Measuring and Assessing Software Test Processes Using Test Data*

Yutao He and Herbert Hecht
SoHaR Incorporated
8421 Wilshire Blvd. Suite 201
Beverly Hills, CA 90211
{yutao, herb}@sohar.com

Raymond A. Paul
OASD/C3I
Pentagon
Washington DC
ray.paul@osd.pentagon.mil

Abstract

Testing of large-scale software systems is a complex and expensive process that involves both technical and managerial issues. To improve its cost-effectiveness, the process should be continuously monitored, consistently measured, and carefully assessed. This paper proposes an assessment methodology, called process-oriented metrics-based assessment (POMBA), towards this direction. Novel concepts include considering test problems as one of the dependent variables of the test process and the use of test intensity as one of the independent variables. As a proof-of-concept, the methodology is applied to test data collected from high-level Y2K tests of seven large-scale transaction software systems. It is found that, in testing large-scale software, test problems that prevented completion of tests due to insufficient test planning and setup activities tend to occur frequently. This not only wastes resources but also affects the effectiveness of the overall process.

1. Introduction

Testing has been estimated to account for about 50% of the total software development cost [1], with even higher percentages being associated with high assurance systems. In spite of a huge body of research on theory and practice of software testing [2-7], the majority is focused on developing and evaluating one particular testing technique. However, testing of large-scale software systems is a complex and expensive undertaking that involves not only a set of individual testing techniques, but also such other factors as experience of testing staff, management, resource limitations, etc., which have been largely ignored in previous work. For example, as discussed later in the paper, in testing large-scale software systems *test problems* are often encountered that prevent the completion of test. To account for such events, the cost-effectiveness of testing practice should be viewed as an output of a systematic process that integrates all testing aspects and that should be assessed carefully for continuous improvement. This point has been recognized by recent efforts that aim to advance software testing through process improvement [8,9]. Consistent with this work, this paper proposes a methodology, called *process-oriented metrics-based assessment* (POMBA), that is intended to measure and assess the test process as a whole and hence to lead to the better understanding and improvement. The approach is to view test activities as a process composed of different phases that can be consistently measured by a set of metrics for test activities, software under test, and test results. In particular, *test problems* are considered as one of the dependent variables of the test process and *test intensity* is defined as one of the independent variables to measure the thoroughness of the test.

Data from DoD Y2K End-to-End (E2E) testing of seven large-scale transaction systems have been analyzed using the POMBA methodology. The tests were conducted by different test groups at several facilities during the latter half of 1999 in an environment that simulated the calendar year transition to January 1, 2000 and the leap year transition February 28 to 29. Although a general guideline for the test process was defined [10], its implementation among seven E2E test processes was far from uniform [11]. Each test group applied it as a "best-effort" without meaningful indicators to justify that the effort was indeed the best. In addition, all test documentation concentrated on proving that the software system under test satisfied its requirement in

* This work was partially supported by the U. S. Army Research Laboratory under the contract DAKF-11-99-P-127.

the Y2K environment, with no attempt to assessing the test process itself.

The rest of the paper is organized as follows. Section 2 describes the methodology and its application to the data set of seven E2E test processes. Section 3 presents analysis. Conclusions are drawn in Section 4.

2. The POMBA Methodology

2.1 Overview

The Test Maturity Model (TMM) developed in [8] characterizes a test process with five levels in order of maturity growth, *1: initial; 2: phase definition; 3: integration; 4: management and measurement;* and *5: optimization and defect prevention and quality control.* It is evident that to achieve the maturity growth, a set of relevant measures must be developed that provides consistent visibility to a complete process for continuous monitoring and assessment, and hence improving the cost-effectiveness. This is the motivation for the proposed methodology, process-oriented metrics-based assessment (POMBA).

