

developer.***A Web Magazine for Software Developers**

Your Own Software Development Company

By Daniel Read

A reader named Turki Al-Aseeri from Saudi Arabia wrote to me this week. After passing along some nice compliments, Turki posed the following question:

I'm about to open my own software development company. What could you tell me to keep in mind?

Each of us who works as an employee or contractor for someone else thinks about starting our own company from time to time—some of us more than others—but not everyone has the temperament for it. I myself, at this stage in my life, very much enjoy the freedom that comes from working for someone else. I go to work, I do my best to deliver a good value to my employer and support my fellow team members, and I go home. Work stays at work. But the siren song of “true independence” is always there, and I'm sure someday I will make the leap. When I do, I hope I can keep my own advice in mind, and remember the lessons I have learned helping and watching other people run a software company.

I learned much of the following while working for my friend and mentor, Bryan Sedwick, who has owned his own software company for several years. Many of these lessons have also come from my experiences as a contractor and employee for small companies and large corporations. I think much of this applies whether you work for yourself or for someone else, but most of the comments below are aimed at a developer who is thinking about going into business for him or herself.

Tell the truth. Don't lie to your customers, don't lie to your employees, don't lie to your subcontractors, don't lie to your suppliers, and most of all, don't lie to yourself. If you don't have your integrity, you have nothing.

Do your best to manage expectations. Most misunderstandings and disputes can be traced back to a disconnect between what one person expects to get, and what the other person expects to deliver. Managing the expectations of your customers, employees, suppliers, and subcontractors is something of an art, but the basic technique involves being as explicit as possible as often as possible.

Make quality a very high priority, but keep in mind that quality is relative. You should start by deciding what quality means to you. Your own barometer for quality might be based on the clarity of your code, the amount of documentation you produce, the performance and scalability of your system, the elegance of your database design, or all of the above.

However, you must keep in mind what “quality” means to other people: your investors, your employees, your customers, and the employees of your customers (that is, your

users). You must balance your quality concerns with those of these other “stakeholders.” You must simultaneously be willing to give a little on your own quality standards to meet someone else's and also know when to refuse to compromise because it's the right thing to do. Sometimes you have to protect people from themselves. To quote Gerald Weinberg:

“Quality does not exist in a nonhuman vacuum....Every statement about quality is a statement about some person(s)...The definition of quality is always *political* and *emotional*, because it always involves a series of decisions about whose opinions count, and how much they are relative to one another.” (1)

Focus on your customers, because they are your lifeblood, but know where to draw the line. Don't let your lifeblood become bloodsuckers. Your customers will try to rule your life, but you must not let them. Don't give in to every demand they make, or they will make more and more demands. Be willing to let go of a customer who is such a pain that they are not worth it. Don't let a customer know how much you need him, because he will try to use that as leverage to make you do extra things for him. Do everything you can to ensure that you are not so dependent on any one customer that the loss of his business could destroy you. (That's hard to do, especially at the beginning.)

Set reasonable boundaries with customers, and stick to them. If you start giving away free stuff, then you will set an expectation for free stuff. Put things in writing, and get the customer's signature on a contract. Get some money up front, and work on multiple-milestone deliveries, with additional payments due at each milestone. Price yourself and negotiate as though you value yourself and your abilities, or your customers will not respect you and will try to take advantage of you—and certain customers they will not hire you at all. I don't mean to sound so negative about customers, but customer management will make or break you.

Court your customers, and develop personal relationships with them. Get to know them—their likes, dislikes, hobbies, family details, etc. Do some free stuff for them. Show up one day and give them something they did not ask for. If you're courting a big client, or developing an ongoing relationship with one, see what books she has on her shelf, and read them. Show them with your actions that you are thinking about them, and that you are responsive to their needs. When you come across a magazine article or web site that you know would interest them, send it their way.

Let your customer know that you have their best interests in mind, and show with your actions that you aim to produce a quality product that meets their needs. When speaking with business people like CEOs and owners, demonstrate (either directly or indirectly) that you are aware that their primary goal is to make money or deliver value to their stockholders, and the product or service they need from you is necessary to help them make or save money. Always smile, and never let your customer see you sweat. Confidence smiles. Smile in person, and smile on the phone. A positive attitude is essential.

Do what you say you are going to do. This has got to be the most common mistake that people make with their customers, and it is probably the most deadly. How many times have you, as a customer, dealt with a lawyer, plumber, software vendor, employment recruiter—whatever—and had that person *not* do what they said they were going to do? Maybe they promised they would call you the next day, or that they would e-mail you some information, or that the check was in the mail. What judgments did you make about that person at that time? How mad or annoyed were you? How fast did you blow them off and go do business with someone else? How many times did you let them get away with that before you dropped them? How many people did you tell about your bad experience? Turn the tables, because now *you* are the vendor, and you better make sure you don't drop the ball.

As we say here in America, cover your ass. Keep a diary of your daily activities: what you worked on, for whom, and for how long. This is especially important if you are billing by the hour. Also, when you write database software, always put a created, modified, and deleted timestamp on every record, along with the user id who created, modified, or deleted a record. Don't ever delete a record—mark them for deletion with a flag, and then archive them periodically.

Someday your customers are going to screw themselves up, and they will come after you, nostrils flaring, blaming your software for deleting or corrupting their data. When you can go into the data and "undelete" the records, or show them which user changed the price of a product record, then one of two things will happen: a) you will be the hero, or b) the hostile customer will be speechless in the face of your proof that Mary Smith changed the price of that product two weeks ago at 11:32:45 PM. Another way of putting this would be to say "Protect your customers from themselves."

