Defining Test Mission, Policy, and Metrics of Success

[This article is an excerpt from an upcoming book, *The Expert Test Manager*, to be published by Rocky Nook this fall and written by Rex Black and Debbie Friedenberg.]

The International Software Testing Qualifications Board (ISTQB) program defines testing expansively. Static testing (via reviews and static analysis) is included as well as all levels of dynamic testing, from unit testing through to the various forms of acceptance testing. The test process is defined to include all necessary activities for these types of testing; planning; analysis, design, and implementation; execution, monitoring, and results reporting; and closure.

What is the point of all this work? We could do all of these activities in a fashion that is technically correct but still fail to deliver any value to the business. That might seem surprising, but we have performed a number of assessments for clients where their internal testing processes functioned reasonably well but their work and deliverables were not valued by any of the key stakeholders. The managers of these test groups typically had failed to identify the stakeholders or, worse yet, in spite of being aware of the stakeholders, failed to work collaboratively with them to determine how to connect the testing process to something valued by each of them.

Identifying Stakeholders

Who are the people who have a stake in the testing and quality of a system? We can broadly classify them as being either technical stakeholders or business stakeholders. A technical stakeholder is someone who understands and participates in the design and implementation of the system. The following people are considered technical stakeholders:

- Programmers, lead programmers, and development managers. They are directly involved in implementing the test items, and so testers and programmers must have a shared understanding of what will be built and delivered. These stakeholders receive defect reports and other test results. In many cases, they need to act on these results (for example, when fixing a defect reported by testers).
- Database administrators and architects, network architects and administrators, system architects and administrators, and software designers. These stakeholders are involved in designing the software and the hardware, the database, and network environments in which the software will operate. As with

programmers, they receive defect reports and other test results, sometimes based on early static tests such as design reviews. These stakeholders also may need to act on these results.

- Technical support, customer support, system operators, and help desk staff. These stakeholders support the software after it goes into production, including working with users and customers to help them realize the benefits of the software. Those benefits come from the features and quality of the software but can be compromised by defects in the software that remained when it was released.

A business stakeholder is someone who is not so concerned with the design and implementation details but does have specific needs that the system must fulfill. The following people are considered business stakeholders:

- Marketing and sales personnel, business analysts, user experience professionals, and requirements engineers. These stakeholders work with actual or potential users or customers to understand their needs. They then translate that understanding into specifications of the features and quality attributes required in the software.
- Project managers, product managers, and program managers. These stakeholders have responsibilities to the organization for the overall success of their projects, products, or programs, respectively. Success involves achieving the right balance of quality, schedule, feature, and budget priorities. They often have some level of budget authority or approval responsibility, which means that their support is essential to obtaining the resources required by the test team. They will also often work directly with the test manager throughout the testing activities, reviewing and approving test plans and test control actions.
- Users (direct and indirect) and customers. Direct users are those who employ the software for work, play, convenience, transportation, entertainment, or some other immediate need, while indirect users are those who enjoy some benefit produced or supported by the software without actually employing it themselves. Customers, who might also be direct or indirect users, are those who pay for the software. They are the ones whose needs, ultimately, are either satisfied or not by the software (though satisfaction is typically more of a spectrum than a binary condition).

These lists are not intended to be exhaustive but to help you start thinking about who your stakeholders are. In addition, in some cases a stakeholder may be simultaneously a business stakeholder and a technical stakeholder. So, don't worry too much about trying to place a particular stakeholder clearly into one category or another.[1]

---

[1] For a more thorough discussion of testing and quality stakeholders, you can refer to Chapter 2 of *Beautiful Testing*.

Understanding the Mission and Objectives

Having identified the stakeholders, the next step is to work directly with each stakeholder, or a qualified representative of that stakeholder group, to understand their testing expectations and objectives. That might sound easy enough, but when you start that process, you'll find that most stakeholders cannot answer a straightforward question such as, "What are your testing expectations?" They simply will not have thought about it before, at least not with any real clarity, so the initial answers will often be something like, "Just make sure the system works properly before we ship it." This, of course, is an unachievable objective, due to two fundamental principles of testing: (1) the impossibility of exhaustive testing and (2) testing's ability to find defects but not to demonstrate their absence. You'll need to discuss the individual stakeholder's needs and expectations in terms of quality and then the extent to which testing can realistically help the organization achieve their objectives.