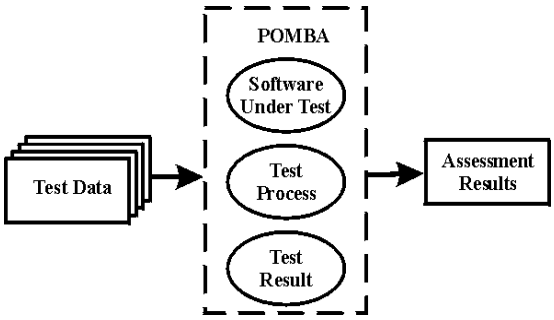

Figure 1. The Top-Level View of POMBA

As illustrated in Figure 1, POMBA takes test data (such as test reports) as input, and identifies three elements of test-related aspects, *software under test*, *test process*, and *test result*, each of which is modeled with one set of orthogonal metrics.

1. Metrics of *software under test:* This set of metrics is used to measure the size and complexity of software system to be tested. Most commonly-used software metrics such as KLOC can serve the purpose.

2. *Metrics of test process:* POMBA views test activities of a software system as a complete process that is composed of three different phases: *test planning, test setup and preparation,* and *test execution.* Each phase can be further characterized by a set of orthogonal metrics. One important measure is how thoroughly a software system is tested. A commonly used metrics, *number of test cases*, does not appear to be the best indicator on how thoroughly a software system is tested - a given number of test cases on a small and simple software system could constitute a very comprehensive test, while for a large and complex one it might only scratch the surface. In POMBA, a new metric, *test intensity,* is defined as number of test cases normalized to the software size and complexity. In addition, other qualitative metrics can be defined to measure a test process, such as the sufficiency of test planning, preparation and etc.

3. *Metrics of test result:* Most research on particular testing techniques assumes that a test is always completed. It is not the case in reality, though, especially when testing large-scale and complicated software systems that may be developed and maintained by different groups of people and deployed in different locations. In such context, in addition to the familiar pass/fail result that indicates failure of the software under test, a test case may also be aborted before it gets chance to exercise the software under test. Such a phenomenon is termed *test problem* in POMBA, that is, failure to completing a test. As the failure of a test case usually indicates a problem in the software development process, an occurrence of test problem is often attributed to deficiencies in the test process. POMBA is intended to account for both situations. ODC [12] is an effective approach to classifying test results into a useful metrics. At the highest level, a result of a test case can be divided into two categories, software failures and test problems.

At present POMBA defines a high-level and generic framework. When applied to a specific test process, the concepts outlined above should be further developed and refined from the available test data to obtain more detailed and faithful metrics.

As a proof-of-concept, POMBA has been applied to assess the Y2K End-to-End (E2E) test processes of seven large-scale DoD mission-critical transaction systems [11]. Seven test reports written by each testing group have been used as the input. The following

sections present the metrics that has been developed for these test processes.

2.2 Metrics of Software under Test

Two data items related to the software characteristics have been recorded for all seven systems: the *number of subsystems (N_s) and the number of threads (N_t)*. Each thread is a data flow path in a software system that represents a complete transaction. Since lines of source code is not available and all test cases are developed in terms of subsystems and threads, the number of subsystems is used as a metric for size, and the number of threads as a metric for complexity. Table 1 shows the software metrics of seven systems in terms of N_s and N_t. Seven test processes are designated with T1 to T7.

Table 1. Metrics of Software under Test

Metrics	T1	T2	T3	T4	T5	T6	T7
N_s	4	7	7	13	35	4	9
N_t	2	7	4	22	53	30	40

2.3 Metrics of Test Process

Analysis of test reports of seven software reports shows that all tests contain three phases identified in POMBA. There also exist some variations. For example, prior to Y2K tests in which Y2K environment were simulated, most test processes conducted the baseline test for tying up the test resources and configuration environment. Test T6 is the only exception. Some test reports also acknowledge insufficient planning and deficient setup activities. Table 2 lists three relevant indicators that have been identified in all test reports and show variation of seven test processes. The corresponding value is scored as Y for "Yes", and N for "No".

