DON'T JUST ROLL THE DICE

by Neil Davidson

A usefully short guide to software pricing

efendi minibooks

redgate

Neil Davidson

EFENDI MINIBOOKS

Don't Just Roll The Dice

Don't Just Roll The Dice

First Published: 2009
This Edition Published: 2012

eBook Version: 2.0.0

Published by **Efendi Books**
First published by **Simple Talk Publishing**

ISBN: 978-0-9571791-2-7

Table Of Contents

About The Authors i
About The Reviewers i
Foreword ii
An Introduction To Don't Just Roll The Dice ii
Chapter 1 - Economics
Chapter 2 - Pricing Pyschology
What Is Your Product? 8
Perceived Value 9
Chapter 3 - Pitfalls
Competitors 18
Fairness 19
Pirates 19
Switching Costs 20
Should You Take Your Costs Into Account? 21
Chapter 4 - Advanced Pricing
Versioning 25
Bundling 31
Multi-user Licenses 32
The Purchasing Process 33
Free 34
Bargains 37
Apps and App Markets 38
Software As A Service 40
Different Ways Of Pricing 45
Choosing The Right Model 48
Chapter 5 - Pricing Perception
Practice Trumps Theory 51
How To Change Your Pricing 52
Chapter 6 - Product Pricing Checklist
Summary & Bibliography 57

About The Authors

Neil Davidson is co-founder and joint CEO of Red Gate software. Red Gate was founded in 1999 and now employs some 300 people across 3 continents. It has been in the Sunday Times top 100 companies to work for six years running. It was founded with no VC money and little debt. Neil is also founder of the annual Business of Software conference.

Jamie Rumbelow is a software developer and writer from Cambridge, UK. He is the founder and director of Efendi Books, a small publishing start-up, as well as a part-time student and freelancer.

About The Reviewers

Phil Factor (real name withheld to protect the guilty), aka Database Mole, has 25 years of experience with database-intensive applications. Despite having once been shouted at by a furious Bill Gates at an exhibition in the early 1980s, he has remained resolutely anonymous throughout his career.

Michael Pryor founded Fog Creek Software with Joel Spolsky in September 2000. He has served as the company's president since the beginning, and has also been the CFO since 2006. Michael graduated from Dartmouth College with an Honors B.A. in Computer Science (Phi Beta Kappa, magna cum laude). After graduation, he joined Juno Online Services, as a Windows client developer. He writes a column for Make Magazine called Puzzle This, and runs the popular interview website TechInterview.org.

Neil's Foreword

At Business of Software 2007, Michael Pryor held an impromptu session on how to price your software. So many people turned up, and so many people kept on arriving, that by the time they'd introduced themselves there was no time left to talk about software pricing. I've had similar experiences; in fact, "How do I price my software?" is probably the most common question I'm asked by software entrepreneurs and product managers.

This handbook is an attempt to answer that question.

But first, I'd like to thank Phil Factor, Tony Davis and Michael Pryor for all their editing, reviewing and suggestions. More people than I can possibly mention have contributed with offers of help, anecdotes and proofreading. This handbook was way better with their input than it ever could have been without. Thanks guys.

Jamie's Foreword

It turns out it's very hard to improve on brilliance.

Don't Just Roll The Dice 1st Edition was a brilliant book. When Neil came to me and suggested that we revise and update his popular free eBook on software pricing, I was terrified. I didn't want it to lose its sense of clarity, its usefully short approach to instruction or its fantastic stories and context.

So don't be afraid that this won't be the book you know and love. It will be. DJRTD 2.0 is an evolution, not a revolution.

We've listened to your feedback and worked hard at revising passages of text, updating statistics and bringing the content into 2012. We've added a whole new section on Software As A Service pricing, as well as a few bits on pricing your mobile apps. We've added some lovely new illustrations and re-designed the book to give it a whole new look and feel.

Welcome to Don't Just Roll The Dice 2. I hope you enjoy it.

An Introduction To Don’t Just Roll The Dice

In 1938, two young engineers were ready to launch their first product. They’d struggled with what to build. After considering amplifiers, radio equipment, air controllers, harmonicas and even muscle-building electrodes for housewives, they’d finally decided to create an oscilloscope. Not wanting customers to be put off by a version one product, they sensibly called it the Model 200A.

The next step? Decide the pricing.

They eventually settled on $54.40. Was that because it represented the cost of manufacturing, plus a decent markup? No. These engineers hadn’t taken that into account. In fact, they soon realized that the cost of building each oscilloscope was more than the price they were asking. Was it based on what the competition charged? No. They hadn’t bothered to discover that General Radio charged $400 for an equivalent model.

They chose $54.40 because it reminded them of the 1844 slogan used in the campaign to establish the northern border of the United States in the Pacific Northwest (“54” 40' or Fight!“).

What a dumb-ass way to price a product.

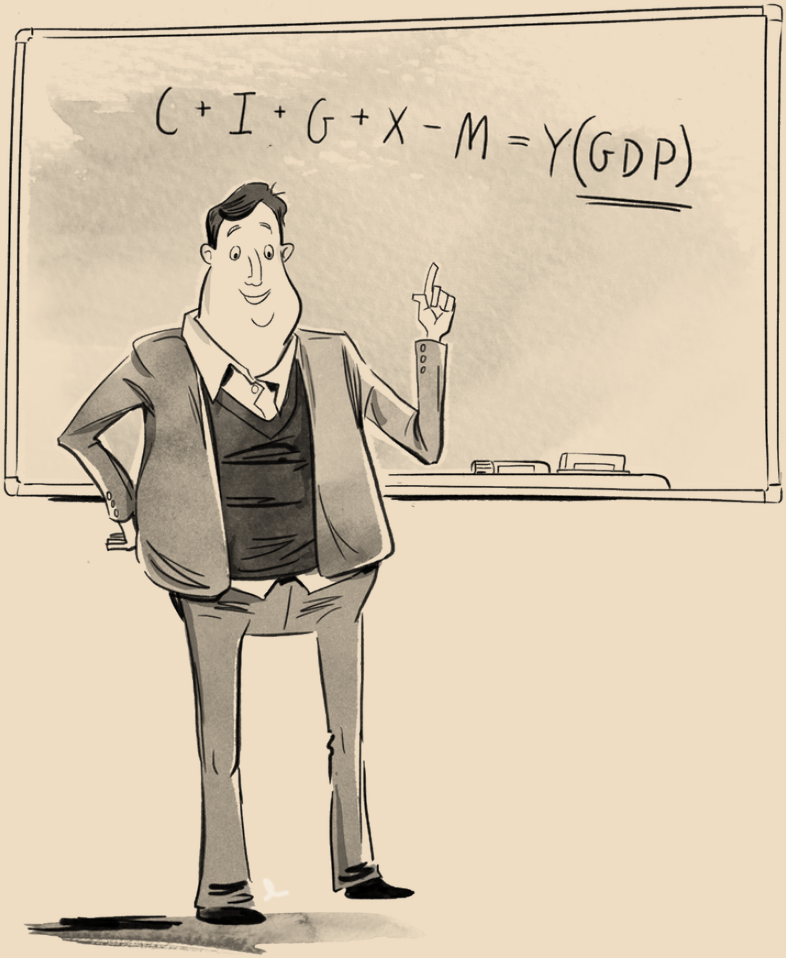
But these two young engineers recovered from their stumble. The Model 200A went on to become the longest-selling basic electronic design of all time, still selling 33 years later. The company they founded became an institution. Their names? Dave Hewlett and Bill Packard.

If Hewlett and Packard, two Stanford graduates with the rosiest of futures ahead of them, can flounder so badly when faced with the problem of how to price their products, what hope do the rest of us have?

Quite a lot, as it turns out.

Chapter One

Economics



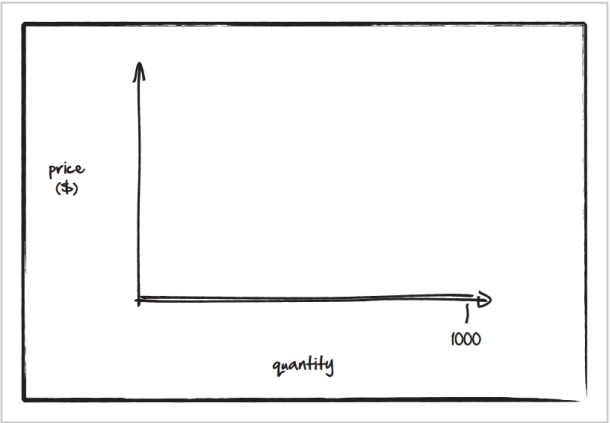
To understand product pricing, it helps to understand some, but not too much, economics. The easiest way is through a simple example.

Let's say you've just launched the Time Tracker 3000. It's a downloadable piece of software that logs which applications you use throughout the day, and sends the usage information back to a central web site. From there, you can find out what you've been up to all day long.

You've decided to charge a one-off fee for it, but how are you going to price it?

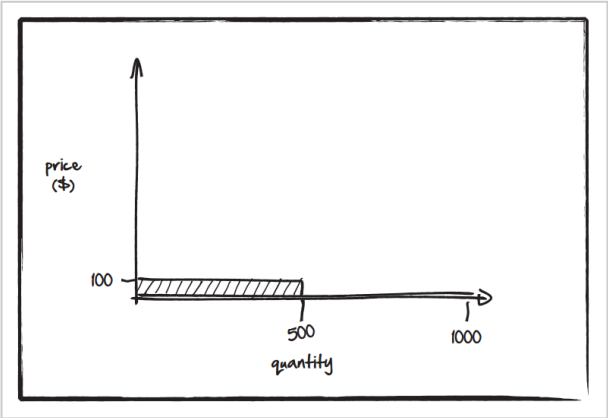
If you give it away for free, you'll get lots of customers. A thousand, say, including Belinda the bargain hunter, Stewart the student, Willhelm the web start-up founder, Pat the product manager and Ernest the enterprise developer.

Let's represent these thousand customers, paying no money, as an infinitely thin horizontal bar (representing the $0 price), 1000 units long (representing the quantity):

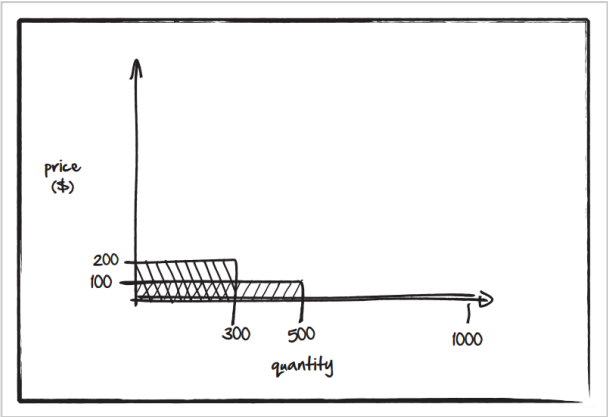


If you charge $100 instead, the number of purchasers will instantly be a lot less. Belinda is a bargain hunter and was only using the software because it was free, and Stew is a student, so neither of them will buy. You'll get, for the sake of argument, five hundred customers instead of the original thousand. Let's represent that as a bar with a height of $100 and a length of 500 units. What's the revenue you generate from this? It's the area of the bar, so $100 x 500 = $50,000.

