



Metrics for Software Testing: Managing with Facts: Part 1: The Why and How of Metrics

Provided by Rex Black Consulting Services (www.rbc-us.com)

Introduction

At RBCS, a growing part of our consulting business is helping clients with metrics programs. We're always happy to help with such engagements, and I usually try to do the work personally, because I find it so rewarding. What's so great about metrics? Well, when you use metrics to track, control, and manage your testing and quality efforts, you can be confident that you are managing with facts and reality, not opinions and guesswork.

When clients want to get started with metrics, they often have questions. How can we use metrics to manage testing? What metrics can we use to measure the test process? What metrics can we use to measure our progress in testing a project? What do metrics tell us about the quality of the product? We work with clients to answer these questions all the time. In this article, and the next three articles in this series, I'll show you some of the answers.

Why Should We Have Metrics?

Sometime I hear people asking why metrics are necessary, or worse yet disparaging metrics with smug comments like: "Not everything that you can measure matters, and not everything that matters can be measured." To me, such remarks are like questioning the value of literacy, or saying that reading is unimportant because you don't need to read to appreciate good art.

Metrics allow us to measure attributes. Metrics allow us to understand. Metrics allow us to make enlightened decisions. Metrics allow us to know whether our decisions were the right ones, by assessing the consequences of those decisions. Metrics are rational.

Plus, you really don't have much alternative to metrics usage. The other option is to base your understanding, decisions, and actions on subjective, uninformed opinions. This is not a sound basis for management.

You might be thinking; "I'm a reasonable person, and I make all sorts of smart and reasonable decisions in my everyday life without metrics." Well, maybe. First off, you might have grown so accustomed to all the metrics we have around us that you didn't notice them. When we drive, we refer constantly to a key metric: speed. When we shop, we mostly use the metric of price. In many situations, when you find yourself without the usual metrics, you might feel lost.

Second, when people make decisions or reach conclusions without metrics, based on what sounds reasonable, they can be wrong. My favorite example of this comes from the Greek philosopher, Aristotle. Aristotle was a smart fellow, and he said a lot of smart things. However, he also said that heavier objects fall faster than lighter objects.

That sounds reasonable, and anecdotal evidence like feathers and stones are all around us. Two thousand years later, though, Galileo dropped two cannonballs of very different weights from the Leaning Tower of Pisa. Both hit the ground at the same time. Simple experiment. Simple metric. Two thousand years of misguided thought overturned with a single thud.

In my consulting work, I often tell clients that the most dangerous kind of mistake is the mistake that sounds reasonable. Something that is wrong and that sounds stupid is harmless, because people will reject those statements out of hand. Reasonable-sounding mistake, that just might trick people. In fact, we've seen that happen.

Here's an example. We do a number of test process assessment engagements for clients, and one of these clients makes complex industrial-control systems that run oil refineries, pharmaceutical plants, and other critical equipment. In our assessment, we found that they had a very high rate of bug report rejection. Bug report rejection occurs when the report turns out to describe correct behavior, rather than the symptom of a bug. When the bug reporting process is working well, the bug report rejection rate should be under five percent. In this case, we found the client had about 20% of bug reports rejected as not due to faulty behavior. When I asked why the rate was so high, almost everyone believed that the reason was insufficient end-user experience using industrial controls.

It sounds reasonable, huh? People who don't understand complex systems might not draw the right conclusions about what constitutes correct behavior, right? Well, it turns out that the data easily disproved this reasonable – but mistaken – opinion. I created the scatterplot shown in Figure 1. The scatterplot shows the percentage reports rejected, on a tester-by-tester basis, versus the number of years of actual plant experience each tester had. As you can see, the R^2 value – which measures the level of statistical correlation – is very close to zero. So much for reasonable.

