# Chapter 4
# Choosing a Process

*"You must choose wisely."—Guardian of the Holy Grail*
*—Indiana Jones and the Last Crusade*

**Now that I've identified the types of projects and processes that make up the gamut of software development, it's time to discuss how to mix and match them. When you first talk to a customer, they probably don't know any of this. It's likely that all they know is that they want some software development work—of some sort—done. It's up to you to determine what type of project they've got, and to either educate them as to the available possibilities, or just choose yourself and present the best option to them. But you need to know what the options are first, and then you have to know how to present them to your customer. That's what I'm going to cover in this chapter.**

The purpose of this chapter is two-fold. First, it's to show you how to choose a development process—what things to consider, and what options you have. But that's the easy part and, frankly, one that you may not need that much help with.

The second purpose of this chapter is to help you deal with your customer during this process. In some situations, you'll choose yourself, and then simply make a proposal to the customer. Particularly if you've got a certified process, you may not be able to offer them any flexibility, because you would stand to lose your certification.

In other cases, you'll present several possibilities, and work with the customer to choose. In both of these situations, though, it can get tricky—not every customer has all that much common sense. It's easy for you to decide yourself—but not quite as easy to maintain the strength of your conviction when you're looking at a six-figure project, but the customer is being irrational or asking for unreasonable things.

In many cases, you should work hard to have your customer understand what your process is and why it benefits them. If you can show them this much organization, you will likely blow them away.

It's very easy for a customer to want to pick and choose pieces of various processes, as if they were ordering from a smorgasbord. You and I both know it doesn't work like that, but they don't, and, often, they don't care, either. So it can become a contentious meeting, and you have to be able to argue first, rationally, about why one thing will work and another will not, and then be able to stand your ground and just say No if they insist on an irrational decision.

I wish I had a nickel for every story where the development group got suckered into a fixed-price job without having a fixed spec. In most of the cases, the sales guy got the client to sign the contract, gave them a number of, say, $245,000, and the only details were a two-page memo describing a vague idea of an e-commerce or distributed sales processing system.

So that's what I'm going to do—provide you with ammunition if it comes down to a confrontation.

Before I begin, though, I need to mention that this chapter is not an exact flow chart where I ask a series of Yes/No questions and lead you to the magic answer. There is too much

variability and imprecision in software development, not to mention your varied experience level, which I cannot gauge, for that to work. Instead, what I can do is present and organize a number of considerations that you can use to help organize your own thoughts and make a better choice.

I also want to mention here that you don't want to play newbie teacher with this process. I'm referring to the brand-new teacher who reads the lesson the night before presenting it to the class. Read this entire book before undertaking this choosing process. If you choose based solely on what you've read so far, you're going to miss the details later on that you'll need to make the best choice. Read the whole book so you understand the whole process, not just the overview presented here. Then read this section a second time to get the choosing process fresh in your mind again.

## Type of project

Part of your role as a software developer, assuming you're part of the initial meetings with the customer, is to provide analysis of the customer's situation and then propose a solution. Since this book is about software development, it would be logical to assume that the fundamental solution is going to be a piece of software.

That may or may not be true, but determining that either a new or upgraded piece of software is indeed the answer is a fairly easy conclusion to reach. The next decision, though, is much more difficult, because the variables that come into play in making the decision are more complex, but also because explaining the choices to the customer is significantly more difficult.

I'll cover the initial contacts with the customer in the next few chapters, but first it's time to discuss how to do the analysis yourself so that you're prepared when you come into contact with the customer.

Generally, a customer doesn't really care about buying software. They care about solving a problem they're having, and figure that some sort of computer hardware/software combination is the way to do it. Given the pervasiveness of computers, they've probably given this some thought and so have an idea of the type of software they need. Furthermore, in that they've contacted you—a specialist in development software—it's likely that they understand that they're looking for custom software of some sort, but not necessarily.

Thus, the first question you'll have to ask is what they're looking for. Do they want you to write a new software application for them? Do they want you to modify, upgrade, convert, or replace an existing application? Do they want you to simply provide some sort of up-to-now unspecified services to solve a problem (or problems) they're currently having with their current software?

