

So You Want (or Have) to Be a Project Manager

Brian J. Geiger

bgeiger@pobox.com

Abstract

Project management is a difficult task that is usually little understood by the people who do it. Very few people, when they are growing up, hope one day to be a project manager. Generally, you are promoted to project management either because you are good at it or because you're the only person who is available to do it. Project management is a full time job, and it encompasses several duties that most programmers don't want to do. That's why project managers are so valuable to programmers.

Why, oh why?

You are a programmer. Chances are, you're a good programmer, and you enjoy programming. Why, then, do you want to become a Project Manager?

There are a few reasons to do so, some good, most bad. If you become a Project Manager for the wrong reasons, then you're either going to be a bad Project Manager, or you're going to hate what you do. Often, the two go hand in hand.

The most common reason to become a project manager is because there's a need, and someone figures that it's not a difficult one to fill. The perception is that Project Managers do very little work aside from keeping programmers from doing their work by asking foolish questions, and if you already know what's going on with the project, which you do, you've just eliminated most of the work. It should take upwards of 45 minutes per week to take on the additional duties, and the pay raise is certainly worth it. This is, by far, the worst reason to become a Project Manager.

Another reason to join the ranks of Project Management is because your boss asked you to. You're an organized person, you generally know what's going on in your project, and you aren't too difficult to work with. This is not a bad thing, per se, but it's not necessarily

the right reason to become a Project Manager.

Perhaps the most rare and wonderful reason to become a project manager is because you find that you prefer keeping the project, and the people working on the project, on track far more than you like coding. If this makes no sense at all, then resist all temptations to become a project manager. You will have little time to code, and most of your time will be taken up interacting with people, usually people who are not happy for one reason or another.

So, what does this "project manager" do, anyway?

Being concise, a project manager ensures that a project gets done, preferably on-time and within budget. This involves a myriad of tasks, depending on the abilities of the individual in question and the constraints that the project is under. If you have an unlimited budget, all the time in the world, the proper people to do the job, and a management team that is strangely unconcerned about the progress of the project, then you will likely have an easy time at project management.

Unfortunately, the dot.com era is long over, so a project manager's life is likely to be more difficult. Consequently, expect to have to have regular meetings with whomever pays the bills (we'll call them "the client"), which will take away from your time working on the project itself, so you'll have to have regular meetings with everyone working on the project to keep up-to-date with the various parts. Because you're responsible for getting the project done, that means that if anyone has a problem, you're going to be expected to fix it. If you don't have everyone you need to finish a project, then you have to convince the people who control the checkbook to hire you someone new. If you're lucky, you'll get to sift through the resumes, perform the interviews, and make the final recommendation to Human Resources or its local equivalent. If you're not, then your boss will give you someone who is just perfect for the role, who may or may not be his wife's unemployed nephew.

That's a quick overview of the things likely to take up your time as a project manager, but it's not as dismal as it may seem, if you're up for it. If the above description is an accurate definition of a deep circle of Hell, then I suggest staying away. However, if the description above is an accurate definition of one of the more shallow circles of Hell for you, then you may just have what it takes to be a project manager.

Interfacing (with people, not peripherals)

The fastest way to get a given project done is to make sure that the people who are working on the project do nothing except work towards the final, release version of the project. There are many ways to keep the workers on track, but I like to separate them out into the Right Way and the Wrong Way.

The Wrong Way

Bad managers, and by extension bad project managers, think that if an employee isn't coding or coloring or diagramming or whatever is that person's primary function, then they aren't working towards the final version of the project. Therefore, they figure, in order to optimize the employee's work ability, they spend their time walking around and making sure that Instant Messages aren't going on, that Personal Phone Calls aren't being made, and that Web Sites are not being surfed.

Unfortunately, this is a big waste of time. If the employees are goofing off, then you've either hired the wrong person, or they're not happy with what they're doing. If an employee is working hard, it's okay for them to take a break every now and then and perhaps even to take care of some personal business. If you treat someone like they are nothing except resources to get the job done, eventually they're going to get the job done elsewhere.

One of my project management jobs was at a small software development house. There were a lot of great people there, but because it was just getting started, everyone had to work a lot of hours more than the standard 40. Sometimes in the middle of the day, we'd want to play some *Heretic* or *Duke Nukem' 3D*, or whatever was popular at the time. The boss would then proceed to throw a fit and threaten to ban all LAN games, and then would tell us that we had to work until 10 that night to get a last minute change from the client done. It took a long time for me to convince him that his way was far too heavy-handed for the type of team we had. Not that I'm advocating LAN games in the middle of the day for all projects, but you have to temper the extra working with some perk, and as long as they don't abuse it, it's better to let it go or join in than to squash it mercilessly.

