Increased transparency also extends to individuals and teamwork. Because agile teams tend to work more closely together, it's difficult for one team member to operate deviously (e.g., hanging on the coattails of more talented programmers and taking the credit for their work).

The Goals of Agility

When deciding whether to adopt agile practices for your project, it's important to take a step back from the hype and ask what's in it for you (or, more accurately, for your project). So, what does your project stand to gain from adopting agile practices? What are the benefits, the goals, the problems that agility sets out to solve? And do these problems exist on your project?

The Agile Manifesto website (see `www.agilemanifesto.org`) describes the principles and values that agile teams need to follow to describe themselves as "agile." But oddly, it doesn't describe the actual goals of software agility—that is, why you would want to make your project agile in the first place (although some goals are described by the group's 12 principles, listed at `www.agilemanifesto.org/principles.html`).

We've interpreted the goals of agility as the ability to

- Respond to changing requirements in a robust and timely manner.
- Improve the design and architecture of a project without massively impacting its schedule.
- Give customers exactly what they want from a project for the dollars they have to invest.
- Do all this without burning out your staff to get the job done.

These goals are sometimes overlooked in the rush to address the values. The agile values are important but, we feel, are really there to fulfill these goals.

The overriding agile goal is responsiveness to changing requirements. This was the main problem out of which agile processes grew. Offset against responsiveness is the need for the project to be robust. Usually, *robustness* means contingency—that is, having safety nets in place to mitigate risk and provide a backup plan if (when) things do go wrong. Robustness implies more effort; more practices to put in place that, while reducing risk, can slow down the project's overall pace (see Figure 1-2). (Sometimes, though, robustness can also mean simply using smarter practices to reduce risk).

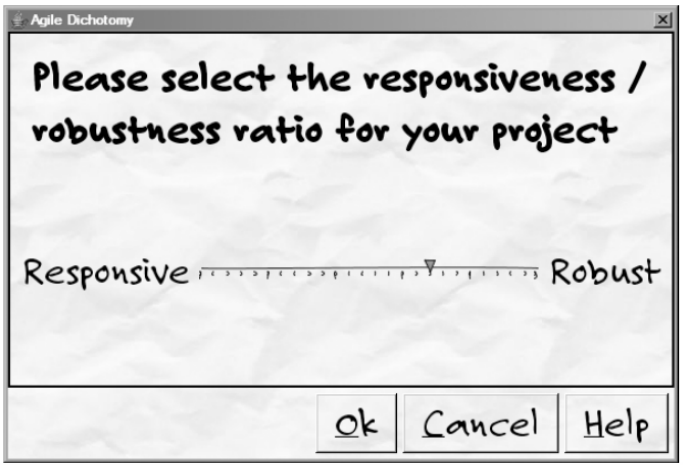

Figure 1-2. *The agile software dichotomy: responsiveness versus robustness*

Responsiveness in this context means the ability to react quickly (without regard for keeping all your safety nets in place, which is where robustness comes in). If you react to a change quickly but immediately introduce lots of defects as a result, that's very responsive but not particularly robust! Safety nets include things like unit tests, customer acceptance tests, design documents, defect tracking software, requirements traceability matrices, and so on.

So, ironically perhaps, to be really responsive (and to not introduce lots of defects every time the code changes), you need to push the slider further toward robustness. But then, the more safety nets you have in place, the more difficult it becomes to react quickly (i.e., the project becomes less agile).

Agility, then, is the ability to adapt to change in a timely and economic manner, or the ability to change direction while moving at speed. So a leopard is agile; an elephant isn't. But an elephant can carry a lot more than a leopard. But a horse is a lot more agile than an elephant and can carry a lot more than a leopard. ICONIX Process, without the agile extensions we present in this book, can be thought of as a "workhorse" process. By adding in a core subset of agile practices, we're aiming to create a "thoroughbred racehorse" process.

In other words, agile techniques work best for small projects but don't always scale very well. The techniques we describe in this book scale better than those in some agile processes because they place a greater emphasis on up-front design and documentation.

SCRIBBLED ON THE BACK OF A NAPKIN

The screenshot in Figure 1-2 looks like it was drawn on a crumpled-up piece of paper, although it is actually from a "real" program. It uses a readily available Swing look and feel called Napkin Look & Feel (see `http://napkinlaf.sourceforge.net`; this page also includes a link to a Java WebStart demo).

The idea behind this look and feel is that sometimes, when a developer shows a prototype GUI to a manager or customer, the customer assumes that what he's seeing is working software. (Appearances *are* everything, after all. . .)

On the other hand, if the working prototype was presented looking like a user interface mockup that had been scrawled on the back of a napkin, then the customer would be more likely to see it for what it actually is: a slightly working but mostly nonfunctional prototype that was cobbled together quickly so as to give the customer a rough idea of what the finished product will look like.

Seems like a great idea to us!

Why Is Agility Important?

Agile development has virtually exploded onto the software development world, bringing with it a shift in the way that software is developed. Agility makes a lot of sense because it addresses some of the common reasons why projects have failed over the years.

Back in 1995, the Standish Group's CHAOS Report[4] (a commonly cited report focusing on reasons for project successes and failures) showed that the primary reason for project failure was "lack of user input." The same study also found that the top element that dramatically increases the chance of success is "user involvement." So it isn't surprising that user involvement (and, in particular, user feedback, as early in the process as possible) is a hugely important driver behind agility.

Another commonly recognized contributor toward project failure is the "big monolithic project" syndrome, where a project doesn't get to see the light of day for a year or two. Finally, when this monstrosity rolls out of the programmers' workshop, the customer grows purple-faced and demands to know what this has to do with his original specification. It's quite likely that what the customer thought he was specifying was completely different from the way in which the programmers interpreted the spec, or the requirements might simply have changed beyond recognition in those 2 years, and somebody forgot to tell the programmers. It sounds absurd, but it happens—a lot.

Agile development addresses these issues: it cuts down the amount of time before the customer sees working software, and it encourages increased communication among everyone involved in the project (the programmers, the analysts, the customer, the users, the managers, the tea lady, and so on). If business customers are concerned

4. See `www.standishgroup.com/press/article.php?id=2`.