C. Stringfellow, A. Andrews, C. Wohlin and H. Petersson, "Estimating the Number of Components with Defects Post-Release that Showed No Defects in Testing", Software Testing Verification and Reliability, Vol. 12, No. 2, pp. 93-122, 2002.

Estimating the Number of Components with Defects Post-Release that Showed No Defects in Testing

C. Stringfellow[1], A. Andrews[1], C. Wohlin[2], H. Petersson[3]

[1]Computer Science Department
Colorado State University
Fort Collins, CO 80523
USA
aaa@cs.colostate.edu. stringfellow@cs.nmhu.edu
+1 970-491-7016(ph) +1 970-491-2466(fax)

[2]Software Engineering
Department of Software Engineering and Computer Science
Blekinge Institute of Technology
SE-372 25 Ronneby
Sweden
Claes.Wohlin@bth.se
+46-457-385820(ph) +46-457-271-25(fax)

[3]Department of Communication Systems
Lund University
Box 118
SE-221 00 Lund
Sweden
hakan.petersson@telecom.lth.se
+46 (46) 222 3329(ph) +46 (46) 14 58 23(fax)

Abstract

Components that have defects after release, but not during testing, are very undesirable as they point to "holes" in the testing process. Either new components were not tested enough, or old ones were broken during enhancements and defects slipped through testing undetected. The latter is particularly pernicious, since customers are less forgiving when existing functionality is no longer working than when a new feature is not working quite properly.

Rather than using capture-recapture models and curve-fitting methods to estimate the number of remaining defects after inspection, these methods are adapted to estimate the number of components with post-release defects that have no defects in testing. A simple experience-based method is used as a basis for comparison. The estimates can then be used to make decisions on whether or not to stop testing and release software.

While most investigations so far have been experimental or have used virtual inspections to do a statistical validation, our investigation is a case study. This case study evaluates how well the capture-recapture, curve-fitting and experience-based methods work in practice. The results show that the methods work quite well. A further benefit of these techniques is that they can be applied to new systems for which no historical data is available and to releases that are very different from each other.

1 Introduction

Software developers are concerned with finding defects as early as possible in the development of software. Certainly, they would prefer to find defects in system test rather than after release of the software. There is also a concern whether the specific components in which the defects appear are old or new. Defects in new components that add functionality are more acceptable to users than defects in old components. Defects in old components indicate that existing functionality was broken. Old components that have no defects in system test but have defects reported after release concern developers and testers, because it may indicate that added functionality has had a negative effect on some of the old components and current test approaches are unable to detect this. It is useful to know how many components fall into this category at the end of testing as this information can be used to make release decisions.

Experience-based estimation methods [2, 11] use historical data to predict the number of defects remaining in components. The data may come from either an earlier phase within the same release or from prior releases. Capture-recapture models [9, 14, 16] and curve-fitting methods [16, 5] estimate the number of remaining defects without relying on historical data.

Traditionally, capture-recapture models and curve-fitting methods are used to estimate the remaining number of defects based on inspection reports from several reviewers. This paper applies these methods in a novel way to estimate the number of components that have defects after release but are defect-free in system test. The estimation is based on data provided by test groups from different test sites. Each test site takes the role of the "reviewer" in the models.

While most investigations so far have been experimental or have used virtual inspections to do a statistical validation, our investigation is a case study. In the case study, several test groups (at the developer site and the customer site) test the software in parallel. This means that they are "reviewing" the same system. Similarly, one could use this approach by defining subgroups within a system test group that test the same software. The case study illustrates how well this approach works. It applies the methods to three releases of a medical record system. A simple experience-based method is also presented and used for comparison.

Section 2 provides background on two major approaches to defect content estimation. Both experience-based approaches and methods based on capture-recapture models and curve-fitting methods are described. Section 3 describes the approach used to apply capture-recapture and curve-fitting methods to estimate the number of components that have defects in post-release, but do not have defects in system test. Application of these methods is illustrated in a case study in Section 4. The estimates and stopping decisions based on the estimates are analyzed. Section 5 summarizes the results of the case study and presents conclusions.

2 Background

2.1 Defect estimation

Two major approaches to defect content estimation exist. First, it is possible to build prediction models from historical data to identify, for example, fault-prone components either within the same release or between releases [2, 1, 8, 11, 17]. Second, prediction models can be built using various statistical methods using data available only from the current release [3, 4, 5, 7, 9, 10, 13, 14, 15, 16].

The first approach is referred to as experience-based, since it is based on building models from data collected previously. The second approach is used to estimate the fault content with data from the current project, and hence the methods are more direct and relevant. Two types of models can be used for this approach, namely capture-recapture models [3, 4, 5, 7, 9, 10, 13, 14, 15] or different curve-fitting approaches [5, 13, 16]. These models can briefly be described as follows:

- Capture-recapture models, i.e., models using the overlap and non-overlap between reviewers (or test sites in our case) during defect detection to estimate the remaining defect content. The models have their origin in biology where they are used for population estimations and management [7, 12].
- Curve fitting models, i.e., models that plot test data from the test sites and fit a mathematical function. The function is then used to estimate the remaining defect content [16].

2.2 Experience-based estimation methods

Methods that use historical data for predicting the number of defects remaining in components may be based on defect data [2, 17] or code change data [1, 8, 11]. The data may come from either a prior phase within the release or from prior releases. The models in [2, 1, 8, 11, 17] assume that repair is imperfect.

Yu, et al. [17] and Biyani and Santhanam [2] explore the relationship of the number of faults per module to the prior history of a module. Their results show that defect data from development is useful in predicting the number of faults in a module after release. Biyani and Santhanam [2] also show that defect data from previous releases, and from the prior release alone, are good measures for predicting the number of defects found in a module during development or after release.

Basili and Perricone [1] compared new modules to modified modules in assessing error distributions. They found that new and modified modules behaved similarly with respect to the number of defects, although the types of defects and the efforts to fix them differed.

Christenson and Huang [8] investigated a "fix-on-fix" model to predict the number of defects remaining in software. A "fix-on-fix" is a software change to fix a defect introduced in an earlier software fix. This model performed reasonably well in predicting the number of remaining defects. The fewer the number of defects in fixed code, the fewer the number of remaining defects in a module.

Graves, et al. [11] developed a model that attempts to predict the number of defects found within a module based on the file changes made to the module. They investigated several other models based on defect data and product measures for comparison purposes. They found a good and very simple model, based on defect data, which assumes that the number of future faults in a module is a constant multiplier of the number of faults found in the module in the past. In cases where code has had a number of changes, models based on size did not perform well in predicting the number of faults, and models based on other product measures were no better. They found a sucessful model based on a process measure, which involves changes to code within a module. The number of changes to the code in a module over its entire history appeared to be a successful predictor of faults. In addition, when large, recent changes were weighted more heavily than smaller, older changes, the model improved.

2.3 Capture-recapture models

In this paper, the focus is on applying capture-recapture models to data from different test sites. The major application of these types of models in software engineering has so far been during the review or inspection process. Thus, the description below is adapted for testing.

Different capture-recapture models use different assumptions regarding test sites and defects. Test sites may have the same or different ability to find defects. Defects themselves may be equally difficult to find or they may vary in how difficult they are to detect. Thus, capture-recapture models come in four categories:

1. Test sites are assumed to have the same ability to find defects, and different defects are found with the same probability. This type of model is denoted M0. It neither takes variations in the test sites' ability nor in the detection probabilities into account.

2. Test sites are assumed to have the same ability to find defects, though different defects are found with different probabilities. This type of model is denoted Mh (variation by heterogeneity[1]). It takes the detection probabilities into account, but not the test sites' ability.

3. Test sites are assumed to have different ability to detect defects, and all defects are found with the same probability. This type of model is denoted Mt (variation by time[2]). It takes the test sites' ability into account, but not varying detection probabilities.

4. Test sites have different profiles for detecting defects, and different defects are found with different probabilities. This type of model is denoted Mth (variation by time and heterogeneity). It takes both variations in the test sites' ability and in the detection probabilities into account.

