

Integrating Defect Estimation Methods to Make Release Decisions

C. Stringfellow

Department of Computer Science
Midwestern State University
Wichita Falls, TX 76308 USA
stringfellow@mwsu.edu

A. Andrews

School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164 USA
aandrews@eecs.wsu.edu

ABSTRACT

This paper describes an integrated method using various techniques to improve testing efficiency. It uses several empirical techniques (capture-recapture methods, an experience-based method, and a selection method for software reliability growth models) and describes ways to combine the defect estimates obtained to make release decisions.

Results of a case study applying the integration method to project data from a large software product from industry demonstrate that the integrated method can improve the efficiency of system test. Based on post-release data and interviews with the test manager, the method recommended the right release decisions using estimations of remaining defect content.

KEY WORDS

testing efficiency, defect estimation, release decisions

1 Introduction

System testing is an important and costly phase of software development. In software testing, a tester is faced with two challenges: delaying the release of software when it is likely to still contain too many costly errors and limited testing time. The first challenge relates to testing effectiveness, while the second relates to testing efficiency (testing too long wastes valuable resources).

The decision to stop testing is usually controlled by management based on marketing goal. In some organizations, testing stops when testing yield saturates, where yield is defined in terms of either coverage elements found (for example number or proportion of branches covered) or number of faults (or failures) exposed. Unfortunately, it is not always clear whether the software is ready for release at that point, as it may contain many undiscovered errors. It would be helpful if testers could use testing data to determine the right point at which to stop testing and release software. There are quantitative techniques that can be used to forecast the number of problems that are likely left in the software at the time of release. Depending on the estimates, testers can make decisions on whether to

test more or recommend release, giving management the information to make the right release decisions.

Since no one tool or method works on all data, this paper proposes a way to utilize and integrate methods described in [1, 2] to aid in making release decisions. A method proposed in [1] selects the best software reliability growth models (SRGMs) to use to estimate the number of remaining defects. Several static defect content estimation techniques are analyzed in a case study in [2] and include capture/recapture models [3, 4], curve fitting models [5, 6], and a simple experience-based model. Multiple evaluations provide more credible information for decision making.

This paper also describes the application of the integrated method in a case study. The case study uses defect reporting data from several development phases for three releases of a large medical record system to evaluate current testing practices of a group of testers from industry for efficiency.

Section 2 describes techniques used in the integrated method to estimate remaining defect content. Section 3 describes the approach of the integrated method. Section 4 describes the data used in the case study, while Section 5 applies the integrated method to the data and shows how the set of methods work together to make release decisions. Section 6 draws conclusions, pointing out advantages and limitations of the integrated method and presents possible further work in this area.

2 Background

Research in the area of defect analysis on software components has focused on identifying fault-prone components or predicting the number of defects remaining in components based on their characteristics. Models that predict the number of defects in a release based on the number of errors in earlier phases of development have the potential to aid in making release decisions, thereby improving testing efficiency by potentially saving weeks of testing. The approach for the integrated method is based on several of these existing methods. This section describes these defect estimation methods. It also describes the use of defect and

failure estimates for the purpose of making release decisions.

2.1 Defect Estimation using Static Models

Several approaches that use process metrics to estimate defect content exist. Experience-based approaches are based on building models from data collected previously. Studies in [1, 7, 8, 9, 10] predict the number of defects either within one release or between releases. Yu et al. [7] investigate using the number of errors detected in earlier phases of the life cycle to predict the number of defects found later in system test and post-release. Biyani, et al. [8] show that the prior release is sufficient for predicting the number of defects during development or in the field. The study in [2] considers new functionality added during system test, which is probably a fairly common procedure in industry.

Christenson and Huang [9] and Graves, et al. [10] developed models that successfully predict the number of defects found or remaining within software based on the file changes made. Consideration of large, recent changes and changes to fix defects improve the models.

