Surprise!

Now You're a

Software Project Manager

Bas De Baar

Surprise! Now You're A Software Project Manager

Bas De Baar

First Edition

Multi-Media Publications Inc.

Lakefield, Ontario

Surprise! Now You're a Software Project Manager

by Bas De Baar

Acquisitions Editor: Kevin Aguanno
Copy Editing: Josette Coppola
Typesetting: Tak Keung Sin
Cover Design: Cheung Hoi

Published by:
Multi-Media Publications Inc.
R.R. #4B, Lakefield, Ontario, Canada, K0L 2H0

http://www.mmpubs.com/

Published in Canada.

Library and Archives Canada Cataloguing in Publication

De Baar, Bas

Surprise! Now you're a software project manager [electronic format] / Bas De Baar. -- 1st ed.

Includes bibliographical references.
Ebook available in PDF, LIT, PRB and PDB format. Also available in print.
ISBN 1-895186-76-5 (Adobe PDF)
ISBN 1-895186-77-3 (Microsoft LIT)
ISBN 1-895186-79-X (Palm PDB)
ISBN 1-895186-78-1 (Mobipocket PRC)

1. Computer software--Development--Management.
2. Project management. I. Title.

QA76.76.D47D42 2006 005.1068'4 C2006-902757-9

Preface

Have you ever woken up thinking, *How did I get here?* I'm not referring to uncertainty about where you are physically but to concern over the status of your project activities. One morning, you had a delay in your software project on your hands. Thinking back, you know how the assignment fell behind schedule, as all assignments do, one day at a time. And the worst part is that you know how it could have been avoided, or at least kept from escalating into the situation you now face on this particular morning. You know what you could have done to prevent the delay, and yet you didn't do it.

If you have woken up like this on one or more mornings, we share some similar experiences. I know what to do. I know the best procedures by heart. But sometimes it

seems that the heart is a long way from the brain. Attempts are made, more then once, but there is always that morning.

This book started out as my personal vendetta against all mornings of waking up preoccupied with worries about a project. It first appeared a couple of years ago as a small book called *The Microwave Way to Software Project Management.* While writing the content of that original book, I kept one thing in mind: *if you had only one day to talk about software project management, what would you say?* I surely wouldn't be yapping about great graphs.

Some years have now gone by. I spent more time in the project trenches implementing information systems, and I got hooked on some new, emerging techniques that I thought might help me out. My interest was actually fired up by the debate between modernists who favored new agile approaches and traditionalists who supported the plan-driven method of tackling projects. I thought that both sides were right.

How can that be? Did I miss something? Is someone from one or the other group lying?

The reason for my confusion at that time lay in a footnote to the stories of the two camps: both approaches work well, under the right circumstances. But what makes up those circumstances?

As I'm a technically oriented person by education, my first reaction was to consider the difficulties of the system being constructed. Of course that is one important aspect to a project's success, but after taking in a deep breath

and a nice cup of coffee, it all came back to me. What really makes all the difference in a project? People.

Managing software projects in organizations is a people-intensive process. The fears and wishes of stakeholders determine the path that the project will follow. If you look at the subject from that point of view, the obvious becomes even clearer: every method or technique ever invented is an attempt to address a certain problem or reduce the risk that a specific event will take place. Progress reports provide managers with feedback on issues such as how the project is going, whether too much money is being spent, and how many delays are at hand. Putting a developer and a user close to each other is a way to prevent communication problems that can occur when a cold paper trail is used. And so on and so forth.

If different techniques solve different problems and each project is unique, can there be one correct way of managing a software project? Of course not, and that is exactly why I agreed with both agile and plan-driven supporters in our industry.

So I picked up the old word processor again and set out an updated goal for the book that now lies before you: *if you had only one weekend to talk about software project management, what would you say?* (You see, I added a full day in respect to the previous book.) I would tell you about stakeholders and how they operate in a project, I would tell you about risks, and I would tell you about the basic problems you try to solve as a software project manager.

In this book I explain how stakeholder analysis and risk management provide you with the means to analyze the project circumstances. After that, I show you how you can tailor a project approach that should at least cover the basic risks you must address to manage a successful project. You see, even if I have an entire weekend with you, I will try to avoid yapping about graphs. With Internet resources nowadays, you can find techniques and methods all over the place. The difficulty lies in knowing what you have to look for, and to know that you must have a clue about what problems you are trying to solve.

The content of this book was written by me as an individual, based upon my own ideas and experiences, and it therefore represents my own opinions. There is no connection whatsoever between this book and any company or organization.

Book Overview

In this section I provide a summary of the chapters in this book.

Chapter 1: Introduction

I once saw a program on public television that revealed how magic tricks are performed and explained how magicians saw a woman in half. Ever since then, David Copperfield bores me. Learning how acts of illusion are accomplished will shatter your dreams. As software projects take place in reality, dreaming is not an option, and therefore it's very

profitable to understand the workings behind the tricks of the stakeholders concerned in the project (if you find this a weird twist in a sentence, wait until you read the rest).

