Metrics Collection and Analysis for Web Sites

STAR East 2001, Track W8

Joe Polvino, CSQE

Joe_Polvino@elementk.com

To many organizations, the concept of metrics is foreign. Even after taking training on metrics usage, few organizations take advantage of the value metrics can bring. This paper presents the special challenges online companies face, describes a practical plan for rolling out test metrics, and shows how test metrics collection and analysis can reduce cycle time and provide meaningful information to the development team.

Introduction

The challenges faced by software testing today are great, regardless of whether the product is traditional or web-based. The complexity of software continues to increase, and the demand for quicker defect turnaround is great, making the pressure on the whole development team even greater. For software companies involved in web-based software development, this is compounded by the need for continuous enhancements and demands to be first-to-market.

Traditional standalone software differs from web-based software on many counts, and the need for test metrics becomes more apparent in the latter. Standalone software is afforded a predictable operating environment, common code base, predictable interfaces, and the luxury of releases measured in months or years. Internal spins are common, and the test group is able to test a product that is becoming more functional with each internal release. In this case, test metrics can give a good representation of test progress and defect status, provided the test group clarifies the reference points for measure. (For instance, does 80% tested mean that 80% of the tests run are linked to functioning requirements or all requirements?)

The most obvious difference between standalone and web-based software is the time factor. For web-based software, being first-to-market with a functional product that uses the latest technology is the overriding goal for many companies. Coupled with the fact that many of these companies are young and eager to please investors, taking the time to learn and invest in test metrics is often seen as a hindrance as opposed to a benefit. Regardless of the situations, understanding the value test metrics can provide will enable software groups to decide what they want to measure and how they will make decisions based upon the output.

For many software groups, the perception of metrics is a mixed bag. Perhaps they cannot bear the word “metrics” because someone tried force-feeding the concepts to the group and met a great deal of resistance. Maybe everyone was forced to take a metrics course and rebelled subconsciously by reverting to business as usual when the class was finished. Whatever the

reasons, it is important to overcome these barriers and try the soft-sell approach; let the group come to the conclusion that collecting test metrics can improve their products.

Experience has taught me that metrics collection strategies fall into four categories: non-existent, post-release, packrat, and practical.

Non-existent metrics collection is simply that: there is none. For some groups, simple ignorance may be the underlying reason. This is typical of groups that find themselves re-inventing the wheel each time and not learning from their mistakes. For others, it may go back to a bad experience with metrics, and people loathe the concept all together.

Groups that realize the value of metrics only after launch fall into the post-release category. Many times, the group tries to resurrect stale or non-standardized data after launch to see if anything can be learned for the next release. This usually results in skewed results that provide misleading information; this has the effect of propagating misinformation into the next release.

Packrats are groups that lack a clear strategy of what to collect and why. Rather than determining the necessary data points, personnel are required to collect a large number of metrics without clear understanding of how it will be used. If data is not used immediately, it is archived with the hope that it can be used after the dust settles. Unfortunately, the dust never settles and the data is rarely, if ever, used. This turns people off because of the large time investment with little or no perceived benefit.

When a group understands exactly what they need to report and collects just the data necessary to support that, they fit into the practical category. By working backward from the goal to the raw data, it is possible to reduce the amount of waste and potentially leverage common data points. For example, if the average number of test cases run per tester is an important metric, then the number of testers per day and the number of test cases per day need to be collected. Additionally, if the average defects found per tester is needed, then the number of testers is already known, and the number of defects discovered per day is needed. This approach is the least intrusive way to collect and report metrics, and allows everyone to see how the data is being used. [1]

Rolling out a Solution

Assuming a group wants to follow the “practical” approach described above, the next step is to determine what test metrics are to be reported, and how. This can be done several ways, but the most time-effective way (particularly important to web-based software) is to interview those who will consume the metrics. You may get requests for percent tested of total, percent passed of total, remaining open defects, and test activity, for example. Whatever the results of this exercise, you will need to then look at a way to collect data to support these views. We have found that a spreadsheet with only a few daily input fields that feeds charts and summary sheets provides an unobtrusive method for collecting data and reporting metrics.

