Rex Black on Software Testing Best Practices

"*Back in 2010, at the launch of* Core Magazine, http://www.coremag.eu/, *I wrote a series of columns to welcome people to the magazine. As a sort of Throw-Back-December, here they are, as they appeared in the original magazine issues. I hope you enjoy them.*"

-Rex Black

Column 1
Agile When It Works

Greetings, and welcome to my quarterly column on software testing best practices. When I was asked to write this column, I had to choose the approach, the theme. The writers' aphorism says, "Write what you know." So, what do *I* know?

Well, if *you* know me and my consulting company, RBCS, you know that we spend time with clients around the world, in every possible industry, helping people improve their testing with training or consulting services, or doing testing for them with our outsourcing services. Our work gives me insights into what goes on, the actual day-to-day practice of software testing.

Now, not all of what goes on is good. There are bad practices, and we help clients fix those. But you don't need me to write about what not to do. Aren't there enough scolding bloviators in our business? With a click of your mouse, you can read these people's disdainful rants about testers they think are stupid, testers they think are in the wrong "school of testing," testers they love to hate. Lecture, scold, rant, bloviate. How tedious!

So, being a contrarian, I will do the opposite: With the exception of the paragraph above—where I poured well-earned scorn on people who write bad things about other testers—this column will be 100% good news. I will discuss testing best practices that my associates and I have observed other smart people doing. That's right. No negativity and no bragging about myself either. A simple theme: *What other people do right when they test and why we love it.*

I want to start with Agile testing when it works. No, I'm not recanting. Yes, I've written about the testing challenges of Agile, and I stand by what I wrote. Yes, I can talk about testing worst practices in some Agile teams, and I might in some future article—but not in this column. In

this column, I focus on what's right about Agile. Here are five testing best practices we've found in Agile done right:

Unit testing. Okay, it's true that most programmers, even Agile programmers, still have a lot to learn about proper test design. But if you're a professional tester like me, you love hearing programmers talk about the importance of unit testing. We all know that unit tested software is easier to system test.

Static analysis. Not only do smart Agile programmers like unit testing, they like static analysis, too. Coding standards are hip again. Cyclomatic complexity is back. Writing more testable, more maintainable code: that'll make testers' lives easier in the long run.

Component integration testing. This under-appreciated test level exists—on properly run Agile projects. You can go years on sequential-model projects without seeing component integration testing. However, on good Agile teams, people look for integration failures, and, because of continuous integration, the underlying integration bugs aren't hard to find.

Tools, tools, tools—and many free. All of this talk about unit testing, static analysis, and component integration testing would be just that—talk—without tool support. Fortunately, the Agile—err, what should we call it?—movement, revolution, fad, concept, pick your term, has brought with it a lot of tools to support these best practices, along with other best practices. For those of us without unlimited budgets—and isn't that all of us?—a lot of the best tools are free, too.

Tester and developer teamwork. At the beginning of our latest assessment, I had a great conversation with a test manager who works on Agile projects. Among areas of agreement: our shared joy at the death of a bad idea. The bad idea in question was this: the idea that the role of the test team is the quality cop, the enforcer, the Dirty Harry to the punks of the software team. "Seeing as I can refuse to approve the release, you gotta ask yourself one question: Do you feel lucky, programmer?" Instead, we see more people working together, collaborating for quality, and that's especially true on good Agile teams.

Just this morning, I spent three hours talking to two programmers—real seasoned professionals with years in the field—talking to them about testing. The testing that they did. In fact, it wasn't so much about testing, but testing as an essential tactical element in a larger strategy for higher quality code. They really knew testing, and they knew how the Agile approach and tools were helping them to achieve better testing and thus better code. At the end of our talk, I mentioned how much I enjoyed talking to programmers about good testing and good code.

He replied, "Yeah, we spend a lot of time around here talking to each other about that. How to be better craftsmen. How to test better. How to build better code."

Wow. If the entire methodology, the lifecycle, the tools, and every other aspect of Agile fades away, leaving behind only the habits of programmers serious about code quality, and testers working cooperatively with them to achieve it, that will be a signal achievement in the software engineering profession. Best practices, indeed.

So, there are the first five best practices. What comes next? That depends on which great practices my associates get to see in the next three months. See you then.