Table 2. Metrics of Test Process

Metrics	T1	T2	T3	T4	T5	T6	T7
Insufficient Planning	N	Y	Y	N	N	N	N
Deficient Setup	N	Y	Y	N	N	Y	N
Lack of Baseline Test	N	N	N	N	N	Y	N

2.4 Metrics of Test Intensity

Number of test cases (N_{tc}) is explicitly recorded for some test processes and has been estimated from associated material contained in test reports for others.

$$I_1 = \frac{N_{tc}}{N_s + N_t}$$

$$I_2 = \frac{N_{st}}{N_s \times N_t}$$

Test intensity has been defined in two ways as follows:

Denominators in the above equations represent the composite factor of the software size and complexity in two ways. Its usage is empirical yet intutively relavent.

Table 3 shows number of test cases as well as two test intensity metrics calculated for seven test processes, by using the above equations and values in Table 1.

Table 3. Metrics of Test Intensity

Metrics	T1	T2	T3	T4	T5	T6	T7
N_{tc}	32	68	40	48	212	6035	377
I_1	5.33	4.86	3.64	1.37	2.41	177.5	7.69
I_2	4	1.39	1.43	0.17	0.11	50.3	1.05

2.5 Metrics of Test Result

Using the ODC approach [12], we have developed a taxonomic model for seven test processes and classified each negative test result recorded in seven test reports. Details of the taxonomic model and its application can be found in [11]. Here we only describe the relevant model. At the highest level, a negative test result is classified as either a test problem or a software failure. A software failure is further broken down into *Y2K-related* or *non-Y2k-related*. This information has been recorded in all reports. Table 4 shows metrics of test results of seven processes: number of test problems (N_{tp}), number of software failures (N_{sf}), number of Y2K-related software failures (N_{sf1}), and number of non-Y2K-related software failures (N_{sf2}).

Table 4. Metrics of Test Result

Metrics	T1	T2	T3	T4	T5	T6	T7
N_{tp}	1	11	10	3	0	504	0
N_{sf}	1	1	2	3	5	0	0
N_{sf1}	0	0	0	0	4	4	6
N_{sf2}	1	1	2	3	9	4	6

3. Results and Analysis

With the metrics developed in Section 2, this section presents the analysis and assessment by using POMBA.

3.1 Effect of Test Preparation

With respect to whether planning or setup is sufficient, seven tests can be broken down into two groups, *deficient* or *adequate* preparation. The contrasts of their test results are listed in Table 5. It can be seen that there is a very significant effect of test preparation on the number of test problems encountered. Where preparation was deficient, the average number of test problems was more than 100 times that of the systems that had adequate preparation. Even if excluding T6, in which number of test problems is exceptional large, the average number of test problems in deficient preparation group is still 10 times that of adequate preparation. These documented test problems include errors in test planning (invalid configurations or test conditions), in setup of test environment (connections or data recording equipment), and in execution (tester error). It appears that test problems not only delayed the completion of tests (as admitted in test reports) but also reduced the effectiveness of tests – the chance of exercising the software under test has been reduced. The latter point can be observed in Table 5, the group with deficient preparation found 2.3 software failures on average, and twice less than that of in the group with adequate preparation (4.75 on average). From this perspective, the test process with adequate preparation is more effective and less costly.

Table 5. Effect of Test Preparation I

Metrics	Deficient				Adequate				
	T2	T3	T6	Ave.	T1	T4	T5	T7	Ave.
N_{tp}	11	10	504	175	1	3	0	0	1
N_{sf}	1	2	4	2.3	1	3	9	6	4.75

It could be argued that T6 caused significant large number of test problems simply because it conducted significant large number of test cases. To investigate this claim, number of test problems per test cases (N_{utp}) is calculated for all seven test processes and listed in Table 6. It can be seen that the value of N_{utp} for T6 is 0.083 and the two other test processes that have larger values of N_{utp} are T2 (0.16) and T3 (0.25). These three test processes have all been explicitly stated as having inadequate test preparation, as shown in Tables 2 and 5.

