

preface

In the summer, Texas rivers are dry. To find white water, kayakers have to follow the storms. One summer day in 1996, a partner and I left Austin at 8 P.M. to drive into the teeth of a huge thunderstorm and to follow it to the Cossatot River in Arkansas. When we arrived, the bad weather had played a cruel joke, swinging right around the river. Exhausted and disappointed, we pitched our tent on the bank of the river. We heard no raindrops that night.

In the morning, groggy and still disappointed, I stepped out of the tent and almost tripped ... right into the river. The Cossatot, notorious for rapid flooding, had gotten 6 inches of rain in 2 hours a scant 10 miles upstream. Now we were facing the prospect of running a river that was too *high*. We decided to save the difficult white water for the following morning and run the easier sections upstream. The meandering, beginner-level Class I creek had become a Class III cauldron of turbulence. It took us just 20 minutes to complete what the guidebooks called a “4-hour run.” The “intermediate” section downstream was even worse: a Class IV flume of exploding violence. After extensive scouting, we took turns standing on the bank with a safety rope while the other worked the section. We then parked our kayaks at the tents and hiked down to check out the advanced section. To our surprise, dozens of locals on lounge chairs relaxed on the banks. They faced what is normally a series of Class IV waterfalls now completely hidden beneath a maelstrom of mayhem. We had never seen more than a handful of spectators before. They were there to see the local hotdogs crash and

burn. Surprised by the scene, my buddy and I each sat on a boulder to watch the action ourselves.

Fast-forward to 2000. I leave a comfortable and successful career at IBM to join a startup called allmystuff, Inc. The economy is beginning to falter, but the company has just received funding while other Austin startups are biting the dust. This startup's business model does not depend on now dwindling advertising revenues, and the talent on allmystuff's team is amazing. When I join the company, it has \$10 million in the bank and customers and technology that indicate a likelihood of success. I have seen many of my friends leave IBM for less money and security, but plenty of adventure. I reason that I can always go back in a pinch. In the approaching darkness, I head out into the storm.

As the Austin reporters gleefully chronicle the demise of the once high-flying Austin startups one by one, allmystuff too begins to struggle. We work insane hours to deploy solutions for our few customers. In spite of our strong record for quality, the waning economy eventually catches up to us too. The venture capitalists opt to close down and restart with a new concept better suited to a receding economy. Despite the harshness of the events, I learned more during that time than any other in my career.

Like the locals on the bank of the Cossatot, most of us cannot resist a story when it involves real excitement and adventure, and even danger. Whether we are viewing a well-honed Greek tragedy or the latest pop-culture offering like the *Survivor* television series, we cannot get enough. Programmers are no different. We love what we call *merc talk*, mercenaries chatting about the latest battle's adventures, for many of the same reasons. Some of my most vivid memories of work are from around the ping-pong table at allmystuff. We talked about what prompted the brutal hours we were working. We talked about management philosophy, whether the code base was getting out of control, and whether XML with specialized viewers would give us a simpler solution than our increasingly complex JSP model. We talked about whether our graphic designer could handle the mapping of her user interfaces onto increasingly complex Java commands, in light of our increasingly delayed schedules. The enthusiasm generated by these conversations is what had prompted me to leave a safe career for a pay cut, insecurity, and a whisper of a hope at millions. And these experiences make me more valuable as a programmer, as a manager, and as an architect.

In a context I no longer remember, previous IBM Chairman John Akers once said that there were too many people "hanging around the water cooler." I remember that we were outraged. He didn't get it. Around a water cooler, or

bar, or ping-pong table is where you will hear the stuff that makes or breaks a project—or a company. This, the programmer’s mythology, must be nurtured and fed, for it is the stuff of life. I attempt to capture some of it in *Bitter Java*.

Turning time back again to before allmystuff, I am speaking at a conference. The title of my presentation is “Bitter Java.” During the conference I meet a well-respected Java programmer who is one of the inventors of JSP. He tells me he’s been in Pamplona and has run with the bulls. He was even gored. He proceeds to explain his bull-running strategy. My mind resists. Of all the people in Pamplona that day, he should be the last to tell me about how to avoid getting gored. I might as well consult O. J. Simpson about marital relations. Tens of thousands of crazed, adrenaline junkies run every year, and only a handful ever gets gored. But gradually my mind focuses: if I were to run, he might well be just the person I’d want to talk to. I would want to know how he planned his run, how he executed that plan, and what went wrong. That’s information I could use. As it turns out, the gored programmer is the vice president of engineering at allmystuff, and he recruits me to help start his services organization. Back to my presentation, although it might put the allmystuff job at risk, I decide to use the Pamplona story to start my talk. It captures the fundamental concepts of *Bitter Java* perfectly. If it helps avoid a subtle trap or process pitfall, a story about failure can be worth 10 stories about success. The story catches the audience’s attention, and ... I get the job anyway.

Like many programmers, I love extreme sports. We boaters are drawn to tragedy and sometimes play dangerous games. A well-known cautionary kayaking rule by author William Neely is that the time spent staring at a nasty hydraulic is directly proportional to the amount of time you’ll spend getting trashed in it. Said another way, if that hole looks nasty enough to eat you, it probably will. Kayakers have a good strategy for describing a run down a river. The guidebooks will point out a route *and the dangerous spots outside and along the route*. A guidebook might say, “Next, you will see a boulder in midcurrent. Go left. Should you blunder to the right, the *Terminator* hydraulic will certainly show you the error of your ways.” I learned long ago that even if you don’t know a river intimately, you will want to know its trouble spots. I want to know if a rock is undercut, and if that undercut is likely to trap me. I want to know where to punch that riverwide hydraulic, or how to miss the rocks on the bottom of the waterfall. I want to know if anyone has ever died on the river and how it happened. With enough information, I can usually avoid the hazards with skill or even by walking around them on the bank with my boat firmly on my shoulder.

The programmer in me wants to do the same. I need to understand where applications and projects fail. I need to know if I am making too many communications over that interface, and the techniques that can steer me through that turbulent spot. I need to understand where a technology is likely to break and if it is likely to scale.

It is my conviction that in order to be successful, our software development culture must embrace failure and learn from it. I have yet to see an example of an organization that systematically learns from its mistakes and is diligent at capturing the reasons behind the modification of a flawed design pattern or process in a regular, systematic way. I have seen a lot of code and not all of it was sweet. I have learned to love bitter Java. I hope you will too.