Stakeholders are all the people involved in a project: the customer, the supplier, the boss, the user, you name it. As a project manager, you have to deal with all of them, and, even worse, these people will determine if your software project will be a success or a failure. This first chapter describes an image that should be engraved in every software project manager's mind. If you see it, you will surely recognize it. It's short and simple, so much easier to keep in your head than the picture of a huge binder crammed with pages about various methods.

Chapter 2: Project Potion

In spite of what you might think, there is no one magical way to carry out software project management. There is no universal approach that works every time and on every occasion. Every project consists of different people and distinct circumstances. You should design the ideal approach for a certain project based upon all the methods and techniques that are available. You should mix and match, cut and paste from every approach that sounds reasonable, creating your own magical Project Potion in this way.

In this chapter I outline the basic process for designing a specific project approach, using stakeholder analysis and risk management as input and applying project strategy, project organization, and feedback mechanisms as the means to create the most suitable method.

Chapter 3: Stakeholder Analysis

From the start of this book to the very end, I keep on emphasizing the importance of stakeholders in and around a project, but to do something proactive with this understanding, you need to analyze the stakeholders. Yes, you have to put on your psychologist hat and play the shrink.

There are ways to identify all the stakeholders involved and gain some insights into what they want from the project and how they are most likely to operate. You can use this knowledge to your advantage, as it will shape both your project organization and the way you communicate with each and every one of the stakeholders.

Chapter 4: Risk Management

I cannot tell you what the future will bring. No one can predict what will happen with any certainty, although some people believe they have the power to do so. This whole project concept is based on assumptions and estimations, on guesses and hunches. Sometimes we are wrong, and while working on a project we have to deal with this as a fact of life. Enter *risk management*. A risk is the possibility of loss of some kind, and it's all about the potential for conditions to be different from what you believe them to be right now. Risk involves not only whatever can go wrong but also whatever may arise if the future brings something unexpected. "Dealing with Uncertainty" would be a good subtitle for this chapter.

Chapter 5: Tailoring the Approach

Using risk management and stakeholder analysis, you will collect a lot of information about the stakeholders and the uncertainties in general surrounding your project. This chapter goes back to the basic ingredients for a proper project approach and explains how you can put them together to handle the data collected in the previous chapters.

I then tell you how:

- Business priorities and uncertainties (risks) can be resolved by using the proper strategy and feedback with the stakeholders
- Insight into stakeholder interests can result in the optimum project organization and correct feedback mechanisms about the process and the product

Chapter 6: Getting the Requirements for the Product

Whatever you do or whatever you make, know all you can about doing it or making it. After an initial analysis, the global contours of the project are outlined by goals and scope. You should get more specific, though, and being explicit about the requirements related to the product is what this chapter is all about.

Chapter 7: Getting the Requirements for the Process

We could name this the "party-pooper" section: it is about the constraints that stakeholders put on the process, such as budgetary restrictions or time limits.

Chapter 8: Feedback on the Product

As a kid, I played this little game at school called *Telephone Line*. Twenty children huddled into a circle and one of us started by whispering a sentence into the ear of the neighboring kid, so that no one else could hear what was said. The message would be whispered from one child to the next in succession until it had come "'round circle." The fun of the game was in comparing what the last kid heard to what the first one originally said. Usually, the two sentences didn't even come close to sounding the same.

While the project is underway, the requirements stated in the beginning should be validated. This process addresses two points: are the initial requirements still valid, and are we meeting them? This chapter discusses the requirements set for the product part of the project.

Chapter 9: Feedback on the Process

In Holland, we have this huge (200 kilometer) ice-skating event, the Elf Steden Tocht, which in English means "Eleven Cities Tour." On the course there are a number of checkpoints that are in fact the eleven cities, and you have to get a stamp in each one. Getting these stamps is very

important, as the race is actually about passing all the checkpoints.

In a project, it's all about getting the approvals to keep on going. This chapter talks about providing feedback on the requirements set for the process part of the project. Are we still within time and budget limits, and are the project constraints still the same?

Audience

This is a book on software project management, so it is obviously intended for people who either manage software projects or have the ambition to do so in the near future. However, the less formal you are, the more suited this book is for you. I treat the subject in a direct manner, and I try to make the experience as entertaining as possible.

Experienced managers can use this book as a guide to a stakeholder-centered approach to software project management. Students, or project-managers-to-be, can use it as a crash course. It is written by someone who studied the concepts of project management methods, went to work as a project manager, and was totally confused by the differences between the methods and the realities. This book provides the glue that connects what you see in the real world with the techniques that are available to software engineering and project management professionals.

CHAPTER ONE

Introduction

A software project is a very curious thing. Consider what happens when a typical project is launched:

> A man sits at his desk on a Friday afternoon thinking, I want a new accounting system. He calls his contact at the IT department and informs him of his wishes. Just before he hangs up the phone, he adds, "Oh, and by the way, this means that you are the project manager for this assignment. It would be great if you could start on Monday. Have a nice weekend."
>
> The IT guy—let's call him Hank—stares at the phone he still holds in his hand. He is confused. What new accounting system? What project manager? What Monday? After his first six-pack later that evening, Hank is feeling more relaxed. He can do this. Get a system.

Install it. Nothing to it. He'll make some phone calls first thing on Monday morning and he'll have this project taken care of in no time. As he pops open another brewskie, Hank is feeling empowered.