Column 2
Rational, Quantitative Quality Management

Greetings. Welcome back to my quarterly column on software testing best practices. In this column, I discuss testing best practices, observed by my associates and by me, applied to solving real-world problems. This column has a simple theme: *What other people do right when they test and why we love it.*

Over the last few months, we've been working with a client to help them understand what software bugs cost their company. We've used a technique called *cost of quality* to do so. Cost of quality is a simple, powerful, and well-established technique for understanding the costs of bugs. While space doesn't allow a description of that technique here, you can find all the details at www.rbcs-us.com/software-testing-resources/library/basic-library in the article entitled "Testing ROI."

Our client now has a solid estimate of these bug costs. We estimate the costs between 15% and 25% of their total software budget. While this number may seem large and startling, it is in fact not an unusual figure. We've seen similar figures with some other organizations that have applied this technique.

Our client is now putting steps in place to improve the accuracy of their bug reporting information so they can develop smart process improvement plans. Over the next six months, they expect to gather the data they need to decide how to reduce the cost of bugs to their organization. The data is focused on identifying the benefits available from three possible process improvement opportunities:

Phase containment: Industry studies have shown over and over again that many of the bugs are introduced in the requirements and design phases, and that the cost associated with a bug increases each time a bug escapes from its phase of introduction into a later phase. Phase containment refers to the extent to which bugs are detected and removed prior to escaping into later phases. Our client is interested in what kinds of savings would be available by increasing phase containment.

Defect reduction: The Pareto Principle (also called the 80/20 Rule) tells us that relatively few causes account for most of the outcomes. In the case of defects, that means that a handful of the possible mistakes that can occur in the software engineering

process create most of the bugs. Our client is gathering data to identify those critical mistakes behind the majority of their bug costs.

Invalid bug reports: A certain number of bug reports will turn out to be invalid, due to problems with test environments, vague requirements or design specifications, undocumented changes in functionality, and tester error. While a given level of invalid bug reports is to be expected—we typically target 5% or less in our assessments—our client has found their number to be well above that. They are interested in understand why their rate is higher than normal and identifying ways to reduce it.

Within six months, they expect to have solid data to determine what the right next steps are.

While this plan of action might seems obvious, it is in fact quite extraordinary. Most such organizations can produce reports showing what they spend on office supplies, but relative few software engineering organizations have measured their bug costs, even though the technique is quite simple. Even fewer organizations have rational, fact-based, quantitative quality management plans, such as the ones my client will develop in the next six months.

I'd encourage you to take a look at the cost of quality technique. Use it to quantify the cost of bugs in your organization. Then, take the next step. Gather detailed data about the bugs, when they are introduced, what mistakes cause the bugs, and how many invalid bug reports you have. Armed with these insights, you can then reduce your bug costs. This straightforward process will allow you to join the ranks of the elite few that practice rational, fact-based, quantitative quality management.

Column 3
Knowing When to Ask for Outside Help

Greetings. Welcome to another edition of my quarterly column on software testing best practices. As readers will remember this is about testing best practices, observed by my associates and by me, applied to solving real-world problems. This column has a simple theme: *What other people do right when they test and why we love it.*

In this column, I am going to talk about something applicable to testing, though not specific to testing. How do smart managers and companies recognize when to ask for outside? (By "outside help," I mean training, consulting, or staff augmentation/ outsourcing.) This is not a terribly complicated question, but people often fail to consider it properly. These unconsidered decisions can result in using outside help when it's not required (which wastes money) or failing to use outside help when it is required (which results in lingering, unsolved problems). Let's look at the three scenarios separately.

Training: It might seem that, any time you have a skills gap in a test team, outside training is required. In fact, through self-study (using books and the Internet) or cross-training (using other people who already have the skills), a number of skills gaps can be filled. However, self-study and cross-training typically require a larger investment of in-house staff time than online or classroom training, and have a longer learning curve with a higher rate of mistakes during that learning curve. So, for skills gaps that must be filled quickly, or filled in such a way that the rate of mistakes is low, outside training is the best option. Remember to ensure that skills acquired during outside training are applied immediately to tasks at work, or else those skills will quickly be lost.

