Chapter 11
The Sales Call

"You had me at hello."—Dorothy
—Jerry McGuire

In one of his early comedy albums, Bill Cosby hypothesized that graduation from a karate class was not the ultimate goal of taking such a class. Rather, he said, "Eventually, you want to get attacked." In a similar vein, while the goal of marketing is initially to get phone calls, ultimately, it's to make a sales call. More so than the phone call, an in-person visit is your best chance to set expectations while at the same time determining whether you want the job. This chapter covers the in-person sales call in detail.

Call me old-fashioned, but I still think software development requires face-to-face contact. The endeavor you're about to undertake is hard work, full of stress, and often gets touchy. Having a face to attach to a voice—the personal contact—makes getting through those tough times a lot easier.

This works in both directions. How often have you said, "You don't look anything like I pictured, based on what you sound like?" The gruff, coarse voice on the phone may belong to Santa's twin brother, or a taller version of your older sister. It's easier to put yourself in their shoes if you've met with them face to face.

Likewise, remember that we're scary folks. Programmers aren't like normal people—well, on average. Sure, there are a few of you who go to baseball games and teach your kids to ride a two-wheeler on the weekend. But there are a disproportionate number in our business who, to put it delicately, lack some fundamental social skills. And in this stressful situation—the spending of a large amount of money for something they can't touch or feel, or even understand very well—people are inclined to fear the worst.

Meeting you in person can help put those fears to rest. (Or, I suppose, intensify them to the point that they don't leave their house for a year.) When they find out you're a human being just like most of the other people at their company, they'll relax and be able to focus on the matter at hand: assisting you in developing good software.

Purpose

Going into a sales call without a clear set of goals in mind is a waste of time. You should have four items foremost in your mind.

First, you want to determine whether or not there's a match between you and the prospect. Next, you want to get your arms around the size and scope of the project, in at least some rough fashion. The primary purpose of this is so that you can provide some reasonable answer to the inevitable question—"How much is this going to cost?"—but this information will also

be needed so that you can make a decision as to which type of process and payment mechanism is best suited for this project.

The third item is to explain to the prospect how the custom software development business works, and to either propose the type of process and payment mechanism you've selected, or to suggest the options and work out with the customer which is best. And finally, the last is to, if you want to, sell the prospect on having you do a specification for the project they're contemplating.

Identify the likelihood of a match

The sales guys argue that the purpose of the sales call is to get the job, but it's not. You first want to determine whether you want the job and, if that is the case, then you want to do the sales thing—actually work to get the job. This is called "qualifying the prospect." Custom software is in high demand and the available suppliers are scarce, so it's not difficult to find people who want the product you supply. Finding people who have reasonable expectations, who are going to be good to work with, and, yes, who have money, is not quite as easy.

You've already weeded out some of those folks who are obviously not a match during the phone call, but due to the ethereal nature of our business, it's not possible to winnow out everyone.

As my friend Ed Leafe has said, "It is better to be sad that you lost the work than to be sad that you got it." The gain you realize from a good job is much smaller than the potential loss you can incur from a bad job—much like short selling in the stock market has significantly more risk than simply buying stock. The most you can gain from a good job is a healthy profit—but if a job goes south, you can lose your company. It pays to be very careful when doing the initial courtship with a new customer.

Once you've got a handle on them, it's your job to convince them that you can do the work they want done, and you'll spend the rest of the sales call doing so. By your actions as well as your words, you'll set expectations—you want to talk and act so that the expectations they develop are appropriate for the job, so that they are not disappointed in the long run.

Remember, most likely, this customer has not bought custom software before, so they may well have a set of expectations that are unreasonable or impossible. And no matter how skilled you are, or how good a Joe they are otherwise, if they leave this meeting harboring a false set of hopes, the relationship can turn ugly in short order.

First pass at identifying size and scope

The previous section focused on you—what your goals are during the sales call. Don't be fooled into thinking that what you want matters. If you eventually decide to work with this company, they are going to be the one making the decisions and writing the checks, so you need to take care of them all the while. It's just that there's a lot more going on inside your head than what the prospect may perceive.

After the customer gets past the "Are we comfortable with this programmer" stage from their own point of view, they'll want to know the answer to two questions: "How much is this going to cost me?" and "Can you have it finished sometime late next week?" At the same time, you're going to have to answer the question, "What type of process and payment mechanism is going to be best for this project?"

If you can provide a good, albeit very rough, answer to the first two questions to your customer, and garner enough information so you can answer the third question to yourself, you'll make your prospect more comfortable and enhance the possibility of putting together a relationship.

This part of the sales call usually takes the longest—you'll spend time with them reviewing what they're looking for, all the while asking questions and gathering information so you can determine the answers to these questions yourself.

Pitch the price and best process

Once you've determined what the approximate price and best process to use are, it's time to deliver the news to the customer. These two issues are very much interrelated. And there are two approaches to take.

The first is when the customer asks, "So, how much?" Note that they won't word the question exactly like this—more likely, they'll have spent anywhere from 20 minutes to a couple of hours describing what they're looking for, and then they'll say, "Well, naturally, we won't hold you to this answer, but can you give us a ballpark figure for this project?" No matter what you say, they'll remember the number you give them. If you try to remind them later that they promised not to hold you to this preliminary number, they'll always respond with "Yes, but…" and deliver some incredible rationalization (or irrationalization?) as to why that promise is no longer valid.

The second approach is when they don't ask, "How much?" You can take this as a sign that they either aren't interested, or have something up their sleeve such that the price is inconsequential. Customers *always* want to know how much during the sales call; if they don't, you have to watch out! For this reason, if they don't bring up price, don't heave a sigh of relief and think that you're getting away with something—cuz you aren't. Bring up the question yourself, and see how they react.

In both cases, you need to cover a broader ground than just giving them a dollar figure. You may propose a single development process, and not let them have another choice, because their circumstances preclude any other choices. Or you may propose that the two of you work out the choice of development process together.

I'll explain how to do this later in this chapter.

Sell them on the next step

The last item on your internal "To-do" list will be to close the meeting and confirm the next steps. What those steps are depend on 1) whether or not you want the gig (and they want you), and 2) what type of process you've decided on.

If you don't want the job, you need to excuse yourself gracefully from further consideration—sort of the same as claiming that you have to stay home Friday night because you were going to wash your hair. Well, maybe a bit less transparently than that. And if they don't want you, they may come right out and say so ("We are looking for someone who <fill in a condition>") or they may let you down gently, such as with a line like "We'll have to do some more research and will get back to you."

Assuming that the two of you want to continue together, the next step depends on the type of process you've decided on.

If you're going with a process that involves a specification, then the software development project is divided into two pieces. Just like a building, a custom software application requires a blueprint to be used during its construction.

Taking the building construction analogy a step further, a blueprint is designed by an architect. And that architect may or may not be involved with the actual construction of the structure. Similarly, my firm always proposes the development of a specification as a service separate from the development of the application itself.

This approach gives the customer peace of mind. They feel that if they're not happy with your work after a certain point, they can walk away from you and go somewhere else—providing them with a safety valve.

