The No Bullshit Bible : Creating Web 2.0 Startups & Programming

Written by James Gillmore
Edited by Holly Welch

FaceySpacey
www.faceyspacey.com

Business

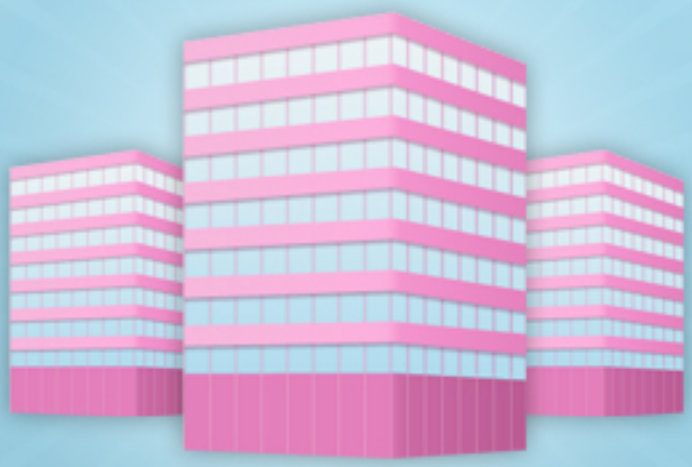


TABLE OF CONTENTS

Non-Technical

Chapter 10

BUSINESS 2

Hiring

1. Introduction to outsourcing 4
2. Finding the firm to hire 5
3. Managing your new outsourced team 9
4. Offshoring 13
5. Building a team 17

Funding & Legal

6. How to fund a startup 20
7. Startup contracts 25

Contracting

8. How to get in & out of contract development 27
9. Top 10 easy site creator tools 30
10. Must know SaaS tools 31

Conclusion & Further Reading 32

10-1 BUSINESS HIRING | INTRODUCTION TO OUTSOURCING

At FaceySpacey we obviously are “outsourced” to all the time by startups. We too do a lot of sub-contracting as well for areas that are not our specialty. So we know a thing or two about how to outsource a job properly. Before I became a coder, the skill of outsourcing was one of the 2 biggest skills I prided myself of (the other was speccing). Before I was a coder, I basically had to outsource everything, especially before we had some in-house developers. So I built a formula.

The formula has been two-fold:

1) Know what you want (i.e. spec/plan your product deeply)

2) Hire the best team you can for your money

In the Speccing tutorials we cover point one at great length. So we only need cover point number two here.

Before we cover techniques to finding the best team and making sure they succeed at building your product, let’s discuss the reasons you’re outsourcing to begin with.

In an ideal world you’d build your startup with your own in-house team. Period. If you think otherwise is the right idea, all these tutorials are too advanced for you and you shouldn’t be building a Web 2.0 startup. If you’re McDonald’s and are fine paying the fees of a top-notch firm to execute some side-project initiative, sure. If you’re a small company that needs to put a web “shingle” up, fine. However if you’re serious about making a successful startup and even getting it to launch, you need to understand that the ideal way to roll would be to have your own in-house team.

So that all said, the reason your outsourcing is because you can’t hire full time developers and guarantee them a professional monthly wage over who knows how many months. You need to be able to hire a firm for a fixed price where you know what

you'll get for your money. You can't afford to go through the growing pains of building a team, learning and growing your own process of working, etc.

So that said, you got basically one shot to find that perfect web development company to build your application. And once you lock in with that team, you need to make sure they succeed.

10-2 BUSINESS HIRING | FINDING THE FIRM TO HIRE

Finding the right firm for an outsourced project is different than how you hire developers when building an in-house team. In the latter, for example, you'll care less about the technologies they know, and more about their overall quality. For the latter, you'll follow the process taught by Joel Spolksy (creator of Stack Overflow and Fogbugz) in his book Smart & Gets Things Done: http://www.amazon.com/Smart-Gets-Things-Done-Technical/dp/1590598385, which we also cover in Hiring 5 - Building a Team. That book says you look for 2 things in a developer: A) that they're the smart, and B) that they have a get things done attitude. The programming languages and frameworks they are skilled in is secondary since they are sharp and get things done and will learn the tools you learn quickly, and they'll have time to do so since it's a long-term hire. For an outsourced project, they need to be proven masters of the technologies you need. If you're building an iPhone app, you're looking for developers that do primarily iPhone apps. If you're a coder and you use the Yii Framework, they need to work almost exclusively in Yii.

So if you know what technologies are involved, you know where to look. That's the point. Browse the Yii Framework forum for example. The more granular the skill you require is, the easier it will be to pinpoint developers with those skills. Searching for Javascript developers for example will be more difficult than searching for jQuery developers. Overall, the point is that developers congregate in certain places related to the technologies they're using. Also, the best developers won't be looking to be hired--they're busy! So you need to find where they talk about coding on the web, again such as the Yii forum, and peel them away from their current project or book them for after they're done with their current project. So I'm not saying search oDesk or Elance for people that use Yii. Go to the Yii Forum and find guys that make tons of posts and are clearly very passionate about Yii. Then have a compelling project and make an even more compelling offer to get them to come to you after they're done with what they're currently doing.

The next thing to note is you really want a firm that's already a firm, not just one guy that brings on another one or two guys. You want to hire a firm with a web presence and reputation to maintain. You need to be able to pinpoint where their reputation specifically is kept: i.e. on Stack Overflow, Elance, oDesk, and with past important

clients. You're a small new company with X amount of dollars under $150k. You need to know that the firm you're going to hire will take your project extremely seriously. They need to be looking forward to adding your completed project to their portfolio.

The next thing you do is you produce as big of a list as you can of all the prospective firms you would like to possibly hire. For one of our projects, ThirstyVIP, we found a list of thousands of iPhone developers on twitter, i.e. we found their twitter URLs. We visited them all, and compiled a list of all the URLs to their sites/blogs. And then continued to narrow the list down based on what we liked on their site. We also found lists on the web of a bunch of iPhone developers. Lists do exist for all sorts of programming specialities. Find them all, and filter them down to your own list. This process could end up taking a few weeks. Just do it. Don't even bother contacting them all until you have the list.