Table 6. Effect of Test Preparation II

Metrics	T1	T2	T3	T4	T5	T6	T7
N_{tp}	1	11	10	3	0	504	0
N_{tc}	32	68	40	48	212	6035	377
N_{utp}	0.031	0.162	0.25	0.063	0	0.084	0

3.2 Effect of Software Characteristics

The four smallest systems by size (N_s) are in T1, T2, T3, and T6. The numbers of test problems and software failures for these are contrasted with the three largest systems in Table 7. A similar contrast by complexity (N_t) is presented in Table 8, in which the least complex systems are in T1, T2, and T3.

Table 7. Contrast By Size

Metrics	Small					Large			
	T1	T2	T3	T6	Ave	T4	T5	T7	Ave
N_s	4	7	7	4	5.5	13	35	9	19
N_{tp}	1	11	10	504	131.5	3	0	0	1
N_{sf1}	1	1	2	0	1	3	5	0	2.7
N_{sf2}	0	0	0	4	1	0	4	6	3.3
N_{sf}	1	1	2	4	2	3	9	6	6

The much larger number of test problems for the small group suggests the dominance of test processes with deficient test preparation in this group. The only small system with adequate test preparation is T1, and its test problems are the same as the average for the large systems. The larger number of failures for large systems appears to be due to the size. The average number of failures per subsystem is 0.36 for the small group and 0.32 for the large one. The differences between Y2K failures and non-Y2K failures are too small to disclose a cause. This also holds true for the contrast by complexity in Table 8.

Table 8. Contrast By Complexity

Metrics	Less Complex				More Complex				
	T1	T2	T3	Ave	T4	T5	T6	T7	Ave
N_t	2	7	4	4.3	22	53	30	40	36.25
N_{tp}	1	11	10	7.3	3	0	504	0	126.75
N_{sf1}	1	1	2	1.3	3	5	0	0	2
N_{sf2}	0	0	0	0	0	4	4	6	3.5
N_{sf}	1	1	2	1.3	3	9	4	6	5.5

The greater average number of test problems for the more complex group is clearly due to the inclusion of

T6. The greater number of total failures in the more complex group is probably again due to a size effect (it contains all of the three systems that make up the large classification in Table 7). The average number of failures per thread is 0.3 for the less complex systems and 0.15 for the complex ones.

3.3 Effect of Test Intensity

Excluding Test T6, six test processes are clustered into two groups according to two test intensity metrics defined in Section 2.4 as shown in Tables 9 and 10. The last row in Table 9 lists number of failures per subsystem (N_{ufs}), and the last row in Table 10 lists number of failures per thread (N_{uft}). It appears that a test process with high test intensity tends to produce more failures (detects more faults) than that with low test intensity if the number of failures is normalized to system size or complexity. The effect is most pronounced in Table 10 where there is both a large contrast in test intensity and in the number of failures/threads. Which of these two metrics of test intensity is a better indicator is not clearly and remains an open question.

Table 9. Contrast by Test Intensity (I_1)

Metrics	Low				High			
	T3	T4	T5	Ave	T1	T2	T7	Ave
I_1	3.64	1.37	2.41	2.47	5.33	4.86	7.69	5.96
N_{tp}	10	3	0	4.3	1	11	0	4
N_{sf1}	2	3	5	3.3	1	1	0	0.67
N_{sf2}	0	0	4	1.3	0	0	6	2
N_{sf}	2	3	9	4.6	1	1	6	2.67
N_{ufs}	0.29	0.23	0.25	0.26	0.25	0.14	0.67	0.35

Table 10. Contrast by Test Intensity (I_2)

Metrics	Low				High			
	T4	T5	T7	Ave	T1	T2	T3	Ave
I_2	0.17	0.11	1.05	0.44	4	1.39	1.43	2.27
N_{tp}	3	0	0	1	1	11	10	7.33
N_{sf1}	3	5	0	2.67	1	1	2	1.33
N_{sf2}	0	4	6	3.33	0	0	0	0
N_{sf}	3	9	6	6	1	1	2.	1.33
N_{uft}	0.14	0.17	0.15	0.15	0.5	0.14	0.5	0.38

4. Conclusions

As seen in Table 5, good test preparation pays off in both avoidance of test problems and greater efficiency of the test (detecting more faults). The avoidance of test problems is important not only because they waste test resources and delay the completion of tests, but also because they appear to interfere with the ability of tests to detect defects.

