Chapter 1

Examining the Big Picture of Project Management

In This Chapter

- ▶ Defining what a software project is
- ▶ Examining project management attributes
- ▶ Starting and finishing a software project
- ▶ Dealing with software project nuances
- ▶ Leading and managing project teams

Here's a tough decision for you: Manage a project to create a new piece of software that can make or break your entire organization, or jump from an airplane with a parachute that may — or may not — function. For some project managers the decision is the same either way.

But not for you. At least you're on the right track to capture, improve, and successfully lead your projects to completion.

The adrenaline rush in skydiving (and in project management) may not be at the same level, but the butterflies in your stomach definitely are. There's really one secret to skydiving and it's the same secret to successful project management. (No, it's not "don't do it.") The key to successful software project management and skydiving is preparation.

Many projects fail at the beginning rather than the end. After you do the prep work, you must execute your plan, take control of your project, and ultimately bring it to its natural (and successful) conclusion.

Defining Software Projects

Software project management is a type of project management that focuses specifically on creating or updating software. Just as there are billions of ice cream flavors, there are billions of types of software. Project managers, effective ones, can lick them both.

A project, technically, is a temporary endeavor to create a unique product or service. For some people, everything is a project; for others, projects are special, lofty activities that occur infrequently. A project is a unique entity. In other words, the creation of a new application is unique, whereas the maintenance and day-to-day support of an existing application is not so unique. Projects can have many attributes:

- They change or improve environments in organizations.
- They get things done.
- They are unique from other work.
- They have a defined start and end date.
- They require resources and time.
- They solve problems.
- They seize opportunities.
- They are sometimes challenging.

Defining Software Project Management

For some people, *project management* is just a stack of work doled out to a group of people by a goober called the project manager. For other folks, project management is a foggy, scary science directed by a different goober with a slide ruler. And for others still, a project manager is a goober that touts formulas, certifications, and facts without ever really getting things done.

But in effective project management there ain't no room for goobers. Effective project management centers on the serious business of getting work done on time and within budget while meeting customer expectations. Effective project management is about accomplishment, leadership, and owning the project scope. It's an incredible feeling to sign off on the project and know that you and your project team contributed to the project's success.

TIP

Management is concerned with one thing: results.

Project management involves coordinating people, vendors, and resources. Project management requires excellent communication skills, a strong will to protect the project scope, and leadership skills to enforce quality throughout the project work.

According to the Project Management Institute (`www.pmi.org`), the defining resource on all things related to project management, project management is centered on nine *knowledge areas.* Events in each knowledge area affect what happens in the other eight knowledge areas. Table 1-1 gives you the lowdown.

Table 1-1 The Nine Project Management Knowledge Areas

Knowledge Area	***What It Does***
Project Scope Management	Controlling the planning, execution, and content of the project is essential. You need to pay special attention to both project and product scope so that the software you end up with is what you intended to make in the first place.
Project Time Management	Managing everything that affects the project's schedule is crucial. Who wants tax software that comes out on April 16?
Project Cost Management	Projects cost money, and this knowledge area centers on cost estimating, budgeting, and control.
Project Quality Management	No project is a good project if the deliverable stinks. Quality doesn't happen by accident, so this knowledge area works to ensure that the product you are producing is a quality product that meets customer expectations.
Project Human Resources Management	The members of the project team must get their work done. Hiring or assigning people who are competent and managing them well are at the center of this knowledge area.
Project Communications Management	Project managers spend 90 percent of their time communicating. Communications management focuses on who needs what information — and when.

(continued)

Table 1-1 ***(continued)***

Knowledge Area	***What It Does***
Project Risk Management	This knowledge area is about avoiding doom. The focus is on how to anticipate risks, handle them when they arise, and take advantage of the opportunities that can help a project.
Project Procurement Management	Sometimes during the course of your software project, you may be required to work with vendors to purchase goods and/or services. You may even be the vendor that someone else is contracting for their project. This knowledge area is concerned with the processes to create vendor contracts and to purchase goods and services.
Project Integration Management	What happens in one knowledge area affects attributes of the other knowledge areas. The ninth knowledge area is fan-freakin-tastic because its purpose is to ensure the coordination of all the other knowledge areas.