By Monday, Hank has had the whole weekend to think things over and has decided to contact the accounting department first to get a better idea of what they want. Will just any accounting system serve the purpose, or do they need a specific kind? Who knows?

That Monday morning doesn't go exactly as Hank had planned. He ends up sitting in a room for four hours listening to five employees of the accounting department. He has no clue as to what they are talking about. All he gets from this meeting are the names of five other departments that also use the current accounting system, the names of seven external companies that exchange information with this system, and three huge binders containing memos, short documents, Xeroxes, and handwritten letters explaining "exactly" what the accounting department needs.

Hank spends the remainder of Monday making brief contact with the five other departments and seven other companies whose names he has gotten from the accounting department. After this short round of communications, he can add another twenty names to the list of those who have something to do with the new accounting system.

> Reflecting back on these developments later that evening,Hank considers taking a different approach the next day. He'll look into the current system and see what it does. He'll talk a little with his buddies in the IT department to see if they can help him get on the right track.
>
> Tuesday makes Hank's state of mind even worse. From the IT department he gets two additional binders with policies describing which technology and architecture the new system should use, which documentation should be included for the support people, and lots more. Oh, and he can add to his list another ten names of people who have something to say about the new system.
>
> It started out so simple: one man, one statement. After only two days, Hank has a list of over forty people he has to involve and five binders stuffed with what looks like requirements.

I'd like to add that this story is inspired by true events. I have to admit that this case is a little extreme; however, suppose you got such a call on a Friday afternoon. You would have just one weekend for preparation. What should you know for your job as a software project manager? How to make a Gantt chart? How to create large sheets?

It is my opinion that you would be helped most by getting a rundown on the basics, the essence of a software project. If you know the dynamics of a project and how people operate and why, you have almost 80% already

covered. You can keep up with the technical stuff as you go along. If you know the "why" of a problem, the "how to solve it" gets a lot easier.

The essential problem of running a project is that you have multiple persons to satisfy, all of them having their own interests in the project, which are not always in perfect harmony with yours. The project manager is faced with the challenge of keeping everyone happy, but often his resources are scarce. Money, time, and people are limited, and in the end a software project is a game of setting priorities and negotiating trade-offs.

Did you notice that I didn't mention even one type of chart in the previous paragraph?

To start you off in the right direction, I talk first in this chapter about the essential mindset of a software project manager. Every project is different, but your primary mindset should remain the same.

In the remainder of this chapter, l address the following two subjects to prepare you for making judgments later on about available methods and techniques (for example, those on the Internet):

- General aspects of methods and some notions about project management and software development processes
- The big question: is there a universal method for all projects? (uh . . . *no*)

Mindset of the Software Project Manager

The thrill of having tickets for the opening night of Andrew Lloyd Webber's new play on Broadway is hard to compare with attending a performance on Amateur Night in our local town hall. Both are plays and both have actors who recite a series of lines to an audience. But ticket prices differ, and the differences in the quality of the players' performances and the stage props may represent quite a gap in relative value. However, is there a difference in terms of the audience's appreciation? Is the evening of those who went to Broadway more memorable than the night of those who didn't cross our county line?

Most people would probably argue that it depends on the expectations of the audience. If theatergoers pay a bundle of money just to see some famous actor, then they should visit Broadway. If all they want to see is a group of folks having some fun as actors, then Amateur Night is surely the way to go. The staging should fit the expectations of the audience, and so all is well.

A project has some aspects common to plays (and not just tragedies). Some tricks are performed by members of a project team (the players) and are closely observed by people with stakes in the project, the stakeholders (the audience). As long as the players perform as expected by the audience, everybody is happy. The principles presented in this section aim at achieving precisely that balance of anticipation and satisfaction. This book covers the subject of

software projects, and any project that has a larger, software-related element to it can be categorized as such.

One of the central principles introduced in this book is the *flow of the stakes*, which should be the mindset of every project manager. You should "know the flow" and have it mentally imprinted on your brain throughout the project. This is the best way to learn the expectations of all the stakeholders, integrate these different expectations, and ensure that all these people know that their expectations are being met. The main function of this section is to make you aware of this principle of flow that runs throughout every project, but it is not my intention to provide an in-depth analysis of every aspect. Although I try to dig that deeply in later sections, the true depth of flow can only be viewed in practice. Benjamin Hoff reminds us that this is true of many things in life:

> A basic principle of Lao-tse's teaching was that this Way of the Universe could not be adequately described in words, and that it would be insulting both to its unlimited powers and to the intelligent human mind to attempt to do so.[1]

This is perhaps a bit much, but it gives you a certain idea. Think of the flow as the apotheosis of this section, but first we must build up some context.

The Software Project Manager's Problem

The management of a software project is really an art; it's a trick that's difficult to master. The difficulty lies in the central problem the software manager is faced with, appropriately named "the software project manager's problem,"[2] as explained by Barry W. Boehm and Rony Ross. They believe that everyone affected by the project, directly or indirectly, has something to say, again directly or indirectly, and will do so. All of them want to get the best from this project for themselves personally or for their (part of the) organization. It's the job of the software project manager to see that everyone gets what he or she wants, in one way or another. He has to 'make everyone a winner."

