PART

I

Goals, Strategies, and Tactics

In this Part

CHAPTER 1

What Does It Mean to Be Pragmatic?

Let's start at the beginning by exploring some obvious questions with some not-so-obvious and not-so-universally-accepted answers about pragmatic testing. From a pragmatic, or practical, standpoint, it involves being effective and efficient when testing software. What is effective software testing? What is efficient software testing? What is software testing, anyway? What is quality?

While these might seem like impractical, philosophical questions, in my experience, they are not. Your answers to these questions determine what *you expect to do* as a tester. Other people's answers to these questions determine what *they expect you to do* as a tester. Having common expectations up, down, and across the organizational chart and throughout the project team is essential to success. Without such commonality, no matter what you do, someone's sure to be disappointed. With common expectations, you can all strive for the same goals, and support others in their endeavors.

What Do Effective and Efficient Mean?

Webster's dictionary defines the word *effective* as "producing a decided, decisive, or desired result; impressive." So, to be an effective software tester, you must decide what results you desire from your testing efforts.

Likewise, Webster's defines *efficient* as "productive of the desired effect; especially to be productive without waste." So, to be an efficient tester, you must allocate resources (time and money) appropriately.

In the next few pages, you'll get a closer look at each of these concepts.

First, though, note that testing usually occurs within the context of some larger project or effort. That larger project could be developing a new system, maintaining an existing system, deploying a system in a new or existing environment, or deciding whether to accept or reject a proposed system. So, when the testing subproject exists within the context of the larger project, you should look at test effectiveness and efficiency from the project perspective, not the test subproject perspective.

What Effects Do You Want?

On most projects I've worked on, I've typically wanted to do the following:

- Produce and deliver information to the project team about the most important aspects of the quality of the system under test.
- Produce and deliver information to the development team so they can fix the most important bugs.
- Produce and deliver information to the project management team so they can understand issues such as the current levels of system quality, the trends of those levels, and the level of quality risk reduction being achieved.
- Produce and deliver information that can help the technical support team, help desk, and other folks on the receiving end of the project deal effectively with the system they received, warts and all.

In short, I often find myself a producer and provider of useful information for the project team. This has some interesting implications, which you'll examine in a minute.

Reading this list, you might have thought of some effects that you — or other people with whom you've worked — typically want from testing that are missing. Good! There is no one right set of answers for this question.

What Is the Right Level of Efficiency?

The definition of efficiency that was presented earlier included the ideas of productivity and a lack of waste. What does that mean?

Avoiding Redundancy

Of course, you don't want to waste time, so you should try to avoid redundancy. You should try to run the right tests and run them as rapidly as possible. You should try to discover bugs as early as possible and find the more-important bugs before the less-important bugs.

That said, you've probably seen more than one project where the old cliché "more haste, less speed" applied. You rushed into things without planning and preparation. You didn't attend to details. You abandoned carefully thought-out strategies at the first sign of trouble or delay. In short, in an attempt to save time, you made mistakes that cost you more time than you saved.

Reducing Cost

You also don't want to waste money, so you should buy only the supplies — software and hardware tools, test environments, and so on — that you really need and that you have the time and talents to gainfully employ. You should carefully evaluate tools before you buy them and be ready to justify them with cost-benefit analyses.

When you do such analyses, you'll often find that good tools and test configurations can pay for themselves — sometimes dramatically. On one project, a manager decided to "save money" by having us use a test server that was in a different country. This server cost about $20,000, and he said there was no money in the budget for another one. However, they paid more than $20,000 in lost time and productivity when people had to wait for the system administrator in the remote location to respond to their calls for help — sometimes for more than a day.

This matter of cost-benefit analyses brings us to another question: What is the cost-benefit analysis for testing? Is testing an investment that pays for itself?

While these questions are more a test management topic, let me briefly mention the concept of *cost of quality*. Cost of quality recognizes two types of quality-related costs that organizations pay:

- Costs of conformance: These are the costs you pay to achieve quality. They include *costs of detection*, which are the costs of testing, or looking for bugs. Conformance costs also include the *costs of prevention,* which are the costs of quality assurance, of improving the people, technologies, and process.
- Costs of nonconformance: These are the costs you pay when you fail to achieve quality. These costs include the costs associated with finding and fixing bugs, plus retesting those bugs and doing regression testing, before you release, deploy, or go live with the system. These are also called the costs of *internal failure*. The costs associated with finding bugs after that point are called costs of *external failure*. External failures can cost you in terms of finding, fixing, retesting, and regression testing, as well as the additional associated costs of support or help desk time, maintenance or patch releases, angry customers, lost business, and damage to reputations.

Usually, costs of conformance are less than the costs of nonconformance and costs of internal failure are less than costs of external failure. So, testing can have a positive return on investment by reducing the total cost of quality.[1]

What Software Testing Isn't...But Is Often Thought to Be

So far, I've talked about effectiveness and efficiency and what those concepts mean in terms of software testing, but what exactly is software testing?

