This paper was originally presented at the Southwest Fox conference in Mesa, Arizona in October, 2007. http://www.swfox.net

So You Want to be an Independent Developer

Session Number 20

Rick Borup
Information Technology Associates
701 Devonshire Drive, Suite 127
Champaign IL 61820
Voice:217.359.0918
Email: rborup@ita-software.com

Hanging out your own shingle as an independent developer can be the scariest yet ultimately the most rewarding experience of your career. What things should you consider before making this momentous decision? Once you've done it, what steps can you take to help ensure success? (Hint: writing good software is not enough!) In this session, Rick Borup brings real-life experiences from fourteen years as an independent developer to help those thinking about or already embarked on the exciting and challenging journey as an independent developer.

What is an independent developer

The term *independent developer* probably means different things to different people. For this session, however, I have a very specific definition in mind. An independent software developer is a software developer who:

- is independently employed
- develops software as his or her primary source of income
- typically develops software for a variety of different clients, and
- personally performs or directly supervises all aspects of the software development process.

By way of analogy, the independent software developer is to a software development project as the architect and general contractor together are to a construction project. In other words, the independent software developer is responsible for the entire scope of activities involved in the software development process, from conceptualization of the design through realization of the end-product.

Independent software developer

Being independently employed simply means not being employed by someone else. Independent software developers might be self-employed owner/operators of a sole proprietorship, or they might be an owner and principal employee of a one- or two-person company. In the latter case, the independent developer may technically be an employee of the business, but because they are also a business owner they are effectively self-employed. Either way, being independently employed means not being an employee of a business you don't own.

Independent software developers do software development work as their primary line of business and as their primary source of income. Contrast this with the owner of a small business engaged primarily in a non-software related activity, say a lumber yard or hardware store, who might write some software for their business on the side. Even if that person sells a copy or two of their software to others, they are not really independent developers because software development is not their primary business or their primary source of income. People like this are a potential source of business for real independent software developers, though!

Part of what keeps an independent developer independent is that they're not dependent on one source of income. Successful independent developers therefore typically do work for more than one client. It's possible to be an independent developer who works for a single client, but that scenario fits the definition of a contract employee more than an independent developer.

Finally, an independent developer performs or directly supervises all aspects of the software development process. This encompasses everything from the initial consultation and requirements analysis through the actual design, development, and deployment of the software, and often continues with ongoing support and enhancements after initial deployment. The independent developer may perform all of these activities personally, or, depending on the scope of the project, may directly supervise others who perform some of these functions. Either way, the independent developer acts as the general contractor for the entire project. Contrast this with the contract programmer, who, while possibly self-employed, performs only a limited subset of a software project's activities and who likely works only for a single client at a time.

Micro-ISV

You may have heard the term Micro-ISV, which was coined by Eric Sink in 2004 to refer to a one-person software company.[1] A Micro-ISV, or independent software vendor, is similar to an independent software developer, but from my perspective they're not quite the same thing. While both are small or even just one-person software shops, the key difference is that a Micro-ISV develops a specific product with the intention of offering it for sale on spec—that is, on the speculation someone will want to buy it once it's completed. By Eric's definition, if you don't have a product you are not an ISV.

An independent software developer, on the other hand, may as likely be engaged in developing and supporting unique, custom software for a each of several individual clients as to be engaged in developing a product for sale to a broader market. The term independent software developer is therefore broader and more encompassing than the term Micro-ISV. In other words, a Micro-ISV is an independent software developer, but not all independent software developers are Micro-ISVs.

In his chapter on *Best Practices for Vertical Application Development*[2], Doug Hennig explains the difference between vertical, horizontal, and custom software applications. A Micro-ISV would typically develop and market a vertical application, which is software targeted for a specific niche or industry. An independent software developer might develop and market a vertical application, but he or she might also develop and support custom applications, which a Micro-ISV would not do.

The term Micro-ISV has also been used to mean small *Internet* software vendor, meaning a small software business that sells its product over the Internet. Bob Walsh uses this definition in his book Micro-ISV: From Vision to Reality. While this session is not specifically about Micro-ISVs, Walsh's book is on my recommended reading list for aspiring independent software developers. See the Resources section at the end of this paper for the complete list.

Independent pros and cons

Being independent certainly has its advantages, but it's not all peaches and cream. If you're considering becoming independent, it's helpful to acquire an informed perspective about both sides of the equation before making the commitment. The following is not a comprehensive list, but it should help to get you thinking in the right direction.

[1] *Exploring Micro-ISVs*, Eric Sink, MSDN, September 2004, http://msdn2.microsoft.com/en-us/library/ms995817.aspx

[2] *Best Practices for Vertical Application Development*, a chapter in Visual FoxPro Best Practices for the Next Ten Years, multiple authors, Hentzenwerke Publishing, 2006

Benefits of being independent

When you're an independent software developer, there's nobody but yourself to tell you what to do every day. Sure, you have clients with specific deadlines and requirements, but in the larger sense you get to decide how to manage your workload, which projects to work on first and which to defer, how much time to spend on billable hours versus how much to spend on administrative tasks and business development activities, and even which projects you want to accept and which you may want to decline.

Being independent offers an incredible degree of freedom as compared to being a corporate employee. For better or worse, you are your own boss. You get to set your own schedule. You get to decide where and when you're going to work. If you're a one-person shop, you don't have to attend any company meetings, deal with a pointy-haired boss (unless you are one), or sit in a cubicle (unless you want to).

As an independent developer, you don't have to worry about navigating the treacherous waters of corporate politics or spend time thinking about how to advance your position within the organization. You *are* your business, in every and all respects.

When you're a one-person shop your business exists for the sole purpose of generating income for yourself. You don't have to worry about meeting a payroll other than your own. Perhaps best of all, you're never going to be fired, laid off, downsized, outsourced, off-shored, or otherwise unpleasantly blind-sided by an employer.

For better or worse, it's all up to you.

Downside of being independent

At the risk of sounding preachy, with great freedom comes great responsibility. The fact there's nobody to tell you what to do every day cuts both ways: it's entirely up to you to figure out what to do to ensure the success of your business. When you have a boss you may not like what he or she tells you to do, but it may in fact be in the best interest of the business even if it's difficult for you to see it that way. When you are your own boss, you make all the decisions and effectively end up telling yourself what to do. Nobody's going to second guess you but neither is anybody going to advise you, and the only indication you've made a bad decision may be the loss of a client or an unprofitable business.

To be successful as an independent developer you have to make a mental shift and get rid of the concept of "they". As in, "they" are making me work too hard; "they" aren't paying me enough; "they" don't appreciate my true talents or my real worth; it's not my fault because "they" told me to do it this way. When you're independent, there is no more "they". *You* are they. You get to take the credit and reap the financial rewards for all success, but you must also be accountable for and bear the financial penalty for all failure. It's called accepting responsibility, and it's not always easy to be honest with yourself about it.

When you're a corporate employee you get a steady paycheck. This is probably not going to be true when you're an independent developer, particularly at first. You can make a lot of money as an independent developer, quite possibly more than you could make as a corporate employee, but there’s risk involved and the income stream is not going to be as steady as you may have been used to.

On top of that, there’s a certain amount of financial overhead involved in running your own business. You’re going to be paying for your own computers, your own software tools, your own education and training, your own business insurance, and your own office expenses, as well as paying other professionals for services such as accounting and legal from time to time. You will also quite probably be footing the bill for your own health insurance and funding your own retirement. These are not insubstantial amounts, and they’re a big part of what you need to take into consideration as you think about launching your own business.

The big decision

You’ve looked at the pros and cons. You’re pretty sure you’re ready to go independent. How to you actually make the big decision? How do you answer the question, “Am I really ready to do this?” The answer to that question has several dimensions, which this section should help you explore.

Financial readiness

In the last section, I discussed some of the financial considerations involved in running your own business. As you prepare to make the leap to independent developer, take the time to put pencil to paper and write down your best estimates of what your startup costs are going to be, what your initial ongoing expenses are going to be, and what your initial income stream is going to look like.

Take into consideration your personal and family financial situation. Are you the sole source of income for your family, or does your spouse have a steady income that can meet your family’s non-discretionary expenses while you build up revenue flow from your business? Do you have substantial savings you can draw on if necessary for business needs or even for ordinary family expenses until the business is standing on its own?

It’s not difficult to estimate your business and family or personal expenses, but you’re also going to have to make some best-guess projections about how much money your business will take in during its first few months of operation. It’s dangerous, maybe even fatal, to be overly optimistic here, but don’t be too pessimistic, either. The obvious worst case scenario is there will be no income at all, but this is not realistic. Neither is it realistic to plan on the best case scenario in which you'll start making money hand over fist starting in the first month. Pick a number you think you can reach and a time frame you think you can reach it in. If necessary, do some research to support this estimate.

Put your estimates of business net income together with your estimates of personal and family expenses and you should be able to calculate your MTBU, or maximum time to belly up. Your MTBU, as I’m using the term here, is the longest period of time you can survive financially under the worst case scenario. If your MTBU is less than six months, you may want to think twice about whether you’re financially ready to launch your own business.

Admittedly, six months is an arbitrary number. If your tolerance for risk is greater, or your circumstances are compelling you to go independent regardless, then go for it. In any case, knowing what to expect will help you make better decisions and avoid unpleasant surprises.

Personal self assessment

Being an independent developer can place demands on you that you may not have anticipated. As with the financial considerations, being aware of these demands before you begin will help you make a better decision and increase your chances of success.

One of the main considerations here is, do you genuinely like working alone? As a one-person business you’re going to spend a lot of time doing things by yourself. If this is not appealing to you—for example, if you know you’d miss the casual, inter-personal interaction with other people during a typical office workday—then being independent may not be for you. If, on the other hand, you would relish the opportunity for sustained concentration on the task at hand without interruptions for unrelated activities, then you will probably find this aspect of being an independent developer quite rewarding.

Are you comfortable making decisions by yourself? As an independent you’re going to have to rely almost exclusively on your own judgment. Certainly you can seek advice and input from others—friends, family, other developers, even customers—but there’s not going to be a management committee, a board of directors, a technology steering committee, or any other group to discuss and vote on it. In the end, all the decisions are going to be up to you. Some people may find this ideal while others may find it stressful; think about where you are on this scale.

Do you think you have the self-discipline to work successfully as an independent? After all, nobody's going to *make* you get up in the morning. Nobody's going to *make* you spend x number of hours a day engaged in revenue-producing activities. Nobody's going to *make* you stay in front of the computer when it's a beautiful summer afternoon and you'd really rather be outside golfing (or whatever). So, take a minute to think about your ability to stick with a task, your ability to resist the temptation to sleep late or do fun things instead of working. When you're an independent, you have to rely solely on your own self-discipline.

Finally, do a quick assessment of your self-confidence. If you lose an important bid to a competitor, or if the software product you worked so hard to develop for the last six months doesn’t do very well in the market place, can you put it behind you and move on to the next project with a positive outlook? In other words, how resilient are you? All jobs have their challenges, but being an independent developer will likely test your resilience more than a conventional job.

What am I going to sell

Knowing what you're going to sell is fundamental to your business plan. Are you primarily going to sell software, services, or some combination of these?

Some of the questions to ask yourself here include: Do I intend to create a vertical market app and sell it over the Internet? Do I have contacts with one or two large businesses to whom I think I can sell my services as an independent developer? Do I know someone who's launching a startup of their own and who needs a good software partner? Am I only going to sell software, or will I also provide hardware and network support services? In addition to software, can I provide management consulting services to a client's IT department or even to the business owner(s) themselves? Developing a set of answers to these questions should help you decide what direction your independent software development business is going to take, at least initially.

One of the great things about being independent is you have the freedom to take advantage of new opportunities as they arise. While it's important to know what you're aiming for from the start, don't get hung up on making exactly the "right" decision right away. One opportunity frequently leads to another. Start somewhere, make it work the best you can, and keep your eyes open for new possibilities. They will come to you – I can almost guarantee it.

What market am I aiming for

Hand in hand with the question of what you're going to sell is the question of who you're going to sell it to. In other words, who or what is your target market? For example, some independent developers have been very successful building and selling software tools to other developers. Others market their services to IT departments that want to bring in outside expertise over the long term on particular projects. Still other independent developers market their software directly to end users, such as building and supporting software applications for small businesses or even selling general purpose software directly to individuals.

Who will be my first customer

Before you launch your software development business you should develop some idea of who your first customer is going to be. Let me qualify that: who your first *paying* customer is going to be! This might be someone you know, like a business that needs some custom software and with whom you have a good contact, or it might be a more abstract customer such as "all Visual FoxPro developers who are frustrated with the VFP Project Manager."

Regardless of how specific or how abstract, developing a good sense of who your first paying customer is going to be provides you with a meaningful target to aim for. As you develop your first software application, ask yourself what you can do to keep that first customer happy even if you don't know specifically who that first customer is yet.

Who is the competition

Take time to try and find out who you're going to be competing against. There are a lot of people developing software these days. It doesn't take a large investment to get started, and it's very possible that someone else has ideas similar to yours. You may never meet in face to face competition, but unless you occupy a very unique market niche you can be sure there are others out there doing pretty much the same thing you're doing and quite possibly talking to the same prospective clients.

If you're marketing a vertical app that's specific to a particular industry, it should be fairly easy to find out who else is selling similar software, what their software does, and how much they charge for it. It may be more difficult to find out who's using their software, but the Internet is a wonderful thing: a quick search can turn up a treasure trove of information. A great source of information in competitors' support forums, especially peer to peer forums. These are a great way to find out what users like and don't like about software they're using. Both the positive and the negative comments should provide ideas for things you can do—and ideally do better—in your own software.

