Hiring Virtual Workers Successfully!

Get Your Work Done Effectively

Richard Tong

Email

Recommended Resources

- Web Site Hosting Service
- Internet Marketing
- Affiliate Program

Table of Contents

Chapter 1

Outsourcing Software

You have a need.

If you didn't have a need, you wouldn't be considering a software development project. (Well, perhaps you just enjoy developing software and hoping that a need for it pops up sometime. That's a hobbyist approach to software, and there's nothing wrong with that, but it isn't how most people approach this kind of work.)

It seems like a simple process: you have your need, you describe the need to someone else, that person writes a software package that fills your need, everyone is happy. Unfortunately, it very rarely works that way!

You have a need—but you might not know what that need is yet.

It's ironic but true—just because you have a need doesn't mean that you accurately understand it. It is common for businesses to think they need one thing when in fact they need something very different. It is also common for a business to know that it has a need—usually a problem that needs to be solved—but to have no idea of how that need can be met.

Jones Metalworking has a computer-automated production process that automatically cranks out stainless-steel widgets. They also have a website that automatically takes orders online and produces a daily listing of what needs to be produced.

Jones Metalworking's management might think, "We need to hire someone to type the sales list into the production computer." Or they might think, "We need a software package that handles both orders and production in one integrated system." However, their actual need is for some type of "bridge" that will take the

sales list and connect it with the production process, so that everything that is ordered will be produced. This could be a person, or this could be software, or it could be something entirely different—but their actual need is for the connection of these two business processes, not for more staff or a different software package.

When considering your needs, try and drive your analysis down to the most abstract level. Figure out what your *business* need is. The odds are good that a software solution can meet that business need—but not always. Don't decide that software is the answer to your need until you accurately understand what your need really is.

But once you do determine that the way to meet your business need is either by changing some existing software or creating a new piece of software, you are on your way defining a software project.

What is a software project, anyway?

A software project can be anything from modifying an existing program to display a particular number in red rather than black, to creating a complete suite of integrated programs that interact with users, each other, and multiple types of hardware, networks and databases. The software itself may be used only occasionally by you or your employees, or it may be an integral part of your flagship product. These factors must be considered when deciding whether outsourcing is appropriate for your project.

Creating a New Software Solution—the Components of a Software Project

A software solution is more than a computer program, and a software development project is more than just hiring a programmer to write some code. When you are considering the use of outsourced resources to work on a software development project, you can choose to outsource the entire project, or any of the following

components. And you may want to choose one firm to take care of all of the components you outsource, or you may want to choose a different provider for each.

Specifications: A written description of what the project entails, this is the blueprint of the project. See Chapter 3 for more information about specifications.

Program Code: This is what most people think of when they think of software: the instructions, written in a language such a C++ or Java, which tell the computer what to do. The program code will usually include a user interface (some method in which the user interacts with the software), instructions to process or act upon the data provided by the user, and perhaps interfaces to other software or to a database.

Ancillary Programs: In order for software to be deployed to end users, it often requires an install/uninstall program, special customization/security modules, or components to allow communication with different types of hardware. These ancillary programs need to be defined, written and tested just as the mainline programs do.

Documentation: On-line help, user manuals, training guides—these can all be very important part of a software package.

Quality Assurance: Although you should expect programmers to check their code for errors, software should be put through a separate, rigorous test cycle to insure that it works as required (and as documented). Testing software is a complex and important part of the development process and should not be left as an afterthought. Chapter 6 covers this area in more detail.

Creating a New Version of Existing Software

Perhaps you have already created a wonderful software product that's popular and profitable in its original market. You now want to make another version of the product that would tap into a different market.

Create a customized version that targets a different type of customer. For example, a company that sells ticket management software to theaters and sports teams wishes to create a version that handles student enrollment for training companies. If your staff are not experts in the new target industry, you may want to bring in experts in that industry to help design the changes. In this case, you may want to outsource the creation of the requirements documents but have your staff handle the creation of the software itself—assuming that you didn't outsource the creation of the original product.

Port software to a different platform. Perhaps your software runs on Windows PCs and you want to have a version that runs on Macs or Linux machines. Or maybe you have traditionally supported Oracle and SQL Server but are getting customer requests to support MySQL. Rather than purchasing a bunch of new equipment and hiring staff who understand the target platforms (or sending your staff to be trained), you might want to outsource the creation of the alternate version(s) of your software to developers who already have the expertise and the necessary hardware and software environment.

Translate software into a different language. Creating a version of your software to sell in another country will almost always involve creating a new user interface and translating the documentation to handle language differences. This may be true even if you are targeting another English-speaking country, since the various "flavours" of English have different spelling and syntax issues. Depending on the type of software, other changes may also be necessary in order to comply with local privacy and data security requirements, or to be acceptable culturally or legally in the intended market. Localization projects are often very well suited to outsourcing, as it is usually more effective to hire a firm that is familiar with the language,

culture and operating parameters of the target country than it is to try to develop that expertise in-house.

Project Visibility

Another way to look at software projects is to consider how visible they are both within your company and to the outside world. Most software can be divided into three broad categories:

Internal software—such as accounting, payroll and inventory systems that are used by you or your employees to manage your business. You customers generally have no direct contact with this type of software. This type of software is often purchased outright (such as Quickbooks for small business accounting or Goldmine for customer contact management). If your company's needs aren't met by an off-the-shelf product, outsourcing the development of the software you want may make sense, because it's unlikely that your employees will have the expertise and time to do it internally.

Software used by/for customers—such as your web site, self-service kiosks in your store, on-line and over-the-phone sales and service systems. Again, your employees will likely know how to use these systems, but they may not know how to create or maintain them, so outsourcing may be sensible here. However, because this software is visible to your customers at least some of the time and because this type of software can be extremely "mission-critical", if you do outsource it, you will want to be very confident in the developer.

