Estimating Software Fault Content Before Coding

Stephen G. Eick‡, Clive R. Loader†, M. David Long‡, Lawrence G. Votta‡, Scott Vander Wiel†

AT&T Bell Laboratories

600 Mountain Avenue
Murray Hill, New Jersey 07974†

1000 East Warrenville Road
Naperville, Illinois 60566‡

ABSTRACT

The standard software development process consists of multiple stages: requirements, design, coding, system test, first office, and finally delivery. An objective of this process is to minimize the number of faults in delivered code. Root cause analysis shows that many of the faults can be traced back to requirements or design faults. As part of the software development process, reviews are conducted to remove these faults before the requirements or design document is passed on to the next step. We have developed a method of instrumenting a review process to record document faults discovered by reviewers during their preparation. Then, using statistical techniques related to capture-recapture methods, we estimate the number of undiscovered faults remaining in the document. The key idea to our method is to look at how many common faults independent reviewers find and then extrapolate to the total number of faults. We do not seed the document with artificial faults - no additional faults are introduced.

We have applied our methods to 13 review sessions (either feature requirements or feature design) and are in the process of a longitudinal study tracing these features. Our results to date estimate that about 20% of the faults are undetected by reviews. When the predicted number of undetected faults is greater than 20%, consideration should be given to reworking design and/or rereviewing the result. One surprising by-product of this study is a quantification of the number of faults found by group reviews. We find that only about 10% of the (discovered) document faults are found at the review (90% are found in preparation) and that the lead time to schedule a review is about ten working days.

1. INTRODUCTION

An objective of all software projects is to minimize the number of faults in delivered code. As some recent, well publicized, failures demonstrate [1] eliminating faults from software is a very important problem. One measure of software quality is the number of residual faults.

The standard software development process consists of multiple steps, finally resulting in a delivered system. As practiced at AT&T, these steps consist of a feature definition, software design, coding, testing, and finally delivery. At the feature definition stage systems engineers formulate and define new software features and produce a requirements document. Then developers, working with the systems engineers, produce a software design document describing the software architecture and changes needed to implement the features. Next coders implement the required changes. The new, enhanced software is then submitted for feature and system reliability testing. Finally, after testing, a beta version code is delivered to the first customers.

At each stage in the process a key objective is to minimize the number of faults[1] passed on to the next stage. For later stages, systems test and first customer application, after the code has been developed, there are well-known statistical techniques to estimate the remaining software fault density [2] and other techniques to determine the optimal testing strategy [3], [4]. At the earlier stages—feature definition, and software design—the code has not yet been written so there is nothing yet to test. To ensure that the requirements and design are of sufficient quality, a review is conducted.

Reviews consist of 4 to 12 engineers who in preparation identify parts of the document that may contain design faults. The author corrects these faults and the document is passed on to the next stage unless the faults are so severe a second review is needed. The process is designed to ensure that all possible faults are caught and eliminated as early as possible.

Unfortunately, as the previously cited examples show, occasionally faults pass through the process and end up as faults in delivered software. As depicted in Figure 1, in a typical software process the design phase accounts for a significant net addition of faults in the development process (see [5] p. 353).[2]

1. We use the IEEE convention of error, fault, and failure from the frame of reference of the process step. Since we are talking about the actual defect in the requirements or design, we consistently choose to use the term requirements or design fault, unless the context is unambiguous in which case we will use fault.
2. In actual practice, the situation can be be much more complicated than portrayed by this simple accounting model. From the frame of reference of the CODE process, one software error (a requirement fault in the frame of reference of the REQS process), can actually cause many software faults.