If you're not selling a vertical market app but are instead focusing on custom software that’s unique to each business, it's probably going to be more difficult to find out who the competition is. Again, an Internet search is good place to start, as is a search of industry association publications and trade shows in your target market. On a more local basis, check the yellow pages: Who else is listed under the heading where your business fits? Find out all you can about who these competitors are and what products and services they offer.

Knowing who you're up against will help you assess the market potential for your software or services. This knowledge will give you a basis for determining how best you may be able to compete. For example, does it look like it's going to be better for you to compete on the basis of price, or service, or features in your software that others lack? Making a well-informed decision here can go a long way toward ensuring success.

SWOT analysis

On way to approach the decision making process for your new business is to frame it in terms of strengths, weaknesses, opportunities, and threats. This type of analysis is referred to as a SWOT analysis, a technique Wikipedia says came out of Stanford University in the 60's and 70's. I was first exposed to SWOT analysis in graduate business school in the 80's, and have found it be a very useful way of looking at many types of decision making processes.

In a SWOT analysis, strengths and weaknesses are internal factors. To the independent software developer, it's quite personal: these are things about *you*. Opportunities and threats, on the other hand, are external factors. These are things about the business environment, the economy, your competitors, and so on.

To apply a SWOT analysis to the decision of whether or not to become an independent software vendor, consider it in these terms.

- **Strengths** are the things you are good at, the things that will help you succeed. Ideally, these are areas where you can be better than your competition. This includes the obvious things such as technical skills—how well and how quickly can you write good software—but also less obvious things such as how hard are you willing to work, how effective are you at communicating and selling solutions to clients, how good are you at building and maintaining strong working relationships with clients over time, etc. Knowing your strengths can help you determine how to compete most effectively.
- **Weaknesses** are of course the things you are not so good at, the things that may hinder your success. Here again it can include technical skills; for example, you may realize you don't know as much about Web technologies as you think you should or as you think your competitors do. Weaknesses can also include less tangible aspects such as poor communications skills outside of a technical environment. This is a common problem among technical people that can hinder their ability to communicate effectively with important people in non-technical roles within a client's business, such as marketing people

and senior executives. You might also identify in yourself a tendency to avoid confrontations even when such avoidance is to your disadvantage. And so on. Be honest with yourself here: Knowing your weaknesses is as important as knowing your strengths.

- **Opportunities** are external factors that can help your business grow and prosper. Identifying opportunities helps lead you to market areas where you can compete effectively and grow your business. An example of an opportunity might be spotting the lack of and the corresponding demand for suitable software for a particular market niche, or the arrival of a new technology others are slow to adopt. Coupled with the appropriate strength, an opportunity defines an area in which you can invest resources with a reasonable expectation of making a profit.
- **Threats** represent factors that can be harmful to your success. This includes not only direct competition from others doing business in the same market segment, but also underlying changes such as shifts in technology or ways of getting things done that may adversely affect your business. Examples of the latter, in today's context, could include Microsoft's decision not to continue ongoing development of core VFP, as well as the apparent beginnings of a shift away from desktop applications and towards software as a service. By their nature, threats are outside of your direct control, but they are not outside of your ability to react and respond accordingly.

I believe it's worthwhile to perform a SWOT analysis on your business at start-up time and again on a fairly regular basis. You probably do this anyway, in an informal way and without even thinking about it in exactly these terms. But now that you know about the SWOT framework, you can use it as a guide in your ongoing decision making processes to help ensure long-term success.

Creating the business entity

Once you decide to launch your own business, you'll need to establish a business entity. There are four major types of business organization in the United States: the sole proprietorship, the partnership, the corporation, and the limited liability company or LLC. Which one you should choose depends on many factors including how many owners there are, what degree of legal separation you want between yourself and the business, whether or not you are going to seek outside funding, and in general what your long-term plans for the business are.

Regulations and requirements for establishing a business entity of any type vary from state to state. There are many resources, both online and in print, that explain in detail the differences between the various forms of organization and what’s required to set up the one you choose in your state. A trip to the business section of your favorite bookstore will almost certainly turn up one you like. In addition, it’s a good idea to consult with your accountant and your attorney before making the decision as to which form of business organization is right for you.

Following is a brief summary of the relative advantages and disadvantages of the various forms of business organization. Disclaimer: I am not an attorney, and what follows should not be construed as nor used as a substitute for professional legal advice.

Sole proprietorship

The sole proprietorship is by far the simplest form of business organization and the easiest to organize. For this reason, the sole proprietorship is probably the most popular form of small business organization when there is only one owner. If you’re doing business as a sole proprietor under your own name—for example, if your name is Bob Tucker and you're going to be doing business as Bob Tucker Software—you usually don’t have to do a thing except to start operating the business. If you’re doing business as a sole proprietor under an assumed name—for example, Pretty Good Software Company—you generally have to publish a notification of assumed name in a local newspaper. Either way, you and your business are one and the same.

All profits and losses of a sole proprietorship flow through to the owner’s personal income tax return. The owner takes his or her salary in the form of draws, which are simply a transfer of funds from the business to the owner. Owner’s draws are not an expense of the business and do not create a taxable event for either the business or the owner, because the business and the owner are the same legal entity.

The primary disadvantage of the sole proprietorship is exposure to legal liability. Because you and the business are one and the same, debts incurred by the business are debts owed by you personally, and any legal liability, such as for errors and omissions, to which the business may become exposed in the course of operations exposes you and your personal assets as well.

Partnership

A partnership by definition involves two or more people who jointly own and in most cases jointly operate the business. General partners own a portion of the business, usually participate actively in its operation, and are responsible for its debts and liabilities. Limited partners also have an ownership position in the business, but do not actively participate in its operation and have limited liability. The profits and losses of a partnership flow through to the partners in whatever proportions are specified in the partnership agreement.

While many professionals such as attorneys commonly form partnerships, it is not a common form of business organization for independent software developers.

Limited liability company (LLC)

The LLC is by far the newest of these four forms of business organization. Because it offers many of the protections of a corporation without some of the restrictions and complexities, it has become an increasingly popular form of organization for businesses both small and large.

The owners of an LLC are called members. An LLC can have as few as one member, making it a suitable form of organization for a business owned by a single individual. Like sole proprietorships and partnerships, profits and losses from an LLC flow through to the members. Unlike a sole proprietorship or partnership, however, an LLC provides the members with legal protection against debts and liabilities incurred by the business.

An LLC is generally easier and less expensive to set up than a corporation, and typically requires less formal paperwork to operate. However, an LLC cannot use the suffix “Inc.” in its name. Instead, it must use the suffix LLC, as in *Pretty Good Software, LLC*. If it’s important to you have "Inc." in your business name—and that's a perfectly legitimate desire, even for a one-person business—then you'll need to form a corporation.

Corporation

A corporation is a legal entity created by the state. As a legal entity, it is separate from its owner or owners. Before LLCs, many small businesses, particularly those with only one or two owners, incorporated as an subchapter S corporation (or S-corp, for short). A subchapter S corporation is a special form of corporate entity under which the corporation's profits flow through to the owners. Larger businesses and those with multiple owners usually incorporate as a subchapter C corporation (C-corp), in which the corporation's tax return is entirely separate from that of any of its owners.

Forming a corporation in not particularly difficult, but running a corporation—even a very small one—requires additional record keeping and a greater degree of formality than running a sole proprietorship or LLC. For this reason, an LLC is often the preferred choice for many small businesses. However, if you really want or need to form a corporation, don't be put off by the expense or the paperwork. Just find out all you can about it first, then talk to your accountant and your attorney to make sure you know what to expect under this form of business organization.

Legal and financial protection

Regardless of the form of business organization you choose, when you are an independent software developer you, personally, *are* your business in the eyes of your clients. While an LLC or corporation offers a degree of legal protection from liability for things the business does or fails to do, it is unwise to rely on that protection alone. If there's a serious problem with software you've developed for a client, and your client feels they've suffered a financial loss because of it, they may very well sue you personally as well as suing the business, irrespective of the type of business organization you've chosen.

To protect yourself, you should first and foremost have a written agreement with every client. For software development work, the agreement should spell out the work you propose to do, how much you are going to charge for it, when you will deliver it, when you will be paid, and exactly what the software is designed to do (the specifications). In addition, the agreement should explicitly limit or if possible even eliminate your liability for any direct or consequential damages the client may later allege to have been caused by errors or omissions in your software or by delays in its delivery.

There are a number of print and online references to help you construct a good software development agreement and/or license agreement. One of my personal favorites over the years has been *Web and Software Development: A Legal Guide* by Stephen Fishman, now in its 4th edition. This book contains a wealth of discussion and explanation on issues with which every independent software developer should become familiar, as well as providing several boilerplate agreements you can adapt to your own needs.

Print and online resources, regardless of how good they may be, are not a substitute for professional legal advice. If you're in business, you need an attorney. Find a good one and use his or her services whenever you feel you need them. When it comes to drafting an agreement, it can often be effective to start with a boilerplate agreement you've put together yourself and then have your attorney review and revise it to be sure it accomplishes what you want.

Even if you have legally and contractually limited your liability, you can still be sued by an unhappy client for alleged damages due to errors or omissions in your software. Insurance against this kind of exposure is available, although it’s sometimes not readily available—not all insurance agencies offer it—and it’s almost always expensive, with the premiums being based on your business history as well as the desired coverage limits. Only you yourself can decide whether or not to acquire this type of insurance for your business, but it’s well worth your time to find out what's available and how much it would cost you.

You will almost certainly want to insure your business against is the physical loss of your equipment and software. It's certainly not inconceivable that your office, your computers, your software tools, and quite possibly your work in progress could be damaged or destroyed by a fire, flood, storm, earthquake, or other disaster. You can protect the physical assets with insurance, and protect your work in progress with regular off-site backups.

Even if insurance does help replace your business's physical assets in the event of loss, you may still be unable to operate your business at full capacity for some extended period of time after the loss. To protect against that risk, you may also want to acquire business interruption insurance to help replace your income stream until you are back to 100%.

Where am I now that I need me

One important decision you'll need to make is whether to operate your business from home or to lease office space in a commercial building. There are of course advantages and disadvantages to each.

Home office

One obvious advantage of a operating as a home-based business is that you don't have to pay rent for your office space. Not only that, but if certain requirements are met the IRS may allow you to take a tax deduction for the space you use for your home office. Check with your accountant or tax attorney to be sure.

Many independent developers operate quite effectively from their home, but it takes a certain amount of self-discipline and of course your home must provide you a suitable space to work in. At a minimum, your home work space should be clean, well lit, and quiet. An area such as a den, a spare bedroom, a loft, or a portion of the basement may be quite suitable for this purpose.

Most importantly, and regardless of what part of the house your home work space physically occupies, you need to be able to separate your work space from the rest of the house. This is particularly important if there are other family members or roommates who are frequently at home when you’re working. Ideally, your home office should have a door you can close and maybe even lock, but even without a door you need some way to keep your office space out the flow and away from the noise of normal household traffic.

Software development work often requires periods of sustained concentration. If you're operating out of your home, it’s important for others in the household to understand that when you’re at work, you’re at work and not available to attend to ordinary household interruptions, even if you’re only ten feet away.

These days, high speed Internet access is a must for any software development business. Fortunately, acquiring high speed access at home is not a problem in most areas of the country, so this is usually a non-issue. However, in addition to high-speed access you may need to acquire a static Internet IP address, particularly if you’re running a server or frequently need to access your home office network via a remote desktop connection from the field. If your ISP can’t or won’t provide a static Internet IP address for your home office, there are solutions such as No-IP.com that can help.

It’s also advisable to have a separate phone number for the home-based business, not only so incoming business calls don’t ring busy if someone else is using the home phone, but also to help support your image as a professional by letting clients know your business is not the same as your home even if they’re co-located. You don't necessarily need a land line; a cell phone or voice over IP (VOIP) service can be an ideal solution here.

Commercial office space

As I see it, there are two primary advantages to leasing commercial space for your software development business. The first is that you automatically solve the problem of keeping your work space separate from your home. The other advantage is that having office space in a commercial building establishes a certain legitimacy for your business in the eyes of prospective clients.

Whether or not the second point is of any importance depends on the nature of your business and how you interact with your clients. If you're a Micro-ISV who sells software over the Internet, or a contract programmer who works for and interacts with others only by telephone and e-mail, it may make no difference at all where your office is located because nobody but you will ever see it. But if your business involves face-to-face meetings with clients at your office, for example for sales presentations or project review meetings, being home-based can be a real disadvantage. "Never mind the diapers, let's just go down to the basement and discuss your $1 million dollar software project" doesn't position you very well to close the deal.

Signing a lease for commercial office space and committing to the associated expense over an extended period of time can be a very sobering experience, particularly when you're first starting out. But if you can find a suitable space with financial obligations that aren't overwhelming, I think you'll find the benefits of locating your business in a commercial building to be worth the risk.

The reason I titled this section "Where am I now that I need me" is that regardless of where you choose to locate your office, there will very likely be times when you'll feel like you need to be both at home and in the office at the same time, especially if you have a family. This need can be psychological as well as physical, so having a home-based office doesn't automatically solve this problem. Finding office space within easy commuting distance of home may be an ideal solution.

Gearing up

It's a truism that "You have to spend money to make money." It's also true that you can spend money and *not* make money! The moral here is, invest wisely in the resources you need to start and run your software development business.

The barriers to entry into the independent software business are quite low. Indeed, this is one reason why it's so attractive to many people. Given that you have the requisite technical skills and are reasonably able to sell your services to people, about all you need to launch a software development business is a computer, some software, and a dry place to sit.

