



Presents
An IT Metrics and Productivity Journal Special Edition

Focus on Rex Black, Expert Software Test Practitioner & Author
A CAI State of the Practice Interview
July, 2006

Biography of Rex Black

Rex Black is president and principal consultant of Rex Black Consulting Services, Inc., a testing and quality assurance firm serving clients such as Bank One, Cisco, Dell, and the US Department of Defense. He is also the author of *Critical Testing Processes* and *Managing the Testing Process*. Rex holds a degree in Computer Science and Engineering from UCLA. He belongs to the Association for Computer Machinery and the American Society for Quality. Our interview between Michael Milutis, Executive Director of the IT Metrics and Productivity Institute, and Rex Black took place in March of 2006.



CAI: Could you tell us a little about yourself, your background, and what you are working on today?

REX BLACK: I got started in software and systems engineering back in 1983 doing work on financial applications that were based on CP/M and MP/M micros. Shortly after that, I moved over to Unix-based applications. I continued doing programming and system administration work for a number of years before eventually falling into software testing in 1987.

I initially worked on automated test scripts, test execution and test project management for an independent software test lab. Subsequent to that, I found employment as a software test manager for several different test companies that operated within a variety of different technological environments. They were good companies with good technology but bad business plans. The result was that by 1994 I found myself going through my second layoff in a two-year period. At that point it occurred to me that if I was going to be out of work because of bad business decisions, I'd like them to be my own bad business decisions. Consequently, I started my own consulting company, Rex Black Consulting Services. And I am happy to say that I have been busy ever since.

Over the years, I've worked with a lot of different technologies, including end-user and IT applications, financial and medical applications, embedded systems, consumer electronics and hardware. My experience really runs the gamut.

Lately, our business seems to be breaking down into three main areas. The first area is training - with a special emphasis on certification using the Foundation and Advanced

syllabi from the International Software Testing Qualifications Board. The second area is consulting - classical consulting assessments where we come in and talk to people about how they are doing their software testing and how they might be able to make improvements. The third area is staff augmentation, either onsite for our clients or offshore at our operation in New Delhi. We have approximately 30 testers in India working for us right now, through our Pure Testing sister company.

Our business is completely international. Just yesterday I signed an agreement with a marketing partner in Brazil and I hope to sign another agreement with a marketing partner in South Africa later this week. Once we sign the South African contract we'll be doing business on all 6 continents. I don't count Antarctica, because the penguins have no budget. They can't hire me and we can't work there.

CAI: In your book *Critical Testing Processes*, you talk about having encountered certain common testing themes - good and bad - over the course of your career. Could you highlight some of these for us?

REX BLACK: A particularly good theme that I sometimes see is high project support levels for software testing and quality. Whenever software testing is viewed as an ally and supported by management, that's a good thing. Another good theme I sometimes see is constancy of purpose at the project management level. Projects that are very reactive and constantly changing direction tend to be very problematic for software test groups. On the other hand, projects that maintain focus and direction are much easier for testers and will have a very good effect on the software testing process.

Bad themes would include unrealistic or inconsistent testing expectations from management or from other project team members. One way to very quickly assess the expectation levels around software testing in your organization is to ask your project managers what they think the defect detection percentage ought to be for an independent software test group that is testing a system prior to its release. The very best software testing organizations will achieve a 95% defect detection percentage, or perhaps a little higher. However, you often run into situations where managers, when asked what the defect detection percentage should be, respond with a blank look and ask, "Shouldn't they find everything?" That's indicative of unrealistic software testing expectations. When unrealistic expectations are present, regardless of how well a job is done, the efforts will always be seen as a failure in somebody's eyes. That's bad.

A less significant bad theme that comes up repeatedly is when personal conflicts exist between certain software testers and certain developers. I've had software testers tell me that they take delight in targeting specific developers and finding lots of bugs in their work just to make them feel stupid. Obviously, this is very corrosive to teamwork. I've also seen developers with extremely defensive attitudes towards testers who tend, as a result, to attack their software testers any time an issue is found. You can see how these things feed off of each other.