Once you have your filtered list, build a little micro-site to represent your RFP ("Request For Proposal"). If you don't know what an RFP is, learn now: http://en.wikipedia.org/wiki/Request_for_proposal . It's basically a document to potential developers describing what you want and describing your individual hiring process. For ThirstyVIP, we built an entire site with our entire project's spec on it. Yes, we publicly put it out there because we knew how important finding the right team was

and that if we didn't we would never get a project completed anyway--i.e. we weren't worried about the project concept being stolen because we realized there would be nothing to steal if we didn't get the right team to take it to completion. We added a form to the site through which firms could supply their bid, and information about themselves, and emailed out a link to this site to hundreds of potential firms. That's how serious you have to be. We even made the first page of the site have a video showing all the pages of the app (which we got designed first), with a voice-over explaining the functionality. And of course the product and tech specs were on point to the T. What this did was 2 things: A) help us get the lowest price, and B) insure the developers themselves don't put themselves in a bad situation because they underestimated the project and therefore can't complete it.

Basically, there are tons of firms out there. You need to put your mind to finding the best one. Take your time. Take 2 months if you have to. Take more. You only have $50-150k to get your one shot of making your dream come true. You're not a coder. You need to bank on the skills you do have, which is communicating with people. Talk to as many of these firms as you can, feel them all out. Don't settle. Don't go for the first one that works. Also since you have an RFP on a public site, they know you have a lot of options. As a result you'll get the best price and the firm most serious about taking on your project. There are a lot of great firms out there that are just not in a good place at the time to take on a project of your size. You need a firm that just completed a large project, and really needs a new one, for example. The timing must be correct. Bad timing with a great firm equals a bad firm for you. You need a great firm with great timing to take on your project. After talking to tons of firms, you'll feel out a lot of these nuances, and have a lot more criteria to fuel your judgement.

10-3 BUSINESS HIRING | MANAGING YOUR NEW OUTSOURCED TEAM

So you found the team, how do you not fuck it up along the way? Yes, I used an expletive in my tutorials, which I usually refrain from doing. The reason I'm using it now is because it's so easy to be the wrench in the machine if you're new to this whole software development thing.

The ways you may fuck it up are:

***1) Changing specs, i.e. changing what you want for your product* -- this is the biggest one**

2) Talking your developers' ears off so they have no time to work. This is a problem because you'll most likely undermine your confidence/trust in them, while making yourself seem less knowledgeable. If you're not technical, you're only going to show them all your flaws. Keep your communication concise.

3) Being funky with money, i.e. changing payment milestones and terms as you go. Just don't do this. If your team is having a hard time meeting a milestone, at the beginning be lenient with paying early if they really need it, and near the end be less lenient.

4) By not having enough milestones. Have as many as you can.

The biggest no-no is obviously changing your specs in the middle. This is obviously easier said than done, and it's the cause of 95% of all software contracting failures. It's so easy to happen. Save extra money in your budget for when it does happen because it doesn't, and of course spec like a motherfucker and read our Speccing Tutorials. If it does happen, be prepared to fairly and decisively work out a new contract for the additional work. Be the the first one to bring up new costs for it. Don't make yourself look like an ass by trying to get work for free, by framing every new feature as a bug or extension of a previous feature. When you do that, you'll just give your developers excuses to screw you in the future by not finishing your project. Be the blueprint for morality when it comes to pricing stuff out and negotiation. You'll get more results by

giving a little (or a lot) in this area. Save aggressive pricing for larger parts of your application, and for little things allow yourself to give your developers a great price.

In general, bad pricing for your developers will always come back to you. You really should be looking to pay them more than they're used to for the same amount of work, and just keep your product small. This way they're happy to build it. If it turns out to take longer than they thought, they'll have less excuses to make because they were once very happy with the pricing. So don't be afraid to take on more than their bid for the initial pricing of the complete project. That said, make sure to get a price for the complete project at the beginning, not just the first phase. Otherwise, the latter phases will inevitably be a whole lot more expensive. The reality of it is you're inventing something new--not just making a company site--and 9 times out of ten the project will be underestimated. You really should be building an in-house team, but you don't have the funds for that. So in this reality, it's just a must you lock yourself into a price for the whole thing from the beginning.

Another reality is take one eighth of your budget and build the initial launch with that. If the economics make sense, i.e. for $20k you can get to launch, do it. More often than not, since you're building something innovative of real value, it will take at least $100k to get your initial launch product done. But again, if you can get it done for a lot cheaper, do so, and save yourself a lot of heart-ache. Still apply these techniques, and just improve upon them from phase to phase. The goal is to always get what you wanted for the amount of money you first planned, and without exhausting your developer resources until they don't want to work with you anymore, which is a very common occurrence. So that said, don't plan half your product for $20k. Do it at $40k then, and save yourself a $110k buffer (i.e. if you're total budget is $150k). You need your firm working towards complete products. It's a lot easier for them to get off the hook and therefore build a lot less and fix a lot less bugs, if all that is expected is a partial product. When they're working for something launchable--and you need to first

make it very clear that they're working for this--they're going to go a lot farther for you. They'll seal up minor product misses on your part.

The point is they need to be on the hook to get a product done, not just your precise specs. What I'm saying is you gave your very thorough specs to help them, but your contract with them still needs to be "complete products." That means they're taking some of the product responsibility on their back. If there are disagreements of the intent of the product, you need to be fair, and able to lose at least half the battles over whether something was in the spec or not (i.e. in order to maintain their trust in you). What this will do is enforce how serious you are about holding them to completing a product, not just the great specs, which will inevitably miss a few things. You need to be disciplined in sticking to your word that you gave them to build your product, and not a different product through feature requests you added. In doing so, they'll honor their word to complete your product, and patch a bunch of product holes.