Write everything down. More from the "cover your ass" department...write it down, and where necessary, get your customer to sign off on what you are going to do. This goes back to managing expectations, and making things explicit. Keep notes from phone conversations and meetings; save e-mails, faxes, and other correspondence; put everything in a "project folder," which is just an organized file folder. If you really want to be thorough, keep a log of everything you add to the project folder and what date and time you added it.

Beware of customers who will ask for complicated bids and analyses, with no intention of hiring you to do the work. Generating bids is a fact of life for a software consulting company, and you of course cannot charge customers for every bid you produce or you'll get laughed out of the business. However, there is a line, and when generating a "bid" is really a process of spending several days in someone's office analyzing their business processes and generating a written proposal for a solution, including hardware recommendations and a detailed cost breakdown, then you have gone beyond simply producing a bid, and it's time to ask the customer to break out the checkbook for your services and expertise.

When a customer asks you to build something for them, don't ever build anything for one-time-only use. Always design and build in a reusable, "componentized" manner. For example, my friend Bryan owns a company that, among other things, makes web sites for people. The first web site they built, they did not just build the site, they built an *engine* for building web sites, and they have re-used that engine dozens of times for many more clients. In fact, now they've enhanced the engine to the point that customers can build and maintain their *own* web sites. The original *developer.** web site was produced and hosted in this very same engine. When you build something, make sure it can be reused or leveraged in some way in the future. Time is money.

Be (or strive to become) a well rounded person. If your sole areas of knowledge and experience are writing code, building networks, and other such esoteric technical subjects, then you may have a hard time relating to your business-oriented customers. You have to convince these people that you understand (or can understand) their business well enough to build proper software for them. For example, if you are going to build accounting systems for people, you better know what the real difference between a credit and a debit is, and be able wax philosophical about cost vs. accrual accounting.

In my opinion, it really helps to be well-rounded in two ways: from a software/systems standpoint and from a business/social standpoint. You have to have a good set of technical skills that runs the gamut from requirements gathering to design to technical writing to coding to hardware design to training to deployment. Where you have weaknesses (we can't all be experts in everything), you will need to team up with people who complement you.

A strong range of technical skills will help you build great software for your clients. On the other hand, a well rounded education and conversational ability in general topics such as business, sports, literature, current events, etc. will help you land clients and develop long-term relationships. The more subjects about which you can hold an intelligent conversation, the more people you will be able to relate to. People like to work with people who are like themselves.

Advice from an expert. I sent an early version of this column to my friend Bryan Sedwick and asked him to offer some advice of his own. He was nice enough to write in the following:

The biggest issue I had to overcome was the "cycle" part of business. You end up doing the same thing over and over again. This has a tendency to get boring really fast, but that is where the profit is. If too much time is spent redoing what you have already done, then profit suffers. The question that has kept me alive for years: how can I use something I already have to make another buck?

Dan, your point in the article to build once and then use and reuse is right on the money (pardon the cliché). It is new once but resold and profitable for years to come. Documentation doesn't change; support doesn't change;

customers learn how to do-it-themselves, which reduces or eliminates support costs.

I have found that "artists" are not the best business owners (unless they hire someone to drive the business). They seem to be all about creating. There is little profit in creating. (I am sure this point would get argued by creators.) However, there is a lot of profit in the same old boring stuff we did last year. This is not to say forget about new products and new solutions to keep your company competitive or viable. But, "new" cannot be the main focus. Artists/developers have to give up what they love to be successful—what irony.

When you desire to be on your own, make sure you're ready for the boring, cookie-cutter methodologies: search for a lead, cultivate the lead, needs analysis, Functional Requirements Document, Detailed Design Document, Project Plan, Contract, Signing, Development, customer acceptance, COLLECTIONS—lather, rinse, repeat as necessary.

When the dating turns into marriage, you have to constantly guard against other suitors (competitors), and protect your long-term service revenue stream. Customers are incredibly hard to get and very easy to lose.

Bitter irony. As Bryan suggests, be prepared to spend lots of time doing things besides building software. Many software developers who start their own consultancies are not prepared for what happens: much (if not most) of your time starts going to things which seem to have nothing to do with building software. And 9-to-5 workdays are out the window. Be sure you know what you are getting into. If building software is what you love, and that's what you want to spend most or all of your time doing, then starting your own business might not be the best route to take.

However, if you have an independent streak, and all this talk about selling yourself and developing customer relationships makes you gag, then your best bet might be to find someone who already has an established consultancy, and work with them instead. You can still work on your own, and be in business for yourself, but you'll be more of a subcontractor. There's a risk in that, since you're depending on someone else to bring the work to you, but there's risk no matter what. Besides, people who are really good at selling and developing client relationships generally have more work in the hopper than they know what to do with.

Reading back over all of the above advice, it occurs to me that I've painted a somewhat negative, if not bleak, picture. This was not my intention. Far be it from me to discourage anyone from chasing their dreams. If there were not great benefits to going into business for yourself, then no one would do it. When you are in business for yourself, there is no ceiling on what you can do, and there is no one to answer to but yourself (well, maybe also your spouse). You make the rules, and you can break the rules anytime you want. If you

want to take off in the middle of the day to go for a hike or go see your kid's school play, you're free to do so. The benefits are there, but unfortunately not without a cost.

References

(1) Gerald Weinberg, *Quality Software Management Volume 1: Systems Thinking*, (New York, NY: Dorset House Publishing, 1992), pp. 5-7

###

Daniel Read is editor and publisher of the **developer.*** web magazine. He lives in Atlanta, GA, where he works as a software architect. He is currently at work on a book about software development crafted for a business audience.