This article is a culmination of observations and recommendations for software engineering and software testing professionals. Some of the material won't surprise the casual reader at all; other material might be considered somewhat provocative. It is based on years of experience by two managers at IBM and is offered with the hope that it will provide a catalyst for readers to think through their approaches to software development, particularly software testing.

Key words: culture, customer involvement, integrity, metrics, quality, test execution, test planning, test process, test schedules, test teams

SQP References

Efficient Test Planning and Tracking
vol. 5, issue 2
Wayne D. Woodruff and Ron Pisechko

Choosing a Tool to Automate Software Testing
vol. 2, issue 1
Mark Fewster and Dorothy Graham

SOFTWARE VERIFICATION AND VALIDATION

Lessons from Experience: Managing Software Test Teams

TED RIVERA AND SCOTT WILL
IBM Software Group

INTRODUCTION

In the authors' combined experience through 40-plus years with IBM, they have been lead programmers, headed worldwide customer support organizations, been responsible for technical quality engineering for entire divisions, and managed both individual test teams and entire test organizations. They have watched as software testing has moved from being mainly an afterthought in the industry to becoming an integral part of most software development teams. What follows is an eclectic mix of their thoughts, ideas, and even some "battle scars" that they have accumulated over the years – with the hopes that the reader will be encouraged to "soldier on," laying the groundwork for the next generation.

As a business paradigm, Sun Tzu's *The Art of War* (Sun Tzu 1993) is unnecessarily contentious. In software testing, however, the analogy is a fit one. The very nature of the enterprise is simple: Find problems in the work of others sometimes and publicize the ugly features of their babies. No matter how closely testers work with their development counterparts, although there may be some positive (and some negative) approaches to communicating to a developer that "Your baby looks like a chimpanzee," and even when using words sprinkled with lilac water, testers are certain to offend. Even the more inclusive, more positive attitude: "*Our* baby looks like a chimpanzee" is rarely met with enthusiasm.

Managing software teams is a challenging occupation and not for the feint of heart. If Willie Nelson had possessed the forethought

to enter the field of software engineering, he would have certainly composed the timeless musical classic, "Mama, don't let your babies grow up to be test managers." Accepting a position as the manager of a software test team smacks of lunacy and warrants psychological evaluation. Consider the playing field:

- When using traditional development methodologies a team will almost *never* get code when it is promised or needed in order to do appropriate testing on a humane timetable. And while agile methodologies have helped tremendously to improve stability for teams that have adopted them in a disciplined manner, where "agile" is used as a mask for chaos, sadness ensues.
- Intense schedule pressures will permanently dominate the landscape.
- Attracting and keeping talented engineers will be a perpetual challenge.
- One serious mistake can crush a team and ruin a hard-won reputation. "How could you have missed that?" and "Wasn't this ever tested?" will be the test team's equivalent of *The Scarlet Letter.*
- Scrutiny from others, including executives, will be frequent, and true support may be rare.
- Many will offer advice about how one could do a better job by working "smarter and not harder."
- The most successful test organizations will often be unrecognized and virtually invisible. The testers will be blended into the development team in a cooperative "whole team" approach. Others may minimize or fail to adequately understand one's contributions or those of the testers: "The developers write great code" will be the frequent conclusion.

Does this seem like a wise career choice? Or is it instead a guaranteed future of antacids and nitroglycerin tablets?

Despite this landscape, against all odds, the authors have loved managing innovative test teams. The challenges are real, but the results can be exhilarating. In the course of working with numerous test teams, the authors have formed many strong—some would say fanatic—opinions about a wide range of issues touching on test in software engineering organizations. And while only opinions are presented in this article, in virtually every instance, they are opinions borne out of the real experiences of the day-to-day management of test teams. As a result, since these are stated only as opinions, readers are free to dismiss them as the ravings of lunatics—but bear in mind that these are opinions that have been forged by real-world challenges.

PROCESS IN TEST TEAMS

More of the opinions the authors have formed about managing test organizations are in the area of process than in any other category. This is surprising. Given the industry literature, one would naturally assume that tools, skills, or something technological would dominate this list. But in fact, *everything falls apart in a software development project* if madness reigns, and this is especially true for software testing teams.

