $0 < r < 1$. That is, the low requirement group has less desire toward the functionality of the software, and they are in general satisfied with less powerful software.[19] Denote the cumulative density function (cdf) of θ_h as F, and M_l and M_h as the sizes of the two segments.

For a particular software with functionality level Q, it attracts a low- (high-) type consumer if $Q \geq \theta_l$ (θ_h). Surplus is again defined as ($i = l, h$):

$$U_i = \begin{cases} \theta_i - p, & \theta_i \leq Q \\ 0, & \theta_i > Q \end{cases}.$$

Demand for software with functionality Q is $D_i(Q, P) = M_i \times \Pr(U_i > 0 \text{ and } \theta_i \leq Q)$. That is,

$$D_l = M_l\left[F\left(\frac{Q}{r}\right) - F\left(\frac{P}{r}\right)\right],$$
$$D_h = M_h\left[F(Q) - F(P)\right].$$

Because $r < 1$, the first part of the demand function D_l is discontinuous, and the two pieces before and after $Q = r$ need to be considered separately. Figure 4 plots the demand functions D_l and D_h when $P = 0$ and θ_h follows a uniform distribution.

The profit function of a monopoly developer who constructs software with functionality Q is

$$\text{if } Q \le r,\ \pi_x = M_l\left[F\left(\frac{Q}{r}\right) - F\left(\frac{P}{r}\right)\right]P + M_h\left[F(Q) - F(P)\right]P - VQ^{\alpha}.$$
$$\text{if } Q > r, \pi_y = M_l\left[1 - F\left(\frac{P}{r}\right)\right]P + M_h\left[F(Q) - F(P)\right]P - VQ^{\alpha}.$$

It is clear that $\pi_y \ge \pi_x$ for all $P \le r$. This is fairly trivial, since having a higher Q implies a higher number of potential customers. We can inspect the FOCs of π_y to identify the optimal Q^* and P^*. The locations of the optimal Q^* and P^* depend on the shape of the cdf F. If M_h is not small (relative to V) and consumer taste is not heavily skewed toward the low requirement end, the monopoly developer would continue to build full functionality software, and the final demand depends only on his pricing strategy. Obviously, if $r = 1$, the two segments are identical and the model reverts back to the single-segment situation; if r is very small, the monopolist may decide to serve only the high segment, since to serve the low segment would require a drastic reduction in price, which will substantially reduce his overall profit.

To examine the implications of segmentation in the duopoly market, note that we can always add up the demands of the two segments for all $Q \in [0, 1]$, assuming $P = 0$. This overall demand can then be used to study the functionality decisions of the two developers, taking into account the reductions in demand when prices are nonzero. Although this demand function may not be continuous, we could follow the same analytical procedures in the previous sections to derive similar conclusions. That is, only a reactive equilibrium exists in the two-stage model.

Finally, at the reactive equilibrium, in cases of relatively "steady" cdf F, if r is close to one, the two developers will serve both segments, whereas if r lies somewhere near the middle of zero and one, there might be cases when optimality requires one of the developers to serve only the high segment. This is because, in equilibrium, the low-Q software may still attract consumers in the low requirement end of the high segment, whereas the high-functionality software may cost so much that it deters all low segment consumers from buying. This appears to be in line with the empirical observation that there are software packages with superb functionality (and very high prices) that are primarily targeted at organizational buyers, who are usually more resourceful and can readily afford to pay for the excessive functions.

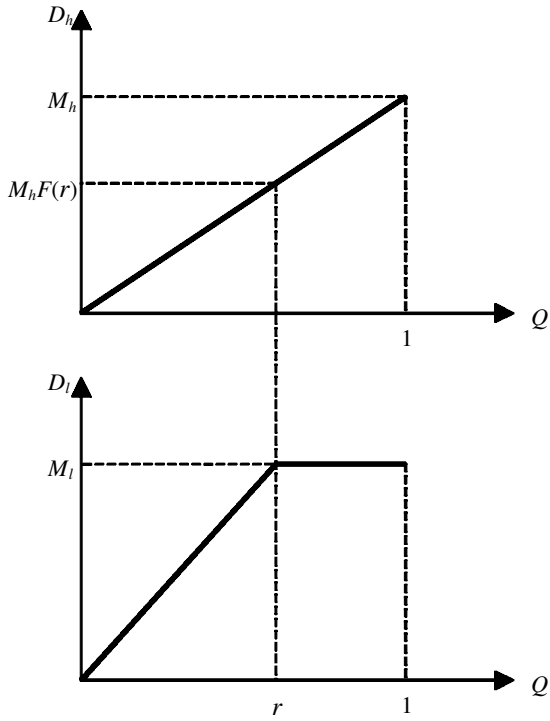

Figure 4. Demand Functions When the Market Is Segmented and $P = 0$

Conclusions and Other Remarks

TABLE 1 SUMMARIZES THE VARIOUS MODELS that we have discussed in this paper, based on uniform taste preferences and constant returns to scale in development.