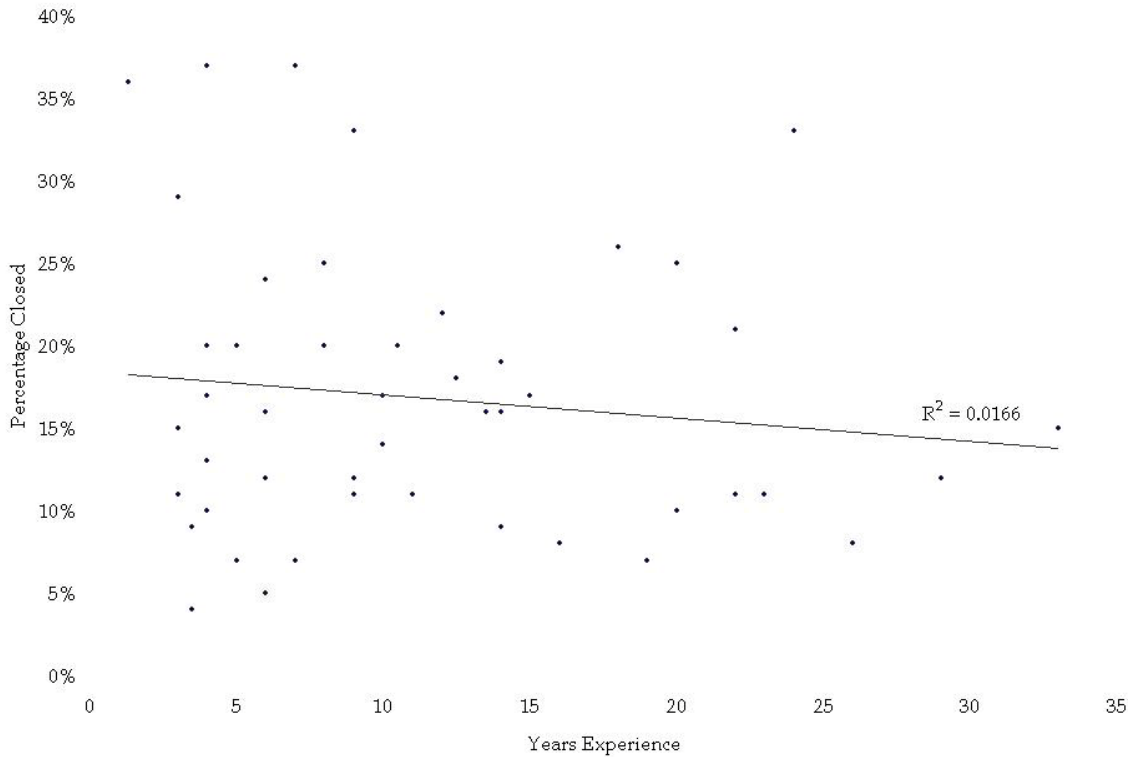


Figure 1: Data Disproves Widely-held Opinion

Metrics are valuable whatever we are doing, but I think they are particularly important for testing. This is true because testing by itself, in isolation from the rest of the project, has no value, but it produces potentially valuable information. In order to obtain the value, this information must be generated and communicated effectively. That involves some form of testing metrics.

Effective communication is communication that serves a purpose. There are three fairly common goals of communication of test information, and all three are enhanced by metrics.

We might want to notify people of the status of testing. For example, we might want to make people aware of the bug backlog that has accumulated. In such a case, it's more appropriate and powerful to say, "We have 24 bugs remaining to close," than to say, "There are still bugs in the backlog."

We might enlighten people as to the impact of some attribute of the process. For example, we might want to help people understand that we are working inefficiently because many bug fixes fail confirmation testing. It's better to be able to measure and report the number of lost person-hours resulting from confirmation test failures than to simply exclaim, "It's very frustrating and inefficient to deal with all these lousy bug fixes that the programmers send us!" Of course it's not always possible to evaluate an exact number of lost person-hours, however even an approximate value will shed some light on the situation we are facing.

We might also want to influence people to choose a particular course of action. Going back to the example two paragraphs ago, where we have a large bug backlog, we might show a breakdown of bugs by severity, and then propose a bug triage meeting to defer unimportant bugs in order to focus attention on the more important ones.

For our clients that are following best practices in their use of metrics, we see that some metrics are reported regularly as part of status reporting. These are sometimes called dashboards. They can be process, project, or product focused. Such dashboard metrics might have as their goals notification, enlightenment, and/or influence. Other metrics are reported as needed, after some analysis of a situation that has arisen. Such metrics would more commonly have enlightenment (why did the situation happen?) and influence (what should we do about the situation?) as their goals, than merely notification.

How Should We Develop Metrics?

