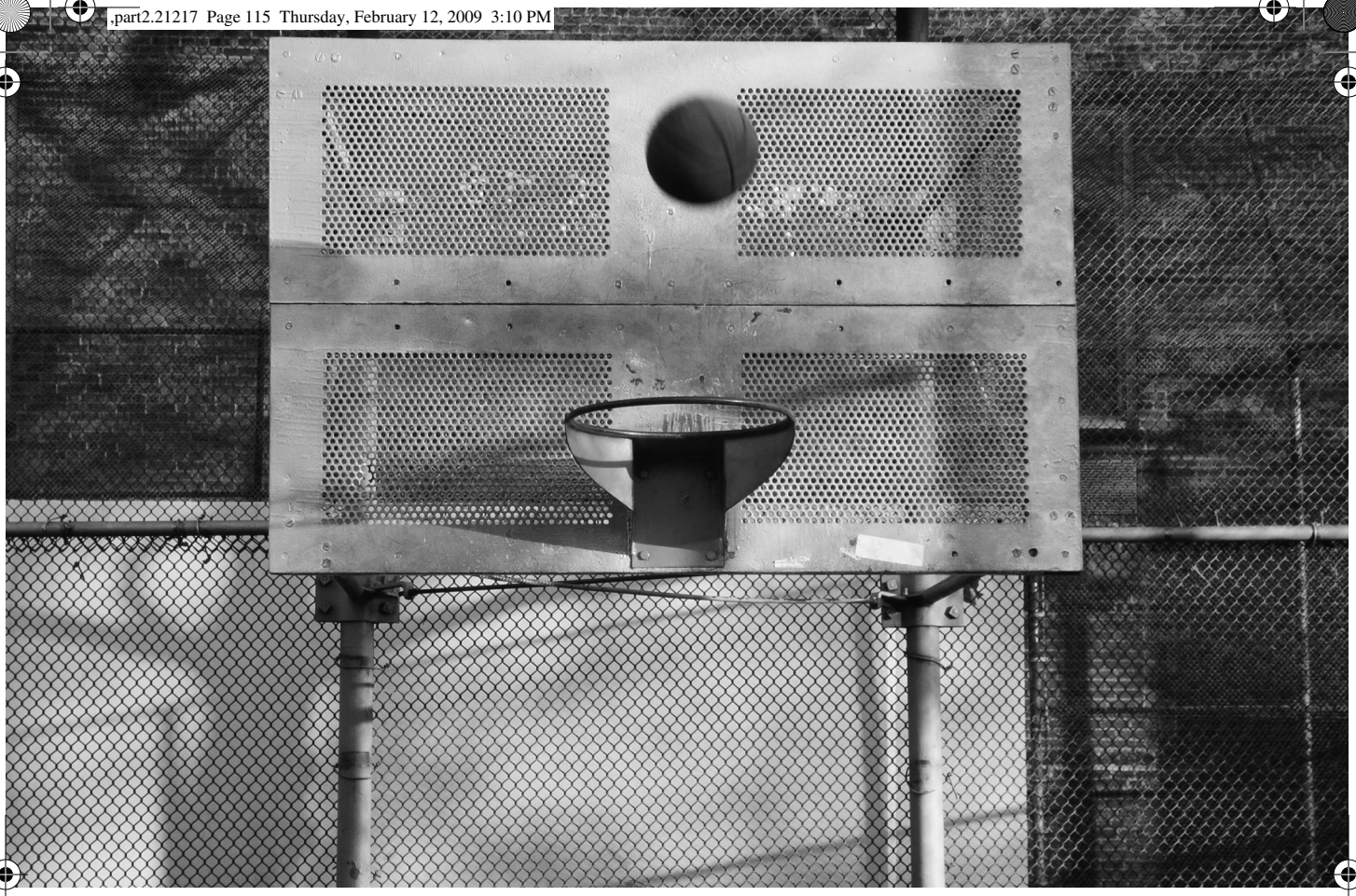

PART TWO

Goals

Let's say that you've got a job to do, and you need software to do it. And let's say that you've got two choices. The first is a brilliantly engineered piece of software. It never crashes, has a beautiful user interface, and has great technical support. It's fast, a pleasure to use, and inexpensive, and it runs on any operating system and practically any computer. But it doesn't do what you want. The second piece of software, on the other hand, is terrible. It's buggy, it crashes all the time, and it's slow. It runs only on an obscure operating system, and needs a very expensive and very fast machine if you want it to be at all usable. It's got a terrible user interface that makes even the simplest task a chore. But it does 20% of what you need. Which one do you choose?

Unfortunately for you, you'll end up with the poorly written piece of software, because it actually does something that you need it to do. And while this seems like a somewhat ridiculous scenario, it's actually not that far from the truth for a lot of us.

The worst possible mistakes in working with teams happen when your team's goals diverge. And, unfortunately, it's a lot more common than a lot of us realize. This shouldn't really be a huge surprise. Anyone who's cracked open a college textbook on software engineering has probably seen a chart that shows that it gets exponentially more expensive to fix a bug the longer it takes to realize that it's in the software. But most experienced programmers don't need a textbook to tell them this—they've almost certainly seen it firsthand.

One of the most common ways that software projects run into trouble is that people on the team are trying to solve the wrong problem. Over the years, we've talked to dozens and dozens of developers, testers, architects, project managers, and other software people who all universally recognize that gut-wrenching feeling that happens when you deliver what you think is the final product to the customer, only to have them say something like, "Well, that looks very nice, but isn't it also supposed to do…?" We all know the feeling of our hearts sinking when we find out about that enormously important feature that nobody told us about. Is there any experienced programmer among us who hasn't had the thought, "I would have built it entirely differently if someone had told me two months ago that it was supposed to do *that*!"

If you've spent any time on an agile team, then you probably recognize the saying, "Embrace change." One reason that programmers on agile teams work well is that they keep revisiting those goals. They'll make sure those goals are out in front of everyone: the team will make sure the goals are up on a whiteboard, and they'll have meetings specifically to make sure they're on top of any changes. And they get the customers involved in the day-to-day project work, because that's the most effective way to make sure that everyone's aligned to the same goals.

But even though we know how much those changes can damage the project, it's far too easy for us, as developers, to dismiss the idea of setting goals at the beginning of the project. And it's even easier for our own customers, users, and stakeholders to send us down the wrong path before we even get a say in the matter.

Customers—the people who we're building the software for—are not very good at telling us what they need. They'll ask for solutions instead of telling you what their problems are. They'll ask for a smoother, faster, less smelly horse and buggy, when what they really want is a better vehicle that will get them from point A to point B. They don't know enough to ask for an automobile, and it's all too easy to go about building a better, more improved horse and buggy. A developer needs to know enough about the whole transportation problem to find a better solution.

On the other hand, we, as developers, have our own peculiar problems. It's very easy for us to *think* that we know exactly what software we're about to build. We habitually interrupt our customers halfway through their explanations and say, "OK, I understand what you want." Then we go off into our cubicles and build the software that we think they need, only to find out that we completely misunderstood their problems. (We've done this ourselves on rare occasions, usually because we really wanted to work with a particularly cool new technology, and were basically looking for an excuse to play with it.)

So understanding your project's goals is critically important, and the stories and interviews in this section show us exactly how this affects our teams. It doesn't matter how good the software is, or if it's the wrong software for the customer. One of the biggest challenges of working with a team is keeping everyone aligned to that goal so that they build the right software. And even the best teams can have conflicts around those goals, conflicts that can tear a team apart. But if you align people to those goals from the beginning, and keep everyone in the loop as they change—and they *always* change—the project is much more likely to be a success.