Okay, that's being a bit flippant, but the point is that it's easy to get carried away and buy stuff you don't need. Sometimes it's difficult to distinguish between what you do need and what you don't. Learning how to tell the difference is an important skill for any business owner.

I've had the opportunity to sit on a few boards of directors over the years, and in the process I've worked with and learned from some very smart people. One person in particular whom I always remember is a highly successful businesswoman in her own right, a person whose advice was frequently sought by others. She had a no-nonsense approach to business that was both refreshing and useful.

We served on the same board of directors for a few years. The board frequently wrestled with budget issues of a "should we or shouldn't we spend money on something" nature. There were often strong opinions on both sides, and the discussion among the board members could become quite intense and sometimes even emotionally heated. At these times, this person could often defuse the situation by asking a simple question: Is the expenditure in question a "gotta have" or a "wanna have"?

This simple technique usually brought the discussion back down to earth and helped the board avoid spending money on things that may have looked necessary at first—things that group-think or peer pressure may have been made to look like "gotta haves"—but which when considered more carefully and without emotion turned out to be only "wanna haves".

This technique is kind of an Occam's razor for making decisions about business expenditures. Learning to apply the technique effectively will help you buy the $2000 computer you need instead of the $3000 one you want, the $200 chair that's quite suitable instead of the $1200 one you saw in your Fortune 500 client's office, and the few software tools you really need instead of all the cool ones you merely want.

Let me admit to a weakness of my own here: I'm a sucker for a good computer book. Frankly, my office looks like the computer section of Barnes & Noble exploded. I'll go in for a mocha latté and come out with a book. In my defense, I'll claim I have in fact gotten something useful out of each and every book I've purchased. But did I really need all of them? Probably not. Fifty bucks here and there does add up over time. By now, I could've had a better chair instead.

Naturally, in real life you'll need to invest in more than just a computer and some software. Gearing up to start and run a software development business also involves planning for ongoing expenditures for such things as office space, heating and air conditioning, electrical power, telephone and Internet connectivity, education and training, marketing and advertising, and your own self improvement both personally and professionally. Learn to spend money wisely and you'll reap the rewards of a profitable business.

How will I attract customers

I'm willing to bet most software developers would rather debug a hexadecimal stack dump than engage in the "S" word. Yes, I'm talking about selling. In many ways, the skill set needed to be an effective sales person are diametrically opposed to those needed to be an effective software developer. But you're going to have to face the fact that in order to succeed as an independent developer, you need to know how to sell what you create.

I'm not talking about cold calling here. Personally, nothing annoys me more in the course of a business day than being interrupted by an unsolicited phone call from someone who promises they're not selling anything and then tries to sell me credit card services, health insurance, off-shore "business partnering", or whatever other crap du jour they're being paid to push. I detest that sales approach so much I made a promise to myself never to do it in the course of my own business. I've kept that promise for 14 years and I'm still in business, so the good news is you don't have to go door to door with a shrink-wrapped copy of *My Pretty Good Software 4U* or call people you don't know on the phone in order to make a living.

However, you *do* need to be able to convince people to buy what you're selling. How do you do that? How do you attract customers?

There are both formal and informal ways to approach this. Formal approaches include conventional advertising programs such as the yellow pages, radio, television (don't overlook local cable TV), newspapers, and of course a business website. When you're first starting out you're trying to build awareness that your business exists, and therefore you may want to try a scattershot approach to find out what works best for you. Later on, a more targeted approach probably makes more sense.

Personally, although I've always had an ad in the yellow pages, I've only closed one piece of business I know resulted from that ad. All the rest has come from personal contacts, word of mouth, or Internet contacts via my website or blog. That's just me, though. What works for me may not work for you, and vice versa.

Informal approaches to attracting customers include business development efforts you can make via personal contact with people at almost any kind of business function. One good way to get to know other business people in your community is to join the Chamber of Commerce and/or one of the service organizations such as Rotary, Kiwanis, or American Business Clubs (AMBUCS). Membership in service organizations has been in decline for decades and doesn't provide as much benefit as it once did, but these organizations still offer a good way to meet other business and professional people in the community. In other words, find ways to get involved, to meet people, and to let people meet you. You never know when a casual contact may lead to a business prospect.

Of all methods of attracting customers, word of mouth is probably the most powerful. A positive word from a trusted acquaintance to someone you don't even know may be all it takes for that person to pick up the phone and call your business with a favorable impression of you before you even answer the call.

Family and friends can also be good sources of information and referrals for your fledgling business. This doesn't mean trying to sell insurance to everybody at your family Christmas gathering. It simply means letting everybody know what you do and that you're open for business.

The value of certification

What's the value of certification for the independent software developer? I think that's a good question. I'm talking about professional certification here, which for developers using Visual FoxPro and other Microsoft tools means MCP, MSCD, MCPD, MCSE, MCDBA, etc.[3]

Taking a page from my own experiences, I operated my own software development business for years without certification. I think I was a pretty competent developer, and believe my clients would agree. By not having certification, I never felt I lacked anything either I nor my clients thought I needed. However, when the certification exams for Visual FoxPro came along I jumped at the chance. I acquired both the MCP and shortly thereafter the MCSD certifications.

Did these certifications automatically make me a better developer, or add value to the services I provide to my clients? Not directly, but yes, I believe they did. For one thing, studying for and passing the certification exams teaches you what Microsoft thinks is important. Knowing this helps you better understand the tools you're using and the environment you're deploying them in. Also, studying for and passing the certification exams exposes you to aspects of the software development profession you might not need to know about for everyday practice, but which help you to see the big picture more clearly.

One tangible benefit of certification is the ability to display your accomplishment on your website, business cards, and other marketing and advertising materials. Whether or not this has led directly to any additional business for me, I can't say. To my knowledge it has not, but I do believe it adds to my up-front credibility when first meeting with or talking to prospective clients. In that way, having certification does open doors and increases my marketability as a professional.

Certainly, certification does not automatically make one a better developer, and by no means are all certified developers better than their uncertified colleagues. Nonetheless, I think certification does add real value both for the developer and for the client.

How will I retain customers

When you're in business, a loyal customer is gold. If you've succeeded in earning a customer's loyalty, strive to do everything you can to retain their business. In my experience as an independent developer who primarily creates custom software for individual clients, a great deal of your ongoing revenue stream is going to come from repeat business with existing customers.

[3] See http://www.microsoft.com/learning/mcp/certifications.mspx for a current list of available Microsoft certifications.

Much of what you can do to retain a customer's business is the same as whatever you did to win their loyalty and appreciation in the first place: treat them right, create great software, deliver it on time, follow up with support—whatever it takes. The next section in this paper discusses way to meet and exceed customer expectations. The ideas in that section should go a long way toward helping you win and retain customers.

Meet and exceed expectations

The best way I know of to win and retain clients is not only to meet but also to exceed their expectations. Meeting expectations is fairly easy. It's also kind of dry and unexciting. Exceeding expectations is more challenging but at the same time more fun and more rewarding, both for you and for your clients. Do to either, however, you must first learn to *manage* expectations.

Managing expectations is built on a foundation of open and honest communication between you and your client. While it's okay and even necessary at times to move outside your comfort zone when agreeing to a client's requests, promising more than you know you can deliver is not being honest. On the other side, clients need to recognize you need clear objectives and reasonable time frames to operate in. The foundation of managing expectations, then, is for both sides to be realistic and reasonable with one another.

In my experience, clients sometimes have little if any idea how difficult a particular request might be to implement, or how long it might take to deliver. On the other hand, they usually do know pretty clearly what they want. Your job as the developer is to find a way to give them what they want without overextending yourself. In other words, you need to manage your client's expectations.

Promising to "have it ready by next week" when you know in your heart of hearts there's almost no way you can do so is a disservice both to you and to your client. It's much better to say "It looks like this will take three weeks" and then deliver in two, than to say "I'll do my best to have it ready by next week" and then deliver it in two weeks anyway. In both cases the client gets what they wanted in two weeks, but in the first case it's a week early while in the second case it's a week late. In the first case, you've managed expectations so you could meet and even exceed them, while in the second case you've let expectations get out of control resulting in disappointment and frustration on the part of your client.

I don't mean to suggest you should be disingenuous about this sort of thing. Remember the part about open and honest communications. Saying you can deliver something next week when you know it'll take two or three is not being honest with your client or with yourself. Saying it'll take three weeks because you know it'll take at least two and you're also allowing for unforeseen delays is being both honest and reasonable. In the long run, clients will appreciate that.

It's been said that if you show up for a job interview on time and with your shoes shined, you're already ahead of 90% of the other candidates. This may sound trivial, but there's a good deal of truth in it. In my view, one of the most important things you can do to meet and exceed expectations is to strive to maintain your professional image at all times.

Striving to maintain professionalism applies to what you do, what you say, how you act, and how you dress for business. The old saying "You only get one chance to make a first impression" is trite but also true. Keep it in mind whenever you answer the telephone at your office, whenever you meet someone in the elevator (because they could turn out to be your next appointment), and whenever you're writing an e-mail or a blog entry (because a current or prospective client might be reading it). A careless word, a sloppy appearance, or poor follow-up on a commitment can damage your professional image, and it takes a good deal more effort to recover from this than to avoid it in the first place.

If you have a long term relationship with a client, there are other ways you can exceed expectations. Don't hesitate to go the extra mile for a client when doing so would really be helpful to them. In fact, look for opportunities to do this.[4] Give an discount now and then, or even do something for free once in a while. Everybody likes to be treated as if they're special. Giving a client an hour of free support when they're in a bind, or delivering that little "Gee, it would be nice if we had this" project for free now and then goes a long way toward letting your clients know how much you value their business. The relatively small investment you make in exceeding their expectations will be richly rewarded over the course of your continuing relationship.

Finally, make a real effort to learn and understand your client's business. One of the things I find so interesting about being an independent developer is the wide variety of business people I get to work with. Because I specialize in developing and supporting what I call mission-critical software applications—applications that support my clients' core business processes—I can do a much better job and create a much more effective software solution if I understand *their* business as well as I understand my own. Take time to understand the nature of your clients' business, the challenges and competitive pressures they face, and the problems they deal with on a daily basis. Only then will you have a real perspective on how your software fits into their business model, and from that perspective you can see new ways your software can help their business succeed.

Who owns the software

The question of who owns the software you create is a legal question, the answer to which depends on many factors. Again, I must issue the standard disclaimer: I am not an attorney, and what follows should not be construed as nor used as a substitute for professional legal advice. For much of the material in this section I am indebted to the book *Web and Software Development: A Legal Guide* by Stephen Fishman, which I cited earlier.

Copyright protection

In general, software is covered by the laws governing intellectual property. Of the several bodies of law that protect intellectual property, copyright law is probably the most important for the independent developer. This is because software is considered a work of authorship, which copyright law is designed to protect.

When you create an original work of authorship—for example, when you create an original software application—you automatically own the copyright. In the United States, you do not need to publish a copyright notice in order to own it, although it is customary to do so. The standard

[4] Eric Sink, Source Gear's founder and a frequent blogger who also lives here in Champaign, Illinois, recently blogged about just such an experience. Read about the effect it had on Eric's opinion of this business in *Absurd Customer Service* at http://software.ericsink.com/entries/Absurd_Customer_Service.html.

form of a copyright notice is the word "Copyright" and/or the copyright symbol © (Alt+0169 on your keyboard), followed by the year of publication, followed by the name of the copyright owner. For example, a copyright notice might appear as Copyright © 2007 Pretty Good Software, Inc. Some foreign countries may *require* a copyright notice in order to establish copyright protection. If you market your software outside the United States, check with your attorney to find out what's required to protect your software in those countries.

Copyright protection for software you create gives you the exclusive right to copy and distribute that software, as well as to modify it and to create derivative works from it. If someone else does so without your permission, you can sue them for copyright infringement. Your legal standing in a copyright infringement suit will be stronger if the software contains a copyright notice. This is one reason the standard Visual FoxPro program header (generated by IntelliSense when you type the word 'header' followed by a space) includes a copyright notice. Mine looks like this:

```
*==============================================================================
* Program:        FOO.PRG
* Author:         Rick Borup
* Date Written:   08/28/2007
* Copyright:      (c) 2007 Information Technology Associates
*                       All rights reserved.
* Compiler:       Visual FoxPro 09.00.0000.3504 for Windows
* Abstract:
* Environment in:
* Environment out:
* Parameters:
* Returns:
* Changes:
*==============================================================================
```

You can use the IntelliSense Manager to edit the script for 'header' to make your program header appear the way you want it. In addition to placing a copyright notice on each individual program within an application, you should also include the notice in the application's license agreement and on its package and distribution media, if applicable.

Source code

Beyond the question of who owns the right to copy and distribute the software, an issue of paramount importance to independent developers is who owns the source code. This question is not as relevant if you are a Micro-ISV selling the same application to the hundreds or thousands of customers, but it if you develop custom software for individual clients this can become a very contentious issue.

In the United States, the general answer to this question is that the creator of the work owns the rights to the source code just as they own the copyright. This means that unless you are working as someone's employee (which the independent developer typically does not do), or you explicitly give your client ownership rights to the source code via written agreement, you as the developer own the source code.

Before beginning any work for a client, I strongly advise you to put a written agreement in place that includes a provision making it clear who owns the copyright and who owns the source code. To some clients it may not matter whether or not they own the source code, as long as they have exclusive rights to use the software you create for them. In other cases, the client may have legitimate business reasons why they need to own the source code, such as in cases where the code you write for them is going to be integrated into an existing application for which they own the copyright. In still other cases, the client may simply insist they must own the source code for their own protection. As an independent developer you'll need to deal with each situation individually, but remember that unless you are working as an employee or contractually agree to hand over your rights, you own the source code you create.

