TIME TESTED. TESTING IMPROVED.

Metrics for Software Testing: Managing with Facts: Part 3: Project Metrics

Introduction

In the previous article in this series, we moved from general observations about metrics to a specific discussion about how metrics can help you manage processes. We talked about the use of metrics to understand and improve test and development process capability with facts. We covered the proper development of process metrics, starting with objectives for the metrics and ultimately setting industry-based goals for those metrics. We looked at how to recognize a good set of process metrics, and trade-offs for those metrics

In this and the next article in the series, we'll look at two more specific types of metrics. In this article, we turn from process to project metrics. Project metrics can help us understand our status in terms of the progress of testing and quality on a project. Understanding current project status is a pre-requisite to rational, fact-driven project management decisions. In this article, you'll learn how to develop, understand, and respond to good project metrics.

The Uses of Project Metrics

As I wrote above, project metrics help us understand progress in terms of project objectives. Project metrics provide a means of measuring where we stand on the project and how close we are to achieving project objectives. They give the project team insights that can help them guide the project toward these objectives if current status and trends indicate that the project team might not achieve these objectives within project constraints. (Typical constraints include schedule, cost, and features to be delivered.) As mentioned in the first article in this series, we can talk about metrics as relevant to effectiveness, efficiency, and elegance.

Effectiveness project metrics measure the extent to which the project is on track to achieve desired results. Efficiency project metrics measure the extent to which a project achieves those desired results in an economical fashion. Elegance project metrics measure the extent to which a project effectively and efficiently achieves those results in a graceful, well-executed fashion.

Project metrics are the most commonly used test metrics. Tables and graphs showing status and trends in terms of bugs and test cases are used on most projects with any

form of independent testing. When test status is reported as part of a project and testing dashboard, project metrics are frequently found.

Project metrics provide practical, often immediately-useful insights about your current project status and trends. By benchmarking your project metrics against past projects and industry norms, you can see where your project stands in terms of current status and trends. This can give you good ideas about what kinds of project course-corrections to make to achieve optimal outcomes for your project. When multiple, balanced metrics are used properly, project metrics can present a complete picture of the project that can help you avoid making project decisions that will result in sub-optimal outcomes.

Unfortunately, people often over-rely on project metrics, and often fail to properly balance project metrics. While project metrics tell you the progress made toward project objectives, they might not give a full and balanced picture, especially in terms of product quality. I talked about balanced metrics in the first article in this series, and it's a topic I'll revisit in this article with respect to project metrics. I will talk about product metrics in the final article coming in the next issue.

It's also important to remember that project metrics measure project trends and status, and, to some extent, reflect process capabilities. As I've mentioned previously in this series of articles, it's a classic worst practice of metrics to mistake project status or process capabilities for team or individual capabilities. Most of the factors that determine project status and control a process's capabilities are under control of management, not under control of the team or individual working on that project. If such metrics are used to reward or punish teams or individuals, then they will often find ways to distort the metrics in order to maximize rewards or minimize punishments. The project metrics will cease to reflect the true status of the project, and thus the project team will lose the opportunity manage the project using those metrics.

Developing Good Project Metrics

In the first article in this series, I defined a process for defining good metrics. Let's review that process here, with an emphasis on project metrics:

1. Define test- and quality-related objectives for the project.
2. Consider questions about the effectiveness, efficiency, and elegance with which the project is achieving those objectives.
3. Devise measurable metrics, either direct or surrogate, for each effectiveness, efficiency, and elegance question.
4. Determine realistic goals for each metric, such that the quality and testing for the project will be adequate upon release.
5. Monitor progress towards those goals, determining project status and making test and project control decisions as needed to optimize project outcomes.

The typical objectives for testing and quality on a project vary, but often include finding most of the bugs and fixing the most important bugs. Let's use these to illustrate the process.