It is no secret that the word "process" is loathed in many software engineering organizations, and from the authors' perspective, this is because too often *bureaucracy* masquerades as *process*. As such, the authors begin their list of opinions with thoughts about process because process issues are the most important to deliberately address as a test manager. It should be noted that prominently featuring test process in this way is, in its own right, a strong opinion.

As a result, the authors have formed specific opinions about test process:

1) **Morale *improves* with appropriate process.** This opinion is counterintuitive, but it is true as long as rewards and recognition, as well as consistency across teams, are ensured. Steve McConnell, in his book *Professional Software Development* (McConnell 2004), makes this point at length, but the authors' affirmation of this point is based on their independent implementation of light process across many different teams and organizations. This is especially important within test teams, given the high scrutiny often faced just prior to product shipment. Without a consistent approach, morale can easily deteriorate as finger pointing and uncertainty dominate the landscape. Note that even chaos is actually a process, just not a very effective one. It must be explicitly stated, however, that process bureaucracy is not the aim. *Appropriate* process is the objective: the consistent, disciplined application of best practices and standards, coverage targets, performance targets, and so on.

2) **Test automation is not optional; however, it should be assessed for appropriate return on investment.** This may be illustrated best through the use of an example:
   - A test engineer has a manual test case she *expects* will need to be run once or maybe twice. In this instance, she has carefully documented the test case in a meaningful fashion, with sufficient detail for someone else to perform the test in the future, if needed.
   - A defect is discovered during the execution of the test case.
   - To verify that this is really a defect, the test is conducted again to ensure that it is genuine.
   - If the developer and tester are not working side by side, once the defect is determined to be genuine, at times the developer will be skeptical, or will at least want to see the defect firsthand, and so the test case is run yet again.
   - The defect is then repaired.
   - Now the test case will need to be run once again to ensure that it is successful.
   - When the next build of the code is run (preferably after the defect is fixed), or regression testing is needed at some other point, the test will likely need to be run again. Potentially, it may need to be run several times depending on the need for regression testing or system testing that might be appropriate.

With this scenario in mind, it becomes easy to justify automating this test case. Or does it?

For test automation planning, it might be helpful—as an example—to estimate automation time as a function of execution time, making the decision as to whether to automate easier. If it takes X units of time to execute a test case and 50X units of time to automate the same test case, then the test case should be expected to run at least 50 times in order to break even. Thus, in the previous scenario, automating the test case would *not* be warranted (with the caveat being the number of times the test case is executed during regression testing—but recall that original expectation was low).

3) ***Never* ship with *any* known high-severity defects.** Technical debt is a cruel mistress. If a company knows about defects now, their customers will certainly know about them soon enough. Exceptions to this rule should be extraordinarily rare and should require agreement from customer support and approval by executives. Never let exceptions become the rule.

4) ***At least* half of all low-severity defects identified before the beginning of product/system test should be fixed prior to product/system test exit.** The low-severity defects that should be fixed are those that customers will likely see initially or often. Refer to the "Red Zone Analysis" paper, which describes this approach in detail (Rivera, Tate, and Will 2004a). When using more agile methods (XP, Scrum, RUP, and so on), the same concept applies: A substantial number of less-severe defects can appear just as serious to end users as a few high-severity defects. Severity is a judgment made by the tester (at times through discussions with others) of the seriousness of the problem as related to the product function. It represents the impact this problem is having on the current test cycle. Sample severity criteria and their respective meanings can be seen in Table 1.

5) **The test organization should have the ultimate "stop-ship" decision.** Without agreement from the test team, whatever one is building should not be considered as finished. If the decision is made to release a product without test's agreement, it is akin to shipping cookie dough instead of cookies.

TABLE 1 Sample severity criteria and their meanings

Severity Codes	Meaning
Severity 1	An unavoidable error that makes a component or the product inoperative, including any error that requires a restart of the product or operating system.
Severity 2	Any abend or problem that severely disables a major function of a component or the component itself. A workaround may be available.
Severity 3	Any minor functional problem or technical inaccuracy in the documentation.
Severity 4	Misspellings, color problems, incorrectly formatted screens, bad grammar, and other nonfunctional problems.