The analysis of the various competitive models on functionality generates a few insights: first, a monopoly developer prefers to incorporate all possible functions into his computer software, and there is a minimum "threshold" of functionality before which the developer could not gain positive profits. Second, there is no stable NE in the one-stage, simultaneous move duopoly market. Given one firm's decision, there is always a profit incentive for the other firm to deviate. Although there is one indifference point in functionality for the two developers to divide up the market and enjoy good profits, the strategies involving that point are not sustainable (as in all other strategies). Third, a unique reactive equilibrium exists in the two-stage duopoly model. The two players can achieve the best possible outcome by serving different consumer groups who differ mainly in functional requirements. The prerequisite for such an equilibrium, however, is that some consumers and one of the developers are patient and they can wait until the first developer launches his software. Finally, although the two players enjoy the same profit in the reactive equilibrium, the two developers sell different software: the first mover sells mediocre software with limited functions, whereas the second mover sells full functionality software.

Table 1. Model Assumptions and Results

Model		Assumptions	Q^*	P^*	Profit π	Remark
Monopoly		$Q \in [0, 1]$ $V \leq M/4$	1	1/2	$M/4 - V$	
Duopoly	One stage: Restricted Q, monopoly pricing strategy $P = Q/2$	$Q \in \{C, D\}$, $0 < C < D \leq 1$ $P = Q/2$ $V \leq M/8$	C	$C/2$	$MC^2/8 - VC$ for both players	Similar to prisoners' dilemma, where both players make suboptimal choices.
	One stage: Unrestricted Q, monopoly pricing strategy $P = Q/2$	$Q \in [0, 1]$ $P = Q/2$ $V \leq M/8$	N/A	N/A	N/A	Best outcome would be one player chooses Q' ($P = Q'/2$), and the other chooses 1 ($P = 1/2$). But it is not sustainable. No pure or mixed strategy NE.

One stage:					
Unrestricted Q and P	$Q \in [0, 1]$ $V \leq M/8$	N/A	N/A	N/A	Best outcome would be one player chooses Q'' ($P = Q''/2$), and the other chooses 1 ($P = Q''$). But it is not sustainable. No pure or mixed strategy NE.
Two stages:					
Unrestricted Q and P	Player 1 moves in stage 1 Player 2 moves in stage 2 $Q_i \in [0, 1]$ $V \leq M/8$	$Q_1 = Q''$ $Q_2 = 1$	$P_1 = Q''/2$ $P_2 = Q''$	π'' for both players	Reactive equilibrium.

Note:

$$Q' = \frac{2}{M}\left[\left(V - \frac{M}{2}\right) + \sqrt{\left(V - \frac{M}{2}\right)\left(V - \frac{3M}{2}\right)}\right],\ Q'' = \frac{2}{5M}\left[(V + M) + \sqrt{(V + M)^2 - 5VM}\right].$$

$$\pi' = -2\left(V - \frac{M}{2}\right) - \sqrt{\left(V - \frac{M}{2}\right)\left(V - \frac{3M}{2}\right)},\ \pi'' = \frac{2M - 8V}{25M}\left[(V + M) + \sqrt{(V + M)^2 - 5VM}\right] - \frac{V}{5}.$$

Taken together, these insights provide useful references to software developers and consumers in making their development or adoption decisions. In particular, in the Internet age, consumers can readily observe the functionality and quality of software products. It is infeasible for developers to take advantage of information asymmetry in selling computer software. Also, owing to the negligible marginal cost of production and distribution, the pricing of computer software affixes more on consumer taste preferences. It would be more rewarding for software developers to specify carefully the functionality of their products and price accordingly in order to capture the highest possible demand.

In view of the result of no NE in the one-stage duopoly model, software developers may find it challenging to make functionality decisions when they face a potential competition from another developer. To mutually reach in viable strategies that are beneficial to both parties, a certain level of predevelopment communication (such as making preannouncements, which are rather common in software markets) or contracting between the competing developers might be advantageous. Such actions help to coordinate the choices of the developers so that they might be able to partition the market and focus on their own niches. The precise effect of these coordinative actions on the final software markets warrants future attentions and research efforts.