CAI: In your book, you outline 12 critical testing processes. Why is process so important in software testing? Could you highlight your answer with a few

examples from this list of 12?

REX BLACK: The 12 software testing processes outlined in my book are critical in the sense that they all impact the efficiency and effectiveness of either the testing effort itself or the overall project.

For example, one of the 12 critical testing processes is bug reporting. Over the course of a medium-sized project, there could easily be 100, 200, 1000 or even 5000 bug reports filed. A process repeated that many times presents many opportunities for efficiency improvements. In this case, two opportunities for improvement are injection of fewer bugs and better communication between testers and developers.

Another critical testing process is the overall reporting of results. This entails the communication to management of what was found, what tests were run (or not run), what risks we've covered (or not covered), where exactly the system might be facing the most serious problems, and what kinds of defects are expected to be found. The results reporting process is how the test team delivers value to management. The goal, in fact, of the results reporting process is to give management the necessary information they need to make very important and difficult decisions regarding product ship dates. In other words, the reporting of results from success testing addresses the quality dimension of the project.

If you look at the dimensions of a project (the features, the schedule, the budget and the quality) you will see that it is fairly easy for a manager to ascertain where the project stands in terms of features, schedule and budget - but not in terms of quality. What's tricky, especially towards the end of a project, is ascertaining quality. However, if the reporting of results from testing is done right, the test team can help the project management team get the product out the door at just the right moment.

CAI: How would you say most IT organizations rate in terms of their software testing processes?

REX BLACK: I'd have to say that most organizations rate pretty low, even the ones that train their software testers. I often meet people with years of paid software testing experience, people who could easily be considered software testing professionals, who aren't familiar with the basics of test design, bug reporting, or how testing adds value. They often aren't aware of testing concepts that have been written down and published recently or in some cases for over 25 years.

CAI: Why do you think this is the case?

REX BLACK: The software testing field has not done a very good job of building on the foundations that were put in place back in the 70s and 80s by people like Boris Belzer, William Hetzel and Glen Myers.

Moreover, software testing professionals, in general, don't do a very good job of connecting software testing to business value. As a profession, we simply don't do a

good job of communicating the value of software testing to project managers and other senior executives. Because nobody in management really understands how software testing ultimately connects to value, software testing tends to be the first thing that gets put on the chopping block when other priorities arise.

CAI: How does one connect software testing to business value? How does one best communicate this?

REX BLACK: I think a smart way to connect testing with overall value is to look at the alignment of defect detection, defect removal, and customer quality satisfaction.

To this end, several key questions need to be answered: 1) what defect detection percentage should the organization have per unit testing; 2) should there be regression testing and how effective should it be; and 3) should there be code reviews, design reviews and requirement reviews and how effective should these be?

I had an interesting situation that came up when I was doing an assessment for a client last year. Their defect detection percentage was good, in the 90-95% range for each project, but their defect removal effectiveness was fairly low, down in the 80% range. In other words, a sizeable percentage of the defects that were being detected during testing were being deferred rather than removed. However, when I talked to management about this, and pointed out the disconnect, they said, "We're not over-deferring, we're over-testing. We're testing things that don't matter." In order to get to the heart of this, I studied the results of the most recent customer satisfaction surveys; specifically, the surveys that were focused on various quality characteristics. In doing this, I noticed that the numbers were well below where they wanted them to be. Consequently, my response to management was, "No, you really are over-deferring. If you were over-testing, customer satisfaction with quality would not be so low. The low customer satisfaction suggests that you are over-deferring defects."

Another thing to keep in mind when connecting software testing to overall value is who you need to be communicating with. I mentioned earlier that I started my own consulting company after suffering a second layoff in two years due to downsizing. An additional variable was actually at play. I was inadvertently ignoring the needs of the field support people, who obviously had a very strong and legitimate need to understand what was going on. It was important that their concerns be addressed. But this didn't happen. Consequently, the field support people were not confident in the testing process, primarily because they didn't believe that the testing would address their needs. In the end, what that meant was that they were very much in support of the layoff of the test team, including me. What I learned from this is that it is very important to identify all the stakeholders on a project and to make sure that all of them are being well served by the testing. That's a critical part of the value communication process.