Let's start with what it's not: Software testing is not about proving conclusively that the software is free from any defects, or even about discovering all the defects. Such a mission for a test team is truly impossible to achieve.

Why? Consider any system that solves complex business, scientific, and military problems. There are typically an infinite or practically infinite number of sequences in which the programming language statements that make up such a system could be executed. Furthermore, large amounts of tremendously diverse kinds of data often flow not only within each subsystem but also across subsystems, with features influencing other features, data stored earlier being retrieved again later, and so forth. Finally, users will use systems in unintended, often impossible-to-anticipate fashions, eventually even on system configurations that did not exist when the software was being written.

As if this were not enough, the complex flows of control and data through your systems create intricate dependencies and interactions. Careful system designers can reduce coupling, but some of these flows arise due to essential complexity. So, minor changes can have significant effects, just as a pebble can produce rings of ripples across a whole pond or, perhaps more aptly, just as the lightest touch of a fly's wing vibrates the whole spider web.

Perhaps you're thinking, "Okay, Rex, I get it, let's move on." The problem isn't that we testers seldom understand the impossibility of complete, exhaustive testing that proves conclusively that the system works or reveals every single bug. The problem is that many managers and even some developers *do think* we can completely, exhaustively test software.

I have heard managers say things like, "Oh, how hard can it be? Anyone can test. Just make sure everything works before we ship it."

I have heard developers say things like, "Of course my units work fine. I ran unit tests and tested everything that could break."

Maybe you've heard statements like this too.

I used to ignore those statements or dismiss them as outsiders' blissful ignorance of the difficulties of my job. I found that testers ignore such statements only at their peril. Statements such as these indicate a fundamental misunderstanding of what testing can accomplish. That means people have expectations of you, the test professional, that you cannot possibly meet. When you don't

[1] For more on the important management issue of testing economics and return on investment, see my other two books, *Managing the Testing Process* and *Critical Testing Processes*.

meet those expectations, people might conclude that you're incompetent. Therefore, one of the first steps to testing success is the alignment of realistic expectations about testing across the project team.

NOTE **One of the reviewers of this book, Judy McKay, mentioned, "We have to be sure that we don't compound this misperception that testing can prove the absence of defects or can discover all defects by publishing testing metrics with statements like '100 percent test coverage [achieved].' Just because we ran 100 percent of our test cases doesn't mean that we've covered 100 percent of what the code could do. We need to be sure our information is accurate, supportable, and understandable."**

The following sections investigate what testing is, and isn't, as well as what it can realistically do for you.

Five Phases of a Tester's Mental Life

One of the three founders of modern software testing, Boris Beizer, identified five phases of a tester's mental life:

- **Phase 0:** There's no difference between testing and debugging. Other than in support of debugging, testing has no purpose.
- **Phase 1:** The purpose of testing is to show that the software works.
- **Phase 2:** The purpose of testing is to show that the software doesn't work.
- **Phase 3:** The purpose of testing is not to prove anything, but to reduce the perceived risk of the software not working to an acceptable value.
- **Phase 4:** Testing is not an act. It is a mental discipline that results in low-risk software without much testing effort.

The first two phases can be thought of as myths of individuals and organizations with immature testing practices and processes.[2]

This leads you to the practical testing reality in phase 2. Some testers stop here. In some organizations, that's fine. If all your peers and managers expect from you is that you find lots of important bugs, and find them quickly and cheaply, then this book can teach you how to do that.

However, you can also progress to phase 3. In this phase, testing is part of an overall strategy of risk management. Testing is risk focused. Testing produces risk-related information. This book can teach you how to do that too.

[2] See Boris Beizer's *Software Testing Techniques* for a more thorough description of the phases and his conclusions about them.

Phase 4 represents more of an organizational mindset than a tester's mindset. Once testing — and the implications of testing — become pervasive throughout the organization, everyone will start to act in a way that reduces bugs before a single test is run. [3]

So, what phase do you find yourself in right now, and why? In which phase do your peers and managers expect to find you, and why? In which phase does your organization need you to be, and why?

Other Founding Views on Testing

Another founder of the testing profession, Bill Hetzel, wrote, "Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results."[4]

This definition, while perhaps implying the dated concept of "proving that the system works," does include the useful concept of evaluating system attributes and capabilities. Indeed, tests *do* evaluate some facet of the system, under some set of conditions, and check system behavior against expected results. If you substitute the word *whether* — as in "determining *whether* [the system] meets its required results" — you can say Hetzel hit the nail on the head — such is the act of testing.

This leads to an obvious question, though: How do you know whether the system behaved as required? The collection of tools, techniques, and documents that testers use — together with the tester's judgment — to make this decision is referred to as the *test oracle*.The name comes from oracles in ancient Greek history (most famously the Oracle of Delphi), supposedly able to foretell the future. Calling the technique used to predict and/or check the expected result an oracle is a nice way of hinting at how hard this often is. This book revisits the challenges of test oracles in various chapters.

