

From: Ron Jeffries et al.
Extreme Programming Installed.
Addison-Wesley, 2001.

Contents

<i>Foreword</i>	xiii
<i>Preface</i>	xv
Chapter 1 <i>Extreme Programming</i>	1
<i>Extreme Programming is a discipline of software development with values of simplicity, communication, feedback, and courage. We focus on the roles of customer, manager, and programmer and accord key rights and responsibilities to the people in those roles.</i>	
Chapter 2 <i>The Circle of Life</i>	13
<i>An XP project succeeds when the customers select business value to be implemented, based on the team's measured ability to deliver functionality over time.</i>	
Chapter 3 <i>On-Site Customer</i>	17
<i>An XP project needs a full-time customer to provide guidance. Here's a summary of why.</i>	
Chapter 4 <i>User Stories</i>	23
<i>Define requirements with stories, written on cards.</i>	

Chapter 5 *Acceptance Tests* 31

Surely you aren't going to assume you're getting what you need. Prove that it works! Acceptance tests allow the customer to know when the system works and tell the programmers what needs to be done.

Sidebar *Acceptance Test Samples* 35

At first it can be difficult figuring out how to do acceptance tests. With a little practice, it becomes easy.

Chapter 6 *Story Estimation* 37

Customers need to know how much stories will cost in order to choose which ones to do and which to defer. Programmers evaluate stories to provide that information. Here's how.

Interlude *Sense of Completion* 45

XP's nested planning and programming cycles keep the project on track and provide a healthy sense of accomplishment at frequent intervals.

Chapter 7 *Small Releases* 49

The outermost XP cycle is the release. Small and frequent releases provide early benefit to the customer while providing early feedback to the programmers. Here are some thoughts on how to make it happen.

Chapter 8 *Customer Defines Release* 55

In each release cycle, the customer controls scope, deciding what to do and what to defer, to provide the best possible release by the due date. Work fits into the calendar based on business value, difficulty, and the team's implementation velocity.

Chapter 9 *Iteration Planning* 61

Inside each release, an Extreme team plans just a few weeks at a time with clear objectives and solid estimates.

Chapter 10 *Quick Design Session* 69

Within each iteration, programmers don't stand alone. Here's a technique to help programmers move forward with courage. Make it part of your team's ritual.

Chapter 11 *Programming* 71

It's called Extreme Programming, after all. Here's how we do the programming part of things.

Sidebar *Code Quality* 83

A little more detail on something close to our hearts: simplicity.

Chapter 12 *Pair Programming* 87

On an Extreme Programming team, two programmers sitting together at the same machine write all production code.

Chapter 13 *Unit Tests* 93

Extreme Programmers test everything that could possibly break, using automated tests that must run perfectly all the time.

Sidebar *xUnit* 105

Use the world's lightest testing tool.

Chapter 14 *Test First, by Intention* 107

Code what you want, not how to do it. Chet and Ron do a small task test first trying always to express intention in the code rather than algorithm.

Chapter 15 *Releasing Changes* 121

Using collective code ownership and comprehensive unit tests, an XP team releases changes rapidly and reliably.

Chapter 16 *Do or Do Not* 127

We've now covered most of the programming aspects of XP. Here's a summary of things we do—and things we don't do.

Chapter 17 *Experience Improves Estimates* 131

With each iteration, we gain experience. Experience with stories helps us estimate future stories more easily and more accurately.

Chapter 18 *Resources, Scope, Quality, Time* 135

Who's doing what? How much is finished? How good is it? When will we be done? What metrics should we keep?

Chapter 19 *Steering* 147

The estimates are wrong. Your priorities will change. You must steer.

Chapter 20 *Steering the Iteration* 151

To steer each iteration, you need to track how many stories are getting done and how well the task estimates are holding up.

Chapter 21 *Steering the Release* 157

To steer the release, you need to track what's done, how fast you are going, and how well the system works.