Software that is sold to your customers—such as a video game, a tax accounting package, or the CD component of a book package. If the software is a major component of your product, you may have the expertise in-house to develop it yourself—after all, that is the business you're in. If it's only a small piece of your product, however, you may want your staff to focus on the other parts and to

outsource the software portion. As with the previous category, this type of software is visible to your customers, so the margin for error is much less than with an internal project (most businesses can cope if their employees must work around an awkward software user interface or wait for an upgrade to an internal system, but asking customers to accept a low-quality or late product can lead to lost sales).

Project Size as a Determining Factor

If your software project is very small, and you have the appropriate resources in-house, outsourcing may not make sense. This can be especially true if you want an existing piece of software modified. By the time you've explained how the software works and what you want changed, you may be able to make the change yourself (or have an employee do so). On the other hand, a small, self-contained software project, such as a developing an Excel spreadsheet for your salespeople to use to create job quotes for customers, may be perfect for outsourcing—especially as the first project you assign to a new contractor. Large, complex projects are often well-suited for outsourcing, but definitely require a developer with sufficient resources and with experience in managing such jobs.

Chapter 2

Blueprinting

Developing software without a specification
is like building a boat in the middle of a raging river
--Software development proverb

Maybe you remember the old elementary school project "making a peanut butter and jelly sandwich." In this project, students are asked to create a set of instructions for making a peanut butter and jelly sandwich, and then the instructions are followed by another set of students. The results are sometimes hilarious, and often disastrous—not because of the inability to follow instructions, but because of the incomplete or vague instructions that they began with.

People who have been developing software for decades still wonder how to effectively articulate their needs and the end results that they want to see. Unfortunately, there is no magic bullet, no unbeatable approach that always produces great results. However, there are some tools and processes you can use to get the best possible articulation of your needs.

Interestingly, one way to get a good articulation of needs is to hire a software developer with good communication skills! A good software developer will ask questions, provide a series of releases of the software, and listen to the person who's buying the software and make adjustments as new information flows in. Providers and users may well know their needs intuitively, but they will often be at a loss when asked to articulate those needs clearly and specifically. Good software developers know this and make adjustments for it.

Some very small projects can have their needs articulated verbally. If you want an Excel macro that will copy data from one spreadsheet to another, it isn't generally necessary to go through all of the steps we are about to outline. Projects that are

more complex than that—particularly when combined with the vagaries of outsourcing the development process to someone who isn't working just down the hall—need a more formal approach.

Just as builders have developed standards for blueprints, years of software development processes have stimulated the evolution of an entire set of documents for articulating software development needs. We will discuss those documents, and provide some tips on how to create them most effectively.

The classic software development process uses three key documents. These documents are:

- The Vision Statement
- The Requirements List
- The Product Specification

The vision statement is a brief summation of what business need the software will meet, or what problem(s) the software will solve or mitigate. The requirements list is a list of what the software package or application needs to do in order to fulfill the vision statement. The product specification describes exactly how it does these things. It may seem like there is a bit of bureaucratic overkill going on, but dividing up a project conceptually in this fashion can increase the clarity of the communications between the various elements of a software development organization.

We will briefly discuss each of these informational documents, and then talk about how you can use this organizational model to create your own software.

The Vision Statement

The first step in defining your needs for a new piece of software is the vision statement. This document ensures that the business professional or entrepreneur

(that's you) and the software developer are in agreement as to the purpose of the software program and what it is designed to do.

If a software developer creates a program that is in opposition to the core priorities of the company that has contracted to build the application, then resources will be wasted correcting the problem after the fact. Far better to avoid those problems in the beginning by getting on the same page about why you're doing what you're doing

Start with a clear articulation of the business problem you are solving, or what business process you are improving. Define how the software product you are building will solve or fix the problem. Establish clear metrics for measuring the success of the software product in helping with the problem.

Your vision statement does not need to be a lengthy document, but it does need to be clear. It might take two lines to define success for your product, or it might take fifteen pages—the important thing is that the statement clearly set the course for your product so that everyone concerned knows what you are doing and why.

The vision document details the goals, the approach, and other necessary details. First, write the scope of the project clearly and state the boundaries explicitly. Be sure to determine what the project will and will not offer. The more specific you are, the more accurately the development process will adhere to the end users' needs.

When the vision statement is complete, it should be reviewed and accepted by key users before work on the requirements and specifications documents begins.

The Requirements Document

The second step in articulating your needs for the software product is the requirements document. A requirements document describes what the product does to fulfill the goals of the vision statement.

In the requirements document, you describe business process that will be in place when the software is operational and how the software will be used. The requirements document will, as its name implies, list all the requirements that are to be met by the software. This includes how the software interacts with the user, how it processes data, how it interfaces with other software or with hardware, and what security restrictions it must adhere to.

This document should also include any performance criteria that the software must meet. For instance, if you are designing software to track customer orders, you may wish to include criteria such as "System must have capacity to process 5000 orders per hour" or "Retrieval and display of order record must take no more than 1.5 seconds."

Similarly, if the software must run on a particular hardware platform or operating system, or must interface with a particular version of another piece of software, specify that in the requirements document.

When you are writing a requirements document for a project that will be outsourced, remember that you are establishing the basis of your contract with the developer. If the developer creates software that fulfills all the criteria as described in the requirements document, then the project will be considered a success. Therefore, you want the requirements document to be specific, precise, and clear.

As with the vision statement, the requirements document should be read and approved by the people who will be using the software. If you are creating software as a product to sell, then the requirements document should be reviewed and approved your salespeople, your customer service staff (who can be an unrecognized source of knowledge of what your customers like and dislike about your products), and your marketing people.

Note that the process of creating a requirements document for a complex software project is not a trivial task. You and your staff are likely to come up with more ideas for software features than are practical or cost-effective to include. And a "must-have" feature to one person or one department may be insignificant or even disliked by another. Deciding on the final list of features to include will often involve negotiation and compromise.

The Product Specification

Once the requirements document is completed and approved, it's time to get to the nitty-gritty of defining exactly what the software will do. This is done in the specification document.