A final thing you want to keep in mind when trying to measure, and ultimately communicate, the value of your testing efforts is the return on investment. You can do this by looking at the cost of quality, which is a way of looking at the cost of defect detection and repair prior to release (as opposed to post-release). You can measure the efficiency of your test team by studying this differential. If it costs, for example,

\$500 on average to find and remove a bug prior to release (as opposed to \$15,000 to find and remove a bug post-release) then every defect that the test team finds prior to release will save your organization \$14,500. That's a fairly easy calculation to make. Management can then use this kind of data to make hard-headed economic decisions about how much money to invest in testing.

CAI: For organizations that want to quantify the ROI being generated from their own testing efforts, are there are specific methods you might be able to recommend?

REX BLACK: In my book *Critical Testing Processes*, I spend an entire chapter outlining how to quantify testing ROI for any given organization. Any organization that wants to measure the value of their software testing effort could follow that process and come up with the return on their test investment. There are also a number of articles and presentations that are out on the library page of our website, www.rexblackconsulting.com, that talk about this issue of testing ROI and how much money can be saved through testing.

In terms of overall industry numbers, a fellow named David Rico once did a study for the Department of Defense in which he came up with 800% as an average return on testing investment. That study can probably be found on the internet simply by searching on "David Rico software process improvement." Capers Jones also has quite a bit of data on industry best practices, as well as the risks associated with poor practices, and he includes poor testing as one of the major software project risks in his book *Estimating Software Costs*. The increased risk to a project that is created by poor testing, particularly for large and complex projects, is really amazing. The risk of project failure can easily double or quadruple.

CAI: Could you quantify for our readers, more specifically, the impact that software testing improvement can have on an organization, in terms of ROI?

REX BLACK: Yes. You can look at the cost of quality model, which is imperfect in that it doesn't capture all of the value delivered by testing, but which nevertheless captures one of the primary value components - the detection of bugs that can be fixed. Using the cost of quality approach, it is not unusual for me to see a return on testing investment in the 700-800% range.

I once conducted an assessment for a bank. We found that the return on their testing investment, in terms of money saved from the avoidance of field failures that might otherwise have occurred, was 3,500%. Actually, I think it would have been higher but once we hit 3,500% I went to the sponsor of the assessment project and explained that we should probably stop counting since if we came up with a number that was any higher nobody would ever believe it. I don't think most people are aware of how expensive field failures can be. That, in turn, impacts their perception of the value that is added by testing.

CAI: What methods and metrics do you think are important for measuring the effectiveness of your testing efforts?

REX BLACK: First of all, you can look at your defect detection percentages for high priority bugs versus all bugs. If your testing is properly focused, you should find a higher percentage for high priority bugs than for all bugs. You want to be finding more of the scary stuff than the trivial stuff.

You also want the test management and project management team, in general, to be looking at this issue of testing ROI. How much money is being saved, and how can we save even more money? This question is going to drive upstream improvements that are going to make the test execution period shorter and cheaper. And that's virtuous from two perspectives. First of all, the cost of testing is generally at its highest during the test execution process. Secondly, anything we can do to shorten the test execution process will reduce the overall testing process and therefore, the length of the project itself. Naturally, test execution tends to be on the critical path for release, so shortening test execution also accelerates releases.

I think you should also examine the metric of confidence. You should be continually trying to increase management confidence. That means figuring out how to report results so that management better understands them.

Other things that I think are worth looking at include the overhead and risks associated with the deployment of test releases, as well as the bug reporting process itself. Regarding the latter, you should be looking at ways to reduce the percentage of bugs that are rejected due to poor writing or insufficient information or improper classification. In short, try to make the defect cycle of "find, fix and confirm" as tight as possible.

In the end, there are quite a few metrics that can be combined in a sort of dashboard view for gaining visibility into: 1) how your testing efforts are doing; and 2) where they can be improved.