Chapter 22 *Handling Defects* 161

Report 'em, schedule 'em, test and fix 'em, avoid 'em. Just don't call 'em bugs.

Sidebar *Advanced Issue: Defect Databases* 165

Sidebar *Advanced Practice: Tests as Database* 169

Chapter 23	<i>Conclusion</i>	171
BONUS TRACKS		175
<i>Here are some things we've paid a lot to learn. Since you bought the album, we wanted to give you a little something extra. Thank you, and we hope we passed the audition.</i>		
Chapter 24	<i>We'll Try</i>	177
<i>"We'll try" can be the saddest words a programmer has ever spoken, and most of us have spoken them more than once. We've covered this material in other forms already, but it bears repeating here.</i>		
Chapter 25	<i>How to Estimate Anything</i>	185
<i>Sometimes estimating stories seems scary. Keep your heads, stick together, and break the story down into small parts. You'll be surprised what you can do.</i>		
Chapter 26	<i>Infrastructure</i>	189
<i>What about that database you need to build first? What about that framework? What about that syntax-directed command compiler? Get over it!</i>		
Chapter 27	<i>It's Chet's Fault</i>	193
<i>Are you looking for someone to blame? This chapter explains how to know whose fault it is. Now move on and solve your problems.</i>		
Chapter 28	<i>Balancing Hopes and Fears</i>	195
<i>Those of you who have heard Ron, Ann, or me speak about XP are probably wondering where are all the war stories. Well, here's one.</i>		
Chapter 29	<i>Testing Improves Code</i>	199
<i>An example showing how writing some tests can help you to improve the code.</i>		

Chapter 30 *XPer Tries Java* 203

After the C3 project ended, most of the team was transferred to work on the human resources intranet. I found how they were using the principles of XP to improve their lives on a new project heartening. What follows is a description of how Rich Garzaniti, exC3er and devoted XPer, is introducing testing and modern development tools into an environment where none existed.

Chapter 31 *A Java Perspective* 211

We would like to thank Bill Wake for allowing us to use this article. It is the second in a series entitled “The Test/Code Cycle in XP.” His website, <http://users.vnet.net/wwake>, contains the entire series plus a whole lot more.

Chapter 32 *A True Story* 225

Ron Jeffries [re]learns something about simplicity.

Chapter 33 *Estimates and Promises* 229

We estimate how long the project will take. We promise to tell the truth about how we’re doing.

Chapter 34 *Everything That Could Possibly Break* 233

Test everything that could possibly break. What does this mean? How is it possible?

Afterword 243

Annotated Bibliography 245

Index 261

The Customer Role

The customer chooses what will deliver business value, chooses what to do first and what to defer, and defines the tests to show that the system does what it needs to.

Every software project needs to deliver business value. To be successful, the team needs to build the right things, in the right order, and to be sure that what they build actually works. Of course, this can't be done without programmers,¹ but in fact the customer's role is critical in steering that process to success.

The customer role on an XP project can be filled by one person, or by several. The team will be most effective if the customer stays on-site and present with the team, full-time. We'll discuss some details in *On-Site Customer* (page 17). Here, we'll talk in more general terms about what the customer does. If you're the XP customer, we're talking to you.

Note that we say "the customer" and not "the customers." Whether they are one person or many people, the XP customer always speaks with one voice. The determination of what will have business value, and the order of building that value, rests solely with the customer. (Don't worry, you get lots of help and advice. But ultimately, you get to make the call.)

An XP team plans and builds software in terms of "stories." Stories are just that—individual stories about how the system needs to work. Each story describes one thing that the system needs to do. Each story must be understood well enough that the programmers can estimate its difficulty. And each story must be testable.

As the customer, you express what must be done in terms of stories. For a project spanning a few months, there may be 50 or 100 stories. Larger projects of course have more stories. We'll talk more about the details in *User Stories* (page 23).