In this respect, the role of the project manager becomes that of a negotiator. The customer always wants to have it all for free, the user wants to have the greatest functionality, and the programmer doesn't want to document anything but wants to use the coolest compilers. The software project manager has to make them all happy.

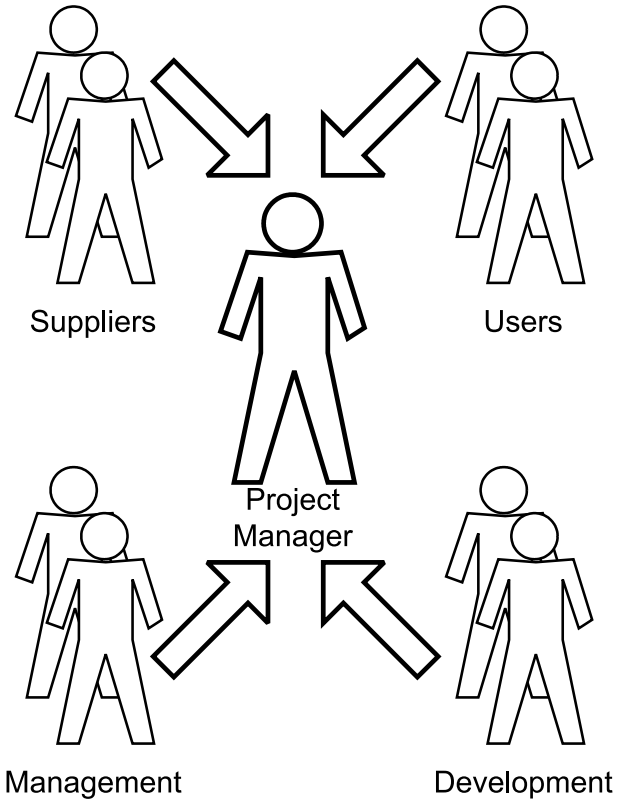

Figure 1-1: Software Project Manager Stuck in the Middle with All the Stakeholders

Focus on the Stakeholders

The entire process of software project management is strongly stakeholder-driven. It's their wishes, fears, dreams—their *stakes*—that determine the course of the project. You have to handle a project to really grasp the impact of people on your endeavor. You have to "live" a project to know the force of political games and power trips. You have to lead a team to cutover under time pressures to appreciate the constructive

power of motivated people or the destructive power of demotivated team members. I can keep telling you that people represent the most important aspect of a project, and I can quote statistics or research to convince you that the human factor is the only real concern of a project manager. But you will not really understand this concept unless you have been there yourself.

Once you *have* been there yourself, you'll realize that all the preparation and knowledge in the world will not guarantee success, because there's always that human element, the inescapable fact that people are involved. And in a project, it's the *people* that are the main cause of problems. Time schedules, financial projections, and software goals may be abstractions, but it's the flesh-and-blood people whose work determines your project's status. It's the programmer that misses a deadline and holds up everyone else, it's the financial manager that goes berserk if you can't produce some good budgetary indications, and it's the key user that doesn't give a darn but didn't tell you about his dismal lack of motivation; these are the folks who can cause serious trouble. Issuing the best procedures doesn't mean that people will live by them. You really have to be in the asylum to understand the lunacy of it all. If you haven't been there yet, take this important advice: *focus on the stakeholders*. Be guided by their fears and their wishes.

A *stakeholder* can be a project team member, an employee of the user organization, or a senior manager. It can be virtually anyone, as long as that person has something to do with the project.

I know that this socio/psycho stuff is not the favorite cup of tea among technocrats, but being unknown shouldn't make something scary. In essence, it's even easy to understand. You are part of it. If you are human, you can practice on yourself and try things out on your spouse and kids. Look around your office and try to think about how the situation there is different from the one at home with your kids.

First of all, your children are more transparent than the people at work. They communicate directly what they want, and if they try some power trip on you, you immediately recognize it for what it is. "I want ice cream right now" is pretty clear. If they don't get this very nice cold snack, kids can use crying as a means to get their way. And you may cave in, perhaps just to stop the annoying sound and, of course, the embarrassment of walking through the mall with your yodeling kid.

Let's start to focus on project management.

Your boss or your employees will not be this explicit and predictable. They may try some mind games to manipulate you or (let's avoid being negative) they will have the best intentions, ones you haven't figured out yet.

But in essence, they have the same mechanisms as you and your children. People do what they do to realize their dreams or to avoid having their nightmares come true; in other words, to achieve their wishes or to escape their fears. Within the context of projects, people are the stakeholders.

What people (and stakeholders) really, really, really want are what can be termed their interests or, as I sometimes call them, their stakes (hence the name "stakeholder"; I will use the terms "stakes" and "interests" interchangeably). With fears there is a stake to lose, and with wishes there is something to gain.

In this context, I consider interests as the aspects that drive people. Before you start drawing your "interest evaluation diagram" or some other neat tool or technique, be aware that in general these interests are hardly ever communicated. It's pure mind stuff, all inside the head of the owner. A four-year-old boy may share his true interests with you, but the fifty-year-old graying accountant will tell you nothing.