Once you have a prototype, the testers and consumers need to be educated on its use. It is important to stress that people will not be measured by metrics, otherwise data may get skewed. For the tester class, having clear policies and a simple data entry form will help avoid

misunderstandings. It also helps spread awareness if they understand how their data is being used, so they can explain to managers changes in trends. Likewise, managers will be able to make fact-based decisions once they understand what the reports tell them. After everyone is educated on understanding test metrics, they will be able to understand what the trends say as opposed to just knowing their definitions.

Test Metrics Usage

As mentioned earlier, time is a critical factor for web-based solutions. Because of the risk associated with launching a feature directly onto the production servers, many companies implement a staged approach to launching and testing features.

The staged approach defines at least one pre-production environment, usually behind a firewall, in which testers can verify the feature, and a production environment visible to the world. Prior to transitioning from pre-production to production, a department policy needs to be met, such as a low number of remaining defects. Published and enforced policies prevent premature launch and test metrics allow the team to see how close they are to these goals.

For the remainder of this paper, 3 stages will be referenced: QA, Beta and Live. The QA stage is behind a firewall, Beta is outside the firewall but accessible to only certain users, and Live is accessible to the world.

Usually, the project manager will have set the dates for the transitions. Representing these dates on a chart gives the impression that there are “finish lines” and can serve to motivate people one way or another. See Figure 1 for an example.

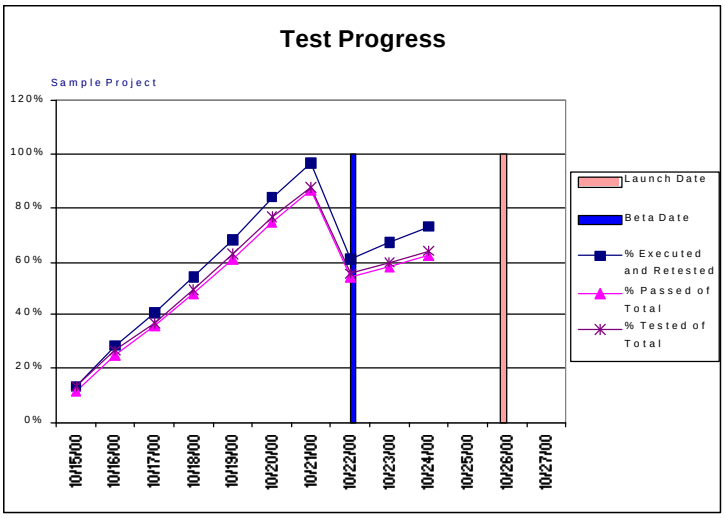

Figure 1: Transition dates define stage boundaries

Daily data entry is essential for accurate test metrics reporting. Without timely reporting, data can become stale and the charts will not represent reality. Metrics are a dish best served warm; it benefits everyone if regular daily reviews occur to discuss the trends.

Test Metrics Categories

The test metrics categories differ according to the needs and constraints of the organization. The final set should be arrived at while considering how useful they will be while the project is in progress, and how they can feed improvement opportunities to future projects via post-mortem reviews. The group needs to balance the effort of test metrics collection with the payback of collection and analysis. Regardless of the test metrics categories, having broad representation during soul-searching sessions will give people the sense that their input matters, and will provide a form of education at the same time.

At a minimum, test metrics categories should include test progress, defects remaining, daily test activity, and test efficiency. Not one of these alone should be used to gauge project health. Rather, all should be considered. For this reason, it is helpful to arrange your charts on a single page so they can be seen simultaneously.

