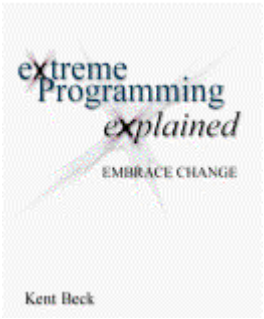
Book Review

eXtreme Programming Explained: Embrace Change



by Kent Beck

Review by Dennis Elenburg

I almost didn't write this review. Extreme Programming (XP) and the whole agile software development movement are somewhat controversial, especially around Rational where the RUP is the party line. I certainly didn't want to make a career-limiting move by advocating a software development methodology contrary to the one embraced by Rational! ☺ The perception (at least in some circles) seems to be that the RUP and XP are opposing forces in the ongoing debate over the best way to build software. This isn't completely true. After reading John Smith's excellent white paper in the RUP titled "A Comparison of RUP and XP," I realized the RUP and XP have a lot in common. Still, I wanted to know about XP apart from the context of the RUP, so I thought the best place to start would be reading a book by one of the key contributors to the XP philosophy, Kent Beck.

When I first got a copy of Kent Beck's *eXtreme Programming Explained: Embrace Change*, I was pleased to discover a thin book with large type and only 160 pages of reading material. I guess it would have been hypocritical if Beck wrote a heavy tome to explain something that is supposed to be "agile," but I did appreciate his brevity in laying out the XP philosophy. The book is divided into three sections: *The Problem*, *The Solution*, and *Implementing XP*. The 27 chapters are short and to the point; some are only a couple of pages. This book can be consumed in a couple of sittings or one US cross-country flight. (If you're going international, you'll need to pack more reading material.)

Addison-Wesley bills *eXtreme Programming Explained: Embrace Change* as "The XP Manifesto" in their seven-book series on Extreme Programming. Author Kent Beck of CRC card fame is the series advisor and coauthor of another book in the series (*Planning eXtreme Programming*) with Martin Fowler, a name many of you may recognize from *UML Distilled*. *eXtreme Programming Explained* lives up to its billing as a manifesto. The book includes a ten-page annotated bibliography with 60 entries under headings such as "philosophy" and "attitude." This is contrasted by a minimalist 2 ½ page glossary. Although Beck may not have been definitive, as demonstrated by the meager glossary, he does do a good job of laying out an enticing philosophy and style for building software. Anyone who has slaved away long hours on a project will definitely appreciate the XP practice of a 40-hour workweek!

The first section of *eXtreme Programming Explained* addresses the important topic of risk and the economics of software development, two regularly recurring themes around Rational. The premise of XP, in fact the author calls it *the* technical premise, is that software development teams would behave differently if the cost of change rose slowly over time and leveled off (see Figure 1), rather than rising exponentially (see Figure 2).

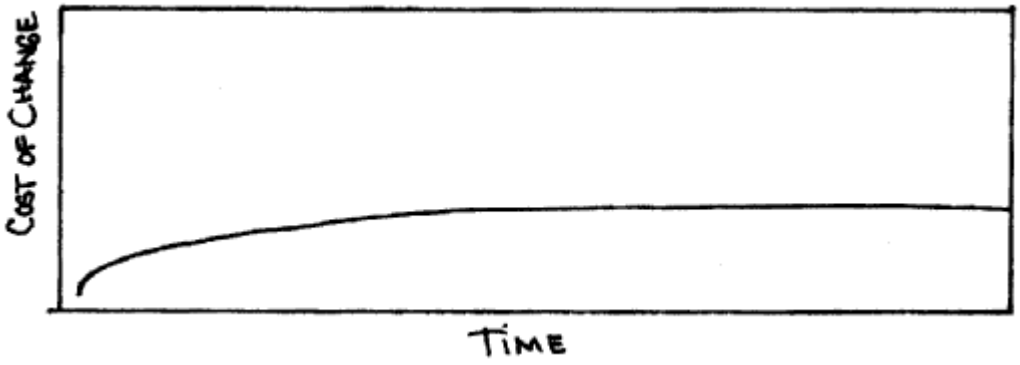


Figure 1. The Cost of Change May Not Rise Dramatically Over Time
(from Beck, Figure 3 Chapter 5)

