The XP Series

Planning Extreme Programming

Kent Beck

Martin Fowler

Foreword by Tom DeMarco

Planning Extreme Programming

Kent Beck
Martin Fowler

Illustrated by Jennifer Kohnke

ADDISON–WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and we were aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers discounts on this book when ordered in quantity for special sales. For more information, please contact

Pearson Education Corporate Sales Division
One Lake Street
Upper Saddle River, NJ 07458
(800) 382-3419
corpsales@pearsontechgroup.com

Visit AW on the Web: www.awl.com/cseng/

Library of Congress Cataloging-in-Publication Data
Beck, Kent.
Planning eXtreme programming / Kent Beck, Martin Fowler.
p. cm. (The XP series)
ISBN 0-201-71091-9
1. Computer software—Development. 2. eXtreme programming. I. Fowler, Martin,
II. Title.

QA76.76.D47 B4335 2000
005.3—dc21

00-064306

Copyright © 2001 by Addison-Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

ISBN 0-201-71091-9

Text printed on recycled paper

1 2 3 4 5 6 7 8 9 10—MA—0403020100
First printing, October 2000

To our Cindies…

Foreword

In *On War*, Carl von Clausewitz tells us that military history is a pendulum swinging back and forth between the relative advantages of armor and of mobility. The knights in shining armor were able to dominate any knight without, but they were no match for the quick, nearly naked pony warriors that swept across the plains with Genghis Kahn and his Mongols. Light cavalry itself was doomed as soon as there were tanks, and tanks were no match for fleet-footed Palestinian teenagers with Sagger missiles. With the Maginot Line, the French were gambling that the pendulum had swung again toward armor, but it hadn't, and the Germans simply went around it.

In the field of IT, we are just emerging from a time in which armor (process) has been king. And now we are moving into a time when only mobility matters. Building a product the right way still *sounds* like a laudable goal, but—let's face it—what really matters today is building it *fast*. Because we are process-obsessed in our field, we have tended to react to this new imperative as we reacted to the imperatives thrust upon us in the 1980s and 1990s. We have asked, "What shall we add to our process to deal with this new situation?" No answer to that question is going to be right because the question itself is wrong.

What the new mobility imperative requires is that we *subtract* from process: We need to get light.

"Getting light" means more than just abandoning heavy process and its attendant mountain of documentation. It means investing in people so they can work quickly and effectively without formal process and

此为试读，需要完整PDF请访问：www.ertongbook.com

without generating a ton of paper. No one has a better vision of how this is done than Kent Beck and Martin Fowler.

The XP movement they have founded is a way to make IT projects light and quick. The principles of XP are not just another methodology, another process. They are the antithesis of process. They are means to make process irrelevant.

Because XP projects are completely different, it stands to reason that managing them is different too. *Planning Extreme Programming* focuses on how a team of XP-empowered developers is optimally led. Beck and Fowler's prescriptions are often wry, sometimes wise, and almost always bang on target.

XP is the most important movement in our field today. I predict that it will be as essential to the present generation as the SEI and its Capability Maturity Model were to the last.

Tom DeMarco
Camden, Maine

Preface

This is a book about planning software projects. We are writing it mostly for project managers—those who have to plan and track the correspondence of the planning with reality. We also are writing it for programmers and customers, who have a vital role to play in planning and developing software.

Planning is not about predicting the future. When you make a plan for developing a piece of software, development is not going to go like that. Not ever. Your customers wouldn't even be happy if it did, because by the time the software gets there, the customers don't want what was planned; they want something different.

Like so many, we enjoy Eisenhower's quotation: "In preparing for battle I have always found that plans are useless, but planning is indispensable."[1] That's why this isn't a book about plans; it's about planning. And planning is so valuable and important, so vital, that it deserves to go on a little every day, as long as development lasts.

If you follow the advice in this book, you are going to have a new problem to solve every day—planning—but we won't apologize for that, because without planning, software development inevitably goes off the rails.

The scope of this book is deliberately narrow. It covers how to plan and track software development for XP projects. It's based on our experience

1. Richard Nixon, *Six Crises* (New York: Touchstone Press, 1990).

as consultants and coaches, together with the experience of the growing band of early adopters who are using XP.