Comparing Projects and Operations

There is a distinct difference between projects and operations. *Operations* are the day-to-day activities that your organization does. For example, a car manufacturer makes cars. An airline flies people from one city to another. A help desk supports technical solutions. Within each of these companies reside various departments working on projects that enable operations to function. A project at an automobile manufacturer might be to design a new sports car. The car manufacturer's operations involve manufacturing that design again and again.

Software creation is special. Imagine you have customers around the world who want you to create a piece of software that helps them keep track of sports statistics. This is your new business — you create sports stat software and you're a gazillionaire.

Each flavor of the software you create could be a separate project; your company has software for baseball stats, football stats, soccer stats, field hockey stats, and everyone's favorite sport, water polo stats. Each project has its own requirements, its own purpose, its own budget, and its own project manager and project team. Each project has its own resources, schedule, budgets, and goals.

Your day-to-day support of the software, the sales of the software, the credit card purchases, and the delivery of millions in cash to your bank account are all part of operations.

Some companies have changed their approach to business by treating all of their operations as projects. This microscale of their enterprise, where every activity is part of a project and all projects contribute to the betterment of the organization, is called *management by projects.*

Examining Project Constraints

A *constraint* is anything that restricts the project manager's options. Constraints are requirements, confines, or, if you're a glass-is-half-empty kind of person, prison walls. Constraints can include

- Resource constraints such as a team member being assigned to too many concurrent projects
- Tight deadlines
- Budgetary limitations
- Government regulations
- Limitations of software
- Scope limitation, such as being required to use a particular existing interface
- Hardware requirements
- Anything else that restricts your options

Understanding Universal Constraints (Time, Cost, and Scope)

The three universal project constraints you will *always* face are

- **Time:** Time constraints may range from a reasonable schedule to an impossibly short timeframe that can't budge because the product simply *must* be on shelves by September 15 (never mind that September 15 was last week).

- **Cost:** Cost constraints are the usual budgetary restrictions that you expect. ("Here's a nickel. Make it happen.")
- **Scope:** Sometimes scope is a no-brainer (you're working on the 700th rev of Acme Wizware to fix a bug). On the other hand, scope can be a bit trickier if you're dealing with an executive who isn't sure what he wants.

  We guarantee that executives will always know when a product is needed and how much money you can have to get it done. If there's a single area that the big-wigs won't have nailed down, it's scope.

These three constraints make up what we affectionately refer to as the somewhat inflexible-sounding nickname the *Iron Triangle of project management.* Check out Figure 1-1. Each side of the Iron Triangle represents one of the triple constraints. For a project to be successful, each side must remain in balance with the other two. You will read more about project constraints in Chapter 3.

In order to achieve quality in the project deliverable, and in the management of the project, the Iron Triangle must remain balanced.

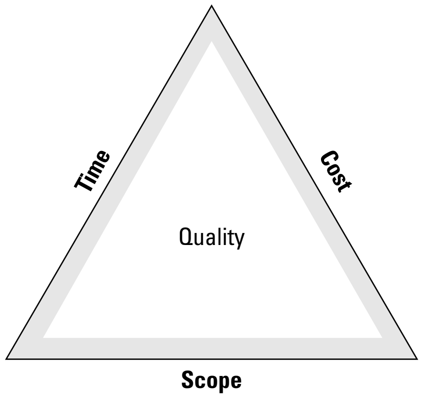

Figure 1-1: The Iron Triangle describes constraints that all projects must face.

For example, say your boss decides to add more stuff to the project scope (now instead of simply fixing a mathematical bug in your Wizware accounting software, you have to create a whole new feature in the software that edits photos and home movies). Even though your boss has changed the scope, you have to deliver more stuff within the same amount of time and with the same amount of cash, as Figure 1-2 depicts. You'll need more time, more money, or both for the triangle to remain equilateral.

