

3

People, Process, and Tools

Successful software projects balance the emphasis on people, process, and tools to fit the work at hand. These are not the only characteristics that need balancing on a successful business project. You can probably think of other attributes that are important. We have found that people, process, and tools (or managing technology) are key factors in making a software project successful. How you balance them depends upon your environment and project. This is where the forces of cost, quality, schedule, market conditions, and many others come into play.

This chapter describes our philosophy about people, process, and tools on software projects. We provide specific suggestions, in the form of guidelines, to help you on your own project. We hope this chapter helps you understand where we are coming from and what we value.

Few projects have a perfect balance of people, process, and tools. Usually, one or more will be emphasized over the others, often with good reason. However, when the emphasis is too great on one of the three, project success is seriously jeopardized. We will consider what can happen when things get out of balance at the end of the chapter.

People

Let's get something clear: *Software development is intrinsically a human activity. People are the most important factor for the success of your project!* In fact, people are the difference between success and failure for most team-based endeavors.¹

¹ We originally talked about people being the most important “resource” for your project. When we reviewed this chapter, Chris pointed out that speaking of people as resources takes away the real

There are many books about effective management, including books that are specific to software project management. See the Recommended Reading list at the end of the book for some of our favorites. If you are a software project manager or program manager, these books are great resources for you. We will not address management issues in detail in this book. We spend most of the chapter, however, talking about people. The rest of the book provides details about how we used process and tools on our project. Whether you are a project manager or an individual contributor on your team, you should read this section and consider how your current project team operates.

Harry and Gwen

We introduce this section on people with a short story. You may have heard about or experienced a similar situation, yourself. We assure you that this is a true story that one of our team members lived through.

People can affect a project's success. This story about two smart senior engineers on a five-person project happened a few years ago. There was a critical part of the system that needed to be written before we could move on. The first programmer, Harry,² took it upon himself to write this bit of code. Since Harry was the project leader he felt this was his right. Gwen, the other programmer, thought she had a better solution, but couldn't convince Harry of this. One day Harry wrote some code. That night Gwen checked Harry's code out and rewrote it to do things her way. The next day Harry saw the changes and changed the code back—after having some choice words with Gwen. That night Gwen changed it back to embody her solution. This pattern went on for a couple of weeks. This was bad for the project because it delayed getting some critical code

importance of people. We all agreed. Changing the names of departments from “personnel” to “human resources” has helped add to the dehumanization of the workplace.

² As you would expect, the names have been changed to protect the innocent and guilty.

complete. What was worse, was that each time Harry or Gwen changed the code, they broke the rest of the system for the rest of us on the team.

Egos and technical reputation came first for Harry and Gwen and the project and team came in a distant second. As you might expect, this project was never completed and it was cancelled without ever delivering a release to the customer. With a slight change in the personnel mix on the project and better management of the personal characteristics of the team, we might have been successful. At least we would have had a better chance.

It's About Communication

At Rational Software, we use the tag line “Software development is a team sport” to describe an approach to software development. As a company, we build tools to help teams. However, the key to success is recognizing that it takes a team to build software, not just a group of individuals with different agendas and different values. Getting the people on your team to work effectively is mainly an issue of establishing good communication between the team members. This leads to trust and respect, which are also necessary people ingredients.

Once you have two or more people working on the same project, you have to deal with people issues. Great managers are the ones who enable the people on their team to work most effectively together. Being a great manager is hard work. It's more than just giving your team pizza and great T-shirts—it's helping them to respect each other's abilities, work off each other's strengths, and help them work effectively as a team.

There are some simple people-oriented project guidelines we recommend you consider for any project, especially small projects. The guidelines might seem most appropriate for project leaders to consider, but all team members should look at their project team in light of the guidelines. The guidelines may seem obvious, but we feel they are worth stating explicitly. The following subsections present each of these guidelines.