If no one will tell you anything, what is the point?

People will tell you *something* if you ask them. They will tell you they want an ice cream cone, a new hyperspeed Internet uplink, or a new financial software package. In essence, they tell you what they expect. It is a statement created by *themselves* about a desired situation: their *expectations*.

If I emphasize that expectations are a one-sided communication, then there must be something else as well; enter *requirements*. Requirements are a set of statements negotiated among a group of people. They can be the original expectations, if all agree on the statement itself, but more often than not, requirements consist of some consensus of conflicting expectations.

Requirements are always clearly defined and describe a state that is desired. This is in contrast to expectations, which are generally vague and abstractly formulated.

Interest/Stake	What people really want, not communicated	
Expectation	What people expect or want, communicated	
Requirement	What should be done, negotiated, and communicated	
Interest/Stake	I want that ice cream	Will not lose face again with a delayed project on my hands
Expectation	If I cry and scream, I will get my ice Cream	Will be finished before the start of Q2 next year
Requirement	If you are sweet, you will get ice cream	Total cutover to production finalized before 1/1/20xx

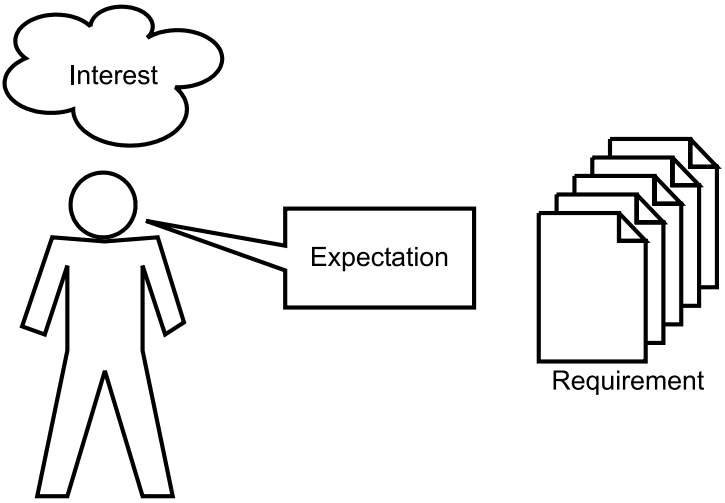

Figure 1-2: People Have Interests and Communicate Expectations that Result in Requirements

Here's an oversimplified example: an interest of a programmer is "to be involved in way cool new technologies like my brother-in-law is" (let's say Java). The corresponding expectation could be, "The system can only

be programmed in Java." The expectation can be stated along other technical arguments, but it's only the interest mentioned that caused it to appear. If everyone agrees on this "fact," then "programmed in Java" can become a requirement.

So the project manager has this project, surrounded by requirements that must be met. But remind yourself: they are not the stakes, they are not the crown jewels that may not be touched. So you can mess with the requirements, can't you?

What Is It Worth?

It sounds simple, but getting the expectations is one thing and discovering their corresponding stakes is another. Why bother anyway? What is it worth? A lot. As mentioned earlier, you can't effectively change the stakes, but you can alter the set of requirements as long as the new requirements continue to support the stakes. In this way, there is room to negotiate a set of requirements for the project that poses no conflict, matches the stakes, and thus makes everyone a winner! Right?

Now let's take it one step back. A stakeholder formulates an expectation for the software project; for example, senior management states, "The project should be finished before the end of August." The project manager then has to deal with this time frame. When this deadline is no problem, he can rest assured. However, it's a software project, so the deadline will be a problem. The way to handle it is to get some information on the stakes that prompted this requirement to be formulated in the first place.

Perhaps it's the old "I don't want to lose face when my projects get delayed" concern. That being the case, the project manager can offer alternatives that don't violate the stake, like keeping the deadline but postponing a subsystem. Chances are good that alternative requirements that keep supporting the stakes will be accepted—maybe not easily, but project managers should do something to earn their money.

An example taken from Frederick P. Brooks Jr. illustrates this relationship between stakes and requirements:

> [...] the reluctance to document designs is not due merely to laziness or time pressure. Instead it comes from the designer's reluctance to commit himself to the defense of decisions which he knows to be tentative. By documenting a design, the designer exposes himself to the criticisms of everyone, and must be able to defend everything he writes. If the organizational structure is threatening in any way, nothing is going to be documented until it's completely defensible.[3]

Stakeholders during the Project

The first influence of the stakeholders on the project is usually the most influential, and it occurs at the start of the project. The demands and constraints issued on the process and on the products that are produced set the stage for the entire duration of the software project. The primary task of the project manager should be to reassure the stakeholders that what they communicate is heard and that their stakes are taken care of. Throughout the entire project, the project manager's

task consists of giving stakeholders feedback on the state of their requirements.