The Right Way

If someone enjoys what they do, the best way to allow them to do it is to eliminate as many barriers to doing what they enjoy as you can. This means that, if they're programmers, chances are that they don't want to talk to upper management or especially the customer. They want to avoid as many meetings as possible, and they don't want to spend their time redoing something that's already been done because somebody changed their mind.

What can you learn from this? Well, first, you want to eliminate as much of the things that the employees don't want to do, freeing them to do the things they enjoy (which, if the hiring was done properly, is exactly the same thing that will get the project done). The second is that you want to minimize the amount of unnecessary changes.

One of my favorite ways to eliminate unnecessary work is through automation. I've made a few licensed toys for Mattel, including a talking Barbie® doll followed by a talking Winnie the Pooh®. When we did the first, I let Mattel give us the sound files on tape, with paper scripts marking which take to use and what is on each tape. It was terribly inaccurate, and caused the sound department about 3 months of work to digitize and name all the files, plus countless hours verifying that everything was correct.

When we did the second project, I insisted on finding a better way. So I found a system that would act as a teleprompter and audio digitizer, and had it record everything into files, naming everything properly and linking it to a database with all the script information. Not only did we save all the time on digitization and file naming, but more importantly, it allowed the sound department to spend their time making music and other creative pursuits that suited them better.

Talking with others

It's impossible to completely separate people from the aspect of the job that they don't want to do, but you can minimize the contact. There are a lot of little questions that keeping track of a schedule will answer, such as "What happens to the project if we're two weeks late receiving this product?" or "Aren't we supposed to be done with this module by now?" Keeping an accurate schedule and having it updated regularly will minimize the amount of work lost due to interruptions. As with multiprocessing, in real life, changing tasks requires a context switch, and if the switch is not expected, the interrupted person will lose a non-trivial amount of working time. Try to avoid that if possible.

Because the people paying the bills want to be assured that everything is okay, as long as you as project manager can demonstrate that you know what's going on, the less often people are going to bypass you to get information from members of the team.

Getting Your Way

Now for the meat of the project management buffet. Project management is a job filled with compromises, and it's the project manager's job to get the compromises together to make the best possible project. It's a constant life of negotiation, and it's important for you to remember that your job is first to the project, and second to the people involved in the project. This means no adding in features just because you can't say no to one of the programmers, or discarding a perfectly good suggestion just because you hate the person who made it.

You also have to convince the client that your way is the correct way, based on the needs of the project. Sometimes this means adding or removing features, sometimes changing the schedule, and sometimes modifying the budget. When you step back, those three items are your primary restraints: Features, time,

and budget. And the one of the trickiest tasks of a project manager is to teach the people with the money that you can control, at most, two of those restraints. If you are doing something that requires a ton of features, and you have to ship by a certain date, you're almost certainly going to have to be flexible with the budget. Fix any two of those, and the third will be subject to change.

The project manager has three tools that correspond to the restraints. You have a Schedule, to track the time, a Spreadsheet, to track the budget, and Requirements, to track the features. These will be your primary weapons in the war of Getting Your Project Done, so keep them in good working order at all times.

Schedule

The primary tool for the date-driven project, the schedule is vital for negotiations. You will need several different versions of your schedule for different purposes, and it's generally pretty wise to go with a fully featured scheduling program such as Fast Track Schedule or Microsoft Project. Actually learn to properly use the software, which may mean reading the manual. An improperly formed schedule is only useful once. If you don't have it set up to change later deadlines automatically if something early in the schedule changes, then you're just wasting a lot of your own time.

The schedule that you will use for day-to-day work is the detailed schedule. This will contain everyone's individual schedule contributions, in as much detail as possible. Don't forget to double any time estimates you get, until you've determined how accurate any given person is at estimating.

For the client, you're usually going to want a high-level schedule, letting them know what you're working on now, if you're on time, and what the potential road blocks that need fixing are.

Occasionally, you will need to use the more detailed schedule in order to prove that, in fact, it will make the project late to change feature X to feature Y. The schedule is a powerful tool for such things, because the client will generally only have a rough idea of how long things take to do, and adding features to a hazy schedule rarely changes the timeline. However, adding features to a fully-formed schedule will let them know exactly what it would cost the project, so you can let the client make an informed decision.

