Essays on Managing Successful Projects

The Software Development Edge

Joe Marasco

Foreword by Mike Devlin, General Manager, Rational Software, IBM

Praise for The Software Development Edge

"This set of articles captures decades of 'in-the-trenches' experience across a broad spectrum of software topics. Joe Marasco has the scars and the smarts to articulate patterns of success that can satisfy a broad audience. He uses mathematics, physics, common sense, and storytelling along with a no-candy-coating style to provide unique perspectives on significant problems in delivering software results as a business. Whether you are a computer science theoretician, a frustrated software project manager, a successful businessman, or a skeptical programmer, you will learn a lot from this compilation."
—Walker Royce, author of Software Project Management: A Unified Framework *(Addison-Wesley), and Vice President, IBM Software Services-Rational*

"Joe Marasco's readable essays on managing successful projects shows that software development managers——no different from all managers——must embrace the fundamentals of management if they are to succeed: working through people and process to be decisive, dealing with politics, keeping on schedule, and, yes, shipping a well-developed product. Marasco uses plain English to explain many integrated skills, ranging from estimating the time it will take to really do things, to negotiating effectively, even to eloquently describing three distinct phases of our personal development. He frequently uses a "can we talk?" conversation with a fictional colleague, Roscoe Leroy, in a Socratic dialogue to illustrate the two sides to a point in many areas (reminiscent of Galileo's writings to explain his then-heretical views); in this case, Marasco's advice will help technology professionals escape the clutches of pervasive Dilbertian incompetence, and enable readers to be more effective in our ever-changing world."
—Carl Selinger, author of Stuff You Don't Learn in Engineering School: Skills for Success in the Real World *(Wiley-IEEE Press), and Contributing Editor,* IEEE Spectrum *magazine.*

"This is a book loaded with practical experience distilled into insights useful for every software manager, and for thoughtful software engineers. Joe brings his keen observations on engineering, physics, software, and management to bear on managing software projects, and the real-world practices and problem-solving techniques required to be successful."
—John Lovitt, Senior VP, Rational Software (retired)

"You don't have to be in the software business to benefit from this book. This is a book that any manager at any level can pick up at any point and enjoy at any time. Highly recommended reading."
—R. Max Wideman, Fellow, Project Management Institute, AEW Services, Vancouver, Canada

"Marasco's book makes for fascinating reading for anyone concerned with management problems in general. The reader is introduced to simple quantitative models, based on the author's experience, that help in assessing performance and choosing a roadmap to successful completion of a complex project. In addition, the author's use of anecdotes and his writing style make for both an entertaining and informative read."
—Professor Martin Lesser, Department of Mechanics, Royal Institute of Technology (KTH), Stockholm

"Individually, each chapter is articulate, engaging, thought- provoking, informative, and well informed. Collectively, much as a compilation of short stories from a skilled, mature author, these chapters invite and help train the reader to see into and to understand what is often only observed in passing or scarcely noticed. Seeing the big picture in little things and identifying critical components of the large landscape, Marasco's analytic and synthetic skills both impress and enlighten."
—Stephen D. Franklin, University of California, Irvine

"Practical advice on project management expressed in an erudite and entertaining style. The insights are an impressive synthesis of management principles and practical experience that should contribute to improved project management in any organizational activity."
—Steven Globerman, Kaiser Professor of International Business, Western Washington University

"Joe is an emotionally intelligent leader, and an insightful holistic systems thinker. Both qualities come alive in his writing."
—Yosi Amram, CEO Coach, and former Founder and CEO, Individual Inc. and Valicert Inc.

"Joe Marasco has assembled a smorgasbord of thought-provoking material that will appeal to anyone who deals with software projects and software developers. Dip into the book at any place and savor new ideas about the nature of software, how to motivate professionals, or one of many other topics that will make you stop and think about your current notions. You may not agree with everything Joe says, but you will respect his position and probably change many of your ideas. Many of the chapters will be required reading for my software engineering students."
—Gary Pollice, Professor of Practice, Computer Science, Worcester Polytechnic Institute

"A unique and passionate philosophical work, with wisdom distilled from both academic and business careers.... A down-to-earth, subversive, and witty guide for managing the software of machines and of the human soul. Joe shows how the foremost ingredient for success and happiness, in business and in life, is integrity."
—Kate Jones, President, Kadon Enterprises, Inc. (www.gamepuzzles.com)