Test progress is a measure of test status over time. It may be comprised of percent tested of total, percent passed of total, and percent passed of tested. Because of the phased approach, there may be test cases that cannot be executed until the project transitions to a specific phase. If this is the case, then the denominator in the percentage calculations increases, resulting in a decrease in these test metrics. This is apparent in Figure 1, above. Part of the education portion is to explain why this happens so people do not panic.

The chart showing defects remaining gives the team an up-to-date representation of the total number of defects that remain open. One practical approach is to log the number of defects discovered each day, and the number of defects closed each day. Collecting the severity of those defects gives an added dimension because people can see what comprises the remaining defects. Your organization may have among its policies a rule that the project will not launch with any open high or medium defects, for example, and this chart could show this at a glance. See Figure 2.

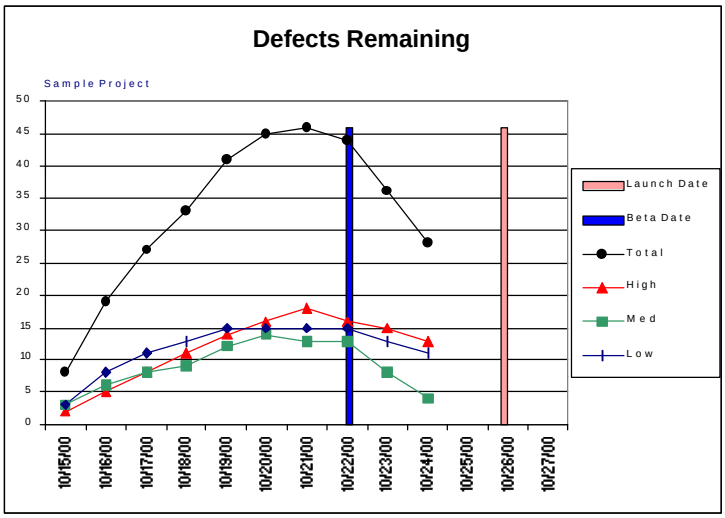

Figure 2: Defects remaining

Daily test activity represents the number of test cases executed per day. If, for example, your project has X number of cases that need to be run in 10 days, then you can overlay your goal rate against the actual rate to determine progress. See Figure 3.

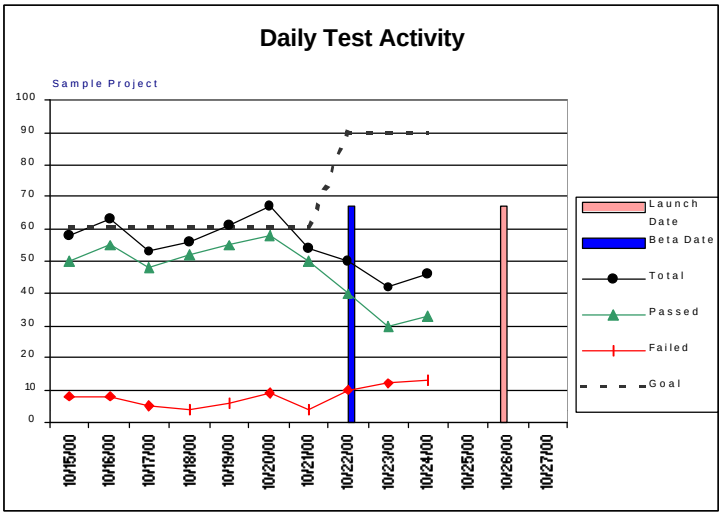

Figure 3: Daily test activity

Finally, test efficiency can be used to measure how well your test team performs, based on rates per tester. Figure 4 shows an example. This may be useful to measure how the execution rate and defect discovery rate relate to the number of testers. Beware that for each tester, there is additional burden associated with complex communication paths.[2]