Alternatively, if a software developer cannot reasonably speculate the likely functionality decision of his competitor, and the demand side of the market is more uncertain toward adopting the software, he might be better off by simply assuming a wait-and-see strategy. Such a strategy allows the developer to more accurately position his product and appeal to a wider consumer group without forgoing too much "initial" profit. In any case, our analysis on the two-stage model suggests that it might not be a good idea for the "innovator" to build too many functions into his software. This is because competitions from subsequent followers might cannibalize the profit enjoyed the incumbent, should he decide to go for high functionality. In contrast, by going for mediocre software, the incumbent might be able to induce followers into another consumer group that is not currently attended to. Our results, however, cannot provide clear-cut guidance when the demand is time sensitive and consumers are very eager toward buying the software.

Our functionality model assumes developers to produce either zero or one software package. When the market size is big and entrance cost is not high, there might be "space" for new developers to come in and share the rents. In this case, a widely pronounced strategy for the incumbent is to develop a full version (but not necessarily high functionality) and then subsequently create low-value versions by excluding subsets of functions [40, 41]. As long as the full version is developed, the cost for generating another low-value version is substantially reduced, which may present the incumbent an initial cost advantage against new entrants who want to invade the low-value segment.

In our model, if the monopoly developer is allowed to sell two versions of the software, and there is no threat of potential entrants, our results on the one-stage duopoly market (particularly the division point Q'') may provide a good guidance toward setting the levels of the two versions. This is because, for the monopolist,

there are no coordination or deviation issues. He can freely decide the Qs for his two versions.

The case of versioning is more complicated in the duopoly model. In the one-stage model, the result of no NE should continue to apply.[20] In the two-stage model, the possibility of versioning might suggest the first mover to develop full functionality software and then sell multiple versions (which vary in functionality) to the market. Nevertheless, as long as the potential rent can offset the development cost of new software, other developers might continue to enter the market and compete with the existing versions. That is, even if the first mover "covers up" the market by two or more versions of his software, the second mover might still enter and launch software with appropriate "in between" functionality. This is actually similar to a market with monopolistic competition, where many software developers enter the market and sell a single edition of the software. Other concerns, such as marketing and advertising costs, entry barriers, or possibility of cannibalizations [32, 38], might also be in play in such conditions. The resultant strategies and outcome of such a "saturated" market is beyond the scope of this paper, and our functionality model might not be rich enough to capture the dynamics of such a market. Further research is needed to characterize this kind of competitive software market under the theme of functionality.

Another possible extension to the present study is to consider multistage settings (or, in the extreme case, infinite stages) in which the decisions are repeated over time. This would be a more complicated market environment in the sense that the decision of each software developer might trigger a series of reactions from all other developers in subsequent periods.

Finally, we considered the development and selling of individual computer software. If several software programs are bundled and sold together (like the Microsoft Office Suite), even if they are completely unrelated and fall under several markets, consumer preferences and their final choices might still be affected because the purchase decision is now based on the whole "package" instead of individual products. In this case, we might need to formulate an extensive optimization program with multiple functionality and user requirement constraints pertaining to the included software. Even so, the principal idea of studying competitions of computer software using functionality as a key attribute stays put, and software developers are advised to conduct thorough market analysis and user preference studies before constructing their computer software.

NOTES

1. It is only necessary, but not sufficient. As pointed out by previous research, the final choice of software also depends on other socioeconomic variables such as vendor reputation, existing size of customer base, or market trend [28].

2. Although we focus on functionality, the model that we present in the next section could also be extended to include the effects of other choice variables like vendor reputation or application domain-specific factors. We defer that discussion until the third section.

3. In Raghunathan's article, the term "feature" is used to characterize the two editions of the software. Based on his descriptions, however, we infer that the meaning of "feature" is indeed function related. For example, he states that "we assume that any feature present in the low-end

edition is also present in the high-end edition . . . editions are upward compatible, an important requirement in software" [38, p. 90]. In this study, the terms "feature" and "function" are used interchangeably to denote routines or components that enhance the capability of the software.