A specification is like a written blueprint for a software application. It shows what the application does, how it does it, and what it looks like while it does it. A properly written specification is an incredibly useful tool that will let your software designer achieve tremendous results with modest investments of time. A badly written specification is like putting a bullet through the brain of your product before you even start. It may seem harsh, but it's quite true: if you want to doom your product right now, it's as simple as writing a lousy specification.

What makes a good product specification? Three things: completeness, clarity, and coherence.

Completeness means that your specification completely describes the product. The specification explicitly lists all core assumptions. The project's functions are all enumerated and described. The user interface is prototyped and annotated to indicate how program functions are invoked and how the software communicates with the user. Where and how the product will interface with other software components is described.

Clarity means that the specification is not ambiguous or vague. It accurately describes what each function does and clearly establishes what is and is not done by any particular combination of user inputs.

Coherence means that the specification makes logical sense. It conforms to the commercially available world of computer technology. It does not contradict itself.

The specification is the document that your software developer will refer to on an ongoing and regular basis while he or she is coding your application. It is critical that the specification be understood by the developer—a specification which isn't understood can lead to wasted time and wasted money. For that reason, it is a good idea to have extensive talks with the developer concerning the specification, to ensure that everyone is on the same page and knows what the product is supposed to be when it is done.

Creating these documents generate several benefits. First, the end users of the software—possibly even you—can see early in the process if their needs will be met. Secondly, the developers can better estimate the effort involved in creating the application, and have a basis for a project plan. Third, the quality assurance expert has a basis for testing the application. Finally, the documenter can use the specification document to begin writing user manuals and on-line help while the programming is being done.

It may be true that in some cases you can get away with not having a formal requirements document and/or specification. If your application is small and will be

developed by a single individual and the user population consists of only a few people or a single workgroup, then it might be possible to achieve a common understanding of what the application will do without committing the knowledge to paper. In other cases, you will probably need a written document.

The general rule of thumb is that if it will take more than about a calendar month from conceiving the project to putting in into production, if more than a single person will work on the application, or if more than one workgroup will use it, you must have a requirements document.

However, if you are outsourcing the development, then the specification documents form the basis of your contract—they define what the developer is promising to build and what you are agreeing to pay for. Even if your outside resource is your college buddy Joe who understands your ideas and your business almost as well as you do, you should have at least an informal specification in writing. Proceeding without these documents is very risky indeed.

We will now look more in-depth at writing these documents.

First, a piece of good news—you don't actually have to write the requirements list and specification as two separate documents!

In a formal software development process, used by a company like Microsoft or Oracle, these documents are separate pieces of work, generally created by different people, and with different functions.

For 90% of software projects—and for 99.9% of projects done by individual entrepreneurs or small companies—these documents can be merged into one piece. You don't need one document to show the marketing people and another document to show the product manager and the developer—you are all of those people. So, rather than make you jump through a lot of hoops to emulate a process that is well-suited for a team of 200 working on the next great operating system, we will

lay out a streamlined planning document and show you how it will help you and the developer you hire to create a really tremendous software product. For purposes of the rest of this book, we will refer to this merged, streamlined document as the requirements document, or just "the requirements".

(You do still need to create a vision statement, though—it's really important in clarifying your own thinking.)

Sectioning the Requirements Document

The requirements document should be partitioned into several sections for organizational purposes. If your application interacts with an older version of a system (a "legacy" version), then you may want to put the legacy integration requirements into a separate document because they tend to be lengthy, very technical, and may not be specifically related to the rest of the document or requirements. If your application requires elaborate security or auditing features you may want to pull those requirements into a separate document. User Interface and external application interfaces are other examples of potential separate requirements documents.

Here is a short list of sections that you may want to include in your document. These are only suggestions, and you may want to add sections or remove them, but the more specific the sections are, the better. Whatever the case, any requirements document should contain two parts, an overview and a description of the system's functionality.

Application Overview

This section discusses the application in broad, general terms. This is a synopsis of the application's purpose and the reason for its development. How will it fit into

your business? Does it relate to other systems you or the industry is using? In what way?

State the objectives of the project and determine what you and your business hope to gain from the development of the software. Without clear objectives your project has little or no chance of success. Work diligently on this section, and ensure that it is clear and all-encompassing. You may want to think about what business objectives you hope to achieve. A few may include reduction of costs, improving customer service, simplifying workflow, or replacing obsolete technology. Also, make sure you understand exactly how the proposed project will help accomplish the stated objective. Also, try to imagine who will benefit from this project. Do the people who will benefit from it consider it the most important improvement that can be made at this time? Should you be devoting resources to a different project instead?

How Business Will Be Served

Here, you should describe the business process and the ways in which your application will be used in that process. In some cases you will need two sections—one describing the existing business process using existing systems and the other describing the future business process using the system you are having created by your developer. This happens any time the business process will change once your system is introduced. When this is the case, the purpose of describing the existing business process is to have a basis of reference to explain the new business process.

If the business process won't change when your application is introduced you should be able to describe it in a single section. However, in this case be sure you understand and communicate what value your application brings to users.

User Roles and Responsibilities

In this section, describe who the users are and how the system fits into the business context. List all users for your system in terms of user roles. User roles are related to meeting a specific business objective. When gathering requirements it's useful to consider roles because you will want to focus only on business objectives that are relevant to the application. For every role you need to list the tasks that pertain to the use of your system. You also need to describe the relationships between the tasks for each user role and the hand-offs from one role to another. This can be represented as a workflow diagram. When working with workflow diagrams, consider time periods in describing actions. Not all tasks will be carried out every day. Some may be daily, but others might be weekly or monthly, and this will change the workflow.

Application Architecture

In this section, describe how the various components of the product work together. This is usually done with a diagram that shows the various layers, or tiers, of the application and how they interface. For instance, you might show the user interface as one layer, a processing layer beneath that, with a security module underneath it to protect access to a database layer and more processing modules below.

