

[Home](#) » [Sell Your Services](#) » [Work Smarter](#) » [Page 1](#)

Turn a Client Site into Saleable Software

By [David Cummings](#)

November 25th 2005

Reader Rating: 9

Imagine you're a Web developer (you may not have to think too hard!), and you've been hired to develop an ecommerce module for -- say -- a bed and breakfast.

You've spent months preparing a site that will let guests book rooms and submit their credit card deposits online, and allow the bed and breakfast staff to manage their own online content. The company that hired you is a small, typical, independently owned bed and breakfast, and it's certainly not the only one in the market for such a product. Sounds like an opportunity!

Opportunity Knocks

A marketable product can offer your organisation better economies of scale than straight consulting work provides. Software costs lie almost completely in the product's development, and increase little whether you sell that product to six people or to sixty. A successful product gives you a better return on your investment than can standard consulting, and frees up your time to focus on larger issues.

If the potential product is a module you've developed for a client, you've done a lot of the work already. In the case of the bed and breakfast ecommerce module, you've already had a company finance its development. If you've done a good job, you've already got a happy client and a glowing reference. From there you can easily develop a case study to support your product. You already have a familiarity with the bed and breakfast industry, so you can relate to your target clients.

You already have a lot of elements working for you, and you haven't even started!

Before You Commit

How do you know if it's worth quitting your day job to sell the product you've developed? Here are a few key questions that you should ask yourself before you take the plunge:

Who owns the intellectual property?

Did you secure the rights to the code you developed? Was this clearly spelled out, one way or the other, in your legal contract?

How specific is the niche?

Is the product you've created general enough to be attractive to many companies? You've created a product that's useful to bed and breakfast accommodation owners; could it also be useful to boutique or small chain hotels? Or is what you've created too specific for use



David Cummings

David is the founder of [Hannon Hill Content Management](#)



[Software](#), a growing software company that develops award-winning content management software for enterprises of all sizes.

David Cummings has written **4** articles for SitePoint with an average reader rating of **6.7**.

[View all articles by David Cummings...](#)

[The Web Design Business Kit](#)

by anyone other than people who run bed and breakfast accommodation?

Is the market big enough?

Is the potential market for your product large enough to keep you in business? Obviously the more generic your product can be made, the better -- but if there really are enough bed and breakfast owners clamoring for your product, then go for it.

Is the market accessible [1]?

Many factors can affect the accessibility of a given market, so make sure you do your research first. There may be thousands of people wanting cheaper diamonds, but when [De Beers](#) [2] controls nearly all the known diamond mines in the world, you may not have much luck meeting that demand.

How saturated is the market already?

There may be a need within the market, but someone else may already be filling it. Do a [Google](#) [3] search for the type of product you want to offer. Will you have five or five hundred competitors? If there are enough potential buyers to go around, you may be fine. But make sure you have a plan that allows you to stand out from the herd.

What's special about your offering?

Whether you're attempting to fill a currently unmet need, or you're trying to break into an existing market, ask yourself, "what do I offer that's unique? Why should buyers pick this product as opposed to a tried-and-true brand name alternative?" A new product from an unknown company needs to have an edge on the competition in order to succeed.

Where is the market going?

Sure, there may be a market for your product now, but is it just a fad? Look at how much the cell phone industry has evolved in recent years. [Samsung](#) [4] has recently released a [phone that incorporates a seven-megapixel camera](#) [5]. Imagine what would happen to a company that still focused on offering superior battery life! Obviously it's still an important feature, but it's not driving the market anymore. To stay competitive, you've got to keep changing with the market, or, better yet, lead the change. Again, do your research. To be successful, your product must meet not only the market demands of today, but market demands in the years to come.

What resources are available to you?

Be honest with yourself. Do you have money to invest in marketing a new product? Do you have investors willing to back your product until it takes off? How much time can you devote to this new venture? 100%? 50%? Do you have marketing experience -- or someone in mind who could give you sound advice? And is this really the best time for you to launch a new product? You may not have all the strategic answers just yet, but being realistic about your resources will prevent some costly false starts should you decide to take the plunge.

If you've thought through all these questions and still feel confident in your product, the only thing left to do is to get out there and do it! Once you've decided to launch your product, it's time to start thinking about how you'll do so. And that's where the fun begins.

Choosing a Business Model



Everything you need to start a Web Design Business or Grow your Existing One:

- Strategies for leveraging yourself to **increase income**
- How to use **surveys** to develop your marketing plans
- Ways to **manage 10+ projects** at once
- **How to open offices, hire and manage employees**



**Download 4
Sample
Chapters FREE**

So you've got your product and you've decided to put it on the market. Great! Consider, though, that software is a complicated commodity. To market your product successfully, you need to select the right business model.

There are nearly as many business models out there as there are companies that use them. The trick is to find a basic model that fits your needs and highlights your strengths, then customize it to suit your product. Let's take a closer look at the major types of business models.