Thus, remember that you are trying to sell them on doing a specification, not on the whole programming job. If they like the spec, then the development will come naturally. I've found it's a rare occurrence to write a complete specification and then have the customer go elsewhere for the development itself. During the sales call, don't look so far out in the future that you lose sight of the short-term goal.

On the other hand, if the gig doesn't involve a spec, then it's time to get rolling with development. Even then, however, it doesn't mean that you head on back to the shop, create a new directory on your development machine, and start slinging code. First, you'll need to formalize the relationship—including the specifics of this type of process—with an Engagement Letter, and then you'll need to meet with them to determine *something* about the application they want.

Note that a prospect who wants you to start coding before specifications have been developed—but do it for a fixed price—is a prospect better left on the mountainside to die a slow, miserable death. Hopefully you've weeded that type out early on!

Goals vs. agenda

There is a lot happening on several levels during the sales call, much like a blind date. On the surface, the two of you are getting acquainted, and after some pleasantries, talking business and maybe talking a bit of tech. Under the surface, you're checking out the customer, and, hopefully, they're doing the same to you. It can be easy to get sidetracked during the actual visit, so I just want to mention again that the previous four sections are items on your internal "To-do" list. The actual agenda of the sales call, which I cover later in this chapter, can be significantly different!

Where to meet

As much as it is a hassle for you, the sales call should always be made at the customer's location. I'll discuss the other options as well, but this has been my experience—meeting first at their place of business has the most advantages all the way around.

Pros and cons of "Their Place"

There are four reasons for meeting at their place.

First, it's a natural courtesy, and most people expect it. They're the customer; you're the vendor, and so you do the legwork.

Second, many times, the customer will want you to look at files, programs, printouts, and so on. There is usually no practical way to do this if they have to pack up everything and bring it to your office. Even if they try, they're more than likely to forget something.

They may also want you to look at their existing applications and systems, and those may not be portable.

Furthermore, you may need to (or they may want you to) meet with other people who don't have to be available for the entire meeting. It's terribly inconvenient for them to bring additional people along in this situation.

The third reason is that you need to scout out the territory. There's a lot you can learn from the digs a customer resides in. First of all, by visiting them, you'll find out a lot more clearly how big or small they are. You will also find out the general atmosphere of the company—are they just basically nice people? I learned a long time ago that there is a percentage of people in the world who simply aren't very nice, and I decided not to do business with them. Let them make the lives of my competition miserable, not mine.

Next, does it feel like the company is on solid ground? Do they have a frugal image, or have they spent enough money to make their people reasonably comfortable? If people are stacked two to a cubicle (no, that's not just in Dilbert's imagination) and the building is in disrepair, could the company be in financial trouble, or are they just grossly stingy with funds? If so, could this be a sign that they're going to try to nickel and dime you? If a company is going to spend $10,000, $25,000, or $50,000 on an application, a cheap mentality isn't going to be conducive to a good working relationship.

You'll also get a feel for how the company operates. The reason you're there, generally, is because they've got a problem and they want you to solve it. Is it possible that the problem is not one that the implementation of a software application will fix, but rather it's an operational problem and automating it will simply make the same mess in a shorter period of time?

The computer environment is another aspect of the company that you'll be able to see with your own eyes. Over the phone, they may have described that they've got "pretty much new computers" with "a brand-new network." Once on-site, though, you see machines with filthy keyboards and dust-laden screens, no-name system units with the covers off, and a laser printer balancing on a stack of books while you're stepping over network cabling strung across hallways.

This tells you two things. One is that either they're lying to you, or their definition of "new" comes from a dictionary you've not seen before. The other is that you may be in for a couple of challenges that you hadn't expected—many a developer has wasted time trouble-shooting a supposed programming problem only to find out that cheap NICs are dropping packets.

Finally, you have some control over when the meeting ends. If worse comes to worst, you can simply walk out. A bit more graciously than that, hopefully, but it's difficult to leave your own office to terminate a meeting.

This much said, meeting at their place isn't always the answer. It's possible that your contact isn't well organized, or that they fool themselves into thinking they're more organized than they really are. In this case, they will often fail to prepare properly or as completely as they would if they had to pack up the whole show and take it on the road.

Second, the company culture may be such that interruptions are the natural order of the day. It's possible to get an hour's worth of work done in a four-hour meeting in this type of place. It still may be valuable to meet in such an atmosphere, if for no other reason than to

learn this, and to then account for it during future work. But if you really have to be efficient, then perhaps another location is the answer.

Pros and cons of "Your Place"

Suppose the prospect wants to (or even demands to) meet at your place? They may want to check you out just as much as you want to check them out. And, if you've implemented some of the ideas discussed later on in this book, you should be proud of your place, and want to show it off. However, you might consider postponing that for a future meeting, if possible.

Second, they may also realize that they're not going to get any work done in their own office, and prefer to get off-site.

If they're a long distance away, they also might feel uncomfortable asking you to spend the better part of a day traveling to and fro, particularly if the sales call is fairly speculative.

There could also be a hidden reason—suppose your office is downtown, and they want to go shopping for the holidays. They can combine a business trip with pleasure, and perhaps even get their company to pay for lunch and parking. (I've had it happen more than once!)

On the other hand, just as with their site, you'll have to control interruptions and other demands on your time—and you'll want to make sure you are as prepared for the meeting as you would be if you visited them.

If you work out of your house, there are a whole host of additional issues involved in having a customer visit you where you live. First of all, you may not want to have to keep the house, filled with children and their accessories, in a constant state of readiness. Second, offices, while they can be lavish or not, reflect the business. A customer may make judgments about you based on your home that would be inappropriate. Finally, you (and they) may simply be uncomfortable holding a business meeting at your home.

Pros and cons of "A Neutral Place"

In some cases, neither your digs nor theirs may work out. I dislike meeting in, say, a restaurant, for a whole list of reasons. It's generally an awkward atmosphere—just about the time you're starting to discuss some critical or touchy issue, the server stops by with another annoying question. They'll often try to hurry you through your meal. There isn't enough table space to spread things out. And try to find a plug for a computer, much less the tabletop space for that as well. And if you've got more than two people, no one can see the screen of a notebook anyway—do you really want to bring an LCD panel or separate monitor into a restaurant?

Bottom line—if it's the only way to go, it beats missing the opportunity to talk to them, but keep this option at the bottom of your list.

What to wear

One of the longer threads in memory on CompuServe's FoxUser forum was titled "Dress for Client?" In the thread, the poster asked whether it was necessary to get dressed up to go to a client for a 10-minute visit that consisted of picking up a new set of data files and installing a new report.

Of course, as many threads on CompuServe do, this one eventually degenerated into unrelated topics, such as the discussion of what items men keep in their pockets that women keep in their purses (one friend said he kept a comb in his left rear pocket, to which I thought,

"Braggart!"). But the gist of the thread dealt with whether or not dressy attire was required, recommended, or simply superfluous.

I have conflicting emotions on this topic. First and foremost is the desire to make the customer feel comfortable. If the customer could possibly be offended by what you're wearing, then it's a mistake. This, of course, goes both ways. Perhaps your prospect is at a downtown bank—you'd want to get dressed up because, more than likely, the bank employees still respect a formal dress code. Your prospect may not care themselves, but what if their boss happens by? Better to err on the side of caution.

