HIGH ASSURANCE SOFTWARE TESTING IN BUSINESS AND DOD

Mehmet Sahinoglu
Department of Computer and Information Science
Troy State University Montgomery
Montgomery, AL

Coskun Bayrak
Computer Science Department
Donaghey College of Information Science and Systems Engineering
University of Arkansas at Little Rock
Little Rock, AR

Timothy Cummings
HQ Standard Systems Group
Maxwell AFB – Gunter Annex
Montgomery, AL

This paper argues that software testing can be less thorough yet more efficient if applied in a well-managed, empirical manner across the entire Software Development Life Cycle (SDLC). To ensure success, testing must be planned and executed within an Earned Value Management (EVM) paradigm. A specific example of empirical software testing is given: the Empirical Bayesian Stopping Rule (EBSR). The Stopping Rule is applied to an actual Department of Defense (DoD) software development to show potential gains with respect to archaic testing methods that were used. The result is that a considerable percentage of the particular testing effort could have been saved under usual circumstances, had the testing been planned and executed under EVM with the Empirical Bayesian Stopping Rule algorithm.

1. Introduction

Across the DoD and the general software industry, there is a drastic disparity in SDLC test planning and management. Businesses waste tremendous resources by not planning, developing, or testing software in an efficient, scientific manner. EVM is misunderstood and misused, planning is not comprehensive, and testing is not pervasive throughout the SDLC. There are methods of efficiently managing an SDLC project whereby the intensified planning and oversight will not cause a negative return on investment. These methods include sufficient planning within an EVM methodology, pervasive SDLC testing, and application of scientific rules for that life cycle testing (Cummings, 2000; Bayrak et al., 2001)

2. EVM Methodology

Within an SDLC, software project managers (SPM) strive to reduce risk while reducing the time it takes to actually develop the product and perform the tests. With a large application, planning can take a considerable amount of time. EVM suffers most often because planning must take place well before requirements are specified. Additionally, testing is either not sufficiently planned or at least not planned discretely. More often than not, the method used to test software units is to throw data at them and view the output. This is “black-box” testing. It can be a trial-and-error process. Thus, it is inefficient.

To apply EVM methods, all software products must be planned, scheduled, resourced and budgeted. What is a software product? It is any artifact of the SDLC. These products include such items as the individual requirement, the requirement specification, the module design, the interface specification, the software unit, the test plan, the test script, etc. In short, every activity in a development effort is associated with some sort of product. Consequently, all of those products can be tested. Thus, all of the items listed above can be tested for accuracy and feasibility. When products undergo this level of planning, the actual testing of those products becomes a part of production. This is of a second nature.

3. Typical SDLC Testing Management

When software undergoes archaic testing methods where testing ‘just happens’ at a particular phase near the end of the SDLC, it can never be efficiently planned and budgeted. Nevertheless, within most testing methods, such as **build all units** followed by **test all units**, a set of canned use-cases with data is input (e.g. black box). When a failure or group of failures occurs, testing halts. The programmer then corrects the condition that caused the failure and the program is recompiled for further testing. This is time-domain sequential software testing. In the archaic manner of test execution, EVM is out-the-window because testing cannot be discretely planned or efficiently managed.

A common archaic approach to testing software is a ‘shotgun’ or ‘testing-to-death’ approach. This is an approach where every conceivable functional procedure is performed on a pass/fail basis, in no particular order. Testing might begin with a random module, without consideration of sequence. The case presented in this paper is an example of ‘shotgun’ testing. A seemingly beneficial aspect of ‘shotgun’ testing is full coverage of functional scenarios. Unfortunately, it is exceedingly expensive and redundant. In addition, an SPM can never be sure that all functionality is tested, no matter how long the testing lasts. ‘Shotgun’ approaches also do not account for the validity of the end-product. There is not a high assurance of testing success in these practices.

4. A New View of Testing

In test planning, software units and objects can be viewed as pass/fail trees and branches and can be predicted and mapped. When design products are planned, the parallel testing products are also planned. Test cases are one example of a ‘Design Phase’ product. During the ‘Testing Phase’, the individual test-case reports are examples of products. The planning and mapping exercise may entail much effort, but it will reveal the redundancies and the statistical likelihood of the branching. The fact that the test planning and analysis are performed at the same time while the code is being constructed does not add extra calendar time to the project. EVM requires this.