Team Composition

How do you put together a good team? You want people who can get the job done. Does this mean that you need a team of superstars? Not necessarily. The sports world provides examples of teams that have the highest payrolls, the biggest stars, yet end up out of the running for the championship year after year. And, of course, the coach is the one who gets the blame.

Some teams don't have any proven superstars but somehow manage to put things together and come in first. This admittedly doesn't happen often, but we love the stories where the underdog overcomes the odds to become the champion. They inspire us and enable us to dream of glory.

The great teams—the really great teams—are more than a collection of superstars. The great teams have a mix of superstars, talented rookies, role players, great coaches, and a great organization. Furthermore, the winning teams keep improving. Teams in business, especially software development teams, are no different than the sports teams. Let's look at what we mean.

In software we have superstars. Some of them are great coders who can just sit down, start typing, and produce perfect code. Some are great architects who can take complex problems and design the right solution. Some are great testers who are able to quickly identify defects and help the developers build better software. Each makes a contribution to the success of the team.

Our experience has been that a software team composed of just superstars cannot be effective for an extended period. There are several reasons for this. The main reason is that egos clash. Regardless of how well people “play well together,” there are times when they don't. The Harry and Gwen story earlier in this chapter is an example of this. Experienced programmers have earned their reputations for technical acumen. They like to be the ones doing the hard technical work. They want to have others do the *uninteresting* work.

Staff your project with people possessing different skills and experience so they complement each other.

Project Guideline 1: Get the right mix of people on your project.

Our first project guideline deals with the composition of the team. It is no easy task to put together a team that has the right mix of skills and experience. As the team forms, you should look to see if:

- There are senior team members who can lead and teach the more junior members
- The senior members have experience with projects similar to your current project
- The team possesses technical expertise for the most critical aspects of the project

For example, consider beginning a project to develop a new Web site for a company. The site must present the company's catalog to the customer and allow the customer to place orders and enter credit card information. Would you trust this project to a group of engineers just out of school? They may know the basic technologies involved, but they would lack experience in the business aspects of your system. You would make sure that you had one or more experienced engineers familiar with your company's business systems and data assets. You would support them with other, possibly junior, engineers who could provide general programming and technology capabilities.

Provide a Learning Environment

Software developers get excited about solving problems and learning new things. One of the worst fates for a programmer is to implement a system where there are no challenges

to meet or new technologies to learn. Maintaining a technical edge is the same as maintaining a competitive edge for software developers.

The most desirable companies for programmers to work in are those that provide a learning environment. Programmers will often join a company because they are promised the opportunity to work on something new. You can make part of your project a learning environment by making sure that everyone on the project has the opportunity to learn. Experienced developers can learn new technologies. Less experienced team members can learn about how software is built from the more senior people.

The next project guideline addresses learning on the project. We recommend that you ensure that there is something to be learned by every member of the team. When you consider the possibilities for learning, it can help you decide which people are most appropriate for your team.

Make sure that every team member has the opportunity to learn on the project. Things to learn can be technical, organizational, or managerial.

Project Guideline 2: Provide a learning environment.

There is a hidden trap in this guideline. Sometimes people get so involved in learning that they fail to deliver the product. Project managers need to be alert for the team members who spend all their time learning and forget the primary goal, to deliver working software to the customer.

There are times when a team member does not want to learn anything new on a project. That person has different motivations for joining part of the team. Project managers should not force everyone to have a learning goal. In general, a project team where people want to learn is more robust than one where people are just trying to get through the project, but there are times when we all want to just do a good job and send the learning parts of our brains on vacation.

Trust, the Glue for the Team

In his excellent book *The Phoenix Agenda*, John Whiteside presents a twelve-step approach to empowering teams. The most important one, in our opinion, is to generate trust. A lack of trust between team members can quickly cause a project to deteriorate to the point where there is little chance for success. The example cited earlier about Harry and Gwen is an example of a lack of trust. Neither engineer trusted the other one.