The assumptions for the four types of models are illustrated in Figure 1 for five defects and three test sites. The heights of the columns represent detection probability. The actual probabilities in the figure

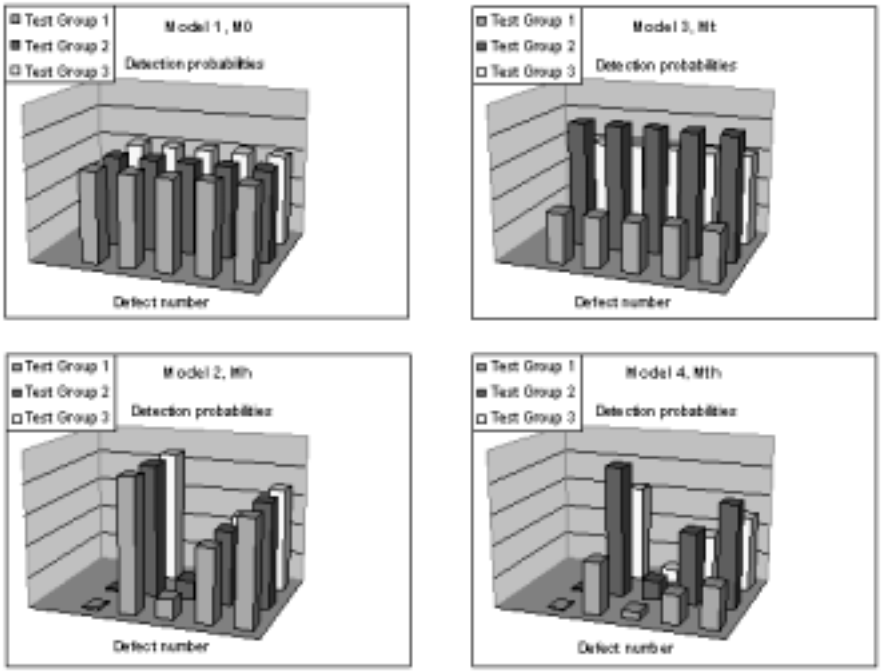

Figure 1: An illustration of the different types of capture-recapture models. The plot is inspired by a similar figure in [4].

are of minor interest. Clearly, model 4 is the most realistic. It also requires more complicated statistical methods and it is more difficult to get stable estimates.

Statistical estimators exist for all models. Table 1 shows the capture-recapture models suitable for inspections [3, 4] along with their estimators. The Chapman estimator for the Mt model (mtChpm) is used in the case of two test sites. It it is assumed that test sites work independently. For more details regarding the models refer to the references in Table 1.

[1]The use of the words heterogeneity and time has its origin in biology.
[2]See footnote 1.

Table 1: Statistical methods in relation to the different types of capture-recapture models.

	Detection probabilities	
Test site ability	Equal	Different
Equal	M0: Maximum-Likelihood (m0ml) [12]	Mh: Jackknife (mhjk)[12]
Different	Mt: Maximum-Likelihood (mtml)[12] Chapman (mtChpm)[10]	Mth: Chao (mthChao)[7]

2.4 Curve fitting models

The basic principle behind the curve fitting models is to use a graphical representation of the data in order to estimate the remaining defect content. Two models have been proposed [16]:

1. Decreasing model: Models based on plotting the detected defects versus the number of test sites that found the defects. The defects are sorted in decreasing order with respect to the number of test sites that found a defect. This means that the plot can be approximated with a decreasing function. Both exponentially and linearly decreasing functions have been evaluated. The exponential model is introduced in [16], and the linear model is proposed in [5] as a way of coping with data sets where the exponential model failed.

2. Increasing model: Models based on plotting the cumulative number of defects found versus the total number of detection events. For example, if the first defect is detected by five test sites and the second by four test sites, then the first bar is five units high and the second bar is nine units high. The defects are sorted in the same order as for model 1. However, plotting cumulative defects leads to an increasing approximation function. An increasing exponential model is proposed in [16].

Wohlin and Runeson [16], obtain estimates from the curves as follows:

- Decreasing model: The estimated defect content is equal to the largest defect number (that is, the largest integer value on the x-axis) for which the curve is above 0.5. The estimators for this model are based on the Detection Profile Method (dpm); the dpm(exp) estimator is based on an exponential curve fit and the dpm(linear) estimator is based on a linear curve fit.

- Increasing model: The remaining defect content is estimated as the asymptotic value of the increasing curve minus the cumulative number of defects found so far. The estimator for this model is based on the Cumulative Method.

In this paper, the main focus is on using the capture-recapture and curve-fitting methods to estimate the number of components with defects in post-release that showed no defects in testing. This is different from the approach taken when identifying fault-prone components. The objective here is not to estimate fault-proneness as such, but to estimate the number of components that seem fine during testing but exhibit problems during operation. The estimate can be used to determine whether it is suitable to stop testing and to release software.

3 Estimating Number of Problem Components

During maintenance and evolution of systems, testers and developers alike worry about negative effects of enhancement and maintenance activities. Of particular concern are old components that appear fine

during development and test of a release, but have problems after the software is released. If testers were able to estimate how many components are likely to exhibit such problems, this knowledge could be used to decide whether to release the system or test it further. The test manager could set a threshold of how many such components are acceptable. If the estimate falls below, software is ready to be released. If the estimate is higher, more (thorough) testing is advisable.

The approach described here applies methods based on capture-recapture models and curve-fitting methods to estimate the total number of components that have defects in system test and post release. From this estimate, one can derive the expected number of components that will have defects after release that were defect-free in system test. The approach uses defect data from groups at different test sites. These methods are used differently here than they have been in prior studies, in which they were primarily used to estimate the number of defects remaining after inspection by several reviewers.

Data is often available from several test sites, such as a system test group at the developing organization, an internal customer, an external customer, or an independent quality assurance organization. If this is the case, it is possible to use capture-recapture and curve fitting models to estimate the number of components that will have problems after relese, but were defect-free during testing. Each test site reports the components for which defects were found.

The steps in the approach are:

1. Collect defect data for components from the different test sites at the end of each week. For each test site, a component is given a value of 0, if the test site has never reported a defect for it. Otherwise the component is given the value of 1.

2. Apply capture-recapture and curve-fitting estimation methods to the data. The estimates give the sum of

   - the number of components with defects found in testing, plus
   - the number of components that are expected to have defects in operation even though they were defect-free in testing.

   Several estimators used in capture-recapture models (m0ml, mtml, mhjk, mthChao and mtChpm, see Table 1) and curve-fitting methods (dpm(exp), dpm(linear), cumulative) are applied to the data. Because the mtChpm estimator is used in the case of two reviewers [10], it is also evaluated.

3. Apply our proposed experience-based estimation method to the data. This method is based on simple multiplication factors and is applied to releases for which historical data is available.

   The experience-based method uses a multiplicative factor that is calculated by using data from previous releases and applied to the current release. This factor is used to estimate the number of components that have defects in post-release that were defect-free in testing.

   This estimate refers to the number of "missed" components, those components for which system test should have found defects. This estimate can be compared with the ones obtained using the capture-recapture and curve-fitting methods.

   The following steps are used to compute the number of "missed" components.

   (a) Calculate the multiplicative "missed" factor as follows:

   $$f_i = \sum_{k=1}^{i-1} p_k / \sum_{k=1}^{i-1} t_k \qquad (1)$$

   where

i is the current release.

f_i is the multiplicative factor for release i

p_k is the number of components with defects after release that were defect-free in test in release k.

t_k is the number of components that were defect-free in testing in release k.

(b) Multiply the number of components that are defect-free in testing in the current release by the factor computed and take the ceiling of the number.

$$\lceil f_i * t_i \rceil \tag{2}$$

4. Calculate the number of components that are expected to have defects in the remainder of system test and in post-release that are currently defect-free in testing by subtracting the known value of the number of components with defects found in testing so far.

5. Compare the estimated number of components that are not defect-free but for which no defects have been reported in system test to a predetermined decision value. Use the information as an input to decide whether or not to stop testing and to release the software.

   In a study in [10], estimates have been used for review decisions. By contrast, the estimates in this study are used for release decisions. Estimates of the number of components with post-release defects that do not have defects in system test may be used as one of several criteria when deciding whether to stop test and release software. In this context, it is more important that the correct decision is made than that the estimate is completely accurate.

   The estimate must be checked against some threshold to make a decision. For example, if defects were found in 50 components and no defects were found in 60 components during system testing and estimation methods predict that 10 components out of the 60 would have defects after release, would testing continue or stop? Threshold values reflect quality expectations. These can be common for all projects or specific to a particular type of project, such as corrective maintenance, enhancement, safety-critical, etc.

While the availability of defect data from several test groups or test sites makes it possible, in principle, to apply existing capture-recapture and curve-fitting models, it is by no means clear whether the estimators work well in practice. Thus we present a case study to answer the following questions:

- Which estimators work best?
- How robust are these estimators when only two independent test sites are available?
- Should the focus be on using only one estimate?
- At what point in the testing process can/should these estimates be applied?

To answer these questions, the estimators are evaluated using two types of measures:

- Estimation error (including relative error [3] and mean absolute relative error [4]).
- Decision error (is the decision to release or to continue testing made correctly).

  The decisions based on the expected number of components with defects in post release, but not in system test, are evaluated and compared to the correct decisions using three different threshold values. The number of correct decisions is the number of times the decisions based on the estimates predict the correct decisions.

[3] Relative error is defined as (estimate - actual total)/actual total

[4] Mean absolute relative error is defined as the absolute value of the mean relative errors for all three releases

The estimators are ranked and analyzed to see whether they exhibit similar behavior for the data sets. In general, the Mh model using the mhjk estimator has worked best for applications published within the software engineering field so far [3]. That does not, however, mean that it is necessarily the best for this specific application, since the models are used in a new context.

A related question is: at what point in the test cycle should or could one apply these estimates for decision making purposes? We attempt to answer this question by using the estimators up to five weeks before the scheduled end of testing. As before, we evaluate the quality of the estimator and the quality of the decision made.

4 Case Study

Section 4.1 describes the defect data used in this case study. Section 4.2 evaluates the estimation methods. Section 4.3 evaluates the estimation methods and the release decisions when they are applied in the last six weeks of testing.

