Foreword

Software development has come a long way.

Back in the dark ages of our profession, we clung to classical models rooted in traditional approaches to project management. Requirements were specified, then frozen, and it was off to the races. The cycle was clear: Plan the work, then work the plan. Results were, shall we say, mixed.

The problem often turned out to be a puzzling one. The software development organization could declare victory and demonstrate that they had delivered on the specified requirements. But from the business's point of view, the system was destined to be a failure, and this was often obvious from the moment of delivery. Why?

Because change happens. All significant software development efforts last sufficiently long so they are bound to suffer from changes due to the external environment. New technology bursts onto the scene, new competitors arrive, fundamental economic drivers shift, and so on. Sometimes you are forced by the outside world to innovate, and that forcing function doesn't always conveniently fall on software development project cycle boundaries. In today's fast-paced world, it is certain that if you build something precisely according to your initial set of requirements, you will build the wrong thing and fail in the marketplace.

Of course, there is also the problem of poor execution along the way, again due perhaps to ambiguities in the original requirements. But the argument, which has been made many times, is that you will fail *even in the face of perfect execution*. Because that approach is enough to drive any manager to drink, a new approach was clearly necessary, one that focused on business success, not just software development success.

If we now define a software development organization's success in terms of its ability to react to change, we have a new view of the problem. Change is expressed through requirements, so the fundamental issue facing any software development organization is its ability to deal with requirements flux. Moreover, changes to requirements arrive randomly; their scope and arrival time are externally determined. You may have a very neat project rhythm or cycle, but the changes to requirements will arrive asynchronously with that heartbeat. As with anything else in life that you can't control, what you need is a mechanism for coping. The message is clear: "Requirements change—adapt or die!"

That, in essence, is what iterative development is all about. It is software development's response to a fundamental business problem: delivering systems that have value. Yes, the "system" is a moving target. We have given up on the irrational belief that we can nail that target to the wall; we now work very hard to track the target and still hit the bull's eye.

Iterative development is a relatively new approach, having begun to gather momentum in the 1980s. Companies that embraced it in the 1990s found that they were doing a better job of delivering value than they had in the past with other methodologies. But, like all other advances in technology, the path was not always a smooth one; in hindsight, of course, it looks like a steady path "up and to the right." The reality is that there were lots of bumps in the road, some things that turned out to only look good at first, and many approaches that were tried and abandoned. What survived is the distillation of many projects' scar tissue, and we need to pay tribute to the managers who pioneered and persevered to give us a robust set of ideas for modern software development project management.

After a few decades, there is always a pause for reflection and summing up. Thought leaders sit down and try to capture, as best they can, the essence of what has been learned. Walker Royce did a great job in his 1998 classic *Software Project Management, A Unified Framework* (Addison-Wesley), and I attempted to add my own views on the subject in *The Software Development Edge* (Addison-Wesley, 2005). Now we have this addition by Kurt Bittner and Ian Spence.

Kurt and Ian are far from newcomers on the scene. They have many collective years of experience working with iterative development. Kurt was on the original Unified Process development team at Rational Software and led many teams applying the iterative approach. Ian's experiences are similar, having spent many years working with project teams applying iterative development. They have lived "in the trenches," and the guidance presented in this book reflects their experience.

They are also experienced writers. Their 2002 book, *Use Case Modeling* (Addison-Wesley), has already guided a generation of analysts and developers in getting requirements right. Their approach in that book was methodical and thorough, without being heavy and pedantic. It was a book that was written to be used, and it has been extremely well received. When I heard they were going to write the definitive book on iterative development, I jumped at the chance to be a reviewer.

They have not disappointed. This is a book for people who want to understand the principles and be successful in their practice. This is not a superficial popularization; this is "the real deal." I would recommend it to any software development manager who is serious about getting better at his craft. No matter how experienced you are, you will learn something from this book.

Great job, guys!

Joe Marasco
President and CEO
Ravenflow
March, 2006