1. The Professional Open Source Model

In an open source model, you basically give away your software. Anyone who wants to use it can download the product for free, and because the source code is open and visible, users may be able to edit your software -- though this depends on the licensing scheme you've chosen. Among the many open source license schemes available are the [GPL](#) [6], [LGPL](#) [7], [BSD](#) [8], and many more. The issues of open source licensing are intricate and specific, and as such, these legalities won't be explored here.

Now, you might think that open source licensing doesn't sound very profitable, but it can be. You're still the creator of the product, so you're the one users will turn to for training and support. And because you're giving away the initial product, you may not need a lot of up-front capital to invest in marketing and sales. [JBoss](#) [9], a highly successful international open source software company, has been profitable from day one because [founder Marc Fleury created a business model](#) [10] that met the need for an alternative to the high-priced application server software that was on the market at the time.

Giving away quality software is a great way to develop a large user base because it involves less financial risk for the consumer. Many consumers are cautious about buying from a small company for fear that it will go out of business and leave them stranded with a buggy product and no support. Giving away the source code can help to negate those concerns.

Offering an open source product often also entails increased user investment, which can reduce the development costs for you: because users can tweak the source code themselves, you have the opportunity to create a strong development community around your product. With more people working out the bugs, writing plug-ins, and adding features, the costs of continued development and programming efforts are much more manageable.

Of course, by providing users access to the product's source code, you make public any special secrets you may have used in writing the software. Depending on how you choose to license the software, it's possible that users may be able to modify your software to create a new, competing product. Because they can see the source code, malicious users may more easily be able to find the holes and hack your product (though this doesn't discourage hackers attacking the proprietary products that are frequently their targets). And if your product is exceptionally good and easy to use, you may find that you have many users, but no one ever comes to you for training or support.

2. The Traditional Model

This model is called traditional for a reason: most software companies use some form of the traditional model to launch their products. With this model your source code is compiled into binary format, to keep it hidden. The resulting proprietary software is then sold as-is, and the original creators maintain control over -- and responsibility for -- any modifications or patches that are required and/or developed.

It can take a lot more time and resources to get started using the traditional model: developers have to aggressively market their products to make sales, particularly in the case of brand new products from unknown companies. But the benefit is that you do have the advantage of making money directly from those sales, and you may have better control over the ways in which customers use your product.

In some cases, keeping your source code hidden can be very important. JBoss didn't worry, because they weren't doing anything revolutionary with their software; their success was due to the fact that they did it well, and were the only ones distributing it for free. But if your product is really unique, you may decide that it's best to protect the source so that other companies can't see how you created the software.

Hiding your source code can, in some cases, more effectively protect the product against the efforts of malicious users. That said, an approach that promotes security through obscurity should not be relied upon -- the dedicated hacker will often find a way into well-protected products. Hiding your source can also make your job harder: it can be more difficult to find weaknesses in the build if the code is inaccessible.

Under the traditional model, the product's owner is the only source of the product, which can make that company more attractive to large buyers than a professional open source company might be. However, because you control exclusive rights to the source code, it can be an expensive and daunting task to keep the product current. Since you don't have independent users submitting patches or add-ons, you must employ a team of developers to fix the bugs and improve functionality.

You'll also have your work cut out for you convincing skittish consumers that your company will be around to provide support and upgrades in the years to come. If they're going to invest money in your software, customers need to know you're not about to go out of business and leave them stranded.

3. The Free Express Versioning Model

A variation of the traditional model occurs when a company creates two versions of its software: a full version for purchase, and an express version, with limited features, for free distribution. If your software lends itself to the easy development of an express version, this can be a useful marketing tool. As an example, think how many computers have the free [Adobe Reader](#) [11] installed, and how useful and powerful that has made Adobe's purchase product, [Adobe Acrobat](#) [12].

The potential downside, of course, is that many users will simply stop with the express version and never bother to purchase the full product.

4. The Hybrid Model

Not all successful software companies have such clearly defined business models. Depending on the nature of your software, it may be best to use some form of hybrid model.

One such model involves selling the actual source code for your product. In this model, you still make money up front, and you'll still have the expense of marketing the software, but your consumers will rest a little easier knowing that, if all else fails, they've got the source code to your product. It also provides the opportunity for users to fix their own bugs and add to the growth of the product as a whole.

Using this approach, you still maintain control over the licensed version of the product, and although you run a greater risk of having someone rip off your source code than you might under a traditional model, you have some legal protection against unlicensed versions of the product.

Australia-based [Atlassian](#) [13] has created a highly attractive issue-tracking software package, and has had success selling the source code along with the product. Because the product is targeted at highly technical users, the ability for clients to customize their installations of the software via the source code is a strong selling feature. Also, since it's not a product that's useful to private individuals, Atlassian doesn't have many concerns about a deluge of unauthorized users taking illegal advantage of the open source code.

The opposite model to this? Give away your product, but protect the original source code. With this approach, you lose the benefits of user input that are inherent in open source models, but you maintain proprietary control over your product, which can be seen to increase the value of your company as a whole.