You probably have a delivery date in mind, though some projects have a fixed feature list rather than a fixed date. We are not content to imagine that everything that you can think of will be done by a given date. Neither should you be. Instead, the XP process lets the team predict, more and more accurately, how much work can be done in any given time period. Using this information, you manage project scope—choosing what to do now and what to defer until later—to ensure successful delivery.

1. In this book the pronouns "he" and "she" are used randomly to reflect the broad diversity that makes our industry great.

You, the customer, have the critical responsibility to choose the stories that will provide the most valuable features, the highest business value, and that can be accomplished by the desired delivery date. The XP development process lets you choose among the stories with great flexibility. There's not much limitation on what can be done first and what second. This is by design; if you are to choose the stories for successful on-time release, you must have the flexibility to make the choice as independently as possible. Read more about this process in *Customer Defines Release* (page 55) and *Iteration Planning* (page 61).

Finally, you specify tests that show whether the stories have been correctly implemented. These *Acceptance Tests* (page 31), whether built by the programmers, by an independent tester, or by you—the customers—yourselves, provide confidence that the system really does what it needs to do.

Define business value, decide what to do and what to defer, and define the tests to show that the system works. These are your key responsibilities as the XP customer.

The Programmer Role

The programmers analyze, design, test, program, and integrate the system. The programmers estimate the difficulty of all stories, and track the pace at which they can deliver stories to the customer.

If the project is to deliver business value, each story must be understood. Software must be designed, tested, and built to implement that story, and all the software must be brought together into a coherent whole. That is the XP programmer's job. If you're the XP programmer, we're talking to you.

In Extreme Programming, the emphasis is on programming. Everything we do looks like programming and is focused on the most critical artifact of software development, the program.

Build the system in small releases, so that the customer benefit is maximized and you get the best possible feedback on how you're doing. We talk about this in *Small Releases* (page 49), *Customer Defines Release* (page 55), and *Iteration Planning* (page 61).

Base the program on simple, clear design. This lets you produce quality software quickly. There's more discussion of this in *Code Quality* (page 83), and *A True Story* (page 225). As you learn more about

what the design wants to be, improve the design using *Refactoring* (page 76).

XP is neither slash and burn, nor code and fix programming. Not at all. Extreme Programming is about careful and continuous design, rapid feedback from extensive testing, and the maintenance of relentlessly clear and high-quality code.

Keep the system integrated at all times, so there's always a good version to look at. Keeping integrated lets you go rapidly without stepping on each others' toes. See *Continuous Integration* (page 78).

Share the ownership of all the code, so no one has to wait and everyone feels able to make everything better. See *Collective Code Ownership* (page 75), and *Releasing Changes* (page 121). Share a single *Coding Standard* (page 79) as well, whether self-evolved or adopted from elsewhere. Make everyone's code look alike—it helps with communication and team focus. Express individuality in the way you wear your XP ball cap, not in your code.

Make sure that the system always works, using comprehensive unit tests that you write, as well as the customer's acceptance tests. These tests allow rapid change and support collective code ownership by keeping change from introducing mistakes. See *Unit Tests* (page 93), *Acceptance Tests* (page 31), *Everything That Could Possibly Break* (page 233), and *Test First, by Intention* (page 107).

Write all production code in pairs, for maximum speed and cross-training, in support of shared code ownership and rapid progress, as described in *Pair Programming* (page 87).

Extreme Programming is an approach to software development that lets programmers do what they do best—program—while giving the customers what they need most—business value. It's a win-win approach and fun, too.

The Manager Role

The manager brings the customer and developers together and helps them meld into a smoothly operating team. You don't do the process—you make the process smoother.

If you're the XP manager, we're talking to you. The XP process specifies how the team does certain things that conventional managers sometimes do. But don't worry—there's plenty for the XP project

manager to do. On an XP project, the manager's role is key, and it is very much focused on management per se.