People often misunderstand what we mean when we talk about trust. Trusting your teammates doesn't mean, for example, that you give a new programmer a critical task, requiring technical skills the programmer doesn't have, and trust they will get the job done correctly. It means that you trust programmers to tell you they are not qualified, or they need help. It means that when someone accepts a task, you trust them to do the job right. You trust that they will ask for help if they need it. You trust that the team will provide help when asked.

Generate trust on the team. Help the team members trust and respect each other.

Project Guideline 3: Generate trust.

One of the best stories about trust and teams was told by Susan Butcher, the four-time winner of the Iditarod dog sled race in Alaska. Susan was the guest speaker at the 2001 Rational Users Conference in Denver, Colorado. Susan told of the time when she was out on the trail over a frozen body of water. Her lead dog kept turning to the right, off the trail. She kept demanding that the dog come back on course. This went on for some time until Susan just gave up and let the dog lead. Almost as soon as the dog took the team off the course, the ground where they were heading collapsed into the freezing water—certain death for Susan and the dogs. The words of wisdom Susan gave to us about this

incident were: “Sometimes you have to lead, and sometimes you have to let the team lead.” What she was really saying is that you have to trust your team.

Disagree Constructively

Some people think that a high-powered team is one that doesn't disagree. Experience has shown us that this is false. Most of the effective teams we have worked with have had significant disagreements between team members about how things should be done. This has, in fact, led to solutions better than any individual had proposed. Healthy disagreement provides the team with a synergy.

Disagreement is good. But when disagreement gets personal and is allowed to get out of hand, it destroys a team. Effective teams are those that have established a way for team members to disagree constructively. They provide a forum for dialog and exploration as well as a way of resolving disagreements. If the team has trust, disagreements are easy to deal with. People express their opinions, you trust that they are concerned about the team's success, and you work on an agreement. Even if you can't work out an agreement, you have a way of deciding on a course of action and everyone gets behind it.

Allow team members to disagree. Provide a way to resolve disagreements.

Project Guideline 4: Allow disagreement.

We can look once again at sports teams for examples of how to handle disagreements. Each team member has some ideas about how best to win. In basketball, the high-scoring shooters want an offensive game where they will pump in basket after basket, not worrying about the other team's score. They are confident that they can meet any challenge by shooting more. This “run-and-gun” style can be effective in many, but not all, cases. There are times when you need to control the ball and slow down the game's pace. You then give the ball to your best ball handlers and put in defensive players who can steal the ball from the opponent and block shots. The coach has the responsibility for

assembling a game plan and motivating the team to execute to the plan. The coach also has the responsibility to alter the plan when opportunities arise, or when the existing plan is not working. If a player notices that a defenseman on the other team is playing in a way that allows him to shoot for easy baskets, he tells the coach. The coach makes the final decision. If the situation requires more immediate attention, the team communicates on the floor and adjusts as necessary. Often, a team captain makes the final decision.

The Power of Requests

What motivates you more: when someone *demands* or *requests* something from you?

Everyone prefers a request to an order. Requests are the topic of Project Guideline 5.

Make requests, not demands.

Project Guideline 5: Ask, don't demand.

When we say that you should make a request, we don't just mean any type of request. Make a *well-formed request*. Gary learned about well-formed requests from John Whiteside when John was Gary's mentor on a project at Digital Equipment Corporation. John calls this type of request a *precision request* in his book referenced earlier in this chapter.³

A well-formed request has certain attributes:

- It says *what* needs to be done.
- It says *when* it needs to be completed.
- It says *who* should perform the work.
- When appropriate, it states *where* the work will be performed.

Let's look at two examples of requests. Which is well-formed, and which isn't?

³ Chapter 6 in *The Phoenix Agenda*.

1. Liz, will you review the Vision document and produce the first draft of the Glossary based upon the domain-specific words in the Vision? Can you send me the draft of the Glossary by noon on Friday?
2. Gary, we need to see some results on the architecture soon.