On the other hand, it's been known to happen that a consultant wearing a $2,000 suit can offend the employees in a factory environment where "dressed up" means buttoning a shirt and washing their hands. You wouldn't want your prospect to feel you were "slumming" when you were visiting their office.

The second issue I personally deal with is that I'm really selfish, and that extends to wanting to be able to choose what I wear, instead of having another entity dictate to me what the uniform shall be. But that's just me. Fortunately, more and more, a casual dress code has become commonplace, and I've been able to survive with a corporate clientele and still not change out of dress slacks and polo shirts for the past couple years.

Of course, your mileage may vary. Here are some options.

A suit

Although the heading says "A suit," I'm really talking about "dressing up." Now, you may ask, "What constitutes 'dressing up' for the customer?" To my way of thinking, you can wear anything you want, as long as, for men, it's a dark suit, white shirt, and conservative patterned tie, and for women, it's essentially the same, a suit with hose and pumps, and a white or light-colored blouse. I read that *Dress for Success* book back when I was an impressionable young weenie, and I've never been able to shake those inclinations. Of course, depending on the time of year, the suggestions laid down may be personally inappropriate. If you've got hair down past your shoulders, wearing a suit and tie looks more out of place than if you were to don a leisure suit.

Perhaps you could break these rules if you were in Hollywood (flashy suits and fast cars) or Miami (Hawaiian shirts and fast cars), but not in the rest of America.

Business casual

A lot has been made of "Casual Days" in corporate America over the past few years, and some of it is beginning to spread to other parts of the world. It's important to note that "casual" doesn't mean the same thing everywhere. At a manufacturing plant, "casual" may truly mean anything but bike shorts, halters, and flip-flops. At the headquarters of a multi-national, "casual" may mean dress slacks, dress shirts, shined shoes, and the related accessories. In other words—just no tie and jacket.

Don't just assume that you can wear a pair of Levi's and a T-shirt asking, "Have you hugged your ISP today?" when your customer tells you "We've got a casual dress code here."

Programmer casual

One of my favorite anecdotes has to do with "programmer casual." One day in the financial district of a large American city, two of my friends met for lunch. One was on a sales call from

a nearby city, meeting with high-level executives on a mid-six figure application, and was wearing a blue suit, white shirt, and Ivy League tie. The other was visiting the city on a different business venture, had hair half-way down his back, had gone unshaven for, well, more than a couple of days, and sported the latest "in your face" admonition of a sneaker manufacturer on a T-shirt worn over a ratty pair of jeans and worn sandals. Said the one more casually dressed, "Gee <name>. People pay me for what I *know*."

While this actually happened, remember that the clothes you're comfortable in while programming may take the idea of "casual dress" too far for a customer. While I will go to outrageous lengths to avoid having to wear a suit myself, at the other end, a shirt with some sort of collar, pants without tears, holes, or marks, and clean shoes are a minimum.

Again, your perspective and environment may dictate a different level.

The point to dressing to see a customer is that what you wear should not detract from the message that you are delivering—in fact, what you wear should be invisible to the customer, so they can concentrate completely on the content of your message. If you're in doubt, dress up. You can always take off a tie or jacket if you've overdressed.

What to bring

You can divide potential things to bring to a sales call into two categories—supporting materials for your company, and demonstration materials such as sample apps.

Materials

Gosh, I'm simply appalled at the number of people who go on a sales call and forget to bring a business card, a notepad, or literature about their company. Not when they don't have them, but when they do, and just forget. How can you ever walk out of the office without a few business cards and a pad of paper?

If you're one of those who never seems to have everything in place, consider putting together a "Things to take to meetings" checklist, and keep a file folder stocked with these items.

You might consider keeping a stack of business cards, a pen, and a legal pad in your car's glove compartment—as a friend does—so that if you run into someone as you're driving to the beach in your swim trunks, you still have the gear to take notes and make a contact.

And, finally, remember that portfolio I mentioned back in Chapter 6, "What You'll Need"? The sales call is why you put it together in the first place!

A PC

Keep the goodies—particularly the newest gadgets you've picked up—to a minimum. You may want to bring a PC to demonstrate some of your applications, but don't get sidetracked showing all the cool things you can do. Customers want to talk about themselves, not you.

Furthermore, they probably won't understand much of what you show them anyway. It's hard to successfully demonstrate anything but the most trivial of applications in a short period of time.

It's also possible that they will focus on extraneous details ("How did you get the title to turn blue?" or "What's that red thingy in the middle of your keyboard?") instead of the business problem being solved. Why expose yourself to that possibility? There will be time later for demonstrating work you've done in the past.

The agenda—your game plan for the meeting

It's very important that you have a plan for the meeting. I usually try to hold a sales call to between an hour and a half and two hours, but I've been on sales calls that took five hours or longer. These don't do you any good—neither you nor the prospect can remember half of what goes on in a meeting that lasts that long, and it usually takes that long because one or both parties is disorganized or easily distracted, or the customer is trying to get some consulting done for free.

You must control the meeting—that's what a sales call is, after all, a meeting. And your best weapon for doing so is to have an agenda. In fact, the best way to control the meeting with an agenda is to fax the agenda to the prospect in advance.

They'll want to show you what they've got. You'll have some questions. They'll want to talk for hours. You'll want to see examples of things they're talking about. They'll think up more things while they're talking. You'll discuss similar projects you've done. They'll have their eyes opened up to the myriad possibilities that two hours ago hadn't been considered. You'll point out some ideas. They'll ask some off-the-wall questions.

Obviously, the conversation could range all over the south forty. There isn't a single cookbook approach to making this an efficient, purposeful meeting, but there are two pieces to this meeting.

The first is the external plan—the agenda that you sent them—that provides structure to this meeting.

The second is a set of checklist items that you're going to want to answer yourself by the end of the meeting. These issues aren't necessarily things that you can bring up with a customer, but they're things you're going to want to keep in mind as you're going through the external plan.

The external plan consists of the following items:

- Reiterate the purpose of the meeting.
- What is their pain?
- What systems do they have?
- What do they want and what do they expect?
- What software tools do they have to go from current to desired?
- What is their budget?
- What is their timeframe?
- The sales pitch—you, dollars, time, and process.
- Agree on the next step.

Your internal agenda should include the following items:

- Who else are they talking to?
- How serious are they about this project?

- Realistically, what are your chances?
- How involved do they want to be during development?
- What's the decision-making process?
- Who is the champion for this project?
- What are the possible processes for this project, and which one is the best choice?
- If the champion doesn't hold the purse strings, who approves the dollars?

It's a tricky juggling act—making sure they adhere to the agenda is tough enough; keeping your own internal agenda in mind as well makes this process quite a challenge.

The external plan

Reiterate the purpose for them

This is a sales call—a get-acquainted meeting so that they can check you out, you can see what they've got cooking, and the two of you can determine whether there's possibly something that can be worked out between the two of you. So they're going to want to determine whether they're comfortable with you, whether they feel they can trust you, and whether or not you might have the technical and business savvy to help them.

Naturally, they'll be thinking along these lines, but they may well not have determined just how they're going to make that determination. More often than not, it comes down to "They sounded like they knew what they were talking about." But in our business, it's not that difficult to sound like you know what you're talking about—all you have to do, really, is be at least one full chapter further along in the "buzz word textbook" than the other guy.