Most of my projects have been date-driven, so I learned to use my schedule to its maximum advantage. At the time, most video game projects were 1.5 years long or more, but I worked primarily with Mattel, who is on a toy schedule. A toy schedule means you have a demo for a project in January, then you have to have the game shipping by September to make the holiday buying season. Naturally, they wanted to pack as many features as possible in, so we would work out how long the features would take to implement, which features would naturally complement other features in terms of programming, and we'd lay out the schedule in front of them. They would see that they've asked for about 6 months more of features than we have time for. Then they would choose which features they needed until we could fit the schedule. Alternately, they would pick the features they needed, and if they didn't fit the schedule, we might see if we redesign the application in such a way that we could finish it in time. The important thing was that they were able to participate in the process. The client hates to feel that the developer is dictating the direction of the project.

For the programmers, you want a detailed view of what they are working on, and any dependencies related to those tasks. The dependencies will show why they're not able to start on something, and what the consequences

of being late on a part of the project will be to the rest of the project.

Budget

The primary tool for the money-driven project, the budget is just as useful of a tool with clients as the schedule. Its primary purpose is, like the schedule, to point out numerically the difficulties with changing the project mid-stream.

Keep in mind, adding resources will not *necessarily* speed up your project. They might, but in order for that to happen, it will require more work from you and possibly your lead for whatever section needs speeding up, such as your lead programmer. There's a certain amount of overhead involved with adding a person to a project team, plus additional work to coordinate that person with the rest of the team. You are the person who has to get that person up and running as quickly as possible, and it won't be easy. It's possible to speed a project by adding people, but it's not easy. It's much easier to plan properly in advance, and add, for example, faster machines (where doing so would speed the finish of the project).

I was recently working on a relatively low-priority web site project. We had some external developers supplementing our internal development staff. As time progressed, the project became more and more of a priority. At one point, one of our internal programmers suggested that he take over for the external programmers, believing that he could program faster than the external team.

Unfortunately, he misunderstood the nature of the problem. The project was going slowly because the Project Manager (me) didn't have enough time to devote towards traffic direction. Since I couldn't ensure that the ball was moving without taking too much time away from my other projects, it went slowly. If I had wanted him to take over, that would mean that I would have to

bring him up to the point where the external team was, and then ensure that he remained on the right track. However, once the project reached its appropriate level of urgency, it was finished quickly enough.

Requirements

Unlike budgets and schedules, which are well respected in the software development world, requirements are the most overlooked-but-necessary item in project management. Oh, sure, everyone overlooks QA, but at least they know they're overlooking QA. Requirements are something that most people don't even know that they're overlooking.

In a good software process, the Requirements Phase is the one before Creative Design. It's where you find out everything that you need your project to accomplish once it's finished, and everything that you know you are going to need in order to get it done. Requirements are difficult, because you have to think about what's needed in abstract terms, in order to keep from limiting the project unnecessarily. There'll be more on this in the Process section.

Keeping the Faith

The final job of the project manager is to ensure that the client does not lose faith in the project and cancel it. Most projects have delays in them, or increased costs, or a loss of features. It's a natural part of the cycle, and one that you shouldn't lose too much sleep over trying to prevent. The problem comes in when you go too far in any one direction, or if you repeatedly have problems with no good reasons for those problems. If you can't demonstrate to the client that these problems won't keep happening, the client won't keep paying for your project to be made.

There are a few standard mistakes that beginning and experienced project

managers make that would cause the client to become disheartened.

“Just one more week...”

The most common mistake is to keep accepting time estimates, or to continue to make time estimates without accurately working out all of the details. You can recognize this behavior by the following loop:

<Start loop>

“How long until the task is done?”

“Oh, it’ll be done in a week.”

<Idle 1 week>

“Is the task done yet?”

“No, I ran into some weird problem, but it’s almost done.”

<End loop>

Listing 1. One More Week Loop

After no more than the third time you have that conversation, you should realize that whoever is performing the task really has no idea how long it’s going to take to complete. That’s when you should look into breaking the task into smaller tasks, in order to find out where the slowdown is and hopefully more accurately track it.

Bear in mind that the conversation in Listing 1 is one that you could have with a member of your project team, or it could be a conversation that the client has with you. If it’s the latter, then expect the client to start thinking that you have no idea what you’re doing after about the third or fourth iteration.