Clearly, the first request is well-formed. It identifies what needs to be done, a Glossary based upon the Vision document. It states who should do the work, Liz. It states when the work is needed, noon on Friday.

The second request is very vague. The only attribute of the well-formed request that it contains is the name of the person. It does not qualify as a well-formed request at all.

There is one other type of statement you should avoid when you are making well-formed requests. Consider the following statement:

John, have your code ready to go by Monday morning.

This is not a request at all. It is a demand. Demands are ineffective. They often have the opposite effect of what you want. People do not react well to demands. They feel they have no control of their own destiny. Remember one thing about well-formed requests: *it isn't a request if they can't say no!* This is important. If you make a demand, you do not allow the other person to give an honest response if they cannot meet the demand. If you follow Project Guideline 3, Generate Trust, you need to trust your teammates to tell you honestly whether they can do the work you request. If they cannot meet your requirements, then you can negotiate a different delivery time, ask someone else, or decide upon a different set of deliverables. If you do not have an environment of trust on your team, these negotiations are impossible.

There is another benefit of using requests. By accepting a request, a person takes ownership. If someone asks you to do something, and you agree, you have a feeling of responsibility to deliver what you promised. If someone tells you that you must deliver something by a certain time, you have no ownership. The task may be impossible. You

may have already scheduled a day out with your family. There are any number of reasons why the work may not be done by the specified time. If you are not able to freely agree to deliver the work, you will not take ownership. If you are free to accept or reject the work, you will do whatever it takes to meet your commitments.

It takes practice to make well-formed requests. This is time well spent. Whiteside presents data showing that the number of requests satisfied is approximately constant. This means that a team with an 80% completion rate satisfies 8 in a week that 10 well-formed requests are made, and 40 in a week that 50 are made. This leads to the obvious conclusion Whiteside states: “To increase your productivity, increase the number of precision [well-formed] requests you make.” Clearly, there is a limit to the amount of work that any team can do. When you use well-formed requests, you tend to converge on an optimal request rate for the team.

If you decide to use well-formed requests on your team, set up a simple system to track requests. You can do this with a simple text file or spreadsheet. You might try something more sophisticated, but it is not necessary. Tracking requests allows you to see how well the team is doing with well-formed requests and if there are any problems on the team.

Recognize Achievement

How do you feel when someone says “Thank you?” Most people feel good, as long as they feel the thanks are sincere. They know they have been recognized for something they said or did for someone. One of the most important skills you can develop as a member of a team is the skill of saying thanks; or to put it differently, the skill of recognizing achievement. This is our next guideline.

Recognize achievement—sincerely and often.

Project Guideline 6: Recognize achievement.

Sometimes you need to be creative about recognizing achievements. Some people are uncomfortable when they are publicly recognized. You may have to deliver the kudos privately. That's okay. The important thing is that you say thanks and you are sincere about it.

You can overdo recognition. When recognition is not sincere, or it is done habitually for day-to-day accomplishments, it loses its effect. You should say thanks for people doing their job well, even if there are no outstanding achievements or hurdles they have overcome. Just don't do it every day. If you went to work every day and your manager greeted you with "Thank you for the work you did yesterday," how long would it mean something to you? Not very long.

Another example of overdoing recognition can be seen in many companies today. These companies provide different "benefits" to their employees, for example, free lunches, popcorn, pizza, soft drinks, T-shirts, and so on. People feel good about getting these things. However, "familiarity breeds contempt." After a while, people begin to comment that their favorite drink isn't provided, the T-shirts are the wrong color, and a litany of other complaints. What was originally meant to be a way of saying thank you becomes an annoyance.

Be careful and creative about how you thank your team. Make it special. One of the simplest, most special ways is a heartfelt "Thank you."