The Functionality Statement

Simply put, you should state precisely what the application will accomplish. This part, more than any other section, details the contract you have with your users. All the functions included in the application will be listed here. If it isn't here, it won't exist in the software. Because this document really does act as a contract, think carefully about what to include and what to exclude. You must use precise language in this section because your developer will use it as the guiding document in coding the application. When reviewing this section with other people, pay a great deal of attention to removing any possible ambiguities or incorrect interpretations of any of the requirements.

This section will consist of many, many subsections. Each of these will be devoted to a single area of functionality. There are several guiding ideas that you can use to decide how to structure this section. If your application has distinct categories of users, for example, you can list requirements by user category. User categories might be delineated by (1) the frequency with which they will use the system, such as heavy users or infrequent users; (2) the purpose for which they will use the system, such as operational decisions or strategic decisions; or (3) the level of expertise of the users, i.e., power-users vs. basic users. So long as each category of user utilizes the system in its own way, structuring the requirements based on user category will make sense.

If your application deals with several kinds of objects, you may wish instead to list the requirements by object. For a reservation system, a booking is an important object and you may want to list all requirements pertaining to bookings in one section. This method of delineation will increase the readability and organization for the developer.

Of course, the most common method of organization is to list the subsections by feature. Organization by application feature makes the coding and development of an application much easier, as each of the features are linked in the code. Features of a word processing application, for example, are file management, formatting, and editing.

Here are a few subsections that might be helpful to you. Each of the following paragraphs describes a subsection.

Administration or customization of the application may need to be addressed. Here, you might list any requirements related to the user's ability to modify the application or to perform any administrative functions, like creating new user IDs.

An auditing section would list any requirements for auditing features. Auditing commonly means being able to tell who made changes, what they were, and when they were made. Understand precisely what your business thinks "auditing features" are before defining them here.

Reporting requirements specify what information is presented in the reports, how frequently the reports are produced, and to whom they are distributed. A reporting section is particularly useful in the administrative maintenance of your application. They do not specify what the reports look like because that is a design issue, but they do specify all of the information contained within the reports.

Security is a profoundly important issue, and a subsection detailing the elements of your security protocols is important to say the least. This section might contain information pertaining to who is authorized to use each part of your application. Be sure you understand what the real security concerns are. If the security requirements are very complex you may need to write a separate document describing them.

No matter how you organize the functions, you need to be sure that you describe exactly what the function does. Specify the source and type of input the function requires, what processing it should perform, how it should handle errors, and what outputs it should produce. For example, if the function is "Display customer transaction history", a starting point for a functional description might be:

Input:	Customer ID from user interface
Processing:	Look up customer master record. If record does not exist, display error message and allow user to enter another ID. If record is found, display customer ID and name, then select all records in transaction history file for the customer ID. If no records exist, display "No transactions for this customer"; otherwise, display transaction date, description and amount for each transaction.

Outputs: Transaction history screen

Creating mockups of entry forms and outputs can be helpful if you have specific requirements for their appearance. When designing user interface components, it's often a good idea to assume that the end user will have little computer knowledge and may not read a user manual or even the on-line help. Some common rules-of-thumb include:

- The program should act in a way that least surprises the user.
- It should always be evident to the user what can be done next and how to exit.
- The program shouldn't let the users do something possibly damaging without warning them.

Scope

In this section, state what functionality will be delivered in which phase. Include this section if your development will consist of multiple phases, or, as an alternative to this section, you might note the planned project phase for each feature in the functionality statement section.

Usually it is best to include a separate scope section for easy communication and reference. Whether or not you use a separate scope section, you may want to prepare a section or a separate document that lists in detail what is not in the scope of the initial release. You may want to write a section like this for all subsequent releases of your application.

Performance

In the Performance section, describe any specific performance requirements. Be specific and use numeric measures of performance. Don't simply state that an application should be "fast" or open files "quickly." These are not specifics and the

cannot be verified in the final product. Programmers thrive on specifics. Instead, state that opening a file should take less than three seconds for 90 percent of the files, and less than 10 seconds for every file. This is a requirement that can be verified.

If you like, you can include performance specifics with each function in the document, rather than including a separate section for the performance specs.

Usability

In this section you describe any specific usability requirements. Include this section only if you have any "overarching" usability goals and considerations. As in the Performance section, use numeric measures of usability whenever possible. For instance, you may want to specify that it should take no more than five clicks for a user to navigate to any page in your website, or that all text on entry screens must be at least 10-point.

Appendices

Finally, consider using appendices to contain any information that can't fit anywhere else but which is too important or relevant to leave out. Supporting and background information are good examples of material that doesn't fit in the flow of the document, but is important to understanding the broader context of the application and its development.

Some of the supporting information might be in the form of graphs. Charts and diagrams can all be included in the appendices. In certain cases you find yourself explaining some part or another in more technical detail than the surrounding material. This part can become its own appendix.

A blank template for your requirements document is found in Appendix B.

Getting People Who Need to Read the Document to Actually Read It

Writing the requirements document is an important step, but it's important to be sure that all related parties read the document you've worked so hard to create. If any of the involved parties fail to have a complete and meaningful understanding of the requirements, disaster may ensue. When trying to get important parties to read the document, think of short attention spans and busy schedules. Use summaries, presentations, outlines, and walkthroughs.

Summaries allow you to get across the "bullet points" without reciting the entire document to a busy associate. It's also important that a summary will result in a document with far fewer pages, increasing the likelihood that they will read it. Presentations can be another way to inform a group of relevant people about the specifics of a project. The benefits of this method are that you can inform a large group at once, and that you can be assured of your audience's attention for the length of the presentation. And of course, document walkthroughs, which can be as simple as reviewing the document during a lunchtime meeting, can also be used to make sure everyone is on the same page.

Writing Tips

Often overlooked in technical writing, the "active voice" helps to make the document more readable and immediate. "The code will create a database" is clearer than "a database will be created by the code."