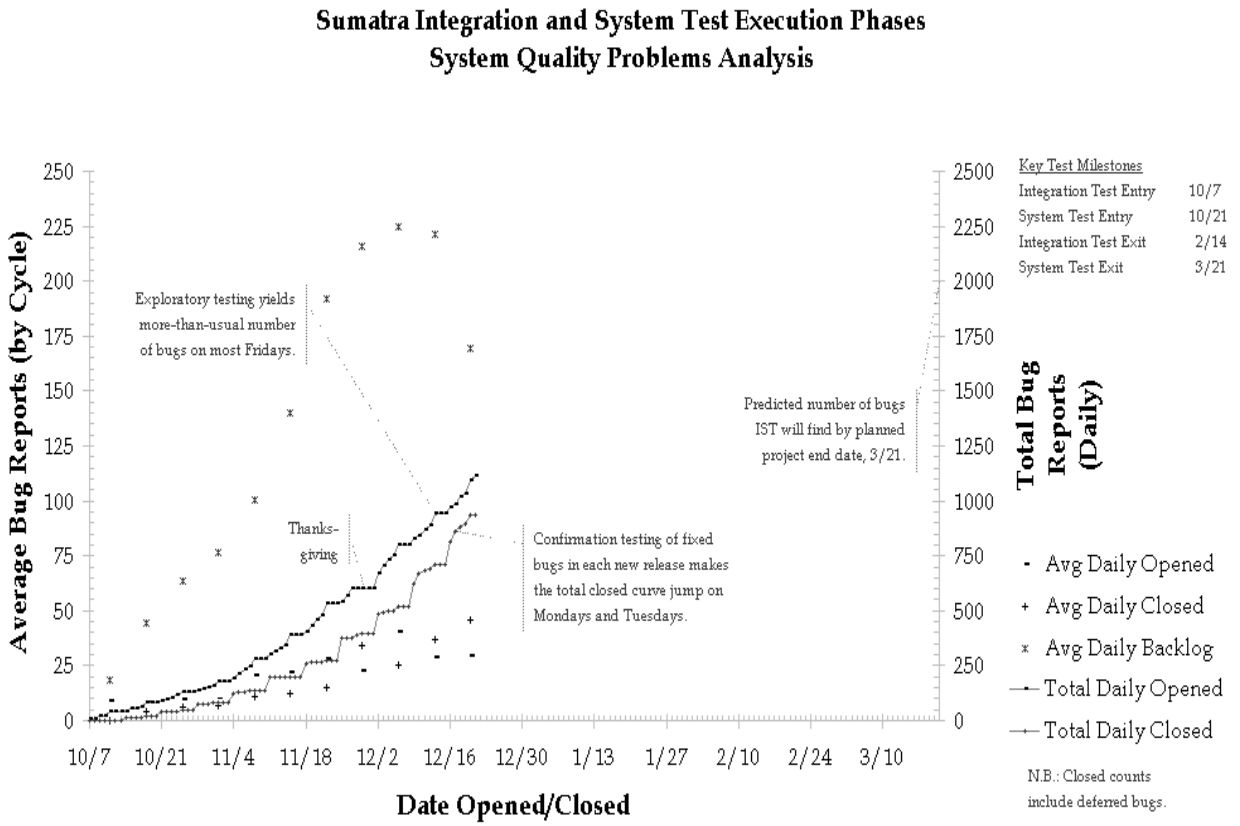

Figure 1: Bugs Opened and Closed on a Project

In Figure 1 you see an information-rich version of a chart often used on projects to understand, report, and manage progress in terms of finding and fixing bugs (defects). It is a trend chart, which means that the horizontal axis is shown in terms of time. In this case the time series starts with the day on which formal integration and system testing (IST) started and ends with the day on which formal integration and system testing is scheduled to complete.

There are five metrics shown on this chart:

- Total Daily Opened: This metric is graphed against the right axis (i.e., on the scale from 0 to 2,500). It shows the cumulative number of bug reports opened as of a given day on the project. (As the project date is currently around December 20 in Figure 1, the measurements for this metric and all other metrics stop about halfway across the horizontal axis.) As we approach the end of the project, we would expect the total daily opened measurements gradually to stop increasing,

eventually showing zero new bug reports opened per day, as the test team finishes their tests and finds no new bugs. As you can see from annotations on this chart, we expect to file about 2,000 bug reports by the planned project end date (March 21), we expect exploratory testing on Fridays to yield a larger-than-usual number of bug reports each week, and we have seen a holiday (Thanksgiving) lead to an unusual flat spot in the total daily opened bug reports.