4.1 Data

The defect data come from a large medical record system, consisting of 188 software components. Each component contains a number of files that are logically related. The components vary in the number of files they contain, ranging from 1 to over 800 files. There are approximately 6500 files in the system. In addition, components may contain other child components. Initially, the software consisted of 173 software components. All three releases added functionality to the product. Between 3 to 7 new components were added in each release. Over the three releases, 15 components were added. Many other components were modified in all three releases. Of the 188 components, 99 had at least one defect in Releases 1, 2 or 3.

The tracking database records the following attributes for defect reports:

- release identifier
- phase in which a defect was reported (e.g., development, system test, post-release)
- test site reporting defect
- defective component
- whether the component was new for a release
- the date the defect was reported
- classification of defect (The classification indicates whether the defect is valid or not. A defect is valid if it is repeatable, causing software to execute in a manner not consistent with the specification. The system test manager determines if a defect is valid or not. Only valid defects are considered in this study. A valid defect that has already been reported by someone else is a "duplicate" defect).

In addition, interviews with testers ascertained that assumptions for the capture-recapture models are met: Three system test sites receive the same system for testing at the same time. The three sites and their main responsibilities are:

1. The system test group at the developing organization tests the system against design documents.
2. The internal customer tests the system against the specifications.
3. The external customer tests the system with respect to their knowledge of operational use.

These different views may have the effect of reducing the overlap of non-defect-free components. Perspective-based reading [13] shows that the capture-recapture models are robust with respect to different views. This is important, because different test sites focus on different testing goals for the software.

Interviews with the testers uncovered other factors that affect the estimates. First, the internal customer test site may have under-reported defects. The primary responsibility of this test site is writing specifications. In addition to writing specifications, they test against the specifications. Because of this, the number of defective components is estimated with and without this site. This way we also evaluate how well the methods perform when data exists for only two test sites.

Second, some scrubbing of the data occurred: before reporting a defect, the personnel at the test sites were encouraged to check a centralized database to avoid reporting duplicate defects. Some defects reported are later classified as duplicates, but the partial prescreening has the effect of reducing overlap and hence increasing estimates. This is one reason why we "lifted" the analysis to the component level, i.e. we do not estimate the number of defects, we estimate the number of components with defects.

This prescreening has a greater impact on components with few defects. For example, if a component has only one defect and a test site finds and reports the defect, no other test site will report any other defects (and are not likely to report the same defect). The component will be categorized as defect-free for those other test sites, reducing overlap. One way to compensate for this problem is to look at estimators that tend to underestimate. If overlap is reduced due to prescreening, estimates will be higher. Estimators that tend to underestimate will compensate for defect scrubbing.

It should be noted that the experience-based method takes scrubbing into account. Experience-based models adjust to the data. If the scrubbing is done in a similar way for all releases, the estimates should be trustworthy.

Table 2 shows the actual values for components with defects and components without defects for test and post-release. Actual values are shown for all three releases using data from three test sites. In cases where data is different for two test sites, corresponding values are given in parentheses. The values in column 6 (the actual number of components with defects in test and post-release) are used to evaluate the estimates obtained in the study. The values in column 5 (the actual number of components that have defects in post-release that did not have defects in system test) are used to evaluate the release decisions.

Table 2: Release data for 3 test sites (2 sites are in parentheses).

	# all components	# components defect-free in test	# defective components in test	# defective components in post release not in test	total # defective components
Release 1	180	128	52 (51)	7	59 (58)
Release 2	185	125	60 (59)	5	65 (64)
Release 3	188	154	34	6	40

Columns 3 and 5 of Table 2 also show the data from Release 1 and Release 2 used to compute the multiplicative factors and the estimates for Release 2 and Release 3, respectively. Table 3 shows the computation for the multiplicative factors using data from Release 1 and applied to Release 2, and data from both Releases 1 and 2, applied to Release 3. Since data from more than one previous release is available, the cumulative data over the previous releases is used. It is possible, however, to use only one previous release that is similar to the current release.

Release decisions are based on the estimated number of remaining defective components. This estimate is derived by subtracting the actual number of defective components found from the estimate of the total

Table 3: Multiplicative factors for all releases.

	Multiplicative Factor	
	3 Sites	2 sites
Release 1	–	–
Release 2	7/128	7/129
Release 3	(7+5)/(128+125)	(7+5)/(129+126)

number of defective components. The estimated number of remaining defective components is compared against three thresholds to determine the correct release decision. To illustrate the quality of the the estimators in making decisions, thresholds of 2, 5, and 10 defective components were chosen. These values were chosen after an interview with the system test manager. He indicated that for this system, two to five defective components would be acceptable, but ten defective components would not be. (Threshold values should be chosen based on quality expectations for the system.)

In order to evaluate the estimates and the release decisions based upon them, we compare them to the actual number of defective components and the actual remaining number of defective components. Table 4 shows the actual number of defective components found in test in earlier test weeks, as well as the actual number of defective components remaining (found in later test weeks and post-release). As the number of found defective components (shown in column 2) increases, the number of remaining defective components decreases. The actual numbers of defective components (shown in columns 3, 5 and 7) are compared to the thresholds to determine the correct release decisions for Releases 1, 2, and 3. Values in parenthesis are for two test sites. (The number of defects found for most of the weeks using only two test sites is one less than it was for three test sites in Release 1. In Releases 2 and 3, the actual number of defects found for all of the weeks using only two test sites is the same as it was for three test sites.)

Table 4: Actual values used to determine correct release decisions.

	Release 1		Release 2		Release 3	
	# defective comps in test	# defective comp. rem.	# defective comps in test	# defective comp. rem.	# defective comps in test	# defective comp. rem.
5 weeks earlier	48 (47)	11 (12)	56	9	29	11
4 weeks earlier	48 (47)	11 (12)	56	9	31	9
3 weeks earlier	49 (48)	10 (11)	56	9	33	7
2 weeks earlier	49 (48)	10 (11)	57	8	33	7
1 week earlier	49 (48)	10 (11)	60 (59)	5	33	7
Actual end date	52 (51)	7	60 (59)	5	34	6

In Release 1, four to five weeks earlier than the actual end of system test, the number of defective components remaining is 11 (or 12 using two test sites). Because this number is larger than all three thresholds, the correct answer at all three thresholds is to continue testing. Using three test sites, one to three weeks before the end of test, the correct answer at thresholds 2 and 5 is to continue testing and at the threshold of 10, the correct answer is to stop testing. Using two test sites, the correct answer one to three weeks before the end of test is to continue testing, as the number of defective components is 11 and

greater than all three thresholds. Using both three and two test sites, the correct answer at the actual end of system testing indicates that testing should continue if the threshold is 2 or 5 and stop if the threshold is 10. One would expect that as more weeks of testing occur, one would see more suggestions to stop testing. These decisions are used to evaluate the decisions made based on the estimates.

Table 4 shows there were a few weeks that had no change in the number of components found to have defects. For example, in Release 3, there was no change 1 to 2 weeks prior to the end of system test. All components were available for test during this time. They did not come in late, nor were any new components integrated during this time. Defects were found in these weeks, but they were found in components that already had reported defects.

4.2 Estimation evaluation

4.2.1 Evaluation of estimates using three test sites

Figure 2 shows the evaluation of the estimates for all three releases for three test sites. (Negative values are a result of underestimation.) The results are encouraging: the relative errors show that most estimators provide estimates close to the actual values, except for mtChao.

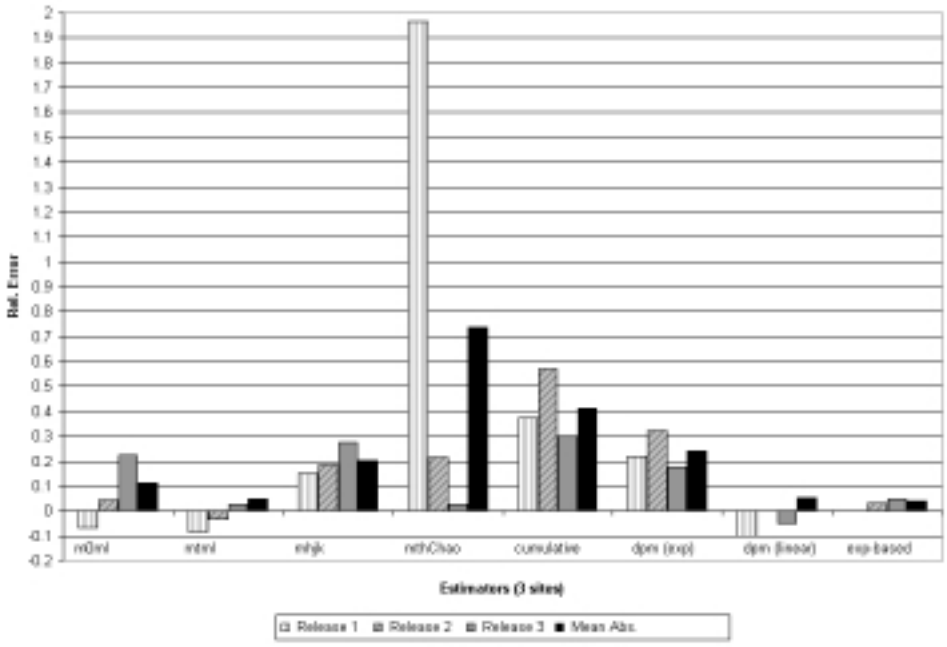

Figure 2: Relative errors and mean absolute relative error for all releases (3 sites).