Use lists, and where possible, shorten the length of sentences and paragraphs. Readers like to see a great deal of white space on a page. The more space there is, the quicker the pace, and the easier it will be to maintain the reader's focus and attention.

A few other "common sense" approaches to good technical writing include:

Organization: Be sure that each section of the document follows the previous section in an organized way. A logical order is essential to the ability of a reader to follow your document. Include a table of contents and an index for increased readability. This will allow the reader to navigate the document more easily.

Vocabulary: Use understandable vocabulary. This document is for wider consumption than simply the developer. Be sure that the document isn't full of computer jargon.

Visuals: Don't be afraid to use illustrations and charts to illustrate points and certain functionalities that might be easier to diagram than to explain.

Proofread: Be sure to proofread the document, using spell-check and even another set of eyes. The more proofreaders, the better.

Reflecting Changes in Requirements

In a project under proper management, there comes a time when you must sign off on the requirements. Despite this, requirements can, and probably will, continue to change after the final sign-off. In fact, it isn't unusual for the software to continue to change up to the moment of system acceptance testing or even up to the system release. Nevertheless, the requirements document continues to be the guiding contract between you and the developer. Because of this, it has to reflect the changes.

You must continue to update the document as the requirements change. As you update the document, either before or after sign-off, you need to keep track of how it changes over time. Recording changes through a system of document version numbers will help you understand the business and the users' needs more

completely. Recording the changes documents each change for everyone, ensuring complete transparency.

Use your discretion in deciding how detailed the version tracking should be. At a minimum, keep an archive of the versions of the document that were distributed to a wide audience. In addition, maintain a revision history for major requirements. If the requirements change between the first version of the project and the sign-off, consider including a separate section that describes the major changes and reasons for those changes.

Documenting Requests for Enhancements

After the changes are frozen, usually after the code has been solidified in such a way that returning to development would be costly, it's important to maintain a record of changes that are requested by users so that they might be included in further versions of the software.

Outsourcing Your Requirements Document and Materials

Just as you can outsource the software development project, you can also outsource the creation of your requirements documents. Although at first glance this may seem like a foolish idea, it can in fact be a smart move. It is like hiring an architect to create the blueprints of the house you want built. You may have some ideas about what you want in a house, but not have the expertise to translate those ideas into specific plans that will meet building codes and that will direct a contractor in how to construct it.

Similarly, you may understand your business need without being able to describe that need in a way that helps developers create software efficiently. Some companies specialize in the specification process. Helen Tang of iConclude offers

this relating to assisting business professionals and entrepreneurs with defining the software user's needs:

"We're an innovator in the emerging space of automated incident resolution and problem management," explains Tang. "We ship a platform product called iConclude Repair System, which ships with out-of-box expertise packaged up into Repair Pack modules."

The company has been running a pilot project using Amazon's Mechanical Turk in trying to identify Information Technology professionals skilled at troubleshooting and "contract" with them to capture their expertise in content add-on modules for iConclude's automation platform.

iConclude's response to the need for this service is very specific to the situation. Based on the customer population, the company looks across common infrastructure environments. They collect this information with an array of techniques, including surveys, beta programs, customer advisory board meetings and even personal meetings.

Customers often define their needs in a statement of work, and some companies create appropriate system requirements from those definitions. If the customer only has general ideas but nothing specific, analysts will go onsite to collect and define the software users' needs, based on interviewing, focus groups, surveys and prototypes, user instruction and shadowing. This is a method that is especially effective when it is a redevelopment project with existing systems and user documentation available to define user needs. Without an onsite presence, the information-gathering process can take more time, and require more input from the customer.

Chapter 3

The Best Outsourcer

Nine women and one month do not make one baby

--Project staffing proverb

Once you've defined the project, the next step is to find a resource to implement it. Depending on the size and complexity of your project, you may need one person or hundreds. Regardless of how many people you need, there are two basic ways to engage outsource staff: contract with each person individually, or engage the services of a firm that takes care of all staffing issues and simply delivers to you a final product.

Review Qualifications

A freelance developer should be able to provide you with a current résumé that lists their areas of expertise and training. As with applicants for employment, résumés can be used to screen out candidates who do not have the skills or experience your require. Listing a skill on a résumé, however, does not necessarily mean that the candidate is good at that skill.

Recognizing good developers is not easy because good programming depends on more than knowledge of programming language and syntax. You need a developer who is creative so that they can find innovative ways to address your requirements and come up with solutions to problems that could stand in the way of meeting production schedule and deadlines. Your developer should also be humble enough to accept suggestions, but able to assert themselves and provide leadership in certain situations.

Rather than simply looking for expertise and experience in the exact area in which the developer will work, it is often better to look for general software development talent and ability. Instead of looking for a developer who has a list of specific skills

and credentials, a general knowledge of computing and computer programming may serve you more completely than a developer with singular knowledge in C++ or a designer who only knows Photoshop CS. This will help your developer to come at problems from other points of view and be the innovative thinker that you need to complete your project.

One way to identify qualified developers at such sites as VWorker, Guru, and Elance is by utilizing third-party tests such as Brain Bench and ProveIT. Another is to request examples of previous work they've done. You can also have the candidates take a test, submit a proposed solution to a hypothetical problem, or explain and critique a sample of your current code (or documentation if you are looking for a technical writer). This gives you a sense of their skills, and also lets the developer know what they can expect from your project. Do note that the test problem or code sample should be representative of the type of work they'll be doing, but it should not be an onerous task. Unscrupulous companies sometimes try to get free work done by presenting a real task as a "test"; professional freelancers are quite wary when presented with requests for lengthy or in-depth sample work.

Conduct Interviews

When you've determined that a candidate has the credentials needed for the job, it's time to interview them so you can get to know them on a more personal level. Beyond simple, "yes or no" questions, open-ended interview questions that give you an insight into the developer's personality will help you to know a broad range of important facts about your potential developer's work habits and skill bases. As an example, ask your developer to critique a system or platform that you both have in common, preferably something he or she will use for your development project. For example, ask the developer something open-ended that will require a creative response such as "What aspects of Java don't you like? Why?"