CAI: Do you have any advice for organizations that are trying to integrate testing throughout their entire software development life cycle?

REX BLACK: You really have to involve senior testers in the prevention of bugs, up front and right from the beginning. I think it's a very common mistake to either get testers involved too late in the game, when there's very little that can be done other than grinding out as many bugs as possible, or to put too few testers on the project entirely.

Any attempt to integrate testing into the full development lifecycle has to be taken very seriously. It cannot just be a token attempt.

CAI: Developing software test conditions is critical when reviewing use cases or detailed functional requirements. What techniques do you recommend for

testers when receiving such artifacts, and what do you recommend they should do if the artifacts are substandard?

REX BLACK: I see this as a two-step process.

In the first step, you must do some high-level initial work to determine: 1) what should be tested; 2) in what sequence the testing should be conducted; and 3) how much testing should be done. Typically, I use some form of quality risk analysis to make these determinations. I am going to be looking for specific kinds of problems that could occur and trying to assign risk levels to each so that appropriate sequencing and resource allocation decisions can be made.

The second step involves the design of specific test cases to cover these various risks (to the degree appropriate, based on the level of risk). I've actually written a book on test design called *Effective and Efficient Software Testing*, which we are currently in the process of publishing. In the book, I identify a number of software test design techniques. These techniques have been around for quite a while and are well established – techniques like equivalent partitioning, boundary value analysis, decision tables, transition diagrams and domain analysis. They are techniques that can be used to discover specific conditions for testing and for the mitigation of particular risk areas.

One of the reasons that I like to use risk analysis as the first step in this approach is that so often we find ourselves working on projects where the requirements and design specifications are either poor or missing entirely. In short, it's still quite possible to do a risk analysis in these situations because the risk analysis process is as much about connecting to and understanding project stakeholders as it is about using documents (to essentially determine the same thing). Nevertheless, if you've got a document, the risk analysis is going to be more complete and the test design more efficient. I generally state, as a rule of thumb, that if you have to design tests using poor requirements specifications you will be adding an approximate 20% inefficiency into your software testing process.

CAI: What methods do you recommend for estimating software testing effort?

REX BLACK: I am not a big fan of tester-to-developer ratios as a method for test estimation. I think they are unreliable. In certain very stable circumstances involving small, incremental changes within maintenance releases I have seen this approach work. In general, however, the approach I would recommend for the estimation of software testing effort is the creation of a work breakdown structure developed by the test team as a whole, using something like Microsoft Project or even just a white board with dates on it. You can then measure your work breakdown structure against historical metrics, preferably metrics derived from similar, past projects conducted within the same organization along with some industry averages. At that point, you will be positioned to start negotiating with project management regarding what can and cannot be done.

CAI: What is the role of automation in software testing?

REX BLACK: I think automated software testing is going to be essential for any complex application that involves a significant risk of regression. It's simply not possible to manage regression risk through manual testing on large complicated applications, at least not with any reasonable allocation of effort and time. Organizations that try to get away without automation when they are dealing with complex products, long lifetimes or mission critical applications are really playing Russian Roulette. That's especially true when you get into industries and businesses where there might be certain regulatory compliance issues that can change rapidly, thereby causing results to be accurate one day and inaccurate the next. That can be a pretty dangerous scenario.

One of the concerns that I have about software testing, in general, is that I think we have tried too hard to make testing automation simple. Just as we burned up too much time and energy trying to enable non-programmers to program, through the development of fourth generation languages, we have wasted too much energy and time trying to develop automation tools that could be given to the non-technical business analysts so that they could create their own complete set of reusable regression tests. This has not been widely successful, and it does not seem likely to be successful. The fact is, there is an essential programming element at the heart of automated testing and we need to accept this. We need to accept the fact that automated software testing involves programming a computer to test itself, or to test another computer, and that it is therefore, fundamentally, a software engineering endeavor.