"Joe Marasco's new book is not only a solid introduction in project management discipline, but is also entertaining reading. Using very simple, easy-to-understand-and-relate-to examples, he manages to uncover and describe the most fundamental issues of successful team building and people and project management. His book is a must-read for everyone who wants to get a deeper insight and understanding of this discipline."
—Boris Lublinsky, Enterprise Architect, CNA Insurance

"There isn't a manager worth his salt who is going to look for salvation in a 'how to' book. What he is going to look for, however, is a nugget or two that will help him out of a specific dilemma or, given time for reflection, provide him with reinforcement that he is on the right track, or trigger thought processes about new ways to solve stubborn, recurrent problems. Well, Marasco provides a mother lode of nuggets for a manager to mine."
—Bill Irwin, Retired Executive, High Technology Industry

This page intentionally left blank

The Software Development Edge

This page intentionally left blank

The Software Development Edge

Essays on Managing Successful Projects

Joe Marasco

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Catalog Number: 204118158

ISBN 0-32-132131-6

Text printed in the United States on recycled paper at R. R. Donnelley in Crawfordsville, Indiana.

First printing, April 2005 (Updated 2/15/11)

To Wini, the light of my life.

This page intentionally left blank

Contents

About the Author

Joe Marasco is a retired senior vice-president and business-unit manager for Rational Software, now one of the five brands of the IBM Corporation. He held numerous positions of responsibility in product development, marketing, and the field sales organization, overseeing initiatives for the Rational Apex product and Visual Modeler for Microsoft Visual Studio. In 1998, he served as Senior VP of Operations. He retired from Rational in 2003. He holds a bachelor's degree in chemical engineering from The Cooper Union, a Ph.D. in physics from the University of Geneva, Switzerland, and an M.S.A. from the University of California, Irvine Graduate School of Management. When not writing, he barbecues and plays golf; his ribs are much tastier than his scores.

This page intentionally left blank

Foreword

Why should anyone listen to what Joe Marasco has to say about software development and the people who do it?

In the spring of 1991, we awarded Joe his five-year service award. Back then, Rational Software was small enough to individualize these awards, and Joe's was quite unusual. We obtained the hood ornament from a Mack truck—the classic bulldog—and mounted it on a plaque. Everyone agreed that this award symbolized Joe's dedication and tenacity when it came to getting the job done.

So it wasn't too surprising when, later that year, we selected Joe to lead a watershed development effort at Rational. At the time, our flagship product, the Rational Environment, ran on proprietary hardware, and we realized the importance of moving it to the UNIX platform. While this move was inevitable, it was fraught with risk; in fact, many other companies suffered fatal damage trying to move their software solution from proprietary hardware to industry standard platforms—among the carcasses littering the side of the road were electronic design automation (EDA) companies such as Daisy and language-specific companies such as Symbolics. The task was known to be difficult, the result uncertain, and the need essential.

Joe, on the other hand, was positively ancient by Silicon Valley standards: he was 46. But we believed that Joe's experience and "steady hand at the tiller" would see this project through. We also knew that Joe would do whatever it took to get the project done.

The record speaks for itself. In September 1991, Joe took on the leadership of the new team with a plan to deliver "Rational II" in two years on two UNIX platforms. After seven months, a limited-function subset prototype was up and running. After 16 months, the development team was "self-hosted," which meant that it was able to complete the development of the product using the partially-completed product itself. And, to the minute, the team delivered what became known as Rational Apex on two UNIX platforms—IBM and Sun—in the two years that had been promised.

Apex was an extremely successful product, one that is still delivering value to customers today. Joe was the Business Unit Manager through releases 2.0 and 3.0 and also oversaw its porting to every significant UNIX platform and to the Windows platform as well. More important, for the 10 years following the release of Apex 1.0 in 1993, Joe was the "go-to" guy whenever we had a difficult product delivery problem in the company. As the company grew through merger and acquisition, Joe assumed the role of troubleshooter, helping out wherever the need was greatest, wherever the pressure was most intense.

One of the reasons Joe was so successful in delivering products is that he spent a lot of time with his developers understanding the details of the products and the development problems. But he also spent a lot of time with Rational's customers, developing a keen understanding of their needs. As every product delivery is the result of many compromises, Joe was always well informed to make good judgments when it came to product decisions.