χ^2 analysis showed that only one estimator, the mthChao, was significantly different from the actual release values and the other estimators' values ($p = 0.001$). The mthChao, therefore, is not recommended. The statistical null hypothesis (the estimator's values are the same as the actual values) is not rejected for all other estimators ($p = 0.001$). χ^2 analyis of the other estimators show they are not statistically different from each other. They produce estimates in the same range. This indicates that the models (except for MthChao) produce good estimates, i.e., they are not random numbers. They are also fairly similar to the estimates obtained for the experience-based model.

In Release 1, the actual value was between the estimates given by the m0ml, mtml, dpm (linear) and mhjk methods. The m0ml, mtml, and dpm (linear) estimators normally have a tendency to underestimate.

The partial scrubbing of defect data for duplicates may have reduced the overlap of the test groups leading to higher estimates for some of the estimators. Estimates that usually tend to underestimate (m0ml, mtml) worked well for this situation: They did not underestimate quite so much. Both dpm estimators basically plot data and do not have the same assumptions normally involved in capture-recapture models, hence they are less sensitive to varying defect detection probabilities or test site abilities. The mhjk has shown promising results in software engineering before [3] and is what we would have guessed to be the best estimate. It overestimated here, probably because of the partial scrubbing of data. Thus, it is good that the actual value turned out to be between the estimates provided by these four estimators as it gives us lower and upper limits.

Figure 2 shows that the second and third releases have similar results. In Release 2 and Release 3, the same four capture-recapture and curve-fitting estimators (dpm(linear), mtml, m0ml and mhjk) perform the best. Actual values were close to their estimates, frequently occurring between the estimates provided by the mtml or dpm(linear) and mhjk methods. The mtml and dpm(linear) estimation methods slightly underestimated and the mhjk slightly overestimated. The cumulative, dpm(exp) estimators still overestimate greatly.

The mthChao, the cumulative, and the dpm(exp) estimators did not perform well in most of the releases. They tended to overestimate greatly. The mthChao greatly overestimated in the first, but performed well in Release 3. Its inconsistency, however, makes it difficult to trust.

The experienced-based method performed very well in the second and third releases. It is interesting to compare these results with the results for the capture-recapture and curve-fitting estimation methods. The capture-recapture estimates were close to the experience-based estimates and are quite good.

Table 5 shows the results of evaluating the estimators using the mean of the absolute relative error to rank the estimators over all three releases (where a rank of 1 is the best). The experienced-based method performs the best overall. The mthChao and cumulative do not perform well. The capture-recapture and curve-fitting estimators that perform the best are the mtml, dpm(linear), m0ml, and mhjk. These estimators have relative errors close to the experience-based method and show that they are as good as a method that requires history.

Table 5: Ranking of estimators over three releases (3 sites).

Estimator	Ranking
m0ml	4
mtml	2
mhjk	5
mthChao	8
cumulative	7
dpm (exp curvefit)	6
dpm (linear curvefit)	3
experience-based	1

These results indicate that capture-recapture and curve-fitting methods, in particular the mtml, dpm(linear), m0ml, and mhjk estimators, are able to estimate quite well the total number of components that have defects in test and post-release. The expected number of remaining components with defects that were defect-free in testing can then be computed from these estimates.

4.2.2 Evaluation of estimates using two test sites

Figure 3 shows the evaluation of the estimation methods applied to data from two test sites for all three releases. The two test sites include the system test group at the developing organization and the external customer, which is probably the more common situation. Naturally, one would expect capture-recapture and curve-fitting methods to produce less accurate predictions as the number of reviewers (here, test sites) shrinks. A simulation study of two reviewers by El Emam and Laitenberger [10] shows that capture-recapture models may be used successfully for two reviewers. Our results are quite reasonable and in correspondence with theirs. Several of the estimators have low relative errors.

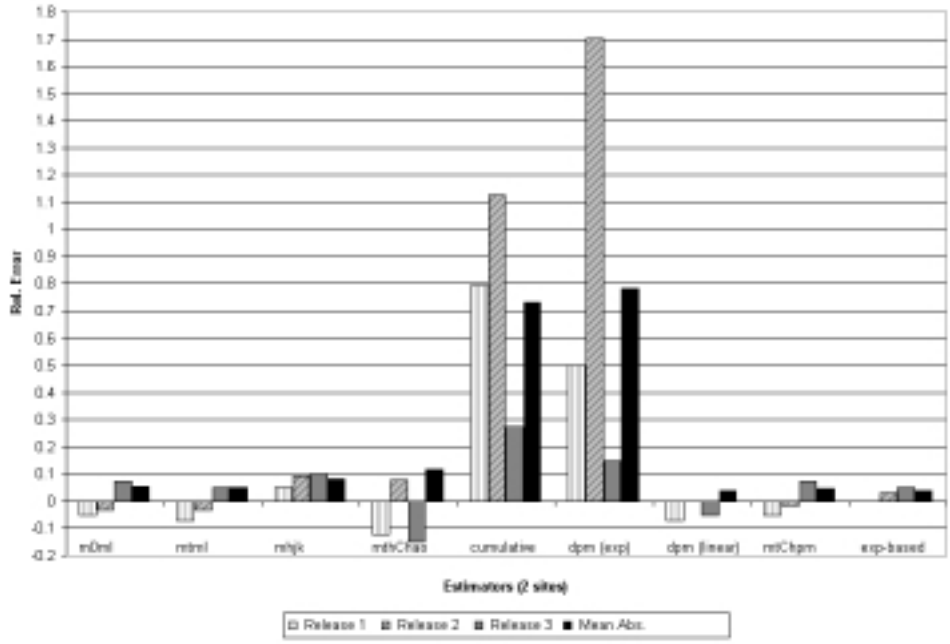

Figure 3: Relative errors and mean absolute relative error for all releases (2 sites).

χ^2 analysis shows that none of the estimators were significantly different from the actual values or from each other ($p \leq 0.20$). Using data from only two test sites, the mtml, m0ml, dpm(linear), mtChpm, and the mhjk estimators performed the best in terms of their errors. The m0ml, mtml, and mtml estimators again tended to underestimate slightly. The mhjk slightly overestimated. The MtChpm estimator, which may be used only in the case of two test sites, performed very well and is comparable to the m0ml and mhjk. The mthChao, cumulative and dpm (exp) did not perform well using two test sites. The cumulative and dpm(exp) overestimated too much. The mthChao, which usually tends to overestimate, underestimated in Releases 1 and 3. In fact, it gave the lower bound for estimates, the number of components that are known to have defects in testing. As such, the mthChao method provided an estimate that is suspect. This effect may be due to the fact that data from only two test sites was used. In any case, the estimates it provided were so low, that its relative error ranks it as one of the worst performing estimators. The mthChao estimator is not useful in this situation.

As in the case of three test sites, the mean of the absolute relative error across the three releases is used to evaluate and rank the estimators using two test sites. Table 6 shows the ranking of the estimation methods applied to data from two test sites for all three releases. The experience-based method and the dpm (linear) rank the highest overall. The mthChao, cumulative, and dpm (exp) rank lowest. The m0ml, mtml, mhjk and mtChpm have mean absolute relative errors that are less than 0.100. They are almost as

good as the experience-based method and do not require any history.

Table 6: Ranking of estimators over three releases (2 sites).

Estimator	Ranking
m0ml	5
mtml	3
mhjk	6
mthChao	9
cumulative	8
dpm (exp curvefit)	7
dpm (linear curvefit)	1
mtChpm	4
experience-based	2

The experience-based method depends on releases being similar. If historical data is available and releases are similar, the experience-based method using the multiplicative factors should be used. The capture-recapture and curve-fitting methods are independent of data from previous releases. Several of the estimates from the capture-recapture and curve-fitting methods have relative errors that are almost as low. These include the mtml, dpm(linear) and the mjhk. If no historical data is available or releases are dissimilar, these capture-recapture and curve-fitting methods provide reasonable estimates.

The results for the estimation methods based on capture-recapture models do not worsen for two test sites; in most cases they improve. This is due to the fact that the third test site had less overlap with the first two test sites than the first two test sites had with each other. The cumulative and dpm(exp) estimators, which are curve-fitting methods, worsen (as expected) with one less test site. Most unexpectedly, the dpm(linear) estimator, which is also a curve-fitting method, performs very well and improves with only two test sites. The curve-fitting methods are most useful when there are a number of reviewers and several reviewers find most of the defects[3].

These results demonstrate that capture-recapture and curve-fitting methods are able to estimate remaining defective components well when only two test sites are involved in system testing. Since this is probably a more common situation, this is good to know.