As a result this isn't a book about the whole of project management. We don't cover typical project manager jobs such as personnel evaluation, recruiting, and budgeting. We don't address the issues of large projects with hordes of developers, nor do we say anything about planning in the context of other software processes, or of planning other activities. We think there are principles and techniques here that everyone can use, but we have stuck to the parts of the process we know—getting everybody on the team pointed in one direction, discovering when this is no longer true, and restoring harmony.

XP (Extreme Programming) is a system of practices (you can use the *m*-word if you want to; we'd rather not, thank you) that a community of software developers is evolving to address the problems of quickly delivering quality software, and then evolving it to meet changing business needs.

XP isn't just about planning. It covers all aspects of small team software development—design, testing, implementation, deployment, and maintenance. However, planning is a key piece of the XP puzzle. (For an overview of XP, read *Extreme Programming Explained: Embrace Change*. While you're at it, buy copies of all of the rest of our books, too.)

XP addresses long projects by breaking them into a sequence of self-contained, one- to three-week mini-projects. During each iteration

- ✧ Customers pick the features to be added.
- ✧ Programmers add the features so they are completely ready to be deployed.
- ✧ Programmers and customers write and maintain automated tests to demonstrate the presence of these features.
- ✧ Programmers evolve the design of the system to gracefully support all the features in the system.

Without careful planning, the process falls apart.

- ✧ The team must choose the best possible features to implement.
- ✧ The team must react as positively as possible to the inevitable setbacks.
- ✧ Team members must not overcommit, or they will slow down.

- The team must not undercommit, or customers won't get value for their money.
- Team members must figure out clearly where they are and report this accurately, so that everyone can adjust their plans accordingly

The job of the daily planner is to help keep the team on track in all these areas.

We come by our project planning ideas by necessity. As consultants, we are usually introduced to projects when they are mostly dead. The projects typically aren't doing any planning, or they are drowning in too much planning of the wrong sort.

The resulting ideas are the simplest planning ideas we could think of that could possibly work. But above all, remember all the planning techniques in the world, including these, can't save you if you forget that software is built by human beings. In the end keep the human beings focused, happy, and motiviated and they will deliver.

Kent Beck, Merlin, Oregon
Martin Fowler, Melrose, Massachusetts http://www.martinfowler.com
July 2000

I have a cunning plan.
—*Baldrick,* Blackadder

Acknowledgments

Thanks to our reviewers: Mark Windholtz, Ralph Johnson, Uncle Bob Martin, John Brewer, Phil Goodwin, Jean-Marc Heneman, Erik Meade, Alan Francis, Josu Oyanguren, Jim Stearns, Joel Jones, Bill Caputo, Randy Coulman, Andrew Nielsen, Brian Button, Don Wells, Gary Clayburg, James Goebel, Paul Sinnett, Bill deHora, Andreas Stankewitz, Frank Westphal, Georg Tuparev, Stuart Donovan, Joi Ellis, Alistair Cockburn, Matt Simons, Rob Mee, and Joshua Kerievsky.

Kent would like to thank Cindee, Bethany, Lincoln, Lindsey, Forrest, and Joëlle for their gift of time. Also, watching my brilliant coauthor Martin learn to skip rocks was the purest joy I've had in a good long time.

Martin would like to thank the folks at ThoughtWorks for trying and pushing beyond many of these ideas and for giving him the time to write this. But especially to thank Cindy for more reasons than would fit in this book.

Together, we'd like to thank our editor, Mike Hendrickson, and the staff at AWL: Heather Peterson, Heather Olszyk, Mike Guzikowski, and Tyrrell Albaugh.

Thanks to Bob Coe and Ron Jeffries for trusting us, then going beyond.

Our deepest thanks go to Robert Cecil "Uncle Bob" Martin. You'll find many of his words and thoughts in these pages.

Contents

Foreword xi

Preface xiii

Acknowledgments xvii

Chapter 1 *Why Plan?* 1

We plan to ensure that we are always doing the most important thing left to do, to coordinate effectively with other people, and to respond quickly to unexpected events.