The last thing you must do is make the team you hire find do their own speccing of the product before you hire them to build it. You should pay them for this pre-phase. This phase includes them internally speccing your project more deeply at a technical level, finding product holes, and getting answers from you for what to do to solve these holes. It's your last time to make sure your plan for what you want is complete.

You can get a price for the overall project before or after this. If you get the price after, it may be bigger because they find that their is a bunch of stuff they didn't think of before. If you get the price before, it will inevitably be less for the inverse reason. So depending on how tight your budget is, you choose what you want to do. If they overprice, it will mean you have longer until they start giving you shit for the project taking forever. Personally, I'll take the quote ahead of time, and earn bonus points for giving out additional money and bonuses if things are taking longer. That will go a long way.

One of the things about developers you must know is that they're not greedy, but need things to be fair with pinpoint accuracy. They're programmers. They believe everything can be programmed, i.e. that there is a science to every equation. Between each other, developers have a very strict code of right and wrong because they can point out precise flaws in each other's code. I.e. if they have a debate about some technical theory, they can actually box it out in a code, while the rest of us are left making empty statements to each other that we don't have to prove. This culture leads to a built-in justice system. And believe me, it will carry over into how you deal with them, even if you're not a coder. That said, I have no problem with developers underestimating. It's their responsibility. I can't afford to be in an all too common bad situation where I have no extra money from a client but need to get a feature done for a client because he's expecting it and also doesn't have an extra money. In the layer cake of contracting that I'm in, I'm at the top and therefore deal with the least technical of clients. Therefore, I'm going to get more product blunders on the part of my client where they forgot to make it clear that they really needed a feature, and truly without it their product isn't launchable. However, developers I hire are lower down the layer cake. They're receiving detailed specs from me, and are on the hook for less such bonuses to complete a product. So therefore, because I do provide such detailed specs, I don't need baby developers I hire if I think they're underestimating themselves. I feel completely justified in that. However, I will add bonuses to ease the pain if things get difficult along the way. This is how I stay in equilibrium in this cut-throat justice system between developers. In short, my clients above me in the layer cake will expect more extras missed in the spec and pay more, but developers I hire will give less extras and I'll pay less. Of course there is a markup too, but I'm talking as if that's not part of the equation, i.e. like when you remove inflation from economics calculations.

All in all, money is a very important part in dealing with your developers. Be on point with it as you things you missed in the spec come up. Prepare for that as much as possible by speccing like no other. Save extra money in the bank if you can't. And build

tight relationships with your guys. Not being a doosh bag is currency!

10-4 BUSINESS HIRING | OFFSHORING

Many of you are probably wondering specifically about “offshoring.” The previous hiring tutorials specifically didn’t differentiate between “outsourcing” and the flavor of outsourcing known as “offshoring.” Now we’ll cover offshoring.

Let me say this: offshoring is absolutely perfect for coders that need small bits of help. Assign the offshore developer you’ve hired to a small task, and see if he executes to your liking, and gradually increase the size of the tasks you give him. You’re a coder, so you can monitor and understand every line of code he/she writes. Given the cost benefits and the sheer number of developers you have access to when you open yourself up to the world hiring market, it’s a no-brainer. You’ll find someone that suits your needs. In all honesty, there are tons of fantastic developers outside of the United States. And the economics do work. They can work very very well.

However, it’s no secret that developers not smack-dab in the middle of Silicon Valley don’t have the product common sense--if you will--that ones from the US are more likely to have. It’s very common for coders to not have the product common sense you do in general, especially when you’re product takes the insider knowledge that you

have from doing business in your niche. That's always going to be given. So couple that with the fact that a lot of these great developers come from countries where only a handful of their friends have iPhones, etc. In short, you're going to be in for a painful awakening when you hand a team of these developers your spec, agree to a large fixed price for your project to be executed over several months, and they end up misunderstanding many things along the way. This is a story that's been told too many times.

So we've already covered how important it is to spec deeply, how to find a great team, and generally how to manage them after you hire them. Let's now discuss some techniques specific to managing offshore teams if you're not a coder yourself.

The best way to do it is to make software development your business. I.e. you can be a product guy that project manages, and hires offshore teams. So if you can make software development your business and you have clients you do websites for, what you're looking to do is basically lock down your own exclusive team and evolve a way to work with them over time. Of course start with small projects--may do a company site for the store down the street.

The benefit of having your own exclusive team is that they cannot give up on your project in the middle. They are depending on you for the next project. So you basically eliminate the fear of having them leave you. You'll inevitably have to overlap projects when one project takes too long and ends up being worth less more money per all the time it's taking than originally planned. You can essentially pull somewhat of a Bernie Madoff style ponzi scheme. You're going to have to fix it at some point. I'm not saying to doing anything imorral at all. I'm simply dealing with reality here. You may have times where you have to feed your offshore team a different project while still working on a current project to help them make sure their financial needs are met while they go the mile on the current project. If you're smart, the additional project will be priced

better. If you're new to this business, you may use the currently half-built project to demonstrate to prospective clients what you're capable of. I always use my newest projects to showcase my work anyway, as my newer work is always better than my last.

That said, even if software development can't be your business, you should find other projects to give to them. You should start them on a small non-crucial project first. You should do whatever you can to show them that you're a repeat customer. Period. Build a serious rapport with them that they know will last a long time if they do a good job. This is the technique.

If you actually do have a full time web development business or actually see yourself doing this, here's how you lock down a team permanently:

1) find a budding camp of 2-3 developers (i.e. off odesk, or elance)

2) make sure this camp isn't far enough long in their careers that they have their own company site.

3) be the one that helps take them over the edge to being a professional development firm, or at least be one of their first clients, or at the very least their most serious client ever.

4) don't cramp their style--so let them have their own company they can call their own offshore

5) then make an agreement that you'll be their exclusive client and you'll provide all the jobs. All the problems will occur when they start working on other projects in place of yours.

6) You will need to take responsibility for their minimum financial monthly goals. Figure out what those are. And be prepared to have their back in times when your main projects are taking more time than they should (and therefore costing them more money).