Process

In some respects, this whole book is about process. In this section we present the process-oriented project guidelines that do not require much contextual information. Those that do require more context in their description are presented in subsequent chapters.

The Prime Directive

Let's start with our fundamental guideline about process, also known as the *prime directive* of process. This guideline is shown in Project Guideline 7.

Only do those activities and produce those artifacts that directly lead to delivering value to your customers and stakeholders.

Project Guideline 7: The prime directive of process.

The real trick for the software engineer is trying to figure out how to determine which things are really necessary. We have a simple way of determining this. We ask the following question about everything we do: *If we don't do this activity, or produce this artifact, will anything bad happen?* If the answer is “no,” we don't do it. That seems simple enough. But you have to be brutally honest. It is very easy to fool yourself and tell yourself what you want to hear, rather than recognize reality. You can convince yourself that nothing bad will happen if you don't document your design anywhere except in the code, or address architecture issues early. You may be able to produce software that works today, but is not sustainable for future releases.

Bob Martin, from ObjectMentor, has a slightly different way of stating the question above. He says that you shouldn't produce the artifact or perform the activity unless something *really* bad would happen.⁴ We can quibble over how bad something has to be before you do something, but you get the basic idea.

Address Risk

Another way of thinking about the prime directive leads to the next project guideline. If we're going to do something to avert something bad, then what we're really saying is that we need to make sure that our process addresses risks. Think about it. Why would you do something if it doesn't address a real risk that you face? Now, not every action you take on a project is directly related to a known risk. That is okay. There are things that you

⁴ Bob stated this in a posting to the XP mailing list in 2001.

have to do in order to ship your product. If you don't do them, then a potential risk may become an actual risk.

Make sure you perform activities and produce artifacts that reduce risk.

Project Guideline 8: Have a risk-driven process.

Don't Reinvent the Wheel

So, we have to make sure that we do the right things, and don't do unnecessary things. That's great to say, but what things should we consider? If you have ever tried to establish a process in an organization, you know what a difficult job it can be. You may have a wealth of software engineering experience, but it takes time and a lot of effort to put together a coherent, consistent description of what needs to be done. What should you do? Our next guideline addresses this.

Base your process on proven practices, techniques, and principles.

Project Guideline 9: Start with a proven foundation.

We're concerned about software reuse. We should also be concerned about process reuse. Starting from scratch and reinventing the wheel, so to speak, is never cost-effective for the following reasons:

- There is too much information to wade through and absorb.
- Once you do uncover the information, you still have to decide how the different techniques work together and how to present the information in a coherent, consistent way.
- Not every idea or technique really works for your project.

Let's consider each of these separately, starting with the amount of information. There is so much information about technology, process, management, and so on, that it is impossible for any one person to keep up to date with it. Gary reads the mailing lists for XP and agile modeling. About two hundred mail messages appear each day on just these two lists. There is a lot of noise on these lists, but if you want to understand what is happening in the communities, you need to read them. There are usually one or two messages that really have something interesting each day. Add in the articles and books on XP and agile software development and it's hard to keep up with just one area, let alone multiple areas of interest. Now assume that most people have "real jobs," such as developing software. By using a process framework or reusing a process instance, we let others mine the field and distill the interesting information for us.

Next we have the problem of producing a process that is well-defined, consistent, and understandable. What format should you use to present the process to your team or organization? Are there examples and instructions? Is each person on the team forced to look at all of the details, even if they already know what they're doing? There are many more questions and issues to address. When you start from scratch, you have to think of everything. This situation is similar to the software developer who can either develop a set of cooperating classes from scratch or use an object-oriented framework. Smart developers will always start with an existing set of classes and customize them to fit the situation when they can. Smart process engineers (people who configure processes for development groups) start with an existing process and customize it for their needs.