Let's overlay it on top of the first bar:

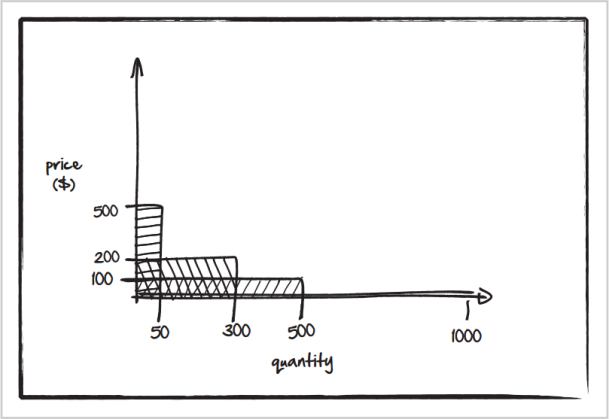


What happens if you increase the price to $200? Some of the people who would have bought at $100 will no longer buy, so your number of sales will decrease again. Willhelm runs his own company and can't justify the price, so he's no longer interested. Let's say 300 people will still buy; represent this with a rectangle and overlay it onto the chart. Again, the revenue you make is the area of the rectangle. $200 x 300 = $60,000:

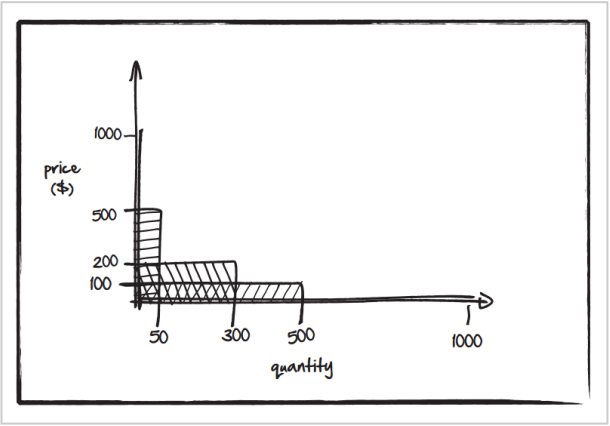


Let's increase the price once more, to $500. Pat the product manager drops off because at that price she'd rather buy a competitor's tool. Let's say 50 people will still

buy, and represent this as a rectangle, overlaid on the chart. Once more, the value of this rectangle is its area: 50 people buying at $500, so 50 x $500 = $25,000.

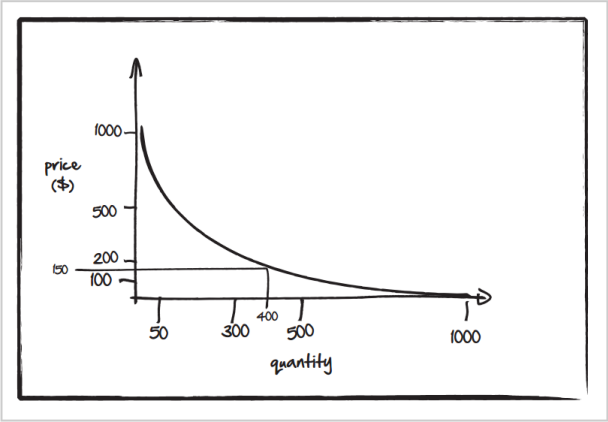


Finally, since the Time Tracker 3000 is valuable, but not that valuable, let's assume that nobody will buy if you price it at $1,000 and represent this as a bar of width 0 and plot it on the chart:



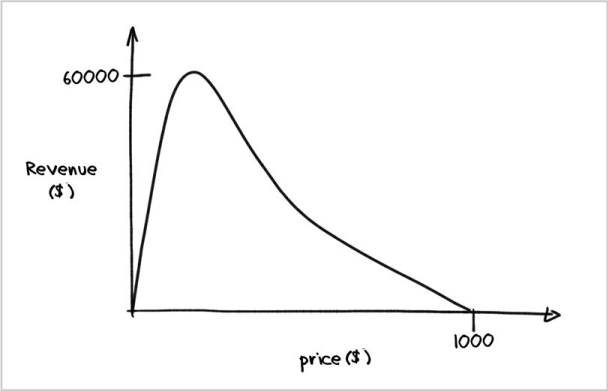
We've plotted five points on what is becoming a curve of price against the number of people who will buy the Time Tracker 3000 at that price. What's more, you can work

out the total revenue you will get at any particular price by looking at the area of the rectangle (price multiplied by purchasers) under the point at that graph:



Economists call this a **demand curve**.

To maximize the revenue of the Time Tracker 3000, we need to find a point on the graph that maximizes the size of the rectangle underneath it. To understand that, it helps to plot the area (i.e. the total revenue) against the price. For the Time Tracker 3000, this looks something like this. (I've plotted the five data points we've already got):



From the diagram, you can see you should price the Time Tracker 3000 at around $200. It's not where you'll sell the most units, but it's where you'll make the most money.

Plotting a demand curve is, in theory, straightforward, but in practice it is way harder. In the real world, you don't know what the shape of the demand curve is, or where your current price sits on it. On some curves, and on some points of the curve, you'd be right to increase your prices: the reduction in the number of people buying your product will be outweighed by the increased revenue from each person. On other curves, or on other points of the same demand curve, increasing your prices will lead to a massive drop-off in sales and you'll lose money.

What's more, the shape of the demand curve depends on a bunch of variables: your competitors, how they'll react to any price changes you make, your customers, their environment and situation, and the type and quality of your product.

Chapter Two

Pricing Psychology

The demand curve, discussed in the previous section, might be dynamic and depend on many factors, but you can still exert some influence on its shape. In this chapter, I'll talk about how people decide how much they'll pay for a product, and how you can change this.

But first, you need to be able to answer a simple question: what is your product?

What Is Your Product?

You might think that your software product is just the bits and bytes that your customers download, or the HTML pages that they view, but you'd be wrong. In reality, your product is much broader than that. It's not just the software or the web site: it's the documentation, the help required to get it working and the promise of support when things go wrong. It's the future roadmap of the product, the pledge to carry on developing future versions. In some cases, it's a dream, an experience; a way of life.

One of the clearest examples comes from the accounting industry. At Red Gate, we use Sage's accounting software. We're not the only ones: Sage is a software business with 6.3 million customers. They employ 13,000 people and that has a market capitalization of nearly four billion pounds. It dominates the accounting market in the UK, and has seen off concerted attacks from, among others, Microsoft and Intuit.

But their software sucks.

It's slow, and it's hard to use. When I first used it in 1999, the buttons on the toolbar didn't depress when I clicked them: they were just static pictures painted on a grey background. The application is so ugly that the product walkthroughs on the Sage web site barely feature the product itself.

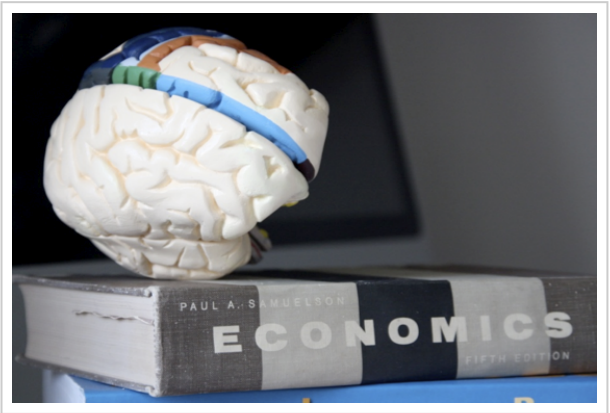
These two facts–the awfulness of the product and the magnitude of its success–can be reconciled if you understand that Sage's **product** is more than just the software.

When you buy Sage software, you are not just buying software. You are buying **reassurance**: when the tax laws change, the software will get updated too. You are buying **familiarity**: if you buy Sage, the odds are that your accountant or bookkeeper will already be able to use it. You're buying **support**: if you don't understand some of

the accounting codes or procedures you need, then you can phone somebody for help. Tens of thousands of people call Sage's help line per day.

The reason that Sage is so dominant in the UK is because Sage understands exactly what their product is. You need to do the same.

Perceived Value



Once you've determined what your product is, you need to consider its value to your customers. In the case of the Time Tracker 3000, let's say that it will save a particular customer, Willhelm, three hours of work and that Willhelm prices his time at $50 an hour. That means that Willhelm should buy the Time Tracker 3000 at any price under $150, assuming he has nothing better to spend his money on. This is its **objective value**.

Of course, this assumes that Willhelm is the rational, decision-making machine that economists love. In fact, Willhelm is a flesh-and-blood, irrational human being who doesn't price his time and calculate costs and benefits. He has a **perceived value** of the Time Tracker 3000, which may or may not be linked to its objective value.

The perceived value of a product may be higher than its objective value. In 2003, Gartner released a report that claimed that almost half of all customer relationship management (CRM) systems lie unused. That's several billion dollars of software that smart people thought was worth it, but wasn't.

Beats by Dr. Dre headphones are another example where perceived values are much higher than objective values. Beats outline in promotional materials that with most headphones, listeners are not able to hear "all" of the music, and that Beats would allow people “to hear what the artists hear, and listen to the music the way they should”. The actual boost in quality is, of course, much less than the boost in cost.

A product's perceived value may be lower than its objective value too. A few years ago, I stumbled on somebody who insisted on using Excel as a word processor. According to this user, the additional expense of buying Microsoft Word wasn't worth the benefits he'd gain. This was almost certainly a perception rather than a reality.

Back to Willhelm and the Time Tracker 3000. If you want to change how much Willhelm will pay for your product, then changing the product is one option, but only if you can also change his perception too. In fact, it turns out that you can change Willhelm's perception of your product's worth without touching the product at all. That's one of the things marketing is for.

How People Set Their Perceptions

So how do people generate their perceived value of a product? How do they decide how to think?

For a start, it's extremely hard for them to do so in a vacuum. Try asking a British Member of Parliament how much a pint of milk costs, a contestant on The Price is Right for the value of a chest of drawers, or the average supermarket shopper how much they should pay for a bottle of bleach. They'll struggle.

People base their perceived values on **reference points**. If you're selling a to-do list application, then people will look around and find another to-do list application. If they search the internet and discover that your competitors sell to-do list applications at $100 then this will set their perception of the right price for all to-do list applications.

When Microsoft released DOS 1.0 in 1982, they set a price of $50. At the time, an operating system for a mass-market computer was a brand new category. Since consumers had no reference point, $50 became accepted as a 'fair' price. When IBM launched OS/2 1.0 in 1989 and priced it at $340, consumers baulked. DOS and OS/2 were very different operating systems, and $340 could well have been a fair reflection

of the additional benefit consumers would have extracted from the more-advanced OS/2, but Microsoft had already defined the reference point, and when IBM tried to challenge it they failed.

This doesn't mean you need to copy the reference point. If your product genuinely is better than your competitors', and you can demonstrate the value of this difference, or create a perception of that value, then you can charge more.

Of course, if your product is significantly less valuable than your competitors' then you may have no choice but to charge less. Competing on price may be the only option you have. Take pharmaceuticals. My local supermarket stocks thirty or so different sorts of painkillers. You can buy a pack of 16 Nurofen for £1.97 ($3.40), or a pack of 16 Tesco analgesics for £0.32 (about 50 cents). The physical good – the 200mg of ibuprofen – is identical in both the generic and named brand product. But don't forget that the entire product is more than the chemicals. It includes the marketing, brand name and packaging. Using this wider definition, Nurofen is the superior product, and the only way Tesco can compete is on price.