7) Make sure to price everything out at a fixed price. They are offshore, not in your office. This is the only way they can will feel they have to take full responsibility for the work your sending

them. You're not Microsoft building whole offshore companies in Romania or Google doing the same in India, etc. Those companies can pay developers consistent salaries rather than do fixed price because they're not really offshoring--Microsoft and Google are just expanding their companies to those countries and some may call that "offshoring."

8) Have their back when things get messed up (i.e. in terms of money). Show them that you can be flexible too and you're not out to screw them. This is a must when they hit hard times on your all-important main project and it's harder than they expected. It's what will make them go the mile to complete it.

9) Find one main guy offshore that leads the team, takes on all the responsibility, etc.

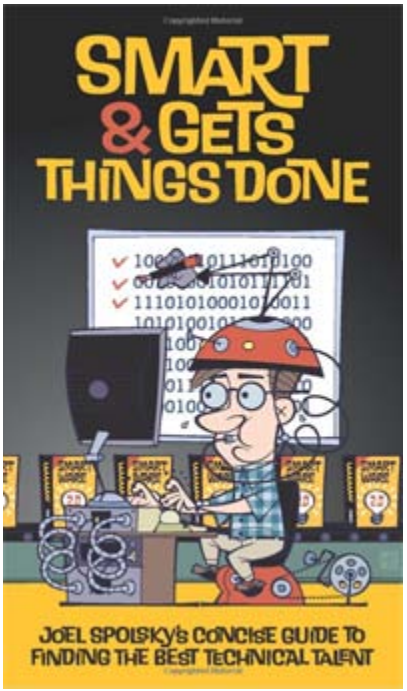
So that's basically it. Let me describe building that main relationship with your offshore team lead more. You need to build an organic and tight relationship with that one lead offshore guy. One guy over there must have his ass on the line. By helping him build a company, and do things like get office space (which should be cheap where you're offshoring too), and have status by being the boss of several guys over there, you're already making his dream come true. Usually in these countries, their dream is to be the owner of a contracting company, not to take all the risk in building a startup like your dream is. It's very true that people in the US are a lot more risk-averse while people in 3rd world countries have lots of doubts of what's possible. I've had these discussions point blank with many offshore developers I've worked with, and it's not just a myth. It's completely the truth and they know it and admire the US for this. You need to own this fact and use the economics it produces to your advantage.

There is very few examples in their homeland of successes like Facebook, Twitter, etc. They just can't afford to dream that high, and don't. Therefore, the bar for the dreams they must accomplish is a lot lower, and you can basically be the one to make it happen by helping them build their little software contracting company. And again, it's all about pinpointing one main guy that will hold down the fort. You won't be able to have that relationship with all the developers he finds because everyone doesn't have

a business owner mentality. So that's the point I'm trying to make--make sure the guy you find has a business owner mentality, will work weekends and do whatever it takes. You're going to need it. In short, you need someone else to share the burden of the responsibility for completing your project with.

10-5 BUSINESS HIRING | BUILDING A TEAM

This tutorial is based on the following book by Joel Spolsky: http://www.amazon.com/Smart-Gets-Things-Done-Technical/dp/1590598385 . Joel Spolsky is the creator of Fogbugz and Stack Overflow. He's prolific developer, who build his career working at Microsoft on the Excel project. His book, Smart & Gets Things Done, is a small book that details the must-knows about hiring developers. Here I'll highlight the main points.



His main point is do not settle on who you hire, and to do so you need to look through tons of developers, and send them through a rigorous interview process with technical tests and multiple interviewers. Only one interview is allowed to not like the developer.

Everyone else must basically love the developer, and if there is an inkling of a doubt, as an interview you must mark the developer as a no-go.

Another key point is do not hire based on what skills the developer already has, i.e. what languages they know. They'll be with you for a while, and will be able to learn new languages and tools quickly. Why? Because they're smart and get things done. These two criteria points basically mean the developer can't be someone who's really smart but one of those developers who gets caught up in talking too much theory that he's not proactive enough. On the other hand, he can't be the sort of guy that gets work done, but hacks stuff in other developers later have to fix up. Joel puts it a lot more eloquently.

Joel also describes where to find these great developers. In short, guys on the job market looking for you for the most part suck and are on the job market for a reason, and the great developers must be found in their native environment, i.e. finishing school and at other jobs. This book was written 5 years ago. Now, you basically want to find guys with great blogs that talk about the technologies their adept with. Period. Don't expect a great developer to land in your resume heap. Go grab them from where they are, and have something great to offer them. One great practical tool you can put into use is paid internships. Joel describes in depth their internship program at Fogcreek (the company that makes Fogbugz). The internship program's main purpose--i'd say--is really to make the student love working at Fogbugz, not to suck extra work out of a younger lower paid developer. It's an investment into the future hiring Fogbugz does. So the intern must love working at your company, which leads me to the next point.

His next main point is basically be an awesome job to have. When you're hiring guys, fly them into your city and pay for them to stay at fancy hotels. Give them private offices with that famous $750 spinny chair. etc. Great developers have options of

where to work at. Joel takes his developers seriously. He used an analogy of a story about Samurais used to protect a village. Developers are any startup's biggest asset. So pay them well, and treat them great.

He made one final point that I thought was awesome: paying a premium for good developers is not like paying a premium for hardware parts. The reason is because once software is coded, you can sell more and more copies of it for near to nothing. Your code is your biggest asset, but itself is extremely cheap to reproduce, i.e. resell. So don't skimp on the developers that make it. It will continue to pay for itself in ways that other harder expenditures cannot do. Basically, great developers leads to exponential gains, while the benefits provided by, say, enhanced hardware parts in a hardware startup, lead to more linear gains. A great developer can at times do the work of 10 other developers--which in sum will cost way more. Great developers also add finishing touches to the software that mediocre ones simply won't, and these touches are what make products magical. In the day and age of beautiful interfaces and user experience, such as those seen in Apple iPhones and iPads, it's all about those magic finishing touches that take your product over the hump to outperform your competitors.