Finally, when you have a set of practices, principles, artifacts, and so on, you need to decide when they apply. We often talk about using best practices. What does that really mean? Are there universal best practices that apply in every situation? Perhaps; however, many, if not most, apply best in certain cases and you need to know when to use the practices rather than blindly follow them all the time. This means that you need to

configure your process for specific contexts and projects. See Project Guideline 10 for more on this subject.

Make Your Process Yours

Tailoring your process means more than just selecting a subset of the practices, activities, and artifacts that might be specified by your process. It means you should select those that fit together in a consistent way. It also means that you should feel free to modify them by removing things that are not necessary (the prime directive) and adding things when necessary. This brings us to our next guideline and its corollaries.

Tailor your process for every project. Don't assume that every aspect of a process will automatically apply to your situation.

Project Guideline 10: Configure your process.

Accepting process guidance blindly is a sure recipe for disaster. You might think of this in the context of a road map. If you want to travel from Lexington, Massachusetts to Cupertino, California, there are thousands of possible routes. You need to decide which is appropriate for your needs. If you go to some of the Web sites that provide driving instructions, you are offered a choice of routes, depending upon your ultimate goal.⁵ You may choose one process if you are under extreme time pressure because of a rapidly changing market, and another if you have significant overhead imposed on you by customer requirements and regulations.

One thing we have found in our experience is that people tend to configure too much into their process. They think that if they are not sure whether they need something, they should do it. We recommend doing just the opposite, as stated in Corollary 10.1. If you

⁵ The MapQuest site <http://www.mapquest.com> asks if you want the shortest distance or shortest time route. You could consider many other parameters, based on the sights you want to see or other interests.

omit something you really do need, it usually shows up quickly and you can add it. If you add something you don't need, you often will just keep doing it and never remove it from your process because you don't know that it isn't necessary. The choice is yours, but we prefer doing less rather than more. We find that the risk is usually small.

When configuring your process, if you are not sure whether to do something, don't do it. If you later discover that you need it, you can add it.

Corollary 10.1: When in doubt, leave it out.

How do you decide if you are doing something that's unnecessary or not doing something you need? If you are doing something that is not necessary, you can usually spot it during your iteration reviews. During these reviews you not only assess the work done and progress made, but you evaluate the effectiveness of your process. Some symptoms that can indicate useless work are:

- You produce an artifact that no one looked at.
- You did something that did not lead to delivering software or reducing risks.

Always ask yourself, why am I doing this and who cares?

Configuring your process is more than just deciding what to do, when to do it, and how to do it. It also requires you to decide how formal you need to be. Once you have decided that you need to perform an activity or produce an artifact, you should decide how rigorous you have to be. Should you use a tool? Will the artifact need to be modified continually? Who will look at what you produce? These are some of the questions you have to ask yourself in order to determine how formal you need to be. We find that small teams are usually less formal than large teams. This makes sense because there are fewer lines of communication that need to be established, and those that do exist are usually quite informal—possibly one team member leaning over into another team member's

work area and asking a question. Whatever you decide, don't be formal for the sake of formality. Don't lose sight of your primary purpose, to deliver working software to your customer.

How do you configure your process? There are several approaches to this. We find the easiest is to configure it by focusing on artifacts. That is, we decide what artifacts are necessary to help us reduce risks and deliver the software, then base our process on producing the artifacts. If we know what we have to produce, we can figure out how to produce it. In the next chapter you will see how we configured the process for the PSP Tools project.

Configure your process based upon the artifacts you need to produce. Once you know which artifacts you need to produce, you can then figure out how to produce them.

Corollary 10.2: Take an artifact-centric approach to process configuration.

We will see more process-oriented project guidelines in later chapters. The ones presented here are the ones that are most important to understand.

Use Your Brain

We have one final process-oriented guideline, Project Guideline 11. It almost seems silly to mention it, but we have often been surprised by the number of projects that just forget to apply it.

Having a process to guide you in your development efforts does *not* absolve you from using your brain and applying common sense.

Project Guideline 11: Apply common sense liberally.