Since most anyone who walks in the prospect's door can yak about something vaguely technical for a while, you'll want to set yourself apart by explaining to them how they should evaluate you. You can tell them that not only are you going to learn about what they're looking for, but you will also explain how your firm works, and what they can come to expect from your company. And, finally, you'll tell them how you're going to go about sizing and scoping the application they're interested in. Remember that you gave them homework during that initial phone call, so they're already a little bit used to taking direction from you.

At the same time, as I've discussed, you're going to be evaluating them. Of course, you can't exactly say that to them. Instead, you'll want to couch your mission in terms of "seeing whether there's a fit between our firms." After all, it's entirely possible that they may have requirements that you can't meet. Those requirements might be to provide 24-hour on-call service, or to have expertise in an area of process control that you're not able to provide, or to do most of your work on the weekends. It may be that the job is too small for your firm—or too big. Another requirement might be that you'd have to be able to get along with a company populated with jerks, or to do $50,000 applications for $10,000.

No matter what the requirements, if you can't meet them, you should bow out. And that's what you're trying to determine—can you fulfill their requirements?

It's extremely important to remember—and to communicate—that this is *not* a problem-solving session—that type of intellectual work is what you get paid for!

Depending on the nature of the company and whom you're meeting with, you may want to tour the office, meet some people, and so on.

What is their pain?

This was briefly discussed in the previous chapter about the phone call. Again, the most important thing you can do is to define or pinpoint exactly what is causing them pain. If they can't describe it succinctly, do not let go until they can. This is your ultimate benchmark for determining whether the final product is a success.

Along these same lines, ask yourself, what is the customer's biggest fear, and what can you do to alleviate that fear? They really aren't all that interested in computer code or cool development tools—they've got a business to run and they want their problems solved. They may feel that computer programmers are weird and scary, so consider what you can do to put their mind at ease. You may not be able to ask them flat out "What can I do to make you less frightened?" but I've found a fair measure of success in asking similarly blunt questions. For example, "What else can I do to help make this project a success?" and "What critical success factors do you know about for this project already?"

What systems do they already have?

You'll need to check out current software, current hardware, and the ancillary things like manuals, source code, documentation, and so on. Yes, you've been over this on the phone, but it's easy for them to make assumptions about things during the "heat of the phone call" or to make mistakes. Your contact may not even have known the answers to these questions, but didn't want to admit so for one reason or another.

They may tell you that they have Windows 95, but, in actuality, they've got the box—still in shrink-wrap—or it's actually running, but only on the machines in Purchasing. That environment would kind of put a damper on the possibility of writing with a 32-bit development tool, wouldn't it?

The military's credo says to "Trust, but verify." Nonsense. Just verify.

When it comes to data, don't take their word for it. Get the documentation, and even more so, get samples. I wish I had a nickel for every time I was given file layouts or site maps that were six months out of date compared to the actual data sets.

What do they want and what do they expect?

This, of course, is the longest, and most important, part of the meeting.

Your goal is to get a rough idea of how big this system is and how they think it might work. Issues regarding whether or not the Company Type field can be filtered depending on who is using the system are irrelevant at this point, but it's usually the level of granularity the user wants to cover.

Begin this part of the meeting by reviewing their homework. This will do two things for you. First, if they've actually done it, you'll find out that they're, to some extent, serious. And second, if they've done it, whether or not they followed directions, and how much detail they've provided will tell you *how* serious they are as well as how technically savvy they are.

I once had a customer attempt to prototype some screen layouts in Visual C++—using a command button to represent every control on the form. Yes, that's right—there were about 100 command buttons on the form, representing text boxes, labels, check boxes, option buttons, and even command buttons. The customer had been shown, by a friend, how to draw a form using Visual C++, and how to put a command button on the form as well. The friend assumed that the customer would be able to extrapolate that knowledge so that he could put other controls on the form as well—but, obviously, that assumption was wrong. It took me two days to undo the damage done by the customer's friend and the customer's good intentions.

The work the customer has done gives you a place to start. It's interesting to note (although you can't tell the customer this) that the actual work they've done is not that important at this stage. For example (albeit a trivial example), they might have listed 15 screens in their system: Customer Add, Customer Edit, Customer Delete, Customer Query, and Customer Find, and the same five for Vendors and for Employees. More important is that they know they want the ability to perform a variety of maintenance tasks on three entities. How these tasks are specifically implemented isn't at issue yet. You're trying to get to the point where you've identified that the system basically consists of three entities and some fairly standard tasks for each of them.

By listening to them discuss what they've put together, you can also get an idea of their expectations—both the general level of their sophistication as well as specific functionality issues. They have probably already painted a picture in their mind of what this application is going to do, and they're probably going to have trouble explaining it all to you. Some things are going to be difficult to explain, and other things are just going to be "assumed"—"Doesn't all software work like that?"

I once had a customer who kept comparing the custom app I was putting together for them with Quicken. They were continually disappointed that the interface didn't sport every fancy feature that Quicken did. First, they assumed that since Quicken lets you do "X," and that since Quicken is a Windows application, every piece of software that runs on Windows should be able to do "X." Second, their general attitude was that since it's in Quicken, it's obviously possible, and since Quicken only cost $179, it couldn't have been very hard—so why don't you do that? Don't you know how?

It took several meetings to get it through to them that Quicken also cost more than the $120,000 budgeted for their application to develop—and that if they wanted to pay the same millions of dollars that Intuit spent on developing Quicken, it'd be no problem to duplicate any feature they wanted.

The point is that their expectations—no matter how unreasonable—were important to them, and until I was able to reconcile their expectations with reality, we were going to have conflict and someone was going to be unhappy.

The ideal would be for the prospect to have done their homework exactly as you had asked on the phone, and to be able to identify every screen, report, process, table, and external entity in the system in a 45-minute discussion. If you can do this, then when the "How much is this going to cost?" question rears its ugly head, you can cock your head to one side for a moment, and then give them an answer.

I've done this before, and, given a couple of caveats, it works remarkably well. First, the user has to have a very clear idea in their head of what they're expecting. For instance, they might want a complete rewrite of an existing system with a new tool or for a new platform, so they've already figured out the functionality they want. Or perhaps the prospect has recently joined the company and worked with a system at their old job that has given them a head start. In these types of cases, the user can very likely give you a rundown of what they're looking for in some detail.

Second, you have to have some experience in developing specifications and applications of this type. They may have an extremely detailed set of requirements, but if this is only the second intranet application you've developed, you're not going to be able to give them an answer. Of course, that's the case with costing out anything—if you haven't done it before, then you can't say—period.

However, if you do have some history, you can do it. I'll explain exactly how to determine a dollar figure to give the prospect later in this chapter.

What if they haven't done their homework, or, they've done it, but very badly? In either case, they're still going to have the same expectations by the end of the meeting—that they're going to get an idea of how big the system is, how much it's going to cost, and whether or not you have a clue about developing it.

In this scenario, I help them sketch out major entities and functions related to those entities. Then I go back and identify any other major tasks that aren't tied specifically to an entity. This process is reasonably quick, and I can capture a large percentage of the primary functionality quickly. From this I can make a guess at how many interfaces, reports, and processes might be needed, and that's what I'm looking for—a rough idea.