Another way to incorporate testing into everyday operations efficiently is through a *mixed testing strategy* (Chen et al., 1999, 2000; Sahinoglu et al., 1999). This strategy allows the manager to increase testing accuracy and efficiency while keeping costs down. The *mixed* nature implies that the empirical rules can be applied in varied fashions, depending on the nature of the tests. In other words, one starts for instance with a functional testing strategy (least sophisticated) and moves to a more discriminatory testing strategy (higher sophisticated), hence the mixed testing strategy. In practice, testers switch strategies when testing yield saturates. The question they must solve is to determine the right point at which abandon on the current technique and switch to a new one. Beyond that it is about how to efficiently sequence testing techniques. There are no hard and fast rules.

4.1. The Empirical Bayesian Stopping Rule

The Empirical Bayesian Stopping Rule (EBSR) uses mathematical principles of Poisson counting process applied to the number of the test cases, with a logarithmic-series distribution (LSD) applied to the clump size of fault or coverage at each test case (Sahinoglu, 1992, 1997, 1999). It satisfactorily applies to time domain sequential software testing as well as effort testing such as the case presented here (Sahinoglu et al., 1997, 1999, 2002). There exist other works in this field (Forman et al., 1977; Singpurwalla, 1991; McDaid et al., 2001; Ross, 1985) The SPM should set up the test plan with this testing method in mind. A thorough case history of similar projects and programs should be used to arrange the test cases logically to fit the model. The testing employs a ‘convergence factor’ that can be set as high as necessary. The engineer derives the convergence factor from how well the cases are organized and from how similar the case history is to the current project. This factor is function of cost constraints (Randolph et al, 1995). The phenomenon of clustered test case failures is observed in software testing practice. Programmers may call it the ‘domino effect.’ Effectively, a series of failures can often be attributed to cause-effect. If the distribution of the total number of clumped failures fits the compound Poisson behavioral model, the Empirical Bayesian Stopping Rule can then be derived by updating the apriori parameters as the field data is collected. Based on the case histories, the SPM sets an economic criterion “d” to signify the convergence level desired to establish that a sufficient level of testing has occurred. If for the i^{th} unit interval beginning at time t or for test-group i, the expected cost of stopping is greater than or equal to the expected cost of continuing, then it is economical to continue testing for the next group of test inputs. This convergence threshold can be represented as,

$$d = \frac{c}{a - b} \tag{1}$$

where d signifies the ratio of the *cost (c)* of performing a test over the cost of catching a failure *after(a)* release minus the cost of catching a failure *before (b)* release.

On the other hand, if the expected cost, $E(X_i)$ of stopping is less than the expected cost of continuing, it is more economical to stop testing with the following strategy: $aE(X_{i+1}) > bE(X_i) + c$. If we were to stop at interval or test-group i, we assume that the cost of coverage items as yet uncovered is a per coverage item. Thus, there is an expected cost over the interval $\{i, i + 1\}$ of $aE\{X_i\}$. If we were to continue testing over the interval, we assume that there is a fixed cost of c for testing, a variable cost of b related to the covered elements, and a variable cost of a related to the uncovered elements discovered after testing. Note that a is usually larger than b. The equation in the one-step-ahead formula can be rearranged into the following form:

$$e(x) = k_{i+1}\frac{a + X_{i+1}}{b - 1 + k_{i+1}} - k_i\frac{a + X_i}{b - 1 + k_i} < d \tag{2}$$

where a and b are apriori parameters for the LSD*(q)* in the Empirical Bayesian analysis where $0<q<1$ denotes the positive-correlation-coefficient-like parameter q of LSD (Sahinoglu et al., 1997, 1999) and k is a constant to be computed at each step. l , which is used in the subsequent Tables 2 and 4, is the ratio of test cases with any activity (non-zero error) to the past experienced number of test cases.

4.2. A New Test Planning Concept

The SPM and test manager plan the test units and establish a cost of each unit test *c*. Then, they would establish the cost of catching and correcting an error prior to release *b* vs. the cost of catching and correcting an error after release *a*. Then, they would set their threshold *d*. After these variables are input, the tests are run and failures counted. At each step, the equation runs and eventually the threshold *d* will be breached. At that time, the equation will indicate that further testing of that specific nature on the set of units will not add any value. It will be up to the SPM and test manager to assess the results and make a decision to stop testing. The basis of this premise is not to intentionally field errors in code. With proper planning of units and test cases, the SPM will determine the causation-matrix of the units and will determine the likelihood of errors being introduced into the code. With this arrangement, unit testing can be arranged in an order by which the most important or most questionable units are in the front and tested first. This ordering is important to the success of managing the testing.

5. DoD Case Study

An Air Force munitions tracking system recently underwent a hardware/OS re-platforming effort. During the Unit and Integration Testing (UIT), the developers tested 31 modules in order to validate the functionality on the new platform. Those 31 test cases were arranged in order of functionality and were not correlated within the thresholds of cause and effect. It was a brute-force ‘shotgun’ approach, which required 4 full-time programmers to perform a range of tests across the 31 modules.