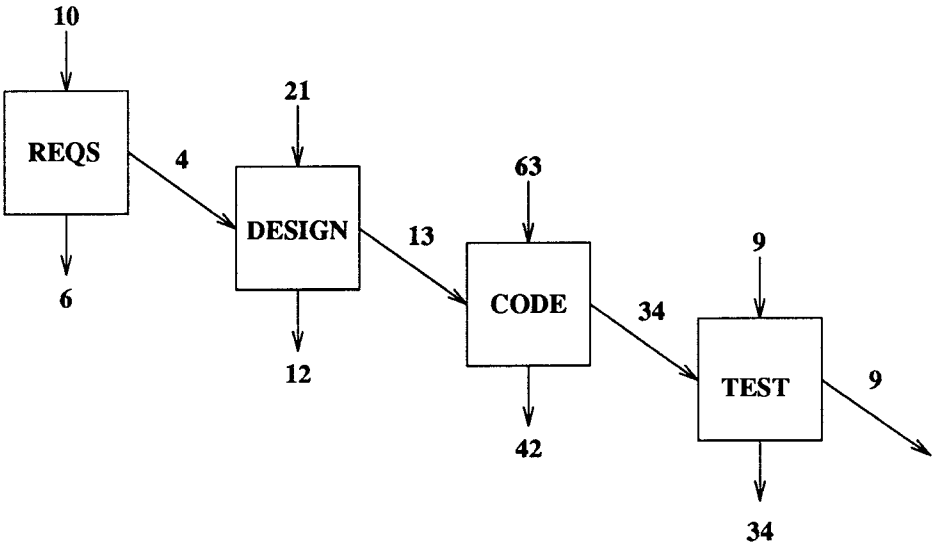

Figure 1: TYPICAL SOFTWARE DEVELOPMENT PROCESS
Vertical arrows going into box represent faults/size measure injected at that step. Vertical arrows out represent faults/size removed at that step. Arrows pointing down and to the right represent faults/size being propagated from one step to the next. Thus for the **REQS**, 10 faults/KLOC are generated, 6 faults/KLOC are removed, and 4 faults/KLOC are passed on to **DESIGN**.

Root cause analyses are performed to identify in which stages of the process the faults originated and understand how to minimize them in the future. One conclusion from these studies is that some of the most insidious and expensive to repair faults have root causes in the software design stage.

In this paper we describe a novel new technique to estimate the number of residual faults at each stage in the software development process. Our technique uses a class of statistical methods known as capture-recapture [6]. This technique is most commonly used in the estimation of wildlife populations. Other researchers (see [2], p. 114 for discussion) have applied capture-recapture methods to study software reliability but in a very different context. Their approach was to seed software with known faults, and then at system test they could use the number of undetected seeded faults to estimate the number of remaining faults. They found their estimates to be unreliable, because it is difficult to seed the software with faults similar to those naturally occurring.

Our approach is different in that we introduce no artificial faults into design documents. Instead, prior to each review meeting, we assume that each reviewer has independently[3] searched for faults in the design document. If most of the faults are found by two or more reviewers, this intuitively suggests that there are few undiscovered faults that could have been found by additional reviewers. Conversely, if each reviewer identifies disjoint sets of faults, this suggests additional reviewers would find yet more faults. We use capture-recapture techniques to translate these intuitive results into statistical estimates. This behavior is portrayed by the simple two reviewer estimate given by equation (8) in Section 3.

To test our methods we have started a longitudinal study of new software features on a large (several million lines of code) real-time telecommunications system. We have instrumented the review process and collected data from 13 reviews for 9 different features. 4 of the reviews are for requirements documents and 9 are for design documents. The 5 design document reviews had 9 different review sessions, because 2 features were large enough to require multiple sessions. Table 1 contains a brief summary describing these reviews.

We intend to monitor the development of these features to relate our pre-code residual fault estimates with those actually measured in subsequent process steps (standard practice is to root cause every software fault back to process introduced and reason(s) for the lack of detection earlier).

In the remainder of this paper we describe the instrumentation of the development process (Section 2), the mathematical formulation of the maximum likelihood estimate for the total number of faults (Section 3), and initial data analysis (Section 4). We conclude by observing that the number of faults predicted using capture-recapture methods is consistent with developer and tester intuition (Section 5). More definitive conclusions await the completion of the full longitudinal study.[4]

3. We are also studying methods that allow dependence between reviewers. This would be the case, for example, if reviewers collaborated or specialized in particular areas.