The value people perceive your product to have can depend on their **taste** and **culture**. Some people are passionate about good wine and will pay $50 for a bottle, but others like the taste of $5 wine just fine. As Dave O'Flynn wrote to me:

> I never thought Apple products were worth the premium until I joined Atlassian. 12 months later, I gladly paid a large premium for a Macbook Air. The people I was surrounded with valued design and elegance. Prior to that, I was surrounded by people that valued bang for the buck and my laptop then was a generic AMD that weighed a ton in comparison. I was essentially the same person; the changes were in the expectations and sense of value of those around me.

How much money they have affects their perception of value. Dennis Kozlowski, ex-CEO of Tyco, felt that $15,000 was a reasonable price to pay for a dog-shaped umbrella stand, but most of us don't.

Finally, and importantly, **knowledge** influences the value people place on products hugely. A laptop with a 2.7Ghz quad-core Intel Core i7 processor, 8GB of RAM and a Retina display is worth more than one with an N-Series Intel Atom Processor and a VGA display to me, but not to my mum.

Increasing Perceived Values

The pharmaceutical industry holds another good example of how marketing can increase the perceived value of a product, without changing its substance. In 1981, when Glaxo wanted to release Zantac, their anti-ulcer drug, they faced a marketplace dominated by SmithKline's Tagamet. Although Glaxo felt their drug was more effective than SmithKline's, the US FDA rated Zantac as providing little or no benefit over existing treatments. Rather than marketing Zantac as a me-too product, at a similar price to Tagamet, Glaxo decided to spend heavily on saturating their sales and marketing channels. This ubiquitous promotion increased Zantac's perceived value, and they were able to price the product higher to reflect this added value. By the end of the 1980s, Zantac had knocked Tagamet off its perch as the best selling drug in the world.

Here are some more ways of increasing the perceived value of your product:

Increase its objective value. Perceived and objective values aren't identical, but they're still correlated. As Joel Spolsky wrote in 2006[1]:

> With six years of experience running my own software company, I can tell you that nothing we have ever done at Fog Creek has increased our revenue more than releasing a new version with more features. Nothing. The flow to our bottom line from new versions with new features is absolutely undeniable. It's like gravity. When we tried Google ads, when we implemented various affiliate schemes, or when an article about FogBugz appears in the press, we could barely see the effect on the bottom line. When a new version comes out with new features, we see a sudden, undeniable, substantial, and permanent increase in revenue.

Give your product a personality. 37signals may not sell the best project management software in the world, but it has personality. The 37signals team stands for something: uncompromising simplicity. Want an extra feature? Tough. If you want features, buy something else.

Link your product to yourself, and then define, and promote, yourself as an expert. In its early days, before his company was bought by Symantec in 1990, Norton Utilities and Peter Norton were synonymous. All of Norton's products featured a picture of himself, with his arms crossed.

Make people love your brand. Innocent Smoothies have become huge very quickly by filling their packaging with small, friendly little notes and cute Easter eggs. They make it clear that Innocent is built and run by human beings, and focus on natural, not-from-concentrate juices and smoothies. People love this friendly, congenial approach.

Provide a better service. When somebody buys software, they want reassurance that it's going to work and that you'll be around if it doesn't. If you're a small company with big competitors, this is something you can do better than they can. Capitalize on it.

Create a tribe. Products can be symbols of belonging. If you can turn your product into a badge that people wear to make a statement about who they are, which groups

---

1.http://www.joelonsoftware.com/items/2006/12/09.html

they belong to and those they don't, then that's valuable. When Black and Decker introduced its DeWALT line of drills, it went to building sites and lumber yards at lunch times to hand out pulled pork sandwiches, give product demos and hold drill-off competitions, with prizes. They went to NASCAR races and rodeos, where their end users hung out. This meant a lot of professionals purchased DeWALT drills, and created a tribe around them. Amateur DIYers don't *need* to spend $400 on a DeWALT drill, but they like feeling part of the 'professional' tribe too.

Remind people of how much work you've put into your product. People are more likely to pay for years of your time than for an easily-copied software product. The twenty year old Bill Gates used this technique in his now-famous 'open letter to hobbyists' in the Homebrew Computer Club newsletter in 1976:

> Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds $40,000.

Appeal to people's sense of fairness. When coffee shops charge an extra 10 cents for coffee made with Fairtrade beans, they're lining their pockets with your ethics. How much of those ten cents go to the farmer who originally farmed the quarter of an ounce of coffee beans that went into your Fairtrade latte? Under a penny.

Sell the experience. Starbucks do this brilliantly. At its core, a cup of coffee is just a few beans, worth pennies. If you crush the beans up and put them in a packet on a supermarket aisle, you can charge a few more cents. If you mix it with hot water and milk, you can charge $1.50. But if you place it in a pleasant environment, with comfortable furniture and chilled music playing, you can ask for $3.50. You're not just buying a coffee, you're buying some time in the Starbucks experience.

Ultimately, it comes down to **differentiating your product.** It almost doesn't matter on what – features, benefits, the way that you sell, the service that you provide, the country you're based in – more or less anything will do.

Signposts

Now that you know that customers will find reference points to compare your product's price against, you should do all you can to encourage favorable references and discourage unfavorable ones. If you want to sell a to-do list at $200, when the market price is $100, then you need to add a couple of features so your customers cannot make a direct comparison, and then promote comparisons to other companies' $300 productivity suites, not their to-do lists. At the same time, avoid all comparisons to open source alternatives.

If your customers can't find a reference point for your product, then they look for proxies, or **signposts**. Supermarkets take advantage of this: consumers decide whether luxury ice cream (something they don't buy regularly) is reasonably priced based on whether diet coke (something they buy all the time) is good value. If a supermarket sells a can of diet coke for $4, consumers assume all their other products will be expensive too.

Say you sell two products: the Time Tracker 3000 and the Task List 400, a to-do list application. When somebody thinks of something they need to do, they store it in the Task List 400. Later on, they can prioritize their tasks, split them up into sub tasks, track their progress and smugly mark off the tasks as done.

Let's say the Time Tracker 3000 has no competitors, but the Task List 400 has plenty. Your customers will judge your Time Tracker 3000 price on how you've priced the Task List 400. Charge a reasonable $25 for your to-do list application and customers will take your word that $300 is a good price for the Time Tracker 3000. Charge $1000 for the first app, and they'll assume you're fleecing them on the new one too.

If your product is unique, and customers can find no reference points or signposts, then you have a chance to set your customers expectations, and define their perceptions. If you tell your customers that the Time Tracker 3000 is worth $300, then the odds are they'll believe you. We've already seen how Microsoft did that with the first version of MS-DOS.

If you have **competitors** in your market, then your customers will be more conscious of cost, but if your product **creates a new category**, then early adopters are less likely to be price sensitive. If you can create a teleporter, a brand new category of product, that will beam you, unharmed, from New York to Paris then not only can you

define your price, but you can also raise your price from $20,000 to $25,000 and people will still buy it. But if you create a car, a new product within a category that already exists, and increase your price from $20,000 to $25,000 then your sales will suffer.

Chapter Three

Pricing Pitfalls

So far, we've looked at some economic theory, and the psychology of pricing. Hopefully, you've now got some idea of how to set a price. But there are some other factors to bear in mind too, and some pitfalls to watch out for.

Competitors

When you set your product's price you need to think about how your competitors will react. If you undercut them, will they start a price war? Even if your competitor has a high-cost business model and cannot compete on price in the long term then there's a risk they'll respond in kind if you pose a serious enough threat, and just hope you go out of business before they do.

The airline industry gives the best example of the futility of starting a price war. On September 26th 1977, Freddie Laker's first ever Skytrain flight to New York took off from London Gatwick. The price for the return flight was $238.25 (plus an extra few dollars for a meal), well under half the price of rivals' tickets.

Five years later, Laker Airways was bust, the victim of the vicious, dirty price war that it had initiated. As Laker had found out, and as EOS Airlines (founded 2004, closed 2008), Silverjet (founded 2006, failed 2008) and MaxJet (founded 2003, failed 2007) subsequently relearned, taking on an incumbent on the basis of price is highly risky at best, suicidal at worst, especially when your competitors cannot afford to lose and have no option but to fight to the death.

If you are going to compete on price, then you should minimize the possibility of a counter-reaction from your competitors. Don't bang your drum and tell the press how you're going to destroy them (a mistake that Marc Andreessen of Netscape made when he said “we're gonna smoke 'em”, referring to Microsoft (or “those idiots up in Redmond” as Andreessen put it). Focus on their marginal customers and hope that by the time they notice you it will be too late.

On the other hand, if you set your product price too high, will other competitors emerge? Price the Time Tracker 3000 at $10,000 and you could create the market, only for a competitor to produce the Tyme Trakka 3000, undercut you, and steal your business.

Microsoft is famed for this. They wait for competitors (and often partners) to prove markets with low volume, high price products – whether it's CRM, testing tools or business intelligence – and then jump in with a low-cost, high-volume model.

Fairness

However you price your product, remember that consumers have an acute, although often irrational, sense of fairness. Think twice before you betray that.

Books provide a good example. An economist would point out that I derive the same value (traditional economics is all about value) from reading a paperback version of my favourite piece of erotic fiction 'Fifty Shades of Grey' as I do from reading the electronic version [Jamie wrote this bit, not Neil - editor]. But the list price is very similar (~£7). It's just not fair that short-sighted book publishers charge the same for paper as they do for electrons. I feel screwed over, and I don't like it.

Pirates

If your price is way off whack, you will provide an opening for a special type of competitor: the pirate. Price software too high, or at a price point that most people judge 'unfair', then be prepared to be ripped off in return.

If you're observing widespread piracy of your software, perhaps it's *just too expensive*?

But pirates can also be your friends, in two ways:

Firstly, if your strategy is to achieve world domination by providing a product to every potential customer, at a price he or she can afford, then pirates provide a cheap back channel. They put a copy of your software into the hands of people who will not pay, cannot pay, are too dishonest or too unprincipled to pay, or who simply don't value your work that much. However, a pirated copy will end up, eventually, in the hand of somebody who *will* pay.

That's how early shareware software operated. In the early 1980s, bulletin boards and user groups were a network commonly used to pass around pirated software. In 1982, Andrew Fluegleman and Jim Knopf piggy-backed onto this pre-existing network, added

a notice in their software asking people to pay them if they liked their software, and invented shareware.

The second reason that pirates can be your friends is that they are a bellwether. They indicate the existence of a market failure. Most people aren't natural crooks, but high prices can force them to do things against their better nature. Apple realized that the success of illegal download sites indicated the need for cheap, downloadable music. Their strategy of satisfying people's needs worked far better than the ostrich-like behavior of the music labels.

Switching Costs

If you're trying to persuade people to switch to your product from a competitor's then you'll need to position the price to overcome the switching costs your customers face.

Say you're trying to persuade a customer to switch from his garbage $500 word processor to your superior $100 one. First of all, you'll need to price to overcome the economic switching costs. It'll take him time, and therefore money, to convert his files to a new format and to learn the new menu layouts.

