## HOW ARE WE GOING? GOOD TEST METRICS TO COLLECT

One of the biggest questions facing software development projects is… are we there yet?

Firstly, you need to define your concept of "completion". Are you finished when you run out of time? Or are you finished when the software meets a certain level of quality? The challenge is in achieving the latter (acceptable quality) prior to reaching the former (the deadline).

MANAGEMENT BY FACTS

Software testing is one activity that can provide visibility into product and process quality. Test metrics are among the "facts" that project managers can use to understand their current position and prioritise their activities, so that they can reduce the risk (or impact) of running out of time before the software is ready for release.

Test metrics can be a very powerful risk management tool. At the same time, however, it is important to understand a couple of basic metrics-collection pitfalls:

- Only collect data that you will actually use (to make informed decisions and alter your strategy). That is, if you were not going to change your strategy regardless of the findings, your time would be better spent doing more testing.
- Do not base decisions solely on data that is variable or can be manipulated. For example, measuring testers on the number of tests they write per day could actually reward them for speeding through superficial tests or punish them for tackling trickier functionality.

With metrics collection, timing is everything. Testers should start collecting metrics as soon as they get software from the developers that is stable enough to meaningfully run tests. If you are only testing (or collecting test metrics) near the end of the development lifecycle, then it is too late – you have lost the opportunity to use the information to make a difference.

Test metrics collection programs do not have to be extensive to be effective. We have identified eight test metrics that we collect on our testing projects. These fall into two categories – problem report (PR) information and test information.

PROBLEM REPORT INFORMATION

1. Severity:

   Rate the severity of the problems found during testing. The number of "high" severity PRs (in particular) will be an indicator of the overall quality of the product and its release readiness. You should not release functionality that contains "high" or "showstopper" bugs.

2. Affected functionality:

   Link PRs to the test step/s being run when the problem was found. Identify whether problems are clustering in particular areas of functionality (ie, which tests have uncovered high severity problems; in which tests were the most PRs found). This will help to focus attention on the weakest (ie, "buggiest") areas of functionality. As your deadline approaches (and particularly if you do not have time to fix all bugs and run all tests), use this information to prioritise your efforts so that you get the most value for the time spent.

3. Date raised, date closed:
   Track when PRs are raised and when they are closed. When you plot the (cumulative) number of PRs raised vs closed over time, the "raised" line should level off eventually and the lines should converge. That is, with each successive drop of software, fewer new problems are being found and previously raised PRs are being closed. This will tell you whether the overall quality is improving and whether problems are being found and fixed at a rate that is consistent with the schedule.

  

4. Fix rate:

   Track how long it has been taking, on average, to evaluate and remediate each PR. This will give you an indication of how long it will take to fix the remaining PRs, and how many PRs can be fixed by the planned release date. If you do not have time to fix them all, concentrate on the problems that are of high severity and that are linked to key functionality and weaker functionality.

5. Rejection rate:

   Identify the percentage of PRs that have been declared as "fixed" by the developers, but are then rejected during re-testing. Also identify how many new faults are seen while running tests that used to pass. This will indicate some measure of bug fixing efficiency and product stability, and will also provide a factor of realism to the PR fix rate. Rejected PRs could also highlight communication problems between the development and test teams, with either the testers not providing detailed enough information as to the nature of the problem or developers not taking the problems seriously enough.

TEST INFORMATION

1. Test rate:

   Track how long it has been taking to run each test and how long it takes to run the entire test suite. This will let you know how much time should be planned for re-testing and will indicate whether or not you have time to run all tests again prior to the release date. Remember that even in environments that aim to automate the testing, there will almost always be *some* tests that are manual. If you do not allow for them in your time estimates, you will probably end up not re-running them.

2. Key functionality tests:

   Identify the tests that cover high risk or high use areas (ie, key functionality). Use this information in conjunction with the "affected functionality" data to work out your priorities for fixing problems and running tests. Always try to include the "key functionality" tests in your regression test suite, particularly if these tests have uncovered high severity problems or a cluster of related problems.

3. Software version tested:

   Identify the software version under test for each test run. This will indicate how current your test results are. That is, have enough changes been made since the last time the test was run to invalidate your latest test results and warrant a regression test. It is important to track changes to the test environment, particularly in large system development environments where there can be many concurrent changes (often by someone other than the test team).

Taken together, these metrics can provide valuable insight into how the project is going. Armed with these facts, and the wisdom to use them to control the strategy (plus a bit of luck), managers can greatly improve the likelihood of developing a quality product that will be released on time.

  