The first and last job of a good manager is to get things out of the way of the people who are doing the work. Look for things that are slowing the team, and use your special managerial powers to resolve them. Expedite purchases, make sure the workspace is arranged effectively, keep the computers up-to-date, lean on the LAN guys to fix problems, and so on. A manager's success depends on removing everything from the team's path that doesn't contribute to the objective of delivering good software on time.

When it comes to the day-to-day process of planning, designing, testing, coding, releasing, managers don't do any of these things directly. However, you do something more important: You cause these things to be done, coordinate their doing, and report the results.

It may seem that the entire team just magically appears at the planning table when it's time for the next release plan. It's not magic; it's your doing.

As manager, you cause that meeting to happen, and you coordinate it into existence. At a stand-up meeting a bit before release planning time, mention the need for the meeting and suggest a date. If there's general agreement, go ahead. If there are scheduling conflicts, go around to the team members and find a suitable date and time. If necessary, encourage someone to change a conflicting appointment.

When the date is chosen, prepare the ground. Arrange a room, send out the invitations, order the refreshments, or cause these things to be done if you have administrative help.

Before any planning meeting, check with the customers, reminding them to be ready and to bring any new stories, and so on. If they need help, provide it.

If necessary, coordinate or facilitate each meeting—or designate someone to do so. Help to keep the team on process, make notes on the proceedings, offer to get special resource people if they're needed, and so on.

After each meeting, if reporting needs to be done, do it or cause it to be done. (Internal reporting generally is not needed. The plan is on the white board and in the minds of the team. But keep some stakeholders outside the room up-to-date.)

During the iteration, it's the same: cause the right things to happen, coordinate the activities, report results, and always remove obstacles.

The project manager usually has responsibility for personnel and this is a very important one. Even on the best teams, there are differences between individuals, and sometimes there can be temporary or permanent people problems.

When people have a conflict, you need to fix it. If someone's behavior is harming the team, you have to address the problem. If the individual cannot or will not correct the behavior, you must remove him or her from the team. This should not be done lightly or precipitously, but sometimes it must be done, and it is the project manager's responsibility.

There can sometimes be political problems that impact the team. These are major obstacles and the manager leaps in to resolve them. A stakeholder may have difficulty allowing the customer to schedule the stories or may put pressure on the programmers to adjust their estimates. Watch for outside forces that can impact your team and, when needed, firmly and productively step in.

On the good side, the project manager gets to give rewards. There is the annual rating and salary adjustment ritual. We can't tell you how to do this—extreme teams are all over the map on compensation policy. It's the manager's responsibility to have fair and consistent assessments of the staff and to have compensation fairly reflect those assessments.

And think about small rewards as well. Recognition is important. New toys or tokens for the team. A round of Laser Tag, a round of beers, a night at the opera. A little time off from work and off the books. And don't forget the families.

This only scratches the surface. The project manager's role is very important to the project. If done creatively and effectively, it can greatly ensure the team's success.

Cause, coordinate, report, and reward. And always: remove obstacles.

Rights and Responsibilities

Extreme Programming tries to provide certain benefits to the managers, customers, and developers involved in a project. We express these as rights because they are very important to the success of the project and to the team members.

Manager and Customer Rights

1. You have the right to an overall plan, to know what can be accomplished, when, and at what cost.
2. You have the right to get the most value out of every programming week.
3. You have the right to see progress in a running system, proven to work by passing repeatable tests that you specify.
4. You have the right to change your mind, to substitute functionality, and to change priorities without paying exorbitant costs.
5. You have the right to be informed of schedule changes, in time to choose how to reduce scope to restore the original date. You can cancel at any time and be left with a useful working system reflecting investment to date.

Programmer Rights

1. You have the right to know what is needed, with clear declarations of priority.
2. You have the right to produce quality work at all times.
3. You have the right to ask for and receive help from peers, superiors, and customers.
4. You have the right to make and update your own estimates.
5. You have the right to accept your responsibilities instead of having them assigned to you.