These discussions may initially be hazy and often unrealistic, but ultimately they should coalesce into a crisp, achievable mission and set of objectives. The specific mission and objectives that you and your stakeholders arrive at will vary, so we can't provide a one-size-fits-all list here. However, a typical mission statement might be something along the following lines:

> To help the software development and maintenance teams deliver software that meets or exceeds our customers' and users' expectations of quality.

Notice the word *help* here, which in this context means to assist (i.e., cooperate effectively with) others involved in the software process to achieve some goal. In this case, the goal is the delivery of quality software. You want to avoid mission statements like this one:

> To ensure that our customers and users receive software that meets or exceeds their expectations of quality.

There are two problems with this mission statement. First, it leaves out the software development and maintenance team, making software quality a testing problem entirely. Second, it uses the word *ensure*, which in this context means to guarantee that customers and users are satisfied or even delighted with the quality of the software. Due to the two principles mentioned earlier, both of these flaws are fatal. Testing is an essential part of a broader strategy for quality, but it is only a part of that strategy and can never guarantee quality by itself.

Given a realistic and achievable mission for testing, what objectives might you establish that would support achieving that mission? While your objectives will vary, typically we see one or more of the following:

- Find critical defects. Critical defects are those that would significantly compromise the customers' or users' perception that their expectations of quality had been met; that is, defects that could affect customer or user satisfaction. Finding defects also implies gathering the information needed by the people who

will fix those defects prior to release. The person assigned to fix a defect can be a programmer (when the defect is in software), a business analyst (when the defect is in a requirements specification), a system architect (when the defect is in design), or some other technical or business stakeholder.

- Find noncritical defects. While the main focus of testing should be on critical defects, there is also value in finding defects that are merely annoyances, inconveniences, or efficiency sappers. If not all noncritical defects are fixed, which is often the case, at least many of them can be known. With such defects known—especially if the test team can discover and document workarounds—technical support, customer support, system operators, or help desk staff can more effectively resolve production failures and reduce user frustrations.
- Manage risks to the quality of the system. While testing cannot reduce the risk of quality problems to zero, anytime a test is run, the risk of remaining, undiscovered defects is reduced. If we select the tests that relate to the most critical quality risks, we can ensure that the maximum degree of quality risk management is achieved. If those tests are run in risk order, the greatest degree of risk reduction is achieved early in the test execution period. Such a risk-based testing strategy allows the project team to achieve a known and acceptable level of quality risk before the software is put into production.
- Build confidence. Organizations, especially senior managers in organizations, don't like surprises. At the conclusion of a software development or maintenance project, they want to be confident that the testing has been sufficient, the quality will satisfy customers and users, and the software is ready to release. While it cannot provide 100 percent surety, testing can produce, and test results reporting can deliver, important information that allows the team to have an accurate sense of how confident they should be.
- Ensure alignment with other project stakeholders. Testing operates in the context of a larger development and maintenance project and must serve the needs of the project stakeholders. This means working collaboratively with those stakeholders to establish effective and efficient procedures and service-level agreements, especially for processes that involve handoffs between two or more stakeholder teams. When work products are to be delivered to or from testing, then test managers must work with the relevant stakeholders to establish reasonable entry criteria and quality gates for those work products.

At this point in the process, you will have established a clearly defined mission and objectives for testing. What should be done with this important information? The best practice is to document them in a test policy. The test policy can be a standalone document, or it can be a page on a company intranet or wiki.

In addition to the testing mission and objectives, the test policy should talk about how testing fits into the organization's quality policy, if such a policy exists. If no such

quality policy exists, it's important to avoid the trap of calling this document or page the "quality assurance policy" because testing by itself cannot assure quality. If your group has the phrase *quality assurance* in its title, and therefore you must call this page the "quality assurance policy," be careful to explain in the policy that testing is not a complete solution to the challenge of delivering high-quality software.