5. The Hosted Model

A final model that works well for certain types of software is the hosted model. When you host your own software, you are essentially leasing access rights to outside users. By acting as host, you retain full control over the source code and any updates

or improvements. If the product is intended only to run on your own servers, you need not worry about excessive compatibility issues to make your software marketable.

Charging a monthly leasing fee offers the financial advantage of a more consistent income than does the traditional lump-sum-at-point-of-sale model, and offers improved economies of scale: typically, you can host many clients at the same cost as hosting one.

However, hosting the product means that you have the added responsibility of ensuring that the host servers are up and running twenty-four hours a day. You'll also have to work a bit harder to convince consumers that your business is stable. Consumers who are cautious about purchasing a software package from a young company will be even more hesitant to place their trust in your continued ability to host their valuable data.

Which Model is Best?

Which of these options is the right one for you? Obviously, the answer to that question depends on a variety of factors. There are highly successful companies running on each of the business models we've discussed. They are successful, in large part, because they took the time to weigh their options and select the right models for their endeavors.

When evaluating your options, here are some questions to ask:

- **How will you fund your product?** Do you have capital already set aside, or investors lined up to support your product's launch? If not, the traditional or hosted models may not be the best options for you. If you want to sell, you have to spend, and if you cannot spend, it may be wisest to look at an open source or hybrid model in which you can build a user base quickly and cheaply, making money from support once you have a userbase that has invested in your product.
- **Who are your competitors?** The level of competition in the market makes a difference! If the software you've developed is, for example, a new operating system that will compete against Microsoft [14] and Apple [15], you'd probably be wise to take Linux [16]'s lead and go the open source route. If, however, your competition is largely fragmented and comprised of medium to small companies, a traditional model may well be the best way to go.
- **How unique is your code?** Have you hit on something truly unique? Is your software dependent on a "special sauce" you'd like to keep secret? If you're worried about losing control of your product, it might be best to use a traditional or hybrid model that would keep your software proprietary.
- **Does your product lend itself to hosting?** Hosting has its advantages, but it's not always appropriate. Adobe Acrobat, for example, would simply not work as a hosted module. But if you've written email-marketing software, or a content management system targeted at mom and pop stores, leasing the software and hosting it yourself may offer the best solution for your users.
- **Who will use your product?** Never forget to keep asking -- and reminding -- yourself who will use your product. This, perhaps more than anything else, should guide you in deciding how to present and distribute your software. Will you target technical users who would want to have access to the source code, or business users who'd rather just type in a password and start using the product? Is your software useful for individuals, small businesses, or large corporations? Will users need to access a hosted version of the product online, or is yours an application that will be best installed locally on each user's computer?

As you ask yourself these questions, it should become clearer which type of business model will work best for your product. Remember that these are not ironclad rules: the above is just a series of guidelines. Do your research on other companies in your field. What do they do right? What could they do better? Is there a need that's not being met by current business models? Many companies have had a rough start and turned out OK, but that doesn't mean you can't learn from their mistakes and give

yourself a winning edge!

Once you've chosen a business model, it's time to take the next step, and consider how you'll market your product.

Product v. Service Marketing

Let's assume that until now you've only been marketing your services as an independent contractor. No matter how successful you've been at this, you'll have to change some of your tactics in order to create a successful marketing campaign for your new product.

As a starting point, consider what you have probably already been doing to market your services: selling yourself.

When you're selling your own services, you're essentially selling yourself. You need to convince potential clients not only that you are capable of delivering the technical capabilities they need, but also that you'll be able to communicate and work well with the in-house team. You have to treat every consultation as a job interview.

As an independent contractor, you're selling experience. You typically convince potential clients that you can do the job based on your previous work record. You need references, a portfolio -- anything that can prove you've done this work before.

Your scope will most likely be local, so networking is important. You attend trade shows, and make contacts within your field as well as in your target market. You want to be considered as a leader: you write articles for trade magazines, release white papers, and speak at conferences. This improves your visibility and gives you the credentials to convince people you know what you're doing.

That's all well and good, but now that you have a product to sell, it's time to expand your scope. It's time to go national. With a quantifiable product, it's a lot easier for people to see what they're getting before they make a purchase, but you've got to find your potential buyers first, and be able to hold their attention.

Product Marketing Considerations

In marketing your new software product, you'll want to consider a number of potential tools and tactics. The below comprise the basis of most successful online product marketing strategies.

Search Engine Marketing

Here is where search engine marketing really becomes important. If you haven't optimized your Website and collected some valuable inbound links, now is the time to start. You've got to be visible when people across the country type in your keywords.

In addition to the natural search, pay per performance advertising is a good way to gain exposure for your site. You can bid on various key words to have your ads shown, and then pay for every click or visit that originates from that listing on the search engine.

There is a lot of debate over whether it's better to focus on the natural search or pay per performance search. In my experience, I've found that a combination of both yields strong results. The good thing about search engine marketing is that it doesn't require a lot of commitment. You can experiment for a while to [find the balance that works for your company](#) [17].