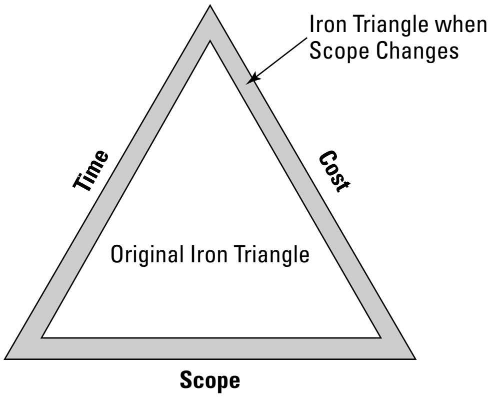

Figure 1-2: Increases to the project scope enlarge the Iron Triangle.

Managing time constraints

Time constraints are simply deadlines. You have a project to create a new piece of software within six months. Or there's an opportunity in the marketplace for a new application, but the window of opportunity is small, so you have no time to waste. Time can also be calculated as labor: Working or billable hours, processor speed, database consistency, and even network latency issues can be used to estimate time constraints.

REMEMBER

Introducing the law of diminishing returns

Time is time. Don't be fooled into thinking you can buy more time — no one can. You can buy more labor if you think it will help your team do more work faster, but that's not the same thing as adding time to a project. The *law of diminishing returns* dictates that adding labor doesn't exponentially increase productivity; in fact, at some point productivity can even go backwards (is that antiductivity?). When that happens, you've hit a plateau, and then everyone is sad because you just can't do more with the labor you have.

For a real-life example of the law of diminishing returns, consider that you may have two hard-working, experienced programmers working on a section of code. In your quest to finish the project on time, you add one more programmer to the mix. Now the programmers may be completing the code more quickly, and you're so excited that you decide to add six more programmers so that you can finish even sooner. You soon realize that although adding one programmer increased your productivity, adding six more only created chaos, with programmers stepping on each other's toes, inadvertently neutralizing each other's code, and creating a contentious environment. You reached the point of diminishing returns when you added six programmers.

Time constraints require more than just hitting a target deadline. Unavailable resources (your ace programmer is on vacation), skewed milestone targets within the project, conflicting versioning deadlines, and so on, all present constraints on the project's timeline. A time constraint is any factor or issue that changes or impacts the original timeframe of the project. (No time machines allowed in project management, sorry.)

Managing cost constraints

Cost constraints are easy to identify because they deal with cash money. Well, it's not always cash, but you get the idea; the miniscule funds in your project budget to complete the project work create a unique constraint. Your costs include computers and languages to code in, labor, and anything else you need to buy in order to get the job done.

For some folks the funds are *blue dollars,* departmental dollars that shift from one department to another based on the project costs. For other people the budget is a very real number in dollars and cents: Customers hire you to complete work for them; then they give you a satchel full of cash.

TIP

Projects almost always cost somebody something. Be sure to factor in hidden costs for labor, resources, computers, pizza, celebrations, training, bribes, and more. Just kidding about the bribes part. As far as you know.

Managing the scope

The third part of the Iron Triangle is the scope. There are two scopes within project management:

- **Product scope:** The product scope describes, lists, and categorizes all the features and components of the finished deliverable. This is what the customers see in their minds' eye.
- **Project scope:** This is where you focus. The project scope is all the required work, and only the required work, to create the project deliverable. The project scope focuses on work, activities, and progress to achieve the product scope. The project scope must be protected from unapproved changes because it dictates what the project team will do and what the end result of the project will be.

The product scope and the project scope are in love. The product scope kisses details in the project scope and the project scope returns the favor. It's romantic. Each scope depends on the other, and what happens in either scope affects the other. If there is disharmony between these two scopes, trouble is brewing.

REMEMBER

Delivering what's promised (and *only* what's promised)

In the Iron Triangle the project manager's concern is on the project scope — the project work. The project manager must direct the project team to do the required work, nothing more or less, to deliver exactly what the product scope calls for.

Nothing more? Shouldn't the project manager and the project team always deliver more than what was promised? No, no, no! This may shock you, but the job of the project manager and the project team is to deliver exactly what you and the customer have agreed to create.

Let me write that again so you don't think it's a typo: The project manager should deliver exactly what the customer expects.

You and the project stakeholders should define everything the project should deliver as soon as possible. When value-added changes are made after the project scope has been created, the analysis of these changes takes time and money and may impact the schedule.

We're not saying the project manager should hold back good designs, ideas, and incredible features that the customer may want and can use. We're saying that neither the project manager, nor any stakeholder, can arbitrarily add features to the software because doing so would be to change the project scope.