People, especially software engineers, are paradoxical. On one hand, they want to be creative and hate being told what to do. On the other hand, they want someone to show them the way to do things. The key is that they want to be given the freedom to do things they like to do—the creative activities that challenge their intellect. They are looking for step-by-step instructions for the simple, repetitive pieces they would rather not have to think about. They also want to make sure that the creative parts dominate their time and that they don't spend one second more than necessary on the drudgery.

If you don't consider Project Guideline 11 when configuring your process, you may end up with a process that is too heavy: process for the sake of process. Instead, you want your process to account for the skills of the people on your team, along with your project and organization needs.

When using a process, even if it has been configured, you still need to apply common sense. When the process prescribes something that does not support the people or the ultimate goal, you have the responsibility to question the process element and ensure that it adds value to the project.

Tools

Tools are perhaps the most interesting of the three keys to project success. Adopting tools requires a commitment of time to learn and use them. Sometimes we use tools ineffectively by trying to do too much, or not looking at whether the benefits outweigh the costs. Finally, there are times when we just use the wrong tool. There is an old saying that goes, "If you only have a hammer, everything looks like a nail." We have seen software projects work with only "hammers" and try to build skyscrapers. It just does not work.

We have just a few tool-oriented project guidelines. They are variations on a theme. The first one, Project Guideline 12, sets tools in context of the three keys to successful projects.

Ensure that your tools support the process and people on your project.

Project Guideline 12: Make sure tools are worth the effort.

Tools are not an end in themselves. By definition, a tool is something that helps you do your job better. It is good to have a well-stocked tool set and be able to select the ones that will support your efforts. However, you need to make sure that you don't get too wrapped up in using tools for their own sake. We could restate Project Guideline 12 this way: *Just because you have a tool doesn't mean you have to use it.* This doesn't apply only to software tools and software projects. If you look around, you find that we are a society of gadget users. Look in your garage, attic, closets, and other hiding places and you'll find lots of gadgets that are not used, or were used for a while with poor results. Don't fall into this trap with the tool usage on your project.

The advice from Project Guideline 12 leads us to consider a corollary: Just because you are going to use a tool does not mean you have to use all of its features. We believe that there are few, if any, cases where all of the features of the tools we have are really necessary. Sometimes it takes much more effort to use the feature than to adopt another, simpler technique, or skip it all together.

Use only the features of your tools that make you more efficient.

Corollary 12.1: Use only the tool features you need.

The final project guideline relates tool usage to people (see Project Guideline 2, Provide a Learning Environment). Learning to use the tools properly is important, and often ignored. We assume that the team members will learn how to use the available tools as they need them. This may be so, but they need to have instruction. The instruction can

take many forms: classroom, mentor, books, or online training. Just don't neglect the real time spent in this effort and think it won't affect your project.

Provide instruction on tool usage and schedule the learning time into your project.

Project Guideline 13: Provide time to learn tool usage.

The systems we attempt to build today are complex. We need tools that address the complexity and help us understand it. Few programmers today use just a text editor and a compiler. They use integrated development environments (IDEs) that help them construct the components for their application programs. Testers use tools that automate much of the repetitive work of testing, freeing up time to concentrate on developing effective tests.

Using tools effectively takes time. While most people can learn to use a tool with some degree of proficiency by themselves, it is almost always worth the cost to provide some help to them in their learning effort. We only need look around to see the difference in the productivity between *power users* of the tools and those who have only a basic knowledge.

Building Your Own Tools

When should you build your own tools? Should you build your own tools? Project Guideline 14 states our feelings on the subject.

Build your own tools when necessary, but make sure that the benefits outweigh the cost.

Project Guideline 14: Build tools when necessary.