If you do agree to let your client own the source code for the software you create for them, you need to be careful to exclude source code you yourself do not own. It's frequently the case that a software application is constructed from original code you write yourself in combination with methods, routines, or procedures you've licensed from a third party. While you may receive the source code for these 3rd party methods, routines, or procedures in order to better enable you to work with them, your license to use them typically does not allow you to redistribute that source code, much less to convey ownership rights in it to someone else. Therefore be careful when drafting your software development agreement to explicitly identify the pieces of the application that are licensed to you from a 3rd party, and make it clear in the agreement that you cannot give your client the source code for such 3rd party software, nor any rights to it other than the right to use it as part of your application.

Estimating projects

Does your first meeting with a prospective client often go something like this? "I need a [insert vague description of software application here]. How much will it cost?" In this situation you may be hard pressed to keep a straight face, much less to give a straight answer. The answer, of course, is "it depends." It depends on things that haven't been specified yet.

You can't answer this question in the first meeting, and you shouldn't even try. If you do give a number, you create an expectation that's most likely entirely out of whack one way or the other. Too high and you discourage the client from even talking to you any more. Too low and you'll be expected to deliver "it" (whatever "it" turns out to really be) at that price. Either way, you lose.

The three meeting rule

Avoid getting pinned down to quoting a price or even a price range in the first meeting. Instead, realize that it's probably going take two or three meetings for you and the prospective client to come to a common understanding of what the client wants and what the scope of the project is going to be.

I've operated on the "three meeting rule" for a long time and have found it to be surprisingly valid in a wide range of situations. In the first meeting, you and your prospective client get to know one another a little bit and you form a general sense of what the client wants. In the second meeting you begin to draw out some of the details, which the client often hasn't even considered yet. By the end of the third meeting, your client has a more refined and complete picture of what they need, and you have a fairly good sense of what the general scope of the project is going to be. Only then can you even begin to talk about estimates.

Software estimation techniques

I don't think it's much of an exaggeration to say software estimation is a black art. Volumes have been written on the subject, much of it dense and indecipherable. For software development projects involving dozens or hundreds of developers, formal software estimation methods and project management techniques are essential to success. On the other hand, an independent developer usually does the work solo or with a small group, and the scope of the project is generally smaller and therefore more manageable. This makes it possible for the independent developer to use less formal software estimation methods and project management techniques than are required for larger team projects.

The reason for doing software estimation at all is because clients want to know what to expect and developers want to know what's expected. To put it another way, clients want an answer to the standard question "How long is it going to take and how much is it going to cost?", while developers want to protect themselves against committing to deliver on vague ideas for a fixed price or within a fixed time frame. Any valid software estimation, therefore, must be based on a complete and comprehensive set of specifications for the software to be developed.

Given a good set of clear specifications for the project, how does the independent developer arrive at an estimate of time and cost? Do you invest the time in takes to apply formal and often intricate methods such as function point analysis? Do you do some preliminary design work and then take a count of the number of database tables, stored procedures, remote views, forms, reports, and custom methods that will be required? Or do you simply "get a feeling" for the project and come up with an estimate based on your experience with other projects of a similar nature? All of these are probably valid at one time or another.

You can, and should, realistically expect to be paid for developing the specifications for a software development project. But it's difficult to convince a client to pay you for the time it takes to develop an estimate from those specifications. Therefore you are probably going to want to minimize the amount of time you spend developing estimates, particularly when there's significant uncertainty an estimate will be approved. If creating estimates translates into un-billable hours, you're going to want to do create estimates as quickly as possible.

How formally or how informally you decide to approach the software estimation process for a given project depends on your comfort level with the client and the project, how similar it may be to other projects you've completed in the past, how closely the client will hold you to the estimate, whether is really just an estimate or is in fact a fixed cost bid, and how much uncertainty you and your client are willing to tolerate. In my experience there's little room for uncertainty in a relationship with a new client: You do not want to go over an estimate on that first project or you may damage your credibility for future projects. On the other hand, once you've established creditability with a client over time, and have brought most projects to completion at or near their estimates, clients may be more forgiving of an occasional estimate overrun. This gives you the freedom to be a little less formal in estimating projects for long term customers, assuming they're also willing to accept that informality.

In his book *The Software Developer's Guide*[5], Whil Hentzen describes (in addition to six humorous estimation techniques, presented in Whil's inimitable style) an alternative to function point analysis that he calls action point counting. If you're looking for a somewhat formal method for software estimation that's suitable for independent developers, I suggest you read about action point counting in chapter 17 of Whil's book. If you do, don't stop with chapter 17; the entire book is a worthwhile read.

Regardless of what software estimation technique you use, it's my belief that most developers (including myself) seem to have a tendency to overestimate their ability while underestimating the amount of work involved in a project. Part of it is just ego: we like to think we're pretty good at what we do. Another part of it is that it's easy to overlook the intangibles and the extra or unplanned work that inevitably seems to creep into the finished product. Knowing these things to be true, it would be nice if there were a technique to help avoid making estimates that turn out to be consistently too low. In my own work, I've come up the 80-80 rule.

The 80-80 rule

There is a general principle known as the 80-20 rule, which says that 80% of the effect often come from 20% of the causes. This rule is amazing applicable to a wide range of situations: 80% of the profit comes from 20% of the customers, 80% of the resources are consumed by 20% of the activities, and so on.

Paraphrasing this rule, albeit somewhat humorously, it's been said that the first 80% of a project takes 80% of the time, and the last 20% of the project takes the other 80% (sic) of the time. This give rise to what I call the 80-80 rule. Setting the humor aside, the 80-80 rule provides a way to quantify the tendency to underestimate how much time it will take to complete a project.

For the sake of example, consider a project with an initial estimate of 200 hours to complete. The estimate has been done with due diligence and is based on some formal or semi-formal estimation technique rather than simply being an informed guess. But recognizing there is a built-in tendency, based on past experience, to be overly optimistic and to underestimate projects, the 80-80 rule can be applied to help arrive at a more accurate estimate.

The arithmetic of the 80-80 rule is simple. If 80% of the project takes 80% of the time, and the other 20% of the project takes the other 80% of the time, then the total time for the project (T_2) is 80% of the original estimated time (T_1) plus another 80% of the original estimate, or 1.6 times the original.

$$T_2 = .8(T_1) + .8(T_1) = 1.6(T_1)$$

In the case of the example with a 200-hour original estimate, the revised estimate is therefore

$$T_2 = .8(200) + .8(200) = 1.6(200) = 320 \text{ hours}$$

If this seems flippant or simply a way of padding an estimate, it's not meant to be. It's a technique you can use in good faith to help compensate for a tendency to underestimate projects. If the revised estimate generated by the 80-80 rule seems to high to you, use 70-70 or 60-60.

[5] The Software Developer's Guide, Third Edition by Whil Hentzen, Hentzenwerke Publishing, 2002

The point is that applying this rule gives you a way to build in to your estimates a quantifiable amount that compensates for a tendency to be overly optimistic. Remember that a bad estimate, regardless of how rigorously it was derived, is a disservice to both you and your client. A good estimate, even if it seems high at first, is win-win for both parties.

Working effectively - the three E's

Here I'd like to introduce the three E's of personal endeavor: effort, efficiency, and effectiveness. The three E's are a composite of things I've read and things I've learned from my own experience. To the best of my knowledge they're not take directly from any single source, so I'll claim them as my own with apologies in advance if I'm using somebody else's ideas without realizing it.

Effort

Effort is simply how much time you devote to your business. Hours per week is a good measure of effort.

Effort = hours devoted to the business

If you think you're going to succeed as an independent developer by working a conventional 40-hour work week, think again. While the amount of time you'll spend working will probably vary from week to week—some weeks will be intense while others may be slack—you can count on averaging a lot more than forty hours a week. A peak of eighty to one hundred hours is not out of the question when you're really busy, but on average fifty or sixty is probably a good guesstimate.

You're an independent, though, so it's up to you how much effort to put forth.

Efficiency

Efficiency is the ratio of income-producing activity to total activity. If you bill by the hour, efficiency can be measured as billable hours divided by total hours. If you don't bill by hour you can devise other metrics here, but the concept remains the same.

Efficiency = income-producing activity / total activity

For example, if you average 60 hours a week at work and 45 of those are billable hours, you're working at 45 / 60 = 75% efficiency.

You might be tempted to ask, "Why can't I work at 100% efficiency? Why can't I spend 100% of my time on income producing activities?". The answer of course is that as an independent developer you're not just developing software, you're also running a business. Running a business requires doing many non-income producing activities, or at least activities which, although necessary to grow and sustain the business, don't themselves directly generate income. This includes the time you'll need to spend on administrative overhead—bookkeeping, paying bills, sending out invoices, doing your taxes or working with your accountant, and so on—as well as the time you invest in business development, marketing, training and education, self-improvement, and other such activities.

Obviously, the more efficient you can be, the more income you'll generate for any given effort. But if you're a one-person business, don't expect to work at 100% or even 90% efficiency. That's just not reasonable. When I first started ITA, a friend who was already in business for himself (although not a software developer) advised me to expect to spend about 30% of my time on business development and other non-income producing activities. It sounded like a lot, but it turns out to be about right.

Don't expect to be able to increase your effort (total hours) to an unreasonable level and sustain it over the long haul, either. You can probably handle those eighty or a hundred hour weeks now and then, but if you try to do that week in and week out you'll very likely burn out physically, or psychologically, or both.

Effectiveness

Effectiveness is more difficult to quantify than effort or efficiency. By effectiveness, I mean the extent to which you do those things which serve to produce the desired effect over time, whatever that desired effect may be.

As an independent software developer, the desired effects might include growing your professional reputation and expanding your business so you can handle more clients and perhaps even hire some additional developers. But even if your objective is to stay solo, you can be more successful in the long run if you learn how to choose and focus on projects that have the greatest potential for long term profit, while avoiding those that tend to be risky or problematic over time. Making the right choices contributes to being effective.

It is possible to be efficient without being effective. The things you must do to be effective are consistent with but not the same as the things you do to be efficient. An independent developer who spends 90% of his or her time designing and developing software is being efficient, but might be more effective if he or she worked at only 85% efficiency while investing the other 5% of the time to grow professionally and cultivate contacts that might result in future business.

Another aspect of being effective is being able to differentiate between simply being busy versus doing the "right" things at the right time. In the best-selling book "The 7 Habits of Highly Effective People" by Stephen R. Covey, the third habit is "Put first things first." Dr. Covey gives an example to help illustrate this concept, which I'll try to paraphrase here. Consider a large glass bottle—say, a one gallon milk jug—with a wide mouth at the top. You are handed the bottle, which is half filled with sand and small gravel, along with several larger rocks. You are asked to place the rocks in the bottle along with the sand and gravel. Although each individual rock is small enough to fit through the mouth of the bottle, there is not enough room in the bottle for all the rocks because it's already half full of sand and gravel. The solution, of course, is to empty out the sand and gravel, put the rocks in the bottle first and then pour in the sand and gravel, which will fill in the spaces between the rocks.

This is the principle of "put the big rocks in first." In Dr. Covey's book, which by the way I highly recommend, the lesson is given in the context of achieving balance in your life so you have time for and can accomplish the things that are important to you without letting the little things get in the way. To cast this example in the context of the independent software developer, I suggest the principle of "putting the big rocks in first" means concentrating on those activities which have the most importance to the project at hand, or to the growth of your business in general, while learning to leave the smaller or less important activities for later.

I wrestle with this in my own practice all the time. My to do list usually contains several smaller items, each of which I could do in an hour or two, along with at least one or two larger and more time-consuming items such as delving into the design of a new system or completing a major piece of an ongoing development project. Perhaps your to do list is similar. It may be tempting to want to knock off the most number of items possible from the to do list by doing the smaller things first. To some extent this is okay, and certainly some of the smaller things are genuinely important and may even have material deadlines. But if over time you don't put the big rocks in first, you'll end up piddling around with the small stuff all the time while the larger, more important projects languish. This is not being effective.

Things that can hurt you

In the section on SWOT analysis earlier in this session, I discussed how a business is affected by both internal and external factors. In this section, I want to expand on the discussion of the external factors that can adversely affect you as an independent software developer—in other words, things that can hurt you.

Shifts in business climate

In general, all business are affected by the ups and downs in the economy. When overall economic conditions are good, people are optimistic and willing to invest in the future. For your clients, that future may include investing in new software that you are in a position to provide.

It's important to pay attention to the economic climate of your customer's particular business segment, too. Even if overall economic conditions are good, your client may be in an industry that's suffering a temporary downturn or even a long term decline. It's wise to pay attention to this, because if business turns bad for one of your good clients, your income from that client is likely to decline even if you're still doing the best possible job for them.

Shifts in technology

Given the rate of change in computer and software technology, this is an obvious risk factor for any software development business. As Visual FoxPro developers, we've benefited from a long run (roughly fifteen years) using a single, wonderful tool. But to some extent we've also become dependent on that single tool. Although officially supported until the year 2015, Microsoft's announcement that there will be no further development of core VFP portends a significant technology shift for Visual FoxPro developers over the course of the next eight or ten years.

The type of shift is not limited to Visual FoxPro, of course. One of the other trends worth watching these days is the increasing move toward software as a service, or SaaS. In my opinion it's still undetermined whether or not this model represents a coming sea change, but if SaaS really does take hold the days of the traditional desktop software application may be numbered. Certainly Visual FoxPro can be part of a SaaS solution, but we as developers are going to have to learn new ways of building those solutions.