“Yeah, I forgot about that feature...”

Another common mistake is not accounting for all the features, procedures, or stages. You may have dead-on accurate time estimations for everything you added into your schedule, but you may have forgotten

that you’ll need a help module, and that will have to be written and added to the application. Or, you might have forgotten that Operations needs a week to install the servers after you give it to them, and another week to get the software installed onto the servers. Or, you might have forgotten that the product has to be tested before it goes to the customer (which is a surprisingly common mistake). Each of the above adds more time to the schedule, and each will make your client more disheartened about the prospect of finishing the project.

Not doubling estimates

There are two reasons, in general, to double time estimates. The first is because the people working on your project are wildly optimistic and have no idea how long it takes to get anything done. The second is that for any given task, only about half their time is going to be spent working on the assigned task. The rest of the time will be spent going to meetings, eating lunch, talking on the phone, answering emails, complaining about you, and surfing the web. These are all natural things, within reason, and if everyone is a good worker besides, there’s no reason to try to stop it. However, it is something you need to plan for in advance. So, double any estimates. Sometimes, you might need to double them twice.

Not understanding the Critical Path

This one’s not as common, but it’s very important. The critical path is the series of tasks in which, if any one is late, it will cause the project to slip. There will be plenty of tasks in the schedule that could be slipped without affecting the overall schedule, and you can use those to your advantage. Be mindful of the critical path, however, because every mistake there will cost you.

Hitting cascading delays

Our final common mistake is one of the more esoteric ones. I’ve known some

seasoned project managers who had a difficult time coming to grips with cascading delays. It's not too difficult, though.

A cascading delay is when a delay in your critical path causes a disproportionately larger delay to affect your project. Metaphorically, it's hitting a red light before the train tracks, right before the train goes through. If you had made it through the light, you would have missed the train. Because you hit the red light, you're stuck for a half hour waiting for the train to go through.

In actual terms, it's missing your publishing window for your CD ROM, and you have to wait for two weeks for your publisher to be free again. Alternately, it's not getting your probe of Mars tested before the launch window, causing a delay of several years.

When I was making Multiplayer Online games, we always had to coordinate with many departments. One was Operations, which ensured that server machines were ordered, installed, and ready to accept a game and related processes by the time you wanted to go into serious beta testing. Had I neglected getting that one notification out with the proper lead-time, it would delay getting the game into beta. If my proposed beta test period then overlapped a similar game, I would either have to risk losing good candidates from my testing pool, or I would have to wait until the other game's testing period was over. For some games that mattered less than others, but it was always worth considering.

Any time you are dependant on something in your critical path that works on a cycle that's longer than an acceptable delay, that's a potential cascading delay. Don't miss those, or it'll hurt.

Process

Finally, we arrive at the ultimate expression of a project manager, a working software process. In order to get a software process, you have to have buy-in from everyone, including the client, your co-workers, and the project team. A software process can't work by itself, but with a supportive team behind it, your life can be much more predictable.

A software process is, in its simplest form, a way of managing a project that is consistent and, more importantly, documented. A standard process will go, at the high level, something like: Concept, Requirements, Creative Design, Technical Design, Implementation, Testing, and Release. It is guided by the plan.

The plan

Not all projects, perhaps not any project, will exactly follow your process. A process is merely the baseline that your project should follow. Your project plan is the document that tells anyone who is curious how your project is going to differ from the established process. If you are making a film, and you decide you don't need a Technical Design, you would say so in your plan. If you are making some really simple software, and you decide that you don't need testing, don't even think about removing testing. You need it.

Your plan will change over time. The further you get into a project, the more you're going to know about it, and the better decision you'll be able to make. The plan should keep track of changes you make to it, however, so don't go crazy and revise your plan from week to week without documenting those changes.

Concept

The concept is the earliest phase of any plan, and it's the thing that everyone can do, especially if you're in the games

industry. "I had a great idea! You take Unreal Tournament, and you mix it with Final Fantasy, and make in a Massively Multiplayer Online Game! Brilliant!"

The concept is the high-level overview that embodies the spirit of your project. It may have a sample design, but that design won't constrain a real design later on. It may mention some technical details, but those are only there for fun. The concept merely inspires the later phases to be the best thing possible, without being terribly constraining.