- Total Daily Closed: This metric is also graphed against the right axis. It shows the cumulative number of bug reports resolved (either confirmed as fixed or deferred for this project) as of a given day on the project. As we approach the end of the project, we would expect the total daily closed measurements to converge toward and ultimately intercept the total daily opened measurement, as the project team resolves all the bug reports that pose an obstacle for release on this project. As you can see from the annotation on this chart, we perform confirmation testing immediately upon receipt of a test release each Monday, which leads to a jump in the number of bug reports closed at the start of each week.
- Average Daily Opened: This metric is graphed against the left axis (i.e., on the scale from 0 to 250, one-tenth the scale of the right axis). It shows the average daily number of bug reports opened for a given week. In other words, the average daily opened measurement for the week is calculated by totaling the number of bugs reports opened each day during the week, and dividing that total by seven. As we approach the end of the project, we would expect the average daily opened measurements to sink gradually toward zero, eventually showing zero new bug reports opened in the final week or two.
- Average Daily Closed: This metric is also graphed against the left axis. It shows the average daily number of bug reports resolved for a given week. In other words, the average daily closed measurement for the week is calculated by totaling the number of bugs closed each day during the week, and dividing that total by seven. If the average daily closed measurement (shown as a "+" sign) is above the average daily opened measurement (shown as a "-" sign) for the week, the number of bug reports in the backlog has decreased for the week. As we approach the end of the project, we would expect the average daily closed measurements to consistently stand above the average daily opened measurements as the backlog goes to zero, and also expect the average daily closed measurements to move gradually toward zero as well.
- Average Daily Backlog: This metric is also graphed against the left axis. It shows the average daily difference between the total daily opened and total daily closed bug reports. In other words, the average daily backlog measurement for the week is calculated by subtracting the total daily closed measurement from the total daily opened measurement, totaling those differences during the week, and dividing that total by seven. If the average daily backlog measurement (shown as a "*" sign) goes up for a week, the development team has fallen behind the test team in terms of the bug management process. As we approach

the end of the project, we would expect the average daily backlog measurements to sink gradually toward zero, eventually showing zero (or very close to zero) bug reports in the backlog in the final week.

As you can see from this example, a single chart can provide a graphical view of a number of metrics. In this case, all five metrics shown give us a sense of the progress of bug management for this project, both from the testing side (in terms of reporting bugs) and from the wider project team side (in terms of resolving bugs).

Finally, note that the metrics shown in Figure 1 are direct metrics, not surrogate metrics. We are interested in progress in terms of finding, fixing, and managing bugs on the project, and that's what we're measuring. Remember from the first article that a surrogate metric is one that measures something related to the area of interest. Since we can find direct metrics which are easily measurable in this case (at least given any decent bug management tool and process), we need not develop a surrogate metric.

Understanding Balance in Project Metrics

Project metrics are useful to understand and manage progress on a project. For example, if you refer back to what I said about each of the metrics shown in Figure 1, you'll notice that I described what we would expect to see happen with each metric as we approach the end of the project. If we do not see those trends in the metrics, then that tells us that we have a significant barrier to successful completion of the project, in terms of bugs. For example, if the total daily opened and average daily opened metrics do not show a trend towards zero newly discovered bugs, then we have a situation where the quality of the system under test is not improving. As another example, if the total daily closed and average daily backlog metrics do not show a trend towards zero unresolved bugs, then the quality of the system is known to be insufficient for release.

However, it's important to understand that just because these metrics *do* show the desired trend, they are not sufficient by themselves to confirm that the project can successfully succeed. Remember, the metrics shown in Figure 1 relate only to the project objectives of finding most bugs and fixing the most-important of those bugs. There are other typical test-related project objectives, such as: ensuring that all test cases have been run and currently pass (at least the most important test cases); building confidence that the system will work properly (at least the most important use cases); and, achieving a sufficiently low residual level of quality risk (at least the most important quality risks). It is possible that our bug-related test objectives are met, but one or more of the other objectives is not met.

Let me illustrate this with an example. Assume that we want to evaluate progress in terms of the objective of ensuring that all test cases have been run and currently pass. Perhaps we can use a chart such as that shown in Figure 2? I'll explain how to read this chart, and you'll see how this helps to balance metrics based only on bugs.

