Managing the Client Consultant Relationship

By Vince Kellen
4/25/1997

You work for a large company. The BOSS from ON HIGH has given you the BIG project to implement. Can't fail. Gotta be done on time. Don't have internal staff. Gotta go get a consultant. You look far and wide and you find the PERFECT consultant. They really look like they can handle the job. Their presentation was great. Their bid was 30% lower than all the others. They look hungry -- like they want the job. You sign on the dotted line and things go downhill from there. The project takes longer. Suddenly the project cost is now creeping higher and higher. Your users are starting to whisper amongst themselves. The rising star of the company (you) is now crashing and burning they say. You respond by getting tough with the consultant, enforcing the contract you signed. The project keeps getting delayed. Costs are now ridiculously high. Your boss is now getting involved. You're reading want ads for peanut salesman at the sports arena. Selling peanuts has to be less stressful than this.

Have you heard this story before? How about this one.

You are project manager for a consulting company. The CLIENT from HEAVEN approaches you with the WORLD'S GREATEST project. The BIG ONE. The project that will put you to the next level. You've got the staff. You've got the bandwidth. Things look good. You're so eager to get the project, you make sure your estimate is as competitive as you can stand. The client sounds nice. If it works, your staff will thank you forever for the great project. The deal is signed. You begin work. You deliver the first prototype. The users are bewildered. It looks nothing like what they expected. You try again. The users are even more confused. After six attempts at an acceptable prototype, you call your client contact and inform them that there is no way the project will get done on time and within cost. Suddenly, the client is in your face. You're spending even more money and more time trying to make them happy but you're bleeding to death. Project costs are so out of control there is no way you'll even break even. You are now reading want ads for peanut salesman at the sports arena. Selling peanuts has to be less stressful than this.

Outsourcing application development is frequently like this. In fact, cynical consultants love to follow other consultants around, calling on the other consultant's clients about 3/4 of the way through the project.

Outsourcing doesn't have to be like this. If you stop to think about it, the interests of the consultant and the interests of the client are remarkably close. Both sides want to get done on time and within budget. Both sides want less stress. But why doesn't it seem that way?

Why Client-Consultant Relationships Go Awry

Most client-consultant relationships have problems for largely non-technical reasons, including:

- Lack of effective communication about how the project will proceed
- Improperly setting expectations too high or too low
- Lack of follow through on small yet important details
- Under estimating project complexity or scope

- Failing to understand user requirements.

Consultants and clients live if very different worlds and look at a system project in very different ways. The client is usually preoccupied (and rightfully so) with the business side of the project. Will the project meet organizational goals or objectives? Will the return on investment be realized? Will the software be easy for the users to work with? Will the consultant be responsive (or available) for fixes and changes? The consultant is usually preoccupied with technical (and rightfully so) concerns. How can we implement the request for a fancy user interface? How long will it take us to figure out how to query and combine the data for the export that the user has requested? Why are the users concerned with inconsequential interface issues instead of the more complex data relationships that need to be ironed out?

Because both sides have different orientations, both sides are blind to each other's issues. The way out of this mess is for both sides to be less blind to each other's concerns. In fact, good client-consultant relationships are sort of like marriages: both sides want the other to be happy. In fact, most problems with client-consultant relationships can be solved if both sides communicate with each other clearly and effectively. But before we look at communicating better, let's explore some of the myths out there.

Myth #1. Fixed Price Contracts are Good for Clients

Let's put on the consultant hat first on this one. Because of prior bad experiences, many clients might tell the consultant that they want a fixed price bid. This is done to avoid cost overruns often experienced on bids that are not fixed price (so called time and materials contracts). If a client wanted to be real clever, they would get several consultants in a bidding war, each of them submitting a fixed price. From the consultant's perspective, a fixed price contract means that the consultant will have to bear the responsibility of maintaining cost. It puts pressure on the consultant to come up with an accurate bid that leaves enough wiggle room should things get difficult. In other words, the consultant bears the risk. Let's put on the client hat. From their perspective, they think that they will get the lowest possible price, provided, of course, that the project was put out to competitive bids.