So far, we've seen why metrics are useful, how metrics help to deliver the value of testing, and how metrics can serve specific communication goals. What metrics should we use, though? Is it enough to simply adopt the metrics that a tool like Quality Center produces? In my experience, such test management tool metrics are not sufficient, and in some cases are counterproductive. Such tools produce large amounts of very tactical metrics that can prove useful to test managers, but which are typically overwhelming and even misleading to people without a testing background. While test management tools are valuable to collect the raw data behind metrics, you should use a top-down approach for defining metrics, not a bottom-up approach.

By "bottom-up approach" I mean letting the tool define the metrics you will report. By "top-down approach" I mean starting with a clear picture of the objectives you are trying to achieve, and then deriving the metrics from that.

Identifying the objectives for testing and quality can prove challenging for some organizations, because not many organizations are used to thinking about what the testing and quality objectives are. I realize this statement sounds strange, but it is true. Ask yourself: Do you have well-defined, realistic, documented, agreed-upon objectives for your testing process? When we start working with clients, the answer is usually "no".

Typical high-level objectives for the test process as a whole are:

- Find bugs, especially important ones
- Build confidence in the product
- Reduce risk of post-release failures
- Provide useful, timely information about testing and quality

You might have other objectives, and that's fine.

Given a defined set of objectives, we can ask three types of questions about the degree to which we achieve those objectives:

- To what extent are we effective at achieving those objectives?
- To what extent are we efficient at achieving those objectives?
- To what extent are we elegant at achieving those objectives?

Let's define what these concepts of effectiveness, efficiency, and elegance mean with respect to the achievement of objectives.

Effectiveness has to do with producing a desired result, in this case the objective. Efficiency has to do with producing that desired result in a way that is not wasteful and, ideally, minimizes the resources used. Of course, at some point, trying to increase efficiency starts to reduce effectiveness, as anyone who has driven a small, highly fuel-efficient vehicle knows.

What about elegance? Elegance has to do with achieving effectiveness and efficiency in a graceful, well-executed fashion. Elegance impresses. Elegance work resonates as professional, experienced, and competent.

Some people might ask, if we are effective and efficient, why should we care about elegance? Consider this example. Let's suppose that you go into a café to get a cappuccino. You are in a hurry, and want to quickly get your caffeine fix and go. The line is short and the cost is low. The wireless signal is strong and free, so for the limited time you are waiting, you can get some work done. Within two minutes, you have your cheap cappuccino, it tastes excellent, and you are out the door with your to-go cup in hand. The café effectively and efficiently satisfied your objective. Are you happy?

Maybe not. What if the cashier was rude and lazy, almost overcharging you for the drink until you pointed out her math error? What if the man making your drink was dirty, smelled bad, and had long, greasy hair that was clearly shedding into people's drinks? What if the place as a whole was not very clean or very pleasant to look at? In such a situation, you would not consider the place a very elegant way to satisfy a caffeine craving.

With the need for elegance established, let's return to the issue of metrics. We should devise at least one metric each to determine the extent of our effectiveness, our efficiency, and our elegance. This metric should be something we can actually measure, of course. People having an elegant experience probably have a dopamine release in their brains, but that's not likely to be something you can check easily. A better idea would be a stakeholder satisfaction survey using a Likert scale (e.g., asking satisfaction levels ranging from very satisfied to very dissatisfied).

In some cases, it's very difficult to measure something directly. In such situations, you can use a surrogate metric. As an example, suppose I gave you a tape measure and sent you into a parking garage to weigh the vehicles in that garage. Could you do it? Well, you can't directly weigh the vehicles, because you don't have a scale. However, you could use the tape measure to calculate the volume of each vehicle. You could use the volume as a surrogate metric for weight, simply by making the simplifying assumption of relatively constant density. In fact, if

Metrics for Software Testing: Managing with Facts

you were the owner of just one of the vehicles, you could use the owner's manual in the car to determine the actual weight. This would give you a known density for one vehicle, and you could then use that to calculate (via the volume) the weight in kilograms or pounds for the rest of the vehicles in the garage.

I'll give one example of each type of metric, direct and surrogate, in just a few paragraphs. In subsequent articles in this series, we'll see many examples of metrics, including direct and surrogate metrics.