This feedback can take the following forms:

- Tests
- Test results
- Prototypes
- Reports
- Evaluations
- Plans
- Benchmarks

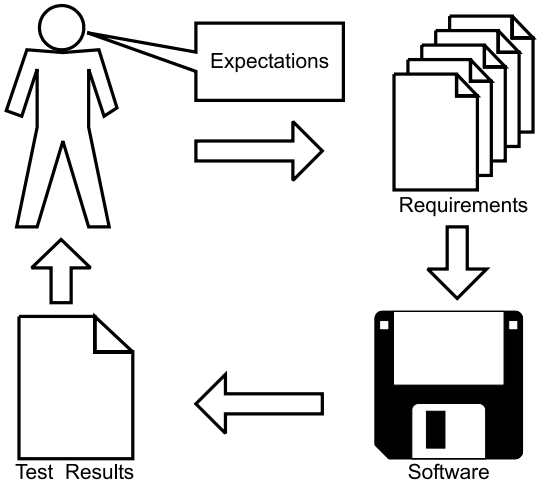

Figure 1-3: Test Results as Feedback on How Requirements Are Translated to Software

This part is essential but easily forgotten. If the project manager does not keep giving feedback, the slightest hint (or rumor) of not sticking to the stakes may set a stakeholder on the warpath against the once so happy project manager.

The Flow of the Stakes

Having said all this, where does it leave the software project manager? In order to have a "happy project," a software project manager should respect the flow of the stakes, as illustrated in the diagram below, and must ensure that stakes go full cycle.

Let's describe this flow in a few steps:

1. Stakeholders have stakes.
2. Stakeholders communicate their stakes by expressing their expectations, which are more formally defined by means of requirements to the process or product.
3. Project management should make every stakeholder a winner by accepting and inventing requirements that continually satisfy the stakes of individual stakeholders and do not conflict with the general process or the product.
4. Project management should give continuous feedback to the stakeholders on the state of the stakes.
5. Based upon this feedback, the expectations and requirements might change, and in this way a new cycle begins.

You Must Know This Flow!

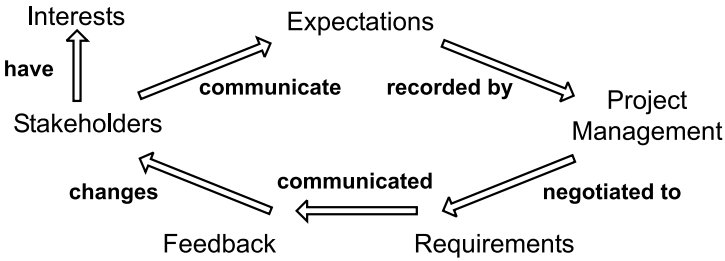

Figure 1-4: The Flow of the Stakes

So now you know the flow. For the coming days, try to fit all that surrounds you into this image of the flow and view the world through these flow-sensitive glasses. As you already might have guessed, the concepts behind the flow are not limited to software projects.

The principles for inventing win-win conditions are discussed in general books like *Getting to Yes* by Roger Fisher and William Ury, who even mention its use in hostage situations.[4] The indirect communication of stakes can even be found in John Gray's *Men Are from Mars, Women Are from Venus.*[5] If a girl tells her boyfriend, "I want you to listen to my problem" (requirement to the process), the stake of the girl typically would be the reduction of her stress by talking about the subject. Unfortunately, the boy's typical interpretation of her stake would be, "She wants me to fix the problem." To understand exactly how this process works, just read the book.

The point I want to make is that you don't have to be currently working on a software project to try The Flow. Try to

view as much of life as possible with your glasses colored by the concepts presented. I'm not claiming these concepts should guide your normal life; it is just an exercise! This is still just a book on software project management.

Not Everyone Will Be a Winner

I know this goes against everything I wrote previously, but in reality you will find out that you cannot make all people as happy as they would like to be. You will not be able to satisfy everyone's needs or some people will try to frustrate the project for the simple reason that they just feel like doing so.

However, try to be positive as long as possible. Go the extra mile to make everyone a winner, but be prepared to encounter some negative souls, people who love to play corporate games. You will meet them, but we will cover them in this book, so don't worry.

Don't Forget to Include Yourself!

After you have been "into" the project for a while, you will have read all the paperwork. You will have spoken with a lot of people. You will have asked, "What do you want?" and "What can I do?" When that time comes, it's time for your gut feeling. Do you trust the statements made by the stakeholders? Do you think top management will stick by you? Do you honestly believe the project has a chance to succeed? Now is the time to know. It is late, but you can still get off the train. Once this window of opportunity slams shut, it's your butt

that's stuck hanging over the sill. Whenever this project is mentioned, you're the one that people will remember.

It makes no sense to associate yourself with a project if you think it's a *Titanic* or *Mission Impossible IV*. Don't make the mistake of thinking that your efforts will be appreciated even if everything goes down the drain. Failure is an easy thing to remember. The true professionals know their limitations, and the responsible course of action is to either configure the project in such a way that it stands a chance of success or stamp it "return to sender."

A quote is always great, but this one from U.S. Judge Thomas Renfield Jackson's statement to Microsoft's legal counsel during a monopoly trial is in this context almost perfect:

> The code of tribal wisdom says that when you discover you are riding a dead horse, the best strategy is to dismount. In law firms, we often try other strategies with dead horses, including the following: buying a stronger whip; changing riders; saying things like, 'this is the way we have always ridden this horse'; appointing a committee to study the horse; arranging to visit other firms to see how they ride dead horses; increasing the standards to ride dead horses; declaring that the horse is better, faster, and cheaper dead; and finally, harnessing several dead horses together for increased speed.[6]

If you take on a project, don't forget to include your own stakes and be a winner yourself! After all, you are a stakeholder as well!