Secondly, you'll need to overcome the **psychological** switching costs. People overvalue what they have, and undervalue what they don't have. I purchased three balsawood penguins from the Kontiki exhibition in Oslo. They cost me ten dollars. Care to raise your price? Didn't think so, but I wouldn't sell them to you for even a hundred. I bet your house is probably full of similar junk too.

Another powerful psychological factor people struggle to overcome is the emotional attachment to money they've already spent. Rationally, it's gone. It's a sunk cost. Your customer shouldn't care that he's already spent $500 on his garbage word processor. But he does.

There are some things you can do to mitigate switching costs, and even to use them in your favor. Here are a couple of examples. Open Office, which includes open source word processing and spread sheet applications, lets you open files saved by Microsoft Word. Early versions of Microsoft Word not only opened WordPerfect files, but had a dedicated section in the help for WordPerfect users, and even allowed you to use the WordPerfect shortcut keys.

Here's another example. If you decide to stop using FogBugz within ninety days of your free trial expiring, then Fog Creek will refund you all the money you've spent, no questions asked.

These strategies have two effects: first of all, they reduce the psychological and economic impact of switching to Word or FogBugz. Secondly, once you have switched, you'll have invested time and energy into using the new software, and will have incurred a whole load of new switching costs, which will then stop you from switching back.

Should You Take Your Costs Into Account?

Clearly, you cannot price your software for less than it costs you to produce, and sell, each unit. These are your marginal costs. You might think these costs are zero, but they are not.

You need to find potential customers and persuade them to buy. If you have a sales team then you might need to pay them commission. It will cost you money to support customers, and chase the customers who don't pay.

If you're planning on not charging the majority of your users, then think very carefully about the cost of each additional user. If you think it is zero then you are almost certainly wrong. If you're running a web site, then each additional user will cost you storage space, CPU cycles and bandwidth. This might be a very low cost – fractions of a penny, even – but if you need huge numbers of users to make money then small costs multiplied by vast numbers can equal big outlays.

Take YouTube. It's a free service and, theoretically, supported by advertising. The cost of serving each additional video is tiny but YouTube see 4 billion video views globally every day. This means that in 2012 it will serve up an estimated 1.5 *trillion* video streams. Multiply together the tiny cost and the large volume and you can understand why YouTube costs Google an estimated $1.5 billion a year to run. It nowhere near covers its costs through advertising revenue.

Also, don't forget about support and customer engagement. With the rise of social networks as legitimate business tools, the cost of looking after your users in this space becomes important. If your website goes down, you may need to respond to customer's

complaints one-by-one on Twitter. If somebody's sharing information about your software on their Facebook account, you will want to jump into the conversation and ask if you can help any further. These interactions cost time, and ultimately, money.

Paperback Software offers another example of how misunderstanding how your software is sold, and failing to account for your costs, can lead to catastrophe. When Adam Osborne set up Paperback Software in 1984, it was founded on the premise that software cost too much. They released VP-Planner for $99.95 in 1986 and marketed it directly against the $500 Lotus 123. Back in the 1980s, most software was sold through dealers. The dealers earned a commission for every piece of software they sold, but so hated the low margins on the low cost VP-Planner that they bad-mouthed it and encouraged people to buy the high-cost, high-margin Lotus 123 alternative. Furthermore, since VP-Planner was essentially a direct copy of Lotus 123, customers demanded as much support for the cheaper product as the more expensive one, destroying any profit that Paperback Software made. Paperback succeeded in harming Lotus's market share, but failed to earn enough money to defend themselves against the lawsuit that Lotus launched.

If the price your customers are willing to pay is lower than what it costs you to sell your software, then you haven't got a business and your product will flop. You need to cut your cost of sales, or change your pricing mechanism so customers end up paying more over the lifetime of the product.

When Panasonic launched the 3DO, its gaming console, in 1994, Time Magazine nominated it its product of the year. With a 32-bit RISC processor, custom math co-processor and 2MB of RAM, it was far ahead of its time. But Panasonic priced it at $699, way above its competition and much higher than what even its target market of early adopters could bear to pay. That, combined with muddled marketing, caused it to bomb.

Other games console manufacturers learned from this mistake. When the PS3 and Xbox 360 were launched, they cost more to produce than the selling price that the market could bear, so Sony and Microsoft charged consumers a low price, and accepted that they would lose money (up to $300) on each console sold. They then recovered the revenue through royalties on games people bought. The real price of the console is hidden; buried in a clever pricing model.

With the Wii, Nintendo took a different approach. They wanted to reach a much wider market than their competitors' 18 - 35 year old male sweet spot, but realized that older people, housewives and families would pay less for a console than hardcore gamers would. So they cut their cost of manufacture and used cheaper, slower components. When the Wii was launched in September 2006, Nintendo made a profit on every console sold. That made the games cheaper to produce too, since royalty payments can be lower, but not necessarily cheaper to buy. Why? By now, the answer should be obvious to you.

Your pricing may need to change over the course of your product's lifetime. In the early stages, you need to attract new users and convince them that your software is the one for them. Lower prices and big marketing efforts are usually crucial to accumulating a community.

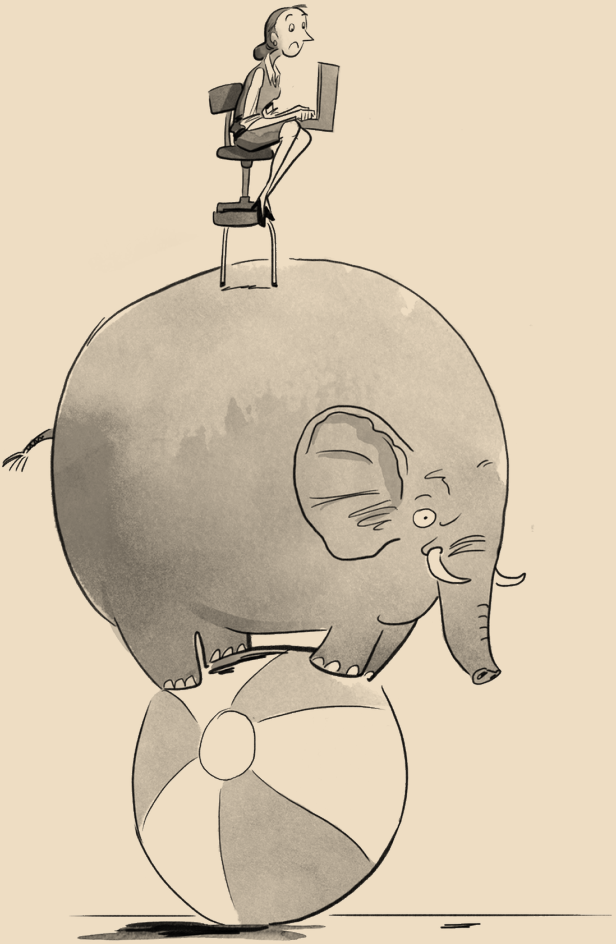
As your product gets maturer, your customers will be on the lookout for more variety and options. By this point they'll have been using your software for a while and will understand its features and how they fit it into their workflow. Providing them the opportunity to customise their experience will make them evangalise your product even more. We'll talk more about this in Chapter 4.

You'll notice that there's one factor I've not mentioned, and that's how much your product has cost to develop. So far I've talked about marginal costs – how much it costs to produce, or sell, each additional unit of your software. Your up-front cost is different. You might have spent one hundred dollars developing your product, or a million, but that money is all spent. Gone. It's a sunk cost.

What matters now is not how much you've spent, but what people are prepared to pay.

Chapter Four

Advanced Pricing



Up to now, we've considered selling single products. But what happens when you have several products to sell, or sell multiple versions of the same product? Or even, what if you're selling a service rather than a product?

Versioning

Each of your potential customers has a price they'll buy your product at. Revisiting our previous example, Belinda (the bargain hunter) and Stewart (the student) will only use the Time Tracker 3000 if it's free. Willhelm will pay up to $150 and Pat's maximum price is $400. Let's say Ernest will pay up to $600.

Here's a chart of the revenue you'll make at each price point:

Pricing	***Who buys***	***Revenue***
$0	Everybody	$0
$150	Willhelm, Pat, Ernest	$450 (three people @ $150)
$400	Pat, Ernest	$800 (two people @ $400)
$600	Ernest	$600 (just Ernest)
$1000	Nobody	$0

If these five people are your entire target market, then, to maximise your total revenue, you should price the Time Tracker 3000 at $400. It's the best single price, but you'll lose out on sales to Willhelm, and you'll lose out on the extra revenue that Ernest would have paid.

If there had been some way to sell the product to each customer at the maximum price that they could afford to pay, you would have been able to sell $1150 of software. That's what's versioning is about. It's a mechanism of segmenting your users according to their willingness to pay. You figure out if you can group your customers in different ways, and then see if those groups are willing to pay different prices for your product.

Here are some of the ways of doing it:

- **By feature.** For example, you can have 'standard' and 'pro' versions of tools. This is extremely common in the software business. Microsoft's Visual Studio 2008 came in five different versions: Express (free), Standard ($299), Professional ($799), Team System ($5,469) and Team Suite ($10,939). That's a price for everybody, with features to match, from the cash-poor hobbyist to the rich, blue chip enterprise developer. In the Time Tracker 3000 example, you might create a professional edition that lets people compare how their usage of different products compares with other people doing similar work.
- **By availability.** Some of your customers might be prepared to pay more to get your product quickly. Hardback books are a good example of this. They have the same content as paperbacks, but are packaged differently and aimed at people who cannot wait for the content. For the Time Tracker 3000, you could sell an additional subscription service that gets customers early access to software.
- **By demographic.** Students have less money than businesses, hobbyists than professionals and school kids than baby boomers. You could provide a version of the Time Tracker 3000 which students could get, but only if they prove they're in full time education.
- **By geography.** Customers in the USA will pay more for the same product than those in India and China. Microsoft, to compete with the threat of open source, provides a cut-down 'starter' edition of its Vista operating system, available only in poorer countries such as India and Mexico. The Time Tracker 3000 might be available in India for 10% of its US cost, but be localized into Hindi, rendering it useless to Westerners.
- **By industry.** Perhaps architects, or software developers or aircraft designers have specific needs, and perhaps your software can be customized to suit

them. The Time Tracker 3000 could come in a special edition, aimed at law firms, that not only tracks application usage, but also bans certain applications.

- **By platform.** Mac users might be willing to pay more money for your software than Windows users, or vice versa. You could sell a Time Tracker 3000 for the Mac at a higher price than the Windows version.

Of course, you need to be aware of the dangers of versioning too. You need to make sure that the features you choose for each version appeal to the segment you're targeting. For example, if you introduce a 'Lite' version of your product, you need to be sure that professional users won't downgrade to it.

When attempting to version by one of these criteria, and if your goal is happy customers, then it's best to remember consumers' keen sense of fairness. Adobe attempt to version on geography; their Acrobat 9 Pro cost $449 in the US, but £445 ($750) in the UK. Economically, this might make sense, but it still leaves me banging my keyboard in impotent rage. And is that good, in the long term, for Adobe?