4.2.3 Evaluation of estimates obtained earlier in system test

Estimates obtained in earlier weeks and the decisions based on those estimates were evaluated for all three releases using three test sites and two test sites. Given that some of the estimators performed rather badly so far, not all estimators were applied in earlier test weeks. The m0ml, mtml, mhjk, and dpm(linear), the mtChpm estimators and the experienced-based method are the only ones applied and evaluated. The estimates from the mthChao, dpm(exp) and cumulative methods were not considered. They recommended testing to continue too long.

Each week during testing a few more components are found to have defects. A few of the 0's in the data (indicating a component does not have a defect) become 1's (indicating a component has a defect). Over the successive weeks of system test, the estimates change slightly (indicating fewer defects are remaining in later weeks). This is an indication of stability. In all releases, estimates tended to approach the actual number of defective components in test and post-release as the estimators were applied in later weeks. As more testing occurred, estimation errors decreased. For example, in Release 1, all the estimators

underestimated until the last week of system test. In earlier weeks, all the estimators, except for mhjk, gave estimates that were below the actual number of defective components found. The mhjk estimates were slightly higher than the others. In Release 3, the mtml started underestimating and then by the end of system test overestimated slightly.

We also applied the experience-based method to Release 2 and Release 3 in a similar approach using data available at earlier points in time. The estimates for defective components relies on the multiplicative factor based on data from earlier releases, as well as the number of components that have been found to have defects in earlier weeks. Because the previous release determines the multiplicative factor used in the current release, the factor's value does not change when applying it to earlier weeks of data in the release for which estimates are derived. Applying the experienced-based method at earlier weeks in system test provided estimates that were quite good. Estimates were close to the estimates obtained by other estimation methods applied at earlier weeks. The experience-based tends to underestimate at earlier weeks, then slightly overestimates closer to the actual end of system test.

Results indicated that capture-recapture methods hold up well in earlier weeks for the case of two test sites. In Release 1 and Release 2, the m0ml and mtml estimates for two test sites were approximately the same as for three test sites. The mhjk and the dpm (linear) performed better with two test sites. The mhjk overestimated less using two test sites and the dpm (linear) underestimated less. Using data from two test sites then did not worsen the performance of the estimators and in some cases improved them.

In Release 2, the m0ml and mtml estimators performed worse in the case of two test sites than in the case of three, but still had relative errors less than 0.10. The mtChpm estimator was only slightly better than the m0ml and mtml estimators. The dpm (linear) overestimated in earlier weeks, but had smaller errors than when it underestimated using data from three test sites.

Overall, the mhjk estimator, which overestimated in the case of three test sites, still overestimated, but not quite as much. The m0ml and mtml performed about the same in both cases. The mtChpm performed as well as the m0ml and mtml estimators. The curve fitting method, dpm (linear), was not consistent: In Release 1 it performed better, in Release 2 it performed worse, and in Release 3 it performed about the same.

The decision to stop or continue testing at the end of a particular week, however, is based on the actual number of defective components found in system test at that point in time and it is the decision made, rather than the estimate, that is of most concern.

4.3 Release decisions based on estimates

Interviews conducted with the developers indicated that it is acceptable to have two components with no reported defects in system test, but with post-release defects. More than 10 such components is unacceptable. Specifically, defects in new components that add functionality are more acceptable than defects in old components. It is most important that old functionality has not been broken. Most of the new components are found to have defects in system test. Only old components had post-release defects, but no defects in system test. These are exactly the components the developers worry about.

Given that some of the estimators performed rather badly, not all estimators are used. The m0ml, mtml, mhjk, and dpm(linear) estimators are used for the case with three test sites. These estimators, as well as the mtChpm estimator, are evaluated for the case with two test sites. The estimates from the mthChao, dpm(exp) and cumulative methods are not considered as their relative errors are too high and would cause testing to continue too long.

The release decisions based on the estimates are compared to the correct decision based on the actual values and evaluated.

4.3.1 Release decisions using three test sites

Tables 7 – 9 show the decisions based on the estimates and the number of correct decisions for the four estimators m0ml, mtml, mhjk and dpm (linear) using three test sites for Releases 1, 2 and 3. The correct decisions are shown in bold.

Table 7: Release decisions using 3 test sites for Release 1 at earlier points in time in system test.

5 weeks earlier	Threshold			# correct decisions
	2	5	10	
m0ml	stop	stop	stop	0
mtml	stop	stop	stop	0
mhjk	**continue**	**continue**	**continue**	3
dpm (linear)	**continue**	stop	stop	1
4 weeks				
m0ml	stop	stop	stop	0
mtml	stop	stop	stop	0
mhjk	**continue**	**continue**	stop	2
dpm (linear)	**continue**	stop	stop	1
2-3 weeks				
m0ml	stop	stop	**stop**	1
mtml	stop	stop	**stop**	1
mhjk	**continue**	**continue**	**stop**	3
dpm (linear)	**continue**	stop	**stop**	2
1 week				
m0ml	stop	stop	**stop**	1
mtml	stop	stop	**stop**	1
mhjk	**continue**	**continue**	**stop**	3
dpm (linear)	**continue**	stop	**stop**	2
last week				
m0ml	**continue**	stop	**stop**	2
mtml	stop	stop	**stop**	1
mhjk	**continue**	**continue**	continue	2
dpm(linear)	stop	stop	**stop**	1

4.3.2 Evaluation of release decisions in last week of system test (3 sites)

To illustrate the quality of the estimators in making decisions, thresholds of 2, 5 and 10 were chosen and evaluated. The estimate of the number of components with defects in post-release but not in test is compared against these three thresholds to determine whether the decision made, using the estimate, is correct. If, for example, the threshold value is 2, then testing would stop if the estimated number of components with defects in post-release that were defect-free in testing was less than or equal to 2. The correct answer is 7 in Release 1. So the correct *decision* would be to continue testing. If, on the other hand, the threshold value is 10, the correct decision would be to stop testing, since 7 is less than 10.

Table 8: Release decisions using 3 test sites for Release 2 at earlier points in time in system test.

3-5 weeks earlier	Threshold			# correct decisions
	2	5	10	
m0ml	**continue**	**continue**	**stop**	3
mtml	stop	stop	**stop**	1
mhjk	**continue**	**continue**	continue	2
dpm (linear)	**continue**	stop	**stop**	2
experience-based	**continue**	**continue**	**stop**	3
2 weeks				
m0ml	**continue**	**continue**	**stop**	3
mtml	stop	stop	**stop**	1
mhjk	**continue**	**continue**	continue	2
dpm (linear)	**continue**	stop	**stop**	2
experience-based	**continue**	**continue**	**stop**	3
1 week				
m0ml	**continue**	continue	**stop**	2
mtml	**continue**	**stop**	**stop**	3
mhjk	**continue**	continue	continue	1
dpm (linear)	**continue**	**stop**	**stop**	3
experience-based	**continue**	continue	**stop**	2
last week				
m0ml	**continue**	continue	**stop**	2
mtml	**continue**	**stop**	**stop**	3
mhjk	**continue**	continue	continue	1
dpm(linear)	**continue**	**stop**	**stop**	3
experience-based	**continue**	continue	**stop**	2

Tables 7, 8 and 9 show the results of the decision analysis in the last week of system test for all three releases using data from three test sites. In Release 1, the m0ml and mhjk estimators provide the correct decision most often. The m0ml recommends stopping a bit too soon. The mhjk recommends continuing testing a little too long. Testing a bit too long, in most cases, is probably preferable to stopping too soon. In Release 2, the mtml and dpm(linear) perform the best, both providing three correct decisions. The decisions based on m0ml and mhjk would result in continuing testing a bit longer than the other estimators. In Release 3, mtml is the only estimator that leads to three correct decisions. The decisions based on the dpm(linear) estimator result in a recommendation to stop testing too early. The decisions based on m0ml and mhjk result in recommendations to continue testing longer than the others.

Not only do these methods provide reasonable estimates of the number of components that will have post-release defects, but no defects in system test, the estimates give a good basis for a correct release decision for the three threshold values analyzed. The mtml estimator makes the largest number of correct decisions for all three threshold values both in Release 2 and Release 3. In Release 1, it recommends stopping too soon. The mhjk estimator consistently recommends to continue testing, because it typically overestimates. In all releases, the mjhk estimator would cause testing to continue until a threshold value of about 15-16.

Table 9: Estimates and decisions using 3 test sites for Release 3 at earlier points in time in system test.