And if you are a winner and higher management is committed to you as a project manager for the project, make sure they send a memo or an e-mail to all stakeholders, declaring you the head honcho, the big cheese, the ruler of your universe, or something more office-like.

Processes: Project Management versus Software Development

Reality is too difficult for us mortals to comprehend. You think I am kidding you? Try to consider every aspect of the world at the same time. Thinking of the earth may cause you to have a visual image of our planet taken from outer space. Focusing on a particular country may cause your brain to zoom in on one spot. At the same moment you zoom in, you neglect the rest of the earth you just had in your brain. Let's face it: we are not capable of seeing everything.

This is a fact of life, so don't get into a sweat about it. Creative as they are, our brains have found their own ways to deal with it: mostly, by simply neglecting information. As with the example about the earth, our minds can zoom in and zoom out. Another strategy of our brains is to focus on only certain aspects. Think about a nation's geography. Now think about its economics. Most people would get very different visual impressions from these two thoughts. With the first one, I get a mental picture of a topographical map. For the second one, I get images of banks, companies, and the government. We can have different levels of complexity and simplify the levels we use by neglecting certain details.

Why am I bothering you with this? I just want to make you 100% aware that what you are about to read, or ever will read, on the subject of software project management is a representation of a certain level that neglects information about real life situations. Underlying every book on this subject is a specific idea of the real world that explains things to you neatly and cleanly, and (hopefully) in an understandable way. To convey or accept this idea, we simplify and we neglect.

Again, this is no real problem. Heck, we don't even have a choice. Just always be conscious of the assumptions that are made when information is presented in a certain way; always inform yourself about the aspects that are neglected. Project reality will kick you in the back if you don't.

As a software project manager just beginning a job, you want to know what you should do. What should I do, when should I do it, what should other people do, and when? The simplest way to illustrate this phase is by the use of a process description. It will set out a sequence of steps to follow. We in the software industry are suckers for boxes and graphs, so here they are for you:

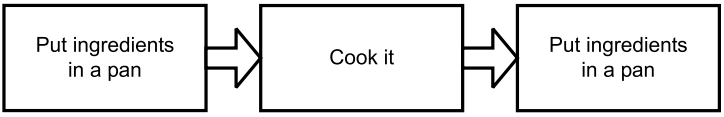

Figure 1-5: A Cooking Process Flow

My process for having a great meal: first, you mix all your ingredients in a pan, next you cook the mixture, and when it is ready you can finally eat it. It's straightforward and simple, with nothing to argue about. Of course, I am assuming you have the ingredients and even a pan. If you want to have a more detailed description, you can include a box before the first one, indicating that you should purchase everything you need before starting to put it in a pan.

There are also no instructions on how to cook it. Of course, it would be very difficult if we just kept on referring to "ingredients" without being more explicit. For more specific directions, we should narrow down what type of meal we are cooking.

I could go on and on about this, but I will not do that to you. Just notice that even a simple process leaves a lot of questions unanswered.

An extra element we introduce in this way of looking at a process is the familiar computer input-output aspect. If a box indicates an activity, it would be nice if this activity resulted in something, the output. And in order to do something, most of the time you need some magic powder to perform it, the input.

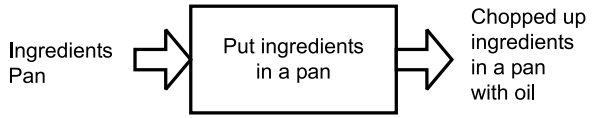

Figure 1-6: Cooking Process with Input and Output

I will let the image speak for itself.

Throughout this book and other texts on the subject, you will note a distinction between the process and the product, between the activities and the items that are produced. A *plan* is a product, like a piece of paper or an MS project file. The process that creates it is called a *planning*.

When focusing on software project management, there are actually two main types of processes involved: the actual creation of the software and the management of the project—the software development process and the project management process, obviously.

If you look at the day-to-day operation of a project, it is sometimes hard to tell that two different processes are involved because they are so closely interconnected. A project management process without a software development process makes no sense (there would be nothing for the project to produce, other than plans and reports), and a software development process without project management would only result in a never-ending, ever-changing process that doesn't fit within the constraints and goals for the project

I make note of this distinction because the failure to appreciate it is sometimes a source of confusion when discussing approaches to software project management. Material written by pure software development people can focus on the software development process and offer only a footnote about the management process, and pure project managers often tell you that a project is a project and so it doesn't matter if you are building a bridge, a spacecraft, or a new accounting system.

In theory, everyone is right; just think about my opening discussion about the brain's way of selectively focusing on or neglecting certain information. In my experience, however, to be an effective software project manager, you have to know and incorporate both the software development process and the project management process.

If you know nothing about software development, it is of course still possible to run the project, but you would have no idea what you were talking about. You would have no clue about what sequence of steps are effective, what the products really are, what problems can occur, and how to tackle them. Software development is still a mystical art. Even though it may seem structured and predictable, sometimes it is not, and ignorance of this fact is what wrecks a project most of the time.