5.1. Quantified Account of Testing Effort

Throughout the UIT, errors occurred within certain units and were fixed before testing continued. Those are identified with cost factor “*b*”. Following the testing effort, errors were found that were quarantined, fixed and recycled through extra testing. This is identified by a cost factor “a”.

Thus, the average cost ”c” of initially testing a module equals an arbitrarily selected ‘two cost units’ (2CU) each. Since there were a total of 31 units, the average cost of catching and fixing an error before release, ”b”, was given to be an arbitrary ‘ten cost units’ (10CU) each. 10 such errors were corrected via the brute force “shot-gun” testing strategy. The total cost units for the conventionally exhaustive unit-testing activity without applying any stopping rule is 162CU as follows:

$$31c = \text{2CU X 31 modules} = \text{62CU}$$
$$10b = \text{10CU X 10 modules} = \text{100CU}$$

Table 1 Modular test strategy (1) results, ordered.

Module	# Errors	Cum. # Errors
1	1	1
2	3	4
3	1	5
4	1	6
5	0	6
6	0	6
7	1	**7**
8	1	7
9	0	8
10	0	8
11	1	9
12	0	9
13	0	9
14	0	9
15	0	9
16	0	9
17	0	9
18	0	9
19	0	9
20	0	9
21	0	9
22	0	9
23	0	9
24	0	9
25	0	9
26	0	9
27	0	9
28	0	9
29	0	9
30	1	10
31	0	10

5.2. Application of the EBSR Algorithm

For the calculation of potential savings, the software units or test cases are ordered with respect to their interdependencies. Additionally, the more critical units are placed first. This cause-effect relationship ensures that the ordering better fits the requirements for the application of the Empirical Bayesian Stopping Rule. After this ordering, the actual error counts encountered during real testing is added. See Tables 1 to 4. It is obvious by the clustering of errors that they correlate to a hypothetical ordering.

Table 2 Analysis and decision table for Table 1's strategy (1) testing.

Module	*l*	k	x	Cum.x	E(x)	e(x)	Coverage %	Decision
1	1.0	0.67922	1	1	3.64	N / A	10	CONT.
2	1.0	0.57711	3	4	4.391	0.751	40	CONT.
3	1.0	0.55345	1	5	4.632	0.24	50	CONT.
4	1.0	0.53307	1	6	4.868	0.236	60	CONT.
5	0.8	0.41763	0	6	4.124	-0.744	60	STOP

Table 3 Modular test strategy (2) results, ordered.

Module	# Errors	Cum. # Errors
1	1	1
2	0	1
3	0	1
4	1	**2**
5	0	2
6	0	2
7	0	2
8	0	2
9	0	2
10	0	2
11	0	2
12	0	2
13	0	2
14	0	2
15	0	2
16	0	2
17	0	2
18	0	2
19	0	2
20	0	2
21	0	2
22	0	2
23	1	3
24	0	3

Table 4 Analysis and decision table for Table 2's strategy (2) testing.

Module	*l*	k	x	Cum. x	E(x)	e(x)	Coverage %	Decision
1	0.0	0.0	0	0	0.0	N / A	0	CONT.
2	0.5	0.31512	1	1	2.157	2.157	25	CONT.
3	0.66667	0.40649	1	2	2.89	0.734	50	CONT.
4	0.5	0.29809	0	2	2.296	-0.594	50	STOP

The Empirical Bayesian Stopping Rule (EBSR) one-step-ahead equation indicates that testing should have stopped after the 5th unit was tested. See Tables 1 and 2. The Stopping Rule could tell an SPM this information while UIT is occurring.

Next, applying the *mixed testing strategy, and therefore using the next strategy that is more sensitive than the initial (functional) testing strategy,* the EBSR could be applied to the remainder of test cases. Beginning with Unit 6 (now the first test-case of the second testing strategy), the Java Code tells us that testing should stop after Unit 9 (now the fourth test-case of the second testing strategy) was tested. That would leave two units in error hypothetically that would be caught after the testing effort is concluded. See Tables 3 and 4.