6) **Ongoing performance testing is vital.** True performance testing often reveals the most expensive defects that are otherwise detected only by customers. Additionally, performance testers should have an opportunity to review and influence design decisions, pointing out likely performance bottlenecks and capacity limitations.

7) **Continuous, or at least daily, builds should be required.** An inability to perform regular, successful builds indicates an overall quality risk and should therefore be of concern to test teams. And notably, a lack of continuous integration impedes an organization's efforts to move to more agile methods of development.

8) **Finding defects is only part of the test equation; *fitness for use*, while an old notion, remains a potent consideration for test.** W. Edwards Deming is a ghost worth dealing with (Gabor 1990).

9) **Chronic overtime may occur, but must be balanced with time for recuperation.** Too often, test teams are abused. "Death marches" must not be the norm (Yourdon 1999).

10) **Iterative development has positive effects on identifying serious problems earlier and makes overall testing more effective.** Speaking from the vantage point of test, the authors are strong advocates for all disciplined approaches to agile, with iterations that are truly stable. Apart from purely technical benefits, communication invariably improves as well.

ISSUES THAT AFFECT PEOPLE ON TEST TEAMS

The second most significant area associated with consistently producing high-quality software relates to cultural elements. Once again, this is both surprising—and true.

It is therefore comparatively easy to take a step back and quickly assess a test organization: How reasonable and how consistent is its process? What kind of organizational and people dynamics are present? How successful is the organization at finding and attracting strong technical talent? Considering these and other factors will provide a significant indicator of how effectively that group will test the software.

More particularly, then, the authors have formed various opinions about people issues:

11) **Test teams should be equally senior to development teams.** How senior the testers are in comparison to the developers is a good indicator as to whether an organizational commitment is in place that supports the notion that test really is a career. This comparison is not just a comparison of teams based on their job titles, but a comparison of overall experience.

12) **The test organization should be neither a development farm team nor a dumping ground.** To reverse the trend, perhaps the development team should serve as a test farm team.

13) **Some of an organization's very best software engineers should work in test, especially since a significant portion of software development expenditures are related to test.** Consider:
   - Can bad engineers hire good ones?
   - What is the incentive for a mediocre engineer in test or development to recommend hiring someone better, especially in an atmosphere where performance appraisals encourage individual efforts instead of team efforts?

14) **Test teams should be active in writing papers, submitting patent disclosures, attending industry conferences (and even presenting when selected), and innovation in general.** Areas ripe for input are test techniques, product vulnerabilities, and test tools.

15) **If budgets need to be cut, test is too often a more common target than development.** If cuts are needed, they must be balanced across functions.

16) **The career development model espoused by McConnell is good.** A valuable contribution to the test community would be a detailed model built up around testing, analogous to the paradigm McConnell provides for developers (McConnell 2004).

17) **Dedicated development teams should necessarily require dedicated test teams, that is, it is wrong to move testers from product to product.** Otherwise, what are developers working on that requires no resources for test design, review, construction, or execution? In the unlikely situation that a case can be made for a developer

in a certain component to work in isolation, the tester of that component cannot. So where is the justification for shared testers and dedicated developers? Worse still, compelling testers to support more applications or products than their development counterparts virtually precludes an organizational move to agile—where constant communication and partnership is essential—along with its many advantages.

18) **When interviewing testers, the authors always include the following questions:**
   - What was the last software engineering book you read?
   - What languages can you program in?
   - Are you interested in a career in testing or just a job?
   - What customers have you worked with?

19) **Most testers should be capable programmers.** Programming skills are required for writing automated tests and, of course, for understanding the code.

BUSINESS ISSUES AND THEIR EFFECT ON TEST TEAMS

If it is too often true that test organizations are far removed from direct customer interaction, and most test organizations are disconnected from the business climate and business issues within which their products are used. The real danger in this is that it becomes impossible to express accurately the risks that will be taken when key defects remain in the software and a desire to ship the code is expressed.