5 weeks earlier	Threshold 2	Threshold 5	Threshold 10	# correct decisions
m0ml	**continue**	**continue**	**continue**	3
mtml	**continue**	**continue**	stop	2
mhjk	**continue**	**continue**	**continue**	3
dpm (linear)	**continue**	stop	stop	1
experience-based	**continue**	**continue**	**continue**	3
4 weeks				
m0ml	**continue**	**continue**	continue	2
mtml	**continue**	stop	**stop**	2
mhjk	**continue**	**continue**	continue	2
dpm (linear)	**continue**	stop	**stop**	2
experience-based	**continue**	**continue**	continue	2
1-3 weeks				
m0ml	**continue**	**continue**	continue	2
mtml	**continue**	**continue**	**stop**	3
mhjk	**continue**	**continue**	continue	2
dpm (linear)	**continue**	stop	**stop**	2
experience-based	**continue**	**continue**	continue	2
last week				
m0ml	**continue**	**continue**	continue	2
mtml	**continue**	**continue**	**stop**	3
mhjk	**continue**	**continue**	continue	2
dpm(linear)	**continue**	stop	**stop**	2
experience-based	**continue**	**continue**	**stop**	3

It is probably preferable to continue testing too long rather than stopping testing too soon and releasing the software. Because of this, the preferred estimator would be one that provides a slight overestimate. Analysis of the estimates provided by the methods shows that the mhjk estimator tends to slightly overestimate in this situation. (Other estimators that overestimate, do so too much.) An estimator that slightly overestimates will best support making the correct decision. This analysis is supported by the opinion of one tester who believed that too many defects were found after Release 1 and Release 2 and that testing should have continued a bit longer.

Estimations provided by the experience-based method, using multiplicative factors, are also analyzed from a decision point of view. Because estimations using such a method can only be made for releases with historical data, decisions for stopping test based on estimations can only be made for Release 2 and Release 3. Compared to decisions based on estimation from the capture-recapture and curve-fitting methods, the experience-based method works quite well. In Release 2, the experience-based method gives us decisions on a par with m0ml. Since testing a bit longer is preferable to stopping testing too soon, this method provides us with a conservative, but not too conservative decision. In Release 3, the experience-based method gives us decisions on a par with mtml. A more conservative method like the mhjk would recommend testing a little longer than mtml and the experience-based method.

4.3.3 Evaluation of release decisions earlier in system test (3 sites)

Table 7 shows that the m0ml, mtml and dpm (linear) estimators recommend stopping testing as early as five weeks before the end of system test in Release 1. These decisions do not agree with the correct answer. The mhjk estimator makes the correct decisions at the threshold values for 2 and 5 for the last 6 weeks for system test. At the threshold value of 10, it makes the correct decision at every week but two. These two weeks are the fourth week before the actual end of system test and the last week of system test. In these cases, the mhjk estimator recommends that testing continue when the correct answer is to stop. The mhjk estimator is, therefore, a little conservative.

For an example of how this method works, consider the following scenario. Assume that the threshold is 10 components and the mhjk estimator is used to make decisions to continue or stop testing in Release 1. Five weeks before the actual end of system test, the recommendation based on the mhjk estimator is to continue testing. This is the correct decision at this time. The following week (4 weeks earlier), the recommendation is to stop testing. The decision is incorrect: It should recommend testing to continue. It would be better if testing continues until a stop decision consistently occurs a certain number of weeks in a row. If we assume that testing continues until three stop decisions occur in a row, testing would correctly continue, because only one stop decision has occurred at this point. The following week (3 weeks earlier), the recommended decision is to stop testing. This recommendation occurs again at two weeks before the actual end of system test. Testing would now have had three weeks in a row in which a stop was recommended. If testing stopped now, they would be making a correct decision. This decision results in saving two weeks of testing. There is a potential that some defects would be missed, but it would be within the threshold set at 10.

Tables 8 and 9 show that most of the estimators, except for mhjk, improve in the second and third releases. The mhjk estimator recommends testing continue for all weeks at all thresholds. The mhjk decisions agree with the correct answer in the earlier weeks, but are perhaps conservative in the last two weeks. The m0ml and experienced-based method perform better than the mhjk estimator in making decisions in Release 2.

Mtml and the dpm (linear) recommend stopping testing too soon for at least two of the thresholds in Release 2. In Release 3, the dpm (linear) estimator incorrectly recommends stopping six weeks before the end of system test at thresholds 5 and 10. Mtml recommends stopping two to five weeks before the actual end of system test and then in the last two weeks recommends testing continue at the threshold of 2. (A threshold of 2 is very sensitive to changes in estimates.) In Release 3, the mtml estimator also incorrectly recommends stopping too soon in the fourth week before the actual end of system test at the threshold level 5 and then, in the last 3 weeks, recommends testing to continue, reversing its decision.

Tables 7 – 9 show that when m0ml, mtml and dpm (linear) provide incorrect decisions, they typically indicate that testing should stop when the correct answer is to continue. Whenever the mhjk estimator is incorrect, it most often says to continue when the correct answer is to stop.

Table 10 ranks the estimators based on the number of correct decisions over the last six weeks (the last week and the previous five weeks) for three test sites. The columns indicate the estimator ranks for all three releases and an overall rank. Table 10 shows that the experience-based method performed very well in Release 2 and Release 3, in which historical data was available. The overall rankings show the m0ml and mhjk estimators perform the best in making correct decisions. When a conservative decision is desired, the mhjk estimator should be used; otherwise the m0ml should be appropriate.

The m0ml and mhjk estimators and the experienced-based estimation method perform very well when used to make decisions. The m0ml and experience-based estimation methods tend to recommend stopping sooner than the mhjk estimation method. If system testers want to save testing time, the m0ml and experience-based methods should perform well in providing information to aid in making the decision to continue or stop test. The mhjk estimation method is recommended in situations in which system testers

Table 10: Ranks for estimators for three test sites.

Estimator	Release 1 Rank	Release 2 Rank	Release 3 Rank	Overall Rank
m0ml	3	1	3	1
mtml	3	4	1	2
mhjk	1	4	3	2
dpm(linear)	2	3	5	4
experience-based	–	1	2	–

want to be more conservative – that is, they would prefer to continue testing longer in order to have fewer defects reported in post-release.

4.3.4 Release decisions using two test sites

Tables 11, 12 and 13 show the results of the decision analysis for all three releases using data from only two test sites. The Chapman estimator is included in the decision analysis as it had low relative errors. The correct decisions are shown in bold.

4.3.5 Evaluation of release decisions in last week of system test (2 sites)

Tables 11 – 13 show that based on data from two test sites, the m0ml, dpm(linear) and mtChpm estimator lead to the correct decision for two threshold values in Release 1. The m0ml, mtml, dpm(linear) and mtChpm perform well in Release 2; all three lead to three correct decisions. All the estimators, except for dpm (linear), perform equally well at all thresholds earlier in system test in Release 3. The mhjk estimator leads to three correct decisions in Releases 1 and 3, but in Release 2, it results in an incorrect recommendation to continue testing. Again, it is probably more desirable to continue testing too long than not to test long enough. The m0ml, mtChpm and mjhk estimators appear to be the type of estimators to best support the correct decision for two test sites.

Comparing decisions based on estimations using two test sites for Release 2, the experience-based method performs better than the capture-recapture and curve-fitting methods. It recommends testing a little longer than the correct answer, but not as long as mjhk, which was determined to be the preferred estimator in Release 2. In Release 3, the experience-based method performs as well as several of the other estimators, including the mhjk.

Overall, the experience-based method performs very well when used to make decisions. Decision making based on estimates using some of the capture-recapture and curve-fitting estimation methods results in decisions that are as good. If historical data is available and releases are similar with regards to defects and their exposure, the experience-based estimation method should be used to complement capture-recapture and curve-fitting estimation methods and provide more input into making the decision to stop or continue testing. If no historical data is available or releases are not similar, capture-recapture and curve-fitting methods may be used to make good decisions based on the estimates they provide.

4.3.6 Evaluation of release decisions earlier in system test (2 sites)

The same kind of analysis was performed using data from two test sites to evaluate the estimators' release earlier in test. Tables 11 – 13 show the decisions based on the estimates and the number of correct decisions for the five estimators m0ml, mtml, mhjk, mtChpm and dpm (linear) for two test sites for all releases.

Table 11: Estimates and decisions using 2 test sites for Release 1 at earlier points in time in system test.

5 weeks earlier	Threshold			# correct decisions
	2	5	10	
m0ml	stop	stop	stop	0
mtml	stop	stop	stop	0
mhjk	**continue**	stop	stop	1
mtChpm	stop	stop	stop	0
dpm (linear)	**continue**	stop	stop	1
4 weeks				
m0ml	stop	stop	stop	0
mtml	stop	stop	stop	0
mhjk	**continue**	stop	stop	1
mtChpm	stop	stop	stop	0
dpm (linear)	**continue**	stop	stop	1
1-3 weeks				
m0ml	stop	stop	stop	0
mtml	stop	stop	stop	0
mhjk	**continue**	stop	stop	1
mtChpm	stop	stop	stop	0
dpm (linear)	**continue**	**continue**	stop	2
last week				
m0ml	**continue**	stop	**stop**	2
mtml	stop	stop	**stop**	1
mhjk	**continue**	**continue**	**stop**	3
dpm(linear)	**continue**	**continue**	continue	2
mtChpm	**continue**	stop	**stop**	2

Tables 11 and 12 show the decisions based on estimates earlier in system test are not very good in the first and second releases. Most of the estimators incorrectly recommend stopping at the three threshold values. Mhjk and dpm (linear) provide the greatest number of correct decisions 2–5 weeks before the actual end of system test. They recommend testing continue when other estimators incorrectly recommend testing should stop. The quality of decisions for the other estimators only improves for the last week of testing. Table 13 shows that all estimators, except for dpm (linear), perform equally well in Release 3 at all thresholds earlier in system test. Most of the estimators correctly recommend testing continue at the thresholds of 2 and 5 for all weeks. The same estimators make the correct recommendation at a threshold of 10, recommending testing at this threshold stop about five weeks earlier.