My final piece of advice in this chapter is to keep an ongoing list of developer blogs. When you get that serious round of funding and can hire the best around, you'll already know where to look. Categorize that list by the type of developer each developer is. Keep track of where they currently work. Keep track of their career. Learn from their blogs yourself, and post comments on their blog. Get to know them years before you hire them if you can. Build a relationship with them on Twitter. Show them that you appreciate them and value them. Bring value to their blog through comments of your own, e.g. helping its readers get the most out of their tutorials. Joel's book was written in 2006, but if it was written now, I'm sure he'd be saying that. He'd also be saying to use Stack Overflow's recently released tools to hire and find

developers: http://careers.stackoverflow.com/employer/candidate-search . I haven't used it yet, but my guess is it will soon be phenomenal if not already. Inevitably with that tool you'll be able to get to know tons of developers by reading all their Stack Overflow questions and answers. You can do that already, but that tool will really help you drill down to them, rather than just find a guy here or there that's using a technology you're using.

10-6 BUSINESS FUNDING & LEGAL | HOW TO FUND A STARTUP

There are two ways to fund a startup, depending on if you're technical or not. If you're technical, build a prototype and get actual users on it. Period. Cry me a river if you can code, but can't design. Save up a few bucks and get it designed. If you can actually code things, you're not worth investing in if you can't get something to launch on your own. Just get something, anything, that's useable and demonstrates where you're going while providing at least a morcel of real value now.

If you can't code, you really shouldn't be doing software startups unless you have 10 years of experience in some obscure niche where the only one that could imagine a startup to create efficiency in that niche is an experienced professional/manager at a large company in that niche. That obviously wouldn't include coders, so therefore only your non-technical self is the only person able to imagine how to create efficiency in that industry. The good thing for you is that because of your senior expertise, you probably have connections to people that would easily invest in you and your idea, even though you have no coding skills. You're not who this article is written for because you'll have an easy time raising money. Who this article is written for is the dime a dozen twenty and thirty year-olds that can't code themselves who are trying to do startups in niches that others have way more experience in or in niches coders can easily compete in since not too much industry experience is required to make the startup.

So, poor soul you, you're most likely going to fail--statistically speaking at least. However, I'm not one to ever consider myself a number and let statistics hold me back. So this article is for you and the warriors out there that refuse to give up on their dream, even though they are not resolute enough to go learn how to code, which I've made very easy in many of my tutorials and have stated many times that you really need to do if you plan to have a successful startup.

What you do is this: get some money together. You need to at least be able to get 3 grand together. If you can't save that up over several months at your waiter job and your other part time job, I don't know what to tell you, but you're a loser and I'm not writing for you. Take that $3k and do two things:

1) build a simple company site (with a blog) on your own domain that pitches your product. It doesn't need to say you're looking to raise money, but it should hint that you're looking for investors. The purpose of the site is to raise your next round of money based on what you've already put together--though quite minimal.

2) graphically design and brand your product, or at least the core pages it has, and put this up on your site. Make a video presentation with you explaining it in a voice-over.

Now armed with these two things you are about 10-100 times more serious than you were with only an idea constantly foaming from your mouth. Just do it. Just get it done. You may have some cockamaney idea that you can get your application done for $15k, and therefore $3k is to precious to spend on this middle-step. But 99% of you guys will simply run out of money before your product is ever launched.

The next thing you do is put a team together--not immediately go looking for money. Tell the potential team members (who should mainly be coders) that you're very close to raising money, even though you're not. When you have you're roster setup, put them on an About page on the site, as if you have the team already built. Give them all descriptions explaining their technical skills and how great they are.

Now, go raise money from whoever you can. If you are savvy enough in the "Techcrunch Web 2.0 scene" to raise seed money from pro Seed investors--think Dave McClure, etc--then go for it. Here's a quick list off the net:

http://ecpmblog.wordpress.com/2009/03/09/a-list-of-angel-investors-a-post-to-keep-a-running-list/

You can surely find more with searches like this:

http://www.google.com/search?sourceid=chrome&ie=UTF-8&q=silicon+valley+seed+investors+list#sclient=psy&hl=en&source=hp&q=list+of+angel+investors+in+silicon+valley&pbx=1&oq=list+of+angel+investors+in+silicon+valley&aq=f&aqi=&aql=&gs_sm=e&gs_upl=19938l20481l2l20733l5l5l0l0l0l2l177l581l0.4l4l0&bav=on.2,or.r_gc.r_pw.&fp=104ae468c6e9bd24&biw=1440&bih=790

Keep in mind that all these seed investors want the primary founder who will put this project on his back to be technical. They invest in people, not ideas. You're going to have to be damn good at product speccing, managing coders, maybe graphic design, and dealing with people if you're going to pull this off. If they do invest, it will most likely be because of the up and coming team you put together.

Side note: if it's not clear, the reason I'm not recommending you go after traditional Venture Capital firms, like the ones in this list:

http://www.livefromsiliconvalley.com/venture/venture.html

is because these guys don't invest in something until it has real users and/or is making money, or you and your team are already super established, coming from Facebook, Google, etc.

...Now, if and when those Seed investors don't bite, you're just going to have to raise enough money from somewhere to build your MVP ("Minimal Viable Product"). Your

site and specs/designs should greatly help here of course too. The trick will be the more prepared you are the better. Prepare the contracts, and setup professional meetings where you're invite your potential coders, and generally make it look like you know what you're doing. At this point, you have everything to start, except the money to pay coders. Your spec should be exceptional by this point in time. After talking to all those seed investors, you probably learned a bunch more about your product. So you may need to re-spec, and re-hire you're designer to hook up the graphic designs for cheap. Overall, you should be simply more prepared to start executing.