As a result, the authors have formed various opinions about test organizations and business issues:

20) **Software testers are businessmen and businesswomen first, geeks and geekettes second.** Professional software testers very definitely need to be geeks, but salaries are paid because of the value that products deliver to customers. Without real insight into the value proposition of the product, and the ways in which the software is used, a test organization is ultimately less effective.

21) **In order to make intelligent release decisions, test teams should be well informed about the real business climate.** If developing an application that will only be used once by an organization, such as to comply with a new government regulation, software that is "good enough" and "on time" may be the perfect solution. By contrast, most software is shipped with a higher proportion of defects than is warranted. Without an understanding of the business, it is virtually impossible for test teams to articulate real risks.

22) **"Testing constitutes as much as half of the typical software development life cycle"** and that includes testing done by developers as well as testers (Beizer 1995). If this observation, dated as it is, is even close to accurate, this one thought will have a dramatic impact on the investment decisions made in software development organizations.

23) **Some proportion of the dollars (perhaps 30 percent to as much as 50 percent) invested in research in software engineering should be expended on issues relating to test,** since this is the proportion of spending that mirrors actual software development. Patents and papers associated with innovations in testing should therefore be common.

24) **A similarly significant proportion of literature about software engineering should be focused on testing.** If a significant proportion of the cost of developing software is related to testing, this only makes sense. But is this the case in the local bookstore?

25) **Mathematics are against us.** The testing permutation matrix includes:
   - Differing supported platforms (for example, operating systems)
   - Release middleware (for example, databases)
   - Earlier releases
   - Integrated products combined together into an overall solution
   - New functionality

   And so on.

26) **Quality is a feature.** Owing to the nature of their role, marketing teams tend to push for new features in products, often not realizing that solid product quality is a key selling feature.

27) **"We will be known as the producer of the highest-quality software in the industry." This is a notion worth seriously striving toward.** While this may not be the primary deciding factor in sales, surely it helps to be a best of breed product.

TEST INTERACTIONS WITH OTHERS

Customers

The authors have separately written extensively about engaging directly with customers in their test organizations. It is *the* most important (and, in practice, the most rare) type of interaction a test team can undertake.

Some key opinions:

28) **Customer residencies are essential** (where customers are on site and the customers and test teams jointly perform testing activities). These residencies should be planned and agreed to far in advance, because direct feedback from customers will likely require changes to the product. Additionally, ignoring feedback provided by on-site customers may damage the relationship with the customer.

29) **Emulating exact customer environments directly should almost never be undertaken, and even then, should only be committed to in the case of short-term, single-issue testing.** For the long term, only parts of customer environments should be approximated, and *transplant-testing* should be used as much as possible. Long-term emulation has not worked because it is: 1) too expensive, 2) continually changes, and 3) is useless to the customer without resources for reporting and analysis.

- If there is a need to directly emulate a customer environment, a good estimate for the resources required is half the size of the customer team responsible for managing the environment. This is expensive, but the only way to do this effectively.

30) **When uncertain of what severity to set on a defect, the number and criticality of testing scenarios, use cases, or user stories impacted should guide the decision.** Where possible, have customers set such severities with the testers. But remember to include larger considerations as well—with unlimited resources or isolated issues, customers will likely demand fixing all known defects.

31) **The cost of not working directly with customers is significantly higher than the cost of working with them.**

32) **It is sometimes right to be obstinate, as long as customer impact can be assessed or estimated. It is wrong to be obstinate on principle with no reference to production impact.**

33) **Implant testing is a notion that warrants further research and investigation.** The authors have written about implant testing elsewhere (Rivera, Tate, and Will 2003); in short, it is the notion whereby tools, techniques, and programs that are useful for test teams are "implanted" in customer shops, typically in their test environments. Similar to development partnerships, test or quality partnerships should be pursued with customers.

34) **Transplant testing is a necessary activity.** In short, transplant testing is any of a wide range of activities whereby one transplants customer artifacts from customer environments and uses them in his or her test beds.