Know your Customers

In order to meet the needs of consumers, you first need to understand those needs. If you have a product that caters to a specific niche market, you need to become an expert in that field. Let's go back to our original example of the bed and breakfast ecommerce module. Since you've worked with one bed and breakfast to create the software, you already have some familiarity with the target audience. Now you need to build on that. Attend national trade shows and conventions for bed and breakfast owners -- soon, you'll be a speaker. Read trade magazines, and start submitting articles.

Every industry has its own specific vocabulary; you need to be able to meet the needs of your target market and to answer their questions in their own language. The success of [Tessitura's arts enterprise software](#) [18] has resulted largely because they had an in-depth understanding of their niche, and could communicate using a familiar language.

Document It

When you're marketing your services, you rely on your own credentials, a portfolio of past projects, and your references. But when you have a product, you can -- and should -- offer even more.

In addition to the marketing blurbs, advertising copy, and white papers, make sure you have product fact sheets and a functionality checklist. Differentiate between features and benefits. Explain what your product does, how it works, and why it's important. Make it relevant to your target market. Much of the success of [Amazon.com](#) [19] can be attributed to its conscious decision to offer extensive product information. The more information and documentation you can provide for potential clients, the better your chances of securing their trust.

Find the Hook

Have you noticed the way every car commercial on television says that particular car is number one? Obviously not every car can be number one at everything, so how do they do it? Auto manufacturers spend big bucks to figure out what key feature they can use to sell their cars: number one towing capacity, number one in customer satisfaction, best value in a four door sedan... you get the idea.

What does your software do that no other software does? Look closely at what you offer, and come up with your own unique selling point [20]. Remember: if it's something your competitors can copy, they will. The key is to keep innovating.

Put a Name to It

You probably never had to worry about it as a consultant, but now that you've got a product, you need a name. Branding your software helps potential customers understand, describe, and remember what it is they want. This is why we hear people say, "I'd like a Coke" much more often than you hear someone say, "I'd like a cold, carbonated, sugary, cola-type drink."

As you come up with the name, remember who your audience will be. Technical people may respond well to a product that sounds powerful and state-of-the-art; non-technical consumers might be more attracted to a product that sounds friendly and accessible. When Steve Jobs named his new company Apple, he immediately set his computers apart from the slew of technical-sounding brand names like IBM. Having such a simple and friendly name helped to change how people thought about computers, and encouraged many non-technical people to bring them into their homes.

However, it is also important to remember that, for the unknown company, it's generally better to go with a more descriptive name. Adobe Photoshop has become such a popular and descriptive name that Adobe has had to release a style guide regulating the product name's use as a verb [21]!

Once you've narrowed down the options, and tentatively decided on a name, test it out. Ask colleagues, gauge the response at naming discussion groups [22], and ask people on the street what comes to mind when they hear the name. Don't make the mistake of sportswear manufacturer Umbro [23] which, in 2002, unwittingly gave a new athletic shoe the same name as a gas used in Nazi concentration camps. However you choose to name your product, the keys are to make it relevant, and make it memorable -- for the right reasons.

Hire It Done

As your new company grows, you'll more than likely need sales people. It can be a strange feeling to pass off the sales responsibility to someone else -- after all, you made the software, it's your baby! But it's all a part of growing up as a company. As an independent contractor, you could hardly hire someone to help you sell yourself. If you had someone whose full time job was selling your time, you'd never have enough hours in the day to handle the business being generated.

Not so with software. One of the reasons you're doing this is because of the economies of scale available through product sales. No matter how many people you have selling your product, or how many licenses are distributed, you're not going to run out.

When it comes time to expand, give your baby to the professionals and watch your margins increase.

Take it to the World!

Going national with a new software product is a pretty natural expansion. Marketing your software internationally, however, can turn out to be a giant leap.

Before you decide to [open your doors to international sales](#) [24], have a think. Does your software currently lend itself to international use? If the ecommerce site you built for the bed and breakfast business supports only your country's currency and tax system, you may need to do some major revisions before you sell the product overseas. Make sure also that you -- and your product -- are prepared for language barriers. Does the software support international languages and special characters? Does the user interface need to have multiple language versions? How will your office handle support calls from drastically different time zones?

None of these obstacles is insurmountable, and chances are that establishing an international market will be worth the trouble. However, when you're just starting out and testing the waters with your product, it's good to have a clearly defined and realistic scope for your marketing plan.

Making the leap from marketing your services to marketing your product need not be scary: you've already got a lot of the skills you'll need. The trick is just to build on that and expand your reach to take advantage of all the opportunities having a great product provides.

Releasing Your Product

Once you've developed your business model and created a marketing strategy, it's time to [get your software ready for release](#) [25]. A lot more goes into an official product release than the work that needs to be done when you've developed a single module for a specific client.