Another creative approach is to ask a developer to create an object model of a chicken. This eliminates uncertainties about the problem domain, and because everyone knows what a chicken is, it also jars developers away from the technical details of a computer. It tests to see if they are capable of thinking about the big picture.

If you are looking for people other than coders to work on the project, such as technical writers or testers, tailor your questions to their areas. You may want to ask how they would organize a training manual, for instance, or how they like to generate test data.

Ask candidates about how they're keeping up with changes in their field. It's important to see if a developer is self-motivated or concerned about improving their own skills. As important as technical ability, or perhaps more important, is a creative and constructive personality.

Always ask about prior work experience and determine the sorts of things that a developer has done. By discovering what a developer is excited about technically, you can learn a lot of important things about him or her. Ask the developer to describe a project that was especially interesting, or that was challenging and successful. Ask what the developer has done that he or she is most proud of. This usually reveals the depth of one's understanding and mastery.

During the interview, you should also ask questions about how the contractor runs their business. Will they be working on other jobs at the same time as they are working on yours? How much time will they be able to devote to your project? What type of warrantee do they offer on their work? How much will they charge for work that is beyond the scope of the original project?

This is also the point where you need to work out details of how the contractor will be paid. Will you pay them on a time and materials basis, or will they provide you with a fixed bid, and if so, at what points are payments to be released? A common

arrangement is to pay a portion (often 1/3 to 1/2) at the time a contract is signed, another portion when the product is delivered, and the final payment when you've reviewed and accepted the work. For large or long-term projects, however, more frequent payments may be appropriate. If your project is time-sensitive, consider offering a bonus for early delivery.

Also, it is important to be clear and decisive on two important issues. First, ensure that you retain the right to all owned intellectual property. Second, all software created by the developer which is paid by you should be retained by your company. This should include software that may not have been directly requested by you but was needed to complete the contract.

Check References

The best criteria for selecting a qualified developer is reputation. Developers who have been successful in the past are likely to be successful in the future. It's like building a house. You want to hire people who have done it before, and who have a good reputation. Hiring someone who hasn't done it much just because they are cheap is probably a mistake.

Be sure to verify the credentials of the developer. How long has the developer been in the outsourcing business? How has the developer managed successes and failure? Are you confident that the developer can manage both the size and scope of your outsourced work?

Also, check to see if your developer has a direct connection with your competitors. If the developer does, you may need to consider whether you'll be confident that your software and secrets are secure. You need to be sure that they have procedures and policies in place to ensure that your data and strategic information remains private and secure.

Choosing a Service Provider

Instead of interviewing and selecting individual software developers yourself, you may want to engage a firm to take on your project, where the firm is responsible for all staffing decisions. For projects that are large or complex, this can be an effective and efficient approach. It is similar to hiring a general contractor to be responsible for building your house. You and the contractor agree on the plan for the house, the completion date, and the cost. The contractor then takes care of hiring people to do the plumbing and wiring and all the other jobs needed to complete the project.

As with hiring a building contractor, there are factors to keep in mind when hiring a firm to handle your software development project. The firm should have experience managing projects of the type and size of your project. They should be able to provide you with testimonials from satisfied customers. And they should have a good reputation in the industry—not only with their clients, but also with their staff and with their subcontractors.

You may choose one firm to handle the entire project—designing, programming, testing and documenting. The advantage to doing so is that it simplifies communication and coordination. You may be able to have one person in your organization (possibly you) responsible for all contact with the project manager at the outsourcing firm. This reduces the chance of conflicting instructions being given to developers and also means that you will not need to worry about coordinating changes in schedules—if a section of code takes longer than anticipated to write, for instance, you don't have to calculate the effects it will have on testing and documentation.

There are some potential drawbacks to having one firm handle the whole project, however. Some firms may not have all of the types of resources you need, or at least not at the quality you want. If you select an offshore development firm, for

instance, they may be to create and test the software to your standards. But they may not be able to produce documentation in well-written, idiomatic English.

Another point to consider, especially if you decide to go with a relatively new or small firm, is that you may want the added assurance of having independent resources checking the quality of each other's work. Human nature being what it is, Firm B is more likely to find and report on flaws in software coded by Firm A or documentation written by Firm C, than they would if the work was done by their own staff. Also, it is often easier for outsiders to notice things that are missing or wrong because they aren't subject to group-think or shared false assumptions.

The most convenient and flexible place to find software developers (and a wide variety of other professionals) are the online marketplaces such as Elance.com and VWorker.com, which are discussed further in Chapter 5.

Chapter 4

The Best Outsourcing Sites

Twenty years ago, if you wanted to find a software developer with a certain set of skills, you resigned yourself to a long and tedious process of placing classified ads in newspapers and personal networking. Those methods still have a place, but in the Internet age it has become far easier to find temporary technical assistance. If you need someone to write a web page, code a software module, or write a technical manual, all you need is a web browser, your checkbook and the ability to manage the project.

There are dozens of web sites that can help you find software developers for your project. This chapter will give general principles for using these sites, as well as some details on three of the largest and most useful services: Elance, Guru, and VWorker.

All three of these sites, as well as most of the smaller sites out there, follow the same basic conceptual model. A buyer (that's you) writes a description of his or her project and posts it to the site, along with information on the project scope, timetable, budget, etc. Interested service providers review the project and, if they are interested in working on it, submit bids for the job. If the buyer finds a bid that he or she can live with, they accept it and directly communicate with the service provider. Sometimes these contractual relationships are "one night stands" and the parties never again work together; in other cases, working relationships that span years are forged. The service provider generally pays all fees to the web site brokering the transaction. The web site also offers side services such as escrow, project tracking, and feedback mechanisms.

General Principles For Using Online Services

Always read the feedback.