4. Unfortunately it is outside the scope of this paper to discuss our validation plans in detail. However, briefly, our plan for this work has three phases. Phase 1 establishes the ability to instrument and measure the faults discovered by reviewers economically and with as little process interference as possible. Further, phase 1 applies some of the simple capture-recapture models to review data and assesses whether results look promising and estimators behave reasonably. This paper summarizes the results of phase 1. Phase 2 and phase 3 are both currently in progress. Phase 2 is a set of experiments where two reviews are performed with the same requirements or design, with/without author repair and with the same/different teams. The intention is to understand the repeatability and reproducibility of the measurement technique. Further, theoretical work will be done on capture-recapture models allowing certain forms of relaxation of the independence and constant probability assumptions. Phase 3 is the longitudinal study where we follow the features through their

Feature Tag	Review Type	Feature Type	Number of Sessions	Number of Reviewers	Number of Authors
FT1	Requirement	New	1	8	1
FT2	Requirement	New	1	9	1
FT3	Requirement	New	1	6	2
FT4	Requirement	New	1	10	2
FT5	Design	Enhanced	1	6	1
FT6	Design	Enhanced	1	5	1
FT7	Design	New	1	8	1
FT8	Design	New	4	8	4
FT9	Design	New	2	6	1

Table 1: Summary Of Document Reviews In Study

2. INSTRUMENTING THE REVIEW PROCESS

Before a formal document review meeting, in the preparation phase, each reviewer individually reads the document noting faults that he or she believes should be resolved before the document is approved and the feature developed further. At the meeting a moderator, who is not the author, "walks" the group through the document and the secretary records all identified faults. At the end of the review there is a complete list of all faults and a record of who found each fault in preparation.[5] There are two kinds of faults: those discovered in preparation and those discovered at the meeting. The second type of fault involves meeting synergy, that is multiple people working together may discover problems that each individually missed. If the document or code is of sufficiently high quality as decided by the group, it is passed on to the next state after the author addresses each fault. There are no further reviews and the author's resolution of each fault is recorded.

Typically there are between 4 and 12 reviewers, some of whom are critical reviewers representing specific areas of expertise. The meeting cannot take place without these critical reviewers. An author must schedule the review meeting in advance. If the document is completed early, the author will usually wait for the previously scheduled review, since it is hard to arrange an earlier review date. If the author completes late, the review must be rescheduled—often weeks later due to the scheduling window. These effects add time to the development schedule.

The output from the design document review for feature FT7 is shown in Table 2.[6] For this review there were 8 reviewers who discovered 24 faults. The margin totals are on the right and the bottom. Fault 5, for example, was discovered, in preparation, by reviewers 1, 4 and 8. Faults 15, 19, 20, 21, 23, and 24 were discovered at the meeting. Reviewer 8 found 10 faults where as reviewer 1 found 3 faults and reviewers 2 and 3 found none. Of the 24 recorded faults, 5 (21%) were found by more than one reviewer.

In the review methodology we intentionally do not record the names of individual reviewers. The reason is that we do not want the results used for performance evaluations, since that might bias the process. For example, reviewers 2, 3, 5, and 6 found fewer faults than did the other reviewers. Possible explanations for this involve training, experience, or that these reviewers may specialize in particular areas for which there were no faults to find.

The faults found in requirements or design documents may not all result in code faults. There have been few historical studies trying to relate document faults to eventual code faults. Establishing the exact nature of this relationship is difficult. Our methodology is to ask each reviewer to indicate whether or not he or she thinks the document fault will result in an eventual code fault. The document faults indicated by an '*' in Table 2 are those that the reviewer thought would result in code faults. Reviewers 4 and 8 think that most of the document faults will result in code faults.