Plan for the Future

As a contractor, you're generally focused on short-term deliverables. Ideally, however, this is a product that will be around for a long time. Now is the time to start planning its future. Building a roadmap for your software is a great way to establish where you would like to see the product go. Although a roadmap cannot predict unforeseen demands or technological advances, it is important to have some vision of what you would like to see happen.

Plan for Your Product

Think about the features you would like to add to the product in the next four or five years. The great thing about software is that you can continue to perfect it as time goes on. A typical plan would outline additions and features to be added for each quarterly release for the next four or five years.

Having a plan for long term development will not only help you stay on track, it will influence how you code the software now. If you have an idea in advance of where the software needs to be, you can facilitate now the capacity for added functions later.

Plan for Your Company

Once you've established a road map for the software, it's time to do the same for your business as a whole. Where would you like to see your company in five years? What steps need to be made to get there? It doesn't need to be ironclad, just a list of goals and the steps you plan to take to accomplish them. This plan will keep you from making too many detours in the development of your business.

Automated Testing

When you're developing a module on a contract basis, chances are you're doing the testing yourself. When you commercialize that product, and plan for quarterly updates for the next five years, it's time to think about [automated testing](#) [26].

Randall W. Rice of [Rice Consulting Services](#) [27], co-author of [Surviving the Top Ten Challenges of Software Testing: A People Oriented Approach](#) [28], emphasizes the need not only for product developers to have the right testing tools, but to understand how they work, and be able to dedicate the time to ensure they accurately check every aspect of the product.

Automated, or regression testing, does take time to set up, he warns, but pays off in the long run and ensures a higher quality of product. There are two distinct types of testing:

1. *Emulated User Testing*
[Emulated user testing](#) [29] is exactly what it sounds like: a computer is programmed to mimic a human user and automatically performs a hundred or so prescribed common functions to ensure that any new coding has not negatively affected the overall functionality of the product.
2. *Unit Testing*
[Unit testing](#) [30] checks code behind the scenes on a micro level to ensure that all the product's moving parts are moving as they should.

Both types of testing are invaluable tools for ensuring all software components are operating properly prior to the release.

Nightly v. Continuous Testing

[Automated testing can be scheduled to run on a nightly basis](#) [31] so that anything you do during the day has a chance to be tested before you start work on it again the next day. Automated testing can also be implemented as [continuous integration unit testing](#) [32]. This, of course, requires more resources, but having newly written code tested every half hour means that you and your developers can fix problems sooner, while they're fresh in the mind.

Automated testing obviously saves developers a lot of time, but it also translates to a lighter product and customer support load further down the track. Automated testing quickly and efficiently finds problems that may easily be overlooked by human testing. In my business, we attribute our low instance of support requests to the fact that we run both continuous and nightly test builds automatically, to ensure that bugs are found before the software reaches our clients.

Managing the Product Release Cycle

The bad news is that, even with all this testing, there are bound to be some elements that don't work quite right, or things that you find you could have done better. The good news is that you don't have to release your product all at once: it's perfectly standard to go through a few versions of a product to get to the one you want to [release to your general user base](#) [33].

Alpha Release

On a quarterly release cycle, you can expect to spend the first two months strictly in development. After two months have gone by, you can issue an [Alpha release](#) [34] either internally, or to a select few clients who are willing to try out the product and offer feedback. During this time, you can continue to add features, fix bugs, and implement suggestions that are offered.

Beta Release

After you've spent about two weeks improving on the Alpha version, it's time to release a [Beta release](#) [35]. If you've kept the Alfa version internal, you might consider releasing the Beta to a few trusted customers.

This is the final step before releasing an official version, so at this time it's advantageous to implement a feature freeze in which no new coding is done. This will give your developers a couple of weeks to fix any remaining bugs and put the product through all the testing needed to ensure its quality. If you're not developing to the quarterly system we're discussing here, a good rule of

thumb is to implement a half-day freeze for each week spent in development. So, if you've been developing your software for ten weeks, for example, implement the freeze five days prior to release.

Gold Release

Now you're ready to release the official Gold version your software! Whether you're releasing your product for the first time, or [issuing an update to existing clients](#) [36], most companies today don't even bother with distribution of physical disks -- it's generally considered an unnecessary extra cost. Once you've made a sale, (or notified your clients of a new release, as the case may be), users can follow a link to the download site and get started using the Gold version of your software.

Planning years' worth of updates and new releases can seem a very daunting task when you're just starting out. It certainly qualifies as uncharted territory for most developers who are used to contracting out their services. But it needn't be: the purpose to all the planning is to break down your goals into little, easily managed pieces. By taking the development of your software step by step, you'll achieve strong results and maintain happy customers throughout the life of your company.

Product Sales and Distribution

Once you've got your product ready and your business plan mapped out, you can focus some time on generating actual sales. It is, after all, the only way your company will grow.

Approaches to Sales

Essentially, sales and distribution can be boiled down into two categories: direct and indirect.