Various statistical methods [3, 4, 6, 11] use process data available only from the current project to estimate the defect content. Two types of models that can be used for this approach are:

- Capture-recapture models [2, 3, 4, 11], i.e., models using the overlap and non-overlap of defects discovered between independent testers to estimate the remaining defect content. These models have different assumptions regarding testers' abilities and the probabilities of detecting a defect.
- Curve fitting models, such as the Detection Profile Method (DPM) and the Cumulative Method [6] plot test data and fit a mathematical function to estimate the remaining defect content. These methods make less restrictive assumptions.

Wohlin and Runeson [6] evaluated the DPM and the Cumulative Method and compared them to capture-recapture models. Their results showed that the methods produced varying estimates, hence the mean value of the estimates was the most sensible approach to estimating the number of remaining defects.

Stringfellow et al. [2] apply several of these models. Their main focus is on using these methods to estimate the number of defective components in post-release. Their results show that the actual value turned out to be between the estimates provided by the $m0ml$ and the $mtml$ model [12], DPM [6] and jackknife estimator ($mhjk$) [3]. This is good as it gives lower and upper limits. These estimates can be used as additional criteria to determine when it is suitable to

stop testing and release software. Results of release decision analysis show that not only do these methods provide reasonable estimates, the estimates give a good basis for a correct release decision. An experience-based estimation method based on a simple multiplication factor presented in [2] performed very well, and for this reason, it should be used as a complement to the other estimation methods.

2.2 Methods that Integrate Experience

Several studies in defect estimation methods [11, 13] attempt to incorporate experience. Runeson and Wohlin [13] combine capture-recapture with a fault classification method and apply an experience-based multiplicative factor to faults found by a single reviewer to make them less sensitive to the composition of the inspection team.

Briand, et al. [11] address this problem by enhancing the Detection Profile Method (DPM). Their method improves on the DPM by providing selection criteria to choose between linear or exponential curves based on goodness of fit or a strict ordering criterion, taking into account the different shapes a fitted curve might have. The authors also present a selection strategy between the Enhanced Detection Profile Method (EDPM) and capture-recapture models. Results show that when using the selection strategy between the EDPM and the capture-recapture models, estimates were better in some cases than when using either EDPM alone or capture-recapture alone. In no case was the selection strategy between EDPM and capture-recapture worse.

2.3 Defect Estimation using Software Reliability Models

Many dynamic models [14, 15, 16, 17, 18, 19, 20, 21], based on various sets of assumptions about the software and its execution environment, try to assess whether some software testing objective has been met to determine when to stop testing.

Goel discussed the applicability and limitations of software reliability growth models during the software development life cycle in [14]. He proposed a procedure that selects an appropriate model based on an analysis of the testing process and a model's assumptions. The problem with this method is that, in practice, many of the models' assumptions are violated; hence none of the models are appropriate.

For the software practitioner, the assumptions or conditions for the SRGMs are an open problem, because they are often violated in one way or another. Several studies [1, 15, 22, 23] have found that even so, SRGMs perform well in practice in predicting failure rates.

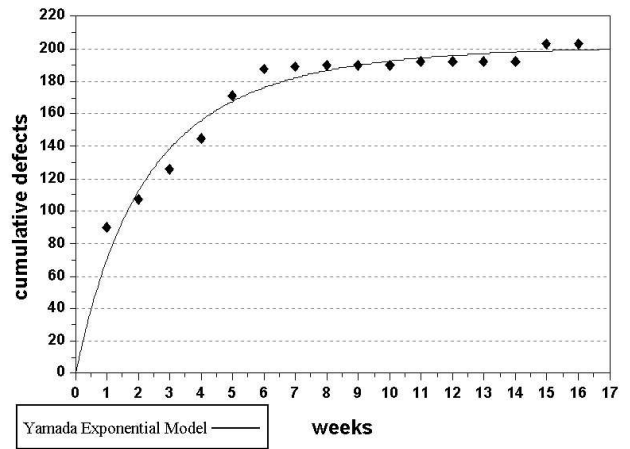
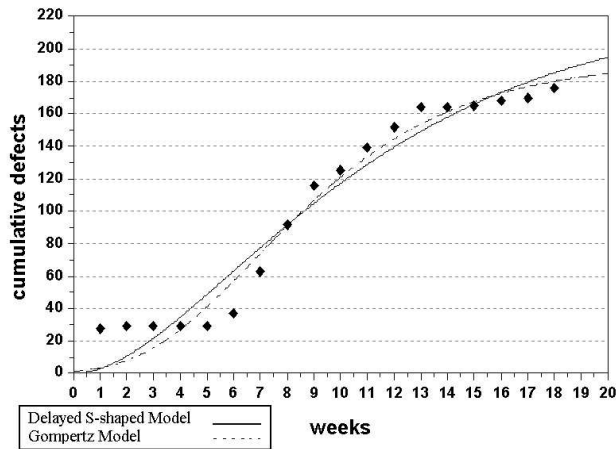