Is a fixed price bid good for the client? Not always. A smart consultant might decide to come in with a low-ball fixed price bid, knocking their competitors out of the picture. The consultant will then do a bang-up job, knowing that they will lose money on the deal. (In fact, I have had clients urge me to do this with the promise that there will be more work in the future!) But the consultant knows one thing that the client may not: the project will have changes. And it is these changes where the consultant will make back their lost money. By over-pricing fixed price changes, over time, the consultant will be assured of making the profit they desire from the client. Some savvy clients anticipate this and upon seeing the overpriced change, squeal and howl and try to haggle over the price. The consultant responds by pricing the next change higher by the amount of the haggle time. The consultant knows that because of their intimate knowledge of the system and the client, other competitors will be at a severe disadvantage to come in and handle the changes. In fact, many consultants are wary of maintenance contracts and price them accordingly.

I think you see where this is going. If both sides are overeager to get their best price without regard to total system costs over the life of the project, fixed price contracts are not always the solution. But then again, a time and materials contract might not save clients any mother either.

The truth of the matter is that whether the contract is fixed price or not is irrelevant. Contracts are designed to spell out what each side is willing to pay in consideration for goods or services. The key to understanding the difference between a fixed price and a time and materials contract lies in understanding one key factor: the risk of running over in cost. In a fixed price contract, the consultant assumes most of the risk (but not all) of running over. In a time and materials contract the client assumes most of the risk (but not all, too). It is only fair that whichever side assumes the risk receives compensation for that risk. This notion is a natural part of many other types of contract negotiations.

In order for the consultant to cover the risk of running over in cost, fixed price contracts should cost more than time and material contracts. Likewise, time and material contracts should cost the client less since they are assuming the risk of overruns. For example, consider a system that should be completed for $50,000 U.S. dollars. A time and materials estimate shows that the cost is going to be $50,000. A fixed price bid should increase that cost by some factor, say an additional 10% to $55,000. The difference between a fixed price contract and a time and materials contract shows how much value the parties assign the risk of cost overruns. For completely predictable, unsurprising projects, that differential can be quite small. In an ideal world, there would be no difference, we would always know how much systems cost, just as we know how much milk costs at the grocery store. Until that day, clients and consultants are going to have to discuss and agree over the cost of risk.

No matter what, in order for the client to get the lowest possible price, both sides must trust each other over the long term. When the relationship is based on trust (with verification!), prices can be more easily controlled. Why? Look at disagreements over price as friction. Other forms of friction include developing a new system in a version 1.0 product, not having a clear specification, having to adjust system design to accommodate other archaic computer systems. Friction is what drives up the cost of developing systems. When a client and consultant start having arguments over price, the consultant has no choice but to start sacrificing either features or quality.

My advice for clients is this. If you have not had clear communication about all forms of possible friction in the project, do so before you sign. How is the consultant going to handle changes? What is their software quality process like? How clear is the system specification? Is the contract profitable for them? Leave nothing unturned. Discuss everything. Don't assume that the consultant has thought about these things. A good consultant has and will have quick answers for you. If you think the consultant's price is too high, than don't use the consultant because if you introduce friction by arguing price, although you might not notice it at first, you will drive the price even higher.

My advice for consultants is this. Expose your pricing strategies. If you are taking a loss on the project, tell the client. They need to know what the total costs will be, including possible changes later on. Make sure your estimating approach is a good one that can be explained to the client. Make sure you are including all costs in your bid: testing, documentation, installation, training and client meetings. Make sure you get a good design specification.

When friction is eliminated, deciding on a fixed price or a time and materials contract is reduced to evaluating the cost of the risk and assigning the risk to one side and the cost to the other. This doesn't need to be an emotional issue. By the way, reducing friction also reduces the cost of risk! Many clients cringe at the prospect of spending money on a detailed analysis. However, I try to convince them that doing so will reduce friction and reduce overall system costs. If they need more security, I will follow up the detailed analysis with a fixed price bid, further allaying their fears.