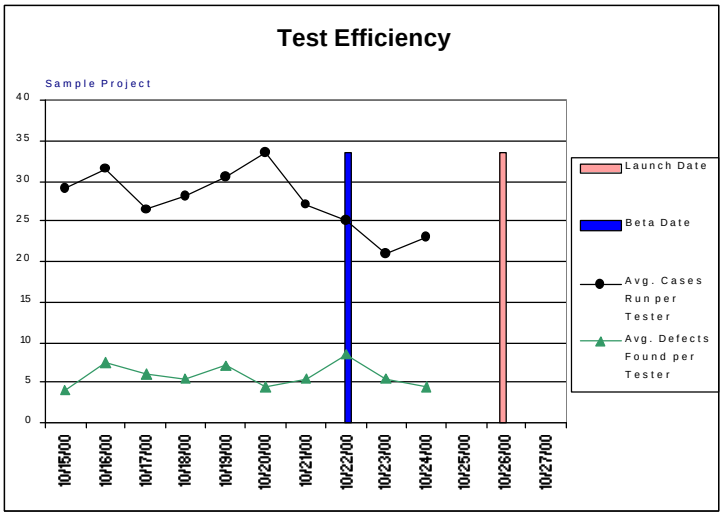

Figure 4: Test efficiency

Interpreting Results

Collecting test metrics and displaying them on charts is only part of the job. Interpreting the results and determining what to do with those interpretations is where groups begin to see the value. Especially in the web-based software development world, time is an important factor. By using line charts with transition dates as indicated in the figures above, the team can see the trends versus the target dates.

In Figure 1, it is clear that if test progress remains steady, there will be unexecuted cases at launch. This constitutes risk because if the test cases are mapped to requirements, there is the potential for untested features to make it to users.

Observing Figure 2, one can see that if the defect discovery/closure rate remains constant, then there will probably be defects at launch. In this case, it may be necessary to take measures to speed the repair and closure of defects. In hindsight, earlier discovery and closure may have prevented the higher defect count just prior to launch.

For the daily test activity shown in Figure 3, it is important to investigate swings in test activity. For example, if there is a sudden drop in daily test activity, possible causes may include a defect that blocks a large number of test cases, network/technical problems, tester turnover, or discovery of a buggy code section that requires a large amount of time to write up defect reports.

Some organizations may wish to continue collecting test metrics for some period after launch, and then observe some pre- and post-launch ratios. Escaping defects can give the team an idea of how effective their test function may be. A good starting point would be fewer than 10% of all defects can occur after launch; this number can be reduced as the organization becomes more mature.

Finally, project post-mortems offer the opportunity to measure the value test metrics collection, analysis and reporting offer during the project's life. The post-mortem gives the team the opportunity to ask questions such as: What could we do to make collection of raw data more efficient? Is there a better way to present the results? Were the actions taken in response to the trends effective? Did we look for correlations between the charts? Mature organizations will then look for ways to learn forward, and affect future projects.

Conclusions

Test metrics can benefit organizations that understand their value and commit to their execution. Especially in the web-based environment, having an accurate metrics reporting solution can help reduce cycle time by identifying trends and clearly showing transition dates. Through repeated use and refinement, test metrics can be instrumental in measuring and improving web-based software and the groups that produce it.

References

1. Schulmeyer, G. Gordon *Handbook of Software Quality Assurance*, New Jersey: Prentice Hall PTR, 1999.
2. Brooks, F. P. *The Mythical Man-Month*, New York: Addison-Wesley, 1975.

Joe Polvino

Joe Polvino has been involved with testing and software quality assurance for over 10 years. He is a senior software test engineer for Element K, with focus on its web-based product: www.elementk.com. Joe is responsible for leading and mentoring a group of test engineers, as well as providing test services and test metrics solutions. He has standardized test metrics reporting and test documentation, and is involved in software process improvement initiatives.

Prior to joining Element K, Joe worked for Eastman Kodak Company, where he performed software quality engineering on storage device drivers, firmware and digital imaging libraries.

Joe is a member of the American Society for Quality (ASQ) and is a Certified Software Quality Engineer (CSQE). He holds a B.S. in computer science from the State University of New York at Buffalo.