4. The major difference lies in trialability. As mentioned earlier, the digital and functional natures of computer software facilitate the practice of product tryout. Since consumers are allowed to gain hands-on experience on the software, they can acquire more accurate information on the performance, features, and quality of the software. In other words, the prepurchase tryout experience can help to establish a solid information base for product evaluation and choice. Such an extensive and precise product assessment, in general, cannot be conducted for pure informational goods. For example, when evaluating the suitability of online reports or data, consumers are often only given descriptive information or at best a small sample of the product. The quality of the remaining parts of the product remains highly uncertain, and consumers need to rely on their subjective judgments instead of objective information to guide their purchase decisions.

5. Obviously, in this situation, the choice of consumers is no longer restricted to functionality, and the formulation of the consumer surplus function depends on other decision variables (those associated with λ and δ). Indeed, the role of the expression $g(Q)$ is more like the "value" (or "quality") function in past strategic software studies, with functionality being the key variable that characterizes the value (or quality) of the software.

6. It should be noted that λ and δ are exogenously determined (possibly by previous track records of software quality or popularity) for each software vendor, and they are not controlled by the vendors. If the competing vendors are sufficiently "similar" (that is, their λs and δs are not too different) so that the same Q leads to comparable $g(Q)$, our subsequent analysis and results will remain largely unchanged (only minor shifts in optimal Qs will be observed in the final equilibria). If, however, the two vendors are overly different so that one enjoys a definite advantage relative to the other, then the low "quality" vendor could only target at the low-value segment (because in this case, he faces a lower upper bound on maximum quality), and the two vendors will naturally segment the market into two clusters, with the high "quality" vendor serving the high-value segment of consumers. To avoid distracting our attention on functionality, we shall cease our discussion here without going into details about the effects of imposing different λs and δs into our model.

7. We assume equal cost parameters (V and α) across developers. Since the development of computer software involves common activities like programming and testing, it is unlikely for two developers to possess widely different development costs. Put another way, if it costs substantially more for a developer to produce the same software than another developer, then it probably indicates that the developer may have made suboptimal development decisions (such as, he may have chosen an inappropriate programming environment or design method). If V and α differ only slightly across developers, our subsequent results will largely hold with only minor shifts in optimal Qs.

8. This is opposite to previous scale economies studies, where an $\alpha > 1$ (< 1) indicates diseconomies (economies) of scale. We need to reverse the meaning of α here because $Q \in [0,1]$, and having an $\alpha > 1$ (< 1) actually reduces (increases) the overall cost, indicating increasing (decreasing) returns to scale.

9. The software engineering literature suggests that the development cost increases with the size of computer software. Traditionally, software size was mainly measured by either function points or source lines of code (SLOC), and robust technique was devised to reduce measurement errors and calibrate cost models using these software metrics [17]. Furthermore, the two size measurements (SLOC and FP) were shown to be highly correlated with each other [27]. Hence, the positive association between functionality and development cost is just a direct extension of the size–cost relationship.

10. For instance, using published software project data, Banker and Kemerer [5] show that the scale economies parameter ranged from 0.85 to 1.49 if FP was used as the size metric, whereas it ranged from 0.72 to 1.11 if SLOC was used. Surprisingly, the majority of the estimated parameters fell closely to one, indicating minor economies (or diseconomies) of scale.

11. If the developers do not know the cost parameters of each other, they need to rely on a priori beliefs on the development efficiency and estimate the cost of their opponent. The developers can gather information from multiple sources to guide their estimations. They could base

their estimations on their own expertise and experience on software developments. Alternatively, if preannouncements (or other market news) were previously released, it might be possible for them to infer the development tactics or approaches of their competitor. Therefore, the developers should be able to estimate their competitor's cost with reasonable accuracy, and the common knowledge assumption is not particularly restrictive in this context. In fact, even if minor deviations exist in their estimations, the results of our model remain intact, and similar optimal strategies can be readily derived.

12. The new price $P = C$ is still below the valuation of consumers in the interval $(D, 1]$, whereas it is higher than the price $D/2$. Hence, the high Q player can raise his price up to C without driving away consumers.

13. The negative quadratic root is ruled out since Q' is nonnegative.

14. That is, when the per-function, onetime development cost is very minor compared to the size of the overall market.

15. Again, the negative root is ruled out since Q'' cannot be less than 0.5.

16. The firms' pricing strategies must follow some stringent conditions. Recall monopoly demand $D = M(Q - P)$. A rational firm will never charge P higher than Q, since that will drive away all consumers. The pricing strategy $P = aQ$ (with $a \leq 1$) can effectively prevent such an irrational pricing behavior, and yet it is general enough to cover (by suitable adjustments of a) the optimal pricing behaviors that we discussed in the previous sections.