Versioning has a couple of subtleties. Take a fast food restaurant that serves the following sizes of diet coke:

Product	***Price***
Small	$1
Medium	$1.50
Large	$2

These prices have been chosen, presumably, to maximize the fast food chain's profits. People with little money, or who aren't very thirsty, buy the small drink; those who are marginally thirstier buy the medium one and very thirsty people buy the large one. The additional fluid ounces cost the restaurant virtually nothing: this is all about finding a price point that works for everybody.

You can also see the use of reference points here. Consumers see the 'small' drink, and consider the 'medium' drink a bargain (a lot more drink for just a few more cents).

So far, so blatant, but here's one subtlety: adding a 'jumbo' drink will increase the sales of the 'large' drink, even if nobody ever buys the 'jumbo' one. Adding more choices at the edges drives people to the middle of the range. They don't want to appear stingy, or greedy, so go for the safety of the middle. In this example, adding a 'jumbo' version on top shifts where the middle lies, so makes more money.

Here's the second subtlety: this only works if people can easily compare the products being versioned. For the sodas, it works. The jumbo soda is clearly larger than the large soda, which is clearer larger than the medium soda, which is clearly larger than the small soda. So people go for a safe option, somewhere in the middle.

But the effect reverses if people struggle to compare the different versions of the products. In that case, people flee the middle and head for the extremes. Take laptops. Say you ask people to choose between the following products:

Laptop type	***Features***	***Price***
‘Standard’ laptop	Normal features	$1000
X100	Standard + DVD player	$1100
X102	Standard + Wireless card	$1100
X103	Standard + Faster processor	$1100
X104	Standard + DVD + Wireless	$1200
X105	Standard + Wireless + Faster processor	$1200
‘Extreme’ laptop	Standard + DVD + Wireless + Faster processor	$1300

Rather than migrating towards one of the middle options, people are pushed towards the edges. They go for the 'standard' laptop or the 'extreme' one. This is because it's impossible to compare the benefits of the different items being offered. Is a wireless card a better option than a faster processor? Or how about a DVD drive? As a result, people take an easy “all or nothing” decision.

When people are presented with a bunch of confusing options they cannot compare, going for an extreme isn't their only option. They also have a tendency to defer: to simply not buy, or go for a competitor's product.

This counter-intuitive behavior has some interesting consequences. If consumers are faced with a choice of, say, a Sharp or a Panasonic microwave then roughly half of them will plump for a Sharp and half for a Panasonic. If they are asked to choose a microwave from a selection that contains a single Panasonic and multiple versions of the Sharps, then one of two things can happen.

If they can easily compare the Sharps (for example, because they differ solely in price and one other attribute, such as size or power), then more people will buy the Sharp than the Panasonic. This is a demonstration of how providing multiple versions of a product will increase the product's sales.

On the other hand, if they cannot easily compare the Sharps then the effect is reversed. For example, if one Sharp has an adjustable speed turntable, another has a moisture sensor, one has programmable menus and another has a 'hold warm' feature, then consumers will shun the Sharp, reject confusion and go for the Panasonic. This shows how providing multiple versions of a product can decrease a product's sales.

Here are just some of the possible versions of Microsoft's Vista operating system:

Features	Home Basic	Home Premium	Business	Ultimate
Suggested Retail Price (MSRP)	$199.95	$259.95	$299.95	$319.95
Most secure Windows ever...	✓	✓	✓	✓
Quickly find what you need...	✓	✓	✓	✓
Elegant Windows Aero experience...		✓	✓	✓
Best choice for laptops...		✓	✓	✓
Share documents and collaborate...		✓	✓	✓
Use a secondary screen on your mobile PC...		✓	✓	✓
All-in-one media center functionality...		✓		✓
Protect against hardware failure...			✓	✓
Scan, fax, and receive documents and images...			✓	✓
Automatically back up your files...		✓	✓	✓
Remotely access your business resources...			✓	✓
Easier networking connectivity...	✓	✓	✓	✓
Better protect your data against loss...				✓
Easily make DVDs...		✓		✓
Have more fun on your PC...		✓		✓
Create high-definition movies...		✓		✓
Additional programs and features...				✓

Is protecting against hardware failure more important than having all-in-one media center functionality? And does being able to remotely access your business resources outweigh being able to easily make DVDs? It's hard to tell, so consumers will tend to do one of three things:

- Go to the extremes – buy Home Basic (at $199.95) or Ultimate (at $319.95)
- Defer a decision – stick with Windows XP
- Buy a competitor's product.

I bought a Mac – my first one ever.

Bundling

Bundling is another way of giving your customers better value, persuading them to buy and generating more revenue. Most straightforwardly, people love a bargain.

The idea of getting $5140 of software for $1,595 (in the case of the SQL Toolbelt that Red Gate sells) is clearly compelling.

But even without price discounting, bundling makes sense.

Say you've got two products, the Time Tracker 3000 and the Task List 400. Willhelm, the web start-up founder, is hyper-focused. Once he gets going on something, he'll see it through to completion. But he struggles to organize the list of things he has to do. Pat, on the other hand, doesn't see much point in task lists, but she has the feeling that she wastes much of her working day, and would like to know how she spends it. Pat and Willhelm are therefore willing to spend different amounts on each product.

If you sell your products individually then, product-by-product, you need to choose the maximum price that the person who values that product the least will still pay. You need to price the Time Tracker 3000 at $150 (so Willhelm will buy it) and the Task List 400 at $150 (so Pat will buy). That means that Willhelm and Pat will each give $150 for each product, and you generate revenue of $600.

Let's say you create a bundle of the Time Tracker 3000 and the Task List 400. Willhelm and Pat will both pay $400 for the product they need (Task List 400 and Time Tracker

3000, respectively). At that point, the bundle is worth $550 to Willhelm and $550 to Pat. Set the price at $550, sell the bundle to both people and you generate revenue of $1100. Willhelm and Pat have got all the software they want, and you've generated an extra $500.

However, bundling has drawbacks too. When you bundle software together it becomes harder for your customers to understand what they're paying for. In turn, that might mean they are less likely to use it.

For example, a diner eating a fixed price menu is more likely to skip coffee than a diner who's paid explicitly for the coffee. The coffee is bundled, so the disconnect between what the diner is paying for and what he is consuming makes it easier to not consume.

For software, if a customer is less likely to use a piece of bundled software then he might be less likely to buy a future version, or to continue to spend money on maintenance contracts. One way of counteracting this effect is to continue to be explicit about the worth of each item in a bundle.

Multi-user Licenses

Multi-user licenses are one more way of bundling software. But before you decide to offer multi-user discounts to your customers, remember three things:

1. Larger companies tend to buy more copies of software since they have more users. Offer them a three for the price of two discount then they'll get a better deal than individual users and small businesses. Larger companies also have more money and tend not to be so price sensitive. This means that the poor are effectively subsidizing the rich.
2. You might lose sales in the long term. The company who paid for two licenses may have paid for three if you'd asked.
3. On the other hand, they might not have done. Everybody likes a discount, even large companies.
4. Larger companies might have more money, but they can also have stricter purchasing policies.

It's hard to know which of these factors are the strongest in any given situation. I know of one company who moved away from multi-user deals based on the first two reasons

above. They moved from selling a 'five for the price of three' bundle to a simple 10% discount per copy, for multiple copies.

It turned out that their customers preferred the convenience of buying multi-user bundles. If their customers had two users, they liked being able to buy a five user license for the price of three and get the possibility of a bargain–two 'free' users–if another person started using it. This outweighed the risk of overpaying, and never having a third user. Two months and several hundred thousand dollars of lost revenue later, the company switched back to multi-user deals.

The important point is that theory cannot tell you about the wisdom or otherwise of multi-user deals. The only way to find out is to try it out.

Site Licenses

You need to be careful with site licenses. Sell a site license to Microsoft or Walmart and, unless you've customized your pricing accurately and high, you could be forgoing enormous amounts of future revenue. If you insist on selling a site license then make sure you define 'site' well. Is it for a specific office, or country, or worldwide?

The Purchasing Process

You must consider your customers' purchasing process when you set your prices. If you're selling to businesses, then there will be a number of thresholds that you need to think twice about before crossing. For example:

If you sell a product at **$10** or under then an end user will charge it to his personal credit card and not claim it back.

Up to **$50**, he might charge it to his card and claim it back from the company he works for.

Up to **$995**, he might borrow his boss's company credit and charge it directly to the company.

At **$1000**, he might have to fill in some paperwork and justify, strongly, his reason for purchasing to his boss.

At **$5,000** he might have to talk to the head of his department.

At **$25,000** he might have to talk to his CEO.

At each stage, not only does the cost increase, but the hassle does too. If you can figure out where these thresholds lie (and they move around as the state of the economy changes, and according to the characteristics of your customers), then it's worth pricing your software just under a threshold rather than just over it.

Once you cross a threshold, you can often move up to the next one relatively easily. It's easier to persuade somebody to spend $10 instead of $1 than it is to get them to open their wallet in the first place.

This is yet one more reason to provide multi-user discounts and bundles. If you're selling to an organization, then the individual you're selling to will help you cross the thresholds, and once you're past a threshold he may even be keen to help you beyond there.

Say you've persuaded Frank, the IT manager of Blue Door Software, to buy a one hundred user license of the Time Tracker 3000. Frank has negotiated hard and you've agreed on a price of $25,000. Frank knows that he now needs to persuade Victor, the CEO, to authorize this expenditure. Victor is a scary, busy man and hard to persuade. Frank realizes that he may well need some copies of the Task List 400 at some point in the next six months, and doesn't want to have to persuade Victor twice. He also knows that, although Blue Door Software currently has one hundred employees, it will probably grow over the next twelve months. If he's going to ask for $25,000 to buy the Time Tracker 3000, why not ask for $30,000 and get you to throw in some copies of the Task List 400 for free? Or for $35,000 and ask for an extra fifty licenses? It's in Frank's interests, and yours, and Blue Door Software's.

Free

Some people argue that the price of software will inexorably be driven to zero. Economists have proven that in any efficient market, the cost of a good will be driven

down to its marginal cost of production. If you're one of many producers selling wrenches then consumers will shop around for the cheapest wrench. If it costs $5 to produce the next wrench, then wrench manufacturers will compete on price, undercutting each other and driving the price lower and lower, until it's at the lowest price that still allows them to make a profit: $5.01.

Information, the theory goes, has zero marginal cost. It costs nothing to ship the next set of bytes to your next customer. Therefore, the price that consumers will pay for your information, and the cost you must sell it for, will eventually approach zero. The success of open source operating systems such as Linux, the Apache web server and the Open Office suite seem to illustrate this point.

This argument has a number of holes in it. For a start, as already discussed, you are not just selling bits and bytes. You're selling a whole bunch of stuff around it, including support, documentation and hand-holding. Your customers are buying man-years, decades even, of your past, present and future blood, sweat and tears. Is that worth $100? Or $1,000? Heck, yes, and you should tell that to your customers.

Secondly, there is no such thing as a commodity. Or, more accurately, there need not be such a thing as a commodity. Your job is to de-commodify what you are doing. If your potential customers consider your to-do list, or your word processor, accounts package, web site or iPhone app as just one of a hundred indistinguishable others, then the price you can charge will be driven ever downwards. You need to figure out a way to either make it stand out, or impossible to compare.

If Starbucks can de-commodify coffee and charge $4 for coffee beans and hot water, if Stormhoek can de-commodify grapes (the only wine maker I know of who sells branded G-Strings), and if Perrier can de-commodify water, then you can certainly de-commodify the complicated software application that you have created.