Measuring Achievement

The test policy should not only list the objectives, it should also establish ways to measure the achievement of the objectives. Without some way of measuring achievement, the success or failure of the test group becomes a matter of stakeholder opinion, a kind of dangerous popularity contest. Given the test team's role in pointing out defects in other people's work, the wise test manager does not bet on winning such contests. Instead, establish, in the test policy, clear metrics for each objective.

These metrics should be used to measure effectiveness, efficiency, and stakeholder satisfaction. Effectiveness refers to the extent to which an objective has been achieved. An effectiveness metric will often be expressed as a percentage, with the upper end of the scale (100 percent) indicating complete achievement of the objective. Efficiency refers to the economy with which an objective has been achieved. An efficiency metric can likewise be expressed as a percentage. In some cases an efficiency metric can exceed 100 percent, such as measuring the efficiency of testing using cost of quality analysis (as will be discussed in Chapter 4). In some cases an efficiency metric can indicate success through a lower percentage, such as when the metric is used to measure the degree of waste in a process.

Satisfaction refers to the stakeholders' subjective evaluation of the achievement of an objective. Satisfaction is often measured via surveys, asking stakeholders to react to statements in terms of their agreement or disagreement. A typical scale for measuring this reaction is called a Likert scale.[2] In this scale, which is often a five-point scale, a reaction of 1 indicates strong disagreement with the statement, while a reaction of 5 indicates strong agreement. The other points on the scale—2, 3, and 4—indicate disagreement, neutrality, and agreement, respectively. While we might seem to be venturing back into the realm of popularity contests by including subjective satisfaction metrics, note that perceptions *are* realities. If objective effectiveness and efficiency metrics show success but at the same time many stakeholders are dissatisfied with your group's work, something is clearly wrong, and your job as a manager is to fix it.

Because this topic of establishing metrics is new to many test managers, even experienced ones, let's look at some examples of effectiveness, efficiency, and stakeholder satisfaction metrics for testing. We'll start with a couple of effectiveness metrics. For the objective of finding defects, we can measure the effectiveness of the overall test process using the defect detection effectiveness (DDE) metric:

---

[2] For a more detailed explanation of Likert scales, see http://en.wikipedia.org/wiki/Likert_scale.

$$DDE = \frac{defects\ found\ in\ test}{defects\ found\ in\ test + production\ defects}$$

For the objective of managing risk, we can measure the effectiveness of the test design process by looking at the risk coverage (RC) metric:

$$RC = \frac{quality\ risks\ for\ which\ tests\ were\ designed}{total\ quality\ risks\ identified\ during\ quality\ risk\ analysis}$$

Of course, in the RC metric, we should exclude from the denominator any quality risks that were identified but for which the stakeholders explicitly decided not to design or implement any tests. If you do not exclude those risks, you might end up designing more tests than intended by trying to get to 100 percent risk coverage.

Here are a couple of efficiency metrics. For the objective of finding defects, we can measure the efficiency of the overall test process by using cost of quality analysis to calculate the testing return on investment (Test ROI):

$$Test\ ROI = \frac{(average\ cost\ of\ a\ production\ defect\ \times test\ defects) - cost\ of\ internal\ failure}{cost\ of\ detection}$$

You might notice that you will need to be a bit careful about this metric because it could lead to dysfunctional behavior in the organization. For example, finding large numbers of defects is easy with very buggy software, but you don't want to incentivize people to find more defects in testing than in earlier parts of the software life cycle. It's also possible to increase Test ROI by reducing the investment made in test planning, analysis, design, and implementation, but that will tend to drive down the team's overall effectiveness at finding defects (as measured by DDE). If we use both DDE and Test ROI as key metrics, with an emphasis on DDE, then we have a balanced pair of metrics.[3]

A major element of confidence for existing systems is having confidence that the existing features will continue to work. This can be done through regression testing, but manual regression testing is expensive and inefficient in many cases. So for the objective of building confidence, we can measure the efficiency of the regression test implementation process by looking at the automated regression tests (ART) metric:

$$ART = \frac{number\ of\ automated\ regression\ test\ cases}{total\ number\ of\ regression\ test\ cases}$$

This is another metric that must be used with care (as indeed must all metrics be). For one thing, while it addresses efficiency, it does not address effectiveness. To provide this balance, we would also need a metric to measure the percentage of existing features

