# A MATHEMATICAL ANALYSIS OF THE BUSINESS OF SOFTWARE

PAUL ROSENBERG

*Isaac, I hope you like it as much as I do.*

Abstract. In this paper I investigate the behavior of the market share of a hypothetical company as a function of:

- name recognition of the company, which depends on
  - efficiency of its traditional advertising
  - word-of-mouth transmission of its existence
  - FUD tactics of its rivals
- reliability of its software, which depends on
  - the number of its developers
  - the number of $\beta$-testers for its software
  - whether its applications are single or members of suites
- volume of its software titles, which depends on
  - the number of its developers
- price of its software
- brand loyalty
- age of the company
- customer satisfaction

The fundamental question is whether it is possible for one company to dominate its niche indefinitely. The short answer is yes, under certain conditions it is possible.

Contents



Thanks Hari for inspiring a child to wonder.

## 1. Background

I will make use of differential and difference equations to model the growth and interaction of various quantities as a function of time. Furthermore, I will make use of some elementary probability theory.

For anybody who has not taken a calculus course and who wants to understand the derivation of my results, consult with someone who has sufficient experience with calculus and basic probability theory for a satisfactory explanation of the derivations of my results.

Here is a summary of properties/results required from calculus:

1. If $f(t_0) \geq g(t_0)$, $f', g'$ exist for all $t \geq t_0$, and $f'(t) \geq g'(t)$ for all $t \geq t_0$ then $f(t) \geq g(t)$ for all $t \geq t_0$.
2. If $f$ satisfies a first order homogeneous autonamous differential equation,

$$\frac{dy}{dt} = P(y)$$

such that $P(f(t_0)) = 0$ for some time $t_0$ then $f$ is of the form $f =$ constant.

As a rule, most differential equations do *not* have explicit solutions. But using the above results I can recover qualitative information from the differential equations, specifically rates of growth. Rates of growth is the key property any company aspiring to dominate its niche economy *must* know.

The software that I am interested in modelling is the type known as shrink-wrapped software, that is software to be used (and paid for) by a percentage of the computer using business/public as opposed to in-house software, which is meant only for internal use.

I will denote the hypothetical software company as Software Inc.Unless explictly stated otherwise, Software Inc. hypothetically sells proprietary or closed-source software.

## 2. Brand Recognition

To capture, or at least increase, its market share in its target market, Software Inc. must have brand recognition by a sufficient percentage of its target consumers. How can Software Inc. manage this feat?

Two classical methods exist:

**Word of mouth:** Given that some consumers know of Software Inc.'s existence, if satisfied, those consumers will pass this knowledge onto a percentage of their friends/acquaintances. Thus, this will increase Software Inc.'s brand name recognition.

**Advertising:** The type of advertising dictates the efficiency of *informing* target consumers of the existence of Software Inc. For example, advertising via television might be more effective than advertising via "junk" mail. Or, perhaps target consumers might be better reached through advertisements in trade magazines.

The point here is that the type of advertising chosen will affect the rate at which brand name recognition increases for Software Inc.

Since fear, uncertainty, and doubt (FUD) tactics by rival companies of Software Inc. have the unintended effect of increasing brand recognition of Software Inc., we should give a few moments to the consideration of the two-fold effects of FUD:

1. It increases the brand recognition of Software Inc.
2. It alters the probability that target consumers are willing to do business with Software Inc.

**Ineffective refutation of FUD:** FUD tactics of rival companies have the effect of lowering market share of Software Inc.

**Effective refutation of FUD:** FUD tactics of rival companies have the effect of *increasing* market share of Software Inc.

Another factor to consider in our model of brand recognition is the memory retention problem. Without periodic reminders of Software Inc.'s existence, the memory of its existence will decay exponentially.

Let us proceed now to our model. Here are the variables:

1. $t$ — the variable denoting time since Software Inc.'s creation
2. $N$ — the number of target consumers who are aware of Software Inc.'s existence at time $t$
3. $N_{max}$ — the total number of target consumers
4. $N_\infty$ — the stable value of $N$ as $t \longrightarrow \infty$
5. $N_0$ — the initial number of consumers who are aware of Software Inc.'s existence

### 2.1. Word of mouth and advertising are used.

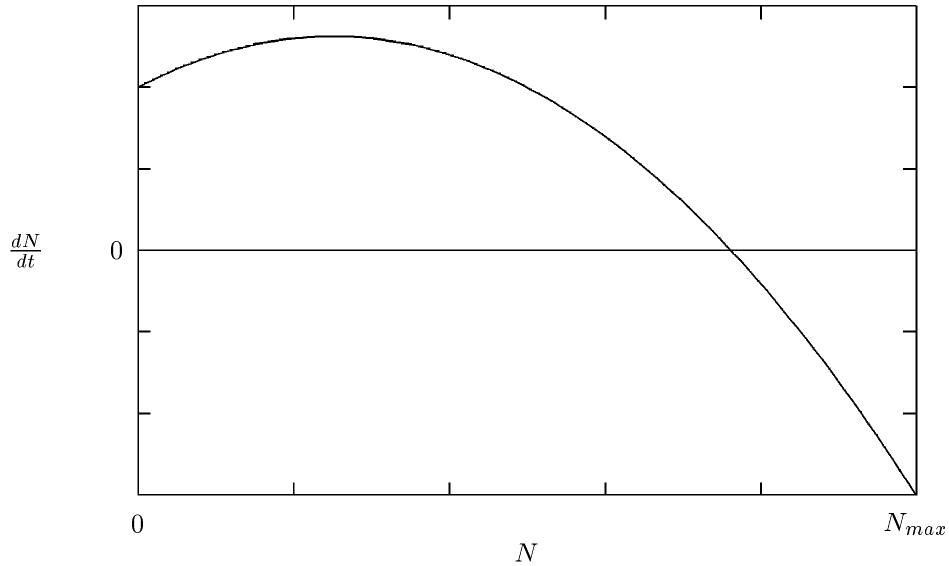
Here is the first model:

$$\frac{dN}{dt} = \alpha(N_{max} - N)N + (\beta + \delta)(N_{max} - N) - \gamma N \tag{1}$$

where $\alpha, \beta, \gamma, \delta > 0$ are constants. The first term, $\alpha(N_{max} - N)N$, represents the effect of word of mouth. The second term, $(\beta + \delta)(N_{max} - N)$, represents the combined effect of Software Inc.'s advertising and its rivals' FUD campaign. The third term, $-\gamma N$, represents the memory decay effect.

Our first question is whether an equilibrium point exists. The answer is yes, as a quick view of a phase diagram will show. Using some simple algebraic methods,

FIGURE 1. A sample phase flow graph of $N$ vs. $\frac{dN}{dt}$




the equilibrium point is computed to occur at:

$$N_\infty = \frac{\sqrt{(\beta+\delta+\gamma-\alpha N_{max})^2 + 4\alpha(\beta+\delta)N_{max}} - |\beta+\delta+\gamma-\alpha N_{max}|}{2\alpha} \tag{2}$$

Notice from the graph that for $N$ small the rate of growth is positive while for $N$ close to $N_{max}$ the rate of growth is negative. This can be verified easily for the endpoints. For $N = N_{max}$,

$$\frac{dN}{dt} = 0 + 0 - \gamma N < 0$$

Hence $N$ is decreasing. For $N = 0$,

$$\frac{dN}{dt} = 0 + (\beta+\delta)N_{max} > 0$$

Hence $N$ is increasing.

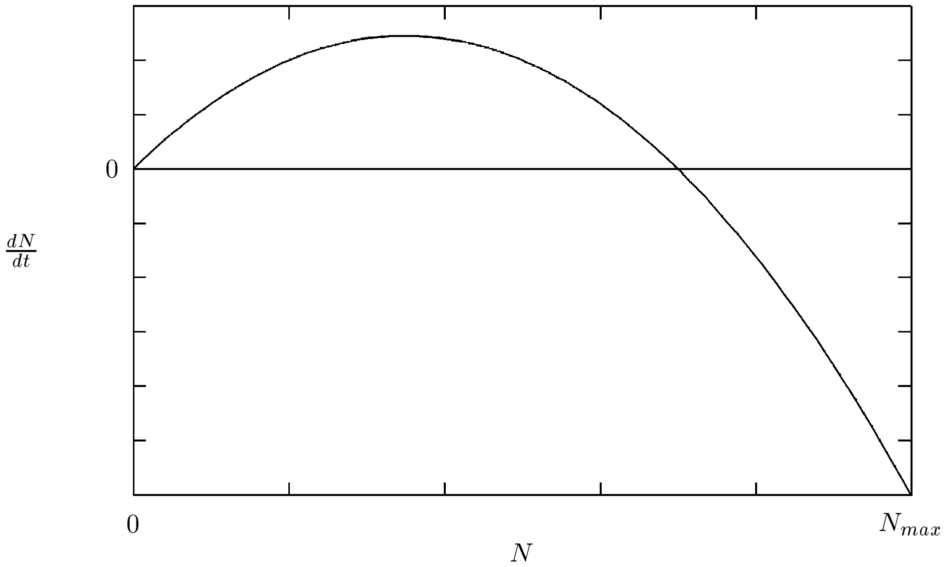
2.2. **Only word of mouth is used.** In the second model let us assume that $\beta$ and $\delta$ are zero. What sort of behavior can we expect?

Firstly, the equilibrium value $N_\infty$ occurs at

$$N_\infty = N_{max} - \frac{\gamma}{\alpha}.$$

Secondly, the essential difference of using advertising versus not using advertising is that in the latter case if no one knows of Software Inc.'s existence at time $t = 0$ then no one will ever know of its existence. This is an unrealistic scenario however since the owners of Software Inc. know of its existence. The second essential difference is that the rate of growth of brand recognition is much slower initially without the benefit of advertising. The third essential difference is that the equilibrium value of $N_\infty$ will be lower than it would have been with advertising. How much lower

FIGURE 2. A sample phase flow with no advertising




depends on the efficiency of word of mouth versus advertising. Or in other words, whether the ratio $\gamma/\alpha$ is large or small.

2.2.1. $\gamma/\alpha \sim O(N_{max})$. In this case word of mouth fails to keep up with memory loss. Hence

$$N_\infty = N_{max} - \frac{\gamma}{\alpha} << N_{max}$$

Thus little brand recognition occurs. This is typical of a pre-internet society in which the effectiveness of word of mouth is limited by the speed of transmission and the effort required to "spread the word" so to speak. While some types of extraordinary information can spread effectively under these circumstances[1], ordinary types of information would not. As a challenge to the reader, think of a single *ordinary* product the knowledge of which spread through a significant percentage of the population and remained *consistently* in its target audience's minds *without the benefit of advertising.* Under what circumstances did this occur?

2.2.2. $\gamma/\alpha << N_{max}$. In this case, word of mouth alone is highly efficient at reaching the target consumers. This could occur in the age of the internet if a significant percentage of the target consumers use the internet. The speed of transmission of e-mail is negligible as is the effort required to compose or forward an e-mail.

But is it realistic? Based on an estimate of 14 million Linux(tm) users before Linux(tm) became the recipient of free advertising by the media versus 300 million users of Microsoft(tm) it would appear to actually work. A 4.5% share of a market without the benefit of advertising is rather remarkable. Taking into account the fact that the amount effort required to acquire Linux(tm) and install it *individually*

[1] Ex. "A state of war now exists between our country and ..."

was significantly greater than having it factory installed is a testament to how well the word of mouth must actually work via the internet.

## 3. Reliability

Assume that Software Inc. has announced a new application which for notational purposes we'll denote as App. An important concern of the target audience (and an indirect concern for Software Inc.) is the reliability of App. By reliability I mean:

> How often does App crash or produce erroneous results given an arbitrary use of the product?

Clearly, if the probability of an undesirable result occurring is significantly high, say 50%, the percentage of the target audience willing to use the product is likely to be small. Given that the performance of App must begin at some initial level of reliability be it introduced publicly as open source, or internally as proprietary, can we predict the future reliability of App?

An obvious prediction is that the reliability of App *must* increase if App has any hope of continued use by some non-zero percentage of the target audience. However, this prediction is not very satisfying because it says nothing of how *soon* or how *much greater* the reliability will improve. Obviously, the reliability cannot increase above 100%, so we know empirically that as the product becomes more reliable we should expect the rate at which the reliabilty improves should *decrease*.

A better approach is to ask:

> What factors influence the rate of improvement of the reliability of App?

Here's a list that I would argue are the essential factors:

- the number of developers, by which I mean those individuals who *submit* code for review, *not* just those individuals assigned to maintain the code for App;
- the number of $\beta$-testers, that is, those indviduals who submit bug reports for App;
- the size of the code; and,
- the interaction between App and other applications, such as the operating system.

I contend that the number of developers has a positive influence *if* the goal is to fix bugs, *not* to develop new features. The justification is simple: the more submitted patches, the more likely the review committee (those individuals in charge of accepting patches for the code base) will *find* an acceptable patch, or that a review committee member will be inspired by the submitted patches to write a more error resistant patch.

To restate Eric S. Raymond in his paper, [3]:

> The more individuals testing the software, the more likely/quickly they will find bugs to report.

Also, the size of the code has an indirect impact on the reliability of App. The larger the code grows, the more difficult to find the source of the bug, the more likely there will be a synergistic effect betweem seemingly unrelated parts of the code. Please note that when I'm referring to the size of the code, I'm not thinking in terms of an application having say 10,000 lines of code. I'm referring to applications whose lines of code are measured in the hundreds of thousands or millions.

To model the effects of code size I will treat the software as if it were a thermodynamic system consisting of $N$ identical particles where $N$ is the number of lines of code. I am treating the lines of code as if they were individual particles in which the interaction between 2 lines of code is independent of the actual details of the 2 lines of code. Any 2 lines of code can potentially interact. This is clearly a profoundly *wrong* assumption, but *if* the number of lines of code is *large enough* the assumption works qualitatively.

Lastly, the interaction of App with other applications cannot be ignored. If App were a member of a suite of applications which of themselves had no apparent bugs, but because of the underlying relationship between the applications App had a hidden influence on the behavior of the other applications and vice versa, the user of App could be confounded by mysterious bugs which the developers of App could not find, because they could not reproduce the results without the knowledge of which applications had to be running concurrently and in which order the applications had been activated.

Let $R$ denote the reliability of App, $0 \leq R \leq 1$, and $t$ denotes time in whatever units seem convenient. Let $D$ denote the number of developers, and $B$ the number of $\beta$-testers. Let $S$ stand for the size of of the code. Presumably $S$ is a function of $R$. I will assume $S$ is a linear function of $R$, $S = S_0 + \lambda R$

### 3.1. App is *not* a member of a suite of applications.

$$\frac{dR}{dt} = \alpha DB(1-R) - \beta\frac{dS}{dt} \tag{3}$$

$$\frac{dR}{dt} = \alpha DB(1-R) - \beta\lambda R \tag{4}$$

Here $\alpha$, $\beta$, and $\lambda$ are constants of proportionality. This differential equation can be solved using the method of partial fractions. For our purposes studying (4) directly suffices:

- $R$ has a single stable equilibrium point at
$$R_\infty = \frac{\alpha DB}{\alpha DB + \beta\lambda} < 1$$
Starting at an initial reliability $R_0$ for an initial $\alpha$-version of the application the eventual reliability of the application is predicted to be $R_\infty$.
- The larger in relative magnitude $\alpha DB$ is to $\beta\lambda$ the closer the reliability approaches 1. This is a surprising result, because it implies that Software Inc. could have incompetent programmers (large $\lambda$ for too many extra lines of code, and large $\beta$ for introducing too many new bugs), but because it has so many developers (large $D$) and/or so many $\beta$-testers (large $B$) App could eventually have superb reliability. Software Inc. could achieve just as much reliability in App by making up its deficit in quality programmers by a surfeit of quantity developers!
- The larger the number of developers and $\beta$-testers, the more positive the value of $dR/dt$, the faster the reliability of App approaches $R_\infty$.