Near the end of his career at Rational, Joe began to write about software development in a series of articles in Rational's e-zine, *The Rational Edge.* Unlike the articles written by our "Three Amigos," these were much more down in the trenches, reflecting his experiences both at Rational and in his previous environments. What we discovered was that Joe was able to cogently articulate his experience and act as a "virtual mentor" for budding software development managers worldwide. The response to these articles was extremely positive, and it is a pleasure to see them all collected here in one place.

This is not a theoretical treatise on software development. That's not what Joe was all about. Joe was about having his teams ship products we could all be proud of, products that were easy to maintain and offered real value to customers. If you want to develop products that you can be proud of, that lend value to your customers, this book is a "must-read."

Mike Devlin
General Manager
Rational software, IBM

Preface

This book draws heavily on a series of columns called Franklin's Kite that appeared in Rational Software's e-zine, *The Rational Edge,* in the early 2000's. These articles were aimed at software development managers, and their goal was to help readers avoid many of the common pitfalls that await software development projects and teams. More than 20 appeared, and we—my editor, Mike Perrow, and I—noticed that readers often began their monthly perusal of *The Edge* with *The Kite.*

My intention here is to not only to collect these articles but to sew them together in a form that makes them even more useful for software development managers *and their managers.* I have done that by reorganizing them thematically, instead of presenting them in the order they originally appeared. This has caused me to do some light editing in places where "forward-referencing" would otherwise take place. I have also paid attention to the footnotes, many of which appeared in the original as URLs and appear here as more formal citations where appropriate. Finally, I have added material at the beginning and end of each chapter so that the context of each article as part of the whole becomes clearer.

The reader will quickly note that the chapters have several different styles. Some of them are expository, some are fairly analytical, and some are folksy "Socratic dialogs" between the author and his avatar, one Roscoe Leroy. Roscoe is an invented character, a good technical general manager who initially knows little about software development. I use him as a foil, allowing his "naïveté" to force me to explain things without using technical jargon. My approach is ecumenical and subversive: I will use any technique that permits me to get the message across. Some of these chapters will appeal to some readers, and others will appeal to others. Whatever works is, by definition, good. I take my cue from Horace, who wrote in *The Art of Poetry,* "He has won every vote who has blended profit and pleasure, at once delighting and instructing the reader."[1]

I divide the work into six parts of four chapters each. Briefly:

- **General Management:** These chapters deal with topics that are useful to managers in general and also expose the reader to my background and biases. I include them so that we have a common baseline for what follows.
- **Software Differences:** In this section, we take a look at those things that distinguish software development from other management challenges.
- **The Project-Management View:** I take the perspective that a software-development project is a variant of the generic project and, as such, amenable to classical project-management techniques. On the other hand, I strive to point out what *is* different about software development.
- **The Human Element:** I turn around in this section and look at software development from the perspective of the people who do it. Once again, I try to compare and contrast that which is similar to that which is different for software-development projects.
- **Thinking Laterally:** Software people come at problems from many different points of view. In this section, I expose the reader to some of the more speculative and original ideas that he or she may not have seen before.
- **Advanced Topics:** The successful software-development manager is like a really good pinball player: His reward for high scoring is given in free games. This additional "stick time" leads to his becoming even more proficient. In this section, I talk about some of the challenges that come with success.

This book has 24 chapters.[2] You can read it serially, or pick out a chapter at a time; they can stand on their own. This is a good "airplane book"; read a chapter and then think about it for the rest of the flight. If you get just one new idea from one of the chapters that covers the price of the book, I will have been successful.

With these prefatory remarks out of the way, let's get down to it.

[1] Horace, *Satires, Epistles, and Ars Poetica* (Cambridge, Massachusetts: Harvard University Press, 1999). The original Latin text is "Omne tulit punctum qui miscuit utile dulci, lectorem delectando pariterque monenendo." It can be found at line 343 of *Ars Poetica.*

[2] Coincidentally, so does *The Iliad.*

Chapter 8

Getting It Out the Door

I have on occasion claimed that I can build the perfect product. Just don't ask me to ever ship it.

As soon as you require that I ship a product on a given date, I can guarantee you that the product will be imperfect. It will disappoint someone along some dimension. It will lack some feature, exhibit some annoying minor bug, or will lack some piece of documentation. No doubt there will be rough edges in its user interface. If only we had more time…

This is not a phenomenon unique to software products. A shipped product is always a compromise between the product we would ideally like to ship—the one that approaches perfection—and the one we really need to ship because we must begin generating revenue. And sometimes, believe it or not, the product we ship *is* good enough, even though it represents a compromise. The test is whether or not it serves the greatest good for the greatest number.