The Direct Approach

The concept of direct sales is pretty self explanatory: in this model, your company assumes responsibility for the sale of your software. There are few large companies today that maintain a strict direct sales policy, but certain companies, such as [Apple](#) [37] and [Dell](#) [38], have made names for themselves as being major supporters of the direct sales model. The idea, basically, is that if someone wants to buy your product, they have to come to you, rather than a retailer.

This puts you completely in charge of the customer relationship, so you can more easily gather valuable feedback and foster better interaction between your company and your users. Because you are the 'manufacturer' as well as distributor, you keep 100% of sales revenue. However, the direct approach can prove a costly venture, and sales may be difficult at first for a new company with an unproven product. You must decide whether it will be cost effective for your company to maintain a customer sales and service department as it grows.

Channel Sales

The concept of indirect, or channel sales starts to get a little more involved. This is where you enlist the help of other companies to sell your product. Essentially it's like marketing to retail stores, but with software it's a little different.

Since there's no packaged product that can be put on a shelf, distributors don't need to purchase stock to keep on hand. Instead, they sell your software licenses for you, and you split the profits with them. A good example of this is [Microsoft](#) [39]. Have you ever purchased a product directly from Microsoft? Have you ever contacted their customer support department? Microsoft sells their products wholly through third party distributors, often as integrated components with other products. When was the last time you saw a new PC in a store that didn't have Windows preloaded?

There are distinct advantages to this type of practice. For one thing, less upfront capital is needed when someone else is doing the legwork for your sales. As you're just starting out, you will no doubt find many distributors that have a much broader reach, and can connect your product with buyers much more easily than you can yourself. Of course it's always beneficial to have a strong, established company backing you, and if your product is one that would lend itself to face-to-face sales consultations with potential buyers, it's useful to have sales partners outside your area to facilitate those onsite visits.

The inevitable drawbacks of this approach are that you lose control over customer relationships, and you don't get direct

feedback as easily as you can through the direct sales model. While you'll save some resources by outsourcing your sales tasks, you will still be responsible for training your new partners and making sure they're familiar enough with your software to adequately demonstrate it and answer questions with confidence. You have to split the earnings, obviously, and depending on the particulars of your sales agreements, it could be some time before your company recoups any of its investment.

A Hybrid Approach

As with a lot of things, most companies find that a mixture of both methods works best [40]. Particularly when you're just starting out, it's important not to get locked in to one form of sales. Further, the balance that works for your young company now may need to be revised as the business grows [41]. My business started out trying to sell strictly through indirect channels until it became apparent that this approach just took too long to start up. Now that the company has matured, we still make the majority of our sales directly, but we also have a number of vendor partners who have greatly improved our reach throughout the nation.

Partnerships

Partnerships can be very beneficial if they are handled well. In order to strike a profitable agreement, you need to understand the typical types of partnerships. But remember: whatever type of partnership you pursue (and you can have more than one), it's important to establish a good rapport and choose your partners well.

Vendors/Resellers

A vendor simply takes your product, sells it, and shares the profits with you. Just think of going to the store and buying Adobe Photoshop [42] instead of downloading it from the Website. You're still getting Adobe's software, but you don't have any contact with the actual manufacturer.

Integrated Service Vendors (ISV)

An ISV partner is a little more subtle than a vendor. Rather than simply selling your software, ISVs offers your product to their client in conjunction with various goods or services. Certain types of software lend themselves more readily to this type of partnership. One example is software that has been preloaded onto computers, such as Microsoft Windows. It can also come as a software add-on: when you purchase a new computer online from Dell, for example, you are asked if you would like to add the Microsoft Office Suite as an additional upgrade.

More often, however, an ISV includes your product with a service they provide. My company has established standing partnerships with a number of Web development and design firms. When clients hire one of these firms to develop a Website, they are then offered our software product for their content management needs. A key advantage to this sort of arrangement is that the new partner does not need to do additional marketing: your software is an add-on, not a separate venture.

Original Equipment Manufacturer (OEM)

Perhaps the most secure form of partnership is the OEM partnership. One company (that's you!) makes a product, then another company sells it under their own brand name. It's a model that's common throughout every industry, from cars to clothes to computers. If you buy a Dell laptop, for example, it's generally accepted that a different company has actually made that PC -- Dell simply puts a logo on it. No matter how good the laptop is, it probably wouldn't sell as well without the Dell name.

In the same way, opening your company up to be an OEM may be a wise way to expand your market and increase your revenue. Although you lose control over customer relationships, you have the benefit of support from an established company, with access to better resources and an established customer base. And the partner, of course, benefits from earning revenue from a quality product they didn't have to develop. OEM relationships are complicated, and difficult to arrange equitably [43], but teaming up with the right company can be a great way to jumpstart a new business.

Striking the Balance

Although there are no hard and fast rules, and most companies must learn by trial and error which sales balance works best,

you may find that, as a young company, you achieve best results from an approach that predominantly focuses on direct sales. This way, you have more control over your product while it's in its infancy, and can receive more direct feedback from customers. As your software matures, it may become refined to a point that its development no longer fluctuates on the suggestions of consumers. As your company expands, you may find that (like Microsoft), it's simply not cost effective for you to meet the demand of customer services. The ratio of direct to channel sales is ultimately up to you: you must decide what will work best for your current state of development.