Table 14 ranks the estimators based on the number of correct decisions over the last six weeks for two test sites. The columns indicate the estimator ranks for each release and an overall rank. The mhjk, the m0ml and the mtChpm perform the best overall. The mhjk, however, is consistently ranked 1 or 2 for all 3 releases using data from two test sites. Based on this analysis, the mhjk is recommended for use in the case of two test sites to make decisions to continue or to stop testing.

The m0ml and the mhjk estimators ranked the highest in making release decisions, whether three test sites or two test sites were used. The mhjk estimator tends to be more conservative, recommending testing

Table 12: Estimates and decisions using 2 test sites for Release 2 at earlier points in time in system test.

3-5 weeks earlier	Threshold			# correct decisions
	2	5	10	
m0ml	stop	stop	**stop**	1
mtml	stop	stop	**stop**	1
mhjk	**continue**	**continue**	**stop**	3
mtChpm	stop	stop	**stop**	1
dpm (linear)	**continue**	**continue**	continue	2
2 weeks				
m0ml	stop	stop	**stop**	1
mtml	stop	stop	**stop**	1
mhjk	**continue**	**continue**	**stop**	3
mtChpm	stop	stop	**stop**	1
dpm (linear)	**continue**	**continue**	continue	2
1 week				
m0ml	**continue**	**stop**	**stop**	3
mtml	**continue**	**stop**	**stop**	3
mhjk	**continue**	continue	**stop**	2
mtChpm	**continue**	**stop**	**stop**	3
dpm (linear)	**continue**	continue	continue	1
last week				
m0ml	**continue**	**stop**	**stop**	3
mtml	**continue**	**stop**	**stop**	3
mhjk	**continue**	continue	continue	1
dpm(linear)	**continue**	**stop**	**stop**	3
mtChpm	**continue**	**stop**	**stop**	3

to continue, when the correct answer is to stop. If a conservative decision is desired, one should use the mhjk estimator. Otherwise the m0ml would be appropriate.

In this case study, the best results come from using two test sites rather than three test sites. (Because the internal customer test site may have under-reported defects, it may have affected the results for three test sites.) For the environment represented by this study, two test sites are recommended. Since the methods perform well when data exists for only two test sites, it is feasible to use our approach with two sites, which may be a more common situation. If there are three test sites and all three test sites have good defect data, then three test sites may be recommended.

5 Conclusions

This study evaluated the use of several existing methods to estimate total and remaining defects in code in a new context. Several test groups or sites concurrently test a software product. Developers want to know whether their software is ready for release, and how many components will exhibit defects after release that had not shown defects during system test. This gives an indication of how many components were "missed" (tested inadequately) during system test.

Table 13: Estimates and decisions using 2 test sites for Release 3 at earlier points in time in system test.

5 weeks earlier	Threshold 2	Threshold 5	Threshold 10	# correct decisions
m0ml	**continue**	**continue**	stop	2
mtml	**continue**	**continue**	stop	2
mhjk	**continue**	**continue**	stop	2
mtChpm	**continue**	**continue**	stop	2
dpm (linear)	**continue**	stop	stop	1
4 weeks				
m0ml	**continue**	**continue**	**stop**	3
mtml	**continue**	**continue**	**stop**	3
mhjk	**continue**	**continue**	**stop**	3
mtChpm	**continue**	**continue**	**stop**	3
dpm (linear)	**continue**	stop	**stop**	2
1-3 weeks				
m0ml	**continue**	**continue**	**stop**	3
mtml	**continue**	**continue**	**stop**	3
mhjk	**continue**	**continue**	**stop**	3
mtChpm	**continue**	**continue**	**stop**	3
dpm (linear)	**continue**	stop	**stop**	2
last week				
m0ml	**continue**	**continue**	**stop**	3
mtml	**continue**	**continue**	**stop**	3
mhjk	**continue**	**continue**	**stop**	3
dpm(linear)	**continue**	stop	**stop**	2
mtChpm	**continue**	**continue**	**stop**	3

Table 14: Ranks for estimators for two test sites.

Estimator	Release 1 Rank	Release 2 Rank	Release 3 Rank	Overall Rank
m0ml	3	3	1	2
mtml	5	3	1	5
mhjk	2	1	1	1
mtChpm	3	3	1	2
dpm(linear)	1	2	5	4

Results show that capture-recapture and curve-fitting methods may be used to estimate the number of components that have defects after release, but no defects in testing. The estimates from several capture-recapture and curve-fitting methods have low relative error and compare favorably with experience-based estimates used as a point of reference. Errors from the best estimators based on capture-recapture and curve-fitting have relative errors between 0.05 and 0.2 compared to the experience-based method that has

absolute relative errors between 0.03 to 0.05. The experience-based method, however, depends greatly on previous releases being similar. The capture-recapture and curve-fitting methods are independent of data from previous releases.

Capture-recapture and curve-fitting estimators that perform best in this study include the m0ml, mtml, dpm(linear) and mhjk. The m0ml, mtml and dpm(linear) estimators tend to underestimate slightly. Due to the partial prescreening of the data to reduce duplicate reporting, these estimators provide estimations that appear to compensate for defect scrubbing. The mhjk tends to overestimate slightly. The mthChao, dpm(exp) and cumulative methods overestimate too much. Because testing too long is preferable to not testing long enough, the mjhk estimator will probably perform the best, especially if there is no prescreening. Estimators for two different capture-recapture models provide a "window". The mtml or the dpm(linear) estimators are good choices for the lower bound of the range and the mjhk estimator is a good choice for the upper bound. These estimates from capture-recapture and curve-fitting estimation methods may be complemented with estimates provided by the experience-based method and the testers' personal experiences.

Estimates can be used as an input to making decisions on whether to stop testing and to release software. Results show that estimates from capture-recapture and curve-fitting methods using several threshold values provide a good basis for a correct decision for stopping. The mhjk estimator appears to perform the best as a basis for decision making. It tends to recommend testing a bit longer and is more conservative than the m0ml, mtml, and dpm(linear) methods. The mthChao, dpm(exp) and cumulative estimators do not perform well in making decisions.

Results show that the same estimators that performed well using data from three test sites also performed well on data from two test sites. In some cases, the estimators performed better. The estimators are also shown to be quite robust for two test sites, even when the sites test the system differently.

This approach is a new application of estimation methods based on capture-recapture and curve-fitting models. These methods may be used to estimate the number of components that have defects in post-release that did not have defects in testing. These are the components that were missed in testing. The estimates in turn may be used to recommend decisions on whether to continue testing or to stop testing and to release software. The recommendation may then contribute as one of several criteria to make a decision. If historical data is available and releases are similar, a simple experience-based method using multiplicative factors should be used as a complement to the capture-recapture and curve-fitting estimation methods. If, however, no historical data is available or releases are dissimilar, capture-recapture and curve-fitting methods work quite well to provide estimates and to help make decisions to continue or to stop testing.

6 Appendix

6.1 Notation for Capture-Recapture Methods

Estimators for the capture-recapture models are discussed in the next section. The notation for these estimators are defined below.

N is the actual number of defects in the inspected object.

n is the number of observed defects in the inspected object.

m is the number of inspectors.

f_k is the number of defects found by exactly k inspectors.

$S = \sum_{i=1}^{m} f_i$.

n_i is the number of defects found by inspector i.

p_i is the probability of detecting a defect by inspector i.

6.2 Estimators for Capture-Recapture Methods

The maximum-likelihood method [14] is based on the assumption that all defects are found by a specific reviewer with equal probability. A simple example for the Mt model that uses only two inspectors follows. The equation is:

$$N = \frac{E(n_1)E(n_2)}{E(f_2)} \tag{3}$$

An estimator for the the number of defects can be derived as:

$$\hat{N} = \frac{n_1 n_2}{f_2} \tag{4}$$

This estimator is known as the Lincoln-Peterson Estimator [5].

The simplest of all models, M0, results from the assumption that defect detection probabilities do not vary by reviewer nor by individual defect. The following equation is maximized for the m0ml estimator [12]:

$$L(N) = \log \left(\begin{array}{c} N \\ n \end{array} \right) + \sum_{j=1}^{m} n_j \log \sum_{j=1}^{m} n_j + (Nm - \sum_{j=1}^{m} n_j) \log(Nm - \sum_{j=1}^{m} n_j) - Nm \log(Nm) \tag{5}$$

This function is maximized numerically over $N \geq n$ to determine $\hat{N}$, the estimate for the number of faults. Subtracting the number of faults found at the review from the estimate of the total number of faults gives the estimate for the number of remaining faults. If most faults are found by two or more reviewers, then few faults are undiscovered. Otherwise, additional reviews are required to find more faults.