Naturally, this will change during the specification process, but what I've done is put a stake in the ground, so that when the scope of the project changes and they see more dollars than originally estimated, I can explicitly explain why. "We started out assuming a system with 10 primary screens, 15 supporting screens, six lookup screens, two dozen reports, and two posting processes. It's now grown to 23 primary screens, 31 supporting screens, eight lookups, three dozen reports, and still two posting processes. This is why we've exceeded our initial estimate."

Naturally you will need to keep them informed along the way about how the bill is growing. The point here is that by documenting, from project inception, how big the system is, you have a baseline that you can refer to later on.

It's easy for the meeting to get off track as soon as this topic is broached. Many developers have run into the user who fancies himself a system designer, and expends a great deal of effort doing what he thinks is a design. Unfortunately, the result is the equivalent of drawing every pane of glass in every window in the front elevation of a house, but never laying out a floor plan, much less deciding how many rooms or what type of electrical, plumbing, and HVAC systems are required.

This is why the assignment of homework over the phone is a good idea. It gives you control over the path of the design discussions, and sets the stage for what you want to get accomplished during this first meeting.

What tools do they have to go from current to desired?

How prepared are they to begin the design of a specification? Do they have a list of desired functionality? Sample screens? Report layouts? Descriptions of processes, operations, flow of work through the system? Or are you starting out with a blank sheet of paper?

Do they have any expectations of how you are going to do your analysis and design? Are they going to expect to be part of the technical design? In other words, are they simply going to read drafts of specifications and Ooh and Aah during prototype demos, or do they want access to the design and modeling tools along with your staff? If so, are they capable of using them?

What tools do they have? Are they going to have to upgrade existing hardware, buy new hardware, upgrade versions of software, install a new operating system, put in a better network?

What kind of shape is their data in? Can it be used as is? Converted from an existing system? Can the existing systems be used in any way? How might they go about a transition from the old data to the new data?

How rigorous are their current MIS procedures? Do they have the ability to audit the information they produce? What are their alternate plans for disaster recovery? Do they perform regular backups? Have they ever done a backup? Do they know what a backup is? Can they spell "backup"?

What is their budget?

Now is the time to find out how much they want to spend—really. They may have talked around the question on the phone, or given you a vague answer, or perhaps just not answered at all. Or perhaps you didn't ask. In any case, you need to know now.

If they say they don't have any idea, either they don't know what they are doing, or they are lying. A lot of people feel that by giving away a budget number, somehow, magically, the price of the project will be just a little bit over the budget number. So I don't ask them what the budget is, but whether they've got a budget—and what ballpark they're working in. If they say they don't even know what their ballpark is, then I suggest that they define the problem that this software system is going to solve very carefully. Then, they should figure out what the solution to that problem is worth to them.

If they don't even want to provide a ballpark figure, if they're not willing to provide even a rough idea, then I tell them that it's going to be more expensive in the long run. If I have to create something without knowing what parameters I'm working with, there's going to be some missteps as we discover where we want to head.

Would you hire an architect to design a house or a building, and refuse to give them any idea of your budget? Of course not. First of all, before you spent hundreds of thousands or millions of dollars on a building, you would have done some thinking about what it is you're trying to build. Is it a two-story cottage with a spare bedroom and 1.5 baths? Or a 10-story office building that's wired for the special needs that multi-national media companies have? And once you pinned down what it was you were trying to build, you'd do some homework to

get an idea of what that might cost. If it turned out to be more than you could (or wanted to) afford, then you'd scale back your needs or find some more money somewhere.

But you wouldn't contract with an architect without doing a bit of this beforehand. If you absolutely had to start without having any idea, the architect's first duty would be to help you define your needs and explain, roughly, what they might cost.

Similarly, your prospect either has to give you some sort of idea about what they might want to spend or be able to spend, or contract with you in order to determine that information. A software developer, just like an architect or other professionals, has one resource that is scarce above all else—time. And thus you can't spend large amounts of your time on a project without getting paid.

Depending on how things feel at this point, you can offer to help them develop the budget, for a price, since they have no idea or aren't "current" on hardware pricing and software development. If they balk at your hourly rate for doing this, then it's definitely time to put on the walking shoes. If they don't balk, this will go a long way toward cementing their relationship with you.

What about the situation where they don't even know what they want to do? They just have this vague sense of dread that if they don't get this problem solved soon, they're going to be in bigger trouble than they are now. But they're not even exactly sure what the problem they're trying to solve is, or what the best approach will be.

You can then, instead of asking for a budget number that they've already determined, simply start developing away, with the explicit understanding that the final result may be the better part of an order of magnitude larger than they first thought. Yeah, I know they said they didn't have any idea—but, that's not really the case, is it?

When someone tells me they "have absolutely no idea," I work through the following exercise with them. First, I ask them if they think it will cost more than, say, $1,000. They usually say yes. Then I ask if they think it will cost less than $10 million. They again usually say yes. So then I suggest that they actually do have some idea. It may not be a really good idea, but they do have some sort of range. And from there, they can winnow down the range even more. Eventually they may get to some sort of figure that they can do something with, even if all they can do is roll their eyes and scream.

This little exercise can be done during the sales call, and help provide an answer to the "What's your budget" question.

Thus, if they don't know their budget, either walk away, or offer to help them determine what it might be. But you don't have to do it for free.

If they still refuse, then run away. They probably don't trust you and it does not bode well for a good relationship. Or they may have good intentions but are so ignorant of the process and unwilling to take your advice that they'll be nothing but headaches.

What is their timeframe?

Realize this about timeframes: They are almost never as ironclad as the customer makes them out to be. At the same time, *the customer almost never realizes this themselves*. I recall reading about a group of consultants brought in to finish a job, and they were given a drop-dead target date by which the system had to be up and running.

Upon starting the project, the consultants found that this deadline was the third such "non-negotiable, drop-dead deadline" for this particular project in the past year.

Most deadlines are arbitrarily imposed. Why? Because someone somewhere wants the project done before they go on vacation, because it's the end of the budgeting period, or because there's a big pow-wow in Las Vegas the following week and "wouldn't it be great if we could show this system to everyone there?"

Furthermore, most deadlines are usually unreasonable because someone who does not understand the scope or the complexity of the system imposed them. There are physical limits to the amount of work that can be done. Some people feel that by adding developers to a project, it can be done sooner. Generally, adding more developers to a late project only serves to make it later. See Frederick Brooks' *The Mythical Man-Month* for a complete discussion of this phenomenon.

It's not just the noodlehead in the corner office, however, who is to blame. Developers are part of the problem as well. How many times have you run into a situation where, each time previously, it's taken you a month to perform a given task, but you're still willing to bet the farm that "this time, it will be different." As a result, you'll estimate it will only take two and a half weeks. We're all optimists—assuming that for the first time in 37 projects, nothing will go wrong with the operating system, all of the tools will work as expected, no one on the team will quit in midstream, and the customer will never change their mind.

Nonetheless, you will have to deal with unreasonable expectations that are apparently cast in stone. The key is to manage and mold those expectations as early on in the project as possible—and the sales call is the best place to start.