Consulting: We get a lot of inquiries from potential and soon-to-be clients that boil down to this: "We have pain and problems related to testing and/or quality, but we're not sure why or what's broken." It might surprise you, but such an inquiry is often the start of an engagement that is rewarding for us and for the client. Here are three good reasons to engage an outside consultant:

You are aware of symptoms—effectiveness, efficiency, or stakeholder satisfaction problems—but have no idea how to accurately diagnose the underlying problem (e.g., you know that your defect report data is inconsistent, but can't imagine why people are entering bad defect reports);

You know how to accurately diagnose the underlying problem, but could not solve the underlying problem even if you knew what the problem was (e.g., you know that you are not optimizing the order of running your tests, and you suspect that risk based testing would help, but you don't know how to institute risk based testing and have failed trying to do so once before); or,

You know how to accurately diagnose and solve the underlying problem, but the political price that an insider would pay for solving the problem is too high (e.g., you believe that management is setting project constraints that make delivering quality products impossible, and you need someone to validate that opinion and report it to executives).

Barring diagnostic, problem-solving, or political obstacles, it's often better to try to solve problems internally and save the consulting budget for the tougher nuts to crack.

Staff augmentation/outsourcing: Staff augmentation refers to bringing non-employees, such as contractors or staff augmentation firm employees, into an organization, working on-site. Outsourcing refers to sending the work to the contractors or staff augmentation firms, rather than bringing them to the work. Either way, somebody other than an employee of your company is doing the work. Basically, those of our clients that make smart decisions about staff augmentation and outsourcing typically consider three factors:

They can get the work done for a lower cost by using the staff augmentation or outsourced team.

The staff augmentation or outsourced team has skills and experience that will allow them to do work that could not be done by the current in-house staff, and it is impractical to solve the in-house staff skills gap.

The current in-house staff could do the work, but other, more-important work precludes assigning the work to in-house staff.

Note that in all three cases, smart clients also consider whether the staff augmentation or outsourced team can do the work with the same or better quality than their in-house staff could. If not, then either they have the wrong team or the job is not one to send to outside people.

Getting outside help, when done properly, is an effective and efficient way to increase skills, solve problems, and get work done. However, you should think critically about

when outside help is needed, and when the job can be done by inside staff. Those of our clients who make the smartest decisions about outside help consider the factors I've discussed. Of course, when outside help is deemed necessary, the *right* outside help must be selected. Since I've run out of space in this column, the question of best practices for vendor selection must wait for a future column.

Column 4
Deciding Which Bugs to Fix, and Which Bugs Not to Fix

Greetings. Welcome to another edition of my quarterly column on software testing best practices. As regular readers will remember, this column is about testing best practices, observed by my associates and by me, applied to solving real-world problems. This column has a simple theme: *What other people do right when they test and why we love it.*

In this column, I am going to cover a best practice related to bug management. This best practice, an approach for deciding which bugs to fix and which bugs not to fix, is sometimes referred to as *bug triage*. The word *triage* is a French word meaning to select sort, but has evolved to include any situation where we must determine priorities for action in an urgent situation [see www.dictionary.com].

While the process of bug triage can vary from one organization to another, generally it involves a meeting where project and product stakeholders review the active bug reports. Active bug reports are those either newly discovered or where some work on the report is currently in progress. The project and product stakeholders involved in the review should represent the participants involved in moving the bug report through its lifecycle (including the test manager) as well as those who can represent the business, customer, and user perspectives on the consequences of fixing or not fixing the bug.

This meeting should occur regularly during test execution, when bugs are being identified and reported. It should occur regularly enough that the meeting is not too long and that bug reports do not languish too long before a decision is made. A good rule is to meet at least weekly and perhaps as often as daily during test execution.

The meeting is a type of project review where the objective is to make decisions and to assign action items associated with those decisions. The participants should decide the ideal action to take next to move the bug towards the best possible outcome, relative to the other actions underway on the project. The possible actions for each bug include the following:

Gather further information: The tester who reported the bug, or a developer or other technical contributor, is directed to investigate the failure or the underlying bug. This

action should be used sparingly, otherwise bugs will move slowly and inefficiently through the bug lifecycle.

Fix immediately: A developer is assigned to fix the bug as quickly as possible. This action should occur when the bug affects the efficiency of the project, such as when the bug blocks important tests.

Fix before release (or in this iteration for Agile projects): A developer is assigned to fix the bug at some point prior to the release. This action should occur when the bug is important for the business, users, and customers, but can be fixed later in the project without affecting the project's ability to succeed.

Fix in next release (or in the next iteration for Agile projects): No work is assigned at this time, but the bug is scheduled for repair. This action should occur when repair of the bug is important, but not urgent.

Fix at some future date: No work is assigned at this time, and the bug will be reconsidered for repair at the beginning of each release or iteration. This action can be taken when the failure associated with the bug is one we can live with indefinitely. However, the triage team must take care not to accumulate too much technical debt by over-using this action.