Development

Whenever the authors speak to large groups of testers, the easiest common ground to walk on is to refer to the development team as "the dark side." There are great development teams and there are weak ones, but in practice, there is sadly too often contention between testers and developers. Too often, developers are seen to hold the intellectual ascendancy. Not surprisingly, the authors have opinions on this as well:

35) **If developers can help testers test, testers can help developers develop.** Either testing is a career, with truly specialized skills and areas of expertise, or it is not. Consider some questions that illustrate potential issues:

- How would short-term rotations that a developer might take as a tester relate to long-term insight?

- How does it help a developer to become an expert at finding defects in his or her officemate's code (for example, via pair programming)?
- What is the long-term perspective when one is thinking as a developer rather than a tester?
- Is it ever a reality in one's organization for developers to be reluctant participants in testing, to desire "finishing" more than "excellence"?
- Pair programming is not spoken of consistently as pairing developers with testers, but it is an approach that warrants consideration and practice. Notably, many, if not most, of the authors' best testers are also able to program.

36) **Since software architects generally come from the ranks of developers, there should be an early career incentive to focus on quality in the career path of developers.** Otherwise, it is difficult to introduce this emphasis later. As an aside, architects must keep their hands in the code, lest they become "PowerPoint Architects."

37) **Development teams should write their own automation and thereby contribute to the overall automation effort.** Some of the better developers use this rule of thumb: Write three lines of test automation code for every line of product code.

38) **In practice, test-driven development often reduces required testing.** Focusing on testing *before* writing code helps ensure that code will not only be testable, but thoroughly unit tested (Beck 2003). Importantly, test-driven development often helps ensure that only that which will satisfy the client need is actually coded, reducing the waste associated with the volume of function that is shipped but rarely or never used.

Others

People are by nature reductionists. It is difficult for people to hold complex problems in their minds; instead, once they have analyzed a problem to their satisfaction, they begin to think of it in more simple categories—they reduce the complexity.

Testing, when considered in isolation, is a comparatively easy problem. The difficulty is that testing is only a part of the larger fabric of software engineering. Testing, when done properly, is a part of a community. For example, a daily Scrum, as part of the agile Scrum approach, should be cross functional.

The authors hold the following opinions about some of these points of intersection with others:

39) **Having people from support, education, marketing, development, and elsewhere help test is a good thing—if done right.**

40) **All documentation must be tested.**

41) **All technical writers should use, and be able to demo, the product they are writing about.**

42) **Test and education should be linked** in at least one of the following ways:
- Education should fund testing to validate steps.
- Education should participate in testing to engage with the larger development effort and to provide their unique insights.

METRICS AND TEST TEAMS

It is important to mention that the authors are advocates of minimalist metrics for teams that employ a disciplined agile approach to development. Working features of high quality by iteration is the way we speak of the notion of *velocity*. Another metric in agile environments the authors consider key is the time between stable builds—longer periods of instability could make it impossible to complete a *story* in a day, for example. Happily, an increasing proportion of the teams the authors work with are employing disciplined agile approaches, but this is by no means universally the case. As a result, metrics abound around testing. The meaning behind those metrics, however, is sometimes lacking. As a result, the authors have formed various opinions about test organizations and metrics:

43) **Consistent metrics pertaining to the right things matters.** In addition, metrics definitely affect behavior—so choose wisely!

44) **The landscape of possible testing should be understood.** The question is not simply, "What tests will we be doing?" but "What tests are possible?" Test coverage should be clearly identified and scoped for effort. This is the only way to establish and communicate a baseline for confidence and risk analysis.

45) **Early trouble in test should indicate that overall quality is at risk.** Early warning signs—of which

significant trouble in test is but one—must be heeded. For example, if technical debt accrues across several iterations, severe problems will result in the end game.

46) **A high defect refix rate is an indicator of one of the following: poor coding standards, bad communication, and/or inconsistent test practices.**

47) **Why, if it is urgent that test provide extensive metrics, is it the case that other teams (for example, marketing and development) rarely or never provide their own set of metrics by which their progress and effectiveness is communicated?** However, this is rarely the case. When a marketing claim is made that "these features will drive $20 million of revenue," what accountability is there in place for such assertions?

