# Foreword to the Second Edition

Wow—the second edition. I cannot believe that five years have already passed since the appearance of the first edition. When Kent pinged me to write a foreword to the second edition I asked him for a manuscript version with change bars. What a silly request—the book is a full rewrite! In the second edition of XP Explained Kent revisits XP and applies the XP paradigm—stay aware, adapt, change—to XP itself. Kent has revisited, cleaned-up, and refactored every bit of XP Explained and integrated many new insights. The result is XP Explained even better explained!

This is an excellent opportunity to reflect on how XP has influenced my own software development. Shortly after the first edition of XP Explained I became involved in the Eclipse project and it is now absorbing all my software energy. Eclipse isn't run under the pure XP flag. We follow agile practices; however, the XP influences are easy to spot. The most obvious one is that we have encoded several XP practices directly into our tool. Refactoring, unit testing, and immediate feedback as you code are now an integral part of our toolset. Moreover, since we are "eating our own dog food" we use these practices in our day-to-day development. Even more interesting are the XP influences one can spot in our development process. Eclipse is an open source project and one of our goals is to practice completely transparent development. The rationale is simple; if you don't know where the project is going you cannot help out or provide feedback. XP practices help us to achieve this goal.

Here is how we apply some of these practices:

- *Testing early, often and automated*—To get a green check mark for our latest builds more than 21,000 unit tests have to pass.
- *Incremental design*—We invest in the design every day, but we have the additional constraint that we need to keep our APIs stable.
- *Daily deployment*—Components deploy their code at least once per day and develop on top of the deployed code to get immediate feedback and to catch problems early.
- *Customer involvement*—We are lucky to have an active user community that isn't shy and provides us with continuous feedback. We listen and do our best to be responsive.
- *Continuous integration*—The latest code is built every night. The nightly builds provide us with insights about cross-component integration problems. Once per week we do an integration build where we ensure integrity across all components.
- *Short development cycles*—Our cycles are longer than the XP-suggested one week cycles, but the goals are the same. Each of our six week cycles ends in a milestone build which have become the heartbeat of our project. The goal of each milestone build is to show progress (which keeps us honest) and to deliver it with a high enough level of quality that our community can really use it and provide feedback (which keeps us even more honest).
- *Incremental planning*—After a release we develop an embryonic overall plan which we evolve throughout the release cycle. This plan is posted on our website early so that our user community can join the dialog. The exception is the milestones, which are fixed in the first planning iteration since they define the heartbeat of our project.

Despite the fact that we have not adopted XP in its entirety, we are getting a lot out of the above XP practices. In particular, they help us to reduce our development stress! All these practices, underpinned by a strong team committed to shipping quality software on time, are our keys to hitting the projected milestones and ship dates with precision.
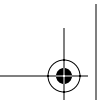
Kent is continuing to challenge my views on software development. While reading the book I've discovered several practices that I will add to my try-list. I suggest you do the same and accept the XP invitation to improve the way you develop software and to create outstanding software.

Erich Gamma
September 2004
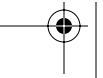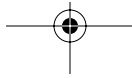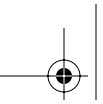
Foreword to the First Edition

Extreme Programming (XP) nominates coding as the key activity throughout a software project. This can't possibly work!

Time to reflect for a second about my own development work. I work in a just-in-time software culture with compressed release cycles spiced up with high technical risk. Having to make change your friend is a survival skill. Communication in and across often geographically separated teams is done with code. We read code to understand new or evolving subsystem APIs. The life cycle and behavior of complex objects is defined in test cases, again in code. Problem reports come with test cases demonstrating the problem, once more in code. Finally, we continuously improve existing code with refactoring. Obviously our development is code-centric, but we successfully deliver software in time, so this can work after all.

It would be wrong to conclude that all that is needed to deliver software is daredevil programming. Delivering software is hard, and delivering quality software in time is even harder. To make it work requires the disciplined use of additional best practices. This is where Kent starts in his thought-provoking book on XP.

Kent was among the leaders at Tektronix to recognize the potential of man in the loop pair programming in Smalltalk for complex engineering applications. Together with Ward Cunningham, he inspired much of the pattern movement that has had such an impact on my career. XP describes an approach to development that combines practices used by many successful developers that got buried under the massive literature
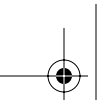
on software methods and process. Like patterns, XP builds on best prac-
tices such as unit testing, pair programming, and refactoring. In XP
these practices are combined so that they complement and often control
each other. The focus is on the interplay of the different practices, which
makes this book an important contribution. There is a single goal to
deliver software with the right functionality and hitting dates. While
OTI's successful Just In Time Software process is not pure XP, it has
many common threads.

I've enjoyed my interaction with Kent and practicing XP episodes on
a little thing called JUnit. His views and approaches always challenge
the way I approach software development. There is no doubt that XP
challenges some traditional big M approaches; this book will let you
decide whether you want to embrace XP or not.

Erich Gamma
August 1999