Myth #2: You Don't Need a Detailed Specification

In order to effectively price out either a fixed price contract or a time and materials contract, a full detailed analysis of the proposed system will need to done. Unless the consultant knows *exactly* what needs to be built, any price is pure fiction. If the client does not have a detailed specification, then price out the cost of developing one. The biggest form of friction is not having a detailed specification. No one can skip this phase. In fact, I contend that a detailed specification is never skipped. If it is not done up front in an orderly fashion at a controlled cost, it is done throughout the process in a haphazard fashion at a higher cost.

I have dealt with many clients who willingly forgo a detailed specification. They tell me up front that that will not pay for a detailed specification. When clients do this, they unwittingly expose a weak negotiating hand: they have inadvertently increased friction and drive up the cost of risk. A smart consultant will recognize this and assign an inordinately high cost to that risk, thus inflating the system cost. But then again, the client may think they are one step ahead of the consultant by getting low-ball bids from other consultants. These low-ball bids are brought to the negotiating table in an effort to get the consultant to lower the system cost without a detailed specification. My knee-jerk advice to the consultant is to call the client's bluff. Let them take one of the low-ball bids! Then call on the client about 3/4 of the way through the project and ask them how it is going. Sometimes cleaning up the mess caused by a low-ball bid can be profitable!

Neither consultant nor client should pretend that they can avoid the cost of a detailed specification. A complete system can't be built unless all facts about a system are known. So whether you like it or not, an ad-hoc detailed specification is done anyway. There are strong cost reasons for doing a detailed specification. It is always less costly to make changes sooner in a project's life cycle. Here are some cost factors and stages I use:

Stage 0. Changes made before a detailed specification is started

While this stage may seem odd, since it implies that the consultant isn't even involved at this stage, there is great wisdom lurking in this phase. At this phase of a project's life, the application is usually a gleam in someone's (or a group's) eye. Any changes here I usually assign a cost of $1. Why? Because there are no documents to update, no papers or memos to write, nobody (usually) to communicate the change to. All the designer has to do to effect the change is change their mind! Obviously, if we could make the most number of changes here we would lower system costs dramatically. Great system designers can conceive of large problems largely in their minds and can iterate through many changes using their imagination, which is a lot less costly than using teams of people and lots of paper or electronic documents. However, for us mere mortals, for organizations lacking such a genius and for enormously large systems that are beyond the abilities of a single designer, we need some tools to help our pitifully puny brains. These tools require manual effort and time to update the various pieces. However, the point is clear: it is easier to make changes in this phase because there are fewer things to update as a result of the change.

Stage 1. Changes made after a detailed specification is started.

This is the second best place to make changes, and the most common, because that's why this stage exists. The nice thing about making changes here is that a system has not been constructed yet, so you don't need to test a product against the changes and you don't need to update project source code, comments or other project documents. You do need to update the design specification documents however and this does take time. Typically I assign the cost of changes at $10.

Stage 2. Changes made after a project implementation is started.

Quite often, developers and users see things that were missed in the design specification phase and ask for changes after implementation has begun. Making changes here is more expensive than the other two categories, but it is often unavoidable. Since nearly all important and even some trivial details have been worked out in advance, the best design specifications, however, will have relatively few changes here. I peg the cost of changes in this category at $100. Why, because changes here require additional, unplanned testing to make sure that other parts of the system aren't affected.

Stage 3. Changes made during product beta testing.

