Measuring and Managing In-process Software Quality

Stephen H. Kan
IBM
Rochester, Minnesota USA
skan@us.ibm.com

Abstract

Using in-process metrics to determine the quality status of a software project under development is easier said than done. How can you interpret a test-phase defect curve correctly to reflect the true quality status of the project? If the defect curve is below a given baseline, is this a positive sign? What if the lower curve is due to slow progress in testing? Likewise, how does one establish meaningful metrics for design reviews and code inspections and interpret them correctly? How about metrics for stability and reliability?

This paper describes the Effort/Outcome Model, which is a framework for establishing and interpreting in-process metrics in software development. The model has been validated and used on large scale software projects in a mature software development organization. The central issue for in-process metrics, the concept and definition of the model, and its use are discussed. Examples of metrics real-life projects are provided.

How Do I Interpret My Defect Metrics During Testing

Defect data during the testing phase of the development process is perhaps the most widely used source of data for in-process metrics. Metrics formed using test defect data can be the overall number of defects, the overall defect density, or the defect arrival curve. To measure the status of in-process quality validly, one has to interpret the in-process metrics and data correctly. Take for example the defect arrival curve, at least several characteristics have implications to the quality status of the project: the shape and level of the tail end of the curve, the time the curve peaks relative to when the product will be shipped, and release to release (or "like" product) comparisons. Just the defect metrics alone, however, may not be able to do the job (providing an accuracy status of in-process quality) adequately. What if the decline of the defect curve is due to poor testing effort?

Effort / Outcome Model -- Testing

In order to have good confidence in interpreting test defect arrival metrics, one have to rely upon the test effort related information. Indicators like testing effort, testing effectiveness and testing progress are related to the effort we extended into testing, and defect rates, defect volumes, or defect arrival curves indicate the resultant outcome based on testing effort. If we take a closer look at most in-process metrics, we can classify them into two groups: those that measures the effectiveness or effort, and those that indicate the outcome. We call the two groups the effort indicators (e.g., test effectiveness assessment, test progress S curve, CPU utilization during test, inspection effort) and the outcome indicators (e.g., defect arrivals during testing-- total number and arrivals pattern, number of system crashes and hangs, mean time to unplanned initial program load (IPL), inspection defect rate), respectively.

To establish good in-process metrics and to understand the in-process quality status of our software projects, we ought to make use of both types of indicators. We can use the two concepts to form a framework, which we call the Effort / Outcome model. The following 2 x 2 matrix (Figure 1) is a representation of the model in terms of testing effort and defect outcome:

Figure1. Effort / Outcome Matrix for Testing Effort and Defect Discovery

(Testing Coverage (Progress)) EFFORT	OUTCOME - Defects Rate (Volume)	
	Higher	**Lower**
Better	**Cell1** **Good/not bad**	**Cell2** **Best/desirable**
Worse	**Cell3** **Worst**	**Cell4** **Unsure/ not acceptable**

For the four cells of the matrix:

- Cell 2 is the best-case scenario. It is an indication of good intrinsic quality of the design and code of the software --low error injection during the development process-- and verified by effective testing.
- Cell 1 is a good/not bad scenario. It represents the situation that latent defects were found via effective testing.
- Cell 3 is the worst case scenario. It indicates buggy code and probably problematic designs -- high error injection during the development process.
- Cell 4 is the unsure scenario. One cannot ascertain whether the lower defect rate is due to good code quality or ineffective testing. In general, if the test effectiveness does not deteriorate substantially, lower defects are a good sign.

It should be noted that in the matrix, the better/worse and higher/lower designation should be carefully determined based on release to release, or actual versus model comparisons. This effort/outcome framework can be applied to pairs of specific metrics. With regard to testing and defect volumes (or defect rate), the model can be applied to the overall project level and in-process metrics level. At the overall project level, the effort indicator is the assessment of test effectiveness versus the comparison baseline, and the outcome indicator is the volumes of all testing defects (or overall defect rate) versus the comparison baseline, when all testing is complete. It is difficult to derive a quantitative indicator of test effectiveness. But an ordinal assessment (better, worse, about equal) can be made via test coverage (functional or some coverage measurements), effort (person-days in testing), extra testing activities (for example, adding a separate phase), and so forth.