Controlling Scope Creep

Changes to the project scope can affect cost and time constraints, melting your Iron Triangle. The Iron Triangle is a key tool in project management and is ideal for negotiations with stakeholders. For example, if your stakeholder insists on adding software functionality to your project scope, you can use the Iron Triangle as a tool to explain that when you increase one side of the triangle (the scope side) the triangle is no longer in balance. To change the scope, you must change the cost or the schedule (or both) to keep the triangle balanced. The Iron Triangle is also a terrific tool to use in discussions with the project team, and to keep your own duties as project manager in alignment (see "Understanding Universal Constraints (Time, Cost, and Scope)," earlier in this chapter).

Unplanned changes to the project scope, sometimes called *scope creep,* are the little extras that expand the scope without reflecting the changes in the cost and time baselines. You'll notice from the graphic that with scope creep, the lines of the Iron Triangle are no longer even. See Figure 1-3.

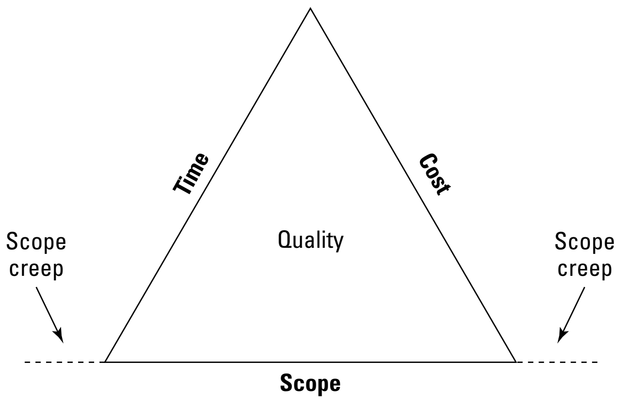

Figure 1-3: Scope creep is project poison that changes the alignment of the Iron Triangle.

The reason scope creep is so poisonous is because it can happen so easily, and so innocently. And yet, it can be so deadly. When the scope goes off track, time and funds are stolen from the original baselines. It's not as if extra money and time are magically added to the project to handle all the little extras. Balancing the three sides of the triangle ensures a high-quality final product.

Changes to the project scope should be controlled and managed through a *change control system,* which you can find out more about in Chapter 13. In essence, a change control system accommodates a process for documenting requested changes and requires obtaining appropriate approval for all requested project changes. The key is to avoid changes that are not directly approved or requested by the customer.

Making Sense of Project Success (Or Failure)

Most projects start with an optimistic attitude about creating a deliverable, keeping the customer happy, and making this the best software project ever. And then things (bad things) happen. The good projects end on time and as planned. Ah, paradise. We'd wager that these projects have three things in common:

- A leader who knows what he or she is doing
- A tight change control system (see Chapter 13)
- Team members who understand what the project is supposed to deliver and can therefore get results

Commonly, projects limp to the finish line, late, overbudget, and after crushing the morale of everyone involved. Done, but maybe not done well. These projects typically have three attributes:

- Poor requirements from the project customers
- Poor communications through the project manager
- Poor morale from the project team

The saddest of projects are the ones that never make it to the finish. This bunch misses deadlines, blows budgets, or experiences a radical change of scope so often that no one (not even the PM) knows exactly what the project should be creating anymore. Failed projects usually have some, if not all, of these attributes:

- No clear vision of what the project priorities are
- Lack of leadership from the project manager and/or sponsor
- A timid project manager
- Lack of autonomy for the project manager
- New resumes being typed in unison

Starting and Finishing Software Projects

All projects, from your software creations to building the bridge over the River Kwai, pass through five process groups as defined by the Project Management Institute. A *process group* is a mini life cycle that moves the project one step closer to completion. Process groups are cycles because the processes don't just happen once; they are repeated throughout the project as many times as needed.

TIP

Figure 1-4 demonstrates a sequence for process groups; the processes flow organically, in the order that best suits the needs of the project. Although we hope you don't have to keep repeating some of these stages, if your project isn't going according to plan you will have to do just that.

All projects, software and otherwise, go through these project management processes. Each of these project management processes has its nuances. We describe them in more detail in Chapter 2.