Try to come up on at least $50k, and give your coders tons of equity, and have them do it part time. The key thing here is don't go offshore it. Just don't. Really focus on getting those coders in your city to really fall in love with it that when you offer them some decent cash and excellent equity that they'll devote serious time when not on their day job executing. Make sure you've reduced your spec to as little as possible, and followed all the FaceySpacey Speccint Tutorials, etc, and generally make it so your coders don't have to do much and will get a good amount of money from you per the amount of time they have to work. This will guarantee it actually gets done.

Now, when you're done--which is easier said than done ;) due to how hard it is to execute software applications--go back to the same seed investors that you met before who rejected you and pitch them on how great you're doing because of all the real users you're using your launched application.

That's pretty much it. It's pretty basic. The main ingredient that you don't hear too often is to build that mini company site representing you and you're spec, and use that as a stepping stone. The point of company websites is to make you're self look bigger and more important than you are. I hardly ever see this done, but I've done it many times, and it works wonders. Potential investors will be impressed with how professional you are. It's the best you can do since you can't code.

Otherwise, I'm not gonna say things like go to kickstarter.com, because you should already know places like these. I have no experience with that site, but in general all these sites that help you meet potential investors I've found to be a waste of time if you can't execute yourself or have an amazing track record. You're going to have to go through contacts you already have (friends and family, friends of friends, old bosses, etc) to get that seed funding in a worse case scenario. I'd skip the grandiose hope of some stranger across the web funding you're startup, and just get to convincing everyone you already know with your beautiful site, spec, and other presentation materials.

The more materials you can present, the better--provided they are concise and in video format. Huge business plans, hockey stick graphs, etc are not used in this industry of inventions where you really have no idea how things will turn up and will ultimately have to alter your product many times as you iterate and learn what users really want. That's how the pro seed investors think. They invest in people, specifically people that have the skills and fortitude to keep chipping away at their product according to user feedback until they get it right.

That said, maybe the less sophisticated investors in your network may be impressed by the hockey-stick curved graphs in your metrics, even though you'll never actually attain them. And the same with huge business plans. But I find that short and poignant videos that hit the spot really get people going--all people across the board, professional investors and less sophisticated investors in your extended family. Videos show that you can put a complete something or other together. It's a complete package, and that deserves merit. A huge business plan requires reading several times over many hours, days, weeks, to know if it's really quality work, or just a bunch of words jumbled together to fill up as many pages as possible. Business plans for startups aren't taken seriously. But product is.

Any product you can deliver will help you get funding, and high quality video production certainly is product at this stage. You get the idea--package something, anything, up, and use it to pitch, and if it doesn't work, figure something else to package up. Try a site. Try some videos. Try a powerpoint. Try the business plan. Try an in person presentation with a pointer using your videos, powerpoint, designs, specs and site, etc. Completing all of these will just make you look even more and more prepared to handle the actual execution of your product. If you're that determined, you'll figure out more and more that you can do to prop yourself up. And if worse comes to worse, take a year and learn how to code. I've seen startups spend 2-4 years raising funds because they can't code. They should have just learned to code and coded it themselves!

10-7 BUSINESS FUNDING & LEGAL | STARTUP CONTRACTS

First off, Silicon Valley VCs don't invest in LLCs. So you will need an Inc. The second gotcha you need to know as an entrepreneur is that it's not just about the percent of equity you own. Read the below article from Techcrunch:

http://techcrunch.com/2011/07/25/one-book-every-entrepreneur-and-vc-should-own/

Take note of this line from that article:

"VCs are anal about things like voting thresholds, seniority of their stock, protective provisions, etc. – entrepreneurs never seem to focus on anything other than ownership percentage."

There's a reason VCs are concerned about all these other details. Imagine you have stock in a company and you aren't able to sell it when you want to, and you basically are last in line to be able to sell it.

So you need to get yourself versed in all the common terms that can and should appear in these contracts. If you know them, you can use them to your benefit.

That said, there are popular standardized contracts used in Silicon Valley. Here are the 2 biggest ones:

1) Created by The Funded:

http://techcrunch.com/2009/08/23/the-funded-publishes-ideal-first-round-term-sheet/

http://www.docstoc.com/docs/10303638/FFI---Plain-Preferred-Term-Sheet

2) Created By Y Combinator:

http://techcrunch.com/2008/08/13/y-combinator-to-offer-standardized-angel-funding-legal-docs/

http://www.ycombinator.com/seriesaa.html

You should also read the following two related articles from New York's most famous tech VCs (Chris Dixon and Fred Wilson):

http://www.avc.com/a_vc/2009/08/the-ideal-first-round-term-sheet.html

http://cdixon.org/2009/08/16/ideal-first-round-funding-terms/

The whole purpose of these contracts is not to go on forever and nail the must-haves for a first round of funding, but there are some complex clauses you may not be familiar with. Read them, absorb as much as you can, and google the points you don't understand. It's ultimately pretty basic.

10-8 BUSINESS CONTRACTING | HOW TO GET IN & OUT OF CONTRACT DEVELOPMENT

So check it, you want to start your own software/web development business, right? The way to do it is all about building your portfolio and client roster one project at a time, and then use each past project to get an even better next client.

The way I got into contract work is because I ran out of money and got into severe debt trying to make my own startup happen. So what I did was use it, ReelProperties.tv, as the first portlier project in my newly created (at the time, '06) web development company, FaceySpacey. I got several small clients at the time, but one major client, SweetSubmit.com, which happened to be in the mortgage industry. ReelProperties.tv is what enticed the client to work with me--i.e. because of the close connection between the real estate and mortgage industries.

If you're new to web development, you're probably insecure about your portfolio projects. But the fact of the matter is: you have to work with what you have. Build your own company site, and make one portfolio project in there look like a million bucks--as big as you can. Give it a huge banner on the homepage, and write a case study and link to it, etc. Take that and go get your next client. By the time you have two or 3 clients/projects, you'll be able to make it look like you have 3 featured projects in your portfolio, and you're just not featuring the rest. When I got to 4 projects, i redid the FaceySpacey site in 2007, and made each one of those projects the feature project of 4 different categories of work we did (i.e. Facebook Apps, Flash Widgets, etc). So to the visitor, it felt like I probably had more projects stashed in my portfolio, but was only showcasing 4.