Table 3 shows how the author resolved each fault. For each fault the author records the scope of the changes needed to resolve the fault, how long it took, and a non-exclusive categorization of the kind and type of fault. The possible time resolution values are <1 hour and >=1 hour and the scopes are 'local fix' (FL), 'global fix' (FG), and 'no change' (NC). Fault 5, for example, required more than 1 hour to fix and the changes are global. The classification of faults into kinds and types categories follow schemes discussed in [7] and [8].[7]

3. USING CAPTURE-RECAPTURE METHODS TO ESTIMATE THE NUMBER OF RESIDUAL FAULTS

Capture-recapture models for ecology are described in [9], [10]. The basic model is as follows. There are some unknown number of faults N in a document being reviewed by m reviewers working independently. For the jth reviewer we let n_j be the number of faults he or she identified in preparation, and p_j be the constant (unknown) probability that reviewer j discovers a given fault. All faults are assumed to be equally likely to be discovered but individual reviewers have different skill levels as modeled by the p_j.[8]

entire life cycle and observe whether the measurements made by capture-recapture agree with root cause analysis of all software faults after several years of the feature's exposure to the field.

5. We performed many *earlier* experiments to determine the *best* way to record the individual reviewer data. Our current method adds <10% more time to the recording of each fault as measured by observers with stop watches. Plans exist to improve this further.

6. We use FT7 as an illustrative example to describe information recorded for all reviews listed in TABLE 1.

7. The importance of this data may not be apparent to the reader. However, for testing and relaxing the model assumptions presented in section 3 it could be crucial, since an individual's ability to find faults may very well depend on type and kind of requirements or design fault present. These results will be reported in a later paper.

8. Work in progress includes relaxing the reviewer independence and constant probability for a fault assumptions. The work develops in a similar fashion to the ecological population models which are outlined in [6].

Fault	Revwr 1	Revwr 2	Revwr 3	Revwr 4	Revwr 5	Revwr 6	Revwr 7	Revwr 8	Total
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	1
3	0	0	0	0*	0	0	1	0	1
4	0	0	0	0	0	0	0	1	1
5	1	0*	0	1*	0*	0	0	1*	3
6	0	0	0	0	0	0	0	1	1
7	0*	0*	0*	1*	1*	0*	0*	0*	2
8	1	0	0*	0*	0	0	0	0*	1
9	0	0	0	1	0	0	0	0	1
10	1	0	0	1*	0	0	0	0	2
11	0	0	0	1	0	0	0	0	1
12	0	0	0	0*	0	0	0	1	1
13	0*	0*	0	1*	0*	1	0*	0*	2
14	0	0	0	1*	0	0	0	1*	2
15^	0	0*	0	0	0	0	0	0	0
16	0*	0*	0*	0*	0	0	0	1*	1
17	0*	0*	0*	0*	0*	0	0*	1*	1
18	0*	0*	0*	0*	0	0	0*	1*	1
19^	0*	0*	0*	0*	0*	0	0*	0*	0
20^	0	0*	0*	0*	0*	0	0*	0*	0
21^	0	0	0*	0*	0	0	0*	0*	0
22	0*	0	0*	0*	0	0	0	1*	1
23^	0*	0*	0	0	0	0	0	0*	0
24^	0*	0	0*	0*	0	0	0	0*	0
Total:	3	0	0	7	1	1	2	10	

^faults discovered at the review meeting.
*will result in an eventual code fault

Table 2: Design Review Faults for Feature FT7

Fault	1	2	3	4	5	6	7	8	9	10	11	12
Time	<1	<1	<1	<1	≥1	<1	≥1	<1	<1	<1	<1	<1
Scope	FL	FL	FL	FL	FG	FL	FG	FL	FL	FL	FL	FL

Fault	13	14	15	16	17	18	19	20	21	22	23	24
Time	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1
Scope	FL	FL	FL	FL	FL	NC	FL	FL	FL	FG	FL	FL