Requirements

Requirements are the good stepchild in the fairy tale of project management. Given the opportunity, they will blossom into a beautiful princess, capable of winning the heart of royalty and, in more progressive fair tales, saving the kingdom. However, most development teams treat them like particularly incompetent slave labor, doing chores that it's really not suited for.

(An alternate interpretation is that they're treated like they don't exist, because they're mythical.)

Requirements perform two major functions. The first is to find out everything that you need the project to accomplish when it's done, at an abstract level. The second is to document what help you're going to need from anyone else to get the project finished.

One of the uses for requirements is to focus your mind on what is needed to get the project done. Another is to have a document that keeps track, at the base level, of why the project is going to take more time to do when the client changes his mind. Finally, the requirements are a base on which to build the rest of the phases.

It's very important not to become concrete with the requirements. "Must develop 3D engine in three months," is a requirement. "Must use Unreal Tournament 3D engine to minimize development time," is an implementation detail. Don't put implementation details in the requirements. That's what the creative and technical designs are for.

Kesmai was huge on software process, and we took our requirements seriously. Some people did so more than others, but it's the people who take them seriously that are both key to making good requirements and the biggest source of serious frustration that you're likely to find from anyone who isn't the client. We had to budget a large amount of time for requirements. They didn't take a large percentage of working time, but they took forever to get approved, and there were many reject-revise cycles.

Creative design

The creative design is the first level of real design. It's where you determine exactly how the project is going to function when it's put in front of the end user. You don't worry about how it's going to do any of these things; you just know that's what it has to do.

Remember that your creative design is based on your requirements, so everything in the creative design should map to at least one requirement. If it is otherwise, either you didn't think out your requirements well enough, or you're adding in unnecessary features.

Technical design

The technical design is the next level of design, where you determine, at a high level, how everything in the creative design is going to work. No actual coding should happen here, but if an artist can look at the technical design and understand it all, it's probably not detailed enough. Well, unless the artist has a Computer Science degree, in

which case that's okay. Still, there should be plenty of details in there about which algorithms you might want to use, what all of the interfaces for the functions/procedures/classes/circuit should be.

Remember that your technical design is based on your creative design, so everything in the technical design should map to something in the creative design, and should, consequently, also map to something in the requirements.

Implementation

This is where you build your project. You probably know how this phase works already, but it's still built on the previous phases, so make sure that everything you do maps back to all the previous phases.

Testing

Testing is the phase where you make sure that your product is doing what it's supposed to, and nothing more. Testers should have been involved in the process from the beginning, and they should be furnished with complete copies of all the documentation for the project, include the Plan, from all of the earlier phases.

If testers are just given the mostly-completed project, then they can only test the project with how they think it works, not how it's supposed to work. However, if the testers have the Creative Design, they know exactly what functionality is supposed to happen when you press the shiny red button, and they know why, because they can see what requirement it's fulfilling. So, not only will they be able to verify that the entire interface functions properly, but they can also verify that all of the requirements were successfully fulfilled.

Release

You're done, so you ship the product and forget about it - unless it's an online

product, or has some manner of ongoing support. Then you have to have procedures set up for keeping the support going, fixing bugs when it's live, gathering customer feedback, etc. A project is rarely done when it's over.

Managing the process

So, you may be thinking, isn't it a lot more work to use a software process? Yes, yes it is. Why, you could think, would I go through all that, when we could just make the product the old-fashioned way?

A software process helps you answer the most difficult question in project management, if you're not ready for it: "How do you know when the project is finished?"

With traditional software development, you code and code, and someone asks for more features, or tries to get some bugs fixed, and eventually the deadline looms, so you fix whatever bugs are left, and put in some last minute changes, then fix those bugs, then, hopefully, release the product. Then you patch it. Maybe you got all the features in you wanted, maybe you didn't. At the end of the day, though, you decided that the project was good enough, so you released it into the wild.

With a software process, you know what you need in order to finish product. It's predictable, it's testable, and it's knowable. Furthermore, with a more mature software process, where you've revised it based on experience and gathered metrics, you'll start to understand how much time and effort it will really take to finish any given project. You'll also be able to see where the problem areas are likely to be. Most importantly, that knowledge will be written down and published for your organization to see, so it won't vanish when you leave.

It's the last point, retaining the knowledge, that is the reason it's so

important to track changes to your process, and to track changes to your project. If you add a new feature that wasn't in the requirements, update the requirements and track the change. Depending on where the change was in the process, you will have to make the change in more places, which is convenient, because it will more accurately document how much trouble the change was likely to cause.