All outsourcing websites provide feedback reporting on the service providers, the buyers, or both. This feedback is the most reliable and objective measure you have at your disposal, and you should take feedback very seriously. (Service providers will look at your feedback, too, so consider the possible long-term costs of acquiring a reputation for being difficult to work for.)

When in the rush to get a quick or small project done, you may neglect to look at the feedback ratings that service providers have compiled. Especially when you've got a field of hundreds to pick from, this can be a daunting task. Avoid the temptation to slack on this one, though – this is your best source of information about the people you're about to hire. You can save some time in reviewing feedback by first disqualifying bids whose characteristics are such that no feedback rating would change your mind.

Some warning signs to look for in a service provider's feedback history:

A long history with few ratings. Service providers who work for a long period of time but acquire only a few ratings should raise an eyebrow. Are they only providing this job on an intermittent basis (in which case you should be concerned about whether they are motivated to do a good job for you over the long haul)? Are they flubbing a lot of jobs and negotiating a no-feedback resolution with their other employers? Just what is going on? This isn't an automatic disqualifier, but it is a flag that should prompt some questions from you for the service provider.

A sudden change in the rating trend. A provider with two years of great reviews and then three bad reviews in a row is a provider who has stopped working at the job,

is having some terrible problem, or is in the midst of some other scenario which you want no part of. Steer clear.

Persistent complaints about a particular work area. Some providers simply have weaknesses in a particular area. They may have trouble accurately forecasting deadlines, or persistently show a poor comprehension of client instructions. The feedback reviews will often show this area of weakness across a number of reviews. You can decide whether the problem is one that greatly undermines the candidate for you; if you don't like to be pestered every day with provider questions, then complaints that a provider takes a lone-wolf approach won't bother you.

Know what you need before you go to the store.

Each online service lets you write a brief, or lengthy, project description when you post a project. This description is the primary source of information about your project for the people who will be bidding on it.

Do you want projects that fail, service providers who don't actually have the skills they need for your project, broken budgets, and timetables that become a joke? Then head on off to Elance right now and post a vague project description, and hire the first equally vague response you get.

If, on the other hand, you'd like things to work smoothly and result in a usable or salable product, then consider the importance of knowing what you want before you start shopping. Remember, the people bidding to do your work don't have access to the brilliant ideas you keep locked in your skull. They only have the project description. This is where your vision statement for the project (you *did* write the vision statement, didn't you?) comes in really handy – if you did a good job on it, it's an excellent first paragraph for your project description.

Your project description should describe the final product you are working towards, and lay out the specific tasks needed from the service provider. It should specify your general budget, your timetable, your technical and user requirements, and any other information relevant to the joint management of the project. Here is an excellent sample project description, using a text editor such as the Windows "notepad" program as the software product we hypothetically want to create.

> "Code, test and deliver a text editor program for the Windows family of operating systems written in Visual C++. Program is a straightforward text editor with file system, print, and elementary search and replace functionality. No WYSIWYG or graphic processing is required; the program will have to deal only with text files under 2GB in size. Program should have a small memory footprint and use very little processing power. Program should follow general Windows UI framework. Test version should be delivered by 11/15, and production version should be complete by 12/1. Our budget for this project is tight and no bids over $2000 will be considered. We prefer to use e-mail for project communications but can be available for phone consultation if developer has questions."

This description tells the developer what is needed – a simple but functional text editor. It tells him or her when it has to be done, what the general level of funding for the project is (permitting an intelligent bid, or an informed self-selection not to bid), and lets developers who like to chat on the phone know that this project won't be run that way. It tells him or her what skills are needed – Visual C++ coding skills, and knowledge of the Windows interface standards.

Be realistic.

"Design and code a word processing application similar to Microsoft Word but with more powerful features. Project should be delivered in six weeks or less. We are a new company and don't have a large budget, so don't bid more than $250."

What is wrong with this picture? We may not be able to tell the future, but we can say with utter confidence that this buyer is not going to get his Word clone for the cost of an hour in Vegas. We can also say with somewhat glummer confidence that there are plenty of buyers out there writing project descriptions that rival this farcical one.

And in the end, it is a waste of everyone's time. Because you may be able to find one or two providers who will take a flyer on a project for you at a rate significantly lower than they normally charge – and they may even do a good job of it. But at the end of the day, we all get what we pay for and no more (and sometimes less).

The classic software project management aphorism runs "Fast, cheap, good – pick any two." It's an aphorism because it describes a physical reality about the development of complex systems like software programs. A good developer can give you something that kind of works on the cheap, but it will be junk. Most developers can develop something that is inexpensive and high quality – if you give them plenty of time to do the project in their spare moments. And some developers can even give you something that is quite good very quickly – and they know who they are, and they will charge you a lot of money for their very expensive services.

If you bow to this reality, and set your priorities accordingly, you can find yourself getting really good deals. If you fight it, you will find yourself with software programs that are always unusable or unacceptable in some area, and it will be very frustrating for you.

Don't be afraid to stop looking.

Particularly on the busiest sites, an invitation to bid on a project may yield dozens or more responses. This can be a daunting prospect, particularly on early projects when you have not yet learned to screen out the dead weight. There is a natural

tendency to want to look at every option and then weigh them all before making a final decision. This can be fatal in the software business, where the people who could potentially handle your project probably number in the thousands or hundreds of thousands. At a certain point, enough is enough; you only need to hire one.

Where that point is will have to depend on the value of your time and the relative importance of the specific project to you. One good rule of thumb is the Rule of Three: after you have found three bidders who meet your criteria for the project, give you a good feeling about their competence, show a good feedback history, and have bid in the right general price range, stop intensively evaluating new bidders. Simply scan the new bids to see if they have something exceptional missing from your first three prospects – perhaps an incredible introductory price, or some valuable specific knowledge – and then go ahead and make your determination.

Avoid the dead weight.

Every service has them – low-rent operations who exist solely to spam the project lists with cookie-cutter bids. The bids have no original content, and clearly are machine-generated and sent by a bored wage slave who didn't read your project description beyond checking that it had the right keyword in it, or some other similarly nuanced screening mechanism. If you actually select one of these losers, you'll pay a deposit and you may even get a few pages of badly written and amateurish code before your service provider fades into Internet invisibility, leaving your project a wreck.

