Introduction

So, you are responsible for managing a computer hardware or software test project? Congratulations! Maybe you've just moved up from test engineering or moved over from another part of the development team, or maybe you've been doing test projects for a while. Whether you are a test manager, a development manager, a technical or project leader, or an individual contributor with some level of responsibility for your company's test and quality assurance program, you're probably looking for some ideas on how to manage the unique beast that is a test project.

This book can help you. The first edition, published in 1999, sold over 10,000 copies in its first two years on the shelf. I've received dozens of emails from readers, some wanting to ask me a question or two, some thanking me for writing the book, and some pointing out errors. As I started work on this second edition in 2001, I received an email from a reader in China who had read the book in its Mandarin-language translation. People have written reviews—mostly positive, but with suggestions for improvement, too—in various publications and Web sites. I am pleased with the reception this book has received, and thank all of you who read the first edition, especially those who have given me ideas on how to make this second edition better.

This book contains what I wish I had known when I moved from programming and system administration to test management. It shows you how to develop some essential tools and apply them to your test project, and offers techniques that can help you get and use the resources you need to succeed. If you master the basic tools, apply the techniques to manage your resources, and give each area just the right amount of attention, you can survive managing a test project. You'll probably even do a good job, which might make you a test project manager for life, like me.

The Focus of This Book

I've written *Managing the Testing Process* for several reasons. First, in what some have termed the "software crisis," many projects suffer from a gap between expectations and reality when it comes to delivery dates, budgets, and quality, especially between the individual contributors creating and testing the software, the senior project managers, and the users and the customers. Similarly, computer hardware development projects often miss key schedule and quality milestones. Effective testing and clear communication of results as an integrated part of a project risk-management strategy can help.

Second, I perceived a gap in the literature on software and hardware testing. I have read books targeting the low-level issues of how to design and implement test cases, as well as books telling sophisticated project managers how to move their products to an advanced level of quality using concepts and tools such as the Capability Maturity Model, ISO 9000, Total Quality Management, software quality metrics, and so forth. However, I believe that test managers like us need a book that addresses the basic tools and techniques, the bricks and mortar, of test project management.

The tips and tools offered in this book will help you plan, build, and execute a structured test operation. As opposed to the all-too-common ad hoc and reactive test project, a structured test operation is planned, repeatable, and documented, but preserves creativity and flexibility in all the right places. What you learn here will allow you to develop models for understanding the meaning of the myriad data points generated by testing so that you can effectively manage what is often a confusing, chaotic, and change-ridden area of a software or hardware development project. This book also shows you how to build an effective and efficient test organization.

To that end, I've chosen to focus on topics unique to test management in the development and maintenance environments. Because they're well covered in other books, I do not address two related topics:

- **Basic project management tools such as work-breakdown-structures, Gantt charts, status reporting, and people management skills.** As you move into management, these tools will need to be part of your repertoire, so I encourage you to search out project management books—such as the ones listed in the bibliography—to help you learn them. A number of excellent training courses and curricula currently exist for project management as well.
- **Computer hardware *production* testing.** If your purview includes this type of testing, I recommend books by W. Edwards Deming, Kaoru Ishikawa, and J. M. Juran as excellent resources on statistical quality control, as well as Patrick O'Connor's book on reliability engineering; see the bibliography for details on these and other books referenced here.

Software production, in the sense of copying unchanging "golden code" to distribution media, requires no testing. However, both hardware and software production often include minor revisions and maintenance releases. You can use the techniques described in this book to manage the smaller test projects involved in such releases.

The differences between testing software and hardware are well documented, which might make it appear, at first glance, that this book is headed in two directions. I have found, however, that the differences between these two areas of testing are less important from the perspective of *test project management* than they are from the perspective of *test techniques*. This makes sense: hardware tests software, and software tests hardware. Thus, you can use similar techniques to manage test efforts for both hardware and software development projects.

Canon or Cookbook?