Consider an *update release* of an existing product, one that will add some new features and fix many annoying defects resulting from the previous compromised release. You can work on this update as long as you like; the longer you take, the more features you can add and the more bugs you can fix. But here's another way to look at it: The longer you wait to ship that update release, the longer your existing customers will have to live with the bugs in the version they are currently using. So the tradeoff becomes this: Is it better to ship 50 bug fixes today, or 55 in another two weeks? If you have thousands of customers who are suffering with Bug #29 on the list every day, I think I can make a pretty good argument for shipping *yesterday*.

Once you realize that shipping the product is not only part of your job but in fact the critical step—Bob Bond[1] would call it "running it through the tape"—you need to consider exactly what is required to go from some assemblage of working bits to a package that you can put on the loading dock or, alternatively, some set of files that you can stage on your download server. You need to consider testing, installation, documentation, preparing the support organization, and many, many other details. Like the death from a thousand cuts, getting this all right can be extremely painful the first hundred or so times you do it. It is one of those exercises that require method and persistence, and extremely meticulous follow-up.

The purpose of this chapter is not to bludgeon you to death with the obvious. What I focus on in this chapter is a small subset of the problem: How do we "close out" development of the software so that we can ship the product? When we are on "final approach" to shipping the product, what changes? The answer is this: If you have been doing it right, the change is imperceptible. If you have neglected thinking about this problem all along, then you will suffer large, severe, and disruptive change at the end, and your ability to ship will be endangered.

If You Build It, They Will Come[2]

In the world of software products, there are successes and failures, determined by the free market system. We must, of course, add to the list of failures those projects whose products never see the light of day—the ones that are worked on for various lengths of time but never ship. As obvious as it sounds, you cannot be successful unless you meet the precursor of shipping your product.

As you cannot ship what you cannot build, actually putting together the pieces becomes critical. Intrinsic in this is the concept of a *repeatable* build process. You will build the product over and over again, until one of your *candidate releases* passes muster and you let it out the door. I now confront the issues involved in creating such a repeatable build process for your product.

[1] Bob Bond ran sales and marketing at Rational for many years. He was a very positive mentor for me.

[2] In fact, the line from the movie "Field of Dreams" was, "If you build it, he will come," the "he" being either Shoeless Joe Jackson or the principal character's father. The line has been so frequently misquoted that most people use this one. Of course, at the end of the movie "they" come, as illustrated by the stream of headlights across the Midwestern plain.

In the Beginning, There Was the Sandbox

Products come out of projects, and projects tend to begin in haphazard ways. Organizations with well-defined processes have developers building their components in local work areas, sometimes called *sandboxes*. They provide for mechanisms whereby the subproducts of these sandboxes can be assembled, sometimes in *ad hoc* ways, so that each development team can test its progress in the context of the whole product. Configuration management systems allow for appropriate partitioning such that each developer (or team of developers) has the autonomy and isolation to work on his piece without stepping on the other guy's toes, while at the same time providing for a loose integration context.

This works fine in the early, chaotic days when everything is changing very rapidly, and before architectures are well-defined and interfaces are nailed down. However, before too long even modest projects outgrow this framework. At that point, one of two things happens: Either the organization makes *the build* a priority and adds some structure, or it doesn't. In general, those that do establish a regular "heartbeat" for the project—a periodic, regular, and dependable build cycle—improve their chances for success. Those that don't establish this rhythm find that entropy begins to take over, and that building the product becomes more difficult over time.[3]

Many organizations vastly underestimate the effort it takes to put a good build process in place. Because of this, projects in their latter stages often have a "new" problem to deal with: In addition to having buggy software, incomplete parts, and so on, they also struggle with something that they have taken for granted—the simple assembly of their product. This is a trap for the unwary. In order to not fall into the trap, you need to understand more about the process of assembling a product.

Why Should the Product Build Be Hard, Anyway?