I've found the best technique to use, upon hearing the target date for rollout, is to slap my knee and laugh loudly, hollering, "You want it *when*?" Well, okay, maybe I don't do this all that often. But you can help form a more reasonable expectation for delivery by playing detective about the initial request for a finish date. What is so important about this date—why does the project need to be done by then? Perhaps it's only part of the project that needs to be finished, or it has to be in good enough shape to be demonstrated, or an interface needs to be defined and complete, so that another project can be started.

I'll talk about dealing with deadlines in more detail later. The key is to find out what your prospect's expectations are now.

The sales pitch—you, dollars, time, and process

At some point, it's going to be a good idea to talk about who you are and what you do. You may have done this early on in the meeting, during introductions, but it's surprising how many potential customers want to jump right into the details of the project. I've found that if they're not ready to hear the "I Love Me" story early on, insisting that they listen to the whole spiel will either be boring or aggravating. And you don't want to bore or aggravate a potential customer in the first 10 minutes of a sales call.

Thus, another common time to tell them about you is once you've been able to hear about their company, needs, and project. During this whole time, you're doing rather little talking, simply asking pointed questions to show that you are interested and understand what they're talking about. It's tough to do sometimes, but be patient—you'll have your moment in the sun soon!

Naturally, you're going to want to spend the whole meeting along these lines, but, while it's fascinating material for you, the customer really doesn't care all that much. Sorry, but that's life in the big city.

You need to boil down your sales pitch into a couple of paragraphs, highlighting the key points that are going to be of interest to the folks on the other side of the desk. You may be impressed with yourself that you wrote a book on the newest snazzy development tool or that you just finished the install of an application that all of the leaders of the Free World use on a daily basis. And I probably would be too. But unless that accomplishment has specific bearing on the customer's situation, it's not worth more than about a sentence, primarily to illustrate that you've been around the block more than once or twice.

The big part of the "sales pitch" is what you do, more than what you say. Listening to them, asking the right questions, and taking care to make them feel comfortable that you can solve their problem—that's what's going to sell them. If you can show a little evidence along these lines, you'll be well off.

Once you're done with the personal sales pitch, you've established yourself as the expert (for the moment, anyway), and so while you've got their attention, take full advantage of it. Thus, it's time to move into the "Based on what you've told me and my experience with other applications, I'd recommend that…" part of the sales pitch.

There isn't a single way to go about this; it depends on the type of development process you're going to propose as well as what type of payment mechanism. I've found it usually ends up being one integrated discussion, all fitting under the heading of "Here is our recommendation."

For example, for a classic waterfall process application, you might say, "We recommend that we develop this application using a traditional waterfall development process. The first thing we'll do is create a complete Functional Specification. The price will be time and materials, and based on <list the factors here>, we estimate that it will take approximately <time> and <dollars>."

"Once we've finished the specification, we'll have a fixed price for the entire application. At this point, it's virtually impossible to provide an accurate estimate of the price for the entire application, because features and functions are likely going to change during the specification process. However, supposing that the information we have right now doesn't change, the application will cost approximately <dollars>."

This discussion is actually longer than the preceding example, but it gives you a rough idea of how the process and price and timeframe are all intertwined into one sales pitch.

I'll cover examples of how to word this discussion for other development processes in the sections of the book that cover those processes.

Decide what the next step is

This is done in two steps. First, you have to, internally, decide whether you want the project. If you do, then determine what the prospect would like to do. If it looks like they want to continue, I like to take out a fresh sheet of paper, write down "To Do" at the top, and then review what the prospect is expecting. This gives them yet another warm fuzzy that you will leave with a very clear idea of what they want, and helps them clarify in their own mind what they will be getting.

At the same time, here's your chance to commit them to action as well. During the meeting, they've undoubtedly come up with things that they're going to have to do research on, make decisions about, discuss with other people, and so forth. By writing down everyone's To Do list here, there's less of a chance of misunderstanding.

And I've found this helps me manage multiple projects better. It always seems really clear to me what the next step is when I leave a meeting. But by the time I get back to the office, unload the briefcase, yak with whoever is waiting for me, and sit down for work, it's easy to have forgotten half of it. And that's if I can get to it right away. If I've got another meeting or it's the end of the day, or something else pops up—who knows when I'll get back to it?

Some people even formalize this into a memo and will copy that document to everybody after the meeting. Whatever works for you and your situation.

Write it down, make sure everyone sees the same thing, and you'll be better off.

Your internal agenda

Now that I've covered the game plan as far as your prospect is concerned, I'll talk about you. You will want to get the answers to the following questions during the course of this initial meeting.

Who else are they talking to?

Believe it or not, you're probably not the only developer on the planet the customer is talking to. Better to find out who your competition is than to wander around blindly in a fog. You may wish to alter your message depending on whether your competition is the owner's nephew or a well-known consulting company.

Who is the champion?

A champion is the person at the firm who pushes the project along, ensuring that the rest of the company will buy in. This person should have fiscal responsibilities so they can authorize the funds. If that's not plausible, they should at least be knowledgeable about how much the company wants to spend on the project.

This person can also act as the fulcrum for decision-making about a variety of issues about the project, and act as the shepherd during development, testing, and installation.

Your job will be to make this champion look good. If you develop an adversarial relationship with the one person who is dedicated to waving the flag for the project, you might consider looking for another job.

If you don't have a champion, you don't have a prayer.

How serious are they?

While, hopefully, you've already determined this to some extent during the phone call, it's a waste of time to go on a sales call only to find out the prospect "was wondering if maybe we shouldn't start thinking about maybe looking at some other kind of system, sometime down the road. Maybe, that is."

However, yes, sometimes those tire-kickers do turn into business. One friend of mine has re-quoted an application annually for almost a decade—the original system was to be written in FoxBase in 1989—before it finally sold early in 1998—and was implemented in Visual FoxPro 6.0. The ongoing effort was worth it, though, because within three months, my friend sold three more installs of the same app to other divisions—all at the same price as the original!

But at the same time, you can't afford to keep updating a proposal for free every three months or two years. As I'm writing this book, I'm currently working with a prospect that has

requested to start development work four times since the sales call. After each request, I send out a new Engagement Letter (described in the next chapter), and it sits on someone's desk for a few months. Then they get all excited again, but something trivial has changed, like the name of the department head, and they ask for a new Engagement Letter. But they never get around to actually making anything happen.

As you'll see, once the sales call is over, the meter starts running. No matter what—the meter starts running on the next meeting. Imagine the amount of time, non-billable, I could have wasted with these people had I not held to that rule, and they started asking me to "come in for just one more meeting."

Sure, some of you may be saying, "But another meeting may be just what you need to get them over the hump and get them to commit to the project." I can see that point. My philosophy is that many of us are busy enough that we don't have to coddle prospects like that. When they're ready to buy, they'll buy. But I'm not going to waste my time on folks who aren't ready yet. If they need assistance, they should pay for it.

So how do you determine whether they're serious? One way to tell is by whether or not your contact is prepared. Are they organized with the materials they said they'd have on the phone? Do they get constant interruptions? Do they have to look up information, call other people for additional information, or hunt through stacks of paper, mumbling, "I know it's in here somewhere?" How many others in the company are involved? And how critical is this system to the company?