It's important to say that it's not enough to just have a metric. We need to know what constitutes a good measurement for that metric. So, once the metrics are defined, we should set a goal for each metric. One way to set the goals is to measure where you stand now. (This is sometimes called *baselining*.) Another way to set the goals is to compare yourself to industry averages or best practices. (This is sometimes called *benchmarking*.)

One way *not* to set goals is to pick arbitrary, extreme values, though I'm afraid this does happen. For example, we have seen situations where tester were expected to find 100% of all bugs, while at the same time programmers were expected to have zero bugs in their code. Both of these goals were instituted in the testers' and programmers' annual performance evaluations, respectively. In this case, the managers had actually made two serious mistakes. They violated best practices for setting goals for metrics and violated the rule that process metrics should not be used for individual performance appraisal.

With proper goals in place, you should think about exceeding those goals. What improvements could you implement that would move the metrics towards higher levels of effectiveness, efficiency, or elegance? It's certainly true that at some point any process will have reached adequate levels of optimization, but it's also true that it's very rare for us to work with clients on metrics programs and find that everything is perfect the first time we baseline the processes, projects, and products.

So, we can summarize the process of deriving metrics as follows:

1. Define objectives.
2. Consider questions about the extent of effectiveness, efficiency, and elegance with which we realize the objectives.
3. Devise measurable metrics, either direct or surrogate, for each effectiveness, efficiency, and elegance question.
4. Determine realistic goals for each metric.
5. As appropriate, implement improvements that improve effectiveness, efficiency, or elegance as measured by the metrics.

With the process clear, let's look at two examples of metrics devised following this process.¹

First, let's look at one of the common objectives for testing, finding bugs. On a project, one of the key questions is whether we are finished finding new bugs. (This is often included as an

¹ For a complete discussion of this process, you can read my chapter in the book *Beautiful Testing*.
Metrics for Software Testing: Managing with Facts

exit criterion in test plans.) As a metric, we can plot the trend of bug discovery over time during test execution. An example of such a metric, at the end of the project, is shown in Figure 2. Our goal is to see the flattening of the cumulative bug opened curve (the upper line in the graph).

Let me point out that we also have another project objective, the resolution of known bugs, which is also shown in this metric (the lower line in the graph). It's not unusual to find ways to combine metrics on a single graph or table, and this combination can be quite useful to compare and contrast process, project, or product attributes that are illustrated by the two or more metrics shown.

This chart also helps nudge us towards two obvious improvements. If we were to find (and resolve) bugs earlier, we could finish the project earlier. So, how can we shift the bug opened and resolved curves left, towards the start of the project? How can we shift the total number that we discover downward on the vertical axis? Fewer bugs, found and resolved earlier: that sounds smart, doesn't it?