Despite all this, there is no doubt that 'free' holds a tremendous power over consumers. And it's a power that you can harness.

Free Trials

Free trials let your customers try out your software for free, to make sure it fits their needs before they buy it. They don't even need to use the trial for you to benefit. The

mere fact that customers could try out your software, if they wanted to, transmits a strong signal about its quality.

When customers do try out your software, it can increase its perceived value. In a famous psychology experiment, people who were able to hold a coffee mug were willing to pay significantly more for it than those who were just allowed to see it. People start to feel that they own an object before they buy it if they're allowed to use it and, as we've already seen, people value what they own more than what they don't.

Free trials aren't always possible. Red Gate used to sell a tool that let you recover deleted data from a SQL Server database. The free trial worked against it: people would download it and recover their data before their free trial expired. Free trials only work for software that people use again and again, and where the free trial doesn't fix the problem by itself.

Similarly, if people require a lot of hand-holding to use your software, or if it is of a low quality, then free trials are unlikely to work.

The freemium model involves providing a free version of your software for some people, and a paid-for version for others. Typically, the 'standard' product will be free, and the 'pro' version will be paid for. Flickr, LinkedIn and Skype all use this model.

However, it's not clear that giving your software away for free is a great way to make money, despite being extremely fashionable. At the very least, you need to be careful, and make sure the free version is good enough to be useful, but not so useful that it cannibalizes paid-for sales. It can also require extremely high volumes to make it work. Flickr only manages to upsell around 5% of its standard users to its professional account. And storing, searching and serving the 3.5 billion images Flickr's free customers store certainly isn't cheap.

In 1993, the UK mobile telephony market was heating up. One2One, a fledgling mobile telecommunications company backed by Cable & Wireless and US West, decided that free was the way to go and offered free off-peak local calls to all new customers. The network was soon overwhelmed as thousands of customers tried, and failed, to get through, for free, on Christmas day. One2One quickly gained a reputation for unreliability, losing nearly a million dollars along the way.

Flickr has Yahoo, and One2One had Cable & Wireless, but if you adopt the freemium model without a sugar daddy, then beware.

Network Effects

There is, however, one situation where free is the best price for your product: where there are strong network effects.

Network effects occur where the value to your customer of using your product increases as the total number of users increases. For example, the value of using a telephone increases as the number of people you can call increases; the value of a social network increases as more people join, the value of e-mail as more people get accounts, and so on. In these cases, you get a feedback loop: more people use your application, it becomes more valuable and more people join, and so on.

Free becomes even more important when your networked product has competitors. In this situation, it turns out there are two stable situations: no customers, or plenty of customers, and that there is a critical point beyond which user numbers accelerate quickly. Get past the tipping point and your user base will accelerate rapidly. If you don't quite reach the tipping point then your user base will shrink back to zero.

If you study Twitter's traffic stats, you can see there's a clear tipping point at the beginning of 2009. Look at the uptake of the fax machine, the telephone and other inventions that rely heavily on network effects and you'll see a similar pattern.

It becomes, therefore, extremely important to reach the tipping point as quickly as possible, and the 'free' price point is a good way of doing that. Of course, once you're past the tipping point you'll need to make money from your product, without losing users.

Bargains

Bargains are closely related to free: people like getting something for nothing. Bundling is a type of free. When you buy Windows, you get Internet Explorer for free. The SQL Toolbelt gives you 12 applications worth a total of $5140 for only $1595. That's $3545 for free.

Put a 'sale' price on one or two products on your web site, and people will assume that they are, in fact, getting a good deal. But put 'sale' on all your products and people will assume you're taking them for a ride.

To work best, bargains should be limited to specific products, or specific times. When Steam, the online gaming community, held a sale on third party games over the holiday season in 2008, a 10% discount led to an increase of 35% in sales (in dollars, not units). A 25% discount led to a 245% increase; a 50% sale to a 320% increase and a 75% discount to a 1,470% increase.

If there's something people like more than getting a bargain, it's getting a bargain and feeling smart. Just before Christmas 2006, Threshers (a UK wine merchant) offered its suppliers and friends a 40% discount if they turned up at any store with a special voucher.

When this voucher was “accidentally” leaked onto the web (on the Stormhoek web site - we've already met them - and promoted by Hugh MacLeod), word spread like wildfire. Eventually, millions of people downloaded it. Threshers felt obliged to honor the voucher. Their customers felt smart and got cheap wine while Threshers made a killing and promoted their brand.

Apps and App Markets

The phenomonal success of the iPhone–and the smartphone revolution that has followed in its wake–has brought a brand new way of purchasing software to the table. Instead of approaching vendors' websites directly, or heading into a bricks-and-mortar store, consumers are now searching and downloading applications from an app store.

Deciding to sell your software on an app store isn't always an easy decision. If you're writing software for iOS, your only choice is to sell through Apple's App Store. For other platforms, there are other options, but all have their own restrictions and fees.

The most immediate concern is the commission each store will charge you for selling your app through their service. Most app stores will take a 30% cut, which is a large chunk of your revenue. You may need to weigh up the extra audience against this fee, reduce your marketing budget, and calculate if it's worth it.

The biggest issue when pricing, however, is the **lower percieved value** of mobile apps. If we decide to redesign the UI and put Task List 400 into the iPhone App Store, we instantly devalue our app. We know Task List 400 iPhone is a great application, and we're selling the Windows and OS X versions for $300 a piece.

But it's the customer's perception of the app that will ultimately dictate the price.

Mobile apps are commoditised. The 'throwaway' nature of lots of apps, combined with the sheer number of applications available for smartphone platforms, means that most apps have very low price tags and it's certainly a low-price, high-volume game.

Perhaps we've spent $10,000 creating Task List 400, and we want to see it make a minimum of $80,000 (a good return, no?) As of 2012, the average app price is $1.92.

How many copies of Task List 400 iPhone do you think we can ship at $1.92? 10,000? 20,000? 50,000?

Quantity	***30% Fee***	***Profit***
10,000	$5,760	$13,440
20,000	$11,520	$26,880
50,000	$28,800	$67,200

We're still not even close to the $80,000 return we want. And selling 50,000 copies of anything is *very* difficult. Let's raise the price slightly, to $2.49. The higher price will mean less people will want to purchase a copy, so let's say we can only sell 30,000 copies (still a terrific number):

Quantity	***30% Fee***	***Profit***
30,000	$22,410	$52,290

Nope, still not there. It doesn't look like we're going to be able to make this $80,000 after all.

The sad fact is, it's *very difficult* to shift such large quantities of applications. If you are confident that your application is unique and provides real value to your customers, you could price your application a little higher than the average and measure the results.

A January 2012 report from Distomo about running sales on apps reveals some interesting information about app prices. On the first day of the sale, the average revenue increase by 41% in the iPhone App Store, and after two weeks, 22% in total. On the iPad App Store, the day one effect was even greater: up 52%.

In general, the report finds that a 'perceivably large' discount had a much better effect: an app dropping from $1.49 to $0.99 had a much greater effect than $7.99 to $5.99.

When you look at the actual figures, you'll find it can be difficult to make a lot of money by selling apps. It really is a quantities game. Spend time working out how many copies you can realistically sell and price your app accordingly. The lower your initial production costs, the easier it will be to make money.

A side note: don't forget that Apple's motives and your motives do *not* align. Apple makes money out of a combination of a few blockbuster, headline-grabbing, successful apps and a massive long tail of apps which individually only bring in a few hundred or a few thousand dollars but when aggregated are valuable. To *Apple*. They really don't care how *your* particular app does. It's a casino, and in the casino it's the bank who wins.

Software As A Service

We've talked a lot about how to price software that is sold in a packaged-and-distributed form; buy a copy and download the executable. Things get more complex, of course, when dealing with multi-user licenses and new / different versions, but the essential form of software stays the same.

An alternative route taken by a lot of companies is to sell their Software As A Service (SaaS). This changes some of the details of software pricing, but the principles of psychology and economics are fundamentally similar.

There are plenty of benefits of this model beyond the obvious one of recurring revenue:

- Paying lots of small amounts is psychologically easier than paying one large amount. That's why people buy cars with credit cards and pay it back at 20% interest, or place the cost of holiday that's over in a week on top of a mortgage that will last 25 years. Although the total amount paid is larger, it somehow feels smaller.

- If you're selling to businesses, then your end user will find it easier to justify a small, regular payment to his boss than a single large, one-off payment.

- Recurring payments promote regular usage. Take members of health clubs. Those who pay a one-off annual fee tend to use the club intensively for a few weeks after their hefty payment, but then stop using it. The usage pattern of people who pay quarterly displays a sawtooth pattern, peaking shortly after payment and then declining until the next payment. People who pay monthly show a steadier, higher usage. Importantly, since they are more regular users, they are also more likely to renew membership and stay members longer.

A lot of the same techniques for pricing still stand: look at your competition's pricing, ensure your costs are covered and you make a good margin–or a small margin with high volume!–and appeal to your customers' sense of fairness. The biggest difference is that packaged-and-distributed software is sold for a one-off fee; SaaS products have a recurring charge that reaches into the future.

There are, however, a couple of implications to SaaS pricing. The first is that it requires more capital than selling a one-off software licence. Let's say you've been selling your Time Tracker 3000 for $200. It costs you $100 to acquire a customer, so you make $100 profit for every customer you sign up. You're getting 100 new customers a month, so you're making $10,000 a month.

You've decided that actually the future of time tracking is as an online service. You've done your research, and you're going to charge $20 a month for the service. On average, you think a customer will use your service for 24 months, so the life time value of a customer is nearly $500. Sounds good, but what will happen *tomorrow*, when you stop selling the Time Tracker 3000 and start selling the Time Tracker Web Edition? Let's say you're still getting one hundred new customers a month. But they're only paying you $20 a month. What's worse is that your customer acquisition cost is still $100, so in the first month you're losing $80 for every customer you sign up. In other words, rather than making $10,000 profit like you did last month, you've now made an $8,000 loss. Of course, in the long term the numbers are going to work in your favour, but in the short term monthly pricing is going to hurt you.

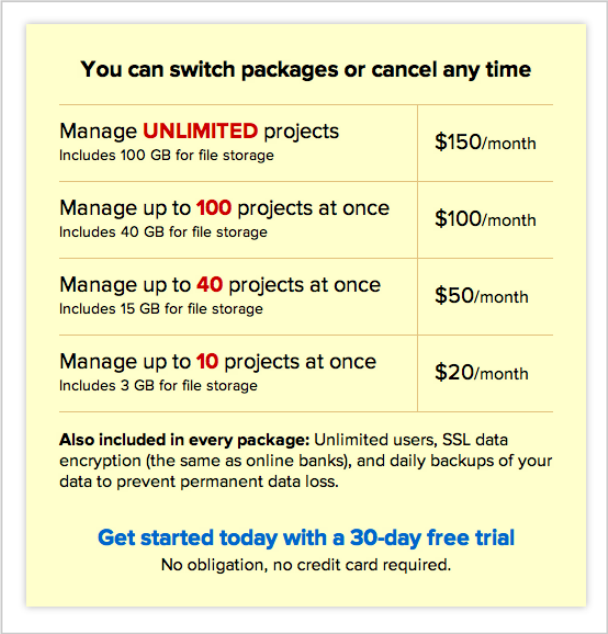
Your profitability is also going to depend heavily on metrics such as the lifetime value of a customer and the cost of customer acquisition, so you need to track these closely. The first number - how much a customer is worth to you over their lifetime - depends on how likely a customer is to stop using your service in any given month. This means that you need to spend more effort in making sure your customers' needs are satisfied over time. Caricaturing the argument somewhat, if a customer who has paid $200 for the Time Tracker 3000 hits a bug, he's already given you the money, so hey. If he hits the bug on the Time Tracker Web Edition after two months then he's going to cancel his contract. Arguably, this is a very good discipline to have, but it does mean that you need to price your product at a point where you can afford to support your customers and improve it based on their feedback.