First of all, the product you are going to ship has more pieces to it than the prototypes you have been putting together for internal consumption. Here's a classic example: Developers and testers rarely look at the help system, because they know the product well enough to play with it and test it. Once you are going to have outsiders try to use it, you need a well-elaborated and working help system for people to use. Further, you need instructions for installing the software in different computing environments, as well as various other adjuncts that you can live without when you are only consuming your software

[3] *Entropy* is the tendency that all systems have to move from an orderly state to a disordered state when left alone. It is a fundamental physical law. One might say that all attempts at progress, by any civilization, fly in the face of entropy. Another way to say this is that to bring order out of chaos takes work, and that once you stop working, entropy will cause the system to spontaneously move to a more disordered state. I will talk some more about this in Chapter 18, "Bad Analogies."

internally. So the first problem that comes up is one that might be dismissed as *packaging*. You need more pieces to ship a product than to use it internally; and further, you need to document all the little details that the internal team has always known or taken for granted.[4] Making the product ready for outside consumers is sometimes called *sanding off the rough edges*. Some of these "rough edges" can be very sharp, and because you don't catch them all the first time, your first consumers may cut their fingers on them.

Let's assume, however, that this is just a logistical exercise and that with enough planning you can avoid the "packaging" trap. In some sense, it can be put in the "annoying detail" category: If you ignore it, it will bite you; but if you are aware of it and plan for it, then it is relatively easy to overcome. So, forewarned is forearmed: Treat packaging as a purely technical problem and you will be fine.

In fact, there are three much more fundamental obstacles to success that come up over and over again. They are distinct and interrelated, and all three must be worked on to achieve a successful build process.

Obstacle 1: Organizational Politics

Many software development managers lose sight of the simple fact that controlling the build process is first and foremost a political problem. To put it simply, he who controls the build has an enormous amount of power. After all, the build cycle itself defines the rhythm of the entire development and test organization. Think of the build cycle as the software equivalent of a factory assembly line. The person who gets to define the characteristics of the line and its speed determines, to a very real extent, the output of the factory. Line workers are very aware of their subservience to the line. The cardinal sin in the factory is to slow down, or—Heaven forbid!—shut down the line. The software equivalent is submitting a set of changes that *breaks the build*.[5]

Now the build process is something that everyone must participate in, but only one group can control. By its very nature it is not a democratic enterprise; it requires a certain amount of hierarchical and structural apparatus to work at all. Everyone agrees on this, more or less. The sticky wicket is determining who gets the responsibility and authority to make it work. That group will, from that day forward, wield a lot of power and clout.

Because human beings are, in general, reluctant to give up this sort of power, the build process becomes a political football. Myriad discussions ensue as to who will have

[4] The standard vehicle for this is called the *release note*. The release note documents the limitations of this version of the software, known bugs, and so on. It is an attempt to characterize the state of the deliverable, as it is better to tell your consumers things you know about rather than have them discover them on their own. Sometimes the release note is called the *readme file.*

[5] There are legitimate reasons for shutting down the line, and sometimes the person on the factory floor is the most appropriate person to do this. On the other hand, shutting down the line by mistake is definitely not a good idea.

the right to do what to whom in the interest of the build process. All of the negative political tendencies of your organization will be exposed during these discussions.

The purists among you will cry out that political tendencies should be discouraged or even condemned, pointing out that the job is hard enough from a technical point of view and should not be "polluted" by politics. In most organizations, however, *wishing* politics away will not necessarily make them *go* away. Politics is a fact of life that must be dealt with.[6] However, you must get through this phase, as unpleasant as it first appears. Else, you will be incapable of dealing with the next two hurdles.

Here are some specific suggestions:

- Try to get the group to agree that someone has to be in charge, because a loose confederation approach is doomed to failure.
- Try to reach a reasonable compromise between the autonomy of the constituent teams and the centralized authority that will be required.
- Always make sure that the management team understands the importance of the issue and has the very best people assigned to the build.
- Later in this chapter, we will talk about having a *czar of the build*. Make sure it is a person who is technically competent, firm, fair, and respected by everyone. Install him or her early in the process and have this person guide you through the political shoals.
- Enlist management's support in crushing "bad politics," should it rear its ugly head.

Obstacle 2: The Process

Having hacked through all the political jungles that accompany conceding power to the build group, the participants must now agree on the process they will use. Just as form follows function, the process will often be shaped to mirror the political compromises that were made to get to this juncture. There is plenty of interaction between the first and second obstacles. In fact, often the process obstacle presents itself early on, in Phase 1, because it is being used as a surrogate by those who don't want to openly admit that there are unresolved political issues. In some organizations, we see these two obstacles mashed together into one giant hairball, which in turn gives "process" a bad name. You cannot use "process" to solve what are intrinsically political problems, much in the same way that you cannot "solve" technical problems through political compromise.