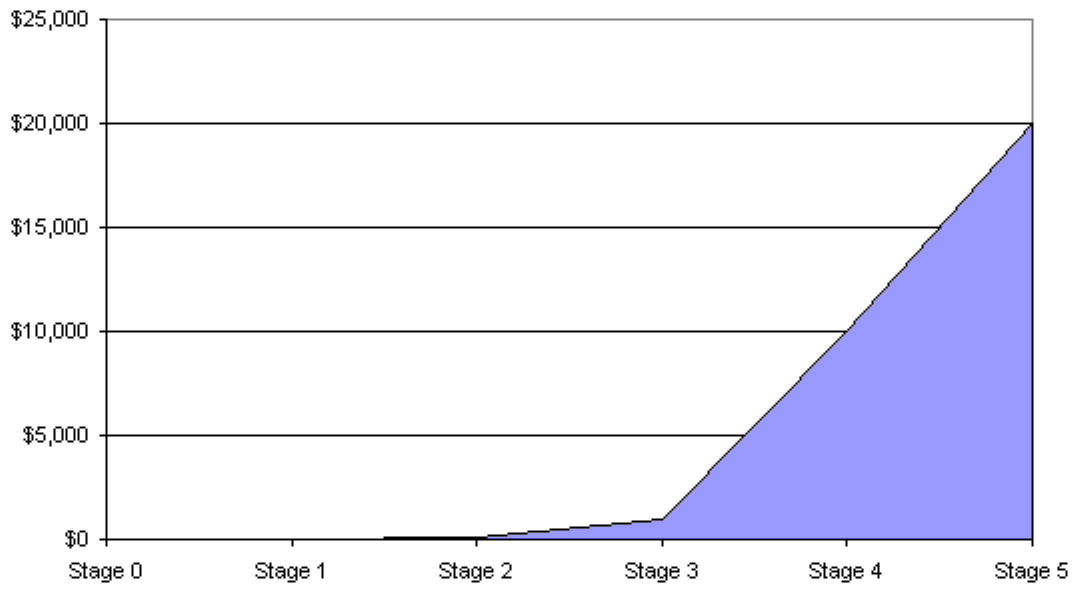
One problem with beta testing is that some users get their first whiff of an application at this relatively late stage. These users often feel left out and immediately ask for changes to get the product to reflect their needs or wishes. In addition, users who were involved early on typically don't realize the full impact of design decisions until they see a working product. Again, the very best designs have few changes here because all the important contributors to the design have had great imagination and insight and have foreseen many of the enhancements and changes that come at this stage. Nonetheless, most systems still have substantial changes at this stage as users and developers understand the system better. The cost of making changes at this stage I put at $1,000. At this point the project has quite a body of interrelated parts which require updating and re-testing.

Stage 4. Changes made during the first few weeks of product use.

Unfortunately, for many projects, this is really the beginning of the detailed specification. These troubled projects have not had the great imaginative thought that a good design specification requires. Instead, at work on these projects were less than brilliant system designers who couldn't clearly imagine a new system before implementation. Users don't see the design mistakes until the application starts running production data under production timing and usage requirements. Excessive changes here means that the detailed specification was not done or was done poorly. To avoid changes here I urge all consultants and clients to move this stage to stage 0. This requires a vivid imagination on the part of the designers, because they have to imagine the product is real well before the product is real. If all clients and consultants did a better job imagining the product as a real product from day 1 of the design specification, costly changes can be largely avoided. I usually assign the cost of making changes in this stage at $10,000.

Stage 5. Changes made after the first year or so of product use.

Once a project has made it to production and lasted a year or so, it has entered maintenance mode. Changes at this stage become difficult because very often, then system creators aren't around to make the necessary changes. Programmers less familiar with the system make the changes. In addition, as more and more changes roll in, the system becomes something other than what it was intended to be. In other words, entropy sets in. Sometimes changes conflict with other changes as various parties misunderstand the system's original design. However, changes do occur at this stage and I usually place the cost of changes at this stage at $20,000.



If you look at the cost of changes, you can see the following graph:

These numbers are approximate only and the cost of making changes late in the game looks unusually high. However, our experience has shown that the total cost of changes is much higher than the typical developer or user realizes for several reasons. The overhead of testing a late change must be considered. When changes are made early in design, before extensive system testing, no allotment for testing needs to be made. When changes are made after system implementation, substantial testing is usually in order to make sure the system behaves properly after the change is made. In addition, the cost of updating user manuals, technical documents and any other administrative procedures must be considered.

What is in a Good Design Specification?

A good design specification should exhibit the most amount of realism possible. We like to include the following in our design specification:

- A detailed data model, showing each table's primary key, field listing and relationships between all tables.
- A prototype version of the data model, implemented
- A detailed list of forms describing the form's use, function and components. Each button and menu option on the form should be described.
- A prototype version of each form so that a screen shot of the form can be placed in the design specification and optionally, a system prototype can be generated.
- A detailed list of reports describing the report's use, functions and components.

- A list of additional services and products required, such as installation, training, on-line help, end-user documentation, technical documentation.

- A list of security requirements.

- A list of performance requirements.