Accept as a permanent limitation: The bug is not worth fixing.

As the word triage suggestions, the objective is to determine the best possible set of actions to take for all of reported bugs. The triage team must take into account all the active bugs, and other project actions, as we are typically acting in a situation of limited resources.

In making these decisions, the triage team must consider the following factors:

Benefits: What advantages will necessarily accrue to project and product stakeholders upon the repair of this bug?

Opportunities: What possible immediate or in the future advantages might accrue to project and product stakeholders by repairing this bug?

Costs: What effort, resources, and time must we expend to repair the bug?

Risks: What bad outcomes might occur as a result of repairing the bug?

If the triage team decides that the benefits and opportunities exceed the costs and risks, then the bug should be fixed, and the team should decide the proper schedule for fixing that bug, selecting the options given above. These considerations must include the entire project team. For example, the cost of repairing the bug includes not only the development effort, but also the confirmation test and regression test effort associated with the bug's repair.

As you can see from this brief description, the essential process, participants, decisions, and considerations of bug triage are simple to describe. Why is bug triage so important to project success? On any project, we have limited people, resources, and time. We must deploy those limited assets in a way that optimizes the outcome for the project and the chances of the product's success. Bug triage is a best practice that allows us to do so for the essential activity of managing bug reports through their lifecycle, a critical testing process. I hope this column helps you institute better bug triage practices on your current or next project.

Column 5
System Integration Testing

Greetings. Welcome to another edition of my quarterly column on software testing best practices. As regular readers will remember, this column is about testing best practices, observed by my associates and by me, applied to solving real-world problems. This column has a simple theme: *What other people do right when they test and why we love it.*

In this installment, I am going to cover a best practice related to testing large, integrated systems of systems. You have a system of systems when you have multiple systems that have to co-exist in the same production or customer environment, often sharing data and interacting in that same environment. This best practice is system integration testing.

System integration testing involves putting these systems together, prior to release. However, it should be more than that. Done properly, system integration testing has the following properties:

- It takes place in an environment that ideally replicates, or at least properly represents, the production or customer environment.
- It uses production or customer data, if necessary anonymized for privacy and security.
- It involves good configuration management so that current versions of each system are in the test environment, including handling changes as systems evolve.
- If new systems are being built, those systems are integrated into the testing environment in risk order, i.e., the systems more likely to have integration problems are integrated earlier.
- Test coverage focuses on potential problems in the areas of interoperability, performance, reliability, security, and data quality across the systems.
- It takes place on systems that have been through adequate earlier levels of testing, so that most bugs in the individual systems have already been detected and removed, i.e., the bugs that remain to be found relate to interactions between systems and the coexistence of systems.

As you can see, system integration testing can be a complex activity. Good test planning is essential to ensure that everything flows properly.

Because system integration testing is complicated, Figure 1 can help you understand it better. Let me explain some of the important elements. There are five projects in progress at the time shown in the figure. (The projects may be following a sequential lifecycle, as shown in the figure, or an Agile lifecycle; the system integration testing remains the same.) There are six system integration testing cycles each year, one every two months. There are two production-like test environments, giving us two overlapping system integration testing activities. This overlap accelerates the frequency of releases without rushing the testing too quickly.