When I first started working as a test engineer and test project manager, I was a testing ignoramus. While ignorance is resolvable through education, some of that education is in the school of hard knocks. Ignorance can lead to unawareness that the light you see at the end of the tunnel is actually an oncoming train. "How hard could it be?" I thought. "Testing is just a matter of figuring out what could go wrong, and trying it."

As I soon discovered, however, the flaws in that line of reasoning lie in three key points:

- The tasks involved in "figuring out what could go wrong, and trying it"—that is, in designing good test cases—are quite hard indeed. Good books have been written on the art of test case engineering, many in the last decade. Unfortunately, my professors didn't teach about testing, even though Boris Beizer, Bill Hetzel, Cem Kaner, and Glenford Myers had all published on the topic prior to or during my college career. As the second half-century of software engineering begins, that is finally beginning to change. However, the level of exposure to testing that most software-engineers-in-the-making receive remains too low.
- Testing does not go on in a vacuum. Rather, it is part of an overall project—and thus testing must respond to real project needs, not to the whims of hackers playing around to see what they can break. In short, test projects require test project management.
- The prevalence of the "how hard can testing be" mindset only serves to amplify the difficulties that testing professionals face. Once we've learned through painful experience exactly how hard testing can be, it sometimes feels as if we are doomed—like a cross between Sisyphus and Dilbert—to explain, over and over, on project after project, why this testing stuff takes so long and costs so much money.

Implicit in these points are several complicating factors. One of the most important is that the level of maturity of an organization's test processes can vary considerably: testing can be part of a repeatable, measured process, or an ad hoc afterthought to a chaotic project. In addition, the motivating factors—the reasons why management bothers to test—can differ in both focus and intensity. Managers motivated by fear of repeating a recent failed project see testing differently than managers who want to produce the best possible product, and both motivations differ from those of people who organize test efforts out of obligation but assign them little importance. Finally, testing is tightly connected to the rest of the project, so the test manager is often subject to a variety of outside influences. These influences are not always benign when scope and schedule changes ripple through the project.

These factors make it difficult to develop a "how to" guide for planning and executing a test project. As academics might say, test project management does not lend itself to the easy development of a canon. "Understand the following ideas and you can understand this field" is not a statement that can be applied to the field of testing. And the development of a testing canon is certainly not an undertaking I'll tackle in this book.

Do you need a canon to manage test projects properly? I think not. Instead, consider this analogy: I am a competent and versatile cook, an amateur chef. I will never appear in the ranks of world-renowned chefs, but I regularly serve passable dinners to my family. I have successfully prepared a number of multicourse Thanksgiving dinners, some in motel kitchenettes. I mastered producing an edible meal for a reasonable cost as a necessity while working my way through college. In doing so, I learned how to read recipes out of a cookbook, apply them to my immediate needs, juggle a few ingredients here and there, handle the timing issues that separate dinner from a sequence of snacks, and play it by ear.

An edible meal at a reasonable cost is a good analogy for what your management wants from your testing organization. This book, then, can serve as a test project manager's "cookbook," describing the basic tools you need and helping you assemble and blend the proper ingredients.

The Tools You Need

Five basic tools underlie my approach to test management:

- *A thorough test plan.* A detailed test plan is a crystal ball, allowing you to foresee and prevent potential crises. Such a plan addresses the issues of scope, quality risk management, test strategy, staffing, resources, hardware logistics, configuration management, scheduling, phases, major milestones and phase transitions, and budgeting.
- *Well-engineered testware.* Good testware (also known as a *test system*) ferrets out, with wicked effectiveness, the bugs that can hurt the product in the market or reduce its acceptance by in-house users. It also possesses internal and external consistency, is easy to learn and use, and builds on a set of well-behaved and compatible tools. I use the phrase "good testware architecture" to characterize such a system. The word *architecture* fosters a global, structured outlook on test development within the test team. It also conveys to management that creating good testware involves developing an artifact of elegant construction, with a certain degree of permanence.
- *A state-based bug tracking database*. In the course of testing, you and your intrepid test team will find lots of bugs, a.k.a. issues, defects, errors, problems, faults, and other less printable descriptions. Trying to keep all these bugs in your head or in a single document courts immediate disaster because you won't be able to communicate effectively within the test team, with programmers, with other development team peers, or with the project management team—and thus won't be able to contribute to increased product quality. You need a way to track each bug through a series of states on its way to closure. I'll show you how to set up and use an effective and simple database that accomplishes this purpose. This database can also summarize the bugs in informative charts that tell management about projected test completion, product stability, system turnaround times, troublesome subsystems, and root causes.
- *A comprehensive test tracking spreadsheet*. In addition to keeping track of bugs, you need to follow the status of each test case. Does the operating system crash when you use a particular piece of hardware? Does saving a file in a certain format take too long? Which release of the software or hardware failed an important test? A simple set of worksheets in a single spreadsheet can track the results of every single test case, giving you the detail you need to answer these kinds of questions. The detailed worksheets also roll up into summary worksheets that show you the big picture. What percentage of the test cases passed? How many test cases are blocked? How long do the test suites *really* take to run?
- *A simple change management database.* How many times have you wondered, "How did our schedule get so far out of whack?" Little discrepancies such as slips in

hardware or software delivery dates, missing features that block test cases, unavailable test resources, and other seemingly minor changes can hurt. When testing runs late, the whole project slips. You can't prevent test-delaying incidents, but you can keep track of them, which will allow you to bring delays to the attention of your management early and explain the problems effectively. This book presents a simple, efficient database that keeps the crisis of the moment from becoming your next nightmare.

This book shows you how to develop and apply these five basic tools to your test project, and how to get and use the resources you need to succeed. I've implemented them in the ubiquitous PC-based Microsoft Office suite: Excel, Word, Access, and Project. You can easily use other office-automation applications, as I haven't used any advanced features.

The Resources You Need

In keeping with our culinary analogy, you also need certain ingredients, or resources, to successfully produce a dish. In this testing cookbook, I show you how I assemble the resources I need to execute a testing project. These resources include some or all of the following:

- **A practical test lab**. A good test lab provides people—and computers—with a comfortable and safe place to work. This lab, far from being Quasimodo's hideout, needs many ways to communicate with the development team, the management, and the rest of the world. You must ensure that it's stocked with sufficient software and hardware to keep testers working efficiently, and you'll have to keep that software and hardware updated to the right release levels. Remembering that it is a *test* lab, you'll need to make it easy for engineers to keep track of key information about system configurations.
- **Test engineers and technicians**. You will need a team of hardworking, qualified people, arranged by projects, by skills, or by a little of both. Finding good test engineers can be harder than finding good development engineers. How do you distinguish the budding test genius from that one special person who will make your life as a manager a living nightmare of conflict, crises, and lost productivity? Sometimes the line between the two is finer than you might expect. And once you have built the team, your work really begins. How do you motivate the team to do a good job? How do you defuse the land mines that can destroy motivation?
- **Contractors and consultants**. As a test manager, you will probably use "outsiders," hired guns who work by the hour and then disappear when your project ends. I will help you classify the garden-variety high-tech temporary workers, understand what makes them tick, and resolve the emotional issues that surround them. When do you need a contractor? What do contractors care about? Should you try to keep the good ones? How do you recognize those times when you need a consultant?
- **External test labs and vendors**. In certain cases, it makes sense to do some of the testing outside the walls of your own test lab—for instance, when you are forced to handle spikes or surprises in test workloads. You might also save time and money by

leveraging the skills, infrastructure, and equipment offered by external resources. What can these labs and vendors really do for you? How can you use them to reduce the size of your test project without creating dangerous coverage gaps? How do you map their processes and results onto yours?

Of course, before you can work with any of these resources, you have to assemble them. As you might have learned already, management is never exactly thrilled at the prospect of spending lots of money on equipment to test stuff that "ought to work anyway." With that in mind, I've also included some advice about how to get the green light for the resources you really need.