At the in-process status level, the test progress S curve is the effort indicator and the defect arrival pattern is the outcome indicator. The four scenarios will be as follows:

- POSITIVE Scenarios:
  - The test progress S curve is the same as or ahead of comparison baseline (e.g., a previous release) and the defect arrival curve is lower (compared with previous release). This is the Cell 2 scenario.
  - The test progress S curve is the same as or ahead of comparison baseline and the defect arrival curve is higher in the early part of the curve -- chances are the defect arrivals will peak earlier and decline to a lower level near the end of testing. This is the Cell 1 scenario.

- NEGATIVE Scenarios:

 The test progress S curve is significantly behind and the defect arrival curve is higher (compared with baseline) -- chances are the defect arrivals will peak later and higher and the problem of late cycle defect arrivals will emerge. This is the Cell 3 scenario.

 The test S curve is behind and the defect arrival curve is lower in the early part of the curve -- this is an unsure scenario. This is the Cell 4 scenario.

Figure2. Effort / Outcome metrics – Testing Progress and Test Defect Arrivals

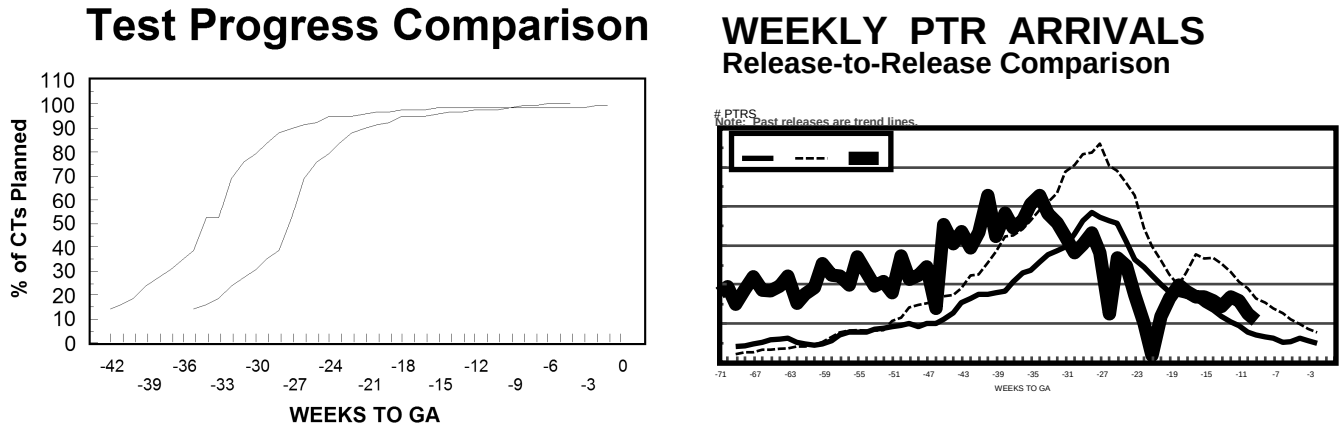

Figure 2 shows the in-process status of a software project via a pair of metrics on test progress and defect arrivals. The panel on the left hand side is the test progress release to release comparison. In this panel the curve to the left (and on top) is the test progress S curve for the current release, and the test curve to the right is the curve for the baseline release. The panel on the right hand side is the test defect arrival (PTR= problem tracking reports) release to release comparison. The thickest curve is the current release. The dotted-line curve is the comparison baseline. A third curve (the thinner solid line) is also included in the chart for reference purpose. By evaluating the pair of in-process metrics simultaneously, we know that the test progress of the current release is ahead of the baseline release (in terms of time to general availability (GA)), and the test defect arrivals in the first part of the curve was higher (than the baseline release), the curves peaked earlier, and in the second part of the curve, defect arrivals declined to a lower level (than baseline release). Therefore this is a positive scenario.