The Mt model assumes reviewers have different probabilities in finding defects. For the more general case of the Mt model where there may be two or more reviewers, the following mathematical equation is maximized [12]:

$$L(N) = \log \left(\begin{array}{c} N \\ n \end{array} \right) + \sum_{j=1}^{m} n_j \log n_j + \sum_{j=1}^{m} (N - n_j) \log(N - n_j) - Nm \log N \tag{6}$$

The following equation gives the Chapman estimator for the Mt model [10] (mtChpm) in the case of two reviewers:

$$\hat{N} = \frac{(n_1 + 1)(n_2 + 1)}{(n + 1)} - 1 \tag{7}$$

The jackknife method [14] may also be used to determine the total number of faults. It is based on the assumption that each reviewer has the same probability of finding a specific defect, while the defects are found with different probabilities.

The jackknife estimator (mhjk) for the Mh model [6] estimates N_j. N_j is chosen as some N_{ji} as described below. Table 15 shows the formulas from [6] for $\hat{N}_{jk}$ for order $k \leq 5$, where $k \leq m$ is the jackknife order.

According to Burnham and Overton [6], in looking at the mean squared error of $\hat{N}_{jk}$, the unique minimum is usually achieved at $k = 1, 2$, or 3.

$\hat{N}_j$ is chosen by testing sequentially the hypotheses

$$H_{0i} : E(\hat{N}_{j,i+1} - \hat{N}_{ji}) = 0 \tag{8}$$

versus

$$H_{\alpha i} : E(\hat{N}_{j,i+1} - \hat{N}_{ji}) \neq 0, \ for \ i \leq 4 \tag{9}$$

Table 15: Jackknife estimators $\hat{N}_{hk}$ for k = 1, ... ,5.

$$\hat{N}_{h0} = S$$

$$\hat{N}_{h1} = S + \left(\tfrac{m-1}{m}\right) f_1$$

$$\hat{N}_{h2} = S + \left(\tfrac{2m-3}{m}\right) f_1 - \tfrac{(m-2)^2}{m(m-1)} f_2$$

$$\hat{N}_{h3} = S + \left(\tfrac{3m-6}{m}\right) f_1 - \left(\tfrac{3m^2-15m+19}{m(m-1)}\right) f_2 + \tfrac{(m-3)^3}{m(m-1)(m-2)} f_3$$

$$\hat{N}_{h4} = S + \left(\tfrac{4m-10}{m}\right) f_1 - \left(\tfrac{6m^2-36m+55}{m(m-1)}\right) f_2 + \left(\tfrac{4m^3-42m^2+148m-175}{m(m-1)(m-2)}\right) f_3 - \tfrac{(m-4)^4}{m(m-1)(m-2)(m-3)} f_4$$

$$\hat{N}_{h5} = S + \left(\tfrac{5m-15}{m}\right) f_1 - \left(\tfrac{10m^2-370m+125}{m(m-1)}\right) f_2 + \left(\tfrac{10m^3-120m^2+485m-660}{m(m-1)(m-2)}\right) f_3 - \left(\tfrac{(m-4)^5-(m-5)^5}{m(m-1)(m-2)(m-3)}\right) f_4 + \tfrac{(m-5)^5}{m(m-1)(m-2)(m-3)(m-4)} f_5$$

Choose $\hat{N}_j = \hat{N}_{ji}$ such that H_{0i} is the first null hypothesis not rejected. The test statistic is

$$T_i = \frac{\hat{N}_{j,i+1} - \hat{N}_{ji}}{(\frac{S}{S-1}(\sum_{i=1}^{m} a_i^2 f_i - (\hat{N}_{j,i+1} - \hat{N}_{ji})^2/S))^{1/2}} \tag{10}$$

where the coefficients for f_i in the formulas for N_{ji} in Table tab:jke are the constants $a_{i1}, \cdots, a_{ik}$.

It is expected that the significance levels, P_i, will be increasing. If P_{i-1} is small, for example $P_{i-1} \leq 0.05$, and P_i is much larger than 0.05, then choose $\hat{N}_j = \hat{N}_{ji}$.

The Mth model assumes that the defect detection probabilities vary by reviewer and by individual defect. An estimator used for this model is the MthChao [7]. The mathematical formula for the MthChao estimator is:

$$\hat{N}_i = \frac{n}{\hat{C}_i} + \frac{f_1}{\hat{C}_i}\hat{\bar{\gamma}}_i^2, \quad i = 1, 2, 3 \tag{11}$$

where

$\hat{C}_1 = 1 - f_1/\sum_{k=1}^{m} k f_k$, $\hat{C}_1$ is an estimator of the expected sample coverage, where sample coverage is defined as the total individual detection probabilities of the found defects.

$\hat{C}_2$ and $\hat{C}_3$ are bias-corrected versions of $\hat{C}_1$ and defined as:

$$\hat{C}_2 = 1 - \frac{f_1 - 2f_2/(m-1)}{\sum_{k=1}^{m} k f_k}$$

$$\hat{C}_3 = 1 - \frac{f_1 - 2f_2/(m-1) + 6f_3/(m-1)(m-2)}{\sum_{k=1}^{m} k f_k}$$

$\hat{\hat{\gamma}}_i^2$ is the estimate of the square of the coefficient of variation and is defined as:

$$\hat{\hat{\gamma}}_i^2 = max\{\frac{n}{\hat{C}_i}\sum_{k=1}^{m} k(k-1)f_k/(2\sum_{j<}\sum_{k} f_j f_k) - 1, 0\},\ i = 1, 2, 3.$$

References

[1] Basili, V., B. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM*, vol. 27, no. 1, (January 1984), pp. 42 – 52.

[2] Biyani, S., P. Santhanam, "Exploring Defect Data from Development and Customer Usage on Software Modules over Multiple Releases," *Procs. of the 9th International Symposium on Software Reliability Engineering*, (November 1998), Paderborn, Germany, pp. 316 – 320.

[3] Briand, L., K. El Emam, B. Freimut, O. Laitenberger, "Quantitative Evaluation of Capture-Recapture Models to Control Software Inspections," *Procs. of the 8th International Symposium on Software Reliability Engineering*, (November 1997), Albuquerque, New Mexico, USA, pp. 234 – 244.

[4] Briand, L., K. El Emam, B. Freimut, O. Laitenberger, "A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content," *IEEE Transactions on Software Engineering*, vol. 26, no. 6, (June 2000), pp. 234 – 244.

[5] Briand, L., K. El Emam, B. Freimut, "A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method," *Procs. of the 9th International Symposium on Software Reliability Engineering*, (November 1998), Paderborn, Germany, pp. 32 – 41.

[6] K. Burnham, W. Overton, "Estimation of the Size of a Closed Population when Capture Probabilities vary Among Animals," *Biometrika*, vol. 65, no. 3, (December 1978), pp. 625 – 633.

[7] Chao, A., "Estimating Population Size for Capture-Recapture Data when Capture Probabilities Vary by Time and Individual Animal," *Biometrics*, vol. 48, (March 1992), pp. 201 – 216.

[8] Christenson, D. S. Huang, "Estimating the Fault Content of Software using the Fix-on-Fix Model," *Bell Labs Technical Journal*, vol. 1, no. 1, (Summer 1996), pp. 130 – 137.

[9] Eick, S., C. Loader, M. Long, L. Votta, S. Vander Wiel, "Estimating Software Fault Content Before Coding," *Procs. of the 14th International Conference on Software Engineering*, (1992), pp. 59 – 65.

[10] El Emam, K., O. Laitenberger, "Evaluating Capture-Recapture Models with Two Inspectors," Fraunhofer Institute for Experimental Software Engineering, Germany, 1999, ISERN-99-08, pp. 1 – 50.

[11] Graves, T., A. Karr, J. Marron, H. Siy, "Predicting Fault Incidence using Software Change History," *IEEE Transactions on Software Engineering*, Vol. 26, No. 7(July 2000), pp. 653–661.

[12] Otis, D., K. Burnham, G. White, D. Anderson, "Statistical Inference from Capture Data on Closed Animal Populations," *Wildlife Monographs*, no. 62, (October 1978), 135 pp.

[13] Thelin, T., P. Runeson, "Capture-Recapture Estimations for Perspective-Based Reading - A Simulated Experiment," *Procs. of the Conference on Product Focused Software Process Improvement*, (June 1999), Oulu, Finland, pp. 182 – 200.

[14] Vander Wiel, S., L. Votta, "Assessing Software Designs using Capture-Recapture Methods," *IEEE Transactions on Software Engineering*, vol. 19, no. 11, (November 1993), pp. 1045 – 1054.

[15] Wohlin, C., P. Runeson, "An Experimental Evaluation of Capture-Recapture in Software Inspections," *Journal of Software Testing, Verification and Reliability*, vol. 5, no. 4, (1995), pp. 213 – 232.

[16] Wohlin, C., P. Runeson, "Defect Content Estimations from Review Data," *Procs. of the 20th International Conference on Software Engineering*, (April 1998), Kyoto, Japan, pp. 400 – 409.

[17] Yu, T., V. Shen, H. Dunsmore, "An Analysis of Several Software Defect Models," *IEEE Trans. on Software Engineering*, vol. 14, no. 9, (September 1988), pp. 1261 – 1270.