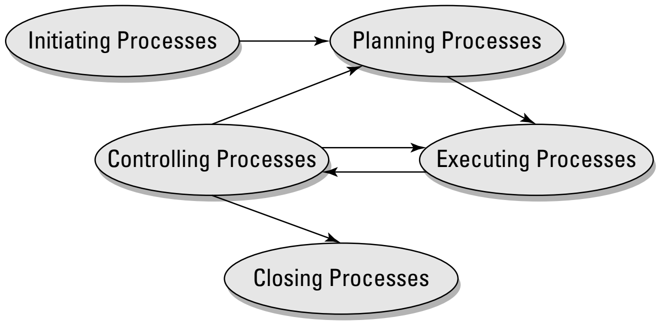

Figure 1-4: All projects follow repeating sequences called process groups.

- **Initiating:** That's really where you are now. The project is in the process of getting selected, sponsored, funded, and launched.
- **Planning:** As you can see in Figure 1-4, planning is an *iterative* process. Planning basically determines how the project work will get accomplished.
- **Executing:** After you get a plan, your project team does the work.
- **Controlling:** Your project team does the work, but you control them.
- **Closing:** Ah, paradise. After the project work has been completed, you tie up loose ends and close out the software project.

Understanding What Makes Software Project Management So Special

There's nothing special about software project management that changes the Iron Triangle or the five process groups. What is special about project management, however, is the nature of the work.

Just as the particulars of designing a new warehouse, building a house, or creating a prototype for a remote controlled airplane are unique, so is the creation of software:

- Software development is weird and requires a specialized skill set to do it well.
- Software creation is tough.
- Software development can be boring, routine, and mind numbing.
- Software creation can create challenges within the development of the code.

Breaking Moore's Law

A long time ago, in 1965, Gordon Moore wrote a scientific paper called "Cramming More Components onto Integrated Circuits." The synopsis of his paper is that the number of transistors per integrated circuit could double every two years. The press loved it. The theory became known as Moore's Law. And he's been pretty accurate on his prediction.

The importance of Moore's Law in software project management is that more transistors per circuit mean faster processors. Faster processors mean more elaborate software. More elaborate software means we need faster processors. And on and on the cycle goes.

Because information technology (IT) drives many businesses today, there is a direct connection between the speed of technology and an organization's bottom line. Between the two is the software the organization relies on. Consequently, businesses demand software that's reliable, secure, and scalable. Your organization's profitability, stability, and ability to attract new customers rely on you and your project team.

Although businesses rely on technology to remain competitive, many software project managers miss this key point: It's not about the technology. Software project management is about the business. It's about helping your company, your colleagues, and even the stockholders of your organization to be successful.

REMEMBER

If you're an IT guru you may easily fall in love with the bits and bytes of day-to-day software development. However, if you're a project manager, you *cannot.* Your focus should be on one thing: getting the project done — on time and on budget.

Dealing with Moore

As your project moves towards completion, chances are there will be leapfrogs in the technology you're dealing with. There'll be new versions of operating systems, service packs to address problems in versions of software your software relies on, and more. Part of software project management is to have a plan to address these potential changes. Every (yes, every) software project manager should have an allotment of time added to the project schedule specifically for planning and responding to Moore's Law. You're saying, "My customers and management won't give me more time just for planning and responding. My customers and management barely give me enough time to complete my project if everything goes perfectly." Notice that we said you "should" have more time. That doesn't mean you will. After all, time is money.

So what do you do? By relying on historical information, you can help your project adjust to Moore's Law. If you have documented instances of past projects that failed because of a lack of time to respond to changing technology, use it. If you have records of your past projects, you can show how the project would have, or at least could have, been more successful with this allotment of time.

This is a good time to remind you to save your project documentation so that you and other software project managers can use it for the same purpose in the future. Check out Part V of this book for more on documentation.

Documented instances are your best argument. We're not saying it's a slam-dunk, but we'd wager dollars to donuts you'll at least have a meaningful conversation about the extra time allotment for planning. Ask your customer or management to try it one time and see what happens. And then document, document, document to prove your point.

If you don't have these project records, well, there's good news and bad news. The bad news is that it's hard to argue for additional time for planning without proof of why the time will be needed. The good news is that you can start now. Without the additional time allowed for your project, here's what we recommend:

- **Do a thorough risk assessment.** Document how the risks due to changes in technology could contribute to failure.
- **Document lost time.** Document any lost time tied to technical changes (research, team training, subject matter experts, and so on).
- **Document lessons learned.** Begin your lessons learned documentation, a document that highlights all the lessons learned, with attention to technology changes, at the start of the project, and as your software project progresses, complete your lessons learned documentation.
- **Communicate proactively.** Communicate to your stakeholders when changes to technology enter and influence your project.

  As a technology professional, it's your job to have your finger on the pulse of change in your industry. You don't want to be blindsided by some major technological event, and you never want to withhold information to your stakeholders that could affect the longevity of a software product. The most important thing you can do is balance cost effectiveness and profitability.

Dealing with the first-time, first-use penalty

One of the most common questions when it comes to software project management is, "How can we tell how much this'll cost or how long this'll take if it's never been done before?"

This is the *first-time, first-use penalty.* The penalty is that you just don't know. It may cost thousands, even millions, if the technology has never, ever been done before. And time? Well, that's even more difficult to grasp.

TIP

You've experienced this. The first time your project team develops code in a new language, productivity slides. The first time an end user loads and uses your application, there's a learning curve.

Leading versus managing

When you think of *leadership* you probably think of positive attributes; a leader is honest, inspiring, and motivating. All true. And when you think of a manager you probably have thoughts like work-centric, accountability, and results-oriented. Also true.

A project manager has to be both a leader and a manager. A leader aligns, motivates, and directs people towards accomplishments. A leader is interested in what others want and what others need. A leader can empathize, inspire, and help others reach their goals. A leader cares.

A manager is concerned about one thing: results. A manager needs the team members to deliver on their promises. A manager needs to see progress and accomplishment. A manager may care about the project team, but not as much as he cares about the project team's ability to get the work done.

A truly effective project manager, regardless of the situation, organizational structure, or technology the project focuses on, must be a leader and a manager. You have to be both.

Take note. If you've got any skills at all and you're just starting out in this business, you are probably either a good manager or a good leader, but not yet both. You'll soon find out which category you fall into. Remember, not everyone has to like you, but everyone has to respect you. If team members refer to you as Mussolini when they're standing around the water cooler, you're probably overdoing the management part of your job. If they call you "all talk and no action," you may need to beef up those management skills and lay off the motivational seminars. As you evolve as a PM, you'll find the right balance between these two extremes.

The first time you stretch your teams, you face challenges with deadlines, cost, and even attitude. Productivity slides, but eventually productivity should curve beyond current levels to a new plateau. At least that's the theory. Actual mileage may vary.

TIP

Chances are your team has worked with the programming language before. Chances are you've done a similar project before. Chances are you have a gut feeling for the time, cost, and feasibility of the project. Chances are, based on your experience, you have some idea of how the project is going to go.

Of course, out here in the real world, you can't go on hunches. Even though it's not feasible to expect these things, your customers, your boss, and your stakeholders want just the facts, only the most definite answers, and the most exact time and cost figures possible.

This is where an *acceptable range of variances* must be introduced. A range of variance is a cushion based on your estimates. No, we're not talking about bloating your estimates, but establishing a level of confidence in the estimates you give. A range of variance is the +/– percentage, time, or cost you append to your estimates. See Figure 1-5.

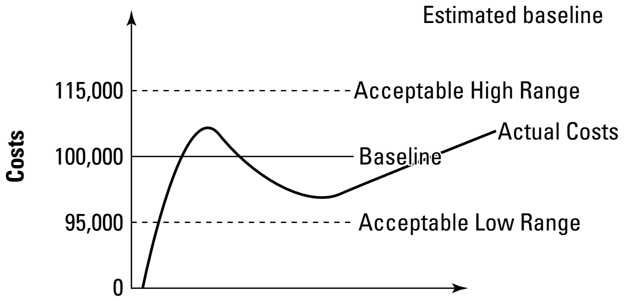

Figure 1-5: The ideal baseline, the accepted range of variance, and the actual results for a typical software project can vary wildly.