Clients often ask us for fixed price bids without first a detailed specification. We refuse to do so because we consider it rather unethical for us to determine a price unless we knew what is being built. After all, it's easy to price something that can't be measured because then any price will do! Seriously, without a detailed design, it is too easy for both sides to deceive themselves as to the real system cost. My advice for clients is to double-check a consultant's bid. Have they considered all aspects of the project? Do they know exactly how many objects need to be built? In detail, how does the consultant go about determining how much effort it will take to build an application? Don't assume that the consultant knows what they are doing and don't assume that your contract protects you if the consultant fails to understand all aspects of the project. A failed project reflects badly on both parties. My advice to consultants: Don't let the client talk you into forgetting a system design unless the client is willing to absorb the rather large risk of doing so.

Who Should be on the Design Specification Team?

Getting the right design specification team is so critical because a good design can substantially reduce overall system costs. So who should it be? My advice is simple: Don't let politics determine team membership. Put the best company and consultant visionaries on the team regardless of politics. Frequently, organizations want to have different departments represented equally on the team. If you have to do this, choose good people. If a department doesn't have a good person available, don't accept a bad substitute. And by all means, don't build a large team made up of all sorts of mediocre representatives from various departments. Instead, select the people who have good company knowledge and a knack for envisioning a product. Big teams don't build better products. Good designers build better products.

The industry has a variety of design methodologies that you can use to analyze and design systems. I really don't care which methodology you use because it is not the methodology that makes for a good design. It's how well the methodology is actually used by the designers than can make the difference. The methodology should exhibit the following traits:

- It should help the designers uncover key technical issues

- It should help the designers document key technical issues

- It should make developing a prototype or a data model easy, not difficult

- It should inject realism into the process by helping the designers think about real functionality that must work for real people

- It should help the designers and the users communicate with each other

- It should help document precisely what needs to be built

The design methodology is a tool to help the imaginations of the designers. It is not a straight jacket that limits or impedes the designer's imagination. Many people look at design methodologies as a *constraint* on the design process which can help *control* and *structure* the design process. I prefer to look at methodologies as a *magnifier* in the design process which can help designers *visualize* and *validate* the application. If you think of design methodologies falling into the former category, you are most likely headed for trouble, or at the very least, the design methodology will provide little help.

A great methodology does not build a great application. Great designers with good tools build a great application.

Myth #3. The Consultant is Entirely Responsible for an Application

Clients often assume that once a consultant is contracted for an application, *they will not have to do anything. Everything will be taken care of for them.* Some consultants even like to give clients this impression. While this might be the case when the client buys a car, when the client buys a well-tailored custom application, the client does have some work to do. In addition, if the project fails, it is not just the consultant who is at risk. The client is at a great risk as well. With this in mind, here are some tips for clients:

- Ask your consultant for resumes of the people they want to put on the project. Good consultants hire good people and will be glad to show off their expertise. If the expertise is not up to your standards, tell the consultant.

- Review your consultant's methodologies, project tracking systems and defect removal and tracking process. Good consultants have well documented, well entrenched systems in these areas.

- Find out what industry leadership expertise your consultant has. Have they demonstrated the necessary expertise in the product or service they advertise?

- Check consultant references carefully.

- Have a close look at other pieces of software the consultant created.

- Get an understanding for your consultant's vision for the future. Does their vision make sense to you? Does it seem aligned with your vision (if you have one)?

- Get a look at the consultant's standard contract. Are the terms of the contract acceptable? While consultants and clients can usually negotiate contracts that are mutually acceptable, sometimes you might find a show-stopper detail in a contract that the consultant will not alter.

Granted, finding a good consultant means that you won't have to verify every little detail. Building a successful, long-term relationship requires this "trust and verify" effort early. Once both sides understand each other's skills, the relationship requires far less maintenance.

Myth #4. You Don't Need to Read the Contract

In their eagerness to get a system up and running, many times, consultants and clients gloss over the contract deals. What I find particularly problematic is when either side has a contract that has terms and conditions that are ignored during the project. For many computer professionals, attending to the details of a contract ranks right up there with maintaining 20 year-old legacy COBOL code. However, when it comes to managing the whole process, the contract is key.