17. As pointed out by Hirshleifer and Riley [24], a key idea (or indeed assumption) in the reactive equilibrium concept is that the defector cannot simply hit and run. That is, player 1 cannot deviate by producing $Q_1 > Q''$ in stage 1, capture enough profit from consumers, and then withdraw from the market before player 2 reacts. This criterion is satisfied here because we explicitly assumed that consumers are patient. They can wait until the appearance of both software packages before making purchases.

18. Because we set $Q_H = 1$, the profit of the monopoly developer remains the same as in the constant returns to scale setting. This is generally not the case when $Q \neq 1$. Obviously, profit is higher when $\alpha > 1$ (increasing returns to scale), whereas it is lower when $\alpha < 1$ (decreasing returns to scale). Note that these results hold only if α is not too "extreme" and $M >> V$. These are largely supported by previous scale economies studies and the nature of COTS software markets.

19. For instance, many commercial software packages serve both individual consumers and business users, with the business users requiring higher levels of functionality. Examples include statistical packages, where most of the time individual users only need basic estimation functions, as opposed to organization users who desire complex analytic functions; or Adobe Photoshop, where home users are, most of the time, satisfied with the default list of filters, whereas professional users often pay for additional sophisticated filters.

20. It is not difficult to see this result if we consider the two "adjacent" versions that are produced by the two developers. In this case, the segment covered by these two versions is similar to a market by itself, and there is always a profit incentive for one of the developers to deviate.

REFERENCES

1. Ackerlof, G.A. The market for "lemons": Quality uncertainty and the market mechanism. *Quarterly Journal of Economics, 84,* 3 (1970), 488–500.

2. Albrecht, A.J., and Gaffney, J.E. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering, SE-9,* 6 (June 1983), 639–648.

3. Bakos, Y., and Brynjolfsson, E. Bundling information goods: Pricing, profits and efficiency. *Management Science, 45,* 12 (1999), 1613–1630.

4. Bakos, Y.; Brynjolfsson, E.; and Lichtman, D. Shared information goods. *Journal of Law and Economics, 42,* 1 (April 1999), 117–155.

5. Banker, R.D., and Kemerer, C.F. Scale economies in new software development. *IEEE Transactions on Software Engineering, 15,* 10 (October 1989), 1199–1205.

6. Banks, J., and Gibson, R.R. Selecting simulation software. *IIE Solutions, 29,* 5 (May 1997), 30–32.

7. Boehm, B.W. *Software Engineering Economics*. Upper Saddle River, NJ: Prentice Hall, 1981.

8. Boehm, B.W., and Papaccio, P.N. Understanding and controlling software costs. *IEEE Transactions on Software Engineering, 14,* 10 (October 1988), 1462–1477.

9. Brown, A.W., and Wallnau, K.C. A framework for evaluating software technology. *IEEE Software, 13,* 5 (September 1996), 39–49.

10. Caldiera, G.; Antoniol, G.; Fiutem, R.; and Lokan, C. Definition and experimental evaluation of function points for object-oriented systems. In *Proceedings of the Fifth International Software Metrics Symposium*. Los Alamitos, CA: IEEE Computer Society Press, 1998, pp. 167–178.

11. Chan, Y.E., and Storey, V.C. The use of spreadsheets in organizations: Determinants and consequences. *Information and Management, 31,* 3 (1996), 119–134.

12. Chartier, A. Gold is the details: Micropayment's big future. *Computing Canada, 25,* 7 (February 1999), 28.

13. Chau, P.Y.K. Factors used in the selection of packaged software in small businesses: Views of owners and managers. *Information and Management, 29,* 2 (1995), 71–78.

14. Choi, S.; Stahl, D.O.; and Whinston, A.B. *The Economics of Electronic Commerce*. Indianapolis, IN: Macmillan Technical Publishing, 1997.

15. Choudhary, V.; Tomak, K.; and Chaturvedi, A.R. Economic benefits of software renting. *Journal of Organizational Computing and Electronic Commerce, 8,* 4 (1998), 277–305.

16. Collier, K.; Carey, B.; Sautter, D.; and Marjaniemi, C. A methodology for evaluating and selecting data mining software. In R.H. Sprague Jr. (ed.), *Proceedings of the Thirty-Second Hawaii International Conference on System Sciences*. Los Alamitos, CA: IEEE Computer Society Press, 1999, pp. 1–11.

17. Ebrahimi, N.B. How to improve the calibration of cost models. *IEEE Transactions on Software Engineering, 25,* 1 (January–February 1999), 136–140.