**3.2. App *is* a member of a suite of applications.** Suppose App is a member of a suite of $N$ applications, $N > 1$. How does the interaction between the members of the suite affect the reliability of App?

Without loss of generality let us denote the $s^{th}$ member of the suite as $\mathsf{App}_s$, $s = 1 \ldots N$ with $\mathsf{App}_1$ denoting App. Let $\vec{R}$ denote the vector of reliability for the applications.

Our additional assumptions are:

- The effect of the reliability of $\mathsf{App}_i$ on the rate of growth of the reliability of $\mathsf{App}_j$ is a linear gauge coupling.

- The number of developers and $\beta$-testers depend on the application.
- The quality of the programmers for each application is *not* necessarily the same.

Let $u_i := \alpha_i D_i B_i$ and $v_i := \beta_i \lambda_i$. With the additional assumptions our equation is now a system of differential equations:

$$\frac{d}{dt}\begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_N \end{bmatrix} = \overbrace{\begin{pmatrix} u_1 & 0 & 0 & \dots \\ 0 & u_2 & 0 & \dots \\ 0 & 0 & u_3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}}^{U} \begin{bmatrix} 1-R_1 \\ 1-R_2 \\ \vdots \\ 1-R_N \end{bmatrix} - \overbrace{\begin{pmatrix} v_1 & 0 & 0 & \dots \\ 0 & v_2 & 0 & \dots \\ 0 & 0 & v_3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}}^{V} \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_N \end{bmatrix} - \underbrace{\begin{pmatrix} 0 & \gamma_{12} & \gamma_{13} & \dots \\ \gamma_{21} & 0 & \gamma_{23} & \dots \\ \gamma_{31} & \gamma_{32} & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}}_{\Gamma} \begin{bmatrix} 1-R_1 \\ 1-R_2 \\ \vdots \\ 1-R_N \end{bmatrix} \tag{5}$$

or more compactly,

$$\frac{d\vec{R}}{dt} = U(\vec{1}-\vec{R}) - V\vec{R} - \Gamma(\vec{1}-\vec{R}), \quad \vec{1} := \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{6}$$

where $\{\gamma_{ij}\}$ are nonnegative constants representing the interaction effect of $\mathsf{App}_j$ on $\mathsf{App}_i$.

As a reality condition we require that $0 \leq R_i \leq 1$ for each $i$ and for all time $t$. It easy to see that $R_i \leq 1$, since:

$$\frac{dR_i}{dt} = u_i(1-R_i) - v_i R_i - \sum_{j \neq i} \gamma_{ij}(1-R_j) \leq u_i(1-R_i) - v_i R_i$$

*Assuming* all of the $R_j$'s are less than 1, but this is certainly true initially and by a continuity argument true for $0 < t << 1$. But this implies the rate of growth for an application which is a part of suite is less than the rate of growth for an individual application whose reliability cannot grow larger than 1. Hence the reliability of a member of a suite cannot grow larger than 1 in this model.

To show that $0 \leq R_i$ for all $i$ is a little harder. We require that the matrix $U - \Gamma$ have the property that $\big((U-\Gamma)\vec{x}\big)_i > 0$ for all $i$ for all $\vec{x}$ such that $x_j > 0$ for all $j$. This can be satisfied if the sup norm of $\Gamma$ is sufficiently smaller than the sup norm of $U$ which is reasonable to expect if the interaction effect between the various members of the suite is sporadic.

To determine $\vec{R}_\infty := \lim_{t \longrightarrow \infty} \vec{R}$ it suffices to determine the equilibrium value for equation (5) which occurs when

$$(\Gamma - U - V)\vec{R} = (\Gamma - U)\vec{1} \tag{7}$$

which has the unique solution,

$$\vec{R} = (\Gamma - U - V)^{-1}(\Gamma - U)\vec{1} \tag{8}$$

if $\Gamma - U - V$ is invertible, otherwise $\vec{R}$ will trace out a periodic or pseudo-periodic path through the $N$-block $\times_{i=1}^{N}[0,1]$. From a measure theory point of view, the set of of $N$ by $N$ matrices which have determinant 0 has measure 0. Hence, with probability 1 we should expect a unique stable equilibrium to occur.

We have already determined that the rate of growth of the reliability of a member of a suite is less than the rate of growth for an individual application so we can deduce

$$R_{i\infty} \leq \frac{U_i}{U_i + V_i} = \frac{\alpha_i D_i B_i}{\alpha_i D_i B_i + \beta_i \lambda_i}$$

3.3. **Reccomendations.** From the analysis of stand alone applications vs. suite members and reliability in general, the strategies for improving reliability are

- Increase the number of developers. To be precise increase the number of patch submissions be they internal or external. Create a review committee to determine which patches are accepted, modified, or rejected to increase throughput.
- Increase the number of $\beta$-testers. That is, increase the number of reports of *distinct* bugs while keeping statistics on the *frequency* of the bugs so as to maximize diagnosis of the bugs.
- Decouple, decouple, decouple. Synergystic effects in a suite can lower the reliability of all of the member applications. *If* interaction bugs are diagnosed correctly, the reliability of the members may actually *reach* their individual limits. That's a big "if" however.

## 4. Number of software titles

Suppose Software Inc. wanted to dominate the software industry through its sheer volume of applications. Specifically, Software Inc. wants to have not just the most number of software titles, but an outright *majority* of software titles for a specific operating system, call it OS-Generic (OS-G for short). Is this feasible?

The trivial answer is yes, *if* OS-G is a *new* operating system, which Software Inc. has first access so that Software Inc. can put its software products out first. But that is a cheap answer, because it doesn't really answer the more important question for the stockholders of Software Inc., which is

> **Can Software Inc. dominate the software industry for OS-G effectively forever?**

To answer that question I will define some variables, state my differential equation for modelling this problem, and, most importantly, I will state the assumptions I am making for the differential equation.

### 4.1. The variables.

- Let $t$ denote time since the creation of OS-G in whatever units that are appropriate, be they years, months, days, etc. Time $t = 0$ is to be interpreted as the first time OS-G is publicly released for use.
- Let $P$ denote the number of software titles that Software Inc. is selling and maintaining at time $t$. (Note: I am *not* making any assumptions on the *reliability* of the software, only that $P$ denotes the number of software titles that Software Inc. is selling and maintaining at time $t$.)
- Let $N$ denote the number of developers that Software Inc. currently employs for writing its software titles. For a relatively mature company, $N$ will be relatively constant, and I will treat it as such. Certainly a start-up company selling proprietary software will *not* have as many developers as the largest software makers, and so we can assume the influence of its developers on its growth rate as smaller than for a mature powerhouse company. Without loss of generality assume Software Inc. is a powerhouse company.

### 4.2. The differential equation.

$$\frac{dP}{dt} = k\frac{P}{1+gt} - jP + hN \tag{9}$$

where $k, j, h, g > 0$ are constants of proportionality defined in my following argument.

**4.3. The arguments for the essential correctness of equation (9).** The first term on the right for equation (9) represents the snowballing effect already written software has on the creation of new software. For example, suppose Software Inc. had written a word processor application. A natural extension of that would be to write a seperate spellchecker application, or a spreadsheet, or a presentations application. Humans have been known to "reinvent the wheel". But it is also true that when an inventor has learned how to make a wheel, the inventor will often go on to inventing collateral inventions. In fact, this is how western technology functions. In the case of Software Inc., the knowledge base is kept internal. This has a direct proportional impact on how many new applications Software Inc.'s developers can create from pre-existing applications.

That explains the constant $k$ and the factor of $P$. To explain the denominator, $t+1$, I appeal to your intuition. Suppose, for example, that Software Inc. has "effectively" mapped out all of the aspects of an office suite. By effectively I mean that all of the *"interesting to programmers"* applications have been written. There is always room for improvement for more applications, but here a brake of human behavior becomes apparent. Regardless of how well treated or well paid the developers might be, if the developers have no real desire to write a pre-assigned application, or have an aversion to writing the application, the time required to write said application can lengthen considerably. Hence, as time goes by and the majority of the "interesting" applications have been written, human apathy works to slow down the rate of growth of $P$.

The second term represents a decay effect that as new software is created which fundamentally supplants older software, the older software is discarded and no longer maintained. By new software I am not referring to bug fixes or security patches. I am referring to software which has new features that the older software did not. Clearly, what proportion of legacy software is kept depends on the decisions of the management of Software Inc. The constant of proportionality is $j$.

The third term represents the original contributions of the developers. I'll use the wheel example again to exhibit this effect. Suppose one developer invents the wheel. A second developer could invent the long bow, a fundamentally unrelated invention to the wheel. Hence, it is possible to have original ideas which are unrelated to earlier ideas. Clearly the number of original developments is proportional to the number of developers.

4.4. **Qualitative predictions of the model.** I want to make some estimates for the upper and lower bounds of growth of $P$. In plain English, how does $P$ grow? Does it grow without bound, or is there some sort of upper bound for which it will not rise above? Since I can't solve (9) explicitly, I will make a lower bound estimate by comparing the solution of (9) with the solution of:

$$\frac{dP}{dt} = k\frac{P}{1+gt} - jP \tag{10}$$

Notice that for the same value of $P$ and $t$ the derivative of $P$ in this equation is smaller than the derivative of $P$ in the first equation. Thus the solution for this equation will be *less* than the solution for equation (9) *given* the same initial condition.

Using a basic separation of variables technique, we can arrive at the solution to equation (10):

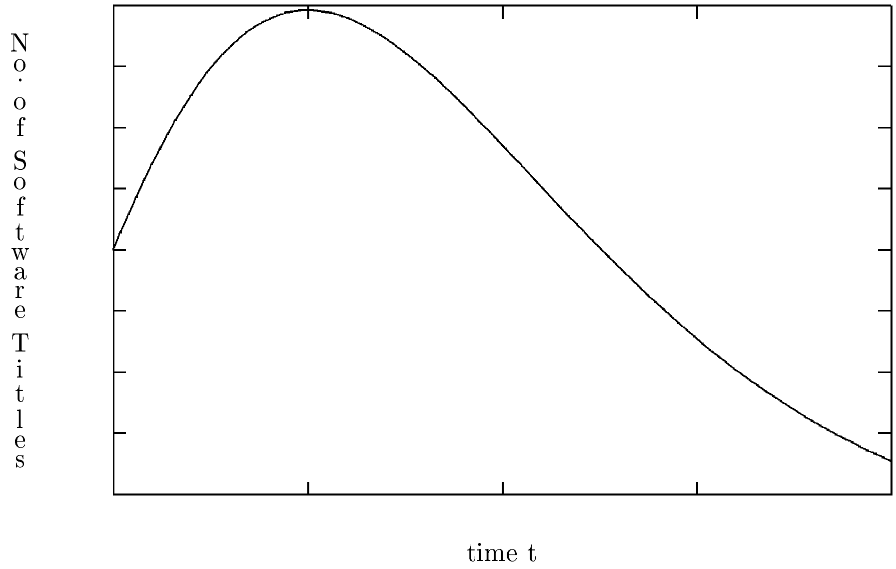
$$P = P_0(1+gt)^{k/g}e^{-jt} \tag{11}$$

where $P_0$ is the initial number of software titles that Software Inc. offers at the release of OS-G. Assuming $j$ is a small constant then we see that $P$ grows initially as a polynomial in time, reaches a max and then proceeds to decay to 0. Keep in mind this a *lower bound estimate*. This is the *worst* that we predict for Software Inc.

As a lower estimate it's worth commenting on what possible scenarios Software Inc.'s set of software titles would follow that pattern. Two scenarios present themselves:

1. Software Inc. decides to get out of the software development business and into the support business. In doing so, there is no longer any reason to

FIGURE 3. Worst Case Scenario




maintain its current software titles as they will be overtaken in functionality and reliability by other software from other companies. The support is not cut off immediately, but gradually.

2. Software Inc. switches operating systems, or equivalently, OS-G becomes obsolete and replaced by a new operating system sufficiently dissimilar as to make Software Inc.'s current list of titles unreasonably inoperable in the new operating system. Obviously, Software Inc. will switch its development efforts to the new operating system and allow its old software line to decay to death as it is no longer the main source of income.

Let us now consider the best case scenario, one in which there is no decay term in the differential equation:

$$\frac{dP}{dt} = k\frac{P}{1+gt} + hN \tag{12}$$

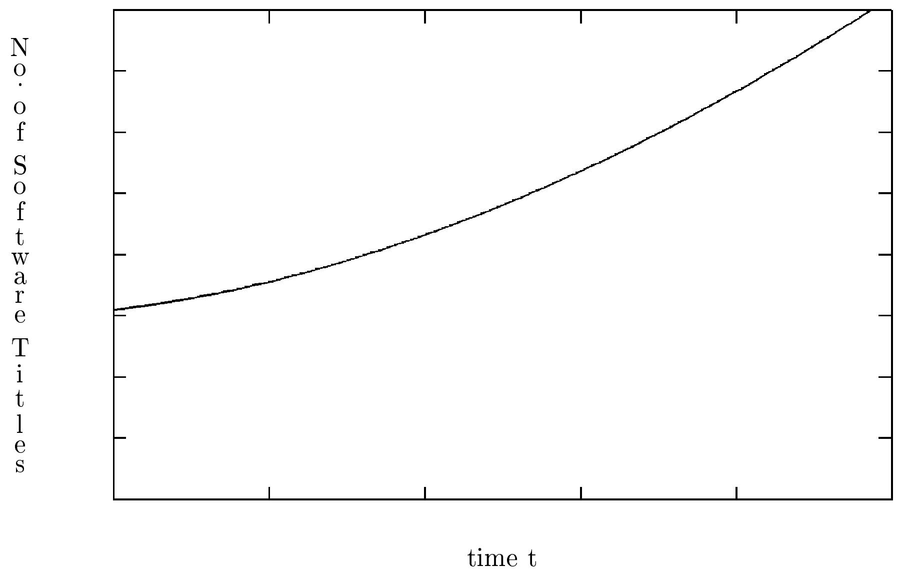
This equation unfortunately is not amenable to the separation of variables trick. An alternative method of deriving qualitative information presents itself: Suppose for some time $t_0$, one of the terms on the right was strictly greater than the other. Then by a continuity argument there would be a neighborhood of $t_0$ for which the larger of the two was always larger in that neighborhood.

Let's consider the two possible scenarios:

**$k\frac{P}{1+gt} > hN$, $t_0 \leq t \leq t_1$:** In this case we can find overestimates of the solution for (12) by finding a solution for:

$$\frac{dP}{dt} = 2k\frac{P}{1+gt} \tag{13}$$

FIGURE 4. Best Case Scenario




Here, a closed solution can be found and in fact is given by the formula:

$$P = P_0 \frac{(1+gt)^{2k/g}}{(1+gt_0)^{2k/g}} \tag{14}$$

where $P_0$ is the number of software titles at time $t_0$

**$k\frac{P}{1+gt} \leq hN$, $t_0 \leq t \leq t_1$:** As in the previous case we can find overestimates of the solution for (12) by finding a solution for

$$\frac{dP}{dt} = 2hN \tag{15}$$

The solution of this differential equation is:

$$P = P_0 - 2hNt_0 + 2hNt \tag{16}$$

Notice that in either case, $P$ is a polynomial function of $t$, which is to be expected in hindsight. After all, exponential growth is not likely to occur in an environment where there is an upper bound on how fast a developer can churn out a piece of code per unit of time.