Another problem is that much of the automated test effort, and even the focus of the graphical user interface itself, does not take advantage of the fact that many applications have command line interfaces or application programming interfaces that could be used to create automated test harnesses. This is actually where I got my start as a test engineer in the mid-80s, creating and automating UNIX test scripts that talked to the UNIX command line interface and the API. I think there's a lot of potential there. We are seeing a resurgence of this now, thanks to the agile methodologists and to freeware tools like J-Unit and CPP-unit (and commercial tools like J-Test and C++ Test). These tools can be just as useful for software testers as for programmers. Hopefully, more testers will start taking advantage of them.

CAI: For organizations that are somewhat sophisticated in their approach to software testing, organizations that already have some form of software testing process in place, is there any advice that you might have for developing an improvement program?

REX BLACK: First of all, conduct an assessment and figure out where you stand. I personally do a lot of assessments. I usually follow my book *Critical Testing Processes* as a general framework and measure up against that. Once the assessment is complete, you can then identify the gaps between best practices and current practices. You will always find plenty of gaps.

The difficult question, after the assessment, will be where to begin. To get started with this, you should first look at your basic pain points. Ask yourself which of the gaps yield outcomes that might get management excited (and not in a good way) or that might cause pain, either for customers or for other stakeholders. If you find an excessive rate of defects in the field, for instance, you might want to identify the holes in your test coverage. If you find excessive costs associated with your testing efforts, you might want to reexamine your testing ROI and determine how best your efficiency can be enhanced. Keep in mind, though, that when people say testing is costing too much, they usually mean they are doing too much testing *at the end* of a project. In these cases, it's generally a good idea to focus on the improvement of the upstream processes so that fewer bugs are delivered to the testing process in the first place.

CAI: We hear so much about outsourcing these days. Are there any particular challenges that arise during outsourcing that are unique, in your opinion, to the outsourced testing model?

REX BLACK: In general, outsourced testing raises challenges in the area of infrastructure. You have to have the right communications infrastructure in place to support clean, quick flows of information and test materials from one side to the other. A lot of organizations screw up here because they don't invest enough in infrastructure.

One of the primary drivers for doing outsourcing, especially offshore outsourcing, is cost. Nevertheless, many organizations don't realize that in order to effectively achieve these cost savings, they first have to have the infrastructure in place to make the process as efficient as possible. It doesn't help to pay a tenth of the hourly wage if it takes ten times longer to do everything.

Organizations interested in outsourcing their testing also need to think about how they're going to manage the testing. It's not the same as having everybody in the same room or in a set of cubicles down the hall. You will need a really well defined process to get this right. For example, change management will have to be much more carefully controlled.

There are also trust issues that will need to be dealt with by management. It's very easy, particularly when people can't see and talk to each other in real-time, for questions to come up about other people's intentions. This can result in a lot of problems.

In general, I think a lot of organizations look at the potential cost savings - based solely on labor differentials - and then really get ahead of themselves. They would be better off slowing down, backing up and first putting a plan in place for how to succeed.

CAI: What are some quick software testing fixes that would be generally applicable for all IT organizations, fixes that wouldn't require a large investment and that could have a positive impact right away?

REX BLACK: Pushing the bug discovery process earlier in the project lifecycle is

probably the most generally applicable quick fix. If you're finding a lot of bugs in system testing, and they're slowing up the release and increasing project costs, then you should be looking immediately to upstream activities like integration tests, unit tests, code reviews, design reviews and requirements reviews in order to see what's going on. Are there any measures on the effectiveness of these processes? Are these processes being short-circuited somehow? Is there any sort of schedule pressure that is causing these steps to be skipped up front?

I think, as a general observation, that we have an accepted reliance in our industry on downstream "grinding out" of bugs as opposed to upstream detection and removal prevention. What this means is that we really need to transform our view of testing from one that is purely bug discovery oriented to one that is more risk management oriented, more integrated with the entire development lifecycle, and more focused on up front prevention.

Questions? Suggestions? Comments? Please contact the IT Metrics and Productivity Journal Editor at michael_milutis@compaid.com.