Table 3: Author Resolution for Feature FT7

The observed data is a number n of faults and an $n \times m$ matrix X with the $(i,j)^{th}$ element $X_{ij}=0$ or 1, according to whether the jth reviewer observed the ith fault (see Table 2). The likelihood is then the probability of the observed data occurring, as a function of the unknown parameters. This probability can be expressed as

$$P(n,X) = P(n)P(X|n) \tag{1}$$

Let us first consider the conditional likelihood $P(X|n)$. Since we assume faults are independent, we have

$$\begin{aligned} P(X|n) &= \prod_{i=1}^{n} P(X_{i1}, \cdots, X_{im} | \sum_{j=1}^{m} X_{ij} \geq 1) \\ &= \prod_{i=1}^{n} \frac{\prod_{j=1}^{m} p_j^{X_{ij}} 1-p_j^{1-X_{ij}}}{1-Q} \\ &= \frac{\prod_{j=1}^{m} p_j^{n_j} 1-p_j^{n-n_j}}{(1-Q)^n}. \end{aligned} \tag{2}$$

Here, $Q = \prod_{j=1}^{m} (1-p_j)$ is the probability that a fault is not observed. The number of observed faults, n, has a Binomial $(N, 1-Q)$ distribution, so

$$P(n) = \binom{N}{n} (1-Q)^n Q^{N-n}. \tag{3}$$

Substituting (2) and (3) into (1) gives the likelihood function

$$L(n,p_1,\ldots,p_m) = P(X,n) = \binom{N}{n} \prod_{j=1}^{m} p_j^{n_j} (1-p_j)^{N-n_j}. \tag{4}$$

An intuitively appealing, but wrong, method of constructing a likelihood is as follows. Since we have assumed reviewer independence, the n_j are independent Binomial (N,p_j) random variables and a likelihood can be constructed as the product of these binomial probabilities. The mistake here is that the n_j are not sufficient statistics for our model, and hence we have discarded some information in the data.

Maximum likelihood estimates of N and $p_1, \ldots, p_m$ are found by maximizing (4). Differentiating (4) with respect to p_j yields

$$\hat{p}_j = \frac{n_j}{N}. \tag{5}$$

Substituting (5) into (4) and taking logs gives

$$L_c(N) = \log \binom{N}{n} + \sum_{j=1}^{m} n_j \log n_j - Nm\log N + \sum_{j=1}^{m} (N-n_j)\log(N-n_j) \quad (6)$$

This can be maximized numerically over $N \geq n$.

An alternative estimator is based on maximizing the conditional likelihood $P(X|n)$. This leads to a nonlinear equation to solve for $\hat{N}$:

$$\left[1 - \frac{n}{\hat{N}}\right] = \prod_{j=1}^{m} \left[1 - \frac{n_j}{\hat{N}}\right]. \quad (7)$$

While the estimate based on the full likelihood is preferable, the conditional equation (7) does have natural appeal since each side can be interpreted as an estimate of Q. In practice, the estimates using either (6) or (7) are usually very similar.

The distribution of these estimates has been studied in [11]. In particular, Darroch showed that $N^{-1/2}(\hat{N}-N)$ has an asymptotic normal distribution. However, we expect this result to be heavily dependent on model assumptions so we would not place too much faith in confidence regions for N derived using this distribution.

For the case $m=2$, observing that $n = n_1 + n_2 - n_{12}$ where n_{12} is the number of faults that both reviewers 1 and 2 discovered, equation (7) becomes

$$\hat{N} = \frac{n_1 n_2}{n_{12}} \quad (8)$$

which is known to wildlife ecologists as the Lincoln-Peterson estimator [12], [13]. Equation (8) is discussed both in the introduction and in the discussion sections.

4. DATA ANALYSIS AND PRELIMINARY RESULTS

This section describes our data analysis methods and our results to date after analyzing 13 review sessions. To illustrate analysis techniques for individual reviews we use the experimental results from the review of feature FT7. Recall that our primary objective is to estimate the number of undiscovered faults in the document. If there are too many, the document may require a revision, a second review, or both before passing it on to the next phase. Other interