**Chapter 9. Understanding Change**

**It may seem like the tools and techniques in the first part of this book** are enough to help you build better software. You've diagnosed problems that affect your organization, you have tools that will help you fix those problems, and you have plenty of tips and examples to help you put them in place. Isn't that enough?

Unfortunately, it's not. Building better software is also about changing the way things are done in your organization, and change makes many people very uncomfortable. Project managers around the world have tried to implement straightforward improvements to the way they build software, only to find that they can't convince the other people in their organizations to agree to discuss those changes, much less to actually alter the way their projects are carried out.

It's very frustrating to see a problem, feel like you have the solution, and not be able to do anything about it. Luckily, there are some straightforward techniques that make it easier to get an organization to change how the software is built. By understanding the most common ways that people respond to change and learning how to convince or reassure the ones who are resistant to change, it is possible to overcome these obstacles and successfully make the changes that your organization needs.

**Why Change Fails**

A project manager who is trying to get an organization to build better software must make changes to the way the work is performed. This is easier said than done. Most project managers are not in a position in which they can simply demand that everyone in the organization abandon their old ways and adopt new tools and techniques. But even in an organization where it's possible to simply issue a memo requiring everyone to change the way they do their jobs, people will usually be uncomfortable changing so quickly.

The planning and the execution of the change are very important, and there are aspects of any change under the project manager's control that can cause it to fail. Not all obstacles are political: if you introduce a poor practice or tool, or you implement a tool poorly, it will not go well. When your projects suffer from poor planning, insufficient training, or simple bad timing, your attempt to change your own project may fail.

But there are problems that are bigger than just your project. Sometimes the biggest obstacles to your projects come from outside of what a project manager typically controls. Sometimes there are people impacted by your change who will resist it. If you feel that the change is necessary despite the resistance, you will need to handle it politically.

Most technical people have an aversion to politics. They feel that the only way for one person to influence another is to first lay out all of the facts; the only logical conclusion should be obvious. Unfortunately, when you are working with other people, it may take more than just facts to win them over. Most people who will be persuaded solely by facts will already agree with you. When you encounter resistance to your ideas, it may be from someone whose objection may seem irrational to you. This is especially true when you are trying to make a change that influences someone outside of your project in your organization (like a stakeholder or senior manager) who can prevent your change and who does not necessarily need your project to be successful. To be successful, you will need to understand this person's motivation and persuade him to come around to your point of view. That's politics, and it may be necessary in order for your project to be successful.

Many project managers—especially ones who have a technical background—tend to ignore the fact that their organizations are made up of people who need to be convinced of the importance of a change before they will adopt it. Some of these people will have an emotional or even irrational response to any attempt at change; it could take a sea change in the organization before they agree to it.

Irrational attitudes about software development usually boil down to two basic beliefs. First, people believe that most or all software projects are delivered late and delivered with many bugs, and that this is just a fact of life. Second, they believe that their organization is unique, and that the problems they are experiencing are particular to their organization and have never been seen before in any other organization.

(This second belief may seem odd, considering the many thousands of software organizations around the world that have all used similar tools and techniques to fix very similar problems and make real, lasting improvements. It's possible that the belief in uniqueness comes from the fact that the software being built truly is unique, in that it has never been built before; it's not a leap to assume—incorrectly—that the software project and all of its problems are therefore also unique to that particular organization.)

Many times, resistance is not irrational at all. Anyone who has been through a change previously—possibly a passing management fad—that didn't fix the problem (or failed outright) may be resistant to another change. It may seem unfair, but if people in your organization have previously gone through poorly planned changes, it will be harder for you to make changes of your own.

When you are introducing new tools, techniques, or practices in your organization, you may encounter resistance for a number of reasons. By exploring the feelings, fears, and justifications for resisting change that project managers commonly encounter, these reactions can be unraveled and understood.

**Change Is Uncomfortable**

Most people in your organization are trying to do a good job. They want their peers and supervisors to see that they are good at performing the tasks assigned to them. When someone has developed a level of comfort and familiarity with his job, the last thing he wants is to have someone come along and make him adopt an entirely new way of doing things.

People can also be afraid of change. Most people prefer to feel like they know what they are doing. They want their jobs to be stable and the tasks assigned to them to remain within their capabilities, and, most of all, they want everyone around them to recognize that they are competent and doing a good job. A new tool or technique may seem like a good idea to you. But if someone in your organization who is affected by your change doesn't understand how it would benefit her, then she will only see and respond to the potential problems that it might create. In other words, it's not enough to show that the benefits of a change outweigh its costs on an organizational level; if people do not see how a new tool or technique personally benefits them, they may see it as a burden with no benefits and will be resistant to adopting it.

Resistance to change is not necessarily a bad thing. For example, when a project's scope or requirements are repeatedly changed, delays and defects usually result. In [Chapter 6](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch06.html), a change control process was introduced to make sure that changes are only made after their costs and benefits have been fully explored. If it is important for projects' changes to be controlled, it is even more important that an organizational change is made only after its costs and benefits are well understood. It would be difficult to work in an organization that welcomes any change from anyone, without considering the impact of that change.

It makes sense to be prudent in evaluating the nature of changes made to the way you build software. Too drastic a change can cost you your project. Everybody on the team needs to understand why new practices are put in place, and each person should feel like he understands how his role fits into the big picture. If the way the team builds software is constantly being "improved," it can become too difficult for each person to keep track of his day-to-day tasks, and people can lose their sense of purpose. This can lead to prioritization problems, quality problems, and almost certain delays.

Yet despite the fact that changes can be hard to introduce, you will want to introduce new tools, techniques, and practices. But even when you are fully prepared to show that the benefits outweigh the costs, people in your organization may still resist those changes.

This can lead to a very frustrating situation. You may make a great case for a change: there is a real problem and you have the evidence to support it; many people, both inside and outside your organization, see the problem. The senior managers and the project team all agree that there is a problem. But when you start suggesting changes that address this problem, everyone gets uncomfortable. As each person starts to think about how he would have to change how he does his own job, some will simply start to resist any change. They may resist it because it would require learning a new way of doing things, because it would require them to do more work, or just because it is different.

And even when you convince others of the need for a change, it can sometimes be difficult to prove to the people in your organization that adopting a particular tool or technique will really solve the problem. Often, people will question whether they should go through all of the pain of changing, without knowing for certain that the change will result in a real solution. It can be very difficult in this situation to convince them otherwise.

**We Already Build Software Well**

Denial is a common response to change. You may have identified a glaring problem, but people around you fail to even recognize it (or simply refuse to acknowledge it). Many professional software engineers and managers have never experienced a project that did not have enormous delays and serious problems; it's often assumed that this is just part of how software is built. After all, they usually delivered something—most projects were eventually completed, and the software they built is now being used. Sure, some projects seem to always be eternally 90% done (with 90% left to go), but most of them seem to get *something* onto the users' desktops (although many patches and bug fixes needed to be rolled out afterward). Isn't that good enough?

The truth is that software development does not have to be painful. It is possible for schedules to come in on time, for teams to work normal hours, and for project managers to feel like they are actually in control of their projects.

The denial mindset should seem familiar to anyone who has tried to implement inspections, only to meet with enormous resistance from the project team. Just as people are not used to seeing their documents criticized and corrected, they are not used to, or comfortable with, their projects being branded "failures." Part of your job as a project manager is to help people in your organization learn from their mistakes. This means that they first need to learn to acknowledge those mistakes.

Sometimes project managers find that people want to know how projects in other organizations have turned out. When people in your organization ask for this, what they are really looking for is the bar that they are being measured against. They don't necessarily want to know what techniques are out there in order to help them build better software; they are just looking for corroboration that they are doing enough already, and don't need to make any changes. But the truth is, projects usually go wrong because of specific problems. Many research groups (including the Software Engineering Institute and the Standish Group) have found that when organizations implement process improvements (like the tools and techniques in this book), those problems often get fixed, leading to software development that costs less and results in better software. Many organizations around the world have verified this in practice.

Also, people in your organization may be willing to acknowledge that there is a problem, only to go back into denial when past performance is discussed. Getting people to talk about their problems is a start, but it's not enough. The same people who complain about project delays and scope creep will often suddenly reverse their opinion when talking about past projects. Most professional people need to see themselves as successful—it's part of each person's identity. For many people, talking about the need for improvement means digging up old, painful projects and examining problems that they would just as soon leave buried. It's easier to look back at a troubled project and see only the successes, forgetting the delays, arguments, difficulties, dead ends, and failures along the way.

In an organization where people are in denial about their problems, it is common for people to shoot down changes by saying that "we've never done it that way before." That's obviously true—it's different, so clearly it hasn't been done that way before; the objection sounds like nonsense. The reason this excuse makes sense to the people who say it is because any change amounts to an implicit criticism of the status quo. If a change is needed, then it means we've been doing things wrong; therefore, any change is bad. Making changes to the status quo seems too risky because the organization has always developed software this way.

Another important reason that changes may seem uncomfortable is that the people in the organization may not fully understand how the software is built. Many senior managers have a great deal of experience with the problems that the software solves: for example, the CEO of a company that builds financial or accounting software knows a lot about finance or accounting. But, frequently, the details of what is involved in building the software are a mystery to them. When senior managers are disconnected from the design and development of the software, they begin to see it almost as a "magic formula." They are clearly making money selling software, so they must have made the right decisions. Now some project manager is coming along and telling them to make changes to this formula, which they don't understand. A senior manager does not like feeling ignorant of his own organization, and he does not like being told that the organization that he built needs to be fixed.

Ironically, in many organizations where people claim that they already build software well, there is no standardized way of building software. Sometimes there are requirements written before the programming begins; other times, there aren't. Sometimes there is a project schedule, but often, there's just a single deadline. In one project, the software may be tested; in the next, it's dumped on the users. Since no two projects are ever done the same way, it's always true that the team has "never done it that way before"—and this excuse is still used to shoot down anyone trying to come up with a way to build better software.

**"Not Invented Here" Syndrome**

"Not Invented Here" syndrome (NIH syndrome) is a name given to a common type of organizational culture where people intentionally avoid research or innovations that were not developed within the organization. When faced with a problem, the people in the organization will typically reject a solution that is known to have worked elsewhere in the industry, solely on the grounds that it did not originate from inside the organization. They opt instead to build their own solution, often at far greater cost.

This may seem ridiculous or silly to people who have not directly experienced it, but NIH syndrome is a serious problem. Some teams will waste many hours defining procedures, creating tools, and building their own solutions to problems that have already been solved elsewhere, rather than adopting or adapting an existing solution that can be purchased off the shelf or learned from a book, training course, or outside expert. One motivator behind NIH syndrome is that people are often rewarded for building new software when they would not be rewarded for buying something that does the same work. For example, a programmer who would get a lot of recognition for spending months building a component might not get any recognition for buying an equivalent one, even though it would cost a tiny fraction to buy rather than build.

If you think about it, you may recognize at least a small example of this behavior in your own organization. For example, many programmers will "reinvent the wheel," building functions or components that could be purchased or downloaded. If your organization commonly develops proprietary technology instead of using an alternative that's available from a third party, it may suffer from at least a mild case of NIH syndrome.

A project manager attempting to change an organization with a bad case of NIH syndrome faces unique challenges. In most organizations, once people recognize that there is a problem, it's often sufficiently convincing to show them that people in other organizations solved the same problem with a solution that already exists. But people in an organization that has NIH syndrome will reject the idea that anyone else in any other organization could possibly have had the same problems. There is simply a pervasive idea that "we're different," which leads to immediate resistance to any ideas that come from outside.

There is an especially virulent strain of NIH syndrome that is commonly found in small entrepreneurial companies. It goes beyond simply believing "we're different"; instead, the people at afflicted organizations believe "we're better." They consider larger companies to be inflexible and laden with bureaucracy. They feel that their small size is an advantage. People in companies like this will often refer to things like their "flat organizational structure" (which, oddly, still features four or five layers of hierarchy, similar to most medium-sized companies), and will talk about how they can respond much more quickly to clients' needs than their larger competitors. Such a mindset often leads to outright rejection of any change based on a tool or technique that works for large companies. Anything that a large company does is dismissed as "too bureaucratic" (see below), and would "clearly" slow down a small, nimble company. In truth, however, every single tool and technique in this book has successfully been used on small projects employing as few as two people. When it comes to NIH syndrome, size really doesn't matter.

**It's "Too Theoretical"**

When an idea does not make intuitive sense, many people will dismiss it as a result of "academic research," which could not possibly apply to the world they live in. For example, to someone without a project management or software engineering background, it may not be immediately obvious that reviews reduce defects, or that it's important to write a specification before building software. To him, these procedures are time-consuming, with no obvious benefit. They sound good in a book, but would never work in the organization. In other words, they are "too theoretical ."

Some managers—especially ones who consider themselves "hands-on" and who value technical knowledge above management skill—will say that many of the tools and techniques in this book are "too theoretical," and therefore somehow do not apply to their particular organizations. This attitude is common among managers who worked their way up from programming or other IT positions and who have only served as a manager in one organization. It is also common among people who have never experienced a successful project that was on time, on budget, and with few delivered defects.

Declaring a particular tool or technique "too theoretical" may seem like an odd response. These practices arose from software engineering practices that were developed, implemented, and refined over the course of countless projects in thousands of organizations. Most people would consider any solution that is well accepted and has been implemented across the industry to be anything but theoretical. But for many managers, a technique described in a book is of less value than one that was learned or discovered in the field. If someone has never seen a project that went smoothly, it seems natural to assume that all projects suffer delays, scope creep, and other problems that stem from poor planning and engineering; they consider any proposed "cure" for these problems to most likely be nothing but an academic theory that would never work in the real world.

It is especially common to hear the "too theoretical" excuse from the manager or leader of a small team facing growing pains. This typically occurs when programmers are added to a very small team. The team slowly gets less and less productive, and adding more programmers does not seem to help. Many of the problems described in [Part I](https://learning.oreilly.com/library/view/applied-software-project/0596009488/pt01.html) start to occur, and the manager gets frustrated. Unfortunately, this is a very difficult (yet common) position for a new manager to find himself in. Typically, he finds himself trying to doing everything the same way it's always been done, but his projects continue to break down. It seems like every trick he's tried has backfired. He's called big meetings that turned out to be worse than useless. He's yelled at his team, other managers, even the CEO. It seems that everyone knows that something is wrong, yet nothing will fix the problem.

Still, this manager will often reject tools and techniques like the ones in [Part I](https://learning.oreilly.com/library/view/applied-software-project/0596009488/pt01.html) as "too theoretical" because he has not personally experienced them. When there were only two or three people on the team, the projects apparently went just fine. It seems intuitive to him that adding a few more people should not make much of a difference in managing the project. To this person it's not intuitively obvious, for example, that writing a vision and scope document or creating a project schedule will help his projects run more smoothly. Anything that he has not tried, and especially anything that does not directly affect the code, must be "too theoretical."

The problems that are diagnosed in the first part of this book really do affect very small teams; they are just not nearly as difficult to overcome when only two or three people are affected. It is not hard for one person to keep track of changes, communications, and project status for a very small team. However, as the team grows, these problems compound. The team starts to lose track of changes to the scope, requirements, and code. The project schedules get more complex and less accurate. Stakeholders and users start to feel that the project is getting out of control.

A manager in this situation will be very cautious about adopting any change. He knows that just making changes at random makes the problem worse. He's probably read management books and tried applying the information in them, only to find that didn't work; he's come to distrust this sort of advice. By demanding changes that are not "theoretical," he's really saying that the team must be careful not to implement any more changes unless they are already proven—preferably on one of his own projects. Unfortunately, that creates a chicken-and-egg situation: how can the team take a particular technique from theory to practice, when every technique is too theoretical to be implemented?

**It Just Adds More Bureaucracy**

An especially damaging attitude in some organizations holds that programming is the only important activity in a software project. Project management tools and techniques are seen as distractions that drain energy and effort away from the programmers' "real job" of writing code. Any project activity that takes place before the programmers begin writing code simply amounts to "spinning our wheels," and the goal of all early project activities should be to get the programmers doing the "real work" as quickly as possible.

A manager in this sort of organization will typically think that if she just adds more programming hours, a project that seems to be failing will get back on track. Since programming is the main activity in the project, all project problems amount to programming problems. In fact, there are project managers who believe that all they need for a successful project is a team of top-notch programmers. They wonder, "Why do we need to quadruple our documentation by creating schedules and project plans and change control procedures?" After all, it's been shown time and time again that a great programmer is 10 times as productive as a mediocre one. Isn't it enough to pay top dollar getting the very best programmers and setting them loose on the problem? Why all the extra "bureaucracy"?

In reality, programming is usually less than 40% of the effort on a successful project. Most project problems are caused by the team not understanding what it is that the software should do. Estimation problems happen when the team members don't explore all of the assumptions that they are making and, as a result, don't have a handle on what information is known and what is unknown. Project planning goes wrong when the scope creeps, or when there are problems that could have been foreseen. When bugs are found in the software, often it's not because the software is broken; the software is usually working exactly as the programmer intended it to work, it's just not doing what the users need it to do.

The solutions to these problems do not involve programming. But, to some people, any action that the team takes that does not directly relate to programming is "bureaucratic." Planning the project, writing down requirements, and holding inspection meetings is seen as just pushing paper around. There may be a project schedule, but this is just used as a tool to help the rest of the organization understand what it is the programmers are doing—in other words, the schedule is created by asking the programmers their opinion, and the project manager's job mostly boils down to simply reporting that opinion to the rest of the organization. (Even worse, of course, is the project manager who simply makes up the schedule out of thin air, or bases it solely on the boss's expectations.)

When the project manager's role is reduced like this, there is usually an implicit assumption that a schedule is just a guess, and an expectation that it will almost certainly slip. At least the project managers and other software engineers believe that it's implicitly assumed. Unfortunately, this is a dirty little secret among the engineering team. The rest of the organization takes these estimates and schedules at face value. They believe that the team made a real commitment, and they make important decisions based on what they believe is a real date. This leads to a general feeling of distrust between the software engineering group and the rest of the organization. It all starts with the project manager's mistaken attitude that it's just not possible to estimate a project accurately, and that you just can't predict what's going to happen when the team starts building the software. (See [Chapter 3](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch03.html) for more information on how to estimate a project accurately.)

In an organization where this attitude is prevalent, it is common to see a much higher value placed on anything that has to do with programming, with a lower value placed on all other areas of software engineering. If the programming team requests new computers, often senior managers will not hesitate to spend thousands of dollars on them. Yet the project manager may be refused permission to spend a few extra hours talking to stakeholders and users in order to write a vision and scope document, even though it would save an enormous amount of project time (and only cost the equivalent of a few hundred dollars of people's time). When this happens, it's usually because it is easy to justify any activity or expense that is done to benefit programming, while all other expenses are regarded suspiciously.

**You Can't Give Me More Work!**

Most of the changes that a project manager makes will increase the workload of other people. Software engineers, managers, and stakeholders who were not directly involved in building software will suddenly find themselves expected to attend status and review meetings, participate in planning and estimation activities, work with someone creating a vision and scope document or eliciting requirements, or perform other tasks that were never expected of them before. If you are making these changes, then you are the person piling additional work onto someone's already overflowing plate. Not surprisingly, there are people who will not be happy with this arrangement.

People are often unhappy to be asked to attend meetings, especially in organizations where most meetings lack direction, focus, or even an agenda. In organizations like this, the meetings tend to meander. Often they boil down to a discussion between two people about a topic that has nothing to do with anyone else at the meeting. Sometimes, meetings are called simply to give a captive audience to a senior manager. If you are in an organization where everyone hates going to meetings, when someone suddenly finds out that she has to attend your weekly status meeting, she might be unhappy—and provoked into doing something about it.

A common response from a recipient of this new workload is surprise and shock. It's very uncommon for people to have tasks assigned to them by people to whom they do not report, yet here's some project manager doing exactly that. Often, when someone is surprised by extra work, he feels especially motivated to take action to remove it from his plate. He will complain to his boss and to his boss's boss, and he may even try to go over your head to get your entire project shut down. When a project manager tries to put new tools or techniques in place, it is frustrating to encounter resistance from the very people who will benefit from the practices simply because they seem like extra work. (It is especially frustrating when team members agree in principle that those practices will reduce the total effort required to build the project, yet do not agree to adopt them!) Even if you win the battle and get your senior managers to agree to force him to do the extra work, you could still lose the war: now a participant in your project cannot be relied upon, and could cause damage to your project by stonewalling and causing delays. He could even sour your organization to any future improvements if the senior managers hear only his complaints and do not see immediate results.

**It's Too Risky**

The celebrated economist John Maynard Keynes once wrote, "Worldly wisdom teaches that it is better for the reputation to fail conventionally than to succeed unconventionally." Legendary investor Warren Buffett put it this way: "Lemmings as a class may be derided but never does an individual lemming get criticized." In other words, any manager who backs a change puts his reputation on the line; if that manager does nothing, he will not be criticized for maintaining the status quo.

When you make a change in an organization, you aren't just altering the activities that the team performs. You're also affecting how people relate to each other in the organization. A change that goes wrong can ruin someone's reputation (and, if it's serious and public enough, their career).

Nobody gets blamed if things stay the same. If a project fails, but many other projects failed in the same way, the failure is usually seen as inevitable and the people in charge of the project are not held accountable for the failure. And even the most flawed deliverable can be painted as a success, even if the software that is finally produced barely does what it's supposed to and requires many patches and bug fixes. (In fact, for many software professionals, this is a fact of life—every project they have ever worked on turned out that way!)

Many people equate questioning what has always been done with insulting the organization as a whole. They consider it to be something a team player would never do. To someone with this attitude, the fact that you are pointing out problems and suggesting changes sets you apart from the accepted culture. They feel that since you are making wide-ranging criticisms of the organization, you do not buy into its culture and are simply being counterproductive. Paradoxically, to these people, it is considered more productive to make bad decisions and let projects fail than to question the way things are done.

This means that when people complain that making changes is risky, they are not incorrect. However, a manager shooting down a proposed change as "too risky" may be sensing risk to his own reputation, rather than to the project.

When looked at from a pure cost-benefit perspective, most of the tools and techniques in this book have a very limited risk to the project. Usually, it only takes a few hours to run a Wideband Delphi session, set up a version control system, write a vision and scope document or hold an inspection meeting. Even if these tools and techniques fail to produce any results at all, the total cost to the project is minimal. Yet these same tools are routinely shot down as "too risky," and in a sense they are—but not with respect to the project.

When people talk about these tools and techniques being risky, often they are not trying to say that the tools will somehow damage the project; rather, the risk is that if the project still runs into problems, the manager who implemented the new tools can be held responsible for those problems. In other words, there's an implied rule in most organizations: "you break it, you bought it." If you make a change to an existing process, you're now responsible for any failures that result, even if those failures are only tangentially related to the changes you made.

People are justifiably reluctant to go against the conventional wisdom in their organization. No matter how misguided, the popular opinion is very powerful and difficult to change. People who are willing to point out a series of singular, unrelated mistakes may be unwilling to admit that there is a more general problem with their organization—especially if admitting the problem calls into question their knowledge of software development. People do not like to feel like they don't have all the answers. And if you point this out in your effort to change your organization for the better, you might inadvertently cause people to feel that you are publicizing their mistakes and calling them out as incompetent.

Incidentally, this is one reason many people are turned off by the idea of process improvement. They have only seen attempts at change partway through a project that has gone awry. Managers start making out-of-context changes in desperation, usually based on something they have read but don't fully understand. Even worse, the changes are frequently used as a means of forcing a stakeholder or client into compliance with an already delayed schedule. This is not so much a way to save the project as it is a way for some manager to protect his reputation. Not all improvements will help in every situation.

For example, one of the most common places where changes should not be introduced—but often are—is when a project is starting to fail and the stakeholder is starting to get angry. This mistake is especially common in consulting companies that gave unrealistically low estimates for their projects and now find themselves in trouble. They see a client who rightly points out that the software has many defects, and that it does not fulfill his needs. A common (and unfortunate) response from the project manager is to try to change the organization, instead of admitting that the client was given an unrealistic plan and working to better fill the client's needs. A change control process is implemented to clamp down on any new feature requests, and a hard-and-fast schedule is put in place that forces the client to accept the work that the team has already done, whether or not it fulfills the client's needs. This results in a frustrated client and increased friction between the client and the project team.

Forcing a stakeholder to accept poor software is a terrible reason to make changes to the software process. Organizations are right to resist changes that are made for any reason other than building better software more quickly. If there are client or stakeholder relations problems, they need to be dealt with directly, instead of trying to solve those problems by changing the way the group builds software.

# How to Make Change Succeed

Progress comes not just from making changes, but from making smart changes. However, it is often difficult to tell the difference—and no organization should be entirely comfortable with change. It's often difficult to tell the difference between a real, substantive change for the better and a management fad that will cost time and effort, but yield little or no reward. Using the techniques in this section can help you demonstrate to the people in your organization that the practices you want to implement are appropriate, and help you to get your proposed changes accepted by the project team and the organization's managers.

However, while these techniques are useful and proven, they are not cure-alls. It's possible to run across organizational problems or roadblocks that are beyond your ability to fix (for example, there may not be enough money to hire a test team). The Achilles' heel of the approach to improving organizations that's described in the first part of this book is that there are people who will resist change for irrational and emotional reasons, and, if they have more power than you in your organization, you simply may not be able to make the changes that your projects need.

You can't stop people from being averse to change, and you can't always stop them from shooting down your ideas. Even if your ideas make perfect sense, you may still be unable to implement them simply because there is someone above you in your organization who feels uncomfortable with your proposed changes. However, if you are creative, forgiving, and flexible, the most daunting organizational problems can be addressed and sometimes even overcome.

One important element for successful change is understanding how the people in your organization think about and react to changes. Good planning, along with some understanding of the psychology of the people affected, will help convince people to accept them. By understanding the ways people think about and react to change, and by applying certain techniques that may make them more comfortable with the changes, you can improve your chances at successfully changing your organization.

## Prepare Your Organization

Before you can begin to implement changes, you need to deal with the attitudes that will cause people to reject those changes. It's up to you to figure out your organization's culture. Try to feel out the sorts of arguments that you will run into, in order to get a feel for your audience. Then you can tailor your pitch for change—and it really is a sales pitch in many ways—to your organization.

There are several possible strategies that you might use to "sell" your change to the people in your organization, depending on the environment in which you are working. Here are a few that have been effective in the past. There's no single solution to any of the problems in the previous section, though—you will have to use your own judgement to figure out an approach that will work with the people in your organization. With luck, you can start to combat the poisonous attitudes in your organization that would stand in the way. Of course, even after all of this, there's no absolute guarantee that you will actually be able to change your organization—but at least now you have a better shot.

### "WE'VE ALWAYS DONE IT LIKE THIS"

Organizations uncomfortable with any kind of change are the hardest to deal with. In this situation, it often makes sense to make it appear as though the change you are making isn't a change at all. Pitch the effort as preserving the status quo: "We've always built software like this; now we're just writing down the best practice so we apply it consistently."

This is especially helpful in growing organizations, where you can talk about a need to write down how things are done already, in order to help new people adjust to the environment. It's important to remember that people have a need for consistency—when your actions are aligned with past projects, it's much easier to persuade others in your organization.

This strategy can also be pitched as a training program. All organizations must train people, so it's possible to pitch your improvement effort as a way to bring them up to speed by writing down how things are done.

Keep in mind that this approach is not completely honest—you're trying to pretend that a change is the same as the status quo, and may be taking advantage of the fact that few people in your organization have sufficient understanding of how software is developed in their own organization to see the difference. But this approach can often buy you enough time to get people used to some of the changes that you want to make.

### BE POSITIVE ABOUT THE WORK THAT'S ALREADY BEING DONE

When people are in denial about the need for change, it is often because they need to see themselves as successful and do not want to admit that their past projects have been less-than-perfect. Use this feeling to help you implement your change by praising the work that's been done in the past, and positioning the change as a way to build on those past successes. Keep your tone positive when talking to people about the need for change.

Find examples in past projects of things that went well that support your changes. For example, you may have had a positive meeting at the outset of a project generally viewed as a success in which the team dealt with scope issues. This can be the kernel of justification for an effort to build a vision and scope document for a future project.

If anyone in your organization has ever discussed the sorts of things that you are proposing, you'll be in better shape. You can use these people to help you make your case by showing them that you are building on their ideas. This can help you build consensus among the project team and in the organization.

### TAKE CREDIT FOR THE CHANGES

It usually helps to show that the changes you are proposing have worked in other organizations. But if your organization suffers from NIH syndrome, this approach could be damaging. It's counterintuitive, but it may make sense for you to act like you thought of the changes yourself.

Don't talk about the changes as a process improvement effort that standardizes the way you build software. Instead, step away from the big picture and concentrate on solving individual problems. Justify the changes as if they are solely in response to specific problems in your organization, and not as tools that are standard across the industry.

Some people are understandably uncomfortable taking credit for the ideas of others. But while you may not have come up with the idea of a vision and scope document or a project plan yourself, you did have the idea of applying it to a specific project in your organization.

This may seem manipulative (although that's not necessarily a bad thing). But it doesn't have to be. A project manager can help the whole team take credit for changes as a way to motivate them and help them move forward. This is especially useful when combined with being positive about the work that's already been done. A project manager might say something like, "Folks, we've done a great job with these first five work items. I'm very proud of what you've done. But I'm convinced these next three are different in nature from those five and we need to approach them differently than we have in the past. Here's how...." It doesn't matter that the proposed change is a tool or technique from a book; it can still be pitched as a change that builds on the work the team has already done.

### MAKE THE CHANGES SEEM STRAIGHTFORWARD

When someone feels that a tool or technique is "too theoretical," what he is really saying is that he's never heard of it before, doesn't fully understand it, and can't immediately imagine an actual situation in which it would work. He typically talks about the "real world," and he may feel like you don't necessarily live in that world, if you're proposing a change that has not been directly borne out by his experience.

Be careful in this situation. Consider someone who has been in the industry for a long time, but has not come into contact with these ideas before. It's likely that he has seen countless projects fail, and has come to grips with—and possibly built his career around—the idea that a lot of projects simply go wrong. By telling him that there is a straightforward way to fix this problem, you are challenging a fundamental assumption in his career. He will rightfully take that personally, and you must take his feelings into account.

If this person is senior to you in your organization, this also becomes an issue of credibility. If you put yourself in the position where you are questioning the experience of someone senior to you, he will want to know where your credibility comes from. It's a mistake to say that your credibility comes from doing research; a better approach is to look for corroboration from within your organization and your projects to prove that your ideas are valid. The best way to handle this situation is to build consensus among your peers. It's much easier to have this conversation with a senior manager when your credibility is already validated by the people around you, and when it's clear that the organization's culture is ready for the change.

Another way to approach this situation is to pitch the changes you want to implement as technical tools, rather than as core software engineering concepts. Most people who have been in the field for a long time are used to routinely applying new technical tools that have never been tested in the organization. This is also a good way to gain consensus among the programming team for your ideas.

### BUILD SUPPORT FROM THE TEAM

If you can bring the programming team on board, you have a much better chance of convincing the rest of the organization to follow. In most organizations, people who do not have technical skills often defer to the programmers any time there is a disagreement. If the programming manager in your organization is on board with the improvements, he can pitch an improvement as a technical change instead of a more general, far-reaching process change. Once the programmers start demanding changes, they usually get their way.

There are a lot of project problems that the team is very aware of. When the scope creeps, the programmers have to tear down code they previously built and replace it with code that is patched together and not built as well as they would like. The programmers would much rather fix a bug before a client sees it. Poorly defined requirements lead to changes and frustration. All of these problems are exactly the ones that you are trying to fix.

Show the team that you are working to help them. For example, using Wideband Delphi estimation and building a project schedule may seem needlessly "bureaucratic" to them—until you show them how it will help them avoid working overtime later on in the project. By showing that there are clear benefits to what you are doing or suggesting, you can avoid some of the knee-jerk reactions from your team against your changes, and instead get them on board.

### SHOW THAT THE CHANGES WILL SAVE TIME AND EFFORT

When someone talks about a change adding too much bureaucracy, what she usually means is that it takes time and effort that she is not used to spending. This is where an explicit justification is necessary. That justification is not about showing them charts, graphs, or numbers; rather, your goal should be to show how you are working to reduce the overall effort required for the team to build the software.

People do not want to come to meetings unless those meetings are proven to work. You need to convince them that for every hour they spend in a meeting, they are shaving off at least an hour at the end of the project. It's not hard to make an intuitive case for this, as long as you tailor it to each individual person that you are talking to. Explain the impact of the problems on each person's work, and show how the tool will reduce that impact.

For example, if a technical support manager is balking at attending inspection meetings, point out that you are trying to get more defects out of the software so she'll have to deal with fewer support calls about them. Programmers are often concerned with scope creep because it requires tedious and unnecessary rework; show them how a vision and scope document or use cases will help to reduce the time spent on rework.

### WORK AROUND STRAGGLERS

There are some people who simply cannot be convinced that a change is worthwhile. They resent being given any extra work. They may even be against the entire project for entirely selfish reasons. For example, it's not unheard of for a programmer who is highly skilled with a particular platform or technology to sabotage efforts to migrate the software to an entirely different platform, simply to protect his expertise. Some stragglers don't even look like stragglers—they may be "heroes" who are used to waiting until there is an emergency before jumping in and saving the day. But whatever the reason, if someone is firmly against your changes, you are not going to be able to bring them on board—at least not by yourself and not now.

Before working around a straggler, see if you can bring him around. One good way to do this is to have a respected team member talk to a straggler so that, instead of ignoring or working around him, you're putting him in a situation in which he can learn to be more productive. Unfortunately, not all people will be convinced, even by those they respect.

The way to handle stragglers who refuse to adopt your changes is to work around them. Build consensus among everyone else on your level in the organization. Don't worry too much about people who oppose you for reasons that can't be dealt with: if your changes eventually become part of the organizational culture, they will either come on board or leave on their own.

Again, the idea of consistency is very important in this situation. If someone takes a public stand on an issue, she feels an enormous psychological need to remain consistent with that opinion. This means that if you push someone who is initially opposed to your ideas, she will feel pressured to fight harder and harder against you: the more she argues against the change, the less she feels like she is "allowed" to change her mind without losing face. On the other hand, if you leave her alone initially, you give her an environment where it could be much easier for her to come around later.

You can also use consistency in a positive way. It is very important that you have gained a real (and written, if possible!) commitment from the people around you before you try to pitch your change upward. Not only will this help you show senior management that you have real support, but it will also help people to stay committed, because they feel the need to remain consistent with their past decisions.

It is important not to go to senior management too early. If you do, you are essentially going over the heads of everyone around you who has not already committed to your change. This is counterintuitive: you have a solid case for change, and you know that you can convince your boss to make it, so why not just go there first?

The problem is that the ideas you are pitching are very powerful and often very convincing. It is often possible to implement wide-reaching changes that affect many people in your organization without actually involving them at all, simply by going over their heads. The minute that one of those people sees additional work that you have managed to get assigned to him without first asking him, he will turn against the change. Had you given him a chance to come on board first, he probably would have seen the benefits and supported you; now he's working against you, simply because you backed him into a corner.

This isn't about the personal aspects of them supporting or not supporting you. If the changes that you are making are good ones that will increase productivity, then people will probably jump on your bandwagon once they recognize the value of the changes. Everybody wants to be part of a winning team, and the key is to make it easy for them to join your team. If people have an adverse reaction to what you are saying, let it go—they'll come around later.

Once you have a real consensus, you can go to senior management. It will be clear to them that the entire culture is asking for this. One of the most important principles of organizational change is that changes do not stay in place without support from senior management.

### STICK TO THE FACTS

People respond well to someone who speaks directly to them and who does not have any hidden agendas. Be clear about your motives; make sure that you talk about the costs as well as the benefits of any change that you suggest. You must make an effort to understand your audience: being a straight-talker to a sales manager is much different than being one to a programmer, so you need to really think about who you are presenting these ideas to before you do it. Learn their perspectives and frame your arguments in ways that are interesting to them. Another important part of being a straight-talker is having a solid grasp on the ideas behind the changes that you are making. If you really understand them, then you can put the ideas into terms that anyone else can understand.

"It's too risky" can be the best possible objection that you can hear. It means that the people you are talking to are listening, thinking about implementing the change, and coming up with ways that it could go wrong. They are really thinking about what it would take to do what you are suggesting. This means that they can be convinced with facts, not just persuaded with emotions and politics.

In a situation like that, you can get approval to start a pilot project, and show the benefits of changes on that pilot. You can present research that shows that the changes you want to make are accepted industry standards. You could get approval to study the problems that exist in the organization, and plan an improvement project to come up with an answer to those problems. You can figure out how much the projects cost in terms of time and effort, and show how your changes will reduce those numbers. And, most importantly, you can show how it's even riskier to keep things the way they are. That's a productive conversation any way you look at it, because you'll come out of it with a list of problems that you need to address with your improvement effort.

(Unfortunately, "It's too risky" can also be the worst possible objection that you can hear. It could mean the person you are talking to has shut you out, and is no longer listening. When this happens, you can't make any more progress from this angle.)

## Plan for Change

Once you have made some headway in overcoming resistance, the most important way to ensure the success of your change in your organization is to plan for the change. Planning for a change is similar to planning for a software project, in that the scope must be defined and tasks need to be assigned to people who will carry them out. By treating a change to your organization like its own project, you can use the same planning tools as you would for a software project.

There is a fundamental difference between a project to change an organization and a project to build software: the bulk of the effort in a software project is devoted to building work products, while the effort in changing an organization is focused on training (or retraining) the people who will then turn around and build the software differently. But in both cases, the scope should be written down and agreed to, and the resources assigned and trained.

### CREATE A VISION AND SCOPE DOCUMENT

The way to make a successful change is to ensure that the problem you are solving is the most important one that the organization faces. Before people in your organization will accept a change, they must be convinced that the change is necessary. To an eager project manager who wants to implement new tools and techniques, this can be a frustrating situation. But it's actually a good thing—if people are too willing to accept changes to the way they do their work, then any changes you make will quickly be replaced with the next popular idea to come along.

In [Chapter 2](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch02.html), the vision and scope document was introduced as an important planning tool that helps ensure that each of the project's features addresses a specific user or stakeholder need. Each person's needs are written down, and each of the planned features is tied back to one of those needs. The document is then reviewed by everyone who will be affected by the project, to ensure that all of their needs and concerns are met.

A vision and scope document can also be used to plan a change to the organization. [Table 9-1](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch09s02.html#appliedprojectmgmt-CHP-9-TABLE-1) shows a typical outline for a vision and scope document.

*Table 9-1. Vision and scope document outline for a change project*

|  |
| --- |
| 1. Problem Statement    1. Project background    2. Stakeholders    3. People affected    4. Risks    5. Assumptions 2. Vision of the Solution    1. Vision statement    2. List of changes    3. Changes that will not be implemented |

This document is developed in exactly the same way as described in [Chapter 2](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch02.html). The stakeholders are identified, their needs elicited, and the scope of the change is defined.

There are just a few differences between this outline and the one for a software project:

* Instead of a "Users" section, there is a "People affected" section. This section describes the specific people in the organization who will have to change the way they do their jobs. This is the first important reality check for a project manager attempting to make changes. It is very easy to think about changes in abstract terms, forgetting that real people will have to change the way they work. It's even easier to gloss over the fact that people will be affected when you're selling the change. This section helps everyone be clear on who is affected from the very beginning. Listing the affected people in the document also ensures that they are included when it is reviewed, which allows them to have a say in how they do their jobs.
* Instead of listing features that will or won't be implemented, this vision and scope document lists changes that will be implemented. Each of the changes in the "List of changes" section should include a full explanation of exactly what changes will be made. For example, if use cases are to be implemented, include the use case template that will be used and a description of any elicitation tasks and other tasks that will be performed in order to create them, as well as any additional resources that will be needed.
* There is no "Scope of phased release" section. If the changes need to take place in a certain order, that should be incorporated into the description of each individual change.

Writing a vision and scope document before making the actual changes allows the project manager to gather evidence that those changes will address real problems in the organization. It is important to point out specific instances that other people will agree are problems. A successful vision and scope document will show everyone in the organization that there is a real, troubling problem. If you only include real evidence in the document, an objective reader will see that there is a serious problem—and by proposing a change that will fix it, you can throw them a rope by offering a real solution.

### INSPECT THE VISION AND SCOPE DOCUMENT

Once the document is written, it should be reviewed by all stakeholders and everyone in the organization who will be affected (see [Chapter 5](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch05.html)). This will accomplish several things. It will make sure that the changes you are proposing really solve the problems that they are supposed to solve, because they will be documented as needs in the "People affected" section. But more importantly, it will ensure that the change is communicated to everyone who needs to see it—and by incorporating their feedback, you will help gather consensus among them.

This is a check both for you and the other people who this affects. It is important for them to see that your changes will help them. But it is also important for you to choose the changes that will do the most good in the organization. It is much easier to propose an unnecessary change in a meeting than it is to do it in writing. By writing down the rationale for all of the changes, you can make sure that people see that there are real problems to be solved, and that you are proposing solutions to those problems.

The reason it is important to write down the rationale for each change is that it is tempting to try to "fix" a problem that doesn't really exist, simply because the change is easy to implement. Many people decide to make a change or implement a certain practice, and then "find" problems to justify it. (In much the same way, if you lose your keys in the dark, it's easier to search for them under a streetlight because it's easier to see.) By identifying the rationale for each change, you can find the real problems that need to be fixed. This helps you prioritize your changes, in order to address those problems that hurt the most.

The audience for the review should be as large as possible, to ensure that everyone who is affected by the change sees that it is coming. The review will give them ample opportunity to give their input. People are less likely to react irrationally to a change if they are given a chance to give input when it is proposed. Most importantly, since everyone who is in a position to object can be included in the review, you can get those objections out in the open early. That way, you can address their objections early on. Many people object to changes simply because they were blindsided; by asking for their input before the change is implemented, you can avoid that problem. Once the vision and scope document is approved, it is much less likely that there will be surprises later on.

### SCHEDULE THE CHANGES

Once the team has approved the scope of the changes that will be made, it is time to implement those changes. By updating the project schedule to include those changes, you can make it more likely that they will actually be carried out on the project.

Building changes into the project schedule is an effective way to "seal them in" to guarantee that they actually happen. Many changes unravel because while everyone agrees that they are a good idea, they never actually make it into practice. By adding tasks to the project schedule that reflect the change, you ensure that time and resources are dedicated to implementing it. You also help the team members see that the change is coming, and give them time to plan for it.

It's also important to allocate time for training. You may understand the details of the new tools and techniques, but many of the people on the project do not have that advantage. Make sure to include meetings to introduce the team members to the new practices. If additional training sessions are necessary, schedule them as well. Plan to give the other people in your organization the time they need to learn how to use the new tools and techniques. This may mean adding time to tasks that will be done differently than in the past, due to the changes. (Allow time for a learning curve—people will not be as efficient at using the new tools and techniques initially as you hope they will become after some practice.)

For example, if you are implementing code reviews, you should add training and review meetings to the schedule. You should also extend the programming tasks, in order to allow the programmers time to make changes found during the review. When the schedule acknowledges that additional time should be spent on these tasks, the programmers are much more likely to actually perform the code review tasks.

### NOTE

A more detailed process for scoping and planning an improvement project is described in Making Process Improvement Work by Neil Potter and Mary Sakry (Addison Wesley, 2002).

## Push for Consensus

It's difficult to change an organization alone. It is much easier to make a change if you have the support of others in your organization. Identifying potential allies is an important step in changing an organization. The most effective way to change an organization is to build consensus within your project team, among your peers, and up through the management chain.

People will be more positive about your change when they see that other people are already on board with it. If you can recruit early supporters, it will be easier to bring other people around to your way of thought. The best way to convince someone to make that investment is to show that there is already a consensus among the software experts in the organization.

The first step in generating consensus is to find people who also recognize the problem that you are trying to solve. This should not be hard—if your organization's problem is serious enough to warrant a change, there will probably be other people around who have noticed this problem as well. Put aside for now the fact that people may not believe you have a real solution; it's sufficient to start with a basic agreement that there is a problem.

Many people feel that change should either be "top-down" (meaning that the changes originate from management) or "bottom-up" (meaning that they originate from the team). However, while it is absolutely critical that you have the support of your organization's senior management before implementing a change (unless it is for a change that affects only your project), there is no need to make this decision while still generating consensus. If you can convince a senior manager that there is a real problem and that you have a solution, that person will be a very valuable ally. But it's also important to convince people who are on your level in the organization. To many senior managers, the most convincing argument is that several people who report to him agree on something. This is why it is especially useful to get multiple people on different levels of the organization to agree that your changes will improve the way software is built.

Once you have found people who recognize that there is a problem, you can work to show them that you have a solution. An effective way to convince people to join your effort is to show them that you are not just suggesting change because you don't like the way things are done: you are also helping them with problems that they wish would be solved. Most people are never really asked if they are having trouble. Take the time to listen to each person's complaints. If you can show someone that her problems are not her fault but, rather, could be attributed to something external (like a lack of planning or change control), she will be much more open to your solutions.

It's not enough just to find people on the team who are willing to talk about their problems. There are many people who just love to complain about work. It's easy enough to get people like this to talk about what's wrong, and even to acknowledge that there are endemic problems. But once it comes time to make real changes, someone who has not really bought into your solution may disappear from your effort at the first sign of resistance. Also, beware of people who come around too easily—it may be that they are simply easily swayed, and will abandon your effort for the next big idea that comes along.

Gathering allies is also a good reality check for your change effort. If you cannot convince even a small number of people that there is a real problem to be solved, and that your proposed change will solve it, then that is a good indicator that you will run into serious problems when attempting to sell your changes to the rest of the organization.

Give yourself a lot of time to do this. Organizations do not change overnight, and consensus is not generated with a single meeting. It is important not to steamroll anyone. If someone has an objection, make sure that he feels that you are taking that objection seriously. It is easy to get frustrated with disagreement; try to find ways to help the people who disagree with you, rather than simply ignoring or going around them. If someone disagrees with you, it may be that he sees that there is a more important problem that you should be concentrating on instead. Show each person that you are trying to keep his best interests in mind. Try to learn what his biggest problems are and work to solve them.

## Use a Pilot Project to Build a Track Record

The best way to build credibility in your organization is to show a real track record of past success. Running a pilot project is an effective way to build a track record.

A pilot project is simply a project that you have selected on which you will test specific changes before rolling them out to the rest of the organization. Before running a pilot project, make sure that the changes that you want to implement are limited in scope. Making more than a small number of changes at a time is usually difficult to manage.

Choose pilot projects carefully. You will want to select ones that will have a visible effect but carry little risk. Some tools and techniques are easier to implement than others—choose the "low-hanging fruit" by selecting the changes that you feel are most likely to succeed.

The best pilot projects are ones that are likely to succeed. A good candidate project might involve problems similar to ones that the team has solved in the past. It should use technology the team is already familiar with. Avoid projects that have a higher risk of failure, such as ones that implement new technology or involve new, unproven team members. Even if a pilot project fails for a reason that has nothing to do with the change being piloted, there is a chance that your change will be blamed for its failure.

During the pilot project, keep careful records of any project problems or issues. Keep senior management in the loop for any serious problems—if it looks like the change is causing the team to miss a goal, it is better that the news come from you, and that it come as early as possible. It is helpful to adopt a scientific attitude toward the project: treat it as an experiment (preferably one with a high likelihood of success), and be as objective as possible about the outcome.

If your pilot project is successful, you now have a valuable publicity tool for your change. It is no longer "theoretical," because it was successful in the organization. And it's much less likely to be seen as "risky," because the change has shown to be successful on a project.

However, it is important to keep in mind that a pilot project is not necessarily a cure for an organization that resists change. The very characteristics that make a project a good candidate for a pilot can also cause it to be vulnerable to criticism. By selecting a project that is smaller in scale and less important than most other projects in the organization, you invite criticism that the tools and techniques you are piloting might work in a small and low-pressure environment, but would fall apart under more difficult conditions. They may claim that the new tools and techniques would never work for more difficult projects in which users, stakeholders, and clients are putting pressure on the programmers, when there are deadlines, and where the projects are much bigger. And they are not necessarily wrong—you have not stress-tested your changes. Just because a change is successful in the least risky environment possible doesn't mean that it will be successful in the rest of the organization. That's not to say that the pilot project is not an important tool; it simply has its limits.

### NOTE

More information on piloting changes can be found in The Art of Project Management by Scott Berkun (O'Reilly, 2005).

## Measure Your Progress

Measuring your improvements is a critical part of changing the way your organization builds software. Measurements provide a way to track progress, as well as a way to communicate this progress to senior management and the rest of the organization.

There are two important ways that most project managers want to improve their projects. They want their projects to cost less, and they want fewer defects in the final product. Showing improvement in both cost and quality provides powerful evidence that the changes you have made are working.

### MEASURING COST

The most common criticism that project managers receive when trying to improve the way they build software is that the new procedures and changes cost too much. Therefore, an astute project manager will gather the actual number of hours that the changes cost. This information should be gathered during a pilot project and any other time changes are implemented.

Every activity that you have inserted into the development process in order to build better software should be measured in terms of time and effort. You can track this information in a spreadsheet (see [Figure 9-1](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch09s02.html#appliedprojectmgmt-CHP-9-FIG-1)).



*Figure 9-1. Spreadsheet to measure the cost of improvements*

Each activity performed over the course of this project is measured in terms of the total calendar time that elapsed while the activity was performed ("Hours"), the number of people involved ("People"), and the total number of person-hours (see [Chapter 4](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch04.html)) that were expended in both preparing for and performing the activity ("Effort"). In addition, the week, activity name, and participants are listed. This is not a difficult spreadsheet to maintain—in the example above, the project manager only had to add one to three lines per week to the spreadsheet—but it is very useful for showing that the benefits of the changes were worth their costs.

This information is easy to gather, and it will be valuable when it comes time to judge whether the benefits of the improvements were worth the cost. You can put the cost of the improvement in context by using the project schedule (see [Chapter 4](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch04.html)—and if you do not have a project schedule yet, it is more valuable to build and maintain one than it is to gather this data!). From the schedule, you can find the calendar time elapsed over the course of the project, and the amount of effort performed over the course of the entire software project.

One goal is to show that the effort required for the improvements is relatively small. By adding up the effort in the spreadsheet and dividing by the total effort in the schedule, you can calculate the percentage of the effort that your improvements cost. Many of the tools and techniques—especially ones that are typically labeled "bureaucratic," such as inspections, code reviews, and developing a vision and scope document—require a very small percentage of the project's effort. And it is often not hard to point to specific results (like problems that were avoided by developing the vision and scope document, or defects that were found during an inspection or code review) that, had the tasks not been performed, would have clearly cost more time than they saved.

Another goal is to show that even though the improvements took time and effort, they did not add calendar time to the schedule (since the final due date was not delayed). This is not hard to do, if your organization has done another project of similar size or complexity in the past and you know how long that project took to build. If the tools and techniques were effective, that project should have taken more calendar time than your project. Subtract the total number of hours required to build the current project from the total number of hours required to build the similar project. If this result is much greater than the number of hours spent on the improvements, it shows that the improvements saved more time than they cost. (The earned value metrics in [Chapter 4](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch04.html) are also helpful when comparing two projects.

In addition to comparing projects, you can also compare tasks within a project. You can compare the hours or effort required for a specific task with the benefits of that task. For example, if another project in your organization took 3 months (13 calendar weeks, or 520 hours) developing a feature that eventually had to be scrapped because it did not meet the needs of the users, you can show that far fewer than 520 hours would have been needed to develop and inspect a vision and scope document that could have prevented the wasted effort.

### MEASURING QUALITY

One straightforward way to measure the quality of the software is to measure the total time and effort required to test the software. This metric covers all of the effort expended, from the time that the programmers deliver an alpha build that they consider complete to the time that the test activities yield results that the senior managers agree are acceptable for release (see [Chapter 8](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html) about the release readiness process). This measurement should include not only the testing activities, but also any work programmers do to fix defects that are found. If the testing team finds requirements or scope defects, this number should also take into account time and effort spent on activities performed by the project manager, requirements analysts, stakeholders, users, and anyone else involved in fixing that defect.

Any tools and techniques that you put in place in order to improve the way software is built should reduce the time and effort required to test the software. To use this measurement, use the schedule to figure out both the calendar time and the effort required to test the software as a percentage of the total calendar time or effort required for the entire project. You can then compare these numbers to other projects of similar complexity. If these numbers decrease over the course of several projects, your improvements are working. In many cases, the change will also lead to fewer delivered defects (but it can take a while to observe that effect).

It is important to select projects for comparison that match the current one in complexity. This means that the comparison projects should use similar technology, require similar expertise from the team, and preferably use many of the same team members. This works especially well when comparing maintenance releases of a single software project.

## Bring In an Expert

Sometimes the best way to make sure that your changes are implemented effectively is to bring in an expert. There are many consultants who will assist in improvement efforts and train your organization to implement the tools introduced in this book. Sometimes, corroborating your improvement initiatives with an expert's opinions is just what people in your organization need, to feel reassured that the ideas have merit.

Experts and consultants are especially helpful in training people in a wide range of specific techniques, including estimation, inspections, code reviews, unit testing, and project planning practices. They can also be pivotal in helping to establish metrics, testing, or requirements efforts. What's more, there are many pitfalls that inexperienced project managers and organizations can fall into: experts can identify these pitfalls and help you avoid them. In this way, bringing in a consultant or expert can more than pay for itself.

Perhaps most importantly, experts' training sessions are morale boosters—there is little that can get a team more excited about a new technique than training them all at the same time on it. Everyone feels like they have a common understanding, and the culture changes to accept that idea much more quickly than it would have, had it only been introduced by someone internally.

Don't be afraid to go outside of your own organization to get training. Some project managers are tempted to develop their own training programs, but they often underestimate the effort involved. Doing it right is extremely time consuming, and it's likely that you won't do nearly as good a job as someone who has used and refined his training many times. In addition, you'd be reinventing the wheel—just like programmers with NIH syndrome do with regard to new practices.

When an improvement effort seems to lose some steam, an effective way to get it back on track and renew the organizational commitment is to engage in group training about process improvement techniques, and raise general awareness about your efforts. Hearing from an outside authority figure that you are on the right track can be enough to gain support from those who might not have agreed with you in the past.

When you bring an outside expert into your organization, the fact that this person is paid to give advice will cause many people to treat this person as an authority figure. People naturally defer to an authority. In many organizations, bringing in an expert will cause people to change their minds, even if that person is saying the same thing you already said. Even if you are in an organization with NIH syndrome, bringing in an outside expert or a consultant can help bring people around. By hiring an expert, the organization already has committed to listening to the person. The expert's advice can immediately become "how things are done here," because money has been authorized and spent on the expert. Once money has been spent to obtain an expert opinion, people are much more likely to take suggestions for change seriously.