On the other hand, considering project management as merely a footnote may deprive you of great techniques and tips you can pick up from other industries. Although software projects are in their own league, other industries may face similar problems and create effective management solutions for them.

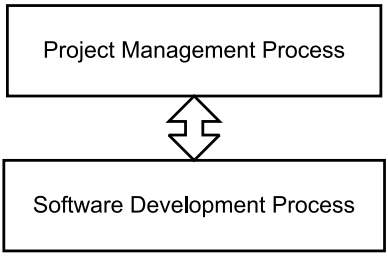

Figure 1-7: Project Management Process versus Software Development Process

In this book I focus on the project management process, but I also beg, borrow, or steal time for software development where needed. And believe me, we will need it a lot, so it is also fair to say that I present an integrated version. Call it whatever sounds more appropriate to you.

One Size Doesn't Fit All

There is no one way to make great lasagna. Of course, the basic idea is always the same, but depending on your tastes, cooking skills, and available ingredients, you can create a lot of variations. Some people use only fresh ingredients, make the pasta from scratch, and spend several hours in the kitchen. I buy prefabricated ingredients and whip up the meal in several minutes. Both dishes are lasagna, but they are very different.

Comparing lasagna to software projects is a stretch, but I'll take my chances. Although some books may lead you to believe that there is only one way to perform a software project, believe me, there isn't. There are many more ways. Just look at the number of methods that are available: Prince2, PMP, Scrum, or Lean Project Management. If you combine these possibilities with the various software development methods that exist, you have plenty of potential combinations that may be useful.

In recent years, there have been some heated debates within the industry about the "right" method of software project management. In one corner you have the traditional *plan-driven* camp, and in the other you find the new rebels,

the *agile* supporters. In essence, the first group believes you should define every step that has to be performed in detail up front: the actual task, the timelines, the organization, and the procedures that should be followed. This will increase the predictability, stability, and high assurance of the process and the products it produces. You design a plan, you issue the orders, and you make sure everybody sticks to this original strategy.

The downside of this plan-driven approach is that a software project is never as predictable as everyone hopes, so the cost for this enforced structure is a higher overhead (a lot of time is spent on creating supporting products like plans, reports, and documentation, stuff that is rarely if ever used in the end result). Interest in reducing wasted resources sparked some new ideas about what was the right approach for software projects.

Agile methods differ from plan-driven approaches by assuming that conditions will continually change and addressing this change through the concept of "just enough": just enough structure in the process to keep things going in the right direction and just enough supporting documents like short-range plans and brief reports to meet the current needs. According to this agile philosophy, it makes no sense to attempt to prescribe the future in plans, as the situation will change anyway. Instead, agile managers focus on creating a process that easily adapts to the situation.

Each camp claims that its own approach works better than the other, and both philosophies have great worth. It's hard to argue reasonably against the value of predictability

and stability, but adaptiveness also sounds like a very wise strategy. Is a defender of one or another group lying? I will definitely say no. Every method has its merits and every approach can lay claim to a sample project where it worked very effectively. But not every project is the same. Each one involves different people, different tools, different end results, different project constraints, and different circumstances.

Different circumstances require different approaches. If you need creativity to solve a problem or to create a design, you need an easy-going, stimulating approach; if you are racing a deadline to achieve production, a rigid, centralized, and controlled environment is more likely the better way to go. Depending on the environment and general circumstances, a project manager should construct a process and organization that serve him or her best. In the ideal situation, you should be able to cut and paste together a method that suits your particular situation, using approaches that have been proven to work well in the exact situation you find yourself in.

A key question is: what makes up the circumstances of a project?

First of all, the desired end result of the project is a main factor. Creating a navigational system for a spacecraft would be very different from designing an application to organize your recipes. Not only would the application be different in size and complexity, but the requirements on stability and reliability would differ enormously.

Secondly, without hesitation I would say that people represent a major element in the circumstances of a project. In my personal experience, all major problems concerning projects are caused by human factors. It may be just the level of skill or knowledge of a project team member, but don't dismiss too quickly the impact of cultural and political influences on your project.

To give you a better idea, Barry W. Boehm and Richard Turner have defined five dimensions that should affect a method selection:

> **Criticality**: If the end results fail, what would be the cost? The scale ranges from a simple loss of comfort to the loss of many lives.
>
> **Personnel**: The level of skill of the project team members
>
> **Size**: The number of personnel involved
>
> **Dynamism**: The number of requirement changes per month (What is the level of people's ability to define the requirements?)
>
> **Culture**: Are people accustomed to handling change and taking their own initiative, or are people used to orderliness and having their work laid out in plans for them?[7]

I list these dimensions not to imply that they are the only ones but to illustrate what kind of aspects can influence your choice of a certain approach. And you have to choose. Just

as the right approach may be more effective, it also holds true that the wrong method can sink your project to the bottom of the corporate ocean.

Letting people make their own decisions in an organization whose culture does not encourage such independence may leave you with indecisive team members or, even worse, decisions made solely on the basis of popularity and social acceptance rather than project needs. It may seem that more roads lead to Rome, but don't forget that it is actually a maze.