The first founder of the testing profession, Glenford Myers, wrote, "Testing is the process of executing a program with the intent of finding errors." Myers was onto something with his definition too. The mind tends to see what it expects to see. People who think they can achieve something often do. A good tester intends to and expects to find bugs because a good tester recognizes that all programs have bugs. [5]

Those of us who have carried on the work started by Myers, Hetzel, and Beizer have continued to refine their ideas on what software testing is all about. Mark Fewster and Dorothy Graham described testing as building confidence where the system under test works and locating bugs so that we can fix them. This definition bring us to the "why test?" question. We would want to

[3] This brings us to topics more properly related to test management again, so I'll refer you to my two books, *Managing the Testing Process* and *Critical Testing Processes*.

[4] See Bill Hetzel's *Complete Guide to Software Testing*.

[5] See Glenford Myers's *The Art of Software Testing*.

ship a system only when we were confident that the important stuff worked. We'd need to fix as many bugs as necessary — and then some, I'd hope — to get to that point.[6]

Testing as a Form of Quality Risk Management

So, now I've cited some people whose thinking on testing has influenced mine. Through my study of risk and risk management, I've come to think about system development, maintenance, deployment, and acceptance projects as dealing with four main variables:

- **Features:** Will the system provide the right capabilities?
- **Schedule:** Will you deploy or release the system soon enough?
- **Budget:** Will the overall effort or project make financial sense?
- **Quality:** Will the system you create satisfy the users, customers, and other stakeholders?

Because the answers to these questions are not clear in advance — that is, there are risks that might lead to answers you wouldn't like — you need some way to manage those risks. One or more risks in any of these areas can, if unmanaged, lead to undesirable outcomes that sink the project. Any attempt to reduce the risk in one category always affects the other categories, usually by increasing risk in one or more of them. If the project team properly manages and balances these risks, the project has a strong chance of succeeding.

So, as a test professional, you should give the management team insight into the fourth risk category, quality. You do that by producing and delivering assessments of quality and other quality-related information in terms management can understand and act upon.

Sometimes, that information leads to actions to improve the quality of the system. The management prioritizes the bugs you've found. The development team fixes the high-priority bugs. Sometimes, though, that information can lead to actions that trade risks in one category for risks in other categories. For example, if you can't get a feature bug free enough to be good enough for release, you might push that feature into a subsequent release.

So What Is the Test Team All About?

How do you summarize the purpose of the test team? Here's a generic statement that you can tailor for any of the organizations you may work in and with:

> *To effectively and efficiently provide timely, accurate, and useful testing services and quality information that helps the project team manage the risks to system quality.*

[6] See Mark Fewster and Dorothy Graham's *Software Test Automation*.

This statement is very similar to what a newspaper publisher wants to achieve. A publisher wants their newspaper to provide timely, accurate, and useful information to their readers.

As with newspaper articles and newspapers themselves, credibility is key for testers. This book focuses on effective ways to produce timely, accurate, and useful information. However, you must always remind yourself that how you produce the information is only the first half of the battle. You also must credibly provide that information. So effectiveness of communication is important, and details and presentation matter.[7]

What Does "Quality" Mean to You?

If you want to assess quality, communicate about levels of quality, and help management understand and manage quality, then you need a working definition for quality.

There are a number of different definitions for quality out there. I happen to like J. M. Juran's definition:

> *Fitness for use. Features [that] are decisive as to product performance and as to "product satisfaction"…. The word 'quality' also refers to freedom from deficiencies…[that] result in complaints, claims, returns, rework and other damage. Those collectively are forms of "product dissatisfaction."*[8]

To some extent, Juran's wording is a bit clunky and oblique. Fitness for use *by whom*? Who is being satisfied or dissatisfied here? Project sponsors, customers, users, and other stakeholders. Juran's definition of quality centers on the needs and demands, the desires and the preferences of the people who pay for, use, and are affected by our systems.

Contrast this with Phil Crosby's definition of quality. Crosby defined quality as conformance to requirements *as specified*, nothing more, nothing less. The problem for software testers is that they often receive flawed requirements specifications — or none at all.[9]

I suggest that, on most projects, testers should be free to call a behavior a "bug" when they suspect that, due to that behavior, the system might dissatisfy a sponsor, customer, user, or other stakeholder. I like Juran's definition because it makes it clear that we can do that, especially in the absence of clear requirements specifications. Even with them, testers, thinking skeptically, should recognize that those specifications are as likely to be wrong as the system implementation itself.

---

[7] See *Critical Testing Processes* for a detailed description of the issues relating to test results reporting.

[8] See J. M. Juran's book *Juran on Planning for Quality*.

[9] See Phil Crosby's book *Quality Is Free*.

Now certainly there are cases, such as contract development, where requirements specifications draw a contractual distinction between an enhancement request and a bug. However, for most IT and mass-market software, does the average project sponsor, user, or customer care about such distinctions? No. They simply want their software to show a preponderance of satisfying behaviors and a relative absence of dissatisfying behaviors. That's quality to them. Test professionals need to know how to assess quality. This book shows you the places to look for bugs and the tools to root bugs out.