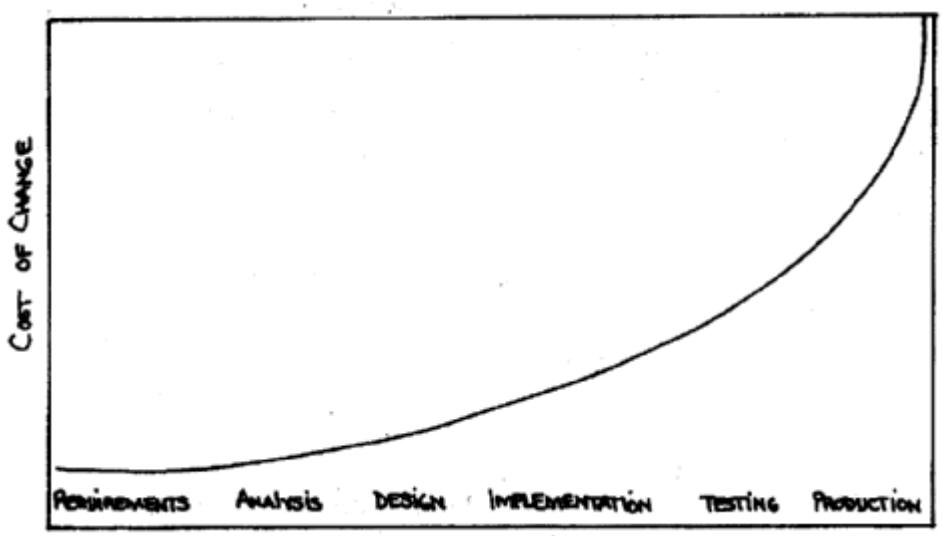


Figure 2. The Cost of Change Rising Exponentially Over Time
(from Beck, Figure 1 Chapter 5)

XP was conceived by considering which behaviors would work best if this inverted cost curve assumption were true while pressing good software development practices to the extreme. Beck admits "keeping the cost of change low doesn't just happen magically," and that aligns well with what we tell our Rational customers. But Beck goes one step further. He almost seems to be saying that if software developers behaved as they would if the cost-of-change curve were flat, then it will become flat. The inverted cost curve assumption is foundational to XP, but it leads to several seemingly counterintuitive practices that ultimately make XP both intriguing and controversial.

XP is "extreme" because it takes good software development practices and pushes them to their extreme in hopes that they will be extremely good when extremely applied synergistically. Beck says his ideas for XP came from considering what would happen if

software development practices were like control knobs, and you turned them up all at once. One of the results is behavior that may at first appear counterintuitive. An example is in the area of architecture. At first the XP practice of delaying design until the last possible minute seemed counterintuitive to me, especially after being indoctrinated in the architecture-first approach of the RUP. I later began to realize that the RUP also advocates delaying design, but only until the appropriate iteration. The disconnect is in XP's extreme view of iterations. XP calls for extremely short iterations that can appear to delay architecture indefinitely, while the RUP insists on having architecture in place before construction. The bottom line is that XP and the RUP *both agree that premature design is undesirable* even if they don't agree on the best way to avoid it.

Even though Beck calls architecture the "A" word, he does admit that architecture is important. Architecture just isn't treated as formally in XP as it is in the RUP. Beck argues that part of the architecture is captured in what XP calls the system metaphor, a single overarching idea that is supposed to provide cohesion and direction for the development team. And while there is no "architect" role in the cast of characters on an XP project, the XP role of "coach" would most likely play a similar role to a RUP architect. I was slightly relieved when in Chapter 17 Beck states that the first iteration of an XP project should address architecture by developing a functional skeleton of the system. But just as I was relaxing, Beck goes on to state that XP leaves room for adjustment to the architecture throughout the project by refactoring the architecture as necessary. My discomfort returned.

In XP, designing for reuse is eschewed in favor of designing only for current needs, not for future ones. Beck uses the metaphor of steering a car. You don't simply point a car in the direction you want to go and then step on the gas. Steering is a method of making a lot of small corrections along the way. XP supposedly allows the development team to capitalize on the fact that in software development the destination of your journey is often fuzzy. Beck positions this as a positive. He argues that the users of the system are in a learning process along with the development team. Instead of blaming users for not knowing what they want or blaming developers for building the wrong system, XP uses the metaphor of steering a car to represent the need for continual course corrections in a development effort. The steering in XP is a continual incremental process with very short development iterations and ongoing refactoring of the design. This short feedback loop is why XP insists on having a customer on the development team, a practice that is difficult to accomplish in many cases and impossible in others. If you cannot get a real customer on your team, or a good facsimile of one, then XP is probably not a suitable development process for your project.