Managing the Customer Experience

In the first article of this series, we spoke about the process of identifying a market and meeting a need. Essentially, it's the customers that inspired your software, and it's the customers that will ensure its success -- if you treat them right.

High-Value Customers

It depends on the type of software you are selling, but for many business models, a sale occurs only once per customer. Adobe may sell new versions of Photoshop to the same people who bought previous versions, but in the bed and breakfast case study we discussed earlier, you'd probably expect that the ecommerce software you built would always be a once-per-customer sale. However, this doesn't mean that we can afford to make the sale and never speak to the customer again! Never underestimate the value of having a reference account.

Reference Accounts

A reference account is basically just a happy customer who doesn't mind taking calls from your potential clients. Knowing that even one customer has taken the plunge and is using your software will set potential customers' minds at ease. Having a neutral party that they can call to discuss their concerns is invaluable when you're trying to establish a solid reputation.

So how can you acquire a reference account? You need three things: really good software, really good customer support, and consequently, a really happy customer. It's not something you can always predict, but if you keep your eyes open, you'll soon identify who would be a good referee. The best thing you can do is to establish a rapport with your customers, make them your friends, and keep them happy at all costs. When you're Microsoft, you can afford to be a little distant. But when you're just starting out, you've got to go above and beyond to prove that you're worth the money they paid.

If you've identified some specific markets to target, it will be beneficial to get reference accounts in those fields. So, your bed and breakfast ecommerce software product will be much more attractive if you have a few successful bed and breakfast operations as references than if you had a glowing reference from a hospital.

Lighthouse Accounts

Lighthouse accounts know who they are and can pretty well demand what they like. These are the leaders that everyone in their field looks up to, and having them on your list of clients is worth a lot of effort. When [Tessitura](#) [44] was developed to meet the ticketing needs of [The Metropolitan Opera](#) [45], there was an almost immediate demand simply based on the caliber of its users. Just five years later, major performing arts organizations all over the world are counted as clients.

Of course, getting an industry leader to invest in untested software from your brand new company will prove a little tricky. You need to decide what it's worth. Even if you end up giving your product away, if you can write up a case study, keep the client happy, and maybe even maintain them as a referee, the long-term benefits may well be worth the expense.

Dissatisfied Customers

In a nutshell, don't have them! Particularly as a small company, you simply cannot afford to have unhappy customers. Regardless of the customer's value to your organisation, it's crucial to keep all of your customers satisfied with your product and your company -- especially in the early stages of your business's growth.

The common 80/20 rule can generally be applied to customer service: eighty percent of customers will generate twenty percent of the support load, while the other twenty percent of customers will generate eighty percent of the support load. When your company is small and young, you've simply got to factor in for the high maintenance users. There will come a time, however, when you have too many customers to give yourself completely to all of them. If a particular customer becomes a problem, and you're honestly convinced that your company is not the cause, it's time to make a value judgment. How much is this customer worth? If it's your lighthouse account and a possible reference, you may find it's worth it to do whatever is required to keep them happy.

Generating Feedback

Regardless of who provides it, customer feedback is important for your growth. Maintaining communication with customers not only helps you generate ideas for improving and expanding your software, it helps your customers feel important to and invested in your company. My business sends an individual email once a month to each of our clients asking for feedback, both good and bad. It's been a valuable tool in helping us determine which features of our offering are useful, and what needs improvement. When we determine a trend in requests for a certain feature, then we can be sure that it will be worth the cost of development. In this way, maintaining customer relationships are an important way to keep the technology moving forward, and to continue meeting the actual needs of consumers.

Establishing a Product Community

Communities most readily crop up around open source products. Think of the difference between [Firefox](#) [46] and [Internet Explorer](#) [47]. Internet Explorer may still control most of the market, but it doesn't boast the active and passionate user/developer community that Firefox has established.

Even in the realm of proprietary software, the need for and occurrence of product communities is widely varied. If you're thinking of establishing a community around your product, stop and think about whether it is really appropriate for the kind of product you sell. Setting up a forum or message board is easy enough, but is it really beneficial to your product? Just because you're able to do something, that doesn't necessarily mean you should. It all depends on the scale of your software and the number of customers you serve. If you have a very small customer base, it may not yet be time to set up an interactive community. An empty forum reflects poorly to potential customers, and if it doesn't seem to be very active, current users are less likely to make use of it themselves.

The usefulness of a community also depends on the scale of your software. If you're selling a fifty dollar packaged application, having a forum where users can communicate between themselves and help each other learn and solve problems will be much more expedient than trying to handle all support calls yourself. However, in the case of a \$20,000 business solution, you'll probably have fewer customers to support, and your customers will expect to go directly to you with all their questions.