After that, you'll just want to redo your site a 3rd time to accomodate the tens of projects you have. By that time you should be home free, established with your own web development company.

If it's not clear what's going on here, you need to basically do what you're supposed to do a good job doing for your clients, for yourself! That specifically is making you appear bigger than you are. That's the whole purpose of websites and software in general. It's too get a lot done with a little--to create efficiency. So you need to very efficiently make your fledgling web development shop look way bigger and more professional than it is. Ultimately, that's what your clients will want when they come to you to build a website for their new law firm, restaurant, consultancy, etc. They want you to make them look like they worth the high prices they want to charge.

Now, as for how to get out of contract development, the trick is to basically join one a startup that comes your way as a principal participant. In reality, to do this right, you'll want to build a special relationship with one of your clients--to the point where you'll come up with an idea together, or perhaps totally alter his/her original idea. Then you'll get yourself an equal chunk of equity to him/her and roll out together. They handle the funding. You handle the labor.

Another way to do it, which is the way I took, is I just saved enough money to work for myself for long enough until I could make my own software successful. It's a harder road because it took a lot of time to handle my financial problems, but I think it's very important for software developers to be able to take their own time to go at their own pace at their own idea without someone who can't code who just provides the money looming over your head. The reason is because you're the true creator. At every step of the way, you're making decisions about your application and how it should function--all decisions that your non-technical partner will never know about or be skilled enough to help with. Yet, they get all the credit because they brought a nugget of a concept. They better at least be bringing the money if they're contribution is fucking ideas! Ideas are worthless. Your execution is everything, and money is valuable as well.

Either way, my point is that you need time to work for yourself, and yourself alone. Many developers are able to complete products for clients, but can't for themselves, and have tons of half-finished projects. As human beings, we definitely revolve around people and function better when we have people whose opinions we care about and don't want to let down. That said, when you work for yourself on your own idea and you complete something all for yourself it's almost good for the soul. You really just did something for yourself.

You may want to start with small plugins and widgets because it's very common to give up on your own project if it's really large. Or you may just want to go for a large project of your own. The main thing is you're going to learn how badly you really want it, and if that particular project is really what you should be focusing your life on. Does it represent what you're really about? Is it how you want to be known to the world? Is it what you want to do for the rest of your life? Are you doing it just to get rich quick? In another article, I'll examine the importance of choosing a project to do not just based on money, but all the other factors in your life that will hopefully bring you true happiness. For now, I'll just say, you're going to find out what you're made of more than ever when working on your own project. Hopefully by this time, you have enough skill that you're very prepared to do it. I think for most, though, it will force you to re-evaluate what you're truly about since you now know how much work these sort of applications take. Many may never up making this project super successful, but they'll be prepared to go for the gold on a project that truly represents them and makes them happy to work on it the next time around. And by that time, you'll be truly out of contract development and on to running your business that you just love to grow, improve upon, and be committed to for a long time.

10-9 BUSINESS CONTRACTING - TOP 10 EASY SITE CREATOR TOOLS

If you're not a coder, it's imperative you're able to build a professional site quickly for yourself, friends and associates. Here's a list of my favorite site creator tools. These tools will allow you to build a website basically by pointing and clicking, and dragging and dropping.

1	http://www.squarespace.com/
2	http://buildorpro.com/
3	http://flavors.me/
4	http://www.devhub.com/
5	http://www.yola.com/
6	http://snappages.com/
7	http://page.ly/
8	http://www.edicy.com/
9	http://www.webnode.com/
10	http://www.yola.com/

10-10 BUSINESS CONTRACTING | MUST-KNOW SaaS TOOLS

Below is a quick list of extremely useful tools I use on the regular.

1	MailChimp.com	email list marketing
2	SocialFlow.com	twitter marketing
3	Yammer.com	internal company communication
4	ViralHeat.com	social media monitoring
5	FreshBooks.com	Invoicing
6	Google Apps	email management (i dont use the other features)
7	CrowdFlower.com	crowdsourcing of data-entry tasks

Conclusion & Further Reading

We at FaceySpacey hope you've enjoyed our FaceySpacey Bible, and are coming away many times more ready to succeed at your next software startup. At the very least, you should have a birds-eye-view of what you need to do to get your startup, and have quelled a lot of insecurities you may have had regarding how you should execute it. That said, I will point to you to what you should do next.

As promised, the following is a list of the precise books I read to master web development using HTML/CSS, Javascript, PHP & MySQL. They are presented in the best order to most efficiently learn the subject at hand. It's similar to the order I read them in, but enhanced based on what I learned and the order I wish I read them in. Good luck:

HTML/CSS:

CSS Mastery: Advanced Web Standards Solutions

http://www.amazon.com/CSS-Mastery-Advanced-Standards-Solutions/dp/1430223979/

Before you start coding PHP, Javascript, etc, understand how HTML works. This is where you start. HTML is easy. Read this book in combination with studying the HTML & CSS tutorials on w3schools.

PHP & MySQL:

PHP & MySQL For Dummies, 4th Edition

http://www.amazon.com/PHP-MySQL-Dummies-Janet-Valade/dp/0470527587

This book--well an older edition--I read a year before I got serious about learning to code. I read it and didn't actually code anything i learned, but what it did was plant seeds in my head with regards to what programming is all about and what databases are all about, and how to connect the two. It assumes very little in what you may

already know, and is an excellent start in your journey to becoming a master programmer.

PHP Object-Oriented Solutions

http://www.amazon.com/PHP-Object-Oriented-Solutions-David-Powers/dp/1430210117

This book is where I learned what OOP is. I didn't get the hang of it until reading the following book. Don't worry if you read this and have a hard time with it. This book and the next each have introductory chapters that go over how OOP works. It took me reading basically this book and the next book about the same stuff to get it. This book is a lot less complicated than the following and dives into practical examples & problems, whereas the next is a lot more theoretical.