We ask then if there is an interpretation for this best case scenario. And the answer is yes, there is an interpretation. In the best case scenario, *none* of the software titles are dropped by Software Inc. Whether or not that is a sound business practice depends on the reality of the software market. It's possible, although implausible, that every software title written and maintained by Software Inc. is so good that it is never dropped or replaced by a new software title which has more features than its predecessor.

This still begs the questions —

> What is the behavior of $P$ when all three terms on the right side of (9) are considered? Does $P$ grow without bound, initially grow then decay, or does it reach some nonzero equilibrium?

To answer those questions we need to sharpen our lower and upper estimates of $P$. Consider again equation (9). Let's rewrite it slightly:

$$\frac{dP}{dt} = P(\frac{k}{1+gt} - j) + hN \tag{17}$$

Notice that the only realistic solutions for us are solutions with $P \geq 0$. With that in mind notice that:

$$\frac{k}{1+gt} - j > -j, \qquad \lim_{t \longrightarrow +\infty} \frac{k}{1+gt} - j = -j \tag{18}$$

which implies any solution $S_0$ for (9) is greater than any solution $S_1$ of:

$$\frac{dP}{dt} = -jP + hN \tag{19}$$

with the same initial condition. Equation (19) has an exact solution given by the formula:

$$P = \frac{hN}{j} + (P_0 - \frac{hN}{j})e^{-jt} \tag{20}$$

which approaches the stable equilibrium value of $\frac{hN}{j}$ as $t$ approaches $+\infty$. So we can safely conclude that the solution of (9) does *not* decay to 0 as $t$ approaches $+\infty$.

From (18) we know that for every $0 < \epsilon < j$, there is a $T_\epsilon$ such that for all $t > T_\epsilon$:

$$-\epsilon > \frac{k}{1+gt} - j > -j \tag{21}$$

which implies $S_0$ is less than the solution $S_2$ of the equation:

$$\frac{dP}{dt} = -\epsilon P + hN \tag{22}$$

if $S_0(T_\epsilon) = S_2(T_\epsilon)$. Which tells us that if we define $P_{T_\epsilon}$ as the value of $S_0$ at time $T_\epsilon$ then the formula for $S_2$ with $S_2(T_\epsilon) = P_{T_\epsilon}$ is:

$$P = \frac{hN}{\epsilon} + (P_{T_\epsilon} - \frac{hN}{\epsilon})e^{-\epsilon(t-T_\epsilon)} \tag{23}$$

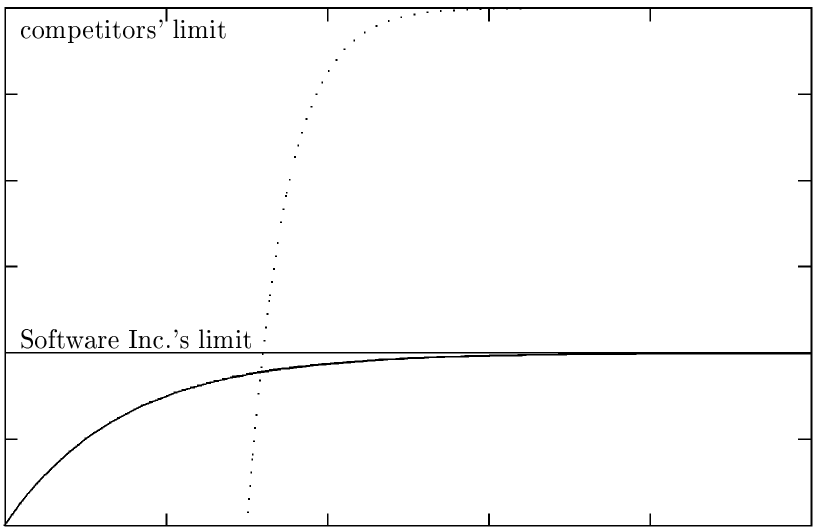
which has a stable equilibrium value of $P = \frac{hN}{\epsilon}$ as $t$ approaches $+\infty$. So here's a sharper upper bound for $P$. But notice that $\epsilon$ can be chosen arbitrarily close to $j$ and the exact solution of (9) would still have a limiting value lower than $P = \frac{hN}{\epsilon}$, i.e.:

$$\frac{hN}{j} \leq \lim_{t \longrightarrow +\infty} P \leq \frac{hN}{\epsilon} \quad \text{for all } 0 < \epsilon < j \tag{24}$$

By the sandwich lemma, $\lim_{t \longrightarrow +\infty} P = \frac{hN}{j}$.

Thus there is a stable limit for the number of software titles that Software Inc. can maintain, $P = \frac{hN}{j}$.

FIGURE 5. Software Inc. vs. its combined competitors if OS-G remains stable




4.5. **Conclusions.** If

- OS-G is a stable operating system that does *not* have periodic changes which break a significant percentage of binary executables for the previous version of OS-G;
- Software Inc. does *not* have advance knowledge of the new changes before its competitors; and
- the value of $hN/j$ is greater for the developers for all *other* companies *combined*; then

Software Inc. can *not* maintain world domination through the majority of software titles in the *long run.*

If Software Inc. had control of the continuous development of OS-G then it would be to Software Inc.'s advantage to release updates of OS-G whenever Software Inc. is in danger of losing its world domination. My recommendations for maintaining world domination through sheer number of software titles would be:

1. Break the operating system periodically with incompatible updates/new features. Software Inc. will always be able to release new software titles and revised current titles before its competitors;
2. Minimize the number of competitors who have access to the source code of OS-G. It would be insane to allow all of the competitors free reign access to the code. Once Software Inc. fell behind in its world domination, it would be difficult if not impossible to regain world domination *even* with periodic updates of OS-G.

   Software Inc.'s competitors could write their own version of OS-G which was compatible with OS-G using none of the original OS-G code. This is actually much worse than it sounds, because if a sufficient number of the competitors collaborated on a clone of OS-G, they could conceivably make a *better and cheaper* version of OS-G. Even if the clone version was not a full implementation of OS-G, it might be sufficient to wrest a majority of Software Inc.'s customers away from Software Inc. in which case Software Inc. would now be faced with the unenviable task of perpetually catching up with its competitors. To re-iterate, *Software Inc. must severely restrict the access of its source code for OS-G.*

What if Software Inc. were an open source company, and OS-G were an open source (GPL'd or equivalent) operating system? How does the above analysis apply to Software Inc. in this case? If Software Inc.'s goal were world domination as an open source company, then its strategy is actually considerably simpler.

Give freely *all* of the open source (GPL'd or equivalent) software. In effect, Software Inc. is co-opting all of the open source developers for OS-G as its own developers. If we are to believe the statements from the Halloween documents ([1],[2]) at face value then the limiting value of $hN/j$ for the open source developers is potentially one or more orders of magnitude larger than for *all* of the proprietary companies *combined*. Software Inc. has world domination without firing a shot. Of course, the problem here is that any other company can do the same, in which case all of the open source companies *each* has world domination in this manner. There is no competition in this case between open source companies, only competition between the open source community vs. the closed source community. Mathematically speaking, the open source companies represent an equivalence class consisting of all the the open source companies, while the closed source companies are distinct equivalence classes consisting of one member each.

How then does Software Inc. make its money? The obvious answer is support of course. But that still begs the question of how does Software Inc. dominate in OS-G if all of the open source companies are equivalent. The solution to this problem is what I call the "peanut butter wars".

## 5. The peanut butter wars, or can Software Inc. dominate its niche market?

Consider the peanut butter market. Let's assume for the sake of argument that all peanut butters are the same.

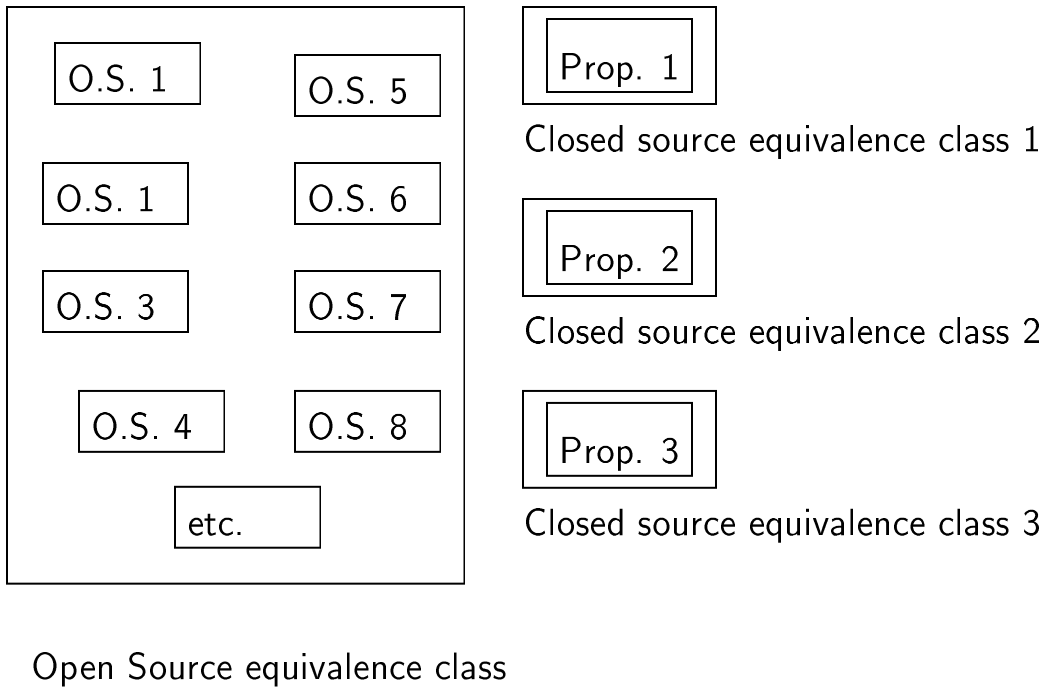
- How does one company dominate the market?
- *Can* one company dominate the market?
- Could it be a random effect, or is there a method of achieving such success?

We'll begin with a highly simplistic model of "us against them," that is, Software Inc. vs. all of the other companies which are offering support services for open source software for OS-G. In this model, I am assuming:

1. The target consumers are aware of all of the support services companies (including Software Inc.);
2. All of the companies are offering the same services for the same fee;
3. There is no brand loyalty locking customers to particular companies; and,
4. the size of the market (the number of customers if you will) is constant.

With these assumptions I can build my model in this fashion: Let $P_r$ denote the probability that Software Inc. eventually controls the entire market if it's current market share is $r$. Let $t$ denote time in some chosen unit (days, weeks, quarters,

FIGURE 6. Equivalence Classes of Companies




etc.). Let $m_t$ denote the market share (percentage of the market) controlled by Software Inc. at time $t$. Assume the value of $m_{t+1}$ is related to $m_t$ due to market fluctuations in this manner:

$$m_{t+1} = \begin{cases} m_t + 1/n & \text{with probability } p \\ m_t & \text{with probability } 1-2p \\ m_t - 1/n & \text{with probability } p \end{cases} \tag{25}$$

where $n$ is a positive constant integer, and $p$ is a fixed probability, $0 < p \leq 0.5$.

This is what is classically known as the Gambler's ruin problem. The question here is to determine the probability of complete domination by Software Inc. of the market given that it has some initial share $m_0 = r/n$ of the market.

The determination is actually quite straightforward. We need an explicitly formula for $P_{r/n}$. To do so, we require two boundary conditions:

- $P_1 = 1$. If Software Inc. already has complete domination it will continue to have complete domination.
- $P_0 = 0$. If Software Inc. has no market share at all it will continue to have no market share.

Using Equation (25) we can generate a corresponding equation for $P_r$:

$$P_r = pP_{r+1/n} + (1-2p)P_r + pP_{r-1/n}, \qquad P_0 = 0, P_1 = 1 \tag{26}$$

which has explicit solution:

$$P_r = r \tag{27}$$

In plain English, if Software Inc. initially controls $r\%$ of the market, then the probability that it will eventually dominate the entire market is $r\%$.

This prediction is unsatisfying (and more importantly unrealistic!) due to the fact that none of the assumptions we made were realistic:

- Market sizes *do* change;
- Not all companies are known to all of the market;
- It's not necessarily true that all companies provide the same service for the same fee;
- Some companies *do* have brand loyalty;
- Fluctuations in market share are not in fixed increments;
- Opposing companies do *not* necessarily operate in collusion;
- Having an initial market share at or close to 0% does not preclude a company from eventually dominating its market;
- Dominating a market is not necessarily a stable phenomenon; and,
- It makes no predictions concerning proprietary software companies.

Since there are so many factors that influence a person's decision making, I will assume a weak version of the central limit theorem applies. That is, given a range of possibles numerical values, the probability of choosing a value by an individual is roughly Gaussian. The only exception to this is the effect of the age of a company on the buying decision of a target consumer. Thus I will assume:

- Given a set of prices $\{p_i | i = 1, \ldots, n\}$ for a product, the probability that a customer is willing to pay a price $p$ is a decreasing function $\phi(p)$ with $\phi(0) = 1.0$ and $\phi(\infty) = 0.0$.
- Ranking customer satisfaction of customer support services as an index from 0 to 100 with 100 "perfect", the probability of choosing a company based on customer satisfaction $s$ is an increasing function $\theta(s)$ with $\theta(0.0) = 0.0$ and $\theta(1.0) = 1.0$
- Given the ages $\{a_i | i = 1, \ldots, n\}$ of establishment for a set of $n$ companies, the probability that a customer is willing to do business with company $t$ years old is a nondecreasing function with initial value near 0 and value 1.0 for $t = \infty$..
- The probability that a customer is willing to purchase a product based on its reliability $r$ is an increasing function $\nu(r)$ with $\nu(0.0) = 0.0$ and $\nu(1.0) = 1.0$.

The brand loyalty effect acts as a retarding effect on the rate of decrease of market share for a company. There are two forms of brand loyalty pertinent to this model:

- brand loyalty to a company because of identification with the company; and,
- brand loyalty to a company due to a large investment in software products with that company.

Finally, I am assuming that all of these aspects of decision making are independent.

Assume there are $N$ companies competing, with some entering the competition at a later time than others, i.e., their initial market share is 0%. Let $M_i$ denote percentage of target consumers who are willing to purchase the product from company number $i$. Let $b_i$ denote the percentage of consumers at time $t$ who recognize the brand of company $\#i$. $0 < fud_i \leq 1$ will denote the effectiveness of FUD campaigns of rival companies on company $\#i$. The smaller the value, the more effective the FUD effect. When there is no FUD or ineffective FUD, $fud_i = 1$. $0 \le \lambda_i \le 1$ denotes the brand loyalty effect for company $\#i$. The greater the brand loyalty for company $\#i$, the larger the value of $\lambda_i$. Let $str_i$ denote the "strength" of the application of company $\#i$. The value of $str_i$ we will assign based on the above factors is

$$str_i := \lambda_i fud_i \phi(p_i)\theta(i)\omega(a_i + t)\nu(r_i)$$

Consider now a target consumer who currently belongs to the $M_i$ part of the market. What is the probability that this consumer will "defect" to another company? The probability obviously depends on the above listed factors, but how are the factors welded to determine the probability?