Two XP practices I'd heard criticized before reading this book were *refactoring* and *pair programming*. Refactoring code is simply rewriting it to enhance some nonfunctional quality. A piece of code that is refactored will have unchanged behavior, but it might be simpler, faster, more understandable, or perhaps more flexible. Beck continually reminds the reader of *XP's core value of simplicity*. *The mantra of XP seems to be to develop only as much code as is absolutely necessary at any point in time, while keeping it as simple as possible by using a test-first design approach*. In an industry that often worships complexity, Beck admits this attitude can be difficult to foster and maintain, and pair programming is one way to address it.

Pair programming is simply two people working together at one workstation. This allows the person controlling the keyboard and mouse to work tactically while giving the person without

the keyboard an opportunity to think more strategically on the problem at hand. Beck suggests that a whole book could be devoted to the topic of pair programming, and he only scratches the surface of this topic. Since the XP philosophy is often counterintuitive to typical developer behavior, one of the many benefits of pair programming is helping ensure that developers remain "test infected" with XP's test-first design principle. For those who are not familiar with test-first design, I recommend reviewing Beck's "Aim, Fire" article on the IEEE web site at http://computer.org/software/homepage/2001/05Design/index.htm.

Beck also emphasizes the importance of a collaborative work area several times throughout this book. XP pair programming leads to a different work style than traditional development, and according to the author the ergonomics of the XP project team's work area are vital to the success of the project. The ideal XP project work area would be a set of tables in a "bullpen" where pair programmers could comfortably share a keyboard and mouse at each workstation. There should also be a machine designated as the integration machine so that pair programmers who have completed some code and the requisite unit tests can move easily to the integration machine and merge their work into the project.

In Section 3, *Implementing XP*, the author begins to address some of the practical concerns about the XP philosophy, and discusses a typical lifecycle for a system developed and maintained using XP. Beck discusses the various roles in XP project teams: customer, tester, tracker, coach, and, of course, programmers. He concludes his manifesto by discussing what makes XP difficult and gives some situations that are not appropriate for XP. This dose of reality gave the book balance by explicitly admitting that XP isn't a development methodology for all projects, in all situations, in all cultures, for all time. One of the critiques of XP is that it does not scale, and Beck doesn't argue with this criticism. He says project team size definitely matters. If you have more than ten project team members, you are starting to exceed the ideal for XP.

Overall I found this book challenging to the conventional wisdom I have heard over the years in regard to software development methodologies. I believe Beck's book gave me a better perspective and appreciation for the RUP by helping me understand where XP's strengths and weaknesses lie. In fact, some of the XP practices and values not already in the RUP could bring additional value to RUP implementation if the size and scale of the development effort is a fit. This book did build my interest in XP, but it is only enough to whet an appetite, not fill one's stomach with substance or provide enough nourishment to implement XP in a software development shop.

eXtreme Programming Explained: Embrace Change. 190 pages. Addison-Wesley. 2000. ISBN 201-61641-6

Extreme Programming is a hugely popular (although not as popular as Scrum) methodology focused on meeting changing client requirements. The first Extreme Programming project was started in March 1996, by Kent Beck at Chrysler. In his 1999 book, Extreme Programming Explained: Embrace Change, he detailed the aspects for software development. Kent Beck was also the pioneer of test-driven development, which put use-case testing on the radar as an improvement over the way things were done then: writing lines and lines of code and then testing it. He was also one of the original signatories of the A 27 Jul 2014CPOL. Extreme Programming Explained: Embrace Change, Addison-Wesley. This article is in the Book Review chapter. Reviews are intended to provide you with information on books - both paid and free - that others consider useful and of value to developers. Read a good programming book? Write a review!Â Today, I have to add Kent Beckâ€™s Extreme Programming Explained 2nd Edition to that elite group. The physical book looks deceptively thin - one hundred and sixty odd pages with an easy text and thick margins. Firstly, the thick margins were useful because they are now covered with notes and between those pages, I believe there is a message worth gold.