Closely aligned with this question are the dual issues of time and budget. You'll need to determine what type of time frame and what size budget they've got in mind at the same time that you're asking about the type of software development services they want.

A customer may not know the answers to these three questions—type of development, time span, and budget—or they may not be, particularly in the case of budget, willing to share them with you. I'll cover specific strategies for helping them determine these answers, or in the case of a reticent customer, getting this information out of them, in the chapters on The Initial Contact and The Sales Call.

For now, though, I'll assume that I've got at least rough ideas of the answers.

## Type of process

At this stage of the relationship, the table is usually wide open for possibilities, so it's the best time to provide the customer with the possible routes to take during this process. In order to do so, you need to know what the possibilities are up front, so that you can guide the customer appropriately.

As described in the previous chapter, you have basically four choices: structured development, some sort of RAD approach, XP, or simply "CLH." Which one you choose depends on your own abilities as well as the requirements of your customer.

## Your own abilities

Naturally, I mean "you" in the big-picture sense—if you're a sole practitioner, the first person singular is appropriate, but if you're part of a bigger shop, I'm referring to the whole group involved in the project.

You have to ask yourself: Do you have the ability to handle the type of project and perform the type of process involved? XP requires a two-developer team in front of the computer, while a waterfall process for a very large project requires significant experience in analysis and design.

## Customer requirements and attributes

What are the specific attributes of your customer? Each customer has specific priorities, requirements, and abilities that will help you narrow in on one possibility and remove others from contention.

### Do they have a fixed budget?

Believe it or not, not all customers have a fixed budget for a project. They may have a business need so pressing that the price, no matter what it is (within some sort of reason), is miniscule compared to the alternative, or that the upside is so large that the price pales.

For example, back in 1999, some companies found themselves behind the eight-ball as the end of the year approached due to software that was not Year 2000 compliant. For these folks, whether the project cost was $40,000 or $400,000 didn't really matter—the alternative was shutting down on January 1, and either losing more than that in sales each day, or even watching the company fold.

This can also be true if they must meet some sort of government compliance (taxation, emissions level tracking…) or adhere to legal requirements—they have little or no choice about what can be done and sometimes only slightly more choice over how it can be done.

Another example is more cheerful. You may be able to save a company significant amounts of money due to the automation of a specific function or task—compared to the $300,000 to $500,000 a year that the company will save, whether the cost of your application is $80,000 or $120,000 doesn't really matter a great deal.

### Do they have a fixed deadline?

As mentioned earlier, a looming deadline may be paramount. The deadline may be imposed from outside, and thus be non-negotiable, such as the Year 2000 target date was. On the other hand, it may be imposed from outside, but while not non-negotiable, still be a fixed deadline

that represents an opportunity. For example, an application that has to go live concurrently with the start of a sports season could potentially wait until next year, but the lost opportunities from missing a full year may make that no choice at all. The league certainly isn't going to postpone a sports season simply because some software programmer missed their deadline.

A deadline may also be imposed from within. A vice president wants the application done before he goes on vacation. The application has to be shown at the annual sales convention although it has nothing to do with the sales convention. The system has to be ready so it can integrate with the upgrade of the network planned six months from now. And so on.

Finally, the deadline may be reasonable, or not. In the case of an unreasonable deadline, the customer is usually loath to sacrifice any other part of the project, preferring the sacrifice to come from the developer's hide. As you well know, the sacrifice will eventually be made from the project, either in terms of features or quality.

Some deadlines are just silly; the result of an executive trying to throw his weight around. Other deadlines are important, but not life-threatening. Still others are critical. Thus, speed of development and deployment may be the first priority, ahead of budget, features, and quality.

## How critical is quality?

As much as every company would like to think that quality is "Job One," most often it takes a backseat to budget, deadlines, and other constraints. Applications that have known bugs can often be worked around through other mechanisms, and that may be a more efficient route in getting the application shipped. For instance, an order entry system may have an error in the calculation used to produce a subtotal when more than two credit memos show up on the same invoice. However, this particular situation may happen so rarely that the customer decides it's not worth spending a half-day or a day trying to track the problem down.