To answer the question, we first partition the set $S$ of target consumers into disjoint sets:

$$S = \bigcup_{k=0}^{N} \bigcup_{\{j_1,\dots,j_k\}\subseteq\{1,\dots,N\}} S_{j_1,\dots,j_k}$$

where $S_{j_1,\dots,j_k}$ denotes the set of target consumers who know of the competing applications of companies $\#j_1,\dots,j_k$ and no others. $S_0$ denotes the set of target consumers who know of none of the applications, are are therefore denoted as the "untouchables". Assuming independence of brand recognition (knowledge of one company's application has no bearing on knowledge of other companies' applications), then the market share represented by $S_{j_1,\dots,j_k}$ is:

$$M.S._{j_1,\dots,j_k} := \prod_{l=1}^{k} b_{j_l} \prod_{i\notin\{j_1,\dots,j_k\}} (1 - b_i)$$

Note as well:

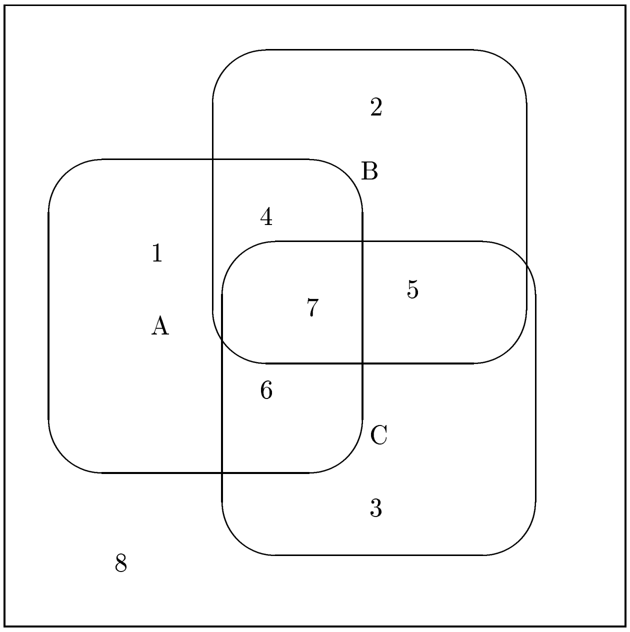
$$b_{j_1} = \sum_{k=1}^{N} \sum_{\{j_2,\dots,j_k\}\subseteq\{1,\dots,N\}} M.S._{j_1,\dots,j_k}$$

Let $M_{j_r|j_1,\dots,j_k}$ denote the market share of of target consumers currently loyal to company $\#j_r$, but who know of companies $\#j_1,\dots,j_k$ of the subset $S_{j_1,\dots,j_k}$ . A few properties follow from the notation:

1. $\sum_{l=1}^{k} M_{j_l|\{j_1,\dots,j_k\}} = 1$
2. $\sum_{k=1}^{N} \sum_{\{j_1,\dots,j_k\}\subseteq\{1,\dots,N\}} M_{j_1|j_1,\dots,j_k} M.S._{j_1,\dots,j_k} = M_i$

Property 1 states that the sum of all target consumers who know of companies $\#j_1,\dots,j_k$ and who belong to companies $\#j_1,\dots,j_k$ must be the entire set $S_{j_1,\dots,j_k}$. Property 2 states that the sum of market share of all consumers who currently belong to company $\#i$ and who may or may not be aware of other companies must be the total of current market share of company $\#i$.

By analyzing the behavior of consumers in each disjoint set $S_{j_1,\dots,j_k}$ we will be able to predict the behavior of each company's market share.




1. Know A, don't know B,C
2. Know B, don't know A,C
3. Know C, don't know A,B
4. Know A,B, don't know C
5. Know B,C, don't know A
6. Know A,C, don't know B
7. Know A,B,C.
8. Don't know A,B,or C

FIGURE 7. A sample partition with 3 companies

Consider the case of $S_{j_1,\dots,j_k}$. Suppose a target consumer belongs to the subset $S_{j_1,\dots,j_k}$. Then the quantity

$$c_{s|j_1,\dots,j_k} := \frac{str_s}{\sum_{l=1}^{k} str_{j_l}} \tag{28}$$

is the probability that the target consumer will choose company $\#s$ ($s \in \{j_1,\dots,j_k\}$) over its competitors. Note that

$$\sum_{l=1}^{k} c_{j_l|j_1,\dots,j_k} = 1$$

since the customer must choose one of the competing products.

With these probabilities determined we must also consider other aspects of the model:

- Company $\#i$ cannot have more market share $M_i$ than brand recognition $b_i$. In the case of open source companies, we treat them as an equivalence class of 1 company. Effectively, that's how open source products function.
- Market shares cannot be negative.
- Conservation of market share must hold: $\sum_{i=1}^{N} M_i(t) \equiv \text{ constant}, \quad \forall t \geq 0$

We proceed onto the model of market share. The rate of change of $M_{j_r|j_1,\dots,j_k}$ is given by:

$$\frac{dM_{j_r|j_1,\dots,j_k}}{dt} = K\sum_{l\neq r}\left(M_{j_l}c_{j_r|j_1,\dots,j_k} - M_{j_r}c_{j_l|j_1,\dots,j_k}\right), \qquad r = 1,\dots,k \tag{29}$$

$$= K\sum_{l}\left(M_{j_l}c_{j_r|j_1,\dots,j_k} - M_{j_r}c_{j_l|j_1,\dots,j_k}\right), \qquad r = 1,\dots,k \tag{30}$$

$$= K\left(1 * c_{j_r|j_1,\dots,j_k} - M_{j_r|j_1,\dots,j_k} * 1\right), \qquad r = 1,\dots,k \tag{31}$$

$$\frac{dM_{j_r|j_1,\dots,j_k}}{dt} = K\left(c_{j_r|j_1,\dots,j_k} - M_{j_r|j_1,\dots,j_k}\right), \qquad r = 1,\dots,k \tag{32}$$

where $K$ is a constant of proportionality.

Observe:

1. For all $t \geq 0$, $M_{j_r|j_1,\dots,j_k} \geq 0$, since as soon as $M_{j_r|j_1,\dots,j_k} = 0$, the rate of change for $M_{j_r|j_1,\dots,j_k}$ is positive, and $M_{j_r|j_1,\dots,j_k}$ can only increase then.
2. The model does not violate conservation of market share as can be seen:

$$\begin{aligned}
\sum_{r=1}^{k}\frac{dM_{j_r|j_1,\dots,j_k}}{dt} &= K\sum_{r=1}^{k}\left(c_{j_r|j_1,\dots,j_k} - M_{j_r}\right) \\
&= K(1-1) \\
&= 0 \\
\sum_{r=1}^{k}\frac{dM_{j_r|j_1,\dots,j_k}}{dt} &= 0 \\
\sum_{r=1}^{k} M_{j_r|j_1,\dots,j_k} &\equiv \text{constant } = 1.0
\end{aligned}$$

3. Since $M_{j_r|j_1,\dots,j_k} M.S._{j_1,\dots,j_k} \leq M.S._{j_1,\dots,j_k}$ it follows that $M_{j_r} \leq b_r$. Hence market share is never greater than brand recognition.

To determine the long term behavior of the model, we need to determine if there is an equilibrium point, or failing that, if there is a constant vector, $\vec{M}_\infty$ for which every solution $\vec{M} := (M_1,\dots,M_N)$ of (29) converges to as $t \longrightarrow \infty$. An analysis of (29) and the definition of $str_i$ reveals a time dependent component, namely the age of the company and the reliability factors. Hence, equation (29) is non-autonomous which means it probably has no equilibrium solution. This does not rule out the possibility of a constant vector $\vec{M}_\infty$ for which every solution approaches.

To determine the existence of $\vec{M}_\infty$ it suffices to observe that as $t \longrightarrow \infty$, the age factor for each company approaches the constant 1, and the reliability for each application approaches its own limiting value. This means that we can ignore the age and reliability factors. That is to say, the terms $c_{j_r|j_1,\dots,j_k}$ converge to constant terms, which I will denote as $choose_{j_r}$. Note that:

$$\sum_{l=1}^{k} choose_{j_l} = 1$$

If we replace the $c_{j_r|j_1,\dots,j_k}$ terms with $choose_{j_r}$ terms in equation (29) an equilibrium solution exists, namely

$$\frac{M_{j_r|j_1,\dots,j_k}}{dt} = K\big(choose_{j_r} - M_{j_r|j_1,\dots,j_k}\big), \quad r = 1,\dots,k \tag{33}$$

$$M_{j_r|j_1,\dots,j_k} = choose_{j_r}, \quad r = 1,\dots,k \tag{34}$$

To see that this is an attractive limiting value, it suffices to study (33) further. Notice that if $M_{j_r|j_1,\dots,j_k} > choose_{j_r}$ then the rate of change for $M_{j_r|j_1,\dots,j_k}$ is negative, while if $M_{j_r|j_1,\dots,j_k} < choose_{j_r}$ then the rate of change for $M_{j_r|j_1,\dots,j_k}$ is positive.

Lastly, I need to say a few words about the fact that equation (29) does not have constant parameters as (33) does. It is not too difficult to see that as $t$ gets larger, solutions to (29) converge to the equilibrium value for (33), since the parameters $c_{j_r|j_1,\dots,j_k}$ in (29) are converging to the parameters $choose_{j_r}$ in (33).

Thus, the eventual percentage of the market share for company $\#r$ for the set $S_{j_1,\dots,j_K}$ is $choose_{j_r}$.

In the war for market share, company $\#j_r$ can expect $choose_{j_r}$ percent of the market $S_{j_1,\dots,j_k}$ where it is competing against $k-1$ other companies. It's total market share is given by

$$\sum_{k=0}^{N-1} \sum_{S=\{j_1,\dots,j_k\}\subseteq\{1,\dots,N\}, j_r\in S} M.S._{j_1,\dots,j_k} choose_{j_r}$$

where $choose_{j_r}$ is dependent on the set $S$.

5.1. **An explicit example of the model.** Suppose two closed-source companies and a newly created open-source company are competing for domination of the office suite niche market. Let's denote the closed-source companies as company $A$ and company $B$, and the open-source company as company $C$. Assume that until the arrival of company $C$ the market share of $A$ and $B$ had reached an equilibrium.

An example table of data for the three companies is provided.

The strength of each company's application is:

TABLE 1. Sample Data

| company | brand recognition | price/ prob. | customer satisfaction | age | reliability | FUD | brand loyalty |
|---|---|---|---|---|---|---|---|
| A | 1.0 | \$200/(0.5) | .60 | 10 | 0.80 | 1.0 | 0.6 |
| B | .30 | \$50/(0.8) | .80 | 5 | 0.90 | 0.7 | 0.8 |
| C | .80 | \$0/(1.0) | .80 | 5 | 0.95 | 0.95 | 0.8 |

$$str_A := 0.5 * 0.6 * 0.8 * 1.0 * 0.6 = 0.144$$
$$str_B := 0.8 * 0.8 * 0.9 * 0.7 * 0.8 = 0.32256$$
$$str_C := 1.0 * 0.8 * 0.95 * 0.95 * 0.8 = 0.5776$$

Before the announcement of company $C$'s product, let's determine what the market share for company $A$ and $B$ had been. To do so, there are 4 subsets of the target consumers to determine:

1. $S_0$ — the untouchables. They know neither company $A$ or $B$.

2. $S_{A\cap B^c}$— they know company $A$, but not $B$.
3. $S_{B\cap A^c}$— they know company $B$, but not $A$.
4. $S_{A\cap B}$— they know both companies.

Since company $A$'s application is known to all, the market share represented by $S_0$ and $S_{B\cap A^c}$ are both zero. The market share $M.S._{A\cap B^c}$ of $S_{A\cap B^c}$ is

$$1.0 * (1 - 0.30) = 0.7$$

Similarly, the market share $M.S._{A\cap B}$ of $S_{A\cap B}$ is

$$1.0 * 0.30 = 0.3$$

In the set $S_{A\cap B^c}$ company $A$ has complete domination. Hence company $A$'s market share is at least 70%. In the set $S_{A\cap B}$ company $A$'s percentage is:

$$choose_{A|A,B} = \frac{0.144}{0.144 + 0.32256} = 0.309$$

So company $A$ has captured an additional

$$.309 * 0.3 = 0.0927 = 9.27\%$$

of the total market. Hence company $A$'s total market was 79.27% before the introduction of company $C$'s product. Also, the total market for company $B$ had been 20.73%.

After the introduction of company $C$'s product a shift of market share occurs. Eventually the market settles into a new equilibrium.. The total market is now partitioned into 8 subsets of which only four are nonempty:

1. $S_{A\cap B^c\cap C^c}$ — the set of people who know only company $A$'s product.
2. $S_{A\cap B\cap C^c}$ — the set of people who know only company $A$'s and company $B$'s products.
3. $S_{A\cap B^c\cap C}$ — the set of people who know only company $A$'s and company $C$'s products.
4. $S_{A\cap B\cap C}$ — the set of people who know all three products.

The market share for each subset is:

| subset | market share |
|---|---|
| $S_{A\cap B^c\cap C^c}$ | $1.0 * 0.7 * 0.2 = 0.14$ |
| $S_{A\cap B\cap C^c}$ | $1.0 * 0.3 * 0.2 = 0.06$ |
| $S_{A\cap B^c\cap C}$ | $1.0 * 0.7 * 0.8 = 0.56$ |
| $S_{A\cap B\cap C}$ | $1.0 * 0.3 * 0.8 = 0.24$ |

Within each subset, market share is broken down into:

| subset | company $A$ | company $B$ | company $C$ |
|---|---|---|---|
| $S_{A\cap B^c\cap C^c}$ | 1.0 | 0 | 0 |
| $S_{A\cap B\cap C^c}$ | 0.309 | 0.691 | 0 |
| $S_{A\cap B^c\cap C}$ | 0.1996 | 0 | 0.8004 |
| $S_{A\cap B\cap C}$ | 0.1379 | 0.3089 | 0.5532 |

The total market share break down is now:

| company $A$ | company $B$ | company $C$ |
|---|---|---|
| 30.34% | 11.56% | 58.1% |

Since the value of $choose_{j_r}$ determines the likelihood of defection from company $\#j_i$ to company $\#j_r$, an obvious strategy for a company to achieve market domination is to minimize the probability of defection from its product to any of the other competitor's products, and if possible, to maximize the probability of defection from competitors' products to its own. Considering how $choose_{j_r}$ is determined by price, customer satisfaction, company age, reliability, fud, and brand loyalty there are a few surprising and not so surprising results here:

5.2. **Not so surprising results:**

- Poor customer satisfaction can decrease market share.
- Large name recognition can help increase market share.
- Large brand loyalty can help stem the rate of loss of market share.

5.3. **Surprising results:**

- If the time scale for reliability is considerably shorter than the time scale for the competition for customers then reliability of the software has *no significant* bearing on market share. Presumably, a company's developers can increase the reliability of the application up to a reasonable percentage near 100%. Putting it another way, a company's product can have the worst reliability consistently, but as long as its reliability improves fast enough in relation to the expected lifetime of the product, the reliability is of relatively minor consequence.
- Ineffective FUD by a company is free advertising for its competitors which boosts the market share of its competitors.
- The age of a company is irrelevant in the long term so long as the company continues to *exist*. It's only at the beginning of its existence that a penalty in market share is paid.
- Prices of the application within a small range have no bearing on market share, i.e., if company $\#j$'s price $p_j$ is within $\delta$ of company $\#k$'s price $p_k$, there is no advantage to company $\#j$ if $p_j < p_k$ or $p_j > p_k$. It's only if the price differential is significant that the prices of applications come into play.
- The only important judges of success are the target consumers. So long as the target consumers have or can be made to have access to competing products in order to compare, the product which has the largest value of strength will have the largest percentage of market share. This is predicated on each product having 100% brand recognition.