Generally speaking, outcome indicators are more common whereas effort indicators are more difficult to establish. Moreover, different types of software and tests may need different effort indicators. Nonetheless, the effort/outcome model forces one to establish appropriate effort measurements, which in turn, drives the improvements in testing. For example, the metric of CPU utilization is a good effort indicator for systems software. In order to achieve a certain level of CPU utilization, a stress environment needs to be established. Such effort increases the effectiveness of the test. The level of CPU utilization (stress level) and the trend of the number of system crashes and hangs are a good pair of effort/outcome metrics.

For integration type software where a set of vendor software are integrated together with new products to form an offering, effort indicators other than CPU stress level may be more meaningful. One could look into a test coverage-based metric including the major dimensions of testing such as:

- setup
- install
- min/max configuration
- concurrence

- error-recovery
- cross-product interoperability
- cross-release compatibility
- usability
- Double-byte character set (DBCS)

A five-point score (1 being the least effective and 5 being the most rigorous testing) can be assigned for each dimension and the sum total can represent an overall coverage score. Alternatively, the scoring approach can include the "should be" level of testing for each dimension and the "actual" level of testing per the current test plan based on independent assessment by experts. Then a "gap score" can be used to drive release-to-release or project-to-project improvement in testing. For example, assume the test strategy for a software offering calls for the following dimensions to be tested, each with a certain sufficiency level: setup (5), install (5), cross-product interoperability(4), cross-release compatibility (5), and usability (4) and DBCS (3). Based on expert assessment of the current test plan, the sufficiency levels of testing are: setup, 4; install, 3; cross-product interoperability, 2; cross-release compatibility, 5; usability, 3; DBCS, 3. Therefore the "should be" level of testing would be 26 and the "actual" level of testing would be 20, with a gap score of 6.

For application software in the external user test environment, usage of key features of the software and hours of testing would be good effort indicators, and the number of defects found can be the outcome indicator. Again to characterize the quality of the product, the defect curve must be interpreted with feature usage and effort of testing data.

Effort / Outcome Model -- Inspections

Not only the effort /outcome model can be applied to the testing phase, but it can and ought to be used for the front end of the development process with regard to design and code inspections. The effort variable is the amount of effort applied to inspections, which can be measured by hours of preparations and inspections time per unit of inspection materials, or other metrics such as the extent of expertise (experience and expertise coverage of product design and implementation of the inspectors). The outcome variable is usually measured by the inspection defect rate (number of design or code defects found per unit of inspection materials). Again, the classification of higher or lower (in inspections effort and defect rate) is based on comparison of previous releases or similar projects. The four scenarios as designed by HH, HL, LH, and LL are similar to those described in the effort/outcome matrix for testing:

- *Best case scenario (HL)—high effort/low defect rate:* an indication that the design/code was cleaner before inspections, and yet the team spent enough effort in inspections, therefore better quality was ensured.

- *Good/not bad scenario (HH)—high effort/high defect rate:* an indication that error injection may be high, but higher effort spent is a positive sign and that may be why more defects were removed. If effort is significantly higher than the model target, this situation may be a good scenario.

- *Unsure scenario (LL)—low effort/low defect rate:* an unsure scenario. Either the design and code was better, therefore less time in inspection was needed; or inspections were hastily done, hence finding fewer defects. In this scenario we need to rely on the team's subjective assessment and other information for a better determination.

- *Worst case scenario (LH)—low effort/high defect rate:* an indication of high error injection but inspections were not rigorous enough. Chances are more latent defects remain in the design or code.

Of course, in addition to the actions recommended for each scenario, the following questions, in addition to the matrix, should be asked for each inspection:

- Have all mandatory reviewers attended?
- Were the inspectors well prepared?
- Were all materials covered?
- Was the meeting time adequate for the amount and complexity of materials?

