Metrics for Software Testing: Managing with Facts
Part 4: Product Metrics

Introduction

In the previous article in this series, we moved from a discussion of process metrics to a discussion of how metrics can help you manage projects. I talked about the use of project metrics to understand the progress of testing on a project, and how to use those metrics to respond and guide the project to the best possible outcome. We looked at the way to use project metrics, and how to avoid the misuse of these metrics.

In this final article in the series, we'll look at one more type of metric. In this article, we examine product metrics. Product metrics are often forgotten, but having good product metrics helps you understand the quality status of the system under test. This article will help you understand how to use product metrics properly. I'll also offer some concluding thoughts on the proper use of metrics in testing, as I wind up this series of articles.

The Uses of Product Metrics

As I wrote above, product metrics help us understand the current quality status of the system under testing. Good testing allows us to measure the quality and the quality risk in a system, but we need proper product metrics to capture those measures. These product metrics provide the insights to guide where product improvements should occur, if the quality is not where it should be (e.g., given the current point on the schedule). As mentioned in the first article in this series, we can talk about metrics as relevant to effectiveness, efficiency, and elegance.

Effectiveness product metrics measure the extent to which the product is achieving desired levels of quality. Efficiency product metrics measure the extent to which a product achieves that desired level of quality results in an economical fashion. Elegance product metrics measure the extent to which a product effectively and efficiently achieves those results in a graceful, well-executed fashion.

In spite of the usefulness of product metrics, these are not always measured during testing. All too often, test professionals rely on project metrics. As discussed in the previous article, project metrics can tell us whether the testing is on target to achieve the plan. However, they cannot reliably tell us about the quality of the system.

Suppose that I give you the following information:

- 95% of the tests have been run.
- 90% of the tests have passed.
- 5% of the tests have failed.
- 4% of the tests are ready to run.
- 1% of the tests are blocked.

Assume further that we are right on schedule in terms of the test execution schedule. Does this tell us good news or bad news? It's not possible to say. If the 10% of the tests which are failed, queued up for execution, or blocked are relatively unimportant, and the defects associated with

the 5% of failed tests are also unimportant, this could be good news. However, if some of those tests relate to important test conditions, or if the defects associated with the failed tests relate to key quality risks for the product, then we could be in serious trouble, especially given that very little time remains.

So, we need product metrics to tell us, throughout test execution, whether the quality is on track for a successful delivery. After release of the software, we need product metrics to verify the conclusions we reached during test execution. We need to include those product metrics in our testing dashboards. This allows us to balance our project metrics, as discussed in the previous article, giving ourselves and other project participants and stakeholders a complete view of project status.

Testing product metrics are typically focused on the quality of the system under test. So, it's important to remember—and to remind everyone who sees these metrics—that the role of testing is to *measure* quality, not to *directly improve* quality. The product metrics reflect the totality of the software process' quality capabilities and the entire project team's efforts towards quality. The software process and its quality capabilities are largely determined by management, and the test team cannot control the behavior of others on the team with respect to quality, so product metrics measure attributes outside the control of testing. So, as I've mentioned throughout this series, such metrics should not be used to reward or punish teams or individuals, especially the testers. If these product metrics are misused, people will find ways to distort the metrics in order to maximize rewards or minimize punishments. The product metrics will then give a distorted— typically an overly optimistic—view of product quality status, robbing the project team of the ability to manage the quality of the product. Disastrous release decisions may ensue.

Developing Good Product Metrics

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

1. Define test coverage and quality-related objectives for the product.

2. Consider questions about the effectiveness, efficiency, and elegance with which the product 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 we can have a high level of confidence in the quality and test coverage for the product prior to release.

5. Monitor progress towards those goals, determining product status, and making test and project control decisions as needed to optimize product quality and test coverage outcomes.

The typical objectives for test coverage and quality for a product vary, but often include ensuring complete coverage of the requirements.[1] A relevant effectiveness question here is: Have we built

---

1 " Analytical requirements based test strategies have their strengths and weaknesses, and are best used in a blend with other test strategies such as risk based and reactive. See my book *Advanced Software Testing: Volume 2* for more information on test strategies.

a system that fulfills the specified requirements? Let's look at a metric related to requirements coverage to illustrate the process.

When using an analytical requirements based test strategy, every requirement should have one or more tests created for it during test design and implementation, with bi-directional traceability maintained between the tests and the requirements so that complete coverage can be assured. During test execution, we run these tests, and we can report the results in terms of requirements fulfilled (i.e., all the tests pass) and unfulfilled (i.e., one or more tests fail) using the traceability.