5.4. **Strategies for Software Inc.'s domination of its niche.**

**Software Inc. is a closed source company:**

- Ensure high customer satisfaction.
- Maximize brand recognition of product while minimizing advertising access to the target consumers for competitors.
- Use FUD attacks against competitor $\#j$ *only* when the result is to lower the the probability of defection from Software Inc. to company $\#j$. A rule of thumb:

  Attack competitors only when they already have large brand recognition so FUD attacks contributions to brand recognition are negligible.

- Release the product early whether or not its ready *assuming* the reliability of the product can be rapidly improved.
- Choose native proprietary protocols and formats for the application. Offer the ability to translate competitors' protocols and formats to the application's, but *not* in the reverse direction. This has the effect of reinforcing customer lock-in to Software Inc.'s software while *reducing* brand loyalty to the competitor's applications.

**Software Inc. is an open source company:** Certainly ensuring high customer satisfaction and early product releases are obvious strategies. The FUD and customer lock-in tactics however run counter to the open source philosophy. Software Inc. should play into its strength — the open source phenomenon. Consider the following question:

> Which is better: to be a little fish in a large ocean or to be a large fish in a small pond?

The answer to this question is highly relevant to Software Inc.'s marketing strategy. As an example, consider the Linux (tm) operating system. Which is better: to have a market share of 50% of 14 million customers, or a 10% market share of 300 million customers? The numbers speak for themselves.

By co-opting the other open source companies, however small they may be, in brand recognition, Software Inc. can increase the *effective brand recognition of its product*, even if it fails to significantly increase the recognition of its own name. The goal of this strategy is to maximize brand recognition for the product, *not* the company.

Reliability concerns for the application should be minimal, since the product will have access to millions of potential $\beta$-testers and developers. This point should be repeatedly *emphasized* with the target consumers.

Software Inc. should offer the ability of translating closed source protocols and formats to open protocols and formats. By adhering to an open protocol and format Software Inc. can assure the target consumers they need not depend on any one company for its application. This has the effect of increasing brand loyalty to the product and to a smaller extent to Software Inc.

If the open source community's belief that open source software is at least as good if not better than proprietary software is correct, then it is vitally important to Software Inc. to make the largest percentage of target consumers aware of its product. In the final analysis, the only opinions that truly matter are the opinions of the target consumers. If target consumers have the opportunity to compare open-source software to closed-source software and if open-source software is truly better then Software Inc. 's product will win eventually with the largest market share. The key is to make it possible for target consumers to compare.

An obvious method is to post so-called "screen shots" of open-source applications on the internet, on the theory that "a picture is worth a thousand words". On the other hand, it is also said, "actions speak louder than words". Since most purchases of software are not made through the internet, but rather through software outlets, it would behoove Software Inc. to seek access to such outlets for advertising its software. By displaying its software for trial usage for the target consumers Software Inc. can ensure its software is given a fair appraisal of its quality.

## 6. The OS WARS

In section 5 we analyzed the behavior of competing products on one operating system. Let us now investigate the market share of operating systems as software products. The previous analysis stills applies, but there is one additional factor to consider: the number of software titles per operating system.

Suppose Software Inc. has the most technically superior operating system by whatever metric we choose to use. If there are no applications for its operating system that are *perceived* as desirable or necessary to the target consumers, then Software Inc.'s operating system will fail to achieve market share dominance. Experimental evidence in the history of computing suggests that technical superiority is an insufficient factor by itself to achieve market dominance.

Let us now assume that there are $N$ companies competing for supremacy in the operating system market. Without loss of generality, Software Inc. will be denoted as company #1. We'll denote the operating system for company $\#i$ as OS-G$_i$. Let $A_i$ denote the number of applications for OS-G$_i$. Since the number of applications is dependent on the number of developers, let $D_i$ denote the number of developers writing for OS-G$_i$. Recall from section 4 that the maximal number of stable software titles $A_i$ for OS-G$_i$ is a linear multiple of $D_i$,

$$A_i = \rho D_i \tag{35}$$

Let $D_{max}$ denote the maximum number of possible developers for the operating systems combined.

For the revised model of market share for OS-G$_i$ we have to redefine $str_i$ as

$$str'_i := A_i \lambda_i fud_i \phi(p_i)\theta(i)\omega(a_i + t)\nu(r_i) \tag{36}$$

And the model for market share is still the same, with $c_{j_l|j_1,\dots,j_k}$ defined in terms of $str'_{j_1}, \dots, str'_{j_k}$:

$$\frac{dM_{j_r|j_1,\dots,j_k}}{dt} = K \sum_{l \neq r} \left( M_{j_l} c_{j_r|j_1,\dots,j_k} - M_{j_r} c_{j_l|j_1,\dots,j_k} \right), \qquad r = 1, \dots, k \tag{37}$$

The strategy for Software Inc. is clear — promote the development of as many software titles as possible for its operating system. The more the value of $A_1$ can be increased, the larger the value of $stri'_1$ will be, and the more the company will dominate the $b_1$ percentage of target consumers. By promoting brand recognition $b_1$ in tandem with maximizing $A_1$, Software Inc. can achieve market domination effectively forever if the value of $str_1$ is larger than the combined sum, $\sum_{i=2}^{N} str_i$..

But here we have the chicken and the egg dilemma. $A_i$ is not a constant. It depends on the number of developers $D_i$ for OS-G$_i$. But $D_i$ depends on the market share $M_i$. For whatever reason, be it ego boost, fame, or profit, from a developer's point of view, the more target consumers there are for OS-G$_i$ the more likely a developer would be willing to write applications for OS-G$_i$.

If Software Inc. already has a majority of the target consumers, then Software Inc. need do nothing. It has already won, and the system is stable *so long as its competitors do not change strategies.*

By change strategies I mean for Software Inc.'s competitors to find an effective method of altering the value of the $str_i'$'s such that

$$str_1 < \sum_{i=2}^{N} str_i.$$

For the rest of this section I will assume either that Software Inc. does not yet have a majority market share, or it is in danger of losing its majority market share due to its competitors finding more effective strategies.

The question now is

How does Software Inc. maximize the number of its developers?

If OS-G is proprietary, Software Inc. cannot publish the source for its operating system lest it lose control of OS-G. Instead, Software Inc. could provide so-called developer kits necessary to create applications for its operating system.

This leads to a set of questions that Software Inc. needs to answer:

1. What price should Software Inc. charge for its developer kits?
2. How can Software Inc. *persuade* the maximum number of developers to write applications for its operating system?
3. If Software Inc. chooses to dominate the software titles for its own operating system, what effect does this have on the number of developers for its operating system?

For my answers to these question I will make a model of $D_i$ as a function of

- price $\mathcal{P}_i$ of the developer kits,
- $M_i$,
- predatory behavior of company $\#i$ on developers for its operating system,
- rewards/incentives of company $\#i$ gives its developers.

I will assume that these four factors independently affect the decision of developers as to which operating system they will choose to write applications. I am assuming that the effect of $\mathcal{P}_i$ is a monotonically decreasing function $f(\mathcal{P})$ which represents the percentage of developers willing to program for OS-G$_i$. $f(\mathcal{P})$ has the properties

$$f(0) = 1, \quad \lim_{\mathcal{P} \longrightarrow \infty} f(\mathcal{P}) = 0$$

The simplest example that comes to mind is an exponential function,

$$f(\mathcal{P}) := e^{-\lambda \mathcal{P}}$$

For a more realistic model, a survey should be done to determine the actual shape of $f$.

Let $0 < \alpha_i \leq 1$ denote the effect of predatory tactics of company $\#i$ on the pool of developers. The greater the likelihood of company $\#i$ competing with its developers, the smaller the value of $\alpha_i$. Similarly, the lesser the likelihood that company $\#i$ will compete with developers, the closer the value of $\alpha_i$ will be to 1.

In this regard open source operating systems, whose licenses are either GPL or BSD-style, have an advantage over proprietary operating systems. Developers on such open source operating systems need never fear that the maintainers of the operating systems will ever attempt to subvert the developers' projects or absorb the projects' market share since the work of writing applications and operating system maintenance have been split into two groups. The first group consists of those developers who write applications for the operating system. The *much smaller* second group consists of those developers whose goal is to improve the performance of the operating system. Individual members of the second group may or may not work on other projects, but the sheer size of the noosphere ([4]) guarantees that the probability of the second group competing with developers for application market share dominance is vanishingly small. Accordingly, for such open source operating systems the value of $\alpha_i$ is 1.

The same analysis does not necessarily apply to proprietary operating systems. If a proprietary operating system belongs to one company whose goal is the generation of profit based on the sale of binary executables of its software then developers have a cause for concern.

Consider the following scenario: A small startup company creates a so-called "killer app" which has been written for a proprietary operating system OS-G owned by Software Inc. Suppose the potential market size for this application is huge, i.e., a significant percentage of the target consumers would use this application if they knew of its existence. Let us consider a possible unfolding of events:

1. The startup company announces and begins the sale of its "killer app" for OS-G.
2. Software Inc. becomes aware of the "killer app" and observes its rapid ascent in popularity. A few notions are considered:
   (a) Publicize the existence of the "killer app" as a means of generating positive advertising for OS-G and thereby increase market share for OS-G.
   (b) Acquire the startup company in order to assimilate the profits of the "killer app".
   (c) Introduce a competing application which performs better in OS-G especially when newer releases of OS-G are made available, and thereby *achieve* greater market share than the killer app. If successful, the market share of the "killer app" is eventually reduced to 0.
3. 
   - If the potential market size for the "killer app" is too small, for example a specialized application for visualizing medical xray scans, then it is in Software Inc.'s best interests to simply promote the product and increase market share of OS-G.
   - If the expected lifetime of the "killer app" is relatively long, the required in-house developmental effort for Software Inc. too large, and the potential market share is modestly large then it is in Software Inc.'s best interests to simply acquire the startup company at the lowest possible price.
   - If the expected lifetime of the "killer app" is relatively long, the potential market share is huge, the startup company refuses to be purchased, and the required in-house developmental effort for Software Inc. can be sustained, then it is in Software Inc.'s best interests to compete directly with the startup company. Software Inc. has the distinct advantage that it can always break its own operating system with updates and recover, but its competitors may not be able to do so.

At first glance the first two of the possible responses by Software Inc. look favorable to developers. In either case the owners of the startup company are looking at a fair profit. Of course, the third response is unpleasant to the startup company. But a further analysis leads to some unpleasant possibilities.

Let us consider the first possible response again. If the market size is small by the nature of the application then Software Inc. should not compete. Software Inc. should advertise the "killer app" as a means of boosting OS-G's market share. However, what if the management of Software Inc. does not understand this point? As an exercise for the reader try to recall if the owners of proprietary companies advertise the specialized products of other vendors for their own operating systems. Try to recall the last time this occurred. As a test I suggest visiting the web sites of various proprietary operating systems and determining for yourselves whether or not the proprietary companies understand the point of free advertising of specialized software. The same investigation should be performed for the various distributors of open source operating systems. As a developer, if you are targeting a specialized market this exercise is relevant to the future success of your product. I urge you to perform the above exercise.

Let us consider the second possibility. In this scenario the "killer app" is sufficiently good and has a sufficiently large enough market size that Software Inc. is willing to purchase the company. Let's consider some numbers. Suppose the startup company consists of four individuals each of which has an equal stake in the company. Suppose Software Inc. offered $4 million for the company. Are the four individuals very happy millionaires?

It is not necessarily true. A simplistic consideration is the matter of taxes to consider. How much of the money is taken by the federal government? Consider the nature of the payment. Is it in cash, or in stock of Software Inc.? So long as the stock price of Software Inc. is rising, it might be to the advantage of the four individuals to take their payment in stock. Making the wrong choice could cause some frustration for the four individuals.

However, the truly important consideration which I have deliberately ignored until now (as could Software Inc.) is the future value of the "killer app". Consider a historical example of the future value concept, the purchasing of Manhattan island by the Dutch from the local Indians in the 17th century. What was the price agreed upon? A few trinkets was the price. What is the land value today of Manhattan? The disparity in value is not a trivial case of inflation. A profoundly poor purchase price was agreed upon, because the Indians did not have access to information about the potential future land value of Manhattan. Perhaps readers can cite examples of poor purchasing agreements in the software business as well.

Let us revisit again the purchase of the startup company by Software Inc. Suppose Software Inc. has considerable monetary assets. Suppose the future value of the "killer app" is $800 million dollars (400 million users at $20 profit per user.) What purchase price might Software Inc. offer? Probably not $800 million, nor any price in that range lest the startup company possibly become aware of its true value. Instead, Software Inc. might offer a "generous" (relative to contemporary annual salaries for programmers) purchase price in at most the low millions range. If the startup company capitulates, four individuals walk away with a face saving "generous" offer. The alternative is to compete for as long as possible before Software Inc. presumably shuts them out. The question is how much profit can the startup company generate before becoming defunct? The possibility of surrender or extinction should be unpalatable to a developer. Why settle for $1 million when in a fair world a developer might earn $200 million?

An argument can be made that a startup company would face the same dilemma in an open source operating system scenario were it to be targeted by a software company with deep financial pockets. But that is competition between application makers, not between an application maker and the power behind the operating system. The application developers cannot be threatened with incompatible upgrades in the operating system for which only the select few developers have access in a timely fashion.

Let $\beta_i \geq 1$ denote the rewards/incentives factor of developing software in OS-$G_i$. The greater the reward/incentive the larger the value of $\beta_i$ will be. We may ask what precisely can company $\#i$ offer to a developer as an incentive to develop software for OS-$G_i$. A monetary incentive is an obvious choice. In theory, company $\#i$ could pay developers for writing applications for OS-$G_i$. However, this is an unrealistic expectation, as it is not profitable in general for company $\#i$ to do so. Consider the questions company $\#i$ would have to answer:

- Which developers should be paid?
- How much should developers be paid?
- Should the payments be identical, or should some sort of metric be used to determine the payments?
- What is the likelihood that a given application written by a developer will not only flourish, but reflect well on OS-$G_i$?

These are non-trivial questions to answer. I believe that this approach is not the correct approach.

A better question harkens back to the motivations of developers:

What does a developer want?

The correct answer to this question leads to the correct approach of rewarding developers. The simplest answer to this question is an audience. What a developer seeks is an audience as large as possible which will reward the developer with whatever form of reward the developer seeks: fame, ego-boost, profit, etc.

What then should the natural choice of reward be by company $\#i$? The answer is obvious: Advertise the products of its developers. Create a forum in which the developers can hawk their wares so to speak. Metaphorically speaking, what company $\#i$ should be providing is a *bazaar*, in which the target users of OS-$G_i$, can shop/browse through the various goods offered by the developers.

A developer might prefer to be the sole "merchant" of a particular type of application in the bazaar, but having a free stall provided by the bazaar owner (company $\#i$) should be a reasonable exchange for having to compete with other developers in the same application category. The developer should have just as large an audience as any other developer of its application category, while the target users have access to the maximal information of the competitors for their business.

In real-world terms, what form should the bazaar take? Advertising in newspapers or TV is either too prohibitive in cost, or impractical. Consider how long in duration a commercial would have to be on TV in which all of the applications available for OS-$G_i$ are displayed. Consider how many pages in a newspaper would be required just to list all of the applications.