18. Ellison, G., and Fudenberg, D. The Neo-Luddite's lament: Excessive upgrades in the software industry. *RAND Journal of Economics, 31,* 2 (Summer 2000), 253–272.

19. Fenton, N.E., and Pfleeger, S.L. *Software Metrics: A Rigorous and Practical Approach,* 2d ed. Boston: International Thomson Computer Press, 1997.

20. Franco, M.D. Choosing a software system. *Catalog Age, 15,* 12 (November 1998), 65.

21. Gibbons, R. *Game Theory for Applied Economists*. Princeton: Princeton University Press, 1992.

22. Goodwin, N.C. Functionality and usability. *Communications of the ACM, 30,* 3 (March 1987), 229–233.

23. Gurbaxani, V., and Mendelson, H. An empirical analysis of software and hardware spending. *Decision Support Systems, 8,* 1 (1992), 1–16.

24. Hirshleifer, J., and Riley, J.G. *The Analytics of Uncertainty and Information*. Cambridge: Cambridge University Press, 1992.

25. Hoxmeier, J.A. Software preannouncements and their impact on customers' perceptions and vendor reputation. *Journal of Management Information Systems, 17,* 1 (Summer 2000), 115–139.

26. Hu, Q.; Plant, R.T.; and Hertz, D.B. Software cost estimation using economic production models. *Journal of Management Information Systems, 15,* 1 (Summer 1998), 143–163.

27. Kemerer, C.F. An empirical validation of software cost estimation models. *Communications of the ACM, 30,* 5 (May 1987), 416–429.

28. Kunda, D., and Brooks, L. Identifying and classifying processes (traditional and soft factors) that support COTS component selection: A case study. *European Journal of Information Systems, 9,* 4 (2000), 226–234.

29. Marshall, G. Finding the right software. *Agency Sales, 29,* 2 (February 1999), 4–7.

30. Matson, J.E.; Barrett, B.E.; and Mellichamp, J.M. Software development cost estimation using function points. *IEEE Transactions on Software Engineering, 20,* 4 (April 1994), 275–287.

31. McGrenere, J., and Moore, G. Are we all in the same "bloat"? In *Proceedings of Graphics Interface 2000.* Available at www.graphicsinterface.org/proceedings/2000/144/PDFpaper144.pdf.

32. Moorthy, K.S., and Png, I.P.L. Market segmentation, cannibalization, and the timing of product introductions. *Management Science, 38,* 3 (March 1992), 345–359.

33. Morisio, M., and Tsoukias, A. IusWare: A methodology for the evaluation and selection of software products. *IEEE Proceedings on Software Engineering, 144,* 3 (June 1997), 162–174.

34. Morisio, M.; Stamelos, I.; Spahos, V.; and Romano, D. Measuring functionality and productivity in Web-based applications: A case study. In *Proceedings of the Sixth International Software Metrics Symposium.* Los Alamitos, CA: IEEE Computer Society Press, 1999, pp. 111–118.

35. Ochs, M.; Pfahl, D.; Chrobok-Diening, G.; and Nothhelfer-Kolb, B. A method for efficient measurement-based COTS assessment and selection—Method description and evaluation results. In *Proceedings of the Seventh International Software Metrics Symposium.* Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 285–296.

36. Padmanabhan, V.; Rajiv, S.; and Srinivasan, K. New products, upgrades, and new releases: A rationale for sequential product introduction. *Journal of Marketing Research, 34,* 4 (November 1997), 456–472.

37. Patch, K. Drop a dime online. *InfoWorld, 20,* 48 (November 1998), 71–72.

38. Raghunathan, S. Software editions: An application of segmentation theory to the packaged software market. *Journal of Management Information Systems, 17,* 1 (Summer 2000), 87–113.

39. Sarvary, M., and Parker, P.M. Marketing information: A competitive analysis. *Marketing Science, 16,* 1 (1997), 24–38.

40. Shapiro, C., and Varian, H.R. Versioning: The smart way to sell information. *Harvard Business Review, 76,* 6 (1998), 106–114.

41. Shapiro, C., and Varian, H.R. *Information Rules: A Strategic Guide to the Network Economy.* Boston: Harvard Business School Press, 1999.

42. Varian, H.R. Markets for information goods. Working paper, University of California, Berkeley, 1998.

43. Voas, J. COTS software: The economical choice? *IEEE Software, 15,* 2 (March-April 1998), 16–19.

44. Wedel, M., and Kamakura, W. *Market Segmentation: Conceptual and Methodological Foundations,* 2d ed. Boston: Kluwer Academic Publishers, 2000.