Change of ownership

If a business client of yours is sold, merges with another company, or otherwise changes ownership, what happens to your relationship with that client? Is the new client automatically entitled to continue using the software you developed for the original client? Is the new client obliged to continue in its relationship with you under the same terms as the original client was? Does your software development agreement contain provisions addressing these things?

The purpose of posing these questions is to get you started thinking about the effect a change of ownership might have on your relationship with a client. In order to protect yourself from any adverse effects or unpleasant surprises, your software development agreement should address what happens to your relationship with a client if a change of ownership occurs.

For example, say you've been developing and supporting software for Mom & Pop's Corner Grocery for the last five years and have a software support agreement in place with them for the next three years. Unexpectedly, Mom and Pop win the lottery and decide to retire and sell the store to MegaFoods, Inc. MegaFoods owns hundreds of stores and has a 500-person IT shop, so it's unlikely they're going to want to continue using your software just for the one store. But is MegaFoods obligated to honor the remaining term of your support agreement? If you've invested fifty hours of development time in a new feature that you haven't yet been paid for and that's now irrelevant, is Mom & Pop's Corner Grocery or MegaFoods still obligated to pay you for it? You can't stop the change of ownership from happening, but contract provisions that address this type of thing should help both parties understand their obligations and avoid unpleasant surprises.

Change of decision maker

Even if ownership of one of your clients doesn't change, it's entirely possible that the decision maker could change. For example, this could happen due to someone's retirement or resignation within the client's business. Or, if it's a family held business, you may have worked with the founders for several years but now reached a point where they are turning over responsibilities to hired managers. In either case, you may find that the person(s) with whom you've always worked are no longer the ones making the decisions about what software to use and who to buy it from.

The challenge to the independent software developer in this case is more personal than the challenge that arises from a change of ownership. You can't contractually specify that the new decision maker will be favorably inclined to continue doing business with you. Instead, you need to quickly learn as much as you can about the new person's background and possible bias, and you need to begin cultivating a good working relationship with them. You can't always win in this situation—the new decision maker may decide the whole business is going Java as soon as your support agreement expires, for example—but you can avoid being blind-sided by being ready to accept the challenge of a change.

It's not always obvious who the decision maker is, by the way. If your client is a very small business, you're very likely to be working directly with the owner or owners, in which case it's clear who makes the decisions and who controls the purse strings. If your client is a larger business, however, your direct contact may be with a manager who appears to make the decisions, but who in fact has to answer to a vice president who actually makes the decisions. In this case, you're as much at risk from a change in vice presidents as you are from a change in managers, even if you've never met the vice president.

There's no easy solution here, just advice to find out as much as possible about who the decision maker really is and to stay alert for changes.

Dependency on a single client

As an independent developer, it's easy to become overly dependent on a single client. After all, who's going to say no to more work from a good client? On the other hand, remaining independent means ensuring you don't become too reliant on income from a single source, because as we've just discussed above, there are several ways that source can dry up unexpectedly.

There's no hard and fast rule here, but from my own experience I offer the following guidelines:

- If 33% of your income comes from a single client, begin to pay attention to how dependent you are on that client. Explore ways you can grow your business in other directions without jeopardizing or unnecessarily limiting your relationship with this client.
- If 67% or more of your income comes from a single client, watch out! You are not really independent any more, and you are at risk of going out of business fairly quickly if that client pulls the plug.

If you find that you're beginning to become overly dependent on a single client, there are only two basic things you can do: grow your business so that the same amount of income from that client becomes a smaller percentage of your total income over time, or keep the overall size of your business constant while reducing the amount of income from the client in question and replacing it with income from new clients. It's the same principle that's used to reduce risk by diversifying an investment portfolio.

Clearly, the first alternative is much more attractive. Needing to find ways to grow your business is a good problem to have. The second alternative risks losing the client altogether because it amounts to telling them you can't do all that they need done. However, there may be ways to finesse this by interleaving income streams from different clients' projects over time, so that client A gets 100% of your attention for some period of time while they need it, while client B gets 100% of your attention for some other period of time.

Getting paid for what you do

You might ask, what's so difficult about getting paid for what you do? You sign an agreement, you do some work, you deliver the software, you send out an invoice, you receive a check in the mail. That's all there is to it, right?

Well, yes and no. If your client is a large corporation, then perhaps that's a pretty accurate picture. Things are fairly impersonal: The corporate wheels grind and you get paid. But when you do work for a small business, say family owned or even a solo owner, the interaction is much more personal. The person writing your check is taking money almost literally out of their own personal wealth to pay you, and part of your job is to make sure they feel they're getting their money's worth. This has implications not only for how much you charge but also for the timing of your invoices.

Each client is likely to be different in this regard. In my experience, some clients are comfortable receiving an invoice on a regular basis, say every two weeks or once a month, because it enables them to forecast their cash flow requirements and avoids long gaps followed by large invoices. Other clients prefer to see some tangible result in the form of delivered software, even if only partial, before they write a check. Naturally, as a small business yourself you should not be expected to finance your client's development costs by waiting several months to get paid. But to the extent possible, you can try to accommodate this type of client by perhaps agreeing to milestone deliveries and milestone payments instead of payments on a fixed calendar basis.

This goes back to understanding your client's business. If their cash flow is cyclical, you may be able to time your work and therefore your invoices to coincide with the times when they're flush and more easily able to pay their bills. In my experience, most clients like to pay their bills on time, but like anyone else they may also feel stressed if they get a large invoice from you on April 16th just after they've forked over a large sum to the IRS.

Naturally, you want to watch out for situations where clients get into real financial difficulty themselves. Warning signs are a change in payment patterns, late payments, and requests for you to get "just one more project" done before getting paid. If you have a long term relationship with a client you'll probably want to work with them to keep the relationship going even during difficult financial times for their business, but take care not to let their problems become your problems.

It's always best to have a clear provision in your software development agreement covering what happens in the event you don't get paid as agreed. There are several ways you can protect yourself here. One is to clearly spell out time timing of invoices and the period of time before they become past due. For example, you might agree to invoice for every forty hours of work or once a month, whichever comes first, with payment being due within ten business days of the invoice date.

It's also a good idea to clearly state what the penalty for non-payment is. For example, you might put into the agreement a provision that if payment is not received on time, you will simply cease work on that client's project(s) until payment is received. This is a short-term solution, based on the expectation that the client is acting on good faith and that you will ultimately receive full payment.

A different approach is needed to handle a long-term problem with non-payment. One way is to make the client's license to use the software contingent on full payment as agreed. The extent to which you could enforce this provision if push actually came to shove in a court of law might depend on several other factors, but having such a provision in the agreement would almost certainly give you a leg up if the relationship with the client disintegrates completely and you are left being owed a substantial sum of money.

What business am I really in

This should be an easy question to answer: you're in the software development business, right? Maybe so, but the answer can sometimes be more than that. As we have seen, there are many ways to be an independent software developer. The nature of the software you develop, as well as the nature of your relationship with your clients, determines the nature of the business you're really in.

Being a Micro-ISV is probably the easiest way to be purely in the software development business. Your create software and offer it for sale to the public over the Internet. Your clients are anonymous, for the most part, and you don't ever have to meet them or go to their home or place of business to make a sale. You are probably also in the software support business to some extent, but even here you can do it impersonally or at arm's length if that's your preference.

If you develop custom software, on the other hand, you have a much closer and more personal working relationship with your clients. You meet them personally and work with them face to face, often on a regular basis. You get to know them, and they you. They will typically come to rely on you for more than just the software you develop for them. You may be invited to consult on other projects that touch on business processes you've automated, or to provide technical support to the staff people who use your software on a daily basis. Sometimes this support extends beyond the domain of your software, into the area of network issues or problems with other software packages, Windows®, or even with your client's business itself. When this happens, you can decide that's not the business you're really in, or you can recognize that perhaps you're really in the service business with software being a primary but not exclusive part of it.

The "business you're really in" can be different for one client than for another, too. Recognizing the business you're really in with each client can help you make better decisions about what to do and what not to do for each one, and from that to grow your relationship with each client to the maximum extent possible.

Basic Financial Management

Just as you're well advised to use an attorney to handle your legal matters, it's also wise to use a professional accountant to handle at least some of your accounting and tax matters. But even if you do use a professional accountant to provide these services for you, you still need to know something about basic accounting and financial management in order to succeed in business. This section looks at two aspects of financial management for the independent software developer: financial accounting and management accounting.

Financial accounting

Basic financial accounting involves doing the necessary record keeping for your business's financial transactions and periodically preparing the three basic financial statements—the balance sheet, the income statement, and the cash flow statement. How formally you need to keep records or how frequently you need to generate financial statements depends in part on the organizational structure of your business. A corporation needs to be more formal about this than a sole proprietorship, but all businesses need to keep suitable records and be able to generate accurate financial reports.

Record keeping

Unfortunately, stuffing credit card receipts and bank statements into a desk drawer does not constitute good record keeping. Even if you're a very small business, it's important to maintain your financial records in a timely and organized manner. While you can do this manually, financial record keeping is much easier if you use small business accounting software.

Even if you hire an accountant to do your some or all of your accounting and tax work, you will probably still find it worthwhile to invest in a small business accounting software package. Products such as the widely used QuickBooks® from Intuit are relatively inexpensive, easy to use, and make record keeping much easier than doing it by hand. In addition, QuickBooks can create and export a file in a format most accountants are able to use, and accountants generally prefer to receive financial data from their clients in electronic format rather than as a pile of canceled check and paper receipts.

Regardless of which approach you take—manual record keeping or software—get in the habit of recording and classifying every expense item and every income item as it occurs. This helps you keep tabs on the current financial condition of your business by making it much easier for you to prepare your financial statements and tax returns on a timely basis.

The balance sheet

The balance sheet is a snapshot of a business's financial condition as of a specific point in time. A balance sheet presents the following information:

- What the business owns (assets)
- Minus what it owes (liabilities)
- Equals what it's worth (capital, or equity)

It's called a balance sheet because the two sides of the equation must always be in balance. The balance sheet equation can be expressed mathematically as

```
assets = liabilities + equity
```

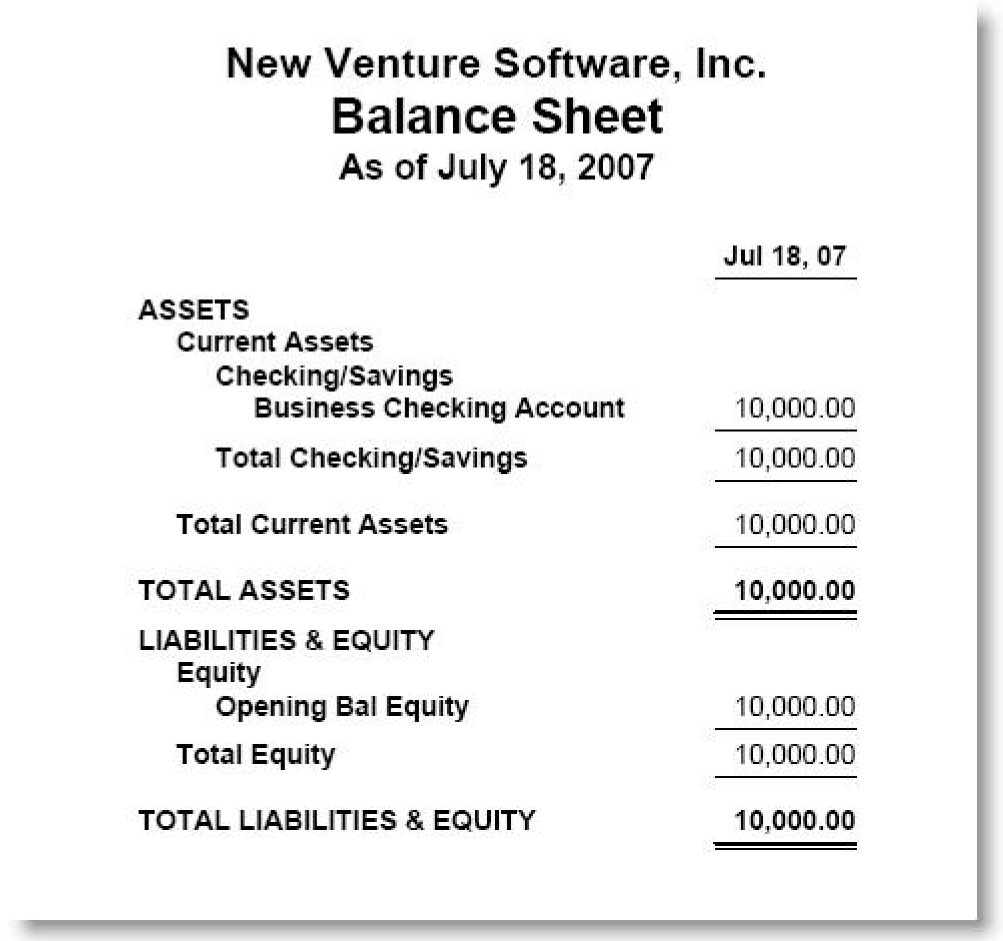
The same equation is often expressed by putting equity first, as in

```
equity = assets - liabilities
```

Assets are the things your business owns, such as computers, furniture, software, and cash on hand. Liabilities are the amounts you owe to others, for example bills you owe to vendors, credit card balances, long-term debt, etc. Equity is the difference between the two, in other words what would be left over if the business liquidated all its assets at cost or basis and paid off all its liabilities. The reason equity is on the liability side of the equation is that it represents the amount the business "owes" to the owners, namely you!