How, then, do you go about pricing a SaaS product?

At the very least you need to charge enough to cover your costs. How much does it cost to acquire a customer? How long do you think customers will use the system before? Is there a cost to setting up and maintaining a user in your system?

Beyond this, the other factors we've already discussed come into play. How do your customers perceive the value of your service? What will they use as reference points? Can you provide different versions of your service at different price points?

Take the following example from kings (and queens) of Saas, 37signals's project management tool, Basecamp:



Their pricing options are simple and elegant. The tiers get progressively more expensive as your usage of the application goes up. You're not caught in the same situation we saw Microsoft Vista in earlier; you know exactly what you're paying for and what the difference is.

Fixed Contracts

Some companies tie their users in to 12- or 24-month contracts, akin to phone companies. This can make sense from a business point of view (you get a guaranteed lump of cash) but will suck for consumers (what if they want to leave?)

A better alternative is to offer discounts to users who subscribe for 12 months. Offer them 2 months for free, maybe, or a 10% discount. This is win-win: it encourages users to pay a wedge of cash in advance, so you get a year's worth of capital up front, and they have the option to pay monthly if they so choose.

Offering a yearly plan can also be a benefit to customer retention. Keep track of the average retention of your customer - the amount of time they keep their account. You may observe a natural attrition where customers eventually stop using your application. If you're charging monthly and realise that your average retention is nine months, it may be worth offering a discounted yearly plan. This will give you three months of extra revenue you wouldn't have had before.

Yearly billing may also work well if you're selling to corporations with complex approval processes - they simply may not have the systems in place to easily pay for services monthly.

Adobe follow an interesting path with the Creative Cloud tools. This offers their Creative Suite - a bundle of tools that you can buy for a one-off cost of $2599 - for a monthly fee of $49.99 if you commit to subscribing for 12 months. If you prefer to pay monthly with no contract then the cost is $74.99. This is interesting on several levels. First of all, Adobe are assuming that they will increase the number of people who use their tools. I had personally purchased Photoshop (a one-off cost of $1,299) but would probably never have upgraded to the last version or bought Premiere Pro (for a one-off cost of $799). I have, however, subscribed to Creative Cloud. I decided to go for the annual contract at a monthly fee of $49.99. I figure I can afford that each month (a reference point: it's less than my phone bill), and I can cancel after a year if I want to. Of course, I won't - I'm too lazy and I'll be too tied into Adobe tools by then. It turns out that I have actually used Premiere Pro a couple of time as well. Given the up-front costs of providing SaaS (see above), this was a gutsy move from Adobe, but in the long term they're going to make more money out of me, they're probably going to attract more customers, and I'm happy with the service that I'm getting. Everybody wins.

Pay As You Go

Your customers may not use your application regularly and be unlikely to pay for for a time period they just won't use. In these scenarios, having a pay as you go price plan is ideal.

A great example of this is email marketing and distribution service Mailchimp. Mailchimp provide a pay as you go pricing tier for their customers that don't want to commit to a monthly or yearly subscription.

You prepay your account with credits, and then use the credits to send out emails. This is perfect for when you use the service infrequently, and is bound to attract a whole new range of customers that otherwise wouldn't use Mailchimp.

It's clearly more expensive, but is a great way of using the system infrequently, or, indeed trialing it with a view to subscribe monthly. Which leads me perfectly on to...

Free Trials... Again

Much like in packaged-and-distributed software, some online services offer trials to customers. This is usually in the form of a free month: 30 days of unrestricted access to the service before requiring the user to enter their credit card details.

This can have a number of benefits:

- Customers are actively engaging with your system, so you have a real chance to show them how brilliant your software is
- You can collect their information for marketing later (as long as you provide an unsubscribe link!)

Whether you like it or not, customers are most likely going to compare your application to your competitors'. If you offer a trial you are stepping ahead of the game and giving yourself the best chance.

After all, if you're comparing a system you can touch and play with versus some screenshots, what would you prefer? As long as your software is good, and you're targeting the right market, you're bound to have a good conversion rate between trial and paid account.

Different Ways Of Pricing

The amount you charge for your product isn't the only decision you have to make. You also need to decide how you want to charge. There are plenty of models:

Subscription. We've already discussed this in the SaaS section.

I've already covered the **Freemium** model, where a small number of paying customers subsidize the majority of freeloaders. The *Gilette* model is a twist on the Freemium model. Gillette famously sells their product in two parts: the razor and the blades. The razor is cheap, but they make their money on the blades. This strategy is surprisingly common. Adobe follows a similar strategy with Acrobat. It's free to read documents, but you need to pay to create them. Hewlett Packard loses money on its printers, but makes it back on the ink. The first Ford Fiestas were sold at a loss, but Ford recovered the money on spares and finance. Microsoft and Sony lose money whenever they sell an X-Box or PlayStation, but make it back on royalties for games.

There are many ways of **pricing per user**. Common schemes include licensing per named user, or concurrent user. At Red Gate, we license per user. If you have a team of ten people, all of whom want to use our software, then you need to buy a ten user licence. If you can't count the total number of users, or if only a few use it at a time, then pricing by concurrent user can make sense. This model is often used for server-based software, such as databases.

Another common licensing model is **per processor** or **per processor core**. The obvious drawback of this model is that processors get faster, and get more cores, quickly. If, say, you're selling a bug tracking system that's tied to the physical power of your customers' hardware then Moore's law dictates that they will get double the benefit of your software every two years, without paying you a penny.

The **per physical / virtual server** licensing model has the same drawbacks as the per processor model. As more processors are crammed into physical boxes, your customers get exponentially increasing benefit for a fixed cost.

The **per usage** model involves charging users based on how often they use your software. This could be per megabyte stored, transaction processed, gigabyte transmitted, or many other options. Historically, this has been less common than other models but will become more usual as cloud computing takes off and people expect to pay for computer usage on-demand. One disadvantage of this model is that it can discourage people from buying since it is unclear, up front, how much the user will need to pay.

Charging your end user isn't the only way of pricing software. You can choose to give it away for free and then make money by, for example, charging for consulting,

installation and training; or selling advertising. The latter, although a common model for web sites, is extremely hard to make work. CPM – the cost per thousand impressions – can be as low as a dollar. In other words, to generate one thousand dollars of revenue you might need to serve up as many as a million pages. To generate enough revenue to support a team of three or four people, that means having ten million page views per month. Most web applications simply aren't going to attract that sort of traffic.

Giving your customers a choice of licensing models can make sense. For example, if you're buying Microsoft's SQL Server 2008 then you can choose to license per processor, or buy a server license and then pay per client who connects. The first model will cost you $5,999 per processor. For the second option, you'll need to pay $885 to run it on a single server, and then $162 for each additional user to access the database.

Many businesses end up with a **mixed model**. For example, Red Gate combines a one-off fee with an annual 10% - 25% support and upgrades fee. That way, we get both up-front revenue and a recurring yearly income.

However, if you choose to do this, you need to be aware of the pitfalls. Support and upgrades fees aren't just a cheap way of generating cash, and they can pressure you into releasing software just for the sake of it, at times that are not right for you, your customers or your product. If you're going to charge your customers regularly, then you need to make sure they get – or perceive – value regularly.

Shortly after launching Windows XP in 2001, Microsoft introduced its 'Software Assurance' program. For an annual fee, enterprise customers could get guaranteed upgrades to next versions of the operating system. In theory, everybody would win: Microsoft would get a guaranteed revenue stream to fund future development and customers could spread costs and would get a cheaper upgrade to Microsoft's new operating system, code name Longhorn, when it shipped in 2003. But Longhorn didn't ship in 2003. Or 2004. Or 2005. It didn't reach the market until the end of the 2006, largely neutered, as Windows Vista. And even then most enterprises refused to upgrade.

Choosing the right model

When choosing your pricing model, here are two recommendations. Firstly, **be boring**. Secondly, license your software as your customers expect it be licensed – fit in with their business model.

Red Gate's first product was Aardvark, an online bug tracking system. When we launched this in early 2000, we decided to follow a usage model. We charged per bug raised. This made sense from our perspective since the cost of providing the service was linked to how much our customers used it, but it didn't fit in with the way our customers worked or expected to be charged. That was our first mistake. Our second mistake was to forget to be boring, and to call the usage units 'cans of worms'. We thought it was pretty cool. Our customers had a different opinion, and we quickly moved to per-user pricing.

There are even worse ways of getting price models wrong. In the late 1990s, The Dialog Corporation was formed through the merger of Knight-Ridder and MAID plc. It was in the business of selling data to corporations and government bodies. Users logged on and searched for information in the six billion pages of information that Dialog stored.

Dialog decided to implement a per-usage model. Subscribers bought 'DialUnits', and different actions cost different amounts of DialUnits, depending on how much resource the action took and the value of the data being accessed. Want to sort your results? That would cost more than saving them. How much more? It would depend on the type of database you were searching, and the intensity of your search. Ranking, or removing duplicate results, was especially resource intensive so cost more DialUnits. Some actions were free. It took four pages of instructions to explain the pricing model to customers, and that was after a round of simplification.

In 2001, Dialog then introduced multiple pricing plans and expected users to choose whether it would be cheaper to use pricing based on usage, or on time. Then there were different platforms - Dialog Transact, Dialog Advantage and Dialog Enterprise. Throw in discounts, multiyear options and differing interfaces such as Dialog Classic, DialogWeb and DialogClassic Web and, as one user put it, thinking of a number, doubling it and adding your mother's age would have been a clear, better pricing strategy.

Chapter Five

Pricing Perception

Prices are never neutral. They send signals. For example, a high price can signal that you have a quality product. Consumers assume that expensive perfumes and wine are better than cheap ones, even in the absence of much evidence.

A low price can tell customers that you're value for money, or that you're special. If your competitors are selling software at $10,000 a seat, and you're selling yours at $100, then that says something about you. Of course, you might be saying 'game changing', but your customers might be hearing 'toy'.

Copy your competitors and you could be indicating that you're just a 'me too' product. If you're a me-too product, with me-too features and a me-too price, why would people buy from you, especially if there's already a strong, dominant product in your market?

Whatever price you choose, the signals it sends need to fit in with your brand, and your brand needs to fit in with your reality. There's no point using a high price to signal that you have a quality product if you're not willing to spend marketing dollars sustaining that brand, development dollars making that quality a reality and customer service dollars providing the level of service people expect from a quality brand.