45. West, R., and Shields, M. Strategic software selection. *Strategic Finance* (August 1998), 3–7.

Appendix A

$$\pi_{(\text{equal})} = \frac{1}{2}M(Q-P)P - VQ$$

$$\frac{\partial \pi_{(\text{equal})}}{\partial Q} = \frac{MP}{2} - V$$

$$\frac{\partial \pi_{(\text{equal})}}{\partial P} = \frac{MQ}{2} - MP = 0 \Rightarrow P = \frac{Q}{2}.$$

Following the same steps in the previous analysis, $\partial\pi_{(\text{equal})}/\partial Q \geq 0$ if and only if $Q \geq 4V/M$. If $Q < 4V/M$, $\pi_{(\text{equal})}$ is non-positive. Hence, optimality requires $Q = 1$ and the maximum payoff that the two players can achieve when they set equal Q is $\pi_{(\text{equal})} = M/8 - V$.

Now compare π' and $\pi_{(\text{equal})}$.

$$\begin{aligned}
&\pi' - \pi_{(\text{equal})} \\
&= -2\left(V - \frac{M}{2}\right) - \sqrt{\left(V - \frac{M}{2}\right)\left(V - \frac{3M}{2}\right)} - \left(\frac{M}{8} - V\right) \\
&= -\left(V - \frac{7M}{8}\right) - \sqrt{V^2 - 2VM + \frac{3}{4}M^2} \\
&= \left(\frac{7M}{8} - V\right) - \sqrt{\left(\frac{7M}{8} - V\right)^2 - \frac{VM}{4} - \frac{M^2}{64}} \\
&> \left(\frac{7M}{8} - V\right) - \sqrt{\left(\frac{7M}{8} - V\right)^2} = 0.
\end{aligned}$$

Therefore, $\pi' > \pi_{(\text{equal})}$.

Appendix B

FIRST, LET $B = V/M$, $0 \le B \le 1/4$ (since $V \le M/4$).

$$Q'' = \frac{2}{5}\left(B+1+\sqrt{B^2-3B+1}\right) = \frac{2}{5}\left(B+1+\sqrt{(B-1)^2-B}\right)$$

$$Q' = 2\left(B-\frac{1}{2}+\sqrt{B^2-2B+\frac{3}{4}}\right) = 2\left(B-\frac{1}{2}+\sqrt{(B-1)^2-\frac{1}{4}}\right)$$

$$Q''-Q' = 1-\frac{2}{5}\left[5\sqrt{(B-1)^2-\frac{1}{4}}-\sqrt{(B-1)^2-B}\right].$$

When

$$B=\frac{1}{4},\ Q''-Q' = 1-\frac{2}{\sqrt{5}} > 0.$$

Hence, $Q'' > Q'$.

When $B < 1/4$, the second term in $Q'' - Q'$ decreases in magnitude. Hence, Q'' always exceeds Q'.

Now recall

$$\pi'' = \frac{M}{4}Q'' - VQ'' \quad \text{and} \quad \pi' = \frac{M}{4}Q' - VQ'.$$

$$\pi''-\pi' = \frac{M}{4}(Q''-Q')-V(Q''-Q') = \left(\frac{M}{4}-V\right)(Q''-Q') > 0.$$

Hence, $\pi'' > \pi'$.

Appendix C

WE FIRST CONSIDER THE PAYOFF to the software developers if they both choose Q according to a uniform probability distribution. Call this strategy S_U. If player 2 sets price $P_2 = aQ_2$ (with $a \le 1$), his payoff is