Nevertheless, communities can often spring up with no prompting from the product's makers. If a need arises, and a community begins to form around your product, embrace it, contribute to its growth, and it will contribute to yours.

Conclusions and Further Reading

As you've no doubt figured out, there's no right or wrong way to commercialize your product.

I hope this article has made you more aware of the challenges ahead, and better equipped you to make strong decisions for your new company. In commercializing a piece of software, you'll need to consider the questions of choosing a business model, planning to market your product, releasing your product, sales and distribution, and customer management in some detail.

The good news is that the work you put into planning, research, and preparation will pay off in the long term, as you bring a product to market, then work to overcome the hurdles that all software producers face. If you've got a great product, and you plan well, you're well on your way to an early retirement!

To look further into software development, management, sales, and business, see the following resources:

- [The Art of The Start: The Time-Tested, Battle-Hardened Guide for Anyone Starting Anything \[48\]](#) by Guy Kawasaki
- [Make a List, Check it Twice \[49\]](#) by Guy Kawasaki
- [The Myth of Open Source: JBoss founder Marc Fleury explains how his hot startup makes profits from its free application-server software \[50\]](#) Edited by Ira Sager
- [Create a Software Release Checklist \[51\]](#) by Patrick Andrews

[Back to SitePoint.com](#)

- [1] [/glossary.php?q=A#term_61](#)
- [2] <http://www.debeers.com>
- [3] <http://www.google.com>
- [4] <http://www.samsung.com>
- [5] <http://asia.cnet.com/reviews/gadgetbuzz/0,39041749,39220933,00.htm>
- [6] <http://www.gnu.org/philosophy/license-list.html>
- [7] <http://www.gnu.org/copyleft/lesser.html>
- [8] <http://www.opensource.org/licenses/bsd-license.php>
- [9] <http://www.jboss.com/>
- [10] http://www.businessweek.com/technology/content/oct2004/tc20041019_2492_tc182.htm
- [11] <http://www.adobe.com/products/acrobat/readstep2.html>
- [12] <http://www.adobe.com/products/acrobat/main.html>
- [13] <http://www.atlassian.com/>
- [14] <http://www.microsoft.com>
- [15] <http://www.apple.com>
- [16] <http://www.linux.com>
- [17] <http://searchenginewatch.com/searchday/article.php/3095871>
- [18] <http://www.tessiturasoftware.com/index.asp>
- [19] <http://www.amazon.com>
- [20] <http://desktoppub.about.com/cs/freelance/a/usp.htm>
- [21] <http://www.adobe.com/misc/trade.html>
- [22] <http://www.talkingnames.com/wwwboard/>
- [23] <http://news.bbc.co.uk/1/hi/uk/2222783.stm>
- [24] <http://www.zdnetasia.com/smb/specialreports/0,39045280,39187340-1,00.htm>
- [25] http://www.infoanarchy.org/wiki/index.php/The_Software_Cycle
- [26] <http://www.stickyminds.com/sitewide.asp?ObjectId=2084&ObjectType=COL&Function=edetail>
- [27] <http://www.riceconsulting.com/>
- [28] <http://www.amazon.com/exec/obidos/tg/detail/-/0932633382/>
- [29] <http://jwebunit.sourceforge.net/quickstart.html>
- [30] <http://www.javaworld.com/javaworld/jw-12-2000/jw-1221-junit.html>
- [31] <http://www.devx.com/Java/Article/15625>
- [32] <http://www.martinfowler.com/articles/continuousIntegration.html>
- [33] <http://freshmeat.net/articles/view/392/>
- [34] http://en.wikipedia.org/wiki/Software_testing
- [35] <http://www.microsoftmonitor.com/archives/005998.html>
- [36] <http://www.faqs.org/docs/Linux-HOWTO/Software-Proj-Mgmt-HOWTO.html#CHOOSEVERSIONING>
- [37] <http://www.apple.com>
- [38] <http://www.dell.com>
- [39] <http://www.microsoft.com>
- [40] <http://www.varbusiness.com/sections/90franchise/ten/10hero.jhtml>
- [41] <http://www.businesstechnology.com/BT/Content/index.cfm/fuseaction/viewArticle/ContentID/71>

- [42] <http://www.adobe.com/products/photoshop/main.html>
- [43] <http://www.reed-electronics.com/electronicnews/article/CA284747.html>
- [44] <http://www.tessiturasoftware.com/>
- [45] <http://www.metoperafamily.org/metopera/home.aspx>
- [46] <http://www.mozilla.org/products/firefox/>
- [47] <http://www.microsoft.com/windows/ie/default.msp>
- [48] <http://www.amazon.com/exec/obidos/ASIN/1591840562/>
- [49] <http://www.entrepreneur.com/article/0,4621,319611-1,00.html>
- [50] http://www.businessweek.com/technology/content/jul2005/tc2005078_5465_tc121.htm
- [51] <http://www.builderau.com.au/manage/project/0,39024668,20267990,00.htm>

» **Page 1**