PHP Objects, Patterns and Practice

http://www.amazon.com/Objects-Patterns-Practice-Experts-Source/dp/143022925X

After reading this book, I basically mastered OOP. It's a very hard book to get through if you're new to OOP, and goes into some very advanced stuff, specifically tons and tons of "design patterns." The design patterns are presented in as basic of a form as possible, but they weren't very practical like examples from the previous book in that you probably will never actually need any of the code used in the book. Either way, this is my favorite Programming of all time because it taught me how to think like a coder and how to solve complex problems with concise refactored solutions. It really showed me what is possible with OOP. You don't know PHP unless you've read this book.

Pro PHP: Patterns, Frameworks, Testing and More

http://www.amazon.com/Pro-PHP-Patterns-Frameworks-Testing/dp/1590598199

I read this book too just to solidify my experience with PHP, and cover all my bases. Check it out. It's optional.

PHP Functions Essential Reference

http://www.amazon.com/Functions-Essential-Reference-Torben-Wilson/dp/073570970X

At some point during my study of PHP I found this book and decided just to learn every PHP function available so that I could better understand the examples in the above books. Start reading this early on, and complete the whole thing. You'll quickly learn patterns in how PHP functions are named, and as a result be able to guess what a function does within the context of the examples in the above books--even if you don't remember precisely what it does.

Agile Web Application Development with Yii 1.1 and PHP5

http://www.amazon.com/Agile-Web-Application-Development-PHP5/dp/1847199585

I read this book in combination with reading the Blog Tutorial and Definitive Guide on YiiFramework.com. When you're done studying all these materials, you'll be amazed with how much power you have. This book isn't hard to read either. You'll love it if you reach this stage!

Javascript & jQuery:

Learning jQuery, Third Edition

http://www.amazon.com/Learning-jQuery-Third-Jonathan-Chaffer/dp/1849516545

jQuery is a framework built on top of the native browser language of Javascript. Usually one would recommend you learn the base language--Javascript--before learning an abstracted framework on top of it--jQuery. However because of the nature of jQuery and how comprehensive it is and because of how quirky Javascript is coming from PHP, I found it best to jump to jQuery and immediately start accomplishing the DOM manipulation tasks I needed. And ultimately because of syntax similarities between PHP and Javascript I was able to get productive in Javascript without studying a single book just on Javascript. One thing I did different

when studying this book from the PHP books is I did every single tutorial as I read it. The reason is because when I learned PHP, I was learning my first real programming language--so it took me a lot of time to just digest things before I could code a single line, which is why I just read PHP book after PHP book before I got started until it all made sense. However, by the time I got to Javascript & jQuery, I understood how programming in general works and found it helpful for memorization purposes to immediately start doing the tutorials.

jQuery 1.3 with PHP

http://www.amazon.com/jQuery-1-3-PHP-Kae-Verens/dp/1847196985

With this book I didn't do all the tutorials like I did with *Learning jQuery*, but what reading this book did for me is taught me precisely how Ajax works and what it's all about. After reading it, coding features that required Ajax using Yii and PHP was obvious and a no-brainer.

JavaScript: The Good Parts

http://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742

This book gave me a deep understanding of the Javascript language and what it's truly all about. After reading it, many hours of debugging and head-scratching when coding Javascript & jQuery were removed from my schedule--because I finally learned the quirks of the Javascript language I needed to know.

Pro JavaScript Design Patterns

http://www.amazon.com/JavaScript-Design-Patterns-Recipes-Problem-Solution/dp/159059908X

Now this book took my Javascript game to the next level, gave me an idea of how jQuery was built, taught me how to do things similar to how you would in a "classical" OOP language like PHP, and completely ended any remaining head-scratching I was having with Javascript, particularly with how "scope" works in Javascript.

Linux:

The Official Ubuntu Server Book, 2nd Edition

http://www.amazon.com/Official-Ubuntu-Server-Book-2nd/dp/0137081332

Note: by the time I read this book I had already learned Linux through blogs on the internet. The best thing I can recommend you do is install Linux on your computer from the Ubuntu website, and start navigating around the command line, practicing Linux commands you learn off the web. Just google "linux tutorials" and you'll be off to a running start. That said, by the time I got proficient in Linux and after I read this book, I felt confident that I really knew what I was doing and had practical solutions for the most common problems you'll face at the command line.

Apache Cookbook: Solutions and Examples for Apache Administrators

http://www.amazon.com/Apache-Cookbook-Solutions-Examples-Administrators/dp/0596529945

This book I treat like a pocket reference and still refer to it often since it's impossible to remember all the different Apache configurations, given how comprehensive this web server application is. I did read it through when I first got it. I kinda skimmed it though--just to get an idea of what is possible. Getting an idea of what is possible without mastering a subject matter is so important in programming because you'll know where to look when you face a challenge that the subject matter can solve.

Pro Bash Programming

http://www.amazon.com/Bash-Programming-Experts-Voice-Linux/dp/1430219971

This is a little advanced for readers of the *FaceySpacey Bible*, but I'm putting it here because it really took my Linux skills to the next level.

NON-TECHNICAL BOOKS:

Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent

http://www.amazon.com/Smart-Gets-Things-Done-Technical/dp/1590598385

If you plan to grow a small application into a large company, this book is a must. It's short. Read it.

SEO Book.com

http://www.seobook.com

This obviously isn't a book, but I read their entire site like a book, and its creator, Aaron Wall, expects you to read it like a book. When I was done reading it, I felt I was completely up to speed regarding what SEO is, how search engines work, and practical techniques to get better rankings in search engines.

For daily Startup Wisdom, checkout FaceySpacey.com/blog daily. And don't forget to download the entire No Bullshit FaceySpacey Bible or more individual chapters here: http://www.faceyspacey.com/resources?section=book .

*Thanks again from FaceySpacey and be sure to check out **FaceySpacey.com** often for further knowledge we kick to take your Startup to the Stars!*