What are realistic goals for requirements testing that will ensure adequate quality and test coverage? When following a requirements based test strategy, every requirement should be tested prior to release. It's up to business stakeholders to determine whether some of those requirements can be unfulfilled (i.e., tested with known failures) when the product is released. Of course, releasing a product with unfulfilled requirements, even the less important requirements, compromises the quality of the product. I will return to the issue of measuring adequate quality towards the end of this section.

Requirements Area	# Reqs	% Tested	% Pass	% Fail	% Block
Browsing	57	56	49	7	2
Checkout	28	96	96	0	0
Store Management	77	25	20	5	0
Performance	15	100	100	0	0
Reliability	12	0	0	0	8
Security	22	0	0	0	0
Usability/UI	27	100	48	52	0

Table : Requirements Coverage Table

We can use a metrics analysis such as that shown in Table to monitor progress towards these goals of complete testing and fulfillment of the requirements. Based on such a table, we can understand the quality of the product and make test and project control decisions to help us achieve the best possible test coverage and quality outcomes.[2]

In Table , you see a detailed table that shows where quality stands on the major requirements groups for an e-commerce site, and the status of each requirement in each group. Note that this is not showing test counts; the status of each test has been traced back to the associated requirement. Based on this analysis, the status of each requirement determined as follows:

- If all of the tests associated with the requirement have been run, that requirement is classified as tested.

- If all of the tests associated with the requirement have been run and have passed, that requirement is also classified as passed.

---

2 "   Attentive readers will notice that this table is a more detailed version of a requirements coverage table shown in the first article in this series.

- If all of the tests associated with the requirement have been run, but one or more have failed, that requirement is instead classified as failed.

- If any of the tests associated with a requirement are blocked (e.g., by known defects, test environment problems, staff shortages, etc.), that requirement is classified as blocked.

Based on these classifications, what can we conclude from this table, and what control actions might we need to take?

- Browsing has only been partially tested, and a significant percentage (about 10%) of the requirements has either failed or is blocked from testing. Browsing being a major feature for e-commerce sites, we can conclude that many problems remain with the quality of our site. We will need a much high level of confidence in this feature before we release, so more testing and defect repair is required

- Checkout, another major quality attribute and typically a more complex feature to implement, is almost completely tested, and all of the requirements tested so far work. We can be quite confident that this feature will be ready for release soon.

- Store Management, like Browsing and Checkout, is a major feature, because without it staff cannot put items in the store, offer discounts, manage inventory, and the like. For this feature, a limited amount of testing has been completed, and one-fifth of the requirements that have been tested do not work properly. This feature area needs a lot more attention, both from the testers and from developers.

- Performance is an important non-functional attribute for e-commerce sites, and here we can feel quite confident. Testing is complete, and there are no problems. We can feel confident of good performance in the field.

- Reliability is another important non-functional attribute for e-commerce sites, but unfortunately we have not completed any reliability testing. Most of the reliability requirements cannot be tested, due to blocking issues. These blocking issues probably need management attention, as well as individual contributor attention, to resolve.

- Security is likewise an important attribute for e-commerce sites, and again we have cause for concern here. No requirements have yet been tested. The test manager will need to have a good explanation for why this is the case.

- Usability and the User Interface, another important non-functional attribute for e-commerce sites, are likewise in a troubled state. We have tested every requirement, but over half of them don't work properly. Obviously, at this time we cannot have any confidence in that users can understand, operate, or enjoy our site.

Notice that, because this table is reporting on requirements status, we can translate these metrics directly into the level of confidence we can have in the system. When test counts are reported—e.g., how many tests cases have been run, passed, and failed—the relationship to the actual requirements is unclear, and therefore it's hard to know what level of confidence we should have.

Are these surrogate metrics or direct metrics? Well, both. From a verification perspective—i.e., does the system satisfy specified requirements?—they are direct metrics, and very good ones. From a validation perspective—i.e., will the system satisfy customers, users, and other stakeholders in actual field conditions when used to solve real-world problems?—they are indirect metrics. What does this mean?

On the one hand, if the requirements are good, they should reflect the needs and desires of the stakeholders. Complete fulfillment of these requirements says something important and positive

about the quality of the product. On the other hand, as discussed in the first article in this series, we are talking ultimately about measuring our level of confidence in the system. I wrote in that article that confidence is often measured through surrogate metrics of coverage, ideally multi-dimensional metrics that include code coverage, design coverage, configuration coverage, test design technique coverage, requirements coverage, and more.