Figure 1. SRGM models not rejected in a) Release 1 and b) Release 2[1].

Stringfellow et al. [1] investigate two concave models, the G-O model [19] and the Yamada exponential model [18], and two S-shaped models, the delayed S-shaped model [17] (a modification of the G-O model) and the Gompertz model [17]. The selection method in [1] applies these SRGMs to cumulative failure data grouped by weeks to determine how well the method predicts the expected total number of failures and the expected number of failures that will occur after release. The steps in the approach are described in more detail in [1], but basically involve applying several SRGMs at the end of each week (after 60 percent of testing is complete), checking if the models' curves converge and fit well. If they do not, they are rejected. If they do and they have stabilized, then their estimates for expected number of failures are used to help make release decisions. Otherwise testing continues. The empirical selection method described in [1] aids in choosing the appropriate model, when assumptions are not met.

Results show that the selection method based on empirical data works well in choosing a software reliability growth model that predicts number of failures. Figure 1, for example, shows the plot of the failure data from Release 1 and Release 2 in their study. Figure 1a clearly shows that the data has an S-shaped curve and the correlation values show that the S-shaped models did provide a good fit to the data in Release 1. Used as a guide for system testing, the selection method suggests that system testing should have continued. This decision is supported by the opinion of one tester who believed that too many failures occurred after Release 1 and that testing should have continued a bit longer. Figure 1b shows that the concave models provided a better fit in Release 2. The selection method is robust in the sense that it is able to adjust to the differences in the data and to differentiate between the models.

3 Approach

The approach used to develop an integrated method to improve testing efficiency involves several defect estimation methods and uses defect data from testing and post-release, based on the idea that past behavior is often the best predictor of future behavior. The steps in the approach are:

1. Use static defect estimation techniques to estimate the remaining number of defective components in the software [2].
2. Apply the SRGM selection method [1] to estimate the number of failures that will occur after release. It is important to have an (iterative) selection method to determine the best model(s) for estimating the total number of failures in the software, based on the failure data collected.
3. Use the estimates to make release decisions.

The approach compares the estimates from the various estimation methods to acceptability thresholds. If the estimates are above the thresholds, the approach recommends stopping test and releasing the software.

Figure 2 shows a flowchart for applying the methods.

4 Data

The defect data in this study come from three releases of a large medical record system, consisting of 188 software components. Each component contains a number of files (ranging from 1 to over 800) that are logically related. Between three to seven new components were added in each release. Many other components were modified in all three releases.