[3] Capers Jones makes a similar point about the weakness of cost-per-defect metrics in his book *Economics of Software Quality*, but he acknowledges the usefulness of cost of quality analysis.

covered by regression tests (whether manual or automated). For another thing, we have to make sure the manual regression tests and the automated regression tests are similar in size and scope. The size of the test case can be measured by the number of screens, data records, or test conditions covered by the test. If the manual regression tests are considerably larger than the automated regression tests (e.g., if the manual tests cover an entire use case while the automated tests each cover only a single screen), then adding the number of manual tests and automated tests to calculate the total number of regression tests is meaningless. It's like adding two temperature measurements, one in centigrade and one in Fahrenheit.

Finally, here are a couple of satisfaction metrics. For the objective of ensuring alignment with other stakeholders, we can measure the satisfaction of project stakeholders who have work product handoffs to or from the test group at any point during the test process via a survey. We can ask them to react to the following statement:

> The handoff of work products between the test team and our team goes smoothly.

For the objective of building confidence, we can measure the satisfaction of all project team stakeholders with the test results reporting process via a survey. We can ask them to react to the following statement:

> The test results, as reported to you, help you guide the project to success.

For both surveys, we can use a Likert scale, as described earlier. We can calculate an average score using that scale as well as flag any responses at 3 (neutral) or below for further investigation.

Note that we have defined metrics not only for test execution but throughout the test process. During test execution, we need project and product metrics that allow us to measure the completeness of testing and the quality of the software being tested. These metrics allow us to report test results and fulfill the information-delivery objective of testing (see Chapter 6). However, the metrics we need in a test policy are process metrics, which measure the degree to which our team and processes are achieving established objectives.

It's important to remember that the objectives of testing, and the metrics used to measure them, are affected by the level of testing in question. For example, while finding critical defects is a typical objective of testing, it takes different forms in different levels of testing. For unit testing, we want to find defects that would prevent the function or class being written from providing services or capabilities to other functions or classes such that the system, when integrated, would not satisfy customers or users; that is, the objective is narrowly focused on the individual unit of code being tested. In comparison, for system testing we want to find defects that would impact the customers' or users' ability to enjoy the capabilities and functions provided by the entire system; that is, the objective is broadly focused on the entire system in the context of actual usage. As another point of comparison, note that finding defects is typically not an objective for user acceptance testing and that finding critical and/or numerous

defects in this test level is often seen by stakeholders as a sign of failure of early quality assurance and testing activities.

As another example, the objective of confidence building applies to unit testing and is often measured using code coverage metrics.[4] The same objective applies to system testing, but here we should not rely on achieving thorough code coverage because that should have happened in unit testing. We focus instead on coverage of quality risks, requirements, design specification elements, supported environments, data and metadata, and similar test basis items. Note that again the focus of the objective and the metrics for that objective is narrow in the case of unit testing and broad in the case of system testing. For user acceptance testing, confidence building is typically the primary objective and is measured through coverage of requirements, use cases, user stories, supported environments, data, and metadata.

Setting and Meeting Goals

You should set initial goals for the metrics once they're established. By goals, we mean target measurements that you want to achieve. You should also establish some rules for when you will measure each metric. For example, some metrics are most appropriately measured on a project-by-project basis, while others can be measured weekly, quarterly, semiannually, or annually across all the project teams.[5]

If you have an existing test process established and operating, you should measure your current capabilities as part of the goal-setting process (i.e., baseline). You should also consult industry averages and experts (e.g., outside consultants) to understand typical capabilities of other test organizations (i.e., benchmark). If your current capabilities fall short of typical capabilities, then you should introduce test process improvement plans to correct the underlying problems.[6]

With the baselined and benchmarked measurements in hand, you should work with the key stakeholders to set current goals and, if appropriate, longer-term goals that can be achieved once process improvements are made. Key stakeholders must be included in the goal-setting process because only with their concurrence can you achieve a situation in which the test team can, measurably and with data, demonstrate that it is succeeding in its mission and achieving its objectives. The test manager or test director must ensure that key stakeholders reach consensus on actually achievable goals. Where goals are not measurable, or are measurable but not achievable (e.g., a goal of 100 percent DDE), success is ephemeral and transient at best and stakeholder expectations are unlikely to be satisfied.

