

# CONTENTS

PREFACE	xi
---------	----

INTRODUCTION	xv
--------------	----

*If a software project is to be successful, every team member must understand the principles, guidelines, and strategies that will result in quality software shipped on time. This book is for every team member. It's a companion to Writing Solid Code, which focused on the most serious "bug" in the development process: too many software bugs. The advice in this book fine-tunes the development process, focusing on the techniques and strategies that software teams can use to become consistently successful. This book contains many anecdotal examples, most of them drawn from experiences at Microsoft. To make the examples easier to follow, the introduction provides a brief account of how software development projects are organized and how they proceed at Microsoft.*

1 LAYING THE GROUNDWORK	1
-------------------------	---

There are a few principles that all successful software project leads keep in mind. Among the foremost is the idea that the programmers should be working only on tasks that either directly or indirectly improve the product. It's the lead's job to clear the way for the primary work of the other team members by ruthlessly eliminating work that gets in the way of improving the product—going overboard on status reports and meetings, for example, or developing features that are not strategic to either the product or the company. To make it easy to determine which tasks are strategic and which are wasted effort, leads should create detailed project goals and priorities. The more detailed the goals and priorities are, the easier it is to spot wasteful work.

2 THE SYSTEMATIC APPROACH	23
---------------------------	----

It's amazing how a relatively trivial work habit or process can produce a major difference in results. Ideally, the habit or process will take little or no effort to put into practice and its effectiveness won't depend on the skill levels of the programmers who use it. To elicit the best strategies for

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.

### 3 OF STRATEGIC IMPORTANCE\_\_\_\_\_45

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.

### 4 UNBRIDLED ENTHUSIASM\_\_\_\_\_.\_\_\_\_\_73

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-

## CONTENTS

ment effort. The simpler such processes become, the better. If programmers are given the opportunity to work unhindered by overblown corporate processes, they have a much better chance of catching a creative wave and moving the project forward. The critical point is that leads should always work to address their actual, rather than formal, needs. Asking for a report or holding a meeting is a common way to gather information, but if there are other, more effective ways to gather information (and there are), why burden programmers with reports and meetings?

### 5 SCHEDULING MADNESS \_\_\_\_\_ 91

In most companies, the development team needs to maintain a schedule so that other groups in the company can coordinate their work with the programming effort. At the very least, the marketing team needs to have some idea of when they should start advertising the product. But as important as schedules are for coordinating the work of the various product teams, they can have a devastating effect on development if they are not devised and used wisely. An unattainable schedule can demoralize the team and ultimately kill productivity. A schedule that is merely too aggressive can lead to slip hysteria, in which programmers take shortcuts to meet the schedule in the short term, jeopardizing the product over the long term. A schedule should be aggressive enough to keep the project running at a brisk pace, but if it is too aggressive, programmers will make stupid decisions despite their better judgments. Any programmer who has decided that he doesn't have time to thoroughly test his code is guilty of putting the schedule ahead of the product. By using "milestone scheduling," leads can not only coordinate better with other teams but also make projects much more exciting and foster creative rolls in which teams crank out high-quality code at a prodigious rate.

### 6 CONSTANT, UNCEASING IMPROVEMENT \_\_\_\_\_ 107

Leads can streamline the development process to a point at which every team member is focused only on strategic work. But if leads want their projects to really take off, they have to focus on training so that every team member is regularly learning a wide variety of broadly useful new skills. One method for ensuring that team members actively grow is to align personal growth goals with the two-month project milestones

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.

## 7 IT'S ALL ABOUT ATTITUDE \_\_\_\_\_ 125

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...

## 8 THAT SINKING FEELING \_\_\_\_\_ 151

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.

EPILOGUE	A WORD ON LEADING _____	171
REFERENCES	_____	175
INDEX	_____	177