How involved do they want to be with you?

There isn't a right or wrong answer here. I can think of some customers with whom I worked hand-in-glove during the development of an application—and I wished every second that they would go away. And I can also think of other customers that signed the Engagement Letter, read the specification, and didn't call again until I had a final build ready for them, but that I sure would have liked to have worked with more closely. And there are those who seemed to get involved at just the right level.

The danger sign you are looking for is one of those "gut feelings." Do they seem to be too involved—that they want to micro-manage every step, second-guess your every decision? Remember, you're the expert. If they're not going to take your advice, then perhaps you should fire them as a customer. It's a tough situation when you normalize a set of tables to the 21st normal form because the DBA at the customer's site insists on it, and then complains about the poor performance of the finished app.

At the other end of the spectrum, it's their application. I once had a customer who spent well into six figures on an application, but hardly looked at the modules as they were being delivered—they simply ran a couple of tests to see whether something really ran and paid the invoices.

At final delivery, they realized that they really didn't understand how the system worked, and ended up spending another 25% for changes to the system and additional training for their personnel—expenses that could have been cut down significantly had they been involved throughout the development.

What's the decision-making process?

It's always nice to be able to deal with the decision-maker directly, instead of having to present a proposal and then wait two weeks while it makes its way up and down the chain of command, or worse, while it wends its way through committee meetings.

However, you're not always that fortunate. Sometimes the decision-maker is too far removed from the front lines. It may be that the company's structure requires decisions of a certain dollar amount or impact to be made at a certain level, but the implementation is done by people with their hands on the day-to-day workings.

Other times, the company will play the negotiation game for all it's worth. This is a particular sore spot of mine. Just like you, I'm a busy guy. I don't want my people to spend time with some doofus in a big office who wants to see if he can squeeze another 2 percent off the price, or if he can get an extra day of training for free. Software development is too inexact a process for someone to get out the calculators, trying to shave pennies here and there. And if he's just trying to turn each discussion into a confrontation where he has to win and I have to lose, then I'd rather not play.

In any of these cases, the issue is to make sure you know what has to happen. If you're dealing with the decision-maker directly, you have quite an advantage—you can find out what hot buttons they have and react accordingly in a direct and immediate manner.

If you have to wait on a faceless decision-maker ("the Regional Vice-President who only visits every other Tuesday"), then you need to do the best job you can, and present your case advantageously. Remember the question about the champion? If you have a good champion, they will help you win the job—telling you what works and what doesn't work at that company and with the decision-making tree in particular. Your job, again, is to make your champion look good.

And if you're going into an adversarial relationship (and I would recommend against it), then you need to keep your nose clean, your I's dotted and T's crossed, and generally hew to all of those other clichés.

If you don't understand what the decision-making process is, and who it involves, then at worst, you'll lose the job for the wrong reasons, and at best, you'll spend a lot more time and energy barking up the wrong trees until you get the job. Like I mentioned earlier, you don't want to drive for 80 minutes for a sales call only to find out you've got the wrong person, and you end up dropping off literature and picking up an RFP that could just as easily have been mailed.

At the meeting

Now that you've got an idea of what needs to be done, I'll walk you through a typical sales call and discuss some of the things that may happen.

Introductions

When you meet people, make an extra effort to remember their names. It's a social skill that isn't expected as often in this business. When you receive a business card, actually look at it—if even for a couple of seconds—and examine the person's name and title. Of course, be sure that you've brought yours along as well.

In the absence of a business card exchange, you can often get away with repeating the person's name if it's uncommon or sounds as if it's pronounced in an unusual way. As soon as

you can, write the person's name down—with the phonetic spelling. When I am in a meeting with more than two other people, I write down their names in a diagram that matches how they're seated, and then, if I have time, I add a physical attribute or two about them (blonde hair, looks like Herman Munster). That way, when I see them in another meeting but they have the audacity to sit in a different chair, I can call them by name again.

Get them on track and control the meeting

By providing an agenda and having an outline from their homework, you can often take charge of the meeting. Sometimes the prospect will expect to lead the meeting, but you can deflect that by speaking first, pulling out the agenda, and asking if they have anything to add.

It's a difficult situation when the prospect acts like the proverbial bull in a china shop and just charges into the meeting by showing you this great screen he "drew up last night and do you think it would be good to use for the main inquiry screen." Software development is a fascinating intellectual challenge, and many a user has been bitten by the bug. As a result, they become so involved in particular aspects of the project that they lose sight of the big picture—and this first sales call is dedicated to the big picture.

How do you wrest control from this type of person? Well, first ask yourself whether you really want to. In other words, are you going to be able to work with this person? You need to give them a few chances to take cues from you, but at some point, if they insist on running the meeting without agreeing to the agenda, then this relationship may not be for you. I know it probably wouldn't be for me.

At best, you'll spend more time than normal with this person, reeling them back in and getting back on track. That can be a considerable problem, both in terms of raw hours being built up as well as the continual distractions causing you to lose track and having to revisit issues. This person, by virtue of not wanting to listen to your expert advice, will pay more than they should have to. You'll want to consider that in your time and dollar estimates.

Existing functionality and new functionality

The majority of the meeting should be spent on two topics—evaluating their current application or system (it might be a manual process), and describing the major functionality desired in the new application.

Since the customer knows more about their company and application than you do, it's only natural to let them lead this process. However, you'll still have to keep them on track—nudging (or prodding, or forcing) them to move from one point to the next.

The trick is to keep them focused on the high-level requirements—much like the bullet points on the back of a package of commercial software. Once they start discussing specific functionality within a screen, they've gone too far, and it's time to move on to the next bullet point.

If they've done their homework properly, you should be able to move from one bullet point to the next consistently; more often, they've done a partial job, and you'll have to coach them along. It's important to remember that this list doesn't commit them to anything—you're simply documenting a starting point from which you can begin determining detailed requirements.

How much will this specification/application cost (in time and money)?

You'll almost never get through a sales call without having to deal with the question of "How much and when?" being posed by the prospect.

This is a tough issue—probably the toughest one in all of software development. The problem comes from a basic conflict—the prospect needs to know numbers so they can budget and plan. However, invariably, they have not defined what it is that they're going to buy well enough to get a price. It would be like expecting to get a price on a shopping cart full of food by simply looking at the items on the very top of the pile.

Given this basic dilemma, what can you do? Here's what I do—it seems to be the best compromise I've seen in nearly 20 years of building apps.

First, I explain that pricing anything requires a definition of what it is that the customer is buying. The more exact the price has to be, the more precise the definition of what it is they're going to buy must be.

Second, I explain that describing software, due to a number of factors, is very difficult and requires a number of iterations. What are these factors? First, software is abstract—it's not like a crate full of paper that you can touch and hold. Second, the tools used to create software are rapidly and continually changing, so we're in a constant state of R&D. And most businesses do not have the processes they wish to automate defined clearly. Finally, the iterative nature of software development lends itself to continual discovery—which by definition changes what is being developed.

The best that can be hoped for, then, is a series of estimates that get more and more accurate as the definition of the project becomes more and more detailed. First, I find out what the size and scope is, provide an estimate for the spec, and then a second, rougher estimate for the development is performed, based on the specification's size. As the spec changes, the estimate for the development is adjusted as well. When the spec is done, the development can be quoted accurately. If the spec isn't finished, then the development estimate is continually at risk.