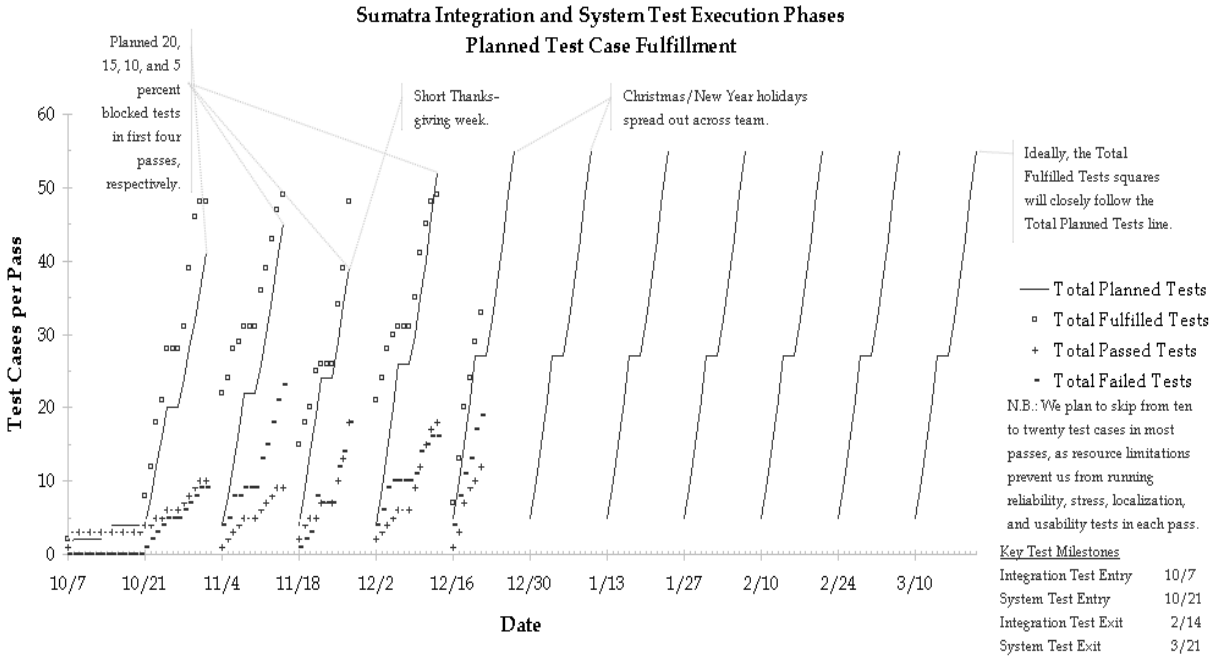

Figure 2: Test Case Completion On a Project

Figure 2 shows four metrics:

- Total Planned Tests: This metric shows the planned test cases expected to be completed as of the end of each day. On this project, testing is broken into two-week intervals, called *test passes*, meaning a sequence of test cases grouped together, based on the level of risk or some other logical grouping. The lines go upward as each two-week period of testing continues. Each test passes' completion metric is counted separately, which is why the lines restart towards the bottom of the chart at the beginning of each two-week period and ascend towards the top of the chart at the end of each two-week period.
- Total Fulfilled Tests: This metric shows the actual test cases which are fulfilled within a pass as of the end of each day. A test case is considered fulfilled if it is either: executed and has passed; executed and has failed; or, deliberately not executed (i.e., skipped). We would expect that, especially as the project proceeds, the boxes representing the total fulfilled test cases would closely track or even exceed the total planned test cases on any given day.
- Total Passed Tests: This metric shows the actual test cases which are executed and passed (i.e., did not find any bugs) within a pass as of the end of each day. We would expect that, especially as the project proceeds, the plus signs ("+") representing the total passed test cases would closely track and eventually equal the number of total fulfilled test cases on any given day.
- Total Failed Tests: This metric shows the actual test cases which are executed and failed (i.e., did find one or more bugs) within a pass at of the end of each day.

We would expect that, especially as the project proceeds, the minus signs ("-") representing the total failed test cases would subside and eventually tend towards zero during the final test pass.

As you see again, a single chart provides a graphical view of a number of metrics. In this case, the comparison of three of the metrics (total fulfilled test cases, total passed test cases, and total failed test cases) against another one of the metrics (total planned test cases) gives us a sense of the progress of test case fulfillment for this project.

To see how this chart balances Figure 1, consider this example. Suppose that, as the project continues, the metrics shown in Figure 1 display the expected trends, but the metrics shown in Figure 2 show a continuing problem in that total fulfilled test cases and total passed test cases do not match the total planned test cases. In this case, the project is not meeting the objective of ensuring that all test cases have been run and currently pass. We thus have a more balanced picture of testing progress on this project, a picture which tells us that something is broken in terms of completing our test cases.