Chapter 2 *Fear* 7

Software development is risky. People involved have many fears of what may go wrong. To develop effectively we must acknowledge these fears.

Chapter 3 *Driving Software* 11

We use driving as a metaphor for developing software. Driving is not about pointing the car in one direction and holding to it; driving is about making lots of little course corrections.

Chapter 4 *Balancing Power* 15

Our planning process relies on clearly separating the roles of business people and software people. This ensures that business people make all the business decisions and software people make all the technical decisions.

此为试读，需要完整PDF请访问：www.ertongbook.com

Chapter 5 *Overviews* 21

The XP process has releases that take a few months, that divide into two-week iterations, that divide into tasks that take a few days. Planning allocates stories (chunks of function) to releases and iterations in reaction to the realities of development.

Chapter 6 *Too Much to Do* 25

When you are overloaded, don't think of it as not having enough time; think of it as having too much to do. You can't give yourself more time, but you can give yourself less to do, at least for the moment.

Chapter 7 *Four Variables* 27

We use four variables to help us think about how to control a project: cost, quality, time, and scope. They are interrelated but affect each other in strange ways.

Chapter 8 *Yesterday's Weather* 33

As the basis for your planning, assume you'll do as much this week as you did last week.

Chapter 9 *Scoping a Project* 35

To get a quick feel for how big the project is, run the planning process at coarse resolution.

Chapter 10 *Release Planning*. 39

In release planning the customer chooses a few months' worth of stories, typically focusing on a public release.

Chapter 11 *Writing Stories*. 45

The story is the unit of functionality in an XP project. We demonstate progress by delivering tested, integrated code that implements a story. A story should be understandable to customers and developers, testable, valuable to the customer, and small enough so that programmers can build half a dozen in an iteration.

Chapter 12 *Estimation* 57

Base your story estimates on a similar story you've already done. This story will take about the same amount of time as a comparable story.

Chapter 13 *Ordering the Stories* 63

The most important stories to do first are the ones that contain the highest business value. Beware of sequencing stories based on technical dependencies. Most of the time the dependencies are less important than the value.

Chapter 14 *Release Planning Events* 71

Various events cause the team to do a little release planning. The customer adds and changes the priorities of stories, developers estimate stories, and the team notices if it has too much or too little to do.

Chapter 15 *The First Plan* 75

The first plan is the hardest and least accurate part of release planning. Fortunately you only have to do it once.

Chapter 16 *Release Planning Variations* 79

Some local adaptations of the release planning are shorter releases, longer releases, and shorter stories.

Chapter 17 *Iteration Planning* 83

Each iteration is planned by breaking down the stories for that iteration into tasks. Tasks are scheduled by asking programmers to sign up for the tasks they want, then asking them to estimate their tasks, then rebalancing as necessary.

Chapter 18 *Iteration Planning Meeting* 87

At the beginning of an iteration the team creates an iteration plan. This plan breaks down the iteration into development tasks of a few days, each with a programmer responsible for its completion.

Chapter 19 *Tracking an Iteration* 95

A couple of times a week, the tracker checks progress on the iteration to see how things are going.

Chapter 20 *Stand-up Meetings* 105

Have a short meeting once a day so everybody knows what's going on, and what's not.

Chapter 21 *Visible Graphs* 107

Anyone should be able to sense the state of the project by looking at a handful of graphs in the team's working area.

Chapter 22 *Dealing with Bugs* 113

Schedule bug fixes with stories so the customer can choose between fixing bugs and adding further functionality.

Chapter 23 *Changes to the Team* 117

When the team changes, how does that affect your planning?

Chapter 24 *Tools* 119

Stick with simple tools, like pencil, paper, and whiteboard. Communication is more important to success than whizbang.

Chapter 25 *Business Contracts* 121

Traditional business relationships require a little tweaking if you're going to plan and execute a project with XP.

Chapter 26 *Red Flags* 125

Here are some dicey situations we've seen more than once and what we wished we'd done about them.