Let's say you decide to set up shop as an independent software developer under the name New Venture Software, Inc. You invest $10,000 of your own money as the initial capital for the business. After legally establishing the business entity,[6] you go to the bank, open a business checking account, and make an initial deposit of $10,000 via a personal check payable to the business. The New Venture Software, Inc. balance sheet now looks like this:

[6] For the sake of the sample financial statements in this paper, it doesn't really matter whether your business is a corporation or not. The basic principles of accounting apply to all types of businesses.



New Venture Software, Inc.
Balance Sheet
As of July 18, 2007

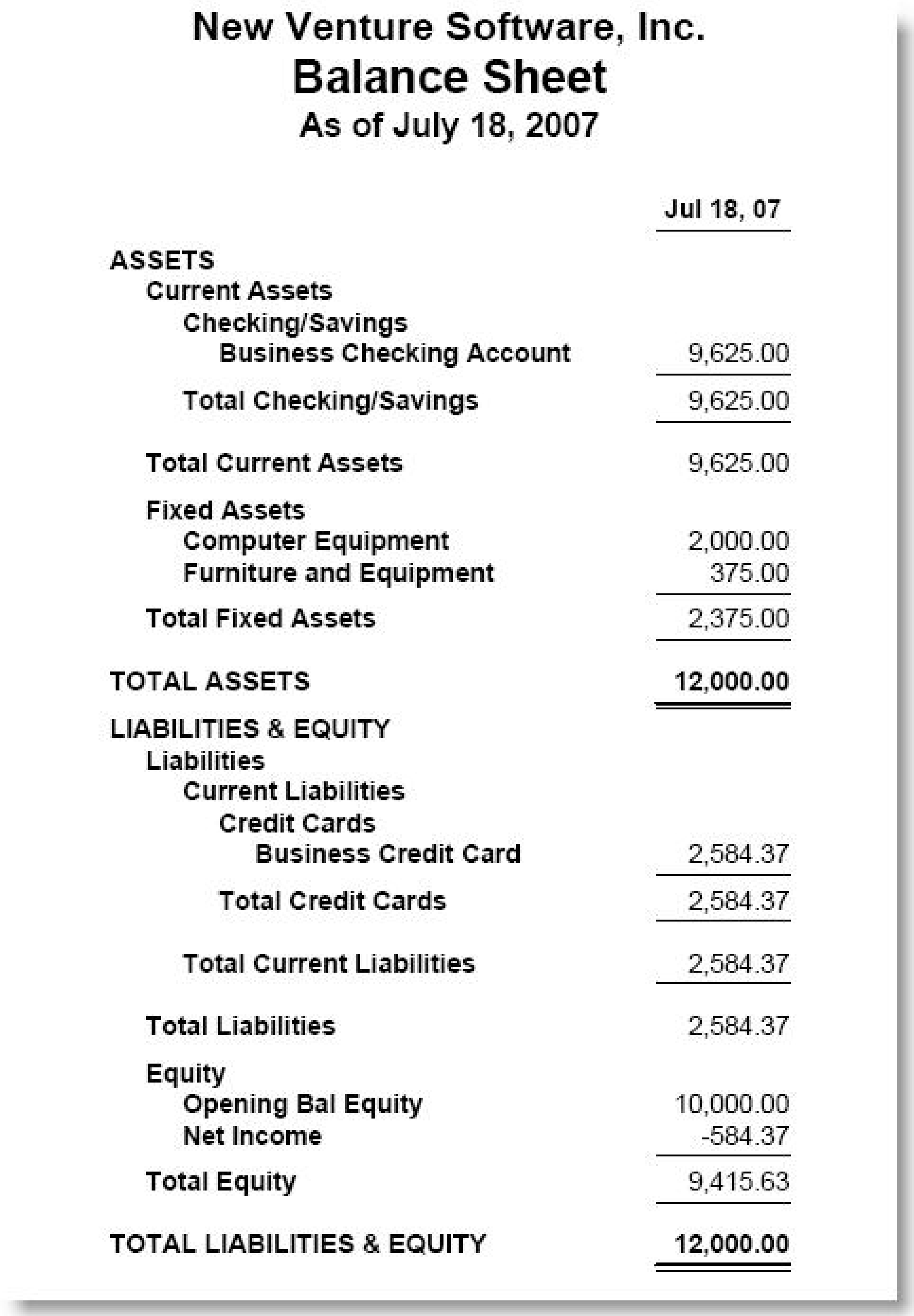
	Jul 18, 07
ASSETS	
Current Assets	
Checking/Savings	
Business Checking Account	10,000.00
Total Checking/Savings	10,000.00
Total Current Assets	10,000.00
TOTAL ASSETS	10,000.00
LIABILITIES & EQUITY	
Equity	
Opening Bal Equity	10,000.00
Total Equity	10,000.00
TOTAL LIABILITIES & EQUITY	10,000.00

Maybe not very exciting, but there you are: you're in business! Your fledgling company has assets of $10,000, all of it in the form of cash in a checking account. Because you business doesn't owe anybody for anything yet, it has no liabilities and therefore the entire $10,000 is equity.

Now it's time to gear up for production. You figure you're going to need a computer, a desk to put it on, a chair to sit on, and some development software – Visual FoxPro 9.0, naturally! The following transactions take place:

- You buy a $2000 computer from an online vendor and pay for it with a credit card. (Let's assume it's a corporate credit card so we don't have to deal with loans from you to your business. Don't have a corporate credit card? No problem. Just wait a couple of days and you'll have six offers in the mail!)
- You write a $375 check to a local discount furniture outlet for a desk and a chair (you've decided the $1000 ergonomic chair can wait until the business has shown a profit).
- You buy a copy of Visual FoxPro 9.0 from an online vendor for $584.37 and pay for it with a credit card.

Your balance sheet now looks like this:



New Venture Software, Inc.
Balance Sheet
As of July 18, 2007

	Jul 18, 07
ASSETS	
Current Assets	
Checking/Savings	
Business Checking Account	9,625.00
Total Checking/Savings	9,625.00
Total Current Assets	9,625.00
Fixed Assets	
Computer Equipment	2,000.00
Furniture and Equipment	375.00
Total Fixed Assets	2,375.00
TOTAL ASSETS	12,000.00
LIABILITIES & EQUITY	
Liabilities	
Current Liabilities	
Credit Cards	
Business Credit Card	2,584.37
Total Credit Cards	2,584.37
Total Current Liabilities	2,584.37
Total Liabilities	2,584.37
Equity	
Opening Bal Equity	10,000.00
Net Income	-584.37
Total Equity	9,415.63
TOTAL LIABILITIES & EQUITY	12,000.00

Whoa! Things got a bit complicated pretty quickly, didn't they? How come there are so many new lines on the balance sheet, and how come assets went up from $10,000 to $12,000? After all, the business hasn't even generated a single dollar of income yet.

Let's look at each of those three transactions individually and see how they affect the balance sheet.

- You expect to use the computer for at least three years, so instead of taking its entire $2000 cost as an expense in the current period, you book it as an asset with the intent to depreciate it over three years.[7] The initial value of the computer, which equals its cost, shows up under the heading Fixed Assets | Computer Equipment. Because you paid for it with a credit card instead of cash, you also incurred a $2000 liability that shows up under Liabilities | Current Liabilities | Credit Cards | Business Credit Card.

  Does the fact that the balance sheet shows $12,000 in assets mean your business is now worth $2000 more than when you started? Of course not, because the asset is offset by an equal liability in the form of the money you owe the credit card company. If you sold the computer for what you paid for it and paid off the credit card company, you'd be right back where you started.
- Similarly, you expect the desk and chair you bought to last several years, so they also become fixed assets. The check you wrote to pay for them reduces the balance in your checking account by $375, so on the balance sheet the Business Checking Account line under Current Assets goes down by that amount. On the other side of this transaction, another asset, namely Fixed Assets | Furniture and Equipment, goes *up* by $375. Because you paid cash for the desk and chair, you did not incur any debt; you simply exchanged one asset (cash) for another asset (furniture).
- Although you expect to be using Visual FoxPro 9.0 for several years, its cost was rather small, so instead of booking it as a depreciable asset you decide to simply expense it right away.[8] This transaction therefore has no effect on the asset side of the balance sheet. On the liabilities side, your credit card balance goes up by $584.37, and you incur an expense for the same amount. Because you've had no income yet, your balance sheet shows a net loss of -584.37, which results in a decrease in equity from $10,000 to $9,415.63.

As you continue to operate your business, the balance sheet changes every time you engage in a financial transaction. Learn to read and understand the balance sheet. It's one of the primary ways to keep tabs on your business's financial condition.

The income statement

The income statement is a summary of revenue and expenses over a specific period of time, for example the current year to date or the last fiscal year. Revenue is money that flows into the business, while expenses are money or other value that flows out. For this reason, the income statement can be thought of as a *flow* report.

The income statement presents the following information:

- How much money the business took in (revenue)

[7] Depreciation is a way of spreading the cost of something over its useful life. The length of time over which you can depreciate an asset depends on several factors; three years is used here simply as an illustration. Consult with your accountant to determine the proper deprecation period for each asset.

[8] It may not be permissible to expense software like this in all cases. Consult with your accountant to determine what can be depreciated and what can be expensed.

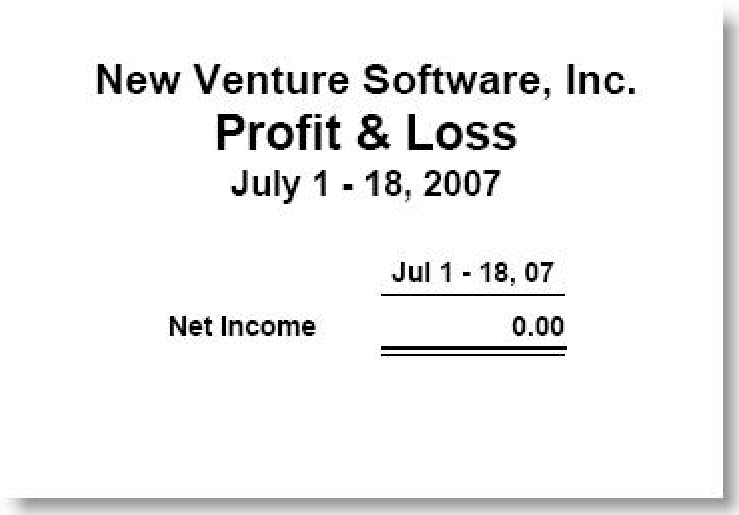
- Minus how much it spent (expenses)
- Equals how much it kept (net income or net loss)

The equation represented on the income statement is this:

```
net income = revenue – expenses && "net income equals revenue minus expenses"
```

A word about the difference between revenue and income: The term *revenue* refers to gross receipts, or all the money that flows into the business, while the term *income* is generally used as a shortened form of the term *net income*, which is what's left over after subtracting expenses. You can think of revenue as a synonym for *total income*.

The day you opened your business and made the initial $10,000 deposit into its checking account, the income statement looked like this:

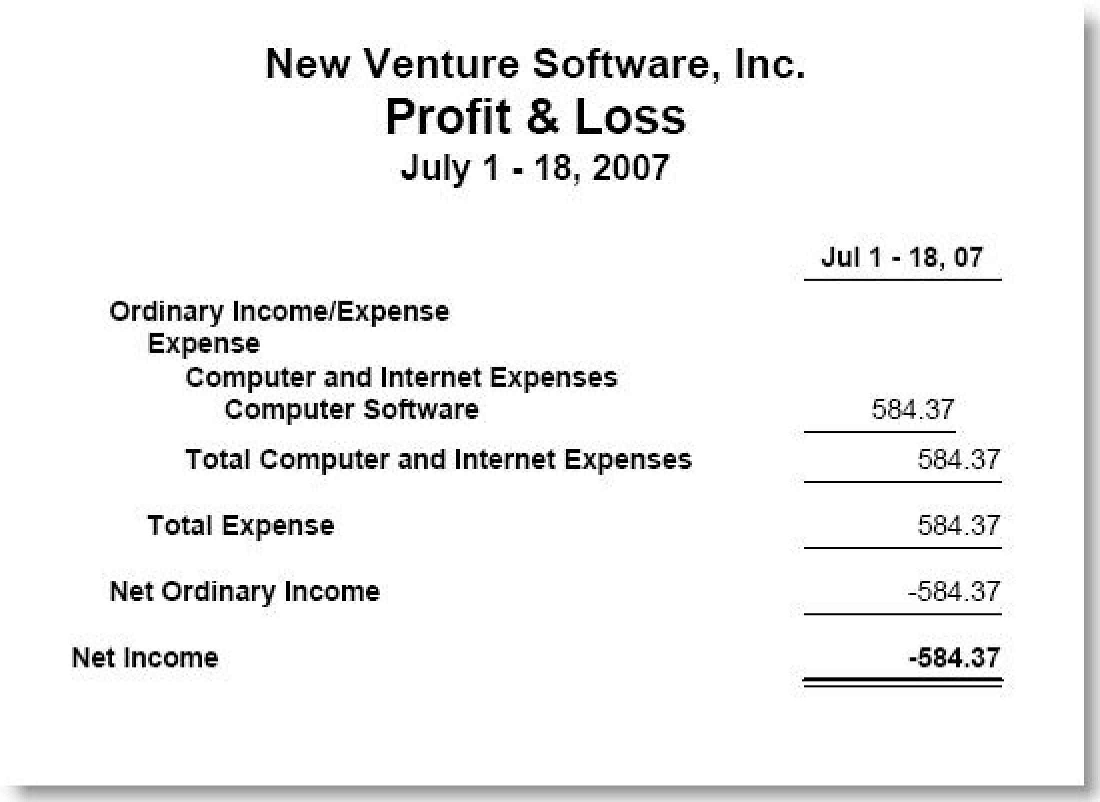


New Venture Software, Inc.
Profit & Loss
July 1 - 18, 2007

	Jul 1 - 18, 07
Net Income	0.00

The implied lines, not shown on the sample above, are total income (revenue) of $0.00 and total expense also $0.00.