Does there exist a possible medium in which company $\#i$ can list all of the products available to OS-$G_i$? The answer is yes, of course. The simplest solution is to advertise the list of applications as a link from the home web page of company $\#i$.

Here we see a fundamental difference in the behavior of proprietary companies versus open source companies. An open source company $\#i$ is not selling its operating system OS-G$_i$. What it is really selling is support for its operating system. Such a company has no conflict of interest with developers for OS-G$_i$. An open source company can afford to create a bazaar for all the developers for its operating system. Hence $\beta_i$ could conceivably be much larger than 1.

It might not be possible for proprietary companies which choose to compete with its developers to have a large value of $\beta_i$. A company $\#i$ which gave free advertising for its major competitors might lose its dominance in some highly profitable market niche. A safer course for company $\#i$ could be to provide a forum for strictly small scale developers whose products do not directly compete with company $\#i$'s products.

An alternative reward/incentive from a proprietary company $\#i$ might be to permit limited nondisclosure agreements with select developers, such that these few developers might see the code for OS-G$_i$, and hence have a tactical advantage over their competitors. This strategy could possibly be a two-sided sword for developers:

1. The select developers could become entirely dependent on company $\#i$ for the periodic privelege to view the code for OS-G$_i$. If company $\#i$ has a reciprocal agreement with the right to view the developers' code, then there is a possibility that company $\#i$ could choose to betray the select developers and create their own competing applications. This is a classic prisoner's dilemma:

   > Two prisoners are being questioned separately for a capital crime. If both prisoners remain quiet, they both will be sentenced to a relatively light punishment. If one remains quiet and the other betrays him, the quiet prisoner will receive a long prison sentence, while the betrayer will go free with a reward for the betrayal. If both prisoners betray each other, then both prisoners receive the death penalty.

   In terms of software competition the developers do not have the same leverage as the owner of the operation system. If the developers and company $\#i$ cooperate each might receive a modest improvement in market domination. If the developers choose to betray company $\#i$, then they will cease to have access to the source code for OS-G$_i$. Consequently they might lose whatever advantage they had. If company $\#i$ chooses to betray the developers, the company may have a good chance of winning any possible consequent lawsuits arising from possible breach of contracts.

   Mathematically speaking, the strategy of betrayal is in company $\#i$'s favor since it controls OS-G$_i$. The developers have relatively little to gain by betrayal. Company $\#i$ might betray a developer when:
   - The market for the developer's applications is sufficiently large to warrant acquiring the market;
   - The expected value of the breach of contract lawsuits is less than the pay out of market domination.
   - More than one developer is creating an application for a given software niche. It might be in Company $\#i$'s interests to levy the largest licensing fees the developers could afford to pay. Company $\#i$ has nothing to lose by having a developer cease development of its application if another developer will continue to create an analogous product.

A software company should become concerned precisely when its product rises in popularity dramatically, or another company has similar access to the source code of the operating system and is developing a competing product.
2. The ill will generated by deciding which developers will become the "select" developers may undermine the reward/incentive process.

From these observations I argue that $\beta_i$ could be relatively close to 1 for a proprietary operating system.

For the sake of clarity I will define:

$$\epsilon_i := \frac{M_i f(\mathcal{P}_i)\alpha_i\beta_i}{\sum_{j=1}^N M_j f(\mathcal{P}_j)\alpha_j\beta_j}$$

The model for the pair (market share, developer share) is:

$$\frac{dD_i}{dt} = k\sum_{j\neq i}\big(\epsilon_i D_j - \epsilon_j D_i\big) \tag{38}$$

$$= k\sum_{j}\big(\epsilon_i D_j - \epsilon_j D_i\big) \tag{39}$$

$$= k(\epsilon_i * 1.0 - 1.0 * D_i) \tag{40}$$

$$\frac{dD_i}{dt} = k\big(\epsilon_i - D_i\big) \tag{41}$$

and

$$\frac{dM_{j_r|j_1,\dots,j_k}}{dt} = K\big(c_{j_r|j_1,\dots,j_k} - M_{j_r|j_1,\dots,j_k}\big) \tag{42}$$

where $k, K > 0$ are constants. Notice that the set of equations are coupled, i.e., $M_{j_r|j_1,\dots,j_k}$ affects $D_{j_r}$ and vice-versa. Equilibria exist, but unfortunately are not unique in general. To investigate the location of equlibria, I have provided the source code for two programs to predict the long term behavior of market share using Euler's method. Normally Euler's method is rather poor at extrapolating long term behavior, but since there is a long term attractive fixed point, this is not a problem.

### 6.1. Example scenarios.

**Two proprietary companies:**

I used a step-size of 0.1 with 500 steps to compute the predicted market equilibriums. In spite of the fact that company 1 has a superior product and a majority initial market, its market share dwindled to a small minority share. How could this have occurred? Company 2 made up its deficit in quality of product by encouraging developers to write applications for OS-$G_2$. This actually worked in spite of the fact that company 2 competes more with its developers than does company 1. One might think that increasing brand recognition for company 1 would improve the chance for company 1 to maintain its dominance. Unfortunately, this is not so as the predictions of the model demonstrates.

We should also observe that initial developer share has no bearing on the final outcome so long as each operating system has a nonzero initial market share, and a nonzero initial developer share. These two criteria are reasonable since the creators of an operating system are trivially developers and users of

TABLE 2. Characteristics of two operating systems

| attribute | company 1 | company 2 |
|---|---|---|
| price of operating system | 0.8 | 0.75 |
| customer satisfaction | 0.8 | 0.5 |
| reliability | 0.9 | 0.8 |
| FUD | 0.5 | 1.0 |
| brand loyalty | 0.9 | 0.7 |
| strength | 0.2592 | 0.21 |
| dev. kit. price | 0.6 | 0.8 |
| predation | 0.9 | 0.8 |
| reward | 1.0 | 1.5 |
| developer strength | .54 | 0.96 |
| brand recognition | 0.6 | 0.9 |

TABLE 3. Initial market share

| Subset | company 1 | company 2 |
|---|---|---|
| 1 and not 2 | 1.0 | 0.0 |
| 2 and not 1 | 0.0 | 1.0 |
| 1 and 2 | 1.0 | 0.0 |
| total market share | 0.60 | 0.40 |

TABLE 4. equilibrium market share

| company 1 | company 2 |
|---|---|
| 10% | 86% |

TABLE 5. equilibrium state with 90% brand recognition for company 1

| company 1 | company 2 |
|---|---|
| 23% | 76% |

the operating system. The only effect of initial developer share is to either delay or accelerate the inevitable journey to the equilibrium point.

From this example we can deduce

- Technological superiority alone is insufficient to achieve market dominance.
- High brand recognition and technological superiority of a competitor can be overcome by rewards/incentives to developers if one's company has high brand recognition.

TABLE 6. equilibrium state with 100% brand recognition for company 1

| company 1 | company 2 |
|---|---|
| 34% | 66% |

- A company which has a superior operating system, but a poor reward/incentive program for developers in comparison to other operating systems, should consider
  - switching its set of target consumers to another set not targeted by its competitors;
  - increasing brand recognition of its operating system;
  - reducing brand recognition of its competitors' operating systems;
  - improving its reward/incentive program for developers; and
  - using FUD tactics to reduce operating system strength of its competitors.

**Two proprietary companies, one open source equivalence class:** As in the previous scenario I used a step-size of 0.1 with 500 steps to compute the predicted market equilibriums. Based on the attributes listed for the three operating systems, the reader might have guessed that since open source was both superior in technology and in developer strength that open source would have dominated its 20% of the target consumers. Indeed, as the graph shows, the market share for open source began climbing, when suddenly its market share began to decrease to zero. How is it possible that open source could become defunct?

In the operating system domain open source simply had no equal in overall strength, but its strength in the developer domain was only marginally stronger than company 2. Coupled with the fact that company 2 has 100% brand recognition while open source has only limited brand recognition, it soon becomes clear to the developers where the reward is to be found — in OS-$G_2$.

If open source developer strength were doubled, by increasing $\beta_3$ from 1.0 to 2.0, then the outlook is more positive. After the initial hump, the market share of open source stabilizes, while the market share of company 1 plummets to 0, and the market of company 2 is over 90%.

Suppose we change brand recognition for open source to 90% and $\beta_3$ to 1.0, i.e. no rewards/incentives for developers. The outcomes becomes very different. Company 1's operating system becomes defunct, but now company 2's operating system decreases in market share, while open source becomes the overwhelming dominant force. The difference of 0.04 in developer strength and the high brand recognition, and technical superiority was sufficient to overcome the proprietary companies. The lessons we can learn from this scenario are:

- Technical superiority with slightly better developer strength is insufficient to maintain a stable share of the market if brand recognition is too low.
- If the predictions provided are qualitatively correct, then proprietary operating systems are *doomed*, unless the closed source companies can
  - minimize knowledge of open source operating systems; or

TABLE 7. Characteristics of three operating systems

| attribute | company 1 | company 2 | open source |
|---|---|---|---|
| price of operating system | 0.8 | 0.75 | 1.0 |
| customer satisfaction | 0.8 | 0.5 | 0.95 |
| reliability | 0.9 | 0.8 | 0.95 |
| FUD | 0.5 | 1.0 | 0.8 |
| brand loyalty | 0.9 | 0.7 | 0.95 |
| strength | 0.2592 | 0.21 | 0.6859 |
| dev. kit. price | 0.6 | 0.8 | 1.0 |
| predation | 0.9 | 0.8 | 1.0 |
| reward | 1.0 | 1.5 | 1.0 |
| developer strength | .54 | 0.96 | 1.0 |
| brand recognition | 0.6 | 1.0 | 0.2 |

TABLE 8. Initial market share

| Subset | company 1 | company 2 | open source |
|---|---|---|---|
| 1 only | 1.0 | 0.0 | 0.0 |
| 2 only | 0.0 | 1.0 | 0.0 |
| 1 and 2 | 1.0 | 0.0 | 0.0 |
| 3 only | 0.0 | 0.0 | 1.0 |
| 1 and 3 | 0.1 | 0.0 | 0.9 |
| 2 and 3 | 0.0 | 0.9 | 0.1 |
| 1 and 2 and 3 | 0.3 | 0.6 | 0.1 |
| total market share | 0.516 | 0.464 | 0.02 |

TABLE 9. equilibrium state with 20% brand recognition for open source

| company 1 | company 2 | open source |
|---|---|---|
| 0% | 100% | 0% |

TABLE 10. equilibrium state with 20% brand recognition for open source, $\beta_3 = 2.0$

| company 1 | company 2 | open source |
|---|---|---|
| 0% | 94% | 6% |

TABLE 11. equilibrium state with 90% brand recognition for open source

| company 1 | company 2 | open source |
|---|---|---|
| 0% | 14% | 86% |

- change their target audience to a set of consumers who neither want nor need the benefits of an open source operating system.

## 7. Conclusions

It is possible for a closed source company Software Inc. and its operating system to dominate the software industry indefinitely. Some of the possible strategies are:

1. Limit brand recognition of open source competitors to a minority of target consumers. At all legal costs, closed source operating systems should not allow consumers to discover alternative models of service;
2. Break the operating system OS-G periodically;
3. Do not reveal the source code;
4. Reward developers for writing code for OS-G;
5. Lower barriers to development for OS-G. For maximal effect give freely developer kits for OS-G;
6. Maximize brand recognition;
7. Maximize customer satisfaction;
8. Maximize brand loyalty;
9. Choose prices that are not too high for its applications relative to its closed source competitors;
10. Release products early whether the products are ready or not;
11. Improve reliability quickly enough in the life spans of its products;
12. Use FUD tactics only when brand recognition is already high for its competitors; and
13. Offer 1-way translation services from its competitors' applications to its native formats/protocols.

Conversely, the open source equivalence class can dominate the market if it can

1. increase brand recognition of its operating systems and applications to a majority if not the totality of target consumers. Specifically, a volunteer network should be created to install at no cost judicious samples of open source operating systems and relevant software at major retail outlets of computer hardware to give consumers an opportunity to see open source operating systems at work;
2. create an electronic bazaar for developers to advertise/demonstrate their applications in order to encourage developers to migrate to the open source operating systems. The bazaar should be sufficiently hyperlinked throughout the webspace of open source developers,distributors, and vendors to promote the perception of a giant monolithic institution offering a huge range of choices and the freedom for consumers to evaluate applications;
3. effectively counter FUD attacks of competing operating system companies.

With apologies and no offense intended to farmers in the U.S. and around the world, the adage that in my opinion sums up the closed source operating systems companies' quandary is:

*How are you going to keep them on the farm once they've seen Paris?*

The adage that sums up the open source model is

Seeing *is* believing.

A curious statement made by some closed source proponents and some open source proponents is that open source operating systems are not "ready for the desktop." Whether or not this statement is correct can be determined in one and only one way: *the set of target consumers who know of open source operating systems will decide if such operating systems are "ready for the desktop"*. I am interested to learn what percentage of users who have actually seen and used open source operating systems in fact do *not* use such operating systems for their "desktop" in some capacity. To summarize my point, *the only judge whose opinion matters is the target consumer.*

Lastly, the reader should be made aware that when I speak of FUD tactics I do not mean the spreading of outright lies of competitors' products. By FUD tactics I mean what is within the bounds of the law: negative advertisement of competitors which do not break any laws. In no way should it be construed that I am advocating the use of illegal acts in order to obtain market domination, or that I am implying that some companies in fact use FUD tactics.

## 8. Acknowledgements

I would like to thank Richard Stallman for his vision. If he had not demanded better software, the open source community would not be what it has become. Thanks also for Emacs and the creation of copyleft.

I would like to thank Linus Torvalds for creating such a beautiful operating system and for taking the notion of cooperative programming to its natural conclusion.

I would like to thank Eric S. Raymond for wresting from the unconscious the key axioms of hacker behavior. His insights inspired the writing of this this paper. I hope that he finds the conclusions of this paper worth analyzing.

I would like to thank the open source community and the free software foundation for inspiring me to try to understand the nature of information technology.

I would like to thank D. Knuth for creating TEX.

Lastly, I would like to thank enya for her voice. Her exquisite music has a stirring effect on my thought processes.

For responses and questions the author can be reached at prose@freemail.com.au.