Chapter 27 *Your Own Process*. 129

No two XP projects will ever act exactly alike. Once you get comfortable with the basic process, you will grow it to fit your situation more precisely.

Index *131*

Chapter 1

Why Plan?

The best-laid schemes o' mice an' men
Gang aft a gley.
—Robert Burns, "To a Mouse"

We plan to ensure that we are always doing the most important thing left to do, to coordinate effectively with other people, and to respond quickly to unexpected events.

When Kent was about ten, he went fly-fishing for the first time in the Idaho panhandle. After a fruitless, sweaty day in pursuit of brook trout, he and his friends headed for home. After half an hour of stumbling through dense trees, they realized they were lost. Kent started to panic—fast breathing, tunnel vision, chills. Then someone suggested a plan—they would walk uphill until they hit a logging road they knew was up there. Instantly, the panic disappeared.

Kent was struck at the time by the importance of having a plan. Without the plan, he was going to do something stupid, or just go catatonic. With the plan he was calm again.

Plans in software development work the same way. If you know you have a tight deadline, but you make a plan and the plan says you can make the deadline, then you'll start on your first task with a sense of urgency but still working as well as possible. After all, you have enough time. This is exactly the behavior that is most likely to cause the plan to come true. Panic leads to fatigue, defects, and communication breakdowns.

But we've also seen plans lead to trouble. They can be a huge time sink, dragging days out of people who'd rather be doing something productive. Plans can be used as a stick to beat people with, and worst of all, they can conceal trouble until it's too late to deal with it.

Why We Should Plan

We don't plan so we can predict the future. Business and software are changing too rapidly for prediction to be possible. Even if it were possible to predict what we needed in three years, it wouldn't necessarily help because between now and then we need so many different things.

The more obvious it is that you should do something, the more important it is to ask why. You must do some planning when tackling a serious software development project. Therefore, before you start planning a project, you have to understand why you need to carry out the project. Without understanding why you need the project, how will you be able to tell if you have succeeded?

We plan because

- We need to ensure that we are always working on the most important thing we need to do.
- We need to coordinate with other people.

- When unexpected events occur we need to understand the consequences for the first two.

The first is the obvious reason for planning. There's nothing more frustrating than working hard on a part of the system only to find that it doesn't really matter and gets scrapped in the next release of the system. Time spent doing one thing is time not spent doing something else, and if that something else is important then we may fail.

Say it's two o'clock and we're in Boston. We want to drive up to Acadia, but we'd also like to get haircuts and hit Freeport for camping gear. Last time we drove up to Acadia it took us five hours with no stops. So we see some options. If we shoot straight up to Acadia we can be there by seven o'clock. If we want to stop for dinner on the way, say an hour, we will be there by eight. To get haircuts we'd need another hour, so that would be nine. Visiting Freeport would add another hour. We look at what's most important to us. If we want to be fed, equipped, not too late, and we could care less about our appearance, we might decide to drop the haircut. A plan helps us see our options.

Coordination is why everyone else wants us to plan. We get a call from our wives to meet for dinner in Bar Harbor. Since it's two o'clock we know we can meet them if we drive right up, stop in Freeport, and be there around eight. Software is full of such coordination: marketing announcements, financial periods, or management promises. Planning allows us to get an idea of what is reasonable.

But planning is only as good as the estimates that the plans are based on, and estimates always come second to actuals. If we hit a horrible traffic jam, all the planning in the world can't help us make that dinner date. The real world has this horrible habit of destroying plans, as Mr. Burns noted in this chapter's opening quote.

Planning still helps when the real world intrudes because it allows us to consider the effects of the unexpected event. Leaving at two o'clock, we hit bad traffic and don't get to Portland until five. We know we usually get there after an hour and a half, so our experience (and plan) tells us to call our friends to put dinner back to half past eight and drop the visit to Freeport. Planning allows us both to adjust what we do and to coordinate with others. The key is to adjust the plan as soon as we know the effect of the event. Our wives would much rather know about our delay at five than at eight, and it would be really annoying to spend time in Freeport and only later realize that we've really screwed

此为试读，需要完整PDF请访问：www.ertongbook.com