5.3. Cost-Benefit Analysis via EBSR

A total of eleven modules, 5+4=9 in two sequential testing strategies, would have been tested during UIT (with a cost of $9c$ = 18CU). During this testing activity, nine errors, 6+2=8 out of 10, amounting to a coverage percentage of 80%, would have been detected and corrected (with a cost of $8b$=80CU). A total cost of 98CU would have incurred by applying the EBSR, as opposed to 162CU without the EBSR. Assuming that the 2 additional errors would have been found after testing, this would be corrected after-the-fact. The value of "*a*", *cost of correcting an after-release error* is, in question to judge EBSR efficient, needs to be calculated. Thus, the simple savings for number of units is now 31-9=22. By ratio, this is more than a 70% savings in the number of modules, by missing out only on two additional undiscovered errors. Now, considering that the additional errors would be caught post-testing, an optimal "*a*" can be solved to render the EBSR algorithm economic and efficient. In order for this argument to hold, the following equations must hold true:

$$98\text{CU} + 2a < 162\ \text{CU}$$

$$a < 0.5\,(162\text{CU} - 98\text{CU}) < 32\text{CU}$$

This implies that it makes economical sense to apply the Empirical Bayesian Stopping Rule when the cost of catching an error *post facto* is less than a certain upper bound. In this case, upper bound for a=32CU. That is, in order for the EBSR to be effective in this exemplary case-study, the cost of correcting an error after release can not be 3.2 times more than that of correcting it before the release of the same software.

6. Conclusion: New Methods Migrated

Migrating to new testing methods requires an initial investment of research and analysis by the software engineering agencies. The engineers must adhere to strict process and technical standards so that they can plan measurable execution criteria. It is a high, yet manageable goal. It requires EVM. The migration to this environment is a Return on Investment (ROI) that will save enormous amounts of dollars in the end, not to mention potentially increasing the quality of the product (Cummings, 2000). This holds true provided that unreasonably high and exaggerated cost factors (“a”) are not attributed to those errors post-facto (Sahinoglu et al., 2002).

The DoD and commercial software industries must migrate to more scientific and quantitative methods of software testing. In fact, DoD is mandated to follow principles of EVM right now. Unfortunately, the requirements of EVM are not normally met. The efficiencies that could be introduced would be phenomenal. Moreover, it would take Software Engineering one more step closer to a true engineering discipline. Also, the cost effectiveness is a prime concern in this work (Sahinoglu et al. 2002).

7. Acknowledgements

We would like to thank to anonymous reviewers who spent time and effort in evaluating this work.

8. References

Bayrak C., Sahinoglu M., Cummings T., 2000, “High Assurance Software Testing Business and DoD,” Proceedings of HASE-2000, Albuquerque, NM, pp. 207-211.

Cummings, T., 2000, “A New Scientific Business Engineering Paradigm for Software Agencies,” Master of Science Thesis in Computer and Information Science, Troy State University Montgomery , Montgomery, AL.

Chen T., Sahinoglu, M., Mayrhauser, A., Hajjar, A., Anderson C., 1999, “How Much Testing is Enough? Applying International Symposium on High-Assurance Systems Engineering (HASE), Washington D.C, pp. 249-256.

Chen T., Sahinoglu M., Mayrhauser, A., Hajjar, A., Anderson C., 2000, “Achieving the Quality of Verification for Behavioral Models with Minimum Effort,” Proceedings of the IEEE First International Symposium on Quality Electronic Design, San Jose, pp. 234-239.

Randolph, P., Sahinoglu, M., 1995, “A Compound Poisson Stopping Rule,” J.Applied Stochastic Models and Data Analysis, Vol. 11, pp. 135-143.

Sahinoglu, M., 1992, “Compound-Poisson Software Reliability Model,” IEEE Trans. Software Engineering, Vol. 18, pp. 624-630.

Sahinoglu, M., 1997, “Alternative Parameter Estimation Methods for the Compound Poisson Software Reliability Model with Clustered Failure Data,” STVR (J. Software Testing Reliability and Verification) Vol. 17, pp. 35-57.

Sahinoglu, M., von Mayrhauser A., Chen T., Hajjar A., Anderson Ch., 1999, “On the Efficiency of a Compound Poisson Stopping Rule For Mixed Strategy Testing,” IEEE Aerospace Proceedings, Snowmass, Colorado, Track#7.

Sahinoglu, M. Al-Khalidi, A, 1997, “A Bayesian Stochastic Stopping Rule for a Compound Poisson LSD Distribution and Application to Software Testing,” ISBA Meeting – Satellite to ISI-97, Istanbul, pp. 16-18.

Sahinoglu, M., Al-Khalidi, A., 1999, “A Stopping Rule for Time-Domain Software Testing,” Proceedings of ISSRE99, Boca Raton, Florida, pp. 11-12.

Sahinoglu M., 2002, “Sahinoglu Software Reliability Model-JPL Data,” Abstract Proceedings of The Sixth World Conference on Integrated Design and Process Technology, Doubletree Hotel, Pasadena, California; p.62.

Sahinoglu M., Glover S., 2002 “An Algorithmic Stopping Rule in Software Branch Coverage Testing”, Abstract Proceedings of The Sixth World Conference on Integrated Design and Process Technology, Doubletree Hotel, Pasadena, California; p.71.