Always look for signs that your bidders are actual people who have shown an actual interest in your project. Don't worry about bids that have boilerplate in them – that's just a sign of someone who knows how to use a text editor to save time and energy. As long as the bid is talking about your project and how the service

provider can help you get what you need done, then you've got a live human on the other end of the string and can get down to business.

www.elance.com – Old School Outsourcing

Elance is the oldest of the outsourcing services, and has developed an extensive community of committed users, both service providers and buyers. Although the system is older and has a number of limitations, its long established nature as a marketplace for freelance work keeps it in the forefront of the outsourcing market. Elance covers most major areas of outsourcing, and has plenty of listings for software developers.

Like most services, Elance is free for buyers to use. This means that there is a fair amount of casual buyer posting going on, but that will not effect your operations. Almost all service providers are paying money to be represented on the site, and so most of the people you will talk to can be safely assumed to be serious about finding work. Some providers put a "courtesy listing" on the site, which allows you to look for them but which does not allow them to bid on projects.

Elance allows service providers to provide "standard service packs" – fixed services at a fixed rate. For example, a Visual Basic programmer might make consulting services available at $75 an hour regardless of the project details. This may be a good solution if your project does not have a well-defined timetable or is speculative/experimental in nature. You can also search Elance for providers who meet your skills requirements and invite them specifically to bid on your projects; this can be a good way to attract good providers immediately, rather than waiting for the vagaries of the bidding process.

In 2005, Elance hosted more than 100,000 projects. Some of these were not software development projects, of course, but nonetheless there is a vigorous

marketplace here. There are more than 10,000 active service providers at Elance; you can find most any professional service here.

www.guru.com – Covering Every Base

Newer than Elance, and far larger, Guru has focused on developing an enormous user base and a huge list of active projects. Guru has a very wide range of types of service providers – you can get coders here, and also writers, artists, editors, accountants, and a lot more.

Like Elance, Guru allows providers to have courtesy listings of limited functionality, but most providers use paid accounts which permit them to bid on projects. More than 500,000 service providers compete for millions of contracts on Guru – it is easy to get lost on the service, but the category features work well and careful navigation will keep you on the right track.

Despite Guru's larger size, many providers report mixed results from the service. The size of the marketplace means that making a personal connection is difficult, and when working at a distance from a service provider, a personal connection can be a very important working tool. Experience will show which service finds you more and better providers for your projects.

www.vWorker.com – Focusing On The Code

VWorker eschews other forms of work and concentrates on the big prize: software development. The site is oriented totally around hiring software developers and as such is the natural place to start if a developer is your primary or sole need. That will often be the case – so why not use VWorker exclusively? The answer, of course, is that you will often need technical writers, artists, or other professionals to help you – and your VWorker feedback and background will do you no good in hiring those folks. All these services have their place.

More than 150,000 software developers call VWorker home; for every operating system, development environment and program type out there, VWorker offers a wide selection of developers who have specialized in exactly what you need. The service has concentrated on building a talent pool, rather than on attracting bidders, which works to your advantage: those 150,000+ coders bid on a relatively paltry 2 or 3 thousand open projects at a time.

Like the other services, VWorker is absolutely free for you to use. All fees are paid by the service providers. VWorker offers strong escrow services that guarantee no payment of project fees until acceptable working code has been delivered – a key advantage that less software-oriented sites are not as well-equipped to deliver.

Whether you use Elance, Guru, VWorker, or some other source for your software outsourcing needs, remember that the most important transaction that takes place is the establishment of a working relationship with your outsourced developer. This individual will hold part of your company's fate in his or her hands; choose as wisely as you can.

Chapter 5

Project Management

There are four areas essential to managing your software development project. These are 1) project initiation, 2) project planning, 3) project execution, and 4) project closure. Fully understanding each of these elements and their component parts as part of your workflow will help you to successfully tackle a software development outsourcing challenge.

The Four Stages of a Project

Project Initiation

Project initiation begins by signing a contract. It's as simple as that. From the outset of the project, the responsibility becomes that of your contractor. But is that really all there is to project initiation? Well, not exactly.

There are other items that have to be checked off your list when beginning a project, such as identifying a manager within your organization who will handle communication with the contractor, authorizing funds, verifying contractual elements with everyone involved, and making sure that everyone is in agreement with the figures that have been quoted by the contractor.

When these administrative tasks have been accomplished, it's necessary to ask yourself, and your institution, whether you fully understand the risks associated with the project, whether you have the financial resources to see the project through, even if it has cost overruns, and whether everyone understands the foundational elements of the software project, such as process and methodological elements.

Project Planning

In this phase, the project goes through a set-up process in which objectives are identified and goals are solidified. Any official requests must be made for the necessary resources, and a project management plan must be drawn up. This plan should include 1) a development plan, 2) a quality plan, and 3) a configuration plan.

Depending on the size and complexity of the project, you may want to use software such as Microsoft Project to create a formal project plan, or you may able to do it with simple spreadsheets.

Project Execution

Project execution is the phase in which things get really exciting. Pieces of the software begin to be developed and come together, and the project begins to take shape. This is also the phase in which some of the first problems need to be worked out between you and your developer.

Technical as well as non-technical issues need to be resolved, and a broad project management approach needs to be maintained. It will be important to conduct progress reviews and monitor the status of the project workflow and the progress of the contractors. This is the point at which reviews need to feed back into your project requirement, schedule, and budget revisions.

Letting something get out of control at this phase could spell disaster for later phases of the development.

Project Closure

Project closure is the wrap-up of the development and includes a series of final report activities and general "housekeeping" such as finalizing the project archives and conducting a final budget and efficiency analysis. Finish with a project closure report and a post-project analysis to identify any issues that should be dealt with and corrected when outsourcing projects in the future.