FIGURE 8. Two proprietary companies

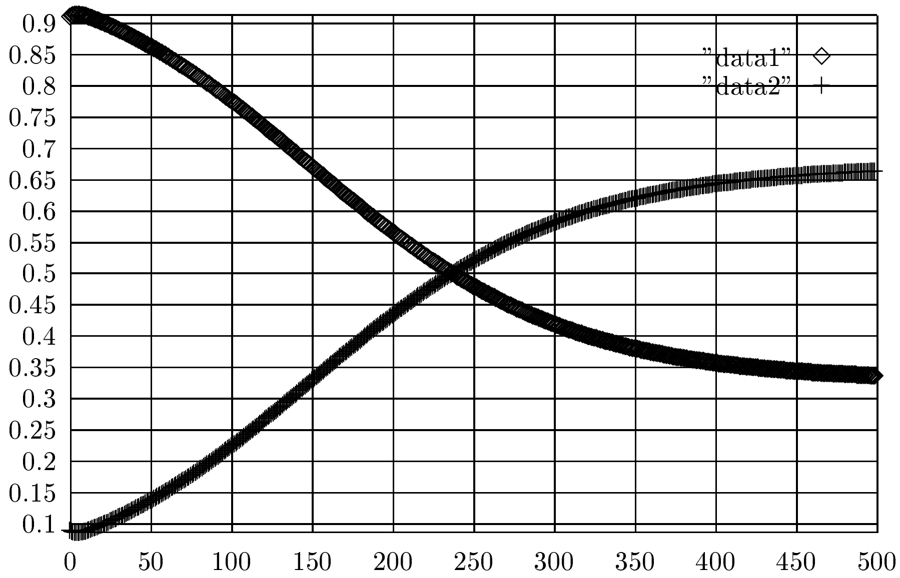
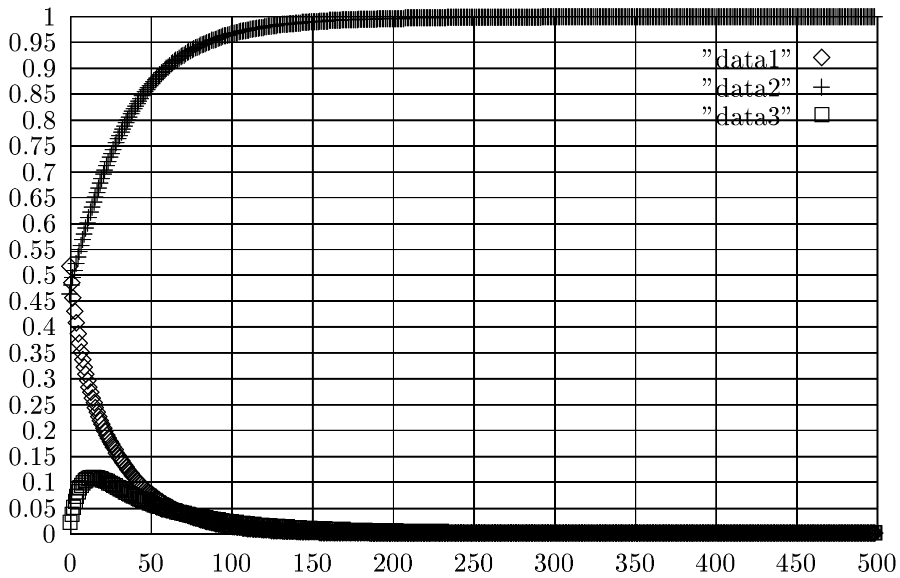



FIGURE 9. Two proprietary companies and 20% brand recognition of open source




## APPENDIX A. SAMPLE GRAPHS OF MARKET SHARE

Figure 10. Two proprietary companies, 20% brand recognition for open source, and $\beta_3 = 2.0$

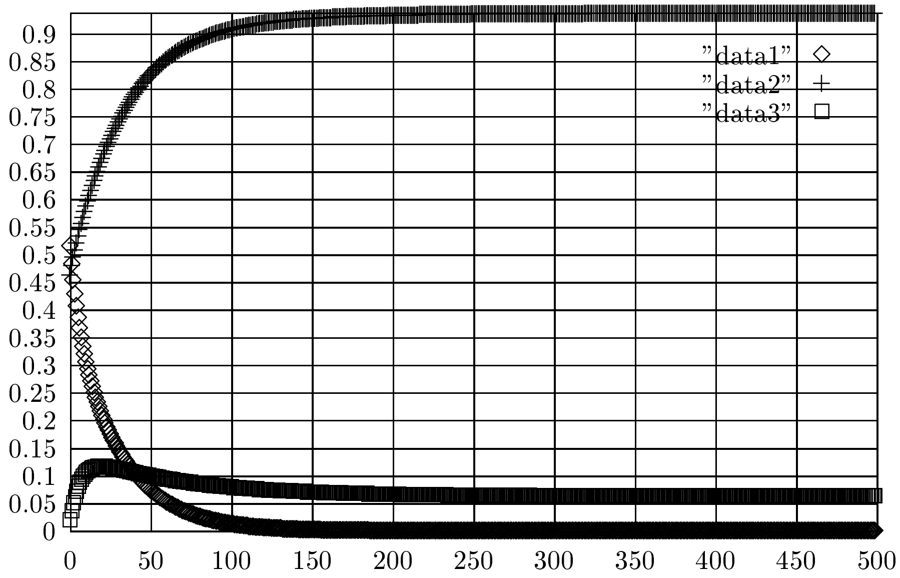
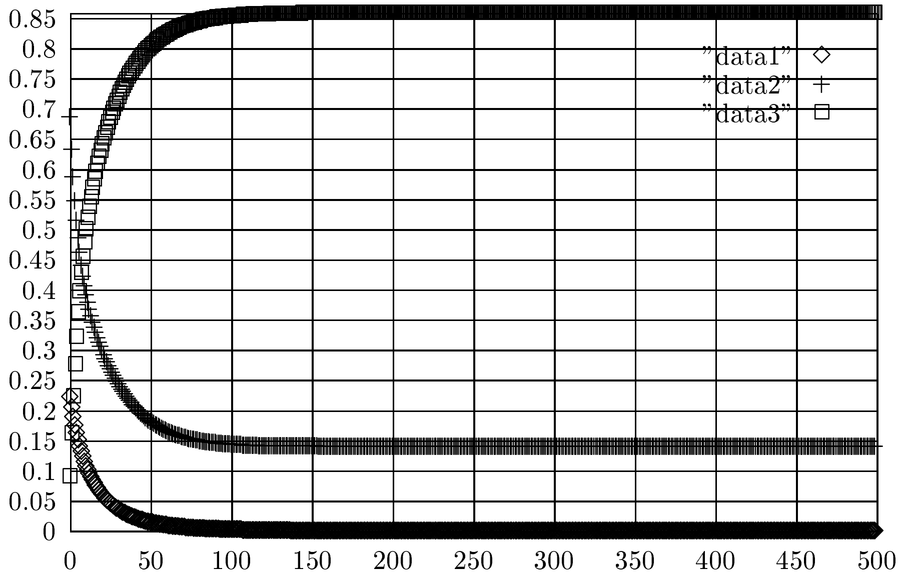



Figure 11. Two proprietary companies and 90% brand recognition of open source



## Appendix B. The source code for osmodel.c

Please feel free to use the code provided and change it as you see fit. If you wish to pass binary executables onto others please be sure to provide the source code as well, and indicate if the source code is mine or a modified version of my code. If you observe any errors in the code please inform me of the nature and location of the errors. Note that the programs were designed to handle at most 16 companies.

osmodel was designed to investigate the long term behavior of a given system in which the initial market share and developer share are specified.

osmodel2 was designed to investigate the location and nature of equilibria for a given system of companies with various operating and developer strengths.

### B.1. How to use the programs.

To use these programs in Unix:

1. Compile osmodel.c with:

   gcc -lm osmodel.c -o osmodel
2. Execute with:

   ./osmodel
3. Compile osmodel2.c with:

   gcc -lm osmodel2.c -o osmodel2
4. Execute with:

   ./osmodel2
5. Input the data.
6. The output consists of $N$ data files, $data1, \ldots, dataN$ where $N$ is the number of companies. The format of the data is in the form: $i^{th}$ entry in the $dataj$ file is the total market share for company $j$ at step $i$. To graphically view the data I suggest the following steps:
   (a) Use gnuplot:

   gnuplot

   (b) Plot the data:

   set ytics 0.0,0.05,1.0
   set grid
   plot "$data1$","$data2$",(keep typing etc.),"$dataN$"

   (c) If none of the operating systems have 100% brand recognition, then the sum of total market share of the operating systems will be a constant value equal to $(100 - untouchables)\% < 100\%$.

### B.2. The source code for osmodel.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
void doa(char *message)
{
fprintf(stderr,"%s\n",message);
exit(1);
}
int theweight(int thenumber)
{
```

```
  /* Compute the weight of a number < 2^16 */
unsigned  int i,j,k,l;

i = thenumber;
i = (i & 0x5555) + ((i>> 1) & 0x5555);
i = (i & 0x3333) + ((i>> 2) & 0x3333);
i = (i & 0x0f0f) + ((i>>4) & 0x0f0f);
i = (i & 0x0f) + ((i>>8) & 0x0f);
return i;
}
/* compute the relative state of a subset with respect to a
   given company. */
int thestate(int whichsubset, int whichcompany)
{
int i,j,k;

i = (1 << 16) - (1 <<whichcompany);
j = (1 << (whichcompany-1)) - 1;
k = ((whichsubset & i) >> 1) + (whichsubset & j);
return k;
}
/* compute the subset given the relative state with respect to
   a given company. */
int illuminati(int whichcompany, int whichpseudoset)
{
int i,j,k;

i = (1 << 16) - (1 <<whichcompany);
j = (1 << (whichcompany-1)) - 1;
k = ((whichpseudoset <<1) & i) + (whichpseudoset &j) + (1 << whichcompany-1);
return k;
}
/* determine which companies belong to a given subset. */
void whichcomp(int thenumber, int* hannibal, int howmany)
{
int i,j,k,l;

for (i = 0; i < howmany+1;i++)
  hannibal[i] = 0;

i = 1;
j = 0;
while (i < howmany+1)
  {
    if (thenumber & (1 << (i-1)))
      hannibal[j++] = i;
    i++;
  }
return;
```

```
}
/* compute new market shares */
void  scorecomps(unsigned int whichsubset, double *stri,
double *developershare,double **companyshare,
double **fakecompanyshare, double stepsize, int nocomp)
{
int **usualsuspects;
int howmany;
unsigned int i,j,k,l,m,n,p,q;
double totstren = 0;

/* determine how many companies are in contention */
howmany = theweight(whichsubset);

/* only 1 company --> it has a lock in. No change. */
if (howmany == 1)
  {
    for ( j = 1; j < nocomp+1;j++)
      if ((1 << (j-1)) & whichsubset)
        {
          fakecompanyshare[j][0] = companyshare[j][0];
          j = nocomp+1;
        }
    return;
  }

/* more than 1 company in contention. Allocate space. */
if ((usualsuspects = (int**)calloc(howmany,sizeof(int*))) == NULL)
  doa("I had trouble with Kaiser.\n");
for (i = 0; i < howmany;i++)
  if ((usualsuspects[i] = (int*)calloc(2,sizeof(int))) == NULL)
    doa("What's the devil's greatest trick?\n");

/* Figure out which companies are in contention. */
i = 0;
j = 0;
while (i < howmany)
  {
    if ( (1<<j) & whichsubset)
      {
        usualsuspects[i][0] = j+1;
        usualsuspects[i][1] = thestate(whichsubset, j+1);
        i++;
      }
    j++;
  }
/* compute total strength. */
totstren = 0.0;
```

```
for (i = 0; i < howmany;i++)
  {
    j = usualsuspects[i][0];
    totstren += stri[j]*developershare[j];
  }
totstren = 1.0/totstren;
/* compute the new companyshares. */
for (i = 0; i < howmany;i++)
  {
    j = usualsuspects[i][0];
    k = usualsuspects[i][1];
    fakecompanyshare[j][k] = companyshare[j][k]
+ stepsize*(totstren*stri[j]*developershare[j] - companyshare[j][k]);
  }
/* free up the memory. */
for (i = 0; i < howmany;i++)
  free(usualsuspects[i]);

free(usualsuspects);
return;
}
/* compute new developer shares. */
void scoredevelopers(double *dstri,double stepsize,
double *mi, double *developershare,
double *fakedevelopershare, int nocomp)
{
  int howmany;
  unsigned int i,j,k,l,m,n,p,q;
  double totstren = 0;

  howmany = nocomp;

  for (i = 1; i < howmany+1;i++)
    totstren += dstri[i]*mi[i];

  totstren = 1.0/totstren;

  for (i = 1; i < howmany+1;i++)
    fakedevelopershare[i] = developershare[i]
+ stepsize*(totstren*dstri[i]*mi[i] - developershare[i]);

  return;
}
/* determine total market share for each company. */
void computetotalmarket(double **companyshare,double *mi,
double *themaster,int nocomp)
{
  int i,j,k;
```

```
  j = nocomp;
  for (k = 1; k < j+1; k++)
    {
      mi[k] = 0.0;
      for (i = 0; i < (1 << (j - 1));i++)
          mi[k] += companyshare[k][i]*themaster[illuminati(k,i)];
    }
  return;
}

main()
{
int i,j,l,k,*hannibal;
double *alpha, *beta;
double *stri,*dstri;
double *bi,*fpi,*themaster;
double *mi, *fakemi, *fakedi;
float tempie,tempie2;
double **companyshare, **fakecompanyshare;
double *developershare,*fakedevelopershare;
double *ci;
FILE **output;
int howmany;
double stepsize;
double sum;
int howlong;
char file[100];
time_t *thedummy;
thedummy = NULL;

l = time(thedummy);
srand(l);


/* ************************************* */
printf("How many companies?");
scanf("%d",&howmany);
if ((howmany < 2) || (howmany > 10))
  doa("That number is out of range.\n");

if ((companyshare = (double**)calloc(howmany+1,sizeof(double*))) == NULL)
  doa("I couldn't allocate space for companyshare[]\n");

for (i = 1; i < howmany+1;i++)
  if ((companyshare[i] = (double*)calloc((1<<(howmany-1)),sizeof(double)))
      == NULL)
    doa("I couldn't allocate space for a companyshare[][]\n");

if ((developershare = (double*)calloc(howmany+1,sizeof(double))) == NULL)
```

```
  doa("I couldn't allocate space for developershare.\n");

if ((dstri = (double*)calloc(howmany+1,sizeof(double))) == NULL)
  doa("I couldn't allocate space for dstri.\n");
/* ------------------------------------------------------- */
if ((fakecompanyshare = (double**)calloc(howmany+1,sizeof(double*))) == NULL)
  doa("I couldn't allocate space for fakecompanyshare[]\n");

for (i = 1; i < howmany+1;i++)
  if ((fakecompanyshare[i] = (double*)calloc((1<<(howmany-1)),sizeof(double)))
      == NULL)
    doa("I couldn't allocate space for a fakecompanyshare[][]\n");

if ((fakedevelopershare = (double*)calloc(howmany+1,sizeof(double))) == NULL)
  doa("I couldn't allocate space for fakedevelopershare.\n");

/* ------------------------------------------------------- */
if ((alpha = (double*)calloc(howmany+1,sizeof(double))) == NULL)
  doa("I couldn't allocate space for alpha.\n");

if ((beta = (double*)calloc(howmany+1,sizeof(double))) == NULL)
  doa("I couldn't allocate space for beta.\n");

if ((bi = (double*)calloc(howmany+1,sizeof(double))) == NULL)
  doa("I couldn't allocate space for bi.\n");

if ((fpi = (double*)calloc(howmany+1,sizeof(double))) == NULL)
  doa("I couldn't allocate space for fpi.\n");

if ((stri = (double*)calloc(howmany+1,sizeof(double))) == NULL)
  doa("I couldn't allocate space for stri.\n");

if ((output = (FILE**)calloc(howmany+1,sizeof(FILE*))) == NULL)
  doa("I couldn't allocate space for output.\n");
if ((mi = (double*)calloc(howmany+1,sizeof(double))) == NULL)
  doa("I couldn't allocate space for the market shares.\n");

/* This is where it's different. */
/* Input the data */
for (i = 1; i < howmany+1;i++)
  {
    printf("brand recognition for company %d (0.0 - 1.0):",i);
    scanf("%f",&tempie);
    bi[i] = tempie;
    printf("Strength of OS-G_%d, stri[%d] (0.0 - 1.0):",i,i);
    scanf("%f",&tempie);
    stri[i] = tempie;
    printf("Initial percentage of developers for OS-G_%d (0.0-1.0):",i);
    scanf("%f",&tempie);
```