Every project needs to customize its process. Projects also have to customize the tools in their environment. In many cases, the tools available to you are excellent, general-purpose tools, but your team has special needs that the tools were not designed to address. If there is a good business case, try to extend existing tools, or build your own tools to do the job. Before you embark on a tool-building subproject, however, we suggest you consider the following:

- Do you really need the features that a special tool will deliver? Is the added functionality something that is nice to have or is it the case that you cannot successfully complete your project without the new features?
- Can you afford the cost of building your own tool and maintaining it? If you build a tool, you must maintain it, just as with any other software product. Project teams and organizations frequently fall into the trap of deciding to build a new tool without considering the full cost associated with it. As soon as you begin to use an internally developed tool, you establish a reliance on that tool. Any problems with the tool must be addressed, and you will probably find that you want to make extensions. Tool-smithing can become a full-time job for one or more team members. This may be all right, but you must be willing to accept the costs.
- Do you need to build a new tool or can you extend an existing one? Many software development tools are designed to be extended. The tool vendors provide an API and documentation for this purpose. This may be your best choice when you decide that you need new tool capabilities for your project.

What Can Go Wrong?

Well-run projects balance people, process, and tools (as well as other things). You don't achieve a good balance by focusing one-third of the emphasis on each area, as shown in

Figure 3.1. Instead, consider the parallels to a well-balanced diet. You need to eat some amount of protein, fiber, and fat every day. The exact amount varies with each person. Similarly, the right balance of people, process, and tools varies with each project.

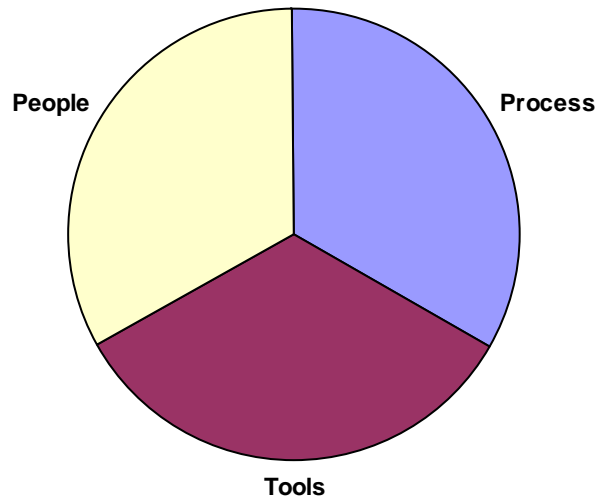


Figure 3.1 A misguided view of balance.

True balance occurs when you emphasize the factors in a way that helps you deliver value to your customer in a predictable, timely manner. Your project team members are working hard, enjoying their work, and gaining knowledge and satisfaction. You have a process that helps each person know their responsibilities, provides help when needed, and is minimal, or the right size, for the team and the project. You have tools that support the process and the people to help them get their job done.

What can go wrong? Usually you can either pay too much attention to one of the items or too little. In both cases, you will find that your team is not making progress. Whenever this happens, you and the team must determine the reasons for lack of progress and address them as soon as possible. This is where a project manager earns his or her salary. A good manager will be able to accurately measure progress and determine if the team is

performing at a less than optimal rate. The manager will then be able to work with the team to communicate the problems and work out a plan for getting back on track.

In the remainder of the book you will find some problems the PSP Tools team encountered and things we tried to correct our problems. Some of them worked. Some did not. They will all give you something to think about and determine if they are appropriate for your team.

Summary

People, process, and tools are important to the success of projects in almost every industry. In the technological industries we work in, it is perhaps even more critical. Yet, intentionally or not, we tend to ignore these issues and assume that they will take care of themselves. We hope that any problems will go away and take no action to resolve them. The time you spend attending to people issues is time well spent. Of course, you also need to provide just enough process to help your team members rather than hinder them, and equip them with the right tools for the job.

Many people have written about people issues and team development. In this chapter we have provided some of our favorite guidelines. They are not exhaustive, but they can't be. People, process, and tools change continually and, to be good managers and team members, we have to change as well.