Foreword

In reading a book or manuscript for the first time I always ask myself the same question, "What is the author adding to the state of the art on this subject?" In Mike's case the answer is twofold: His book adds to our knowledge of "how" to do estimating and planning, and it adds to our knowledge of "why" certain practices are important.

Agile planning is deceptive. At one level, it's pretty easy—create a few story cards, prioritize them, allocate them to release iterations, then add additional detail to get a next iteration plan. You can show a team the basics of planning in a couple of hours, and they can actually turn out a tolerable plan (for a small project) in a few more hours. Mike's book will greatly help teams move from producing tolerable plans to producing very good plans. I'm using my words carefully here—I didn't say a great plan, because as Mike points out in this book, the difference between a good (enough) plan and a great plan probably isn't worth the extra effort.

My early thoughts about Mike's book have to do with the concept of agile planning itself. I'm always amused, and sometimes saddened, by the lack of understanding about agile planning. We hear criticisms like "agile project teams don't plan," or "agile teams won't commit to dates and features." Even Barry Boehm and Richard Turner got it wrong in *Balancing Agility and Discipline: A Guide for the Perplexed* (Addison-Wesley, 2004) when they talk about plan-driven versus agile methods. Actually, Boehm and Turner got the idea right, but the terms wrong. By plan-driven they actually mean "greatly weighting the balance of anticipation versus adaptation toward anticipation," while in agile

methods the weighting is the opposite. The problem with the words "plan-driven" versus "agile" is that it sends entirely the wrong message—that agile teams don't plan. Nothing could be further from the state of the practice. Mike's book sends the right message—planning is an integral part of any agile project. The book contains a wealth of ideas about why planning is so important and how to plan effectively.

First, agile teams do a lot of planning, but it is spread out much more evenly over the entire project. Second, agile teams squarely face the critical factor that many non-agile teams ignore—uncertainty. Is planning important?—absolutely. Is adjusting the plan as knowledge is gained and uncertainty reduced important?—absolutely. I've gone into too many organizations where making outlandish early commitments, and then failing to meet those commitments, was acceptable, while those who tried to be realistic (and understanding uncertainty) were branded as "not getting with the program," or "not being team players." In these companies failure to deliver seems to be acceptable, whereas failure to commit (even to outlandish objectives) is unacceptable. The agile approach, as Mike so ably describes, is focused on actually delivering value and not on making outrageous and unachievable plans and commitments. Agile developers essentially say: We will give you a plan based on what we know today; we will adapt the plan to meet your most critical objective; we will adapt the project and our plans as we both move forward and learn new information; we expect you to understand what you are asking for—that flexibility to adapt to changing business conditions and absolute conformance to original plans are incompatible objectives. *Agile Estimating and Planning* addresses each of those statements.

Returning to the critical issue of managing uncertainty, Mike does a great job of looking at how an agile development process works to reduce ends uncertainty (what do we really want to build) and means uncertainty (how are we going to build it), concurrently. Many traditional planners don't understand a key concept—you can't "plan" away uncertainty. Plans are based on what we know at a given point in time. Uncertainty is another way of expressing what we don't know—about the ends or the means. For most uncertainties (lack of knowledge) the only way to reduce the uncertainty and gain knowledge is to execute—to do something, to build something, to simulate something—and then get feedback. Many project management approaches appear to be "plan, plan, plan-do." Agile approaches are "plan-do-adapt," "plan-do-adapt." The higher a project's uncertainties, the more critical an agile approach is to success.

I'd like to illustrate the "how's" and "why's" of Mike's book by looking at Chapters 4 and 5, which detail how to estimate story points or ideal days and provide an explanation of the pros and cons of each. While I have used both

approaches with clients, Mike's words crystallized my thinking about the benefits of story-point estimation, and I realized that story points are part of an evolution, an evolution toward simplicity. Software development organizations have long looked for an answer to the question, "How big is this piece of software." A home builder can do some reasonable estimating based on square footage. While estimates from builders may vary, the size is fixed (although finish work, material specifications, and more will also impact the estimates) and remains a constant. Software developers have long searched for such a measurement.

In software development we first utilized lines-of-code to size the product (this measure still has its uses today). For much day-to-day planning, lines-of-code proved to be of limited use for a variety of reasons, including the amount of up-front work required to estimate them. Next on the scene came function points (and several similar ideas). Function points eliminated a number of the problems with lines-of-code, but still required a significant amount of up-front work to calculate (you had to estimate inputs, outputs, files, and so on). But what dooms function points from widespread use is their complexity. My guess is that as the complexity of counting has gone up—a quick perusal of the International Function Point User Group (IFPUG) website indicates the degree of that complexity—the usage in the general population has gone down.

However, the need to estimate the "size" of a software project has not diminished. The problem with both historical measures is twofold—they are complex to calculate and they are based on a waterfall approach to development. We still need a size measure, we just need one that is simple to calculate and applicable without going through the entire requirements and design phases.

The two critical differences between story points and either lines-of-code or function points are that they are simpler to calculate and they can be calculated much earlier. Why are they simpler? Because they are based on relative size more than absolute size. Why can they be calculated earlier? Because they are based on relative size more than absolute size. As Mike points out, story-point estimating is about sitting around discussing stories (gaining shared knowledge) and guestimating the relative story size. Relative sizing, as opposed to absolute sizing, goes remarkably quickly. Furthermore, after a few iterations of sizing and delivering, the accuracy of a team's guestimates improves significantly. Mike's description of both the "how" and the "why" of story-point versus ideal days estimating provides keen insight into this critical topic.

Another example of Mike's thoroughness shows up in Chapters 9 to 11, on the prioritization of stories. Mike isn't content with telling us to do the highest value stories first, he actually delves into the key aspects of value: financial benefits, cost, innovation/knowledge, and risk. He carefully defines each of these

aspects of value (including a primer on Net Present Value, Internal Rate of Return, and other financial analysis tools), and then provides several schemes (with varying levels of simplicity) for weighting decisions using these different aspects of value.

Often, people new to agile development think that if you are doing the twelve or nineteen or eight practices of a particular methodology that you are therefore Agile, or Extreme, or Crystal Clear, or whatever. But in reality, you are Agile, Extreme, or otherwise when you know enough about the practices to adapt them to the reality of your own specific situation. Continuous learning and adaptation are core to agile development. What Mike does so well in this book is provide us with the ideas and experience that help take our agile estimating and planning practices to this next level of sophistication. Mike tells us "how" in depth—for example, the material on estimating in story points and ideal days. Mike tells us "why" in depth—for example the pros and cons of both story points and ideal days. While he usually gives us his personal recommendation (he prefers story points), he provides enough information so we feel confident in tailoring practices to specific situations ourselves.

So, this, in the end, identifies Mike's significant contribution to the state of the art—he helps us think through estimating and planning practices at a new depth of knowledge and experience (the how) and then helps us frame decisions about using this new knowledge in adapting these practices to new, unique, or merely specific situations (the why). Of the half-dozen books I regularly recommend to clients, Mike has written two of them. *Agile Estimating and Planning* goes on my "must read" list for those wanting to understand the state of the art in this aspect of agile project management.

Jim Highsmith
Agile Practice Director, Cutter Consortium
Flagstaff, Arizona
August 2005