In some cases, though, quality is paramount, and other requirements are clearly secondary. Software for the space shuttle, as well as medical devices and other mission-critical applications—24-hour-a-day financial trading systems and online order entry systems are two that come to mind—requires a very high degree of reliability. It is likely that a sole programmer with a Code Like Hell approach wouldn't be successful with this type of system.

The point is that shipping is a feature. It can be beneficial to release software even if there are known issues because they are not so critical that they will do harm, and the other features the users will get are so beneficial that the software should be released, and then you work on resolving the problems in a subsequent release. For some applications, of course. Can you picture mission control radioing the Apollo spacecraft: "Neil, take your time on the moon. We aren't quite done coding the landing sequence. We'll upload a new BIOS to you when we're ready."

## How involved do they want to be?

Companies have different capabilities for wanting to be, and being able to be, involved in the development process. And these are two different things. A company may want to be intimately involved, but lack the capabilities, either know-how or resources or political acumen, to do so successfully. You've probably heard the line, when a developer is asked how long a task will take, "Two days. Or five if you stand behind me and second guess everything I do."

At the same time, you might decide that you want to have at the very least some minimal level of customer involvement, but if you have to legislate that, well, it's difficult to legislate an attitude.

Find out whether they want to be involved during the process, and, if so, what personnel they have available, and to what extent.

### How involved can they be?

Once a customer has indicated they want to be involved, and has proposed their personnel, find out the skill level of those people, and whether they are appropriate matches.

### How well defined is the project?

Some projects are extremely well defined, to the point that the major portion of the work is actually just writing the code and testing it. Many other projects, though, are still in the dream stage—where the customer has ideas of what they want, and a folder full of screenshots of other applications that do what they want ("except for this one thing here").

### Do they have an appreciation for software development? Do they buy in?

Some customers are fairly sophisticated in terms of software development—they've been through this before, or may even be veteran developers themselves. Others, on the other hand, want nothing to do with it—preferring that you pull a rabbit out of your hat at the end of the project. And still others go so far as to disdain the process, thinking it trivial or unimportant. (Ed Esber, president of Ashton-Tate, the folks responsible for dBASE, earned the enmity of all of his technical employees in the late 1980s when he declared that his programming staff was no more important to the company than the janitors.)

Beyond the question of whether or not you want to work with a company with this attitude, you should consider whether you *can* work with them.

## Considerations for choosing

As I mentioned earlier, there isn't a rigorous flow chart of how to choose which process; rather, I can offer some guidelines and considerations.

The first consideration is the company itself—the organization and culture. For example, a highly regimented company where all decisions are passed through committee upon committee may well freak out if you suggest an Agile Methodology, while a "dot-com" organization may easily choke on the structure and formal processes required with a waterfall process.

Consider how your customer runs their own business—specifically in terms of how they ensure quality in their own products. Do they have highly structured engineering and quality control departments, or do they shoot from the hip, accepting shoddy quality and the resulting ongoing loss of customers and requirements for replacing bad units simply as a cost of doing business?

### Structured development

Structured development is best when:

- The project is large and/or very complex.
- The customer has a high level of requirements already defined.
- They have a fixed budget.
- They have a fixed, but not rushed, time frame.
- They have a high need for quality.
- They do not want or need to be involved during the development.
- They do not have a high level of sophistication (note that the reverse is not true—a high level of sophistication does not preclude the use of structured development).

## Rapid Application Development

RAD is best when:

- The customer has a high level of requirements already defined, although this could be less than structured development.
- They do not have a fixed budget.
- They have a fixed, but not rushed, time frame.
- They have a high need for quality.
- They do want to and can be involved during the development.
- They have a high level of sophistication.

## Extreme Programming

XP is best when:

- The customer does not have a high level of requirements already defined.
- They do not have a fixed budget.
- They have a rushed time frame.
- They have a high need for quality.
- They do want to and can be involved during the development.
- They have a high level of sophistication or a high level of acceptance of iteration.

## Code Like Hell And See What Happens

CLHASWH is never best. However, it may be the only route to take given a customer's particular requirements. I'll discuss how to protect yourself from the inevitable problems that arise from using this type of process in later chapters.

## Conclusion

You'll need to read this whole book first before making the choice. This section is here to introduce you to the spectrum of development processes. You have to get more involved in each of them before you can advise a customer. You need to understand what you're getting into.