GUIDELINES FOR IMPROVING TEST MANAGEMENT PRACTICES

We hope readers have been challenged in some of their thinking regarding software development, and especially software testing. This last section is offered as a reminder that anytime one decides to "cut against the grain," it can result in some pain. Following are some of the battle scars the authors accumulated over the years. Their experiences here can be leveraged by other teams to improve their test management practices:

48) **To be a good tester or test manager, one has to be willing to say "No" when everyone else is saying "Yes."** This means one has to be able to clearly communicate the expected impact in business terms for poor product quality. To put an especially sharp edge on this item, the authors tell test managers that they essentially have to be unafraid of being fired for saying "No" when everyone wants them to say "Yes."

49) **The true cost of rotating/sharing testers across projects is often higher than dedicating them.** Switching costs are real. Part-time assignment of testers will logically reduce product expertise, reduce effectiveness, and lower morale.

50) **First impressions matter.** Consider five primary stages of failure or assessment: install, migration, configuration, basic operation, and long-term operation. When a client's early experiences with installation or migration go well, their trust and confidence in the company and in its product, as well as their overall satisfaction, rises.

51) **It is a mistake to get used to how the software being tested works.** Naïve users of one's software will not appreciate it if one's test teams become accustomed to and complacent about the ugly warts of another's software.

52) **Even with the best efforts, organizations will still experience a serious defect that escapes to the field.** When this happens, there must be skin in the game for teams outside of test—all are responsible for high-quality software, and all are accountable when issues arise.

53) **Migration testing is rarely thorough enough** (testing the steps needed to move from one release to a newer release). Consider focusing one's transplant test efforts particularly on this problem.

54) **It *is* possible to test quality into the product.** This is by no means optimal, but it is possible. Some might argue that one can't build a fire by burning toothpicks, but by getting enough of them together, one certainly can. While this is an expensive way to build a fire, the analogy holds: You can test quality in, but it is wiser to build an organizational culture where quality is genuinely valued and a preeminent concern. The authors believe that disciplined approaches to agile help teams succeed more consistently, owing to the relative ease with which shorter iterations of stable function can be managed.

55) **Issues relating to product shipment touch on software engineering ethics and integrity.** The IEEE/ACM software engineering code of ethics offers a compelling starting point (IEEE/ACM 1999). "Business decisions" should be clearly identified, not swept under the rug since they can compromise the integrity of individuals involved.

56) **There *are* such things as best practices, and they should be constantly evaluated and new aspects implemented** in addition to the following:

- Red zone analysis (Rivera, Tate, and Will 2004a)
- Understanding "objective quality" and "subjective quality" (both are important). Objective quality is that which can be measured (defect severities over time, defect backlogs, and

so on). Subjective quality deals more with user-perceptions—things like product "fit and finish," ease of use, and intuitiveness. The authors are advocates for outside-in development as providing insight on subjective aspects of quality (Kessler and Sweitzer 2007).

- Static code analysis

57) **Fatigue—and errors in decision making—are most common in the end game.** Also, that window presents the greatest temptation to add new people, when it is hardest and most expensive to train them. Deliberate effort must be undertaken to ensure that, even when the team is tired, the right decisions are made, rather than the expedient ones.

A FEW THOUGHTS FOR THE FUTURE

In the spirit of Vannevar Bush's *As We May Think* (Bush 1945), the authors offer some thoughts that arise from their experiences in managing software test efforts that may be in the offing in the future. Agile software development approaches have crossed the chasm and are in many areas the norm. Consider some of the implications that could arise if teams were to more consistently adopt agile, even in the midst of large and complex projects.