Since Figure 2 is a trend chart—i.e., a chart that shows the trend of one or more metrics over time—it cannot tell us *why* we have a problem. Trends are not causes; they are outcomes with a temporal dimension. To see why, consider a trend chart that shows the increasing incidence of cancer in a nation's population. Such a trend could be the result of increasingly unhealthy behaviors by the people of that nation, or it could be the result of increasing pollution and environmental hazards in the nation's environment, or it could simply be the result of people living longer and thus succumbing more frequently to diseases of the elderly such as cancer. Beware of jumping to quick conclusions about causes for trends based only on a trend chart; it's important to do some further analysis to determine why trends evolve.

To return to the question of balance, let me point out that, even with Figure 2 juxtapose against Figure 1, the picture is not yet entirely balanced. We could be achieving our objectives in terms of bugs found and fixed, and test cases planned, executed, and passed, but at the same time not adequately building confidence in the product or reducing quality risk to an acceptance level. To show you how to do that, I'll discuss two types of coverage metrics in the next article (which is about product metrics). In some cases, test metrics can be both product and project metrics, which makes sense when you consider that some important project objectives concern the current quality of the product.

Project Metrics for Internal Test Management

So far, the two project metrics charts shown are the type of metrics that a test manager would use not only to assess progress, but also to report progress to project team members. (In other words, they are likely candidates for project dashboards, as discussed in the previous article.) Some metrics are more useful internally, for the test manager and the test team, to understand whether the test execution activities are

proceeding effectively and efficiently. Note that the distinction here is that internal metrics are ones we use within the test team, while the other metrics discussed so far in this article are external metrics, in that we might report them to people outside the test team.

Suppose a test manager has the internal project objective of progressing through the planned test cases in the planned number of hours. The efficiency question is, "Is it taking us more effort or less effort to execute our test cases." Figure 3 shows two metrics:

- Planned Test Hours: This metric shows the number of test case execution hours planned for each day. It can be calculated based on the total number of hours of test case execution planned per week, divided by five (i.e., the number of workdays per week). The gaps between the lines representing the planned test hours indicate that no test cases are planned over the weekends. Note that test case execution hours typically are less than the total number of hours worked by the testers per day, because testers have to spend time attending meetings, writing and responding to e-mails, and other directly work-related tasks that nevertheless do not result in progress through the test cases.
- Actual Test Hours: This metric shows the number of test case execution hours actually achieved each day. It is measured by gathering information from the testers, at the end of each day, about how many hours they spent actually executing test cases. We would expect that the number of actual test hours would closely track the number of planned test hours, at least within some margin of normal variation. While the dashed lines indicate planned test hours, the dotted lines above and below the dashed lines indicate this margin of normal variation.

By comparing actual test hours against planned test hours, within the margin of normal variation, we can assess whether test execution is proceeding efficiently. However, remember that the question we are trying to answer is, "Is it taking us more hours or fewer hours to execute our test cases." This means that Figure 3 is not, by itself, balanced, in that it can't tell us whether we are indeed completing the test cases. Thus, Figure 3 must be considered in the context of Figure 2 (or some similar chart showing a similar set of metrics), which can tell us where we stand on test execution.

I should also mention that Figure 3 displays not only project metrics, but also process metrics. In other words, it is also a reflection of process capability. Consider this: It's possible that the planned test hours metric in the chart might reflect an unrealistic expectation about how many hours of test execution that the testers can achieve in a given day, given the current test execution process. In that cause, the actual test hours might reflect true process capability, and the difference between actual and planned test hours might not be due to project exigencies and deviations from normal process capability. To expand on what I mentioned above with respect to causality, a trend chart cannot tell us why a trend occurred; in this case, a trend in test execution

efficiency might be due to project-related incidents which affect test execution efficiency for this project, or it might be due to process realities that dictate what the true process capability is.

