CONTENTS

working effectively, leads should pose the problems they're trying to solve as increasingly refined questions. A lead shouldn't ask, for example, "How can we consistently hit our ship dates?" which can result in a number of undesirable solutions. The lead should instead ask a more specific, more beneficial question: "How can we consistently hit our ship dates without hiring more people and without forcing the developers to work overtime?" Leads should try to incorporate negative feedback loops into the strategies they develop. And when they present work strategies to the rest of the team, they should be sure to remind the team that even a good strategy or guideline won't necessarily be effective in every situation.

Projects can go astray in so many subtle ways that leads must never let projects coast, assuming that their projects are on course and will run themselves. To keep a project running smoothly, a lead must constantly monitor the project, looking ahead and taking care of problems while they're still small. To keep a project on schedule, a lead should ask this question each day: "What can I do *today* that will help keep the project on track for the next few months?" By asking this question every day and seriously looking for answers, a lead can foresee all sorts of problems that might otherwise blindside the project. To prevent wasted effort, a lead should assess every request in order to identify the real problem or goal and should be sure that every task fulfills the project's goals and priorities. Some tasks, such as meeting the marketing team's request to fill out a feature set, or implementing a free feature that has popped out of a programmer's design, might not be at all strategic. A good lead learns to say No.

If a lead wants to get a software development team going on a creative roll, he or she must create a development atmosphere that fosters that kind of enthusiasm. Unfortunately, as companies grow from small mom-and-pop shops to corporate mega-shops, the amount of non-development work that programmers are routinely saddled with rises dramatically. The lead should work to eliminate unnecessary reports and meetings and other corporate processes that hinder the develop-

described in Chapter 5, which could give each team member at least six important new skills a year. Programmers can and do pick up skills in the normal course of the job, but their growth is much slower in that passive approach to learning. By ensuring through work assignments and overt educational goals that programmers actively learn new skills, leads help the project and the company and advance the programmers' careers.

Increasing a team member's skill through active learning is great, but leads can get the most impressive results when they focus on correcting harmful attitudes and promoting beneficial ones. The effects of a new attitude sweep across all work that a programmer will do. That's the leverage behind good attitudes. Chapter 7 takes a hard look at the common programmer attitudes that work to the detriment of project success: bugs are inevitable, I'll fix bugs later, it'll take too much time to do things right, it's good enough for users, it's better to give the user something than nothing, we'll do our thing and you do yours, it's just for in-house use…

When a project schedule starts to slip, a natural reaction is to hire more people and force the team to work longer hours. But throwing more programmers at the project and forcing everybody to work overtime won't correct the underlying problems that caused the project to slip in the first place. If a team is working 80-hour weeks to meet a 40-hour schedule, something is seriously wrong. The lead needs to go after causes and (sometimes) to protect the programmers from assumptions—their own and upper management's—about the tonic effects of long hours. Hiring more people or demanding long hours only masks the problems affecting the project. Leads should find and fix the problems, not cover them over.