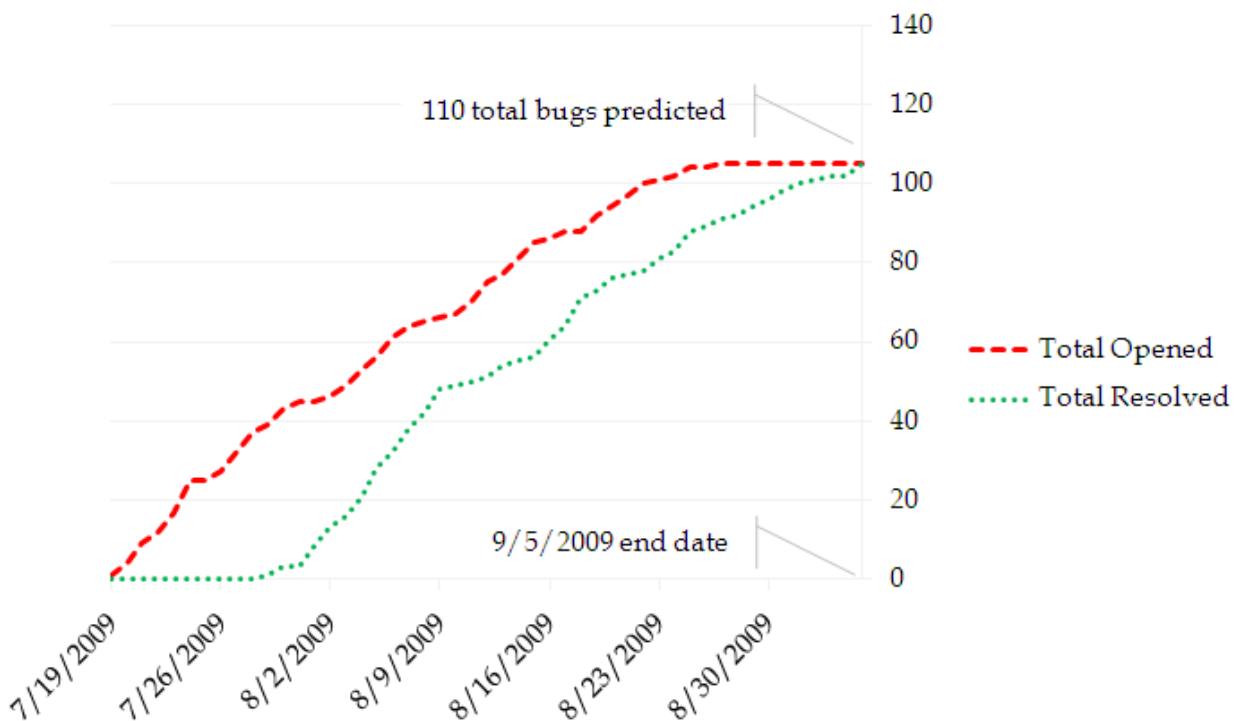


Figure 2: Bug open and resolution trends

Next, let's look at another common objective for testing, building confidence. We would want to achieve a significant level of confidence prior to releasing software to our customers. However, how can we measure confidence directly, as confidence is a state of mind? For

confidence, we can use coverage as a surrogate metric: the more thoroughly tested the product is, the more confident we can be that the system has no surprises in store for us (or the customers or users) after release. Now, coverage is a tricky concept, because coverage has multiple dimensions, including code coverage, design coverage, configuration coverage, test design technique coverage, requirements coverage, and more. Certainly, for higher levels of testing such as system testing and acceptance testing, a key question is whether any requirements have identified failures. So, our metric can include three elements:

- How many requirements are completely tested without any failures?
- How many requirements have failures?
- How many requirements are untested?

These are typically measured as percentages, as shown in Table 1. At the end of testing, the goal is to test 100% requirements, with no known must-fix failures at the end. As an improvement, we can look at ways to reduce the percentage of requirements that fail in testing when first tested.

Requirements Area	Currently Untested	Tested and Failed	Tested and Passed
Functionality	7%	3%	90%
Usability	25%	17%	58%
Reliability	0%	17%	83%
Performance	5%	10%	85%
Installability	7%	13%	80%

Table 1: Requirements Coverage by Area

What Else is True of Good Metrics?

Lots of organizations have metrics programs, but the metrics are not always very good. This is not always due to a failure to follow a good process for developing metrics, such as the one outlined above, though certainly bad metrics-development processes are a major contributor. What else can we say about good metrics?

Certainly, we want our metrics to be simple and effective. *Simple* means not just simple to gather and calculate, but also simple to understand. *Effective* means that the metric is obviously and actually connected to parts of the software process in such a way that we know what actions to take to move the metric in the desired direction. This property is something one of my clients refers to as the “so what?” question for metrics.

Since we were just on these topics, metrics should be efficient and elegant, too. *Efficient* means that we can produce the metric without an excessive amount of work; the effort required to produce an efficient metric is repaid by the value we receive from that metric. *Elegant* means that the metric is seen by the audience as a pleasing and smart way to present the information.

So, what is true of metrics programs that have simple, effective, efficient, and elegant metrics? Such a set of metrics are useful, pertinent, and, especially, concise. While it can be tempting to measure absolutely everything, you should avoid too large sets of metrics. Such metrics will prove too difficult to measure in practice (and thus inefficient) and usually very confusing to participants (and thus inelegant). To be clear, there is value in *considering* a large variety of metrics when first setting up your metrics program; however, once implementation and regular measurement starts, you should settle on a limited number.