The metrics in Table 1 are very useful, especially if they highlight problems. In other words, we can justly and confidently feel quite concerned about the prospects for our product and the quality of it if the requirements are known to be unfulfilled. However, the level of confidence we can have in positive measurements of requirements coverage alone is more limited. You cannot directly predict the satisfaction of customers, users, and other stakeholders based only on requirements coverage. In practice, it has proven difficult to develop direct, objective metrics that will predict satisfaction for software and systems. However, with a good set of multi-dimensional coverage metrics, we can have a higher level of confidence in eventual satisfaction.

Product Risk Metrics

We've considered product metrics that apply for requirements based testing. How about other testing strategies?

In the case of risk based testing, the objective is typically to reduce product quality risk to an acceptable level. Here are two questions we might consider:

- How effectively are we reducing quality risk overall?

- For each quality risk category, how effectively are we reducing quality risk?

The first question is the broader perspective, a question that senior managers and executives might ask to decide whether to classify the product as on track for a successful release or headed for trouble. The second question is the deeper perspective, a question that the project management team might ask to determine which risk categories are progressing properly and which need project control actions. Let's start with the first question.

When using a risk based test strategy, each quality risk item deemed sufficiently important for testing should have one or more tests created for it during test design and implementation. Bi-directional traceability is maintained between the tests and the risk items so that coverage can be assured and measured. Furthermore, the number of tests created for each risk item is determined by the level risk associated with that risk item.

During test execution, the tests are run and defects are reported. We need bi-directional traceability between not only the test results and the risk items, but also between the defects and the risk items. In this way, we can report which risks are fully mitigated (i.e., all tests are run and passed), which risks are partially mitigated (i.e., some or all tests are run, and some tests fail or some defects are known), and which risks are unmitigated (i.e., no failed tests or defects are known, but tests remain to be run).[3]

---

3   "   For more information on risk based testing, how to perform risk based testing, and how to report results for risk based testing, see my books *Managing the Testing Process, 3rd ed.* and *Advanced Software Testing: Volume 2*.
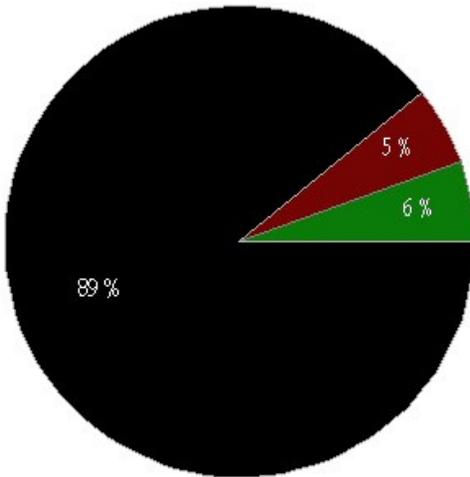
Figure : Quality Risk Status (Early Test Execution)

Figure shows a way of graphically displaying all this information, across all the risk items. The region in green represents risks for which all tests were run and passed and no must-fix bugs were found. The region in red represents risks for which at least one test has failed or at least one must-fix bug is known. The region in black represents other risks, which have no known must-fix bugs but still have tests pending to run.

It's important to mention that the area associated with a given risk is determined by the level of risk; i.e., the risks are weighted by level of risk. In other words, a risk item with a higher level of risk will show more pixels in Figure (in whichever region it is classified), while a risk item with a lower level of risk will show fewer pixels. This property of risk weighting is critical, because it means that the pie chart gives a true representation of the amount of residual risk, across all the risk items. Without such risk weighting, the low-level risks would be exaggerated relative to the high-level risks.

As noted in the caption for Figure , this figure shows the residual level of risk early in the test execution. What happens when we monitor the progress of quality risk mitigation throughout test execution?