In this study, an on-line tracking database records defect reports. For each defect the the following information

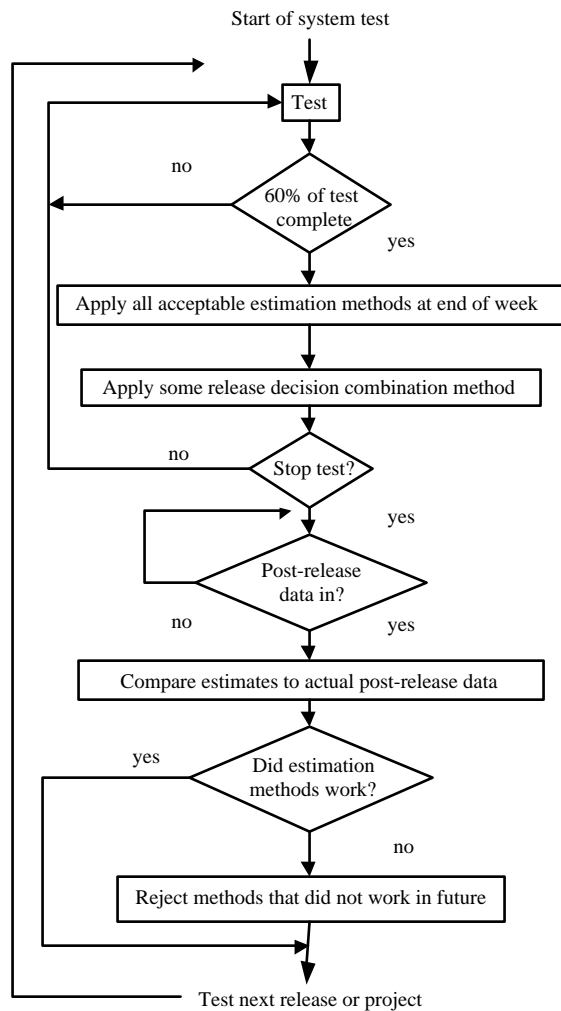


Figure 2. Flowchart for applying the integrated method.

is included: unique identifier, release identifier, phase in which defect was reported, test site reporting the defect, defective entity (code component), whether the component was new for a release, and the report date. Developers, testers, and customers have access to the data through the tracking database. The tracking database performs some validation of the data recorded.

Interviews with testers aided in understanding the testing process more fully. They helped to ascertain which factors other than testing activities could affect the data. These interviews also determined whether underlying assumptions of models were met, what the quality expectations were (this was important as it drove setting thresholds for the integrated method), and what aspects of the testing process could be changed to improve testing efficiency.

Prior to the start of system testing, the system test group performs a qualification test to determine if the system is ready for test. This reduces the number of severe

failures that would prevent testers from exercising all the code. Old components are retested after new components are added. Other interview questions revealed that system testers do not currently use stopping rule methods to make release decisions. Release decisions are schedule-driven or based on the number of defects found in the last few weeks.

The organization in this case study is very good at documenting problems. Not all developing and testing environments are of such high quality. Each project and environment has characteristics that need to be taken into account when determining what techniques to apply.

5 Integration Analysis

Since no one tool or method works on all data, the integrated method uses a set of methods that complement each other with selection criteria for each. Multiple evaluations provide more credible information for decision making. The best methods for this case study include:

- Capture-recapture methods: m0ml, mtml, and mhjk.
- Curve-fitting methods: dpm(linear).
- Experience-based method.
- SRGM selection method.

All these methods are applied iteratively (on a weekly basis) to determine whether testing should stop.

Table 1 shows the decisions based on the estimates for the static defect and SRGM selection estimation methods for Release 3. The threshold for these static methods is 5 defective components remaining. The threshold for SRGM selection method is based on number of failures remaining and is 10. The correct decisions are shown in bold.

Table 2 shows the recommended decisions for the various methods in the last week of system test for Release 3 in the case study. Because the m0ml, mtml and dpm(linear) estimators produce similar estimates, they are grouped together. The table also shows the correct release decision based on the opinion of system test group. Their opinion is based on the number of defects found and the number of failures in post-release.

These methods may be combined in several ways:

1. Logical AND, that is, all methods must say stop (the most conservative).
2. Sequential (apply one rule after another).
3. Majority (least conservative).

