Part I: The Problem

This section sets the stage for Extreme Programming by discussing various aspects of the problem to be solved in inventing a new discipline of software development. The section discusses the basic assumptions we will use as we choose practices covering the various aspects of software development—the driving metaphor, the four values, the principles derived from those values, and the activities to be structured by our new development discipline.

Chapter 1. Risk: The Basic Problem

Software development fails to deliver, and fails to deliver value. This failure has huge economic and human impact. We need to find a new way to develop software.

The basic problem of software development is risk. Here are some examples of risk:

- Schedule slips—the day for delivery comes, and you have to tell the customer that the software won't be ready for another six months.
- Project canceled—after numerous slips, the project is canceled without ever going into production.
- System goes sour—the software is successfully put into production, but after a couple of years the cost of making changes or the defect rate rises so much that the system must be replaced.
- Defect rate—the software is put into production, but the defect rate is so high that it isn't used.
- Business misunderstood—the software is put into production, but it doesn't solve the business problem that was originally posed.
- Business changes—the software is put into production, but the business problem it was designed to solve was replaced six months ago by another, more pressing, business problem.
- False feature rich—the software has a host of potentially interesting features, all of which were fun to program, but none of which makes the customer much money.
- Staff turnover—after two years, all the good programmers on the project begin to hate the program and leave.

In these pages you will read about Extreme Programming (XP), a software development discipline that addresses risk at all levels of the development process. XP is also very productive, produces high-quality software, and is a lot of fun to execute.

How does XP address the risks listed above?

- Schedule slips—XP calls for short release cycles, a few months at most, so the scope of any slip is limited. Within a release, XP uses one- to four-week iterations of customer-requested features for fine-grained feedback about progress. Within an iteration, XP plans with one- to three-day tasks, so the team can solve problems even during an iteration. Finally, XP calls for implementing the highest priority features first, so any features that slip past the release will be of lower value.
- Project canceled—XP asks the customer to choose the smallest release that makes the most business sense, so there is less to go wrong before going into production and the value of the software is greatest.
- System goes sour—XP creates and maintains a comprehensive suite of tests, which are run and re-run after every change (several times a day), to ensure a quality baseline. XP always keeps the system in prime condition. Cruft is not allowed to accumulate.
- Defect rate—XP tests from the perspective of both programmers writing tests function-by-function and customers writing tests program-feature-by-program-feature.
- Business misunderstood—XP calls for the customer to be an integral part of the team. The specification of the project is continuously refined during development, so learning by the customer and the team can be reflected in the software.
- Business changes—XP shortens the release cycle, so there is less change during the development of a single release. During a release, the customer is welcome to substitute new functionality for functionality not yet completed. The team doesn't even notice if it is working on newly discovered functionality or features defined years ago.
- False feature rich—XP insists that only the highest priority tasks are addressed.
- Staff turnover—XP asks programmers to accept responsibility for estimating and completing their own work, gives them feedback about the actual time taken so their estimates can improve, and respects those estimates. The rules for who can make and change estimates are clear. Thus, there is less chance for a

programmer to get frustrated by being asked to do the obviously impossible. XP also encourages human contact among the team, reducing the loneliness that is often at the heart of job dissatisfaction. Finally, XP incorporates an explicit model of staff turnover. New team members are encouraged to gradually accept more and more responsibility, and are assisted along the way by each other and by existing programmers.

Our Mission

If we accept project risk as the problem to be solved, where are we going to look for the solution? What we need to do is invent a style of software development that addresses these risks. We need to communicate this discipline as clearly as possible to programmers, managers, and customers. We need to set out guidelines for adapting it to local conditions (that is, communicate what is fixed and what is variable).

That's what we cover in sections one and two of this book. We will go step by step through the aspects of the problem of creating a new style or discipline of development, and then we will solve the problem. From a set of basic assumptions, we will derive solutions that dictate how the various activities in software development—planning, testing, development, design, and deployment—should occur.