Figure shows what we would expect during a successful project's test execution, right around the late-beginning or middle of that period. The red region—i.e., the risks with known problems —is growing disproportionately fast as we discover the problem areas of the product. (An objective of risk based testing is to discover these problem areas as early as possible during test execution.) The green region is also growing, though, and it will soon start to out-pace the growth of the red region. Both regions will grow rapidly in the middle of the test execution period, pushing out the black region which represents unmitigated risks.
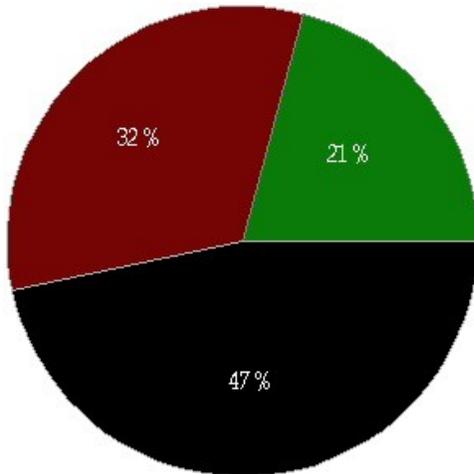
Figure : Quality Risk Status (Middle of Test Execution)

Towards the end of a successful project, we see a picture like that shown in Figure . During the second half of the test execution period, the green region starts to grow very quickly. We have run all the tests which are likely to find important bugs early in the test execution period (this is a property of risk based testing), so that few bugs are found in the last half of test execution, and those bugs which are found are not as important. Testers focus on running confidence-building tests that are lower risk (turning black pixels into green ones), while developers fix the bugs that were found (turning red pixels into green ones).
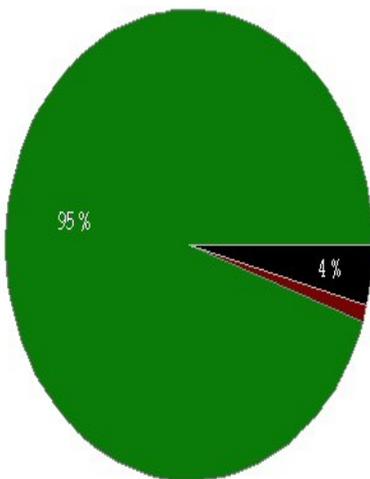

Figure : Quality Risk Status (Late Test Execution)

So, a natural question to ask, looking at Figure , is whether we are done with testing? Must the pie chart be entirely green? What are realistic goals for risk testing that will ensure adequate quality and test coverage?

When following a risk based test strategy, the question of adequate test coverage is decided by the project management team. If the project management team believes that the quality risks posed by known defects, known test failures, and yet-unrun tests are acceptable, at least as compared to the schedule and budget risks associated with continuing to test, then we can say that the risks are in balance and quality risk mitigation has been optimized.

Does this mean product quality is adequate? Maybe. If we have done a good job of identifying quality risks and assessing their levels, then that reflects the impact (on the customer, user, and other stakeholders) associated with failures related to each risk item. But also maybe not. Even if our quality risk analysis is correct, what constitutes an acceptable level of quality risk is subjective. There's no guarantee that the project management team's opinion matches the opinions of the customers, users, and other stakeholders. However, it's better to have an informed opinion on, and to make an educated decision about, product quality before releasing software. Product metrics allow us to do so.

Now let's move into the detailed question of quality risk, category by category. This more detailed view is what project participants typically need. If the higher-level information shown in Figure shows that project adjustments are needed, this detailed information will allow the project management team to make decisions about test and project control activities needed to optimize the level of quality and minimize the residual level of quality risk.

Table shows a tabular presentation of the defects and tests, broken down by risk category. Let's assume again that this table reports status for a large and complex e-commerce application (though not the same project as Table). Notice how Table ties together important project metrics for tests and defects with the quality risks. This bridge between the product metric of quality risk and these two project metrics is what makes this table an effective tool for making test and project control decisions. Note also that the quality risk categories are sorted by the number of defects found.

Quality Risk Category	Defects		Tests		
	#	%	Planned	Executed	%
Performance	304	27	3,843	1,512	39
Security	234	21	1,032	432	42
Functionality	224	20	4,744	2,043	43
Usability	160	14	498	318	64
Interfaces	93	8	193	153	79
Compatibility	71	6	1,787	939	53
Other	21	2	0	0	0
Total	**1,107**	**100**	**12,857**	**5,703**	**44**

Table : Risk Coverage Table

Let's analyze what Table is telling us:

- Performance accounts for the largest percentage of defects reported. While we have run a large number of tests for performance, over 60 percent of the tests remain to be executed. Therefore, we can expect to discover more defects in this risk category. We need to make sure that these defects are being fixed promptly, and that any obstacles to running the remaining tests are removed quickly.