The first combination results in decisions to continue testing in all three releases. Based on subjective expert opinion, these decisions are correct for the first two

Table 1. Release 3 decisions earlier in test. [1, 2]

	m0ml	mtml	dpm (linear)	mhjk	exp-based	SRGM
5 weeks earlier	continue	continue	stop	continue	continue	continue
4 weeks	continue	stop	stop	continue	continue	continue
1-3 weeks	continue	continue	stop	continue	continue	stop
last week	continue	continue	stop	continue	continue	stop

Table 2. Release decisions in last week of test.

Method	Release1	Release2	Release3
m0ml	stop	stop	continue
mtml	stop	stop	continue
dpm	stop	stop	stop
mhjk	continue	continue	continue
exp-based	–	continue	continue
SRGM selection	continue	continue	stop
subjective opinion	continue	continue	stop

releases, but not the third.

A sequential combination may be defined as follows:

1. If mhjk says stop, stop. (Mjkh is the most conservative estimator.) If the mhjk says continue, go to the next rule.
2. If the SRGM selection method and at least one other estimation method says stop, stop. Otherwise go to the next rule.
3. If at least one of the following methods, m0ml, mtml, dpm(linear) and the experienced-based method say stop, stop. Otherwise go to the next rule.
4. Continue test.

This sequential combination results in the correct decisions for all three releases.

The third way of combining methods recommends stopping test, if the majority of the methods recommend stopping. Because the m0ml, mtml and dpm(linear) estimates give estimates that are close to each other, they are grouped together. If at least one of them recommend stopping, the group recommendation is to stop. For Release 1, this gives one recommendation to stop and two to continue, so the decision is to continue. For Release 2, this gives one recommendation to stop and three recommendations to continue, so the decision is to continue. For Release 3, this gives two recommendations to continue and two recommendations to stop, a tie. In the case of a tie,

the system test group may decide to be conservative and continue testing for one more week. Alternatively, they may check the number of defects found in the last week. In Release 3, only one defect was found in the last week, so testing should stop. Using the majority of methods in this way results in the correct decision for all three releases.

Using only one method to make release decisions will not necessarily work all the time. (Although the SRGM selection method works for all releases in this study, more case studies are needed to validate this.) Requiring all methods to recommend stopping is probably too conservative. A sequential combination or a majority of recommendations made by several estimation methods would be more robust.

6 Conclusion

This paper proposed an integrated set of methods to improve the efficiency of system testing. The estimation methods were successful in estimating defect content or failures after release. This paper, however, focuses more on the use of defect estimation methods in an integrated way to make release decisions. Results show that defect estimates provide a good basis for a correct decision to stop testing and release software. Because not one method works on all projects or in all environments, the integrated method recommends a strategy of applying a set of techniques that complement each other, so that it will work in more than one environment. Other methods also need to be evaluated for the purpose of making release decisions. Methods for calibrating and customizing the approach for different projects and environments are needed.

This paper empirically validated the integrated method in a case study by applying the method to three releases of a large software product. Results show that the integrated method is robust. For example, defect estimation methods applied in successive weeks result in small changes in estimates. The integrated method is easy to use - most of the techniques can be easily implemented in a commercial database and spreadsheet package, implemented in tools, or downloaded from the web. However, the integrated method needs to be evaluated in more case studies.

The integrated method provides quantitative information to guide testers in making the right release decision. Results from this case study show that both static and dynamic defect estimation methods provided good estimates of actual post-release defect content using system test data. This study was able to use the estimates to make decisions to stop testing and release software. The integrated method applied weekly can identify the point at which software is ready for release, thereby saving weeks of testing.