[6] My perspective is that there are "good politics," akin to the notion of "fighting fair," and that a healthy political process can and should work toward making good decisions. Then there are "bad politics," which make organizational objectives subservient to personal agendas and self-aggrandizement; this sort of politics needs to be stamped out wherever it is found. The problem, of course, is the gray zone in between. I treat this subject in more detail in Chapter 13, "Politics."

The basic tension at this point revolves around the people who want a strict, rigorous process—sometimes called *lots of rules and no mercy*[7]—versus the people who want a looser set of policies. Acknowledging that there is no single, simple, right answer is usually the best place to start here. Your process will have to be tuned to your organization, because all organizations have their peculiarities.[8]

That does not mean that you need to invent new process. I used the word "tune" advisedly in the previous paragraph because I am firmly convinced that the best way to deal with this issue is to start with a base process that has been demonstrated to work before. Rational Software's Unified Change Management (UCM), for example, has a rich legacy of successful application. We know it works across a broad spectrum of domains, applications, and organizations. Why start over? Do you really think you are going to do better?

There are a few traps you don't want to fall into at this point. One is the *religious wars* pitfall. In every organization there are process gurus who believe that they, and only they, have the magic formula. And sure enough, every time there are others who resist, quite certain of their own convictions.[9] Regardless of who is right or wrong, these crusades are totally unproductive, often revolving around obscure details of little importance. The strong manager needs to identify the religious process fanatics and stifle them early. Sometimes the only answer is to tell them to put a cork in it. Remember always that process is not an end in and of itself; it is a means to an end—shipping the product![10]

Another trap is to think that any process, no matter how good, can substitute for thought or judgment. For every ironclad rule, there is bound to be an exception. You will have to watch what is going on and make midcourse corrections, no matter what your

[7] I believe the world is indebted to James E. Archer for this characterization. Jim is one of the most effective development managers I know, having been the godfather for Rational's programming environments products from the very beginning. He and I had many interesting discussions on the right amount of process.

[8] Some people argue at this point that you should endeavor to get your process "right" and then tune your organization to fit the process. While this is a laudable objective and *theoretically* the right approach, I have rarely found it to be successful in practice. You cannot allow a regressive organization the prerogative of rejecting reasonable process; on the other hand, it is difficult to implement any process that is too far out in front of the organization that must carry it off.

[9] To illustrate how far out of control this can become, the wars are often characterized as struggles between the "process Nazis" and the "anarchists." With such value-laden labels, it is difficult to have discussions that will get to the right place.

[10] In a like manner, the anarchists will be hard put to demonstrate that they can ship product without any process. As is the case in almost all these debates, neither extreme position is defensible.

process is. As called out previously, you will need to modify and tune your process in real time as you discover what works for you and what doesn't.

Lastly, get on with it. Perfect is the enemy of good.[11] You will develop your process iteratively, just the way you develop the software. Get to Iteration 1 quickly. Learn. Change. Improve. Repeat until done.

Obstacle 3: Tools

Just as the first obstacle (politics) and the second obstacle (process) are intimately related, so are the second and the third. The third, of course, is the toolset that you will use to implement the process. Needless to say, choosing the tools first is getting it bass-ackwards, but surprisingly enough, that's the way many organizations go about it. They then wind up with the tool determining the process, which can be loads of fun when the process thus derived is inconsistent with the political philosophy of the organization.

Obviously, you need tools that will automate and enforce the process you have chosen to use. If you have a process that admits mistakes, you will be "backing out" changes from time to time. Does the tool support that easily? Are developers going to be checking in their work to a common baseline from multiple remote sites? If so, then your tool had better support that model. Do you want to build your entire product from top to bottom every night? If so, then I hope your tool has the performance and turnaround characteristics that will permit that. Do you want to automate your regression testing as part of the build? Once again, tool support is crucial.

Even organizations that have done a good job with the first two problems sometimes flounder with the third. And sometimes it is not the tools' fault, either. Once again, using our factory analogy, you need someone to monitor the line and to do quality control for the product coming off the line. Without constant vigilance, it is easy to automate a process that produces a low-quality result. Every successful build process requires a foreman or the equivalent thereof; sometimes he or she is called the *czar* (or *czarina*) *of the build* or, more simply, the *buildmeister*.[12] The buildmeister monitors the health of the line and makes sure that a steady stream of good-quality product is produced.