```
    developershare[i] = tempie;
    printf("Price factor for developer's kit for OS-G_%d (0.0-1.0):",i);
    scanf("%f",&tempie);
    fpi[i] = tempie;
    printf("Predatory factor of company %d (0.0-1.0):",i);
    scanf("%f",&tempie);
    alpha[i] = tempie;
    printf("Reward/incentive factor from  company %d ( >= 1.0):",i);
    scanf("%f",&tempie);
    beta[i] = tempie;
    dstri[i] = alpha[i]*beta[i]*fpi[i];
  }
/* This is where the difference ends. */
/* Ignore the untouchables. */
if ((hannibal = (int *)calloc(howmany+1,sizeof(int))) == NULL)
  doa("I couldn't allocate space for my pointer to companies.\n");

if ((themaster = (double *)calloc((1 << howmany),sizeof(double))) == NULL)
  doa("I couldn't allocate space for M.S. sizes?\n");

printf("Now it's time for initial market share percentages
for distinct target consumer subsets.\n");
for (i = 1; i < (1 << howmany);i++)
  {

    whichcomp((1<< howmany) - 1 - i,hannibal,howmany);
    j = 0;
    themaster[i] = 0.0;
    while(hannibal[j])
      {
        if (bi[hannibal[j]] != 1.00000)
          themaster[i] += log( 1.0 - bi[hannibal[j]]);
        else
          {
          themaster[i] = -200.0;
          }
        j++;
      }

   /* Here the difference begins again. */
    whichcomp(i,hannibal,howmany);
    j = 0;
    printf("********************************************************\n");
    printf("In this  subset consumers know only company(s): ");
    while(hannibal[j])
      {
        printf("%d ",hannibal[j]);
        themaster[i] += log(bi[hannibal[j]]);
        j++;
```

```
      }
    themaster[i] = exp(themaster[i]);
    printf("\n--------------------------------------\n");
    printf("Please enter the initial market share for this subset for
each company. Rememeber that the market shares must sum up to 1.0
in this subset. If these consumers only know 1 company then enter 1.0\n");
    for (k = 0; k < j;k++)
      {
        printf("company %d (0.0-1.0): ",hannibal[k]);
        scanf("%f",&tempie);
        companyshare[hannibal[k]][thestate(i,hannibal[k])] = tempie;
      }
  }
/* Here is where the difference ends again. */
printf("Now it's time to determine how small a stepsize
to use with Euler's method.\n");
printf("Stepsize (0.00001 - 0.1 suggested)?");
scanf("%f",&tempie);
stepsize = tempie;
printf("How many steps to take (roughly 1/stepsize - 10/stepsize suggested)?");
scanf("%d",&howlong);

/* Open the stat files first. */

for (i = 1; i < howmany+1;i++)
  {
    sprintf(file,"data%-d",i);
    printf("Opening file %s:\n",file);
    if ((output[i] = fopen(file,"wb")) == NULL)
      doa("I couldn't open an output file.");
  }
/* Do the simulation now. */
for (i = 0; i < howlong;i++)
  {
    /* compute the current market shares */
    computetotalmarket(companyshare,mi, themaster,howmany);
    /* write out the current data. */
    for (j = 1; j < howmany+1;j++)
      fprintf(output[j],"%f\n",mi[j]);
    /* compute the new companyshares. */
    for (j = 1; j < (1 << howmany);j++)
      scorecomps(j, stri, developershare,companyshare,
                 fakecompanyshare, stepsize, howmany);
    /* compute the new developer shares */
    scoredevelopers(dstri,stepsize,mi,developershare,
                    fakedevelopershare,howmany);
    /* move the data around. */
    for (j = 1; j < howmany+1; j++)
      for (k = 0; k < (1 << (howmany));k++)
```

```
        companyshare[j][k] = fakecompanyshare[j][k];
    for (j =1; j < howmany+1;j++)
      developershare[j] = fakedevelopershare[j];
  }

for (j = 1; j  < howmany+1;j++)
  {
    fclose(output[j]);
  }
tempie = 0.0;
for (i = 1; i < howmany+1;i++)
  if (bi[i] < 1.0)
    tempie += log(1 - bi[i]);
  else
    {
      tempie = 0.0;
      i = howmany+2;
    }
if (i == howmany+1)
  tempie = exp(tempie);
printf("The size of the untouchables is %f\n",tempie);
}
```

B.3. **The source code for osmodel2.** The source code for osmodel2 is identical with osmodel with the exception that in the main subroutine the snippet of code

```
/* This is where it's different. */
/* Input the data */
for (i = 1; i < howmany+1;i++)
  {
    printf("brand recognition for company %d (0.0 - 1.0):",i);
    scanf("%f",&tempie);
    bi[i] = tempie;
    printf("Strength of OS-G_%d, stri[%d] (0.0 - 1.0):",i,i);
    scanf("%f",&tempie);
    stri[i] = tempie;
    printf("Initial percentage of developers for OS-G_%d (0.0-1.0):",i);
    scanf("%f",&tempie);
    developershare[i] = tempie;
    printf("Price factor for developer's kit for OS-G_%d (0.0-1.0):",i);
    scanf("%f",&tempie);
    fpi[i] = tempie;
    printf("Predatory factor of company %d (0.0-1.0):",i);
    scanf("%f",&tempie);
    alpha[i] = tempie;
    printf("Reward/incentive factor from  company %d ( >= 1.0):",i);
    scanf("%f",&tempie);
    beta[i] = tempie;
    dstri[i] = alpha[i]*beta[i]*fpi[i];
  }
```

```
/* This is where the difference ends. */
```

should be replaced with

```
/* This is where it's different. */
/* Input the data */
tempie2 = 0;
for (i = 1; i < howmany+1;i++)
  {
    printf("brand recognition for company %d (0.0 - 1.0):",i);
    scanf("%f",&tempie);
    bi[i] = tempie;
    printf("Strength of OS-G_%d, stri[%d] (0.0 - 1.0):",i,i);
    scanf("%f",&tempie);
    stri[i] = tempie;
    printf("Developer strength of OS-G_%d, dstri[%d] (0.0 - 1.0):",i,i);
    scanf("%f",&tempie);
    dstri[i] = tempie;
    developershare[i] = rand() + 0.001;
    tempie2 += developershare[i];
  }
tempie2 = 1.0/tempie2;
for (i = 1; i < howmany+1;i++)
  developershare[i] *= tempie2;
tempie2 = 0.0;
/* This is where the difference ends. */
```

and the snippet of code

```
   /* Here the difference begins again. */
    whichcomp(i,hannibal,howmany);
    j = 0;
    printf("*******************************************************\n");
    printf("In this  subset consumers know only company(s): ");
    while(hannibal[j])
      {
        printf("%d ",hannibal[j]);
        themaster[i] += log(bi[hannibal[j]]);
        j++;
      }
    themaster[i] = exp(themaster[i]);
    printf("\n---------------------------------------\n");
    printf("Please enter the initial market share for this subset for
each company. Rememeber that the market shares must sum up to 1.0
in this subset. If these consumers only know 1 company then enter 1.0\n");
    for (k = 0; k < j;k++)
      {
        printf("company %d (0.0-1.0): ",hannibal[k]);
        scanf("%f",&tempie);
        companyshare[hannibal[k]][thestate(i,hannibal[k])] = tempie;
      }
  }
```

```
/* Here is where the difference ends again. */
```

should be replaced with

```
    /* Here the difference begins again. */
      whichcomp(i,hannibal,howmany);
    j = 0;
    while(hannibal[j])
      {
        j++;
      }
    themaster[i] = exp(themaster[i]);
    tempie = 0.0;
    for (k = 0; k < j;k++)
      {
        companyshare[hannibal[k]][thestate(i,hannibal[k])] =
                              (double)  (rand()*1.0/RAND_MAX + 0.00001);
        tempie += companyshare[hannibal[k]][thestate(i,hannibal[k])];
      }
    tempie = 1.0000/tempie;
    for (k = 0; k < j;k++)
      {
      companyshare[hannibal[k]][thestate(i,hannibal[k])] *= tempie;
      }
  }
/* Here is where the difference ends again. */
```

## Appendix C. Sample probability graphs

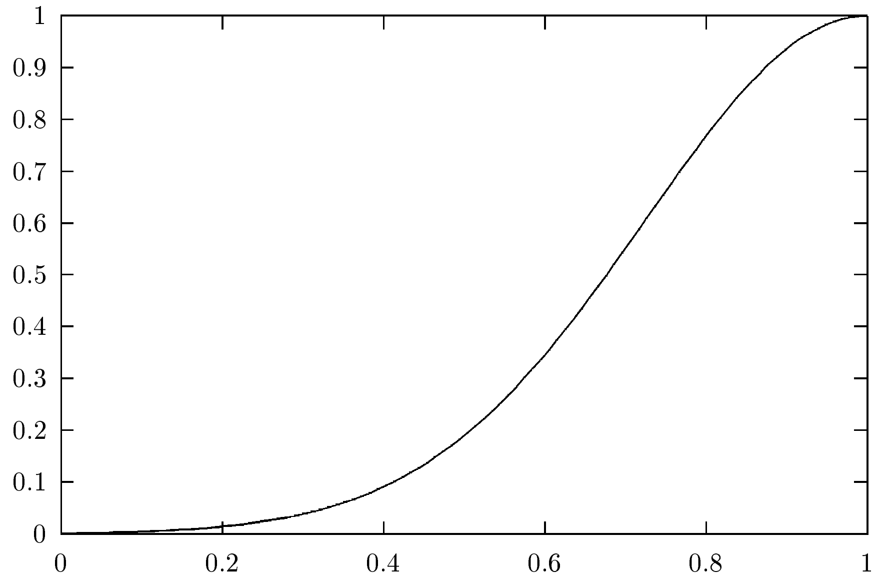



Figure 12. Sample graph of customer satisfaction vs. prob. of acceptance

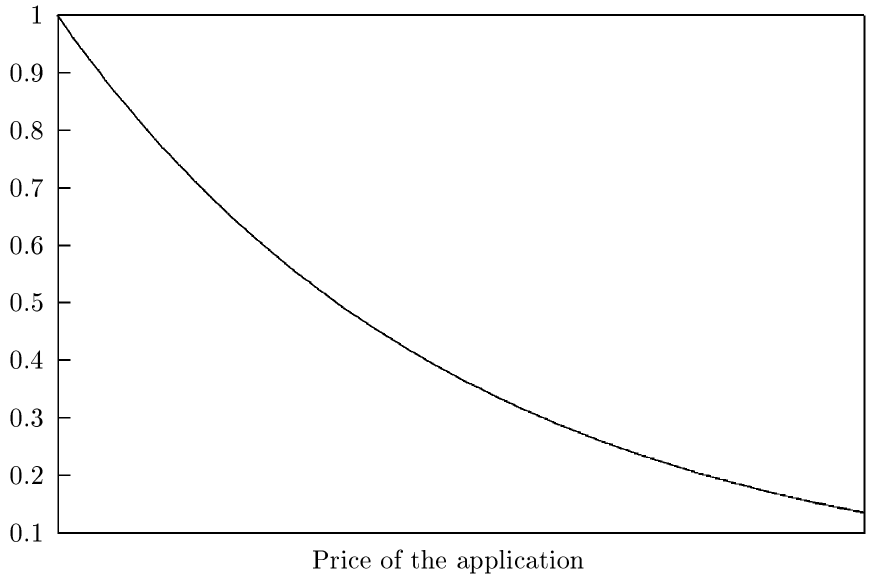



FIGURE 13. Sample graph of price of application vs. prob. of acceptance

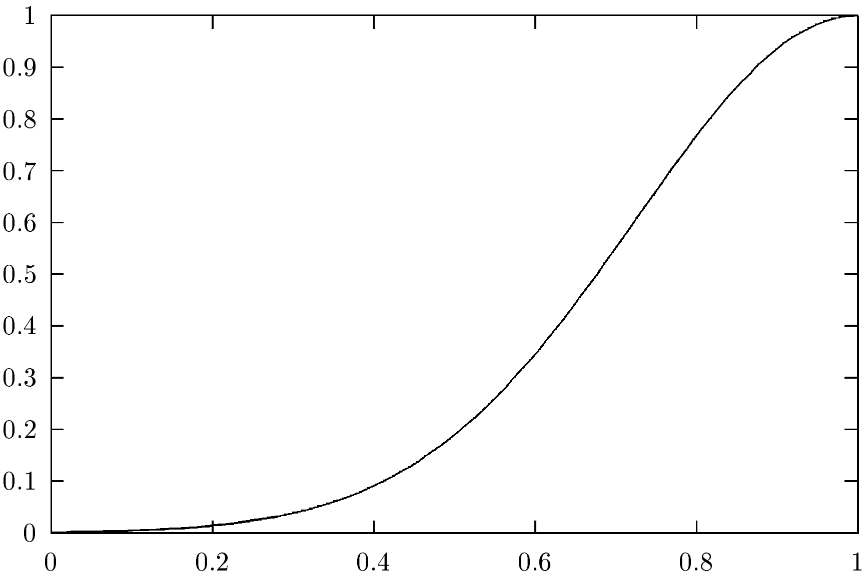



FIGURE 14. Sample graph of software reliability vs. prob. of acceptance

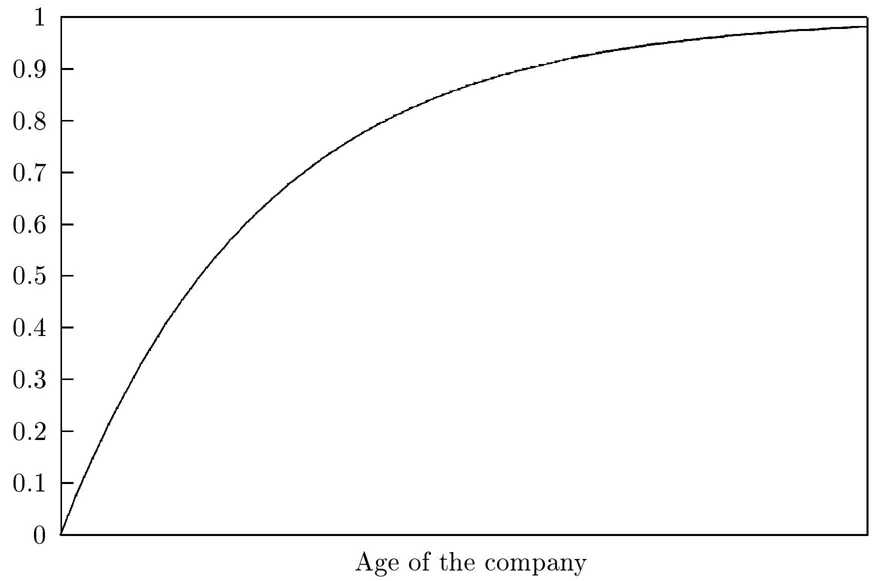



Figure 15. Sample graph of age of company vs. prob. of acceptance

## References

[1] Eric S. Raymond, *Halloween I*, http://www.tuxedo.org/ẽsr/writings/halloween1.shtml, 1998.
[2] Eric S. Raymond, *Halloween II*, http://www.tuxedo.org/ẽsr/writings/halloween2.shtml, 1998.
[3] Eric S. Raymond, *The Cathedral and the Bazaar*, http://www.tuxedo.org/ẽsr/writings/cathedral-bazaar, 1998.
[4] Eric S. Raymond, *Homesteading the Noosphere*, http://www.tuxedo.org/ẽsr/writings/homesteading, 1998.

*E-mail address*: prose@freemail.com.au