References

- [1] C. Stringfellow, A. Andrews, "An Empirical Method for Selecting Software Reliability Growth Models," *Journal of Empirical Software Engineering*, 7(4), 2002, 319 - 343.
- [2] C. Stringfellow, A. Andrews, Wohlin, C., and Peterson, H., "Estimating the Number of Components with Defects Post-Release that Showed No Defects in Testing," *Journal of Software Testing, Verification and Reliability*, 12(2), 2002, 93 - 122.
- [3] S. Vander Wiel, L. Votta, "Assessing Software Designs Using Capture-Recapture Methods," *IEEE Trans. on Software Engineering*, 19(11), 1993, 1045 - 1054.
- [4] C. Wohlin, P. Runeson, "An Experimental Evaluation of Capture-Recapture in Software Inspections," *Journal of Software Testing, Verification and Reliability*, 5(4), 1995, 213 - 232.
- [5] J. Musa, A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Application* (McGraw-Hill, New York, NY, 1987).
- [6] C. Wohlin, P. Runeson, "Defect Content Estimations from Review Data," *Procs. Int'l Conf. on Software Engineering*, Kyoto, Japan, 1998, 400 - 409.
- [7] T. Yu, V. Shen, H. Dunsmore, "An Analysis of Several Software Defect Models," *IEEE Trans. on Software Engineering*, 14(9), 1988, 1261 - 1270.
- [8] S. Biyani, P. Santhanam, "Exploring Defect Data from Development and Customer Usage on Software Modules over Multiple Releases," *Procs. Ninth Int'l Conf. on Software Reliability Engineering*, Paderborn, Germany, 1998, 316 - 320.
- [9] D. Christenson, S. Huang, "Estimating the Fault Content of Software using the Fix-on-Fix Model," *Bell Labs Technical Journal*, 1(1), 1996, 130 - 137.
- [10] T. Graves, A. Karr, J. Marron, H. Siy, "Predicting Fault Incidence using Software Change History," *IEEE Trans. on Software Engineering*, 26(27), 2000, 653 - 661.
- [11] L. Briand, K. El Emam, B. Freimut, "A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method," *Procs. Ninth Int'l Conf. on Software Reliability Engineering*, Paderborn, Germany, 1998, 32 - 41.
- [12] D. Otis, K. Burnham, G. White, D. Anderson, "Statistical Inference from Capture Data on Closed Animal Populations," *Wildlife Monographs*, (62), 1978.
- [13] P. Runeson, C. Wohlin, "An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections," *Empirical Software Engineering: An Int'l Journal*, 3(4), 1998, 381-406.
- [14] A.L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Trans. on Reliability*, 11(12), 1985, 1411 - 1421.
- [15] J. Musa, A. Ackerman, "Quantifying Software Validation: When to Stop Testing," *IEEE Software*, 1989, 19 - 27.
- [16] J. Musa, "Applying Failure Data to Guide Decisions," *Software Reliability Engineering* (McGraw-Hill, New York, NY, 1998).
- [17] S. Yamada, M. Ohba, S. Osaki, "S-Shaped Reliability Growth Modeling for Software Error Detection," *IEEE Trans. on Reliability*, R-32(5), 1983, 475 - 478.
- [18] S. Yamada, H. Ohtera, H. Narihisa, "Software Reliability Growth Models with Testing Effort," *IEEE Trans. on Reliability*, 35(1), 1986, 19 - 23.
- [19] A.L. Goel, K. Okumoto, "A Time Dependent Error Detection Model for Software Reliability and Other Performance Measures," *IEEE Transaction on Reliability*, R-28(3), 1979, 206 - 211.
- [20] B. Littlewood, D. Wright, "Some Conservative Stopping Rules for the Operational Testing of Safety-Critical Software," *IEEE Trans. on Software Engineering*, 23(11), 1997, 673 - 683.
- [21] M. Yang, A. Chao, "Reliability-Estimation & Stopping-Rules for Software Testing Based on Repeated Appearances of Bugs," *IEEE Trans. on Reliability*, 44(2), 1995, 315 - 321.
- [22] A. Wood, "Predicting Software Reliability," *IEEE Computer*, 29(11), 1996, 69 - 78.
- [23] A. Wood, "Software Reliability Growth Models: Assumptions vs. Reality," *Procs. of the Int'l Symposium on Software Reliability Engineering*, 23(11), 1997, 136 - 141.