$$
\begin{aligned}
E\pi_2(a) &= \pi_2(\text{when } Q_2 > Q_1 \text{ and } P_2 > Q_1) + \pi_2(\text{when } Q_2 > Q_1 \text{ and } P_2 \le Q_1) \\
&\quad + \pi_2(\text{when } Q_2 < Q_1) \\
&= \int_0^1 \left[\int_0^{aQ_1} \left[M(Q_2 - aQ_2)aQ_2 - VQ_2 \right] dQ_1 + \int_{aQ_1}^{Q_2} \left[M(Q_2 - Q_1)aQ_2 - VQ_2 \right] dQ_1 \right. \\
&\quad \left. + \int_{Q_2}^1 \left[M(Q_2 - aQ_2)aQ_2 - VQ_2 \right] dQ_1 \right] dQ_2 \\
&= \int_0^1 \left[\left[aM(1-a)Q_2^2 - VQ_2 \right] aQ_2 + \left[aMQ_2^2 Q_1 - \frac{aMQ_2 Q_1^2}{2} - VQ_1 Q_2 \right]_{Q_1 = aQ_2}^{Q_2} \right. \\
&\quad \left. + \left[aM(1-a)Q_2^2 - VQ_2 \right](1 - Q_2) \right] dQ_2 \\
&= \int_0^1 \left[\left(\frac{2Ma^2 - a^3 M - aM}{2} \right) Q_2^3 + M(a - a^2)Q_2^2 - VQ_2 \right] dQ_2 \\
&= \frac{5aM}{24} - \frac{a^2 M}{12} - \frac{a^3 M}{8} - \frac{V}{2}.
\end{aligned}
$$

Now,

$$
\begin{aligned}
\frac{dE\pi_2(a)}{da} &= \frac{5M}{24} - \frac{aM}{6} - \frac{3a^2 M}{8} = 0 \\
&\Rightarrow 5 - 4a - 9a^2 = 0 \\
&\Rightarrow (9a - 5)(a + 1) = 0 \\
&\Rightarrow a = -1 (\text{rejected}) \quad \text{or} \quad a = \frac{5}{9}.
\end{aligned}
$$

Therefore, when both players play S_U (the uniform mixed strategy), each of them will receive the maximum payoff if they follow a constant pricing strategy $P = 5Q/9$.

Payoff in this case is

$$
E\pi_2(a)\Big|_{a=\frac{5}{9}} = \frac{50M}{729} - \frac{V}{2}.
$$

Denote this payoff X. Next consider the payoff to player 2 when, instead of mixing uniformly over [0, 1], he plays the fixed strategy $Q_2 = 1$ and $P_2 = aQ_2 = a$. His new payoff is

$$\begin{aligned}
E\pi_2(1,a) &= \pi_2(\text{when } Q_2 > Q_1 \text{ and } P_2 > Q_1) + \pi_2(\text{when } Q_2 > Q_1 \text{ and } P_2 \le Q_1) \\
&= \int_0^a \left[M(1-a)a - V\right] dQ_1 + \int_a^1 \left[M(1-Q_1)a - V\right] dQ_1 \\
&= \left[M(1-a)a - V\right]a + \left[Ma - V\right][1-a] - \frac{aMQ_1^2}{2}\Bigg|_{Q_1=a}^{1} \\
&= \frac{aM}{2} - V - \frac{a^3 M}{2} \\
\frac{dE\pi_2(1,a)}{da} &= \frac{M}{2} - \frac{3a^2 M}{2} = 0 \\
&\Rightarrow 1 - 3a^2 = 0 \\
&\Rightarrow a = \frac{-1}{\sqrt{3}} (\text{rejected}) \quad \text{or} \quad a = \frac{1}{\sqrt{3}}.
\end{aligned}$$

Hence, the optimal payoff for player 2 in this case is achieved at $a = 1/\sqrt{3}$.

The corresponding profit is

$$E\pi_2(1,a)\Big|_{a=\frac{1}{\sqrt{3}}} = \frac{M}{3\sqrt{3}} - V.$$

Denote this payoff Y.

Now

$$\begin{aligned}
Y - X &= \frac{M}{3\sqrt{3}} - V - \left(\frac{50M}{729} - \frac{V}{2}\right) \\
&= \frac{81\sqrt{3} - 50}{729} M - \frac{V}{2}.
\end{aligned}$$

Again, provided V is not close to $M/4$, $Y - X$ is always positive. If V is close to $M/4$, both Y and X are non-positive. Hence, when the per-function, onetime development cost is not too high relative to the total market potential, the profit given by a fixed strategy $Q = 1$ exceeds the one given by S_U. When the cost is indeed too high, strategy S_U will yield a negative profit, and the firms can benefit by playing the constant strategy $Q = 0$ (staying out of the market). In any case, strategy S_U is not a best response to itself, and therefore it cannot be an NE.

More generally, when consumer type is not uniformly distributed, proposition 2 can always be tailored according to the new demand distribution to eliminate all but the perfectly mixed strategy. However, because the demand is now unbalanced, it is easy to see that another mixed (or, occasionally, pure) strategy can be constructed that pays better than the perfectly mixed strategy. Hence, our proof of no mixed strategy NE is general enough to cover other functional forms of consumer distribution.