On Context

I've used these tools and techniques to manage projects large and small. The concepts scale up and down easily, although on larger projects it might pay to implement some of the tools in a more automated fashion. In that case, the tools I've described here can be prototypes or serve as a source of requirements for the automated tools you buy or build.

The concepts also scale across distributed projects. I've used the tools to manage multiple projects simultaneously from a laptop computer in hotel rooms and airport lounges around the world. I've used these tools to test market-driven end-user systems and in-house information technology projects. While context does matter, I've found that adaptations of the concepts in this book apply across a broad range of settings.

Simple and effective, the tools comply with industry standards and bring you in line with the best test management practices and tools at leading software and hardware vendors. I use these tools to organize my thinking about my projects, to develop effective test plans and test suites, to execute the plans in dynamic high-technology development environments, and to track, analyze, and present the results to project managers. Likewise, my suggestions on test resource management come from successes and failures at various employers and clients.

Because context matters, I've added a new chapter at the end of this edition that discusses the importance of fitting the testing process into the overall development or maintenance process. This involves addressing issues such as organizational context, the economic aspects of and justifications for testing, lifecycles and methodologies for system development, and process maturity models.

Using This Book

Nothing in this book is based on Scientific Truth, double-blind studies, academic research, or even flashes of brilliance. It is merely about what has worked—and continues to work—for me on the dozens of test projects I have managed. You might choose to apply these approaches "as is," or you might choose to modify them. You might find all or only some of my approaches useful.

Along similar lines, this is not a book on the state of the art in test techniques, test theory, or the development process. This is a book on test management, both hardware and software, as I have practiced it. In terms of development processes—best practices or your company's

practices—the only assumption I make is that you as the test manager became involved in the development project with sufficient lead time to do the necessary test development. The last chapter addresses different development processes I have seen and worked within. I cover how the choice of a development lifecycle affects testing.

Of course, I can't talk about test management without talking about test techniques, to some extent. Because hardware and software test techniques differ, you might find some of the terms I use unclear or contradictory to your usage of them. I have included a glossary to help you decipher the hardware examples if you're a software tester, and vice versa. Finally, the test manager is usually both a technical leader and a manager, so make sure you understand and use best practices, especially in the way of test techniques, for your particular type of testing. The appendix includes a listing of books that can help you brush up on these topics if needed.

This book is drawn from my experiences, good and bad. The bad experiences—which I use sparingly—are meant to help you avoid some of my mistakes. I try to keep the discussion light and anecdotal; the theory behind what I've written, where any exists, is available in books listed in the bibliography.

I find that I learn best from examples, so I have included lots of them. Because the tools I describe work for both hardware and software testing, I base many examples on one of these two hypothetical projects:

- Most software examples involve the development of a Java-based word processing package named SpeedyWriter, being written by Software Cafeteria, Inc. SpeedyWriter has all the usual capabilities of a full-featured word processor, plus network file locking, Web integration, and public-key encryption. SpeedyWriter includes various JavaBeans from other vendors.
- Most hardware examples refer to the development of a server named DataRocket, under development by Winged Bytes, LLP. DataRocket provides powerful, high-capacity file and application services as well as Web hosting to LAN clients. It runs multiple operating systems. Along with third-party software, Winged Bytes plans to integrate a U.S.-made LAN card and a Taiwanese SCSI controller.

As for the tools discussed in this book, you can find examples of these at www.rexblackconsulting.com/pages/publications. I have spent a lot of time improving these templates for this second edition. I also have added some case studies from real projects. In those chapters that describe the use of these tools, I include a short section guiding you in the use and study of these templates and case studies should you want to do so. That way, you can use these resources to bootstrap your own implementation of the tools. These tools are partially shown in figures in the chapters in which I describe them, and I have improved the readability of these figures for this edition. However, screen shots of worksheets and forms can only tell you so much. Therefore, as you read the various chapters, you might want to open and check out the corresponding case studies and templates from the Web site to gain a deeper understanding of how the tools work.