Each organization and each consulting firm has its own ideas about what should be in a contract. And sometimes clients and consultants just can't agree over certain terms. If that's the case hen the client and the consultant are not -- how can I put it -- compatible. If this is the case, there is no time like the beginning to determine contract compatibility. Sometimes one or both sides will try to slip an obnoxious contract past the other side late in the game, sometimes even after the project is nearly completed.

You may have your own preferences for what should be in a contract. Here are some of the ideas that I like to see in contracts:

- The ability for the consultant to bring reusable objects to a new application yet still retain ownership of the reusable objects. Consultants tend to develop a comprehensive body of reusable objects which can lower system development costs and increase system reliability. Clients should be able to get the benefit but not have to pay extra for it.

- The consultant should be able to take non-confidential, generic reusable objects developed for this client and add it to their reusable code base for application, without charge, to the next client.

- Terms regarding how the client is allowed to use, sell or dispose of the application should be clearly addressed.

- Who holds the copyrights to what parts of the application should be clearly discussed.

- Clients should not have to pay to fix defects found after the product has been certified by both parties as ready for production. This will ensure that the client won't have hidden costs later in the project life cycle and give the developer incentive to build relatively bug free software. By the way, there is no such thing as bug free software and despite all the talk about six-sigma quality programs, formal methods, code inspection and other quality techniques, creating software is an enormously complex task. The industry is a long way from writing bug-free software.

- Payment terms clearly spelled out so that both the client and the consultant understand when and how the application will be paid for.

- The contract should refer to a design specification that clearly lists what objects and functionality will be built. Any objects or functionality required not listed in the contract should be authorized via a contract rider or a separate change request document.

When contracts as written deviate from the client's and the consultants actual business practices, trouble brews. Contracts should mirror the actual working relationship that develops between both parties.

Birth of an Application: Managing the End Game

I've seen too many applications needlessly killed in the birthing process. In the vast majority of these cases, the reason the project got such a bad reputation had nothing to do with the quality of the construction process, the talent of the designers and builders or the level of user involvement.

In most cases it is because *the consultant failed to communicate to the client how difficult the birth process can be!*

Installing a new software application is a very stressful time. Stressful on people, on equipment on budgets. And just like real births, *the more protracted the process, the more stressful it becomes.* Here are some tips for both clients and consultants that can make installing software easier:

- Set user expectations lower. Don't oversell how easy the install process will be. It usually not as easy as anyone makes it out to be. Do not underestimate the amount of effort and time that users and developers will need to spend to complete a successful install.

- Before you actually do the installation, establish clear expectations for on-site beta testing and repeat them often. How many times do you hear it: "This product is in trouble. Do you believe the number of bugs we're seeing in the beta cycle?" Whenever anyone says this, tell them "Hogwash!" That's exactly what the beta process is for -- *to uncover as many bugs as possible!* Why should anyone criticize the very goal of beta testing? I tell clients to consider beta software like a five-year old child going to school for the first time: it is too early to forever brand the child as a social misfit or an intellectual has-been simply because their first day at school had an incident or two. *The number of bugs discovered in the beta process is not important. The rate at which bugs are removed is!* Good systems may have lots of bugs that are easily removed. It's those bugs that indicate a serious misunderstanding of the system design that are difficult. But then again, if you have a good design specification, there should be few bugs of this kind.

- Don't set a delivery date until you have received enough bug reports. If you don't get any bug reports from your users or testers, tell them you will refuse to ship until you get some bug reports. Keep track of how many bugs you find and fix over time. You can use this information to help you fix a shipping date.

- Clients: make sure your consultant does testing, but don't rely on them for all the testing. You will always want to test their software on your equipment in your environment. Consultants: insist that clients test your software in their environment. Both sides: don't get into arguments over who does testing late in the game. Testing is a simple matter and someone's got to do. All that needs to be done is to adequately estimate, up front, what testing costs should be and who should pay for them. Custom application development can have testing costs that vary from 20% to 40% of the total system costs. If anyone tells you they can fully test all phases of a complex system for as little as 10%, they're lying! Most large software houses (such as Borland and Microsoft) dedicate about 50% of their software development costs to quality assurance and testing. This budget is spent throughout the project life cycle, not at the end.