- Security is the second-leading risk category in terms of defects and also has a significant number of tests remaining to run. As with performance, we need fast defect resolution and a speedy conclusion to this set of tests.
- The status of Functionality is similar to that of performance and security, but the number of tests remaining to run is much larger than for those two categories. Depending on where we are in terms of test progress (i.e., on track or behind schedule), we might need to consider ways to accelerate the execution of the remaining tests.
- Usability is in better shape than these first three categories. While a significant number of defects have been found, the testing is mostly complete. Since we are following a risk based testing strategy, the most important usability tests have already been run. This risk category is probably proceeding acceptably, with no adjustments required.
- For the Interfaces category, relatively few defects have been found, and most of our testing is complete. Assuming timely defect repair and no blocked tests for this category, we are almost done with interface testing.
- For the Compatibility category, the defect count is low, which is reassuring. However, a significant number of tests remain to be executed. Some investigation of what is needed to get these tests completed soon is in order.
- Finally, the Other category is for those defects reported during testing (including especially reactive forms of testing such as exploratory testing) that do not relate to any specific quality risk item identified during the quality risk analysis. (Because there were not specific risk items, there are no planned tests for this category.) If the risk analysis is accurate, then the number of defects in the Other category will be relatively low, as it is here. If more than 5% of defects are found by such tests, then the test manager should determine which risk items may have been missed during the quality risk analysis, and adjust testing accordingly.

I've talked about using this table for monitoring and control. What are some realistic goals?

- The distribution of defects should approximately match the expected distribution, as predicted during the quality risk analysis. If a particular quality risk category contained few high-likelihood risk items, then it should be at the bottom of the table, and vice versa. Unexpectedly high or low numbers of defects indicate a problem with the quality risk analysis.

- As mentioned above, the number of defects found which do not relate to an identified quality risk item should be low.

- Most if not all of the planned tests should be executed by the end of the project. I say "most if not all" rather than "all" because some adjustments to the quality risk analysis might result in the deliberate skipping of some tests in order to add tests in other areas.

If problems are found with the quality risk analysis, then, as part of a test project retrospective, an investigation of why the quality risk analysis was inaccurate is in order.

You probably noticed that this table looks very much like Table . However, the difference is visible if you drill down into the detailed data underneath each row. In Table , the details are tests and their results, and the relationship between these items and the individual requirements elements. In Table , the details are tests, their results, and defect reports, and the relationship between these items and the individual quality risk items identified and assessed during the risk analysis.

Conclusions

Let me close this series of articles by summarizing the key ideas presented in the series.

We have looked at three types of testing metrics: process metrics, project metrics, and product metrics. Some of testing metrics fit cleanly into one type, while others can span types. The type of metric you're looking at can depend on your intended purpose for the metric. You can use metrics to measure, understand, and manage process, project, and product attributes.

In each of the four articles, I outlined a process for metrics, and illustrated that process with examples. The process, a variant on Victor Basili's Goal-Question-Metric process, consists of: defining the process, product, or project objectives; developing questions about how effectively, efficiently, or elegantly we achieve those objectives; creating metrics that allow us to answer those questions; and, setting goals (explicit target measures) for those metrics that indicate success. When these goals do not indicate success, management awareness of the shortcomings should trigger adjustments and improvements in the testing, the broader software process, the project, or the product.

In some cases, the objectives you need to achieve are hard to measure directly. When you find yourself confronted with such a situation, you can use surrogate metrics to provide indirect measures, such as using coverage metrics like the ones shown in this article to measure product quality.

A main theme in this series of articles is that these testing metrics are not only nice to have, they are truly essential for a complete and accurate understanding of the facts and realities with which you are dealing. Managing without metrics means managing without facts, managing only with opinion. While certainly expert opinion is essential for management, opinion is not enough by itself. Metrics help us recognize which of our opinions are accurate and which are misguided.

While metrics are essential, you should remember that the point is quality, not quantity. You should use a small number of metrics to accomplish what you need. Too many test managers make the mistake of subjecting their colleagues to a fire-hose of data, which tends to hide the information within it. Just because something is easy to measure doesn't mean it matters, and just because something is hard to measure doesn't mean it doesn't matter.

Another essential element of successful metrics is consistent stakeholder understanding and support of the objectives, the metrics, and the goals for testing. A successful metrics program is built in collaboration with testing stakeholders, not as a defensive reaction against them. In addition, all stakeholders, including the test manager, must commit to the proper use of metrics information, avoiding misuse such as using process or project metrics to reward or punish individual contributors who are generally not in control of the primary factors that influence the results of those metrics.

Can you put a successful test metrics program in place? Yes, you can. As you have seen in this series of articles, doing so is challenging, but also achievable. Manage with data, and you will be a better manager. I wish you the best of success in instituting your own testing metrics program.

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, 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@rbcs-us.com.