Please note that the tools supplied with the book are usable, but contain only small amounts of "dummy data." This data should not be used to derive any rules of thumb about bug counts, defect density, predominant quality risks, or any other metric to be applied to other projects. I developed the tools primarily to illustrate ideas, so some of the sophisticated automation that you would expect in a commercial product won't be there. If you intend to use these tools in your project, allocate sufficient time and effort to adapt and enhance them for your context.

For those wanting to practice with the tools before putting them into use on a real project, I have included exercises at the end of each chapter. These additions, new to this edition, also make this book suitable as the test management textbook for a course on testing, software engineering, or software project management. (Solutions to these exercises are found on the Web at www.rexblackconsulting.com/pages/publications/mtp_solutions.zip.) Given that testing is increasingly seen by enlightened project managers as a key part of the project's risk management strategy, including material such as this as part of a college or certification curriculum makes good sense.

Finally—in case you haven't discovered this yet—testing is not a fiefdom in which one's cup overfloweth with resources and time. I have found that it's critical to focus on testing what project managers really value. Too often in the past I've ended up wrong-footed by events, spending time handling trivialities or minutiae while important matters escaped my attention. Those experiences taught me to recognize and attend to the significant few and ignore the trivial many. The tools and techniques presented here can help you do the same. A sizeable number of computer test organizations are disbanded in their first couple years; this book will help keep you out of that unfortunate club.

Although it's clearly more than simply hanging onto a job, success in test management means different things to different people. In my day-to-day work, I measure the benefits of success by the peace of mind, the reduction in stress, and the enhanced professional image that come from actually managing the testing areas in my purview rather than reacting to the endless sequence of crises that ensue in ad hoc environments. I hope that these tools and ideas will contribute to your success as a testing professional.

What's New and Changed in the Second Edition?

For those of you who read the first edition and are wondering whether to buy this second edition, I've included the following synopsis of changes and additions:

- I've added a new final chapter that discusses the importance of fitting the testing process into the overall development or maintenance process. I address organizational context, the economic aspects of and justifications for testing, lifecycles and methodologies for system development, and process maturity models.
- I have added case studies for many of the tools and techniques discussed in the book. These case studies came from real projects, which, happily, are not encumbered by nondisclosure agreements. I have included some background information to help you understand these case study documents.

- I have increased the number of templates and examples, and improved the format of some tools. I also added some new metrics. The templates include the tools to generate those metrics. Some of the templates originally published with the book, while usable, contained minor errors. (Readers of the first edition—being test professionals—caught and pointed out these errors to me.) I have corrected those mistakes.
- Some of the figures in the first edition were crowded into small spaces and had tiny fonts. I've revised and expanded the figures for better readability. I've also corrected some figures that became misplaced during the final edit.
- In addition to case studies, I have also added some exercises, some originally used in my three-day "Managing the Testing Process" class, and some created specifically for this second edition. These exercises can be used for self-study, book club, or classroom education. (Professor Patricia McQuaid selected the first edition as the textbook for a software testing course at California Polytechnic State University.) I have worked with some respected members of the software engineering academic community to ensure the usefulness of these exercises.
- The templates, case studies, exercise support files, and other tools, rather than being supplied on a breakable, hard-to-update CD-ROM, are now posted on the Web at www.rexblackconsulting.com/pages/publications. I will update and correct those templates as time goes on, so this book will, in a sense, be continuously improving.
- Finally, little has changed in terms of the challenges facing those who manage test projects since I wrote the first edition. Every time I teach my two- and three-day "Managing the Testing Process" classes, which are drawn directly from this book, at least one attendee tells me, "It's amazing how every issue we've talked about here in class is something that has come up for me on my projects." However, I have learned some new tricks and broadened my mind. For example, I criticized exploratory testing in the first edition, but I have learned through talking with successful practitioners as well as trying it out myself that, carefully managed, exploratory testing is a useful technique.

If you read the first edition, enjoyed it, and found it useful, I think these changes and additions will make this second edition even more useful to you.