- If the process is particularly difficult and mission critical, do a "dress rehearsal" -- that is, actually go through an installation on duplicate hardware or software which is not used for production. Take notes, see how long it takes to complete the conversion, see what data converted and what data did not convert. On one project that I led, we actually went through three separate dress rehearsals in which we did just about everything we were going as part of the real installation. Needless to say the installation, which had to be done on a weekend and involved staff in three different cities, went off without a hitch.

- Use automated defect logging tools. Consultants can have clients submit defects electronically so that there is a clear list of known bugs and so the resolution of those bugs can be tracked. When clients and consultants are linked up electronically, the relationship between the two becomes deeper, since severing the relationship will require work on both sides to devise a new defect submission process.

The problem with installation applications is overcoming the high expectations the client or the consultant may have set in order to get the project approved. The installation and deployment part of the project life cycle is not for the squeamish. Although it is appropriate that clients demand quality software, the process by which high quality software is made requires fortitude. Both sides need to be prepared. Consultants should remind their clients early and often what can be expected during the on-site testing phase.

While on-site testing is important, it is not the best way to get quality software. Reliable software is built when all phases of the development process are reviewed, verified and tested and when reusable components are employed. In fact, the more time both sides spend early in the process in making sure the design specification is top notch, the less time will be spent in testing and bug busting.

Managing Client and Consultant Turnover

There is an old saying that all towns turn over every five years. This means that the bulk of key players in a community change every five years. This is certainly true in application development. In fact, in many programming shops, employees typically last only one to three years. In addition, because of the volatility in the business world today, many companies are not afraid of substantially restructuring themselves, resulting in lots of turnover in key positions.

Projects often fail when key personnel leave either the client or the consultant. The reasons are varied. The new staff sometimes looks for a scapegoat for their problems (the old system) or the new staff just doesn't like the old system or the new staff just doesn't understand the old system. What can consultants and clients do about this?

- Try to keep key staff on the project for as long as possible.

- Plan time, as part of the project budget, to bring new project staff members up to speed.

- When turnover happens, both sides should work hard to make sure new players become acquainted with each other.

- Both sides should remember that the natural tendency is for either side to sever the relationship when turnover occurs. In some cases this may be appropriate. In other cases, it can cost either side significant time and money.

I hate to sound like a broken record, but if the design specification is good and the design and technical documents are kept up to date throughout the project's life cycle, getting new staff members up to speed is easier. At Kallista, Inc., we use Microsoft's Exchange Server as a document repository for all pertinent project documents including:

- Design specifications

- Design notes

- Logs of phone conversations
- E-mail messages between team members
- Change requests
- Technical documentation
- User documentation

We have a standard folder hierarchy for all projects so team members can easily find key documents regardless of the project. And now that Exchange Server has the ability to publish folders as Internet documents, we can make this information available through web browsers.

Some Final Thoughts

Managing the client-consultant relationship requires diligent work. When clients and consultants communicate frequently throughout the all phases of the process and hide nothing from each other, the relationship can be a good one. In addition, there is a big benefit for clients if they can successfully manage this process: they can lower system development costs. For the clients reading this and doubting me, think about the problem from the consultants' perspective. Consultant's don't want to lose money on a project and they don't want to spend "hassle" time with the client. If the client can arrange a relationship in which the consultant is guaranteed to make their profit and the client is guaranteed quality software, the consultant will be less likely to employ pricing games to make back lost money on bad projects. When the relationship is based on trust, both sides can streamline the communication process and save money that would otherwise be spent on screaming, yelling, accusing and complaining.

Consultants should also look at it from the clients' perspective. Clients want software that works reliably built on time and within the budget allocated. Don't wait until the end to see if the software does what it is supposed to do. Get a good design specification! Test the product design specification, the product prototypes and the release candidates. Make sure the product meets client requirements. When requirements are clearly documented, change orders can be negotiated more easily. When the design specification is determined early, it is much easier to build software on time and within budget.

Both sides should remember that the vast majority of projects fail for non-technical reasons. If both sides want to save money then both sides should master the non-technical aspects of project development.