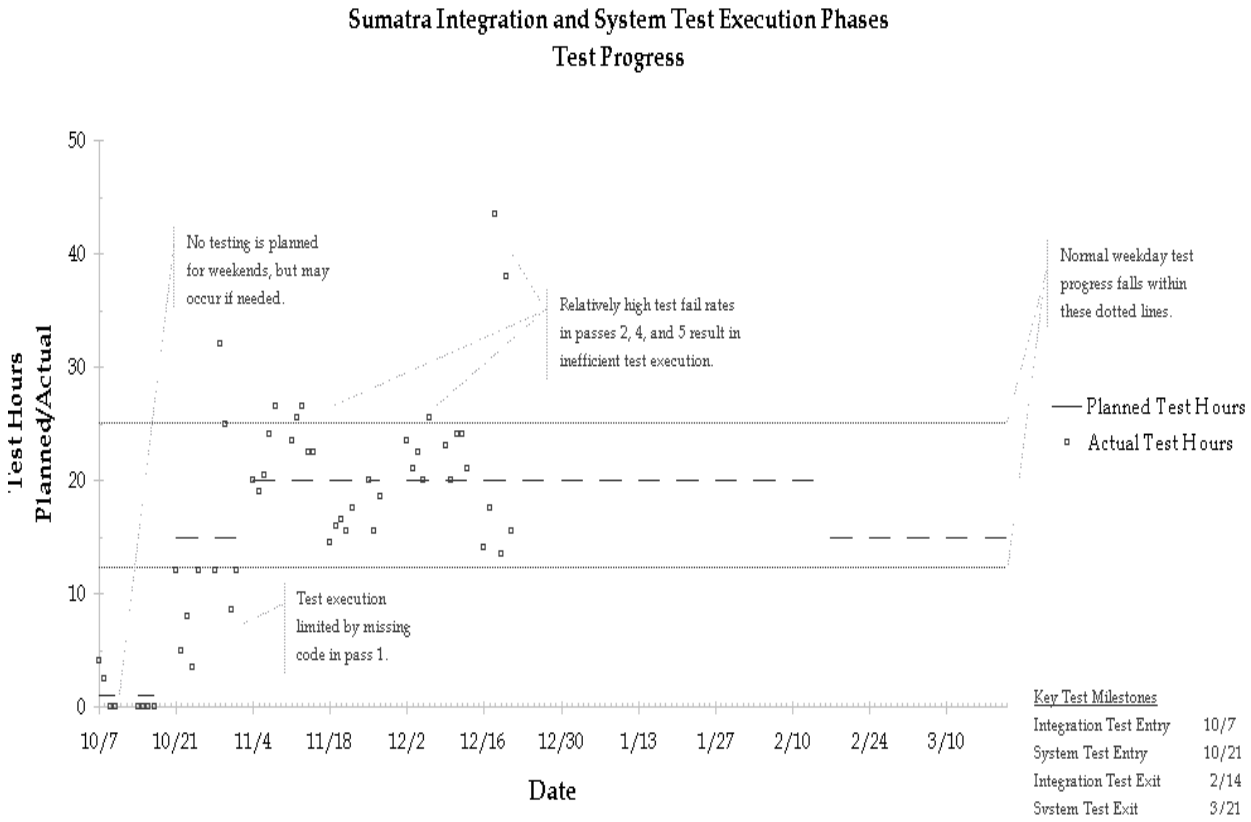

Figure 3: Test Hours Spent on Test Execution on a Project

Moving on to the Product

In this article, I discussed the use of project metrics. I've illustrated the application of the metrics development process discussed in the first article, as applied to project metrics. Using examples of project metrics from industrial use, we've seen how these metrics allow us to understand the progress of testing on a project, and to make decisions about how to adjust the testing to optimize the achievement of project objectives. We've discussed the use and potential misuse of project metrics, which are just as subject to misuse as the process metrics discussed in the previous articles.

In the final article in the series, we'll look at one more specific type of metrics, namely product metrics. These testing metrics are often forgotten, but without product metrics you cannot understand the true quality status of the product. The next article will give you some ideas on how to use test product metrics properly, as well as wrapping up this series of articles. See you in the next issue!

Author Biography

With over a quarter-century of experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For sixteen years, RBCS has delivered consulting, outsourcing and training services to clients ranging from Fortune 20 companies to start-ups. Rex has published eight books which have sold over 50,000 copies, including Japanese, Chinese, Indian, Hebrew, Russian and Spanish editions. He has written over forty articles, presented hundreds of papers, workshops, and seminars, and given about seventy-five speeches at conferences and events around the world. Rex is also the immediate past President of the International Software Testing Qualifications Board and the American Software Testing Qualifications Board. Rex may be reached at rex_black@rbcs-us.com.

