CHAPTER ONE

Wrestling with Dinosaurs

He that will not apply new remedies must expect new evils, for time is the greatest innovator.

FRANCIS BACON

In 1975, Fred Brooks compared the development of large software systems to dinosaurs, woolly mammoths, and saber-toothed tigers fighting the glutinous grip of the tar pit.[1*] Brooks predicted that the software engineering tar pit would continue to be sticky for a long time to come.

The problems that Brooks described more than twenty-five years ago were not new when he described them, and the software community has now had another quarter century to work on them. How much progress has been made?

Many of the problems plaguing the typical software project today haven't changed much. For example, schedule pressure is a common feature of today's projects. According to some estimates, excessive schedule pressure occurs in about 75 percent of all medium-sized projects and in 90 percent or more of all large projects.[2] Overtime is more the norm than the exception.[3] Modern startup companies are known for the long hours they expect from their employees and stories of programmers sleeping under their desks abound,[4] but as long ago as the mid-1960s one report stated that, "In many companies, programmers faced with deadlines have been known to spend nights in the offices."[5] In 1975, Fred Brooks pointed out

*Citations for data cited and further reading can be found in the Notes section at the end of each chapter.

that "more software projects have gone awry for lack of calendar time than all other causes combined."[6] Schedule overruns have been around for at least 30 years and—people's impatience being what it is—probably since time immemorial.

The scope of today's large software projects seems daunting, and a natural tendency is to think that no one ever attempted projects of the scope we now face. Yet even a huge project such as the development of Windows NT has historical precedents. The initial Windows NT project required about 1,500 staff-years of effort,[7] but the development of IBM's OS/360, which was completed in 1966, required more than three times as much effort.[8]

Recent surveys have found that the most frequent causes of software project failure have to do with requirements problems—requirements that define the wrong system, that are too ambiguous to support detailed implementation, or that change frequently and wreak havoc on the system design.[9] But requirements problems are not new. Back in 1969, Robert Frosch observed that a system could "satisfy the letter of the specification and still not be very satisfactory."[10]

Modern developers rack their brains trying to keep up with the frenetic pace of change brought on by Web development. How do you keep up with new languages, shifting standards, and vendors that release new products every few months? To those of us who have been in the industry for a couple of decades, this sounds an awful lot like the mid-1980s when the IBM PC began to revolutionize corporate computing.

When the Fortran programming language was developed in 1954–58, it was supposed to eliminate the need for computer programming—scientists and engineers could simply enter their formulas into the computer, and the computer would translate the formulas for them, thus the name FORmula TRANslation. Of course, Fortran didn't eliminate programming; it just reduced the need for machine-language programming. From time to time we still hear about the promise of automatic programming.[11] Computers will become so advanced that the need for computer programmers will disappear. But this was already a well-worn chestnut more than thirty-five years ago when Gene Bylinsky reported that, "Predictions of businessmen blithely conversing with their omnipotent machines in

plain English still get played up regularly in the press."[12] The reality is that defining problems in painstaking detail is difficult work. That aspect of computer programming will not go away. New tools are useful, but not a substitute for clear thinking. I made that point in my 1996 book *Rapid Development,* but Robert Frosch had already made the same point in *IEEE Spectrum* 30 years earlier.

Internet developers talk about development in Internet time. The Internet makes it possible for developers to roll out revisions to their programs with ease. Users can download upgrades electronically, without requiring duplication of CDs or DVDs, which makes delivery of upgrades quick and inexpensive. This contributes to pressure to release upgrades frequently in response to user requests. Internet developers say that users would rather get the software quickly than have it be perfect. Internet developers sometimes say, "It's better to be first than right."

How unprecedented is this really? Some Internet developers think these dynamics are unique to Web projects, but industry old-timers know better: low rollout cost, easy corrections, low cost of failure—this sounds like a good old-fashioned in-house mainframe production environment.

These common threads tying together 25 years of software development are a source of both comfort and despair. The despair arises from the fact that some problems have been with us for a quarter century or more and are still common. We truly have been stuck in the tar pit a long time. The comfort arises from the same source: we've been staring at the same problems long enough to recognize the patterns, and—as I intend to explore throughout the rest of this book—we seem to be the brink of fixing them.

Notes

1. Brooks, Frederick P., Jr., *The Mythical Man-Month,* Anniversary Edition, Reading, MA: Addison-Wesley, 1995. The original edition was published in 1975.
2. Data on frequency of schedule pressure comes from Capers Jones, *Assessment and Control of Software Risks,* Englewood Cliffs, NJ: Yourdon Press, 1994. For information on common schedule performance, see The Standish Group, "Charting the Seas of Information Technology," Dennis, MA: The Standish Group, 1994; and

Capers Jones, *Patterns of Software Systems Failure and Success,* Boston, MA: International Thomson Computer Press, 1996.

3. I discuss this in detail in *Rapid Development* (Redmond, WA: Microsoft Press, 1996).
4. Bronson, Po, "Manager's Journal," *Wall Street Journal,* February 9, 1998.
5. Bylinsky, Gene, "Help Wanted: 50,000 Programmers," *Fortune,* March 1967, pp. 141ff.
6. Brooks, Frederick P., Jr., *The Mythical Man-Month,* Reading, MA: Addison-Wesley, 1975.
7. This number is estimated. It is based on the reported cost for Windows NT being $150 million (Zachary, Pascal, *Showstopper! The Breakneck Race to Create Windows NT and the Next Generation at Microsoft,* New York: Free Press, 1994) and a fully burdened labor cost of $100,000/staff-year.
8. Effort was approximately 5,000 staff months. Brooks, Frederick P., Jr., *The Mythical Man-Month,* Reading, MA: Addison-Wesley, 1975.
9. Cole, Andy, "Runaway Projects—Cause and Effects," *Software World,* Vol. 26, no. 3, pp. 3–5. The Standish Group, "Charting the Seas of Information Technology," Dennis, MA: The Standish Group, 1994.
10. Frosch, Robert A., "A New Look at Systems Engineering," *IEEE Spectrum,* September 1969.
11. Rich, Charles, and Richard C. Waters, "Automatic Programming: Myths and Prospects," *IEEE Computer,* August 1988.
12. Bylinsky, Gene, "Help Wanted: 50,000 Programmers," *Fortune,* March 1967, pp. 141ff.