Tables 7 and 8 show that size and complexity matter. In both conditions the larger systems had more failures, and the number of failures normalized to number of subsystems or threads was nearly the same.

Tables 9 and 10 show that systems tested with high test intensity experienced more normalized failures and thus were more efficient in finding defects.

All of these findings should be qualified by the small number of test processes examined and the differences in test reporting. However, like other research area in software engineering, the revolutionary advance in software testing methodology does not happen overnight, persistent and incremental effort and focus should be made towards the direction. In particular, a test process should be continuously measured in order to improve it. It is hoped that the POMBA methodology proposed in the paper and the publication of its application to the data set of seven Y2K test processes will lead to further efforts at examining and continuous improvement large scale software test process.

From the analysis of this data set we conclude that more emphasis needs to be placed on reporting and analysis of test problems, and on the evaluation of the effect of test problems on test results. The test intensity metric introduced in this report appears to be a potential indicator of the thoroughness of the test process. The two expressions evaluated here are entirely empirical, and better formulations will undoubtedly be found in further research.

Lack of information on the rationale for test case selection and on many details of the planning and conduct of the test handicapped further evaluation of the data from these reports [2,6]. Understandably, test reports seem to concentrate on the evaluation of the system under test rather than on the test methodology. But a small amount of effort is surely warranted to record data that are available at the test site and that can be used to improve the conduct of tests. The potential reduction in software test expenditures will be an ample reward for such efforts.

Acknowledgement

The authors wish to thank anonymous reviewers for their helpful suggestions and constructive criticism on improving the paper.

References

[1] T. C. Royer, *Software Testing Management: Life on the Critical Path*, Prentice Hall, New Jersey, 1993.

[2] G. Myers, *The Art of Software Testing*, Wiley, New York, 1979.

[3] B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, New York, 1990.

[4] B. Marick, *The Craft of Software Testing*, Prentice Hall, New Jersey, 1995.

[5] J. R. Horgan and A. P. Mathur, Software testing and reliability. In M. R. Lyu, editor, *Handbook of Software Reliability Engineering*. McGraw Hill, New York, 1996.

[6] W. E. Howden, *Functional Program Testing and Analysis*, McGraw Hill, New York, 1987.

[7] W. E. Howden, Good enough versus high assurance software testing and analysis methods. *Proceedings of the 4th IEEE High Assurance System Engineering Symposium (HASE'99)*, pp.166-175, November 1999.

[8] I. Burnstein, T. Suwanassart, and C. R. Carlson, Developing a Testing Maturity Model for Software Test Process Evaluation, *Proceedings of the IEEE International Test Conference 1996*, pp.581-589, Washington, DC, October 1996.

[9] T. Ericsson, A. Subotic, and S. Ursing, TIM – a test improvement model, Software Testing Verification & Reliability, No 7, Vol 4, pp 229-246, 1997.

[10] DoD, *DoD Year 2000 Management Plan*, December 1998.

[11] H. Hecht, Y. He, and M. Hecht, "Testing Lessons Learned from Y2K End-to-End Testing", Final Technical Report under contract DAKF11-99-P-1275, DoD Y2K T&E Directorate, Washington, DC, January 2000.

[12] R. Chillarage, "Orthogonal Defect Classification", in *Handbook of Software Reliability Engineering*, M. Lyu, editor, pp. 359-399, Prentice Hall, New Jersey, 1995.