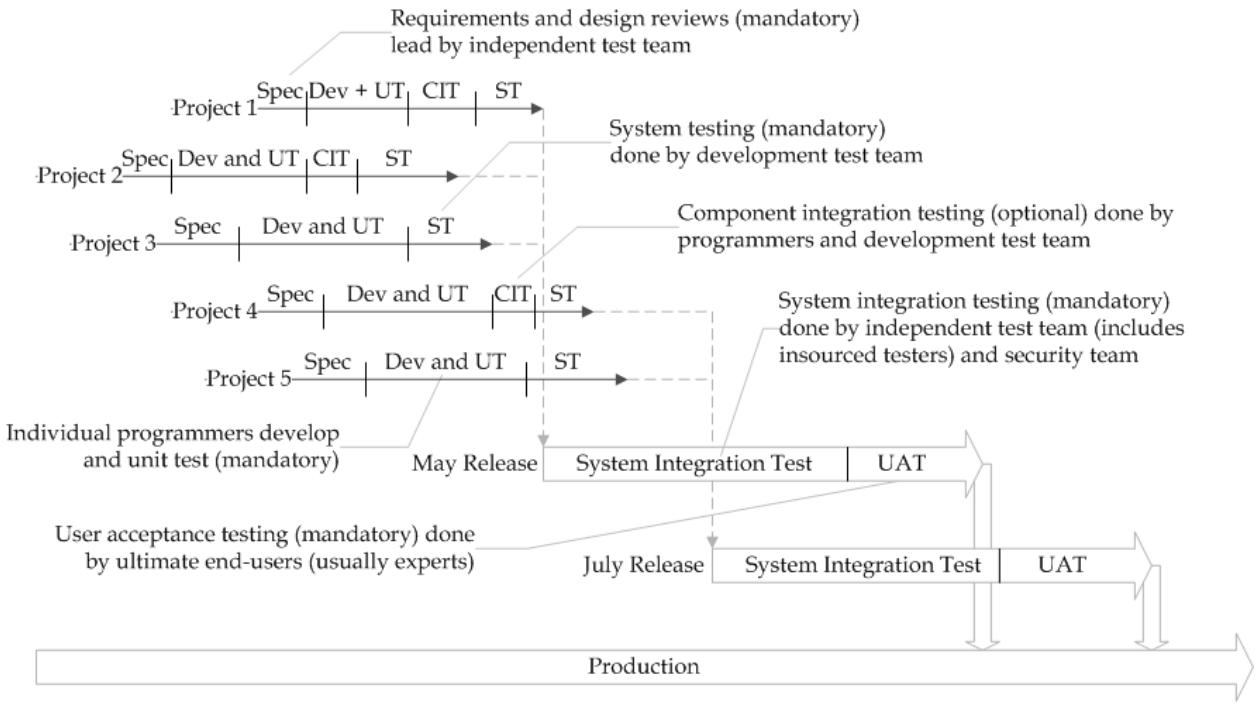

Figure 1: An Example of Good System Integration Testing

Every other month, the projects that have code ready for system integration testing deliver that code into the appropriate system integration test cycle. Being ready for system integration testing means that the system has gone through thorough testing already, including requirements and design reviews, unit testing, component integration testing (where applicable), and system testing. If those tests are not complete, the project will wait for the next bi-monthly test cycle to start.

During the system integration testing, when bugs are found, those bugs are fixed in the systems. New releases of those systems are delivered in an orderly fashion. In some variants of this process, systems can enter system integration testing after it has already started, but before some pre-determined cut-off point. This can speed release of critical projects, but does involve accepting some increased risk. Whether for bug fixes or larger changes, regression testing is necessary when one or more systems change. Good test automation can help with this process, as is the case with other test levels.

This figure shows system integration testing as done by a number of companies we have worked with. These clients regularly achieve very high defect detection effectiveness in their system integration testing (in some cases above 99%), and enjoy very high levels of quality in their production systems. I consider this approach to system integration testing an industry-proven best practice.

There's a lot more to effective system integration testing than I could fully explain in this short column, but I hope I've piqued your interest. Critical problems can arise in the areas of interoperability, performance, security, reliability, and data quality when large, complex systems of systems are assembled. System integration testing is a way to minimize the risk that bad things will happen with those systems when they are released to the data center or the wide, wild world. I encourage you to study this important topic further, and adopting system integration testing best practices that are appropriate to the systems you test.

Bio

With thirty years of software and systems engineering experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For over twenty years, RBCS has delivered consulting, training and expert services to clients, helping them with software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS advises its clients, trains their employees, conducts product testing, builds and improves testing groups, and hires testing staff for hundreds of clients worldwide. Ranging from Fortune 20 companies to start-ups, RBCS clients save time and money through improved product development, decreased tech support calls, improved corporate reputation and more. As the leader of RBCS, Rex is the most prolific author practicing in the field of software testing today. His popular first book, Managing the Testing Process, has sold over 50,000 copies around the world, including Japanese, Chinese, and Indian releases, and is now in its third edition. His nine other books on testing, Advanced Software Testing: Volumes I, II, and III, Critical Testing Processes, Foundations of Software Testing, Pragmatic Software Testing, Fundamentos de Pruebas de Software, Testing Metrics, and Improving the Testing Process have also sold tens of thousands of copies, including Spanish, Chinese, Japanese, Hebrew, Hungarian, Indian, and Russian editions. He has written over forty articles, presented hundreds of papers, workshops, and seminars, and given about fifty keynotes and other speeches at conferences and events around the world. Rex is the past President of the International Software Testing Qualifications Board and of the American Software Testing Qualifications Board.