- Agile methods foster communication. Partnerships could become the norm rather than the contention that sometimes persists between software engineering factions.
- With the advent of wide-scale adoption of test-driven development and the continuing advancement of development and test tooling, methods and tools will continue to assist teams delivery quality more routinely.
- It is a common notion in agile approaches that at the end of each iteration, new function must be demonstrated to real stakeholders. Exactly *how* this is to be done is too rarely broached. The authors believe outside-in development holds promise as a means by which such interaction can become more routine (Kessler and Sweitzer 2007).
- Concepts that have been identified in such works as *Implementing Lean Software Development*, while now largely a key point of discussion in the agile community, will eventually rise to prominence, as teams try to focus on the overall software engineering approach from requirements to customer value, rather than their own individual domains (including test) (see Poppendieck and Poppendieck 2007).

The authors' suspicion is that it appears odd to readers of an article on software test to find a conclusion pointing toward the promise of agile methods. The job of test teams is not simply finding defects, and this is routinely forgotten. Stable code, validated by customers on a regular basis, helps to produce products that delight customers. Isn't this the point?

REFERENCES

Beck, Kent. 2003. *Test driven development–By example.* Boston: Addison-Wesley.

Beizer, Boris. 1995. *Black box testing: Techniques for functional testing of software and systems.* New York: John Wiley & Sons, Inc.

Bush, Vannevar. 1945. As we may think. *The Atlantic Monthly.* (July): 101-108.

Gabor, Andrea. 1990. *The man who discovered quality: How W. Edwards Deming brought the quality revolution to America.* New York: Times Books.

IEEE/ACM-CS Joint Task Force on Software Engineering Ethics and Professional Practices. Software Engineering Code of Ethics and Professional Practice. 1999. See URL: http://www.acm.org/serving/se/code.htm.

Kessler, Carl, and John Sweitzer. 2007. *Outside-in development.* Upper Saddle River, N.J.: Addison-Wesley.

McConnell, Steve. 2004. *Professional software development–Shorter schedules, higher quality products, more successful projects, enhanced careers.* Boston: Addison-Wesley.

Poppendieck, Mary, and Tom Poppendieck. 2007. *Implementing lean software development.* Upper Saddle River, N.J.: Addison-Wesley.

Rivera, Ted, Adam Tate, and Scott Will. 2004a. Red zone analysis. In *Proceedings of the International Conference on Practical Software Quality Techniques & International Conference on Practical Software Testing Techniques Held in Washington, D.C., 22-26* March, Golden Valley, MN: International Institute for Software Testing.

Rivera, Ted, Adam Tate, and Scott Will. 2004b. Implant and transplant testing. In *Proceedings of the International Conference on Practical Software Quality Techniques & International Conference on Practical Software Testing Techniques Held in Washington, D.C., 22-26* March, Golden Valley, MN: International Institute for Software Testing.

Sun Tzu. 1993. *The art of war.* New York: Ballantine Books.

Will, Scott, Ted Rivera, and Adam Tate. 2003. Turn your customers into interns. *Quality Progress.* (November): 31-37.

Yourdon, Edward. 1999. *Death march: The complete software developer's guide to surviving "Mission Impossible" projects.* Upper Saddle River, N.J.: Prentice Hall PTR.

BIOGRAPHIES

Ted Rivera is a software engineering thought leader with IBM's software strategy organization within the software group. For the last 24 years he has been a part of various teams in IBM that deliver software in virtually every role a software team executes, including development, test, information development, and support. Over the last several years, Rivera has been active in helping IBM teams make the transition from traditional approaches to development (such as the waterfall methodology) to more modern agile methods (such as XP, Scrum, RUP, and Lean). He holds a number of patents and has published several dozen articles in software journals and in conference proceedings on a wide range of topics in the software engineering field. He can be reached by e-mail at: trivera@us.ibm.com.

Scott Will is responsible for technical quality engineering for IBM's software group. He has been a lead programmer (programming in C, C++, and Java), a customer support team-lead, a test manager, and now holds the position of quality architect and consultant within IBM. His main responsibilities include helping teams improve product quality through the adoption of leading-edge tools and development methodologies (such as data mining tools, static code analysis tools, and helping teams move into the agile arena). He holds a number of patents and has published articles in software journals and in conference proceedings on a wide range of topics in the software engineering field. He can be reached by e-mail at: sawill@us.ibm.com.