Even after buying the computer and the furniture, the income statement doesn't change because those two transactions involved only assets and liabilities. Therefore they affect the balance sheet but not the income statement. However, when you bought the software and expensed it, you created an income statement item under expenses. After than transaction, the income statement looks like this:



New Venture Software, Inc.
Profit & Loss
July 1 - 18, 2007

	Jul 1 - 18, 07
Ordinary Income/Expense	
Expense	
Computer and Internet Expenses	
Computer Software	584.37
Total Computer and Internet Expenses	584.37
Total Expense	584.37
Net Ordinary Income	-584.37
Net Income	**-584.37**

There's still no revenue—nobody's paid you for anything yet—but you did incur an expense, so net income is now in negative territory. Congratulations! You've just learned business lesson one: it's easier to spend money than to make money.

Not all expenses are actual disbursements of cash. Take depreciation, for example. Depreciation is the steady reduction in financial value of an asset over time. It's a real expense, in accounting terms, but it's not a disbursement of cash. Income statements also capture these types of expenses. Depreciation expenses are typically booked at the end of the month, though, which is why the depreciation on the computer, desk, and chair don't show up on the mid-month income statements illustrated above.

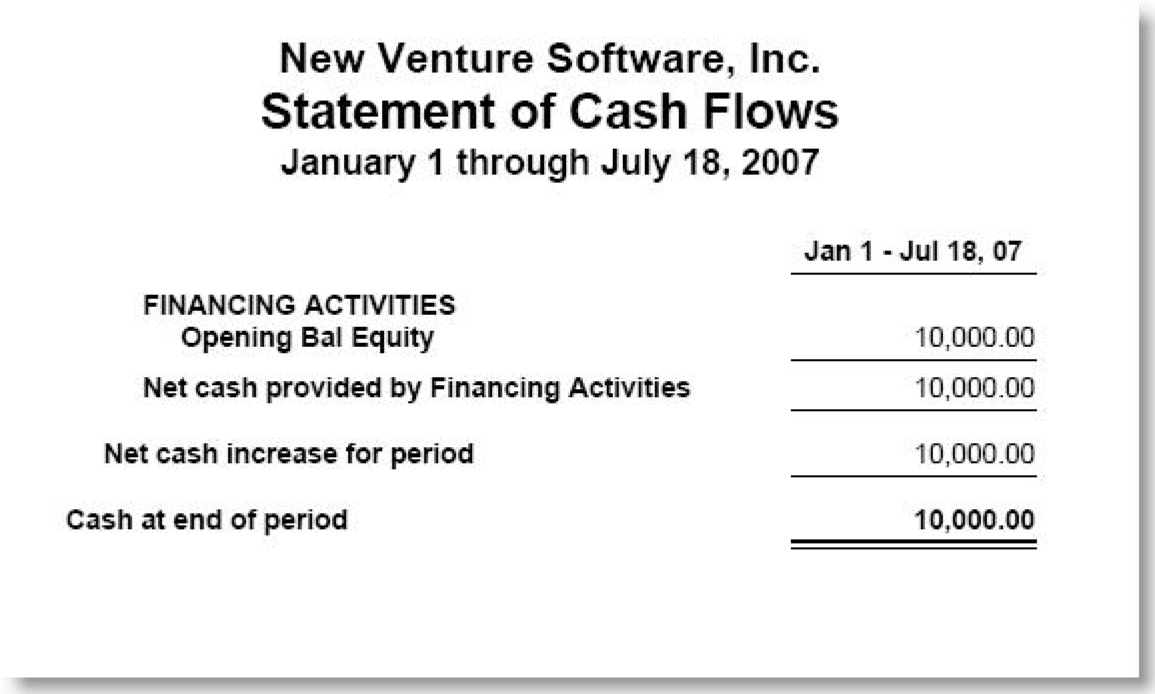
The cash flow statement

As the old saying goes, "cash is king." If a business runs out of cash, or fails to generate enough cash on a regular and ongoing basis, it probably won't be able to continue operations for very long. Therefore it is important for the business owners or managers to keep a close eye on the cash flow.

The cash flow statement is more than just a summary of cash receipts and disbursements. Its primary purpose is to spotlight the amount of cash generated by the operation of the business, but it also includes other items from the income statement and balance sheet in order to show the reasons behind the change in cash position from the beginning of the period to the end of the period.

The cash flow statement can be thought of as a summary of the sources and uses of cash during a specific period of time. The examples below, like all the financial statement examples in this paper, were generated by QuickBooks. Different accounting software may present the cash flow statement in somewhat different formats, but the fundamental concept is the same.

After depositing the $10,000 check to establish the initial capital for New Venture Software, Inc., the cash flow statement looks like this:

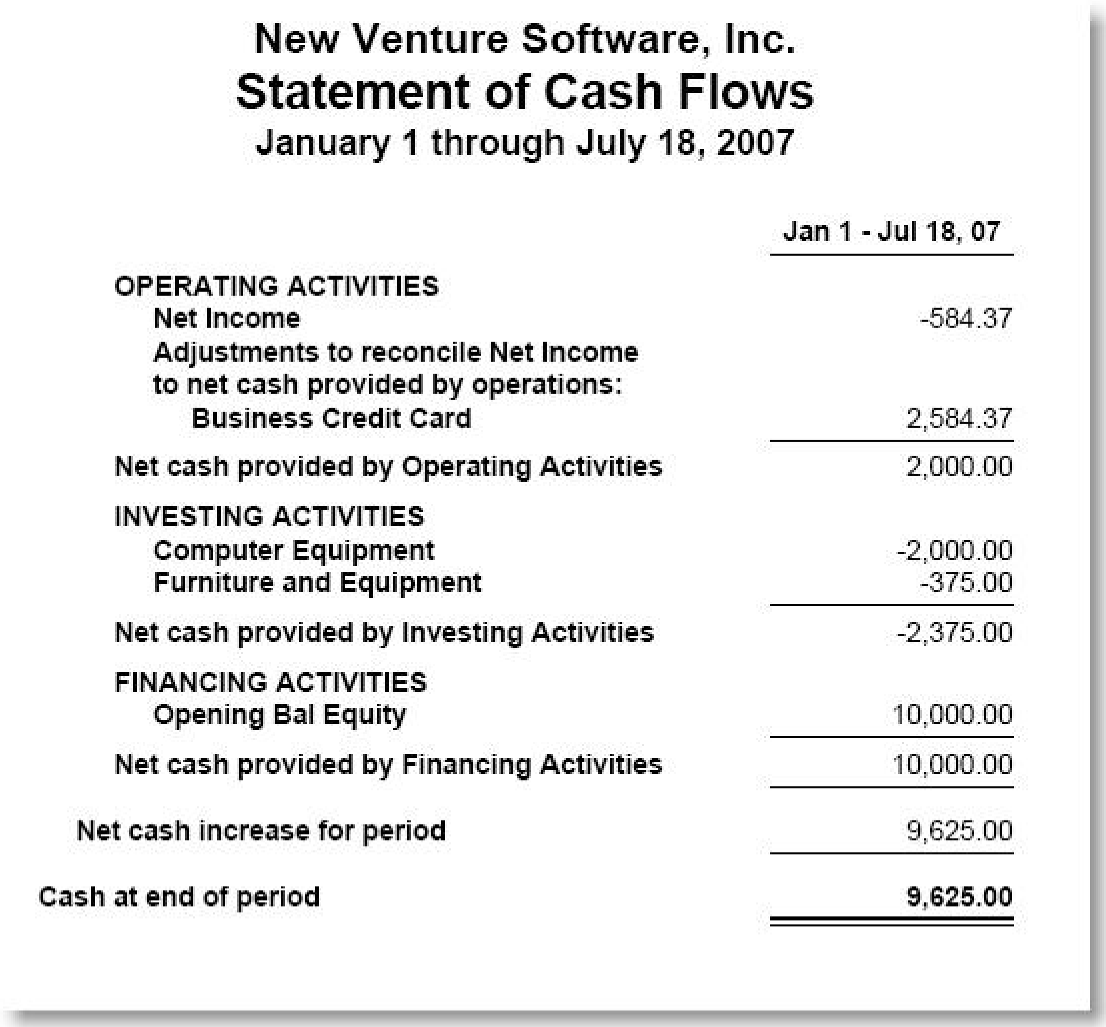


New Venture Software, Inc.
Statement of Cash Flows
January 1 through July 18, 2007

	Jan 1 - Jul 18, 07
FINANCING ACTIVITIES	
Opening Bal Equity	10,000.00
Net cash provided by Financing Activities	10,000.00
Net cash increase for period	10,000.00
Cash at end of period	**10,000.00**

As you can see, there was a net increase of $10,000 in cash, all of it due to the infusion of initial capital by the owner (that's you!). This is not an operating activity—indeed, the business has not conducted any operations yet—so it's presented under the heading of Financing Activities on the cash flow statement.

After buying the computer, the furniture, and the software, the cash flow statement reflects those activities as well.



New Venture Software, Inc.
Statement of Cash Flows
January 1 through July 18, 2007

	Jan 1 - Jul 18, 07
OPERATING ACTIVITIES	
Net Income	-584.37
Adjustments to reconcile Net Income to net cash provided by operations:	
Business Credit Card	2,584.37
Net cash provided by Operating Activities	2,000.00
INVESTING ACTIVITIES	
Computer Equipment	-2,000.00
Furniture and Equipment	-375.00
Net cash provided by Investing Activities	-2,375.00
FINANCING ACTIVITIES	
Opening Bal Equity	10,000.00
Net cash provided by Financing Activities	10,000.00
Net cash increase for period	9,625.00
Cash at end of period	**9,625.00**

Cash flow statements can be a bit difficult to understand. Here's how to read this one, starting from the top.

- Under Operating Activities, the business had net income of negative $584.37 due to expensing the software with no revenue to offset it. The business charged $2,584.37 on its credit card, effectively generating $2,000.00 in cash from all operating activities. Because credit card debit is considered short term debt, it is presented as an operating activity rather than a financing activity, and so the net of these activities shows up as *net cash provided by operating activities.*
- Under Investing Activities, the business invested $2,375.00 cash in two asset categories: computer equipment, and furniture and equipment. This is reflected as a negative number under *net cash provided by investing activities.*
- Finally, under Financing Activities, you injected $10,000.00 cash into the business in the form of initial capital, which appears as a positive number under *net cash provided by financing activities.*

The cumulative effect of these three groups of activities on the business's cash position is shown at the next-to-last line of the cash flow report, which reports a net increase in cash of $9,625.00 for the period. Because the business started the period with zero cash, the cash balance as of the end of the period is the same as the increase for the period.

Cash-basis vs accrual accounting

One more important concept to know about is the difference between cash-basis accounting and accrual accounting. This involves a decision you'll want to make regarding how—or more precisely, when—to book your business's income and expense items.

Under cash-basis accounting, income is recognized (booked) when you receive the cash and expenses are recognized when you disburse the cash. Under accrual accounting, on the other hand, income is recognized when it's earned and expenses are recognized when they occur.

Cash-basis accounting is simpler than accrual accounting. It is commonly used by service businesses that do not maintain a significant level of inventories. It is therefore a good fit for the independent software developer, at least when you're just starting out. Under cash-basis accounting, income shows up on your income statement when you receive payment from a client, and expenses show up when you pay for something.

In the software development business, income tends to be lumpy, meaning it tends to occur in clumps rather than being evenly distributed over time. One downside to cash-basis accounting is that this lumpiness is reflected on the income statement.

Say you work on a project for three months before you receive the first payment (never mind if you would or would not actually agree to do this – it's just an example). Assuming the business has no other income during those three months, but does continue to incur ongoing monthly expenses, the effect of this lumpy income stream is that the income statement shows a net loss in months one and two and a large net income in month three.

Accrual accounting smoothes out lumpy income and expenses by recognizing them as they occur rather than when cash changes hands. Under accrual accounting, for example, you might book as income each month the amount you will charge for the work you have completed at the end of that month, even though you have not yet been paid for it. On the balance sheet, this non-cash income is offset by an asset called work in progress.

Accrual accounting would spread the project's income over all three months in this example, and would thereby make your income statement appear much smoother from month to month. When you get actually get paid at the end of the three months, you merely swap one asset (work in progress) for another asset (cash), which does not affect the income statement.

Under accrual accounting, expenses are similarly evened out so as to smooth the peaks and valleys in the income statement. For example, the expense for the $2,000 computer is taken as depreciation at $55.56 per month over 36 months instead of as a single, large expense in the month it was purchased.

The main downside of accrual accounting, especially for a small business with limited cash, is that the business may end up owing income taxes on income for which it hasn't yet received the cash. For example, if that three-month software development project begins in November and the work in progress is booked as income in November and December, the year-end income statement—and hence the taxable income—will include the income for those two months. The business will need to have enough cash on hand to make the January 15th tax payments even though it won't receive the actual cash payment for that work until sometime later.

If your business has stockholders or venture capital partners to keep happy, it may be important to maintain a relatively smooth income statement, in which case accrual accounting may be the right choice. On the other hand, if you're self funded, as many independent software developers are, smoothing out the monthly or quarterly income statements may be of little of no concern because you're the only one reading them. In that case, cash accounting may be the better choice, particularly for tax reasons. Of course, you can choose to keep financial records on an accrual basis and convert to cash basis for tax purposes, which may give you the best of both.