Effort / Outcome Model – Improvement Paths

Let us revisit the matrix in Figure 1 and examine the paths of improvements across different scenarios. Both Cell 3 (worst-case) and Cell 4 (unsure) scenarios are unacceptable from quality management's point of view. To improve the situation at the overall project level, the test plans have to be more effective if the project is still at the early development cycle. Or if testing is almost complete, additional testing for extra defect removal needs to be done. The improvement scenarios take three possible paths:

1. If the original scenario is Cell 3 (worst-case), then the only possible improved scenario is Cell 1 (good/not bad). This means achieving quality via extra testing.
2. If the original scenario is Cell 4 (unsure), then the improved scenario can be one of the following two:
   - Cell 1 (good/not bad) - this means more testing leads to more defect removal, and the original low defect rate was truly due to insufficient effort.
   - Cell 2 (best-case) - this means more testing confirmed that the intrinsic code quality was good, that the original low defect rate was due to lower latent defects in the code.

For in-process status, the way to improve the situation is to accelerate the test progress. The desirable improvement scenario takes two possible paths:

1) If the starting scenario is Cell 3 (worst case), then the improvement path is Cell 3 to Cell 1 to Cell 2.
2) If the starting scenario is Cell 4 (unsure), then improvement path could be:
   - Cell 4 to Cell 2
   - Cell 4 to Cell 1 to Cell 2

The difference between the overall project level and the in-process status level is that for the latter, Cell 2 is the only desirable outcome. In other words, to ensure good quality, the defect arrival curve has to decrease to a low level genuinely. If the defect arrival curve stays high, it implies that there are still substantial latent defects in the software. Testing ought to continue until the defect arrivals show a genuine pattern of decline, not due to running of test cases in the current plan. At the project level, because the volume of defects (or defect rate) is cumulative, both Cell 1 and Cell 2 are desirable outcome from a testing perspective.

Examples of Effort and Outcome Indicators along the Software Development Process

As illustrated in previous sections, the effort/outcome model for establishing and interpreting in-process metrics can and ought to be used for all phases of the development process. Figure 3 shows a list of effort and outcome indicators for the phases along the development cycle. While some of the indicators are quite specific, many of them need operational definitions to become workable metrics. Therefore, for the same indicator, there could be different metrics. Development teams can specify their operational definitions to make the indicators work in their organizations and development process.

Figure3. Examples of Effort and Outcome Indicators along the Phases of the Development Process

Development Phase	Effort Indicators	Outcome Indicators
Requirements	- Coverage - req analysis and reviews - Coverage - Customer validation	- Requirements problem/issue rate
Design	- Design reviews coverage - Design reviews - inspector-hour per review - Design reviews progress metrics	- Design defect rate - Design rework indicators
Code	- Code inspection coverage - Code inspection progress metrics - static code analysis coverage	- Inspection defect rate - Rework indicators
Unit Test/Integration/Build	- Unit test plan coverage - test progress metrics - static code analysis coverage - Effectiveness/function coverage of build verification test	- Defect metrics
FVT (Functional Verification Test)	- Test plan function coverage/usage - Test Progress metrics	- Overall test defect rate - Test defect arrival pattern - Defect backlog metrics
SVT (System Verification Test)	- Same as FVT - CPU utilization - other indicators for system stress - Coverage of business/customer usage scenarios	- Same as FVT - Metrics for crashes and hangs - Mean time between outage
Beta	- Relative program progress - % of customers on production Vs testing - Customer usage of functions	- Customer problem rate - Cust. satisfaction

Conclusions

In this paper we described the effort/outcome model for in-process metrics and quality management. The model goes beyond the traditional way of interpreting metrics and assessing quality when a software development project is underway. We illustrated the use of the model for the testing phase as well as for the front end of the development process with regard to design and code inspections. Improvement paths were discussed and possible effort and outcome indicators that can be used along the phases of software development life cycle were provided. The effort/outcome model and the metrics thus established, used, and interpreted, are clearly an effective way for quality management.

References

1. Kan, Stephen H., **Metrics and Models in Software Quality Engineering, Second Edition**, Boston: Addison-Wesley, 2002.
2. Kan, Stephen H., Jerry Parrish, Diane Manlove "In-Process Metrics for Software Testing", **IBM Systems Journal**, Vol 40, No.1, February 2001.
3. Jones, Capers, **Applied Software Measurement, Global Analysis of Productivity and Quality**, Third Edition, New York: McGraw Hill, 2008