---

4 You can find a detailed discussion of code coverage metrics in the book by Jamie Mitchell and Rex Black, *Advanced Software Testing: Volume 3*, as well as Rex's book *Pragmatic Software Testing*.

5 You can read more about this topic in Rex Black's e-book *Testing Metrics*.

6 This topic is discussed in detail in the ISTQB's Improving the Testing Process syllabus and Rex Black's e-book *Improving the Testing Process*.

With these examples of a typical mission and objectives, goals, and metrics given, keep in mind that the wider context will affect what your objectives are and the goals you set for them. If you are in a highly competitive and fast-moving market or writing software subject to regulatory deadlines for delivery, you might have to compress the test execution period, which will reduce the defect detection percentage you can achieve. If the organization selects a software development life cycle that reduces the scope of the test effort prior to release, that will result in a higher level of residual quality risk at the end of testing. Basically, to the extent that constraints or priorities limit testing, whether those constraints or priorities come from the business, customers, users, or other realities, the software development lifecycle will reduce what is achievable in terms of the testing goals.

Reducing the achievement of goals does not mean utter failure. Constraints in terms of budget, schedule, resources, and personnel are ubiquitous; no test manager gets everything they want. Priorities in terms of schedule, budget, or feature set always compete with quality; few test managers operate in an environment where quality is the number one priority, and those who do find that such a situation brings with it additional challenges. So, like all managers, the test manager operates within parameters. Those parameters affect the degree to which goals are achieved. That effect shows itself in specific measures for our metrics of effectiveness, efficiency, and satisfaction, which reflect not only the test processes' and test group's capabilities but also the influence of those parameters.

The previous paragraph is not intended to provide an excuse for suboptimal performance and lackadaisical effort. As a manager, you should focus on maximizing the degree to which you satisfy the established test objectives. When establishing the test policy, test strategy, and test plans, test managers must carefully evaluate how those parameters will affect the selected objectives. Goals can be adjusted based on these parameters. During test control, emergent constraints or priorities must also be evaluated and further adjustments potentially made. Executives and senior managers will not be pleased with a test manager or test director who excuses underachievement of goals with a collection of excuses about constraints and priorities when that manager cannot provide evidence that they took steps to achieve the best possible outcome within those parameters.

Another mistake to avoid is writing the test policy in an aspirational fashion or setting so-called "stretch goals" in it. Understand the parameters and how they will affect your processes' and team's capabilities. The test policy, especially the goals, should reflect what can actually be done. You should plan to measure the effectiveness, efficiency, and satisfaction delivery of your team. When goals are not achieved, you should understand why and take steps to avoid such shortfalls in the future. If the team is continuously underachieving, then individual contributors and line managers will perceive the test policy as, at best, irrelevant or, at worst, a cudgel with which they can be beaten regardless of their competency. Either situation is a classic sign of poor leadership.

That said, there is nothing wrong with using the test policy as part of a continuous test process improvement initiative. In fact, you should do so. If during a test retrospective, people note things that went really well, try to determine why and whether that can be instituted as a best practice across subsequent projects. Ask your teams, during test retrospectives and as part of continuous test process improvement projects, to look at the objectives and ask themselves whether new goals can be set or existing goals improved. Should new objectives be added? Are there additional stakeholders who could receive useful services from the test team? A test director or test manager should regularly look for ways to improve the effectiveness and efficiency of their teams and to increase stakeholder satisfaction with their work.

This whole process of identifying stakeholders and working with them to define the test group's mission, objectives, goals, and metrics may sound like a prolonged, political, and painful exercise. Since technical people are often introverts who prefer to focus on the work at hand, and their technical abilities to perform that work, rather than the people doing it, we have noticed reticence to engage with the stakeholders in this way. However, make no mistake about it. If you, as a test manager or test director, do not make the effort to align your testing services with the needs of your stakeholders—your internal customers, really—you have chosen the most risky of omissions that we have encountered in our decades of practicing, training, and consulting. Excellent technical capability will not make the test team successful if those capabilities are being deployed in the wrong directions.