In 1996, McDonalds launched the Arch Deluxe in an attempt to create a burger for a more sophisticated, adult consumer. To recoup the extra cost of the higher quality ingredients and the $200 million dollar marketing campaign, McDonalds priced the new sandwich 32 cents higher than a Big Mac. But the product they tried to create (high quality, premium) conflicted with the McDonalds brand (cheap and convenient) and the Arch Deluxe flopped. One argument could be that they priced the burger too low, and that a 32 cent premium did not send enough of a quality signal.

Your business model and your strategy have to support your pricing model. If you have expensive sales people driving expensive cars, taking your customers' CEOs out to golf, and end users who expect plenty of hand-holding and customization of the software you sell them, then you can't sustain a low price point. Similarly, if you're selling shrink-wrapped, mass-market software over the web then a high price point will be counter-productive.

When Red Gate tried to get into the automated web load testing market one of the reasons we failed (there were plenty of others, including a product that wasn't up to scratch) was that we attempted a low-price, high-volume approach in a market

dominated by high-price solutions. We figured that consumers would love a product that they could just download, try and then buy, but it turned out that our customers wanted much more handholding than we were able to provide. For the most part, they didn't want a product, they wanted a people-intensive service and the reassurance that a big-name, expensive vendor could provide. Our load testing tool was moderately successful, but it achieved nothing like the success we had dreamed of.

The dotcom boom and bust contains plenty of illustrations of companies who failed to align their pricing with their business model. For example, Kozmo.com's business was built around delivering snacks, DVD rentals and Starbucks coffee, within an hour, to city dwellers. Unfortunately, the low price, high volume business model it chose clashed with the reality of the expense of delivering small items by bike courier. This could have worked as a high price, low volume business model, as the butlers of English aristocrats will testify.

Switching strategies can be hard. For example, when Intel introduced the 8080 processor, it priced it at $340. Ultimately, it was selling for $2 a unit, but Intel found it very hard to shake the initial imprinting of the high cost in people's minds.

Practice Trumps Theory

You've read a lot of theory here. Wherever I can, I've based it on my experience and sound research (you can find some of my sources in the bibliography later on). But your own circumstances are different to any of those described here, so never forget that **practice trumps theory**.

Product pricing is as much art and craft as it is science. Sure, it helps to understand the economics and psychology of pricing, but theory can only tell you so much. At some point, you need to make a decision and do it. Use the information in this handbook to make an informed stab at what a good price would look like, and how your customers will react, and try it out. The exact price almost doesn't matter – get it broadly right, don't screw up totally – and you can tweak it later.

You're never going to know if you've chosen the exact right price or not, but you should experiment once you've set your initial price. Not experiment in the scientific sense of forming a hypothesis, changing a single variable, and accepting or rejecting the hypothesis, but in the sense of changing something and seeing what happens.

Scientific experiments are simply too hard to do, and the results too ambiguous, to be much use in pricing. Too many variables change. When you change your prices, you'll probably do it when you release a new version of your product, or it will coincide with a big marketing push, or some other variable you cannot control, such as the state of the economy or the reaction of a competitor, will interfere.

Although scientifically purer, it often doesn't make sense to change a single variable at a time. Theoretically, you shouldn't change the price of your product, your discounting strategy and the types of bundle that you sell, all at the same time. But practically, it can be the right thing to do. It's more useful to fix the problem than to understand why it's broken. When a scientist goes on a blind date that doesn't work out then, in theory, he should fix one variable at a time, and re-run the date. First, he should change the partner but go to the same film and buy the same flowers. Next, he should keep the partner the same, vary the film and keep the flowers the same, and so forth. But the pragmatist in him will, or should, change the girl, the film, the flowers, and buy some new clothes and shave too. If it works, he might not understand why, but at least he'll have a girlfriend.

In the old days, experiments were easy to run. You'd A/B test, splitting your customers into random groups, post each group a different leaflet with a different price and measure the outcome. Nowadays, this is risky. The Internet makes it easy for people to figure out what other people are paying.

You might be tempted to first run a **survey**, testing how customers might react to a proposed new pricing model, or change to an existing one. However, surveys rarely work. There is always a disconnect between customers' words and their actions. When McDonalds launched its Arch Deluxe burger (see above), consumers in focus groups loved it, and 80% said they'd buy it. Few of them did.

How To Change Your Pricing

You might be worried about how your customers will react when you change your prices. Don't be. For most of us, our customers have better things to worry about. If we shift our prices from $100 to $150 then most people won't notice, and of those who do notice very few will care. If you bought a copy of SQL Compare from the Red Gate web site in 2000 it would have cost you $50. Do the same thing now, and you'll find the price is $395. Buy the full suite of tools and expect to pay $1,595.

Of course, at Red Gate we've reached that price over the course of almost a decade. We've spent millions of dollars developing the software, and tens of man years. The increase in value that our customers get from our software vastly outweighs the increase in its cost. But, of the hundreds of thousands of customers we have, only a handful have ever commented when the price went up.

It's not what your customers say that's important, it's how they behave. Whenever you make a price change, pay close attention to what your customers do. If they stop buying, rethink.

If, however, you are running a subscription model then you should consider what you want to do with your current customers. If they signed up at $10 a month, and you've decided to increase the price to $20 a month, think hard about how you want to treat them.

Chapter Six

Product Pricing Checklist

Wrapping up, here's a checklist to help you decide your pricing:

What's your strategy?

Are you going to price low and sell lots, or price high and sell a few? How does this fit into your brand, the product you have and the image you want to project?

What's your product?

Don't forget that it's not just the software that you're selling. It's the entire package around it.

How will your customers judge the fairness of your pricing?

What reference points will they use? How will they determine what seems right? Will they baulk at the price you choose, or will they accept it?

Who are your customers?

How does their business work, and how do they expect to be charged? How much money do they have? Do they prefer a one-off fee, or a monthly subscription? Get under their skin.

Who are your competitors?

How will they react to your pricing? How much more, or less, valuable is your product than theirs? What is their business model? What are their prices? If you undercut them, will you trigger a price war? If you do, are your pockets deep enough for you to win it? Do you want to co-exist with your competitors, or destroy them?

How are you going to sell your software?

Do you need to send out sales people to take customers golfing? Or are you planning low-touch sales over the internet? Will you require a telesales team? How much will

each sale cost you? Do you need to sell via a channel or reseller? What cut will they take?

Can you segment your customers, and create versions?

Is your software worth different amounts to different people, and can you create pricing that reflects that? Students and business people for example, or normal and power users, or maybe you can split by geography or taste.

How can you bundle your software?

Can you create a larger package that contains more than one software product?

Make an informed guess at your price

Despite all the psychology and economics, you ultimately just have to pick a price. Some price – any price – is better than no price.

Try it out

Practice trumps theory. Try out your pricing and see what happens. If you've got your pricing broadly right - and if you've got this far you should do - then you can tweak it later.

Summary & Bibliography

In this usefully short guide to software pricing, we've examined a bunch of useful techniques to better pricing your software. I hope you've got something out of the past 60 pages. If you have, then please forward the eBook onto your friends and colleagues, tweet about it and share it around your network.

If you spot any errors in this book, it would be amazing if you could submit it to the errata page at https://efendibooks.com/minibooks/dont-just-roll-the-dice/errata/. If you think I've left something out, or got something wrong, then drop me an e-mail at neil.davidson@red-gate.com, or get in touch with me via Twitter, @neildavidson.

Bibliography / Further Reading

Anderson, E . and Simester, D. (2003) 'Minding your pricing cues', Harvard Business Review, September 2003

Breckon, N. (2009) 'Valve: Left 4 dead half-price sale saw 3000% increase, beat launch numbers' http://www.shacknews.com/onearticle.x/57308

Chapman, R. (2006) 'In Search of Stupidity' ,2nd ed, Apress, Berkeley

Crampes, C . and Laffont, J-J. (2002) 'Copying and software pricing'

Cusumano, M. (2007) 'The changing labyrinth of software pricing', Communications of the ACM, Vol. 50, Issue 7, pp. 19-22

Davidow, W. (1986) 'Marketing high technology - an insider's view', The Free Press, New York

Distimo (2012) 'Distimo Report'

Gallaugher, J. and Wallace, E. (2002) 'Understanding network effects in software markets: evidence from web server pricing', MIS Quarterly, Vol. 26, No. 4, pp. 303-327

Gilbert, A. (2003) 'CRM software or CRM shelfware?' http://news.cnet.com/CRM-software-or-CRM-shelfware/2100-1012_3-990880.html

Gourville, J. and Soman, D. (2002) 'Pricing and the psychology of consumption', Harvard Business Review, September 2002

Gourville, J. (2006) 'Eager sellers and stony buyers', Harvard Business Review, June 2006

Gourville, J. and Soman, D. (2005) 'Overchoice and assortment type: when and why variety backfires', Marketing Science, Vol. 24, No. 3, pp. 382-395

Gourville, J. and Soman, D. (2001) 'Transaction decoupling: how price bundling affects the decision to consume', Journal of Marketing Research, Vol. 38, pp. 30-44

Gourville, J. and Soman, D. (2007) 'Extremeness seeking: when and why consumers prefer the extremes', Harvard Business Review

Harford, T. (2008) 'Business life: Fair trade or foul' http://timharford.com/2008/04/business-life-fair-trade-or-foul/

Knopf, J. (2000) 'The Origin of Shareware' - http://www.asp-shareware.org/users/history-of-shareware.asp

Levitt, T. (1980) 'Marketing success through differentiation - of anything', Harvard Business Review, January - February 1980

Macrovision (2007) 'Key trends in software pricing and licensing'

Mason, M. (2008) 'The pirate's dilemma', The Free Press, New York

Miller, P. (2006) 'Sony losing mad loot on each PS3' http://www.engadget.com/2006/11/16/sony-losing-mad-loot-on-each-ps3/

Murph, D. (2006) 'Wii Manufacturing Costs ring up to just $158?' http://www.engadget.com/2006/12/15/wii-manufacturing-costs-ring-up-to-just-158/

Packard, D. (1996) 'The HP Way', HarperCollins, New York

Quint, B. (2001) 'Dialog rolls out new connect-time pricing' http://www.allbusiness.com/sales/1012692-1.html

Sink, E. (2004) 'Product Pricing Primer', http://www.ericsink.com/bos/Product_Pricing.html

Spolsky, J. (2004) 'Camels and Rubber Duckies' http://joelonsoftware.com/articles/CamelsandRubberDuckies.html

Spolsky, J. (2006) 'Simplicity' - http://www.joelonsoftware.com/items/2006/12/09.html

Stiff, D. (2007) 'How DeWALT turned customers into influencers' http://credibilitybranding.typepad.com/blog/2007/03/how_dewalt_turn.html

Sutton, J. (2001) 'Technology and market structure', 2nd ed, The MIT Press

Varian, H. (2003) 'Intermediate Microeconomics', 6th ed, W.M. Norton, New York

Wendt O., von Westarp, F. And KÃ¶nig, W. (2000) 'Pricing in Network Effect Markets', ECIS Proceedings, Paper 82

Wayne, B. (2009) 'YouTube is doomed (GOOG)' http://www.businessinsider.com/is-youtube-doomed-2009-4

Wikipedia – '3DO Interactive Multiplayer' - http://en.wikipedia.org/wiki/3DO_Interactive_Multiplayer

efendi books

Smart, succinct books for web developers

https://efendibooks.com