[11] I first heard this from Mikhail Drabkin of Riga, Latvia, and assumed it was Russian folk wisdom. He may have in fact been (mis?)quoting Soviet Admiral Sergei Georgievich Gorshkov, although that citation turns out to be only an approximation: Gorshkov's ended with "Good enough." Dr. Stephen Franklin of U.C. Irvine points out that very similar sentiments have been attributed to both Clauswitz and Voltaire. Clauswitz is much more verbose, and there seems to be evidence that Voltaire "borrowed" from an Italian proverb. Though these pithy sayings sometimes have a provenance that is difficult to pin down, one can deny neither their truth nor their wisdom.

[12] Here's a cautionary, funny, and politically incorrect tale: I once made a big deal about having a czar of the build, and then appointed a fellow who was, shall we say, altitudinally challenged. He unfortunately became known as the czardine of the build. Ouch!

One last semi-technical note: Beware of the old saying, "We can always write a script that can do that." The problem is that these scripts always start out small and simple and then grow in ways that are random and unsupervised. Scripts, unlike programs, are rarely designed; they just grow. They become compendia of special cases and are inadequate to respond to the ever-increasing demands of the organization; they are brittle. They are a maintenance nightmare, especially if the original author moves on. And they are very, very difficult to debug. Just as the road to Hell is paved with good intentions, the road to "build Hell" is paved with the out-of-control products of general-purpose scripting languages.[13]

What About Iterative Development?

Iterative development sidesteps one of the great dangers of the waterfall approach: leaving system integration to the last minute. One of the reasons so many waterfall projects fail is that, very late in the game, developers are trying to assemble their product for the very first time. In addition to finding many bugs, mostly in the interfaces, they grapple with the normal logistical and organizational problems of putting together a build chain for the first time. Often, things that pass for bugs are nothing more than the artifacts of broken builds. But the organization is in such chaos at this point—running out of time, nothing working, people frazzled—that it is hard to separate the sugar from the salt. It is also a very bad time to be trying to solve political and process problems.

By contrast, iterative development requires that you construct your build chain to accomplish the deliverable for Iteration 1—a working program. So you begin to debug this process early in the project, not at the end. By the time you get to Iteration 3 or 4, the build process is actually starting to work pretty well. For the last iteration, the one that will deliver the final bits, the build should be working like a finely lubricated Swiss watch.

As with pretty much everything else in software development, there are a small number of ways to get this right and almost an infinite number of ways to get it wrong. If you view the build as a detail that will "just happen," then the odds are against you. Make sure you attack the build process as a conscious effort that is critical to your success, and devote the time, energy, and resources to it that it demands. To do any less is sheer folly.

[13] Some scripting languages have been favorably likened to duct tape; would you want to begin the final assembly of your jet aircraft by having some guy yell out, "Time to get out the duct tape!"?

Recap

I've frequently been called in late in the software development cycle when projects are in trouble. Often, it is difficult at first to judge the depth of the yogurt. Usually the developers are focusing on how much they are "behind," as measured in things that are coded but just don't work, and things that are not coded at all but should be.

While this is one important aspect, I always begin to also look at the health of the build process. If the build process is non-existent or badly broken, it needs attention *immediately*. The reason for this should be obvious: at this stage of the proceedings, the lack of a reliable, repeatable build process will impede all further progress. You can't test what you can't build, and repeated testing is a necessity at this point; else, how can the developers know what they've fixed and what remains problematic?

Sadly, in many organizations the build process is something that is relegated to the B Team. This is a huge mistake. You must have A-list players in this part of the organization. As soon as people understand how much you as the senior manager appreciate the contribution of this team, you will have no problems getting volunteers.

The other thing that proves very slippery is answering the question, "How do we know when we are done?" Getting agreement well in advance on some clear criteria is incredibly useful; such agreement reduces the odds that the bar will move radically up or down as the ship date looms. One of the major objectives when moving into the Transition Phase of iterative development is to define reasonable criteria for shipment. Without a clear "exit plan," the project risks a series of never-ending last-minute slips.

Thus ends Part 2 on the basics. In Part 3, I will look at software development from a project-management perspective.

This page intentionally left blank

Index