This is like building a building. The first time you walk into an architect's office, they can probably give you some ballpark figures after asking you a dozen or two questions about the project. There are factors that can tilt this estimate one way or the other, but this initial estimate gives you a place to start. After each subsequent meeting with the architect, the price can be figured more accurately.

You can explain to your prospect that costs can vary for a couple of reasons. First, because the level of detail required to be exact is missing. Again, it's like a building. A skyscraper costs anywhere from $50 to $100 per square foot to build. It could be $200 if there are marble floors and working fireplaces in each office, or $25 if it's built in a depressed area with the absolute cheapest materials and a fair amount of finish left to the tenants. But these numbers don't mean anything if you don't know how many square feet the building will occupy. And these numbers still figure in a fairly tight range because building a structure is more of an engineering practice.

Remember, software development is still a lot of R&D. The tools are evolving, the tools break a lot, and there are few, if any, commonly accepted practices. Finally, the skills for a developer vary a great deal more than for the building trades, and thus the "cost per square foot" can vary greatly just because of a variance in the labor component.

The purpose of the homework, then, was to define the major items in the applications, so that I too could provide a ballpark estimate. This ballpark will become increasingly more accurate as I get into the nitty-gritty of the specification.

My company has developed many specifications over the years, and we have tracked our time to a fairly granular degree. As a result, we have enough history that we can now calculate how long it will take to develop an application and write the corresponding spec (and, thus, we know the cost) based on a list of screens, reports, and processes for the application.

For example, we have figured out that it costs, on average, $X per screen, $Y per report, and $Z per process. (I know that sounds very simplistic, but these numbers have been averaged out over a great many screens, reports, and processes—and they are, I stress to the customer, just averages that will get us into the ballpark.) We simply multiple those numbers by the number of screens, reports, and processes and provide an estimate for the development of the specification.

Naturally, the system will evolve from the initial guesses, but this is the starting point—it gives the customer the ballpark number they're looking for.

Then, as far as timeframe—you can figure out how many hours, roughly, are going to be required for the amount of money quoted, and you can then look at your company's schedule to estimate how you might fit that time into your workload.

The most critical point in all of this is to remember that whatever number you give the customer is the only number they'll ever remember. It doesn't matter how sincerely they promise "We won't hold you to this number." I can't count the number of times I've offered a "rough ballpark estimate" that suddenly became a fixed price at the next meeting.

This is why I insist on documenting even this ballpark estimate. Furthermore, I describe how it was arrived at, in writing, by listing every screen, report, and process that we discussed. That way, when the customer comes back four months later and wants to know why this initial $41,000 estimate has become more than $67,000 in invoices, I can point to what we initially started out with (14 screens, six reports, and two posting routines) and where we are currently (23 screens, nine reports, three posting routines, and a reconciliation process).

This process most closely aligns with the fixed price, waterfall methodology, but it's not limited to that. The ballpark number that you come up with may be enough to get the ball rolling, and the customer may want to begin developing the application through the Extreme Programming process next week. The final price won't be known until the project is over, but in that particular case, it's not necessary.

Since most people are still most comfortable with the fixed price, waterfall methodology, it's important to be explicit about how another pricing structure and development process is going to work, what the customer will get, and what their responsibilities are.

Come to closure and define the next step

At some point, you've either run out of agenda or you've run out of patience. Or perhaps the prospect has run out of time. At this point, it's time to determine whether or not you want to work with this customer.

If so, what I do is simply ask the customer what they want to do next. Although it's my decision to work with them, it's also their decision to work with me. If I want to, then it's up to them next.

If not, I explain that I don't think there's a good fit, for whatever reason, and terminate the meeting. I've found in most cases that the feeling is mutual—that if I don't want to (or can't) work with them, then most likely they're not going to want to either. Most likely you have a good business reason—you can't meet one or more of their expectations, either on delivery, cost, timeframe, expertise with tools, or expertise with their business or application. But there's often an undefineable sense of bad chemistry that both parties detect.

Even if you don't want to work with them because of a perceived personality conflict, you need a good business reason. You're liable to end up in court sooner or later if you turn down prospects because "I think you're kind of a jerk." I've found that simply saying "I don't feel we have the resources to do this project properly" works fine. Whether you want to explain those resources are time, expertise, or just personnel ("We don't have anyone on staff who could stand to work with you…") is up to you.

The next step

Supposing that you want to work with them, there are three possible avenues for the prospect to take.

Want an Engagement Letter for design

The optimal closing for the meeting will be a request by them for an Engagement Letter so that you can start performing work for them. If you're going to create a specification first, this Engagement Letter will include a ballpark estimate for the specification work, and, since they're now into the phase of spending money, they may have to get approval. If you're just going to start coding, it's twice as important to get them an Engagement Letter (some of you may know this as a "contract") that spells out the details of what services you're going perform.

It is a good idea to find out what hoops have to be jumped through at this point in order to get the Engagement Letter signed.

It can be aggravating to find out how little many people in larger companies know about the approval process for spending money. Over the years, I've had a surprisingly large number of people tell me with a straight face, "The money's already been approved" or "I can sign off on this" or "My boss said he'd sign it immediately." Then, of course, they find out the proposal has to go before some committee and it's going to be three months before I hear anything. Be skeptical.

Depending on the size of the application, there will likely be several layers of approval to go through before the application gets approved. There may be several layers of approval needed to even go ahead with the specification work. Your closing for the meeting should determine whether they want an Engagement Letter or if they have to get approval first.

Want additional consulting

In some cases, this sales call may have just opened up their eyes with regard to how much (well, how little) they know, and they might want you to come back, on a billable basis, just in order to help define their needs and to size and scope the system better.

In this case, you'll still need to produce an Engagement Letter—for the hourly consulting that they're requesting.

We'll get back to you

Yes, even though you may want to work with them, they may not want to, or be able to, turn around and say "Yes!" themselves. It could be that they're not interested, and are simply blowing you off in a nice fashion. Or perhaps they are interested, but aren't interested enough to make a commitment at the moment. Or perhaps the people you're meeting with don't have the authority to make a commitment themselves.

In any of these cases, play the part of Jimmy Stewart in "Harvey" and when they say, "We'll call you," directly ask, "When? Next Tuesday, about noon?" This way, you'll have a better idea of what to expect. You can also offer to follow up with them, but that decision depends on your personality.

The corporate IS department perspective

This whole chapter has been written from the point of view of an independent developer who is angling for the gig. What about the folks working in a corporate IS department, or the developer who wears multiple IS and related hats at a small company?

The big difference is that the corporate developer (I'll refer to both types of developers with the same term) usually doesn't get a chance to turn the potential customer down. That's not strictly true, and I'll discuss some options along those lines later in this section of the book, but for the time being, let's assume that you (the corporate developer) and your customer (other departments in your company) are stuck with each other.

Following the rest of this process is still a very good idea. Your goals are pretty much the same, although your internal motivation might be slightly different. As an independent, you're trying to avoid problems and make things go smoothly because you want to get paid. As a corporate developer, you generally have more latitude as far as the success of a project goes and still get paid.