Management accounting

Management accounting refers to financial aspects of the business that you measure and track internally. Because they are usually for internal use only, management accounting reports may be considerably less formal than the standard financial statements discussed above.

If you're a one-person business, how much or how little management accounting you do is entirely up to you. On the other hand, if you have partners or are accountable to investors or other stakeholders with a vested interest in the business, you may need to prepare and present management accounting reports on a regular basis and in a more formal format.

Work in progress

As an independent software developer, one thing you almost certainly want to track closely is your work in progress. This is especially true if you bill for your work by the hour rather than by fixed price. If you're using cash accounting, work in progress has no effect on your financial statements, but it's nonetheless critical to closely track how much billable time you're generating. When you bill by the hour, time literally *is* money: an hour lost is an hour never to be regained. If you fall behind your weekly or monthly goal for billable hours, it may be very difficult to catch up.

One best practice is to set a realistic goal for the number of billable hours you expect to average each week or month, and then track your billable time regularly to be sure you stay on target. If you don't keep track of billable hours as you go along, you may be in for a sad surprise when you prepare invoices at the end of the month.

There are software tools to help you track billable time by client and project. QuickBooks Professional edition includes a time tracking module that feeds directly into invoices. Personally, while I do use QuickBooks for this purpose, I also use a front-end time tracking tool to log my billable time as I go along. I then transfer that time into QuickBooks on a regular basis and use it to track unbilled hours and prepare invoices.

For the last several years I've used a very nice time tracking tool called Time is Money from Red Ring Software (www.red-ring.com). It has an easy to use interface and (bonus for VFP developers!) it uses an xBase compatible database so you can write your own reports against it. I developed a couple of reports for my own use that format the data in a way that makes it easy to input into QuickBooks. Another tool I used and liked in the past is TraxTime from Spud City Software (www.spudcity.com/traxtime/traxtime.htm).

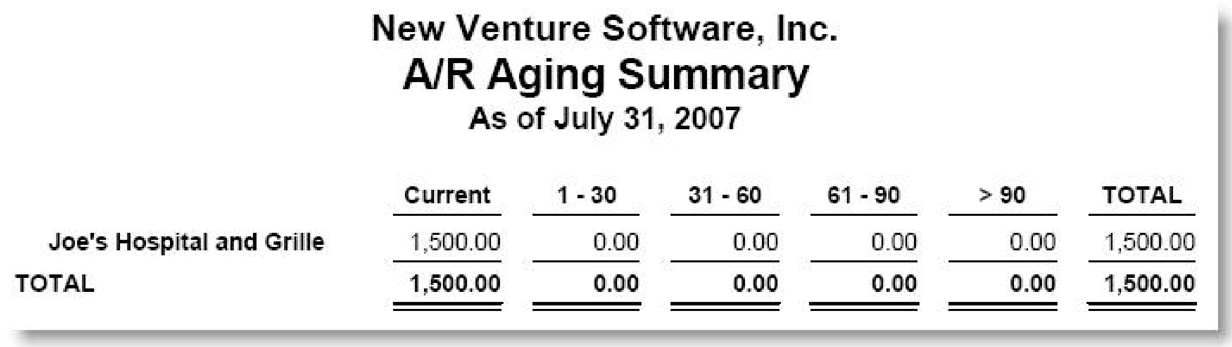
Accounts receivable

Accounts receivable (A/R) represents the amounts owed to you by others. In the software development business, the only receivables you're likely to have are the amounts owed to you by clients for invoices you've sent them and which they haven't paid yet.

If you use accrual accounting, the total of accounts receivable appears on the balance sheet as an asset. If you use cash accounting, accounts receivable is not as asset and does not appear on the balance sheet.

Regardless of which financial accounting method you use, part of running a successful business is managing your accounts receivable. In other words, you need to keep track of who owes you what as well as which clients are paying their bills on time and which ones perhaps are not. The report used for this purpose is the Aged Receivables, or A/R Aging Summary, report.

Let's suppose that by the end of its first month of operation, New Venture Software, Inc. has put in twenty hours of billable time on a project for Joe's Hospital and Grille and sends out an invoice for the amount due. The A/R Aging Summary report as of the end of the month shows the amount due in the Current column.


New Venture Software, Inc.
A/R Aging Summary
As of July 31, 2007

	Current	1 - 30	31 - 60	61 - 90	> 90	TOTAL
Joe's Hospital and Grille	1,500.00	0.00	0.00	0.00	0.00	1,500.00
TOTAL	**1,500.00**	**0.00**	**0.00**	**0.00**	**0.00**	**1,500.00**

The A/R Aging Summary report shows the various amounts owed to you arranged into columns according to how long those amounts have been outstanding. The day the invoice is sent out, the $1,500 owed you by Joe's Hospital and Grille appears in the Current column. Ten days later, it would appear in the 1 - 30 column, meaning 1 to 30 days outstanding. The longer an invoice goes without being paid, the farther to the right it moves on this report. Therefore the A/R Aging Summary report is a tool you can use to see at a glance who owes you money, how much they owe, and how long they've owed it.

Accounts payable

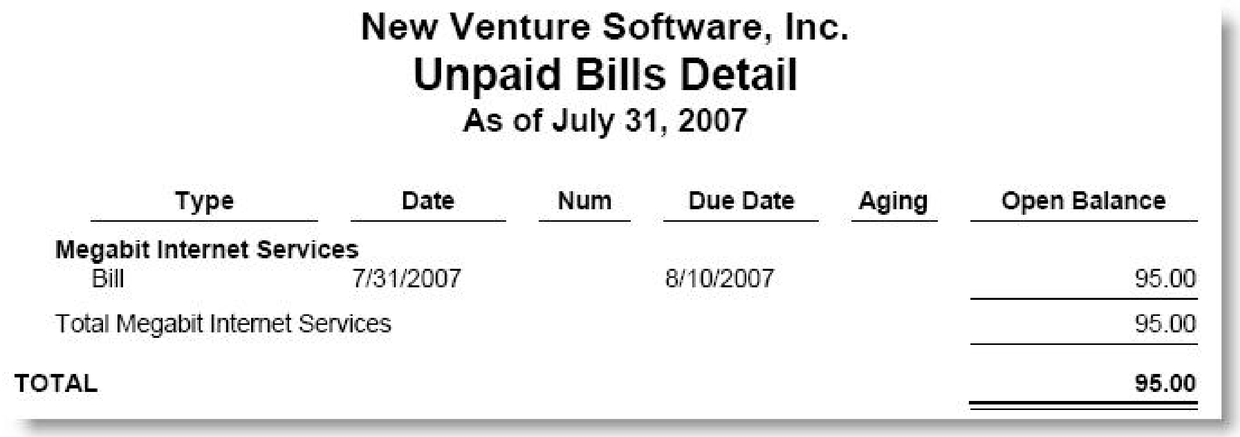
Accounts payable (A/P) are the flip side of accounts receivable. Accounts payable are simply the amounts you owe to others for bills you've incurred.

If you use accrual accounting, accounts payable are liabilities and show up on the balance sheet as such. If you use cash accounting, accounts payable do not show up on the balance sheet.

Just as your personal credit rating is important to your ability to borrow money for personal needs, your business's credit rating is important to its ability to borrow money for business needs. An important part of running a successful business is managing your accounts payable so as to insure your bills get paid on time. If your business develops a history of paying its bills on time, its credit history will reflect that fact and vendors will be more likely to extend credit in the form of a purchase order or open line of credit.

The Unpaid Bills report helps you track the amounts you owe so you can project your cash requirements and pay your bills on time. It lists each bill showing to whom it is payable, how much is owed, and when it's due.

Let's suppose New Venture Software, Inc. signs a contract with Megabit Internet services for a high-speed Internet connection and ten e-mail accounts at $95.00 a month. The first bill arrives at the end of the month and is due ten days later. Your business therefore has an account payable of $95.00, which is reflected on the Unpaid Bills report as shown below.


New Venture Software, Inc.
Unpaid Bills Detail
As of July 31, 2007

	Type	Date	Num	Due Date	Aging	Open Balance
Megabit Internet Services						
	Bill	7/31/2007		8/10/2007		95.00
Total Megabit Internet Services						95.00
TOTAL						**95.00**

Under accrual accounting, the account payable shows up as a liability on the balance sheet, which is offset by an expense for the same amount. When you pay the bill, the liability goes away (accounts payable is reduced) and the balance in your checking account (an asset) is also reduced. Under cash-basis accounting, you don't incur the expense until you pay the bill, but you can still run an A/P report to help manage your accounts payable.

Whatever mechanism you decide use, put a management accounting procedure in place to help track your accounts payable and ensure that you pay your business's bills on time.

Final Thoughts

So you want to be an independent developer. What are your chances for success?

You may have heard it said that 9 out of 10 businesses fail in the first year. One study says that although this is a myth, it is true that less than half of new firms are still open after four years.[9] It's

[9] Phillips and Kirchhoff (1989), cited in *Redefining Business Success: Distinguishing Between Closure and Failure* by Brian Headd, http://www.sba.gov/advo/stats/bh_sbe03.pdf

also true that not all businesses that close do so due to outright failure, but these numbers still show there is significant risk in launching a new venture and that there is no guarantee of long-term success.

I am not aware of any statistics on the success or failure rate of independent software developers per se, but most of the independents I know have been at it for quite a while. While I do know of some people who have left the field for various reasons, I do not personally know of anyone who has outright failed at it. From this, as well as from what I can observe in the industry and from my own personal experience, I conclude that if you approach the challenge of being an independent software developer in the right way, your chances for success are pretty good.

Good luck, and let me know how it works out for you.

Resources

Books I've read and recommend

Collins, Jim. 2001. Good to Great. New York: HarperCollins Publishers, Inc.

Covey, Stephen R. 1989. The 7 Habits of Highly Effective People. New York: Fireside (Simon & Schuster)

Covey, Stephen R., A. Roger Merrill, and Rebecca R. Merrill. 1994. First Things First. New York: Simon & Schuster

DeMarco, Tom. 1987. Peopleware: Productive Projects and Teams. New York: Dorset House Publishing

Fishman, Stephen. 2002. Web and Software Development: A Legal Guide – 3rd ed. Berkeley, CA: Nolo

Gunderloy, Mike. 2004. Coder to Developer. San Francisco, CA: Sybex

Hennig, Doug. 2006. "Best Practices for Vertical Application Architecture." In Visual FoxPro Best Practices for the Next Ten Years, comp. by Whil Hentzen. Whitefish Bay, WI: Hentzenwerke Publishing

Hentzen, Whil. 2002. The Software Developers Guide – 3rd ed. Whitefish Bay, WI: Hentzenwerke Publishing

Johnson, Spencer. 1998. Who Moved My Cheese? New York: G. P. Putnam's Sons

Kawasaki, Guy. 2004. The Art of the Start. New York: Penguin Group

Pountney, Cathy. 2006. "Best Practices for Project Management." In Visual FoxPro Best Practices for the Next Ten Years, comp. by Whil Hentzen. Whitefish Bay, WI: Hentzenwerke Publishing

Purva, Sanjiv and Delaney, Bob. 2003. High-Value IT Consulting. Berkeley, CA: McGraw-Hill/Osborne

Ruhl, Janet. 1997. The Computer Consultant's Guide: Real-Life Strategies for Building a Successful Consulting Career – 2nd ed. New York: John Wiley & Sons, Inc.

Sink, Eric. 2006. Eric Sink on the Business of Software. Berkeley, CA: Apress

Spolsky, Joel. 2004. Joel on Software. Berkeley, CA: Apress

Walsh, Bob. 2006. Micro-ISV: From Vision to Reality. Berkeley, CA: Apress

Weinberg, Gerald. 1985. The Secrets of Consulting. New York: Dorset House Publishing

Yourdon, Edward. 1997. Death March: Managing "Mission-Impossible" Projects. Upper Saddle River, NJ: Prentice Hall

Books I haven't read but which are either classics or look interesting

Berkun, Scott. 2005. The Art of Project Management.

Brooks, Frederick P. 1995. The Mythical Man-Month. Addison Wesley Longman, Inc. (anniversary edition; original published 1975)

Cracas, David J. 1999. Start Your Own Software Company: A Step-by-Step Guide to Setting Up a Computer Software Business

Cusumano, Michael A. 2004. The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad

Hasted, Edward. 2005. Software That Sells: A Practical Guide to Developing and Marketing Your Software Project

McCarthy, Jim. 2006. Dynamics of Software Development.

McConnell, Steve. 2006. Software Estimation.

Spolsky, Joel. 2005. The Best Software Writing I: Selected and Introduced by Joel Spolsky.

Wiegers, Karl. 2003. Software Requirements, Second Edition.

Wiegers, Karl. 2005. More about Software Requirements: Thorny Issues and Practical Advice.

Software Tools

A non-comprehensive list of useful tools for the independent software developer. There are lots of others.

Tools for Time Management

Time Is Money http://www.red-ring.com/

TraxTime http://traxtime.com/

Tools for Project Management

Issue View http://www.issueview.com/

FogBugz http://www.fogcreek.com/FogBugz/

Tools for Project Accounting

QuickBooks http://quickbooks.intuit.com/

Microsoft Office Accounting 2007 http://www.ideawins.com/

Acknowledgments and Copyrights

In writing this paper, I drew a great deal from my own experiences as an independent developer, but I am also indebted to the work of a many smart and talented people who've also written about this topic and about topics dealing with change and self-improvement. In particular, the books by Whil Hentzen, Stephen Fishman, Stephen R. Covey, Spencer Johnson, and Eric Sink, which are cited in the *Resources* section above, have been of special value to me.