That said, it's also important that the metrics be sufficient in number and diverse enough in perspectives to balance each other. For example, consider again the bug trend chart shown in Figure 2. I said that we want the cumulative opened curve to flatten, and for the cumulative resolved curve to intercept the cumulative opened curve, as we get to the end of testing. That's true, but by itself is out of balance, because we might have stopped finding new bugs when our testing is completely blocked. In such a case, notice that the requirements coverage metric I mentioned as the second example balances this bug metric, because the requirements coverage metric will be stuck below 100% tested and passed.

To make the metrics simple to gather, calculate, track, and present, you must consider the implementation of the metrics. Automated tool support can be very helpful in this regard. However, be careful, because it's easy, once the tools get involved, to let the built-in metrics of the tool determine what you will measure (which is back to the "bottom-up" mistake I mentioned earlier).

Implementation of the metrics should also consider the proper way to track and present a given metric, because proper presentation is a major factor in making a metric simple and effective. You have three general options. Metrics can be presented as snapshots of status at a moment in time, as shown in Table 1. Metrics can show trends emerging over time, as was the case in Figure 2. Metrics can also show the analysis of causes and relationships between factors that influence testing and quality outcomes, as we saw in Figure 1. The formulation of clear objectives and questions related to them, as discussed earlier, should help you make the choice here. However, if in doubt, try various options and see which one suits best to your process.²

Making the metrics simple to understand is not likely to happen without some education. Part of a successful metrics program is ensuring uniform, agreed interpretations of the metrics. Clear understanding of what the metrics tell us helps to minimize disputes and divergent opinions about various measures of outcomes, analyses, and trends that are likely to occur when we measure projects, processes, and products. Remember that reporting of metrics should enlighten management and other stakeholders, not confuse or misdirect them.

So, when presenting metrics, be sure to provide objective analysis, tempered with appropriate and balance subjective interpretation. This is especially true when trends emerge that could allow for multiple interpretations of the meaning of the metrics. Of course, we want to avoid

² It's worth reading Edward Tufte's book, *The Graphical Display of Quantitative Information*, for more ideas on how to create good charts, graphs, and tables.

complex and ambiguous metrics that tend towards such confusion, but the problem is not merely one of metrics design and stakeholder education.

When using metrics, we have to be aware of and manage the tendency for people's interests to affect the interpretation they place on a particular metric. Three psychological dynamics tend to create problems in the use of metrics here. The first is confirmation bias, which is the tendency to accept facts and opinions that confirm our own existing opinions, and reject other contradictory facts and opinions. For example, the project manager who is sure the product will release on time (and whose bonus depends on it) will have some significant confirmation bias with respect to test results showing a large and growing backlog of bugs.

The second is cognitive dissonance, which are the feelings of confusion, anxiety, frustration, and even anger that result from trying to simultaneously have incongruous beliefs, attitudes, and understandings. The project manager who starts to understand the implications of the bug backlog will soon experience cognitive dissonance.

This leads to the third psychological dynamic, which is transference. In transference, a person transfers how they feel about some particular situation onto someone or something else. In this case, the project manager might transfer their anger over the delay in release onto the test manager who is reporting the test results, since it was that test results which made the project manager unhappy. Confirmation bias, cognitive dissonance, and transference are all common human psychological dynamics, and you certainly cannot change human nature. However, you should be aware of how these psychological dynamics will affect people's response to metrics.

When thinking about using metrics for reporting purposes, keep the goals in mind: good testing reports based on metrics should help stakeholders and managers improve processes, guide the project to success, and manage product quality. You should check with the people who are using the metrics to make sure that the metrics are working for them. I recommend to use the Likert scale survey that I mentioned early to assess the usefulness of the metrics for the stakeholders.

Moving on to the Process

In this article, I offered a number of general observations about metrics. We've seen the importance of using metrics to manage testing and quality with facts. We've looked at the proper way to develop metrics, top-down starting with objectives rather than bottom-up starting with tools. We've seen two examples of metrics for testing. We've also looked at some rules for recognizing a good set of metrics.

In the next three articles in the series, we'll look at specific types of metrics. We'll start with process metrics, because these metrics are the least used and least understood of the three types. See you in the next issue!

Author Biography

With over a quarter-century of experience, Rex Black is President of RBCS (www.rbc-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 six books which have sold over 50,000 copies, including Japanese, Chinese, Indian, Hebrew, and Russian 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@rbc-us.com.