Foreword

Why do we need another Agile/XP book? If XP and the Agile Manifesto both value simplicity, then why do we need more books, articles, conference talks, user group discussions, Yahoo! Group e-mails, and coffee break debates? Why? Because simple isn't the same as simplistic. Why? Because an effective set of "simple" principles and practices give rise to complex, intelligent behavior.

The 12 practices of XP, the nine principles of DSDM, the 12 principles of Bob Charette's Lean Development, or the 12 principles articulated in the Agile Manifesto (12 seems to be a very popular number for Agilists) are far from simplistic. Complex problems, those that stretch the limits of technology and human capability, are best approached by internalizing a few rules, practices, and principles whose "application" generates an infinite number of creative ideas that in turn enable us to deliver value to our customers.

This is a key point that many rigorous methodology proponents haven't understood. Many of them believe in inclusive rules, procedures, and processes rather than generative rules. Have a problem? Just turn to process 57, activity 24, task 87, step 4 to find the answer. Unfortunately, or actually fortunately for us, complex problems don't yield to solutions by the numbers. Complex problems, the real problems we each face in the day-to-day, rough-and-tumble world of software product development, yield to creative, innovative thinking guided by a few key principles, grounded in a few key practices. "When the business

landscape was simple, companies could afford to have complex strategies," write Kathleen Eisenhardt and Donald Sull in a *Harvard Business Review* article ("Strategy as Simple Rules," January 2001). "But now that business is so complex, they need to simplify."

Simplify doesn't mean simplistic. It means that we need to extract, from the hundreds and hundreds of software development principles and practices, those half-dozen to a dozen that drive us to think clearly and effectively about the problems we face. If simple practices were simplistic, we wouldn't need entire books about single practices: refactoring (Martin Fowler), or pair programming (Laurie Williams and Robert Kessler), or test-first development (Kent Beck).

The diversity of chapters in this book proves my point. Written by both recognized leaders in the Agile/XP field and lesser-known leaders who labor daily to create value for their customers, these chapters reflect the complexity of real-world problems and how they are being solved by helping us understand the incalculable richness of a few key, simple ideas.

Books like this are important. Although not every chapter will be of intense interest to every reader, by skimming some, reading others, and intensely studying still other selections, readers will gain a deeper understanding of how their contemporaries are using Agile/XP practices to solve a multitude of real-world problems. But we must also admit, those of us who participated in the XP2001 conference that generated many of these chapters, that part of the allure of the conference was the Mediterranean beaches of Sardinia.

Jim Highsmith
Salt Lake City, Utah, May 2002