That's the other reason for the process: it lets the client understand, without being rude or patronizing or devious, why a change is going to affect the schedule, despite the fact that it seems like such a simple change. Then you can ask for more time, more money, or fewer features. The client can determine if it's worth it and won't feel that you're trying to make excuses for why the project is going to be late. That will help your project, ultimately, get done.

Miscellaneous and sundries

These are a couple of last-minute items that I added because they are invaluable bits of advice that I give over and over again.

Delegation

Going from a position where you work long hours and produce a product with your own code is different from being in management. Yes, you produce things, and you have things to show for it, but because of the sometimes-nebulous feel of project management, you sometimes get the itch to work.

Worse is when you know that you can get something done faster than anyone else can. Sometimes this is true because you know more about what's going on, and explaining will take nearly as long as just doing it would. Sometimes it's true because you are more suited to the task than anyone else. However, unless you are the only person who can do this task, resist every urge to avoid delegating.

Every time you do a task instead of delegating, you take time away from your real job. You only have a limited amount of time to work in, so you shouldn't squander it doing someone else's work. It doesn't feel right, but it's absolutely necessary.

Also, if you do a task because you don't want to take time to explain it, when it needs to be fixed, modified, or replaced in the future, nobody else will know how to do it. Your job is to ensure the long-term success of the project, so you have to set up a system in which the project can even run for a period of time without you. If you insist on doing too much of the work by yourself, if you get sick, the project stops.

The worst part is that it's almost impossible to reasonably convince someone to delegate. I've trained several people over time, some who were my bosses, some who were my associate producers, and some who just needed some advice. I always hear the same things: "Well, by the time I finish explaining it, I've spent half the time I would have just doing it;" or "Nobody's as good as me with this." It wasn't until years later that one of my associate producers finally admitted that I was right, and she needed to learn to let go.

ASAP

This is a simple but completely counterintuitive lesson, but perhaps the most important one I can teach you: **Any task with a date will be done before a task that needs to be done 'As Soon As Possible'**. What, you don't believe me? Take this example.

You are extremely busy this week. Your boss comes in and says he needs a demo of the project ready for tomorrow morning. Then he comes back a few minutes later and says that he needs you to update the web site as soon as possible. You can't possibly finish the demo before 8 PM tonight, and you

were planning on leaving no later than 6 PM. Do you update the web site?

As soon as possible just isn't an important measure, because you can always say, "It wasn't possible, because I had to get X, Y, and Z done by dates A, B, and C." Always assign a date if you need something done. If you needed it yesterday and every hour missed will cost you money, say you need it done before lunch. If that's not possible, negotiate, but don't let the person doing the task set the priority. That's your job.

In a nutshell

The project manager is the interface between the programmers and the people who have the money. Those people could be upper management, the software publisher, or your clients. These are the people who are paying the bills and who have absolutely no idea what they really want. The programmers are the people who know how to make exciting and/or useful software, but who have little to no patience for people who cause them to waste time writing code that will end up being thrown away because it's either useless or, worse, really useful but something that the person with the money no longer wants to use.

Another function of the project manager is to keep the client from losing faith in the ability of the project team to finish deadlines. If the client feels that the project will never be finished, then the project will eventually be cancelled. Methods of keeping the client faithful are by managing expectations, by mitigating risks, and by having a predictable schedule.

The best way to maintain a predictable schedule is by having an established software process. There are many misconceptions about software processes, and about what they provide to the development world. Having a mature software process will not get a project done faster. If a project is going

to take five years, no process in the world is going to speed it up - only acts of heroism or extremely talented individuals can do that. What a mature software process will do is keep you from believing that your 5-year project is a 6-month project. The software process is usually a trade-off, adding additional time up-front to a project in exchange for fewer revision cycles at the end of the project.

Conclusion

Project management isn't an easy, one-hour a week task. On the other hand, it's not insurmountable. With the proper tools and attention to detail, as well as an ability to talk to people without sounding condescending, the right person could easily make a transition from programmer to project manager.

Hopefully, this paper has helped you to determine if you want to be a project manager or not. If your skin crawls while reading through, and you don't think it's because of poor writing, then you might want to stick with coding, with perhaps a move to Software Architect or Lead Programmer, if such is your desire. If you read through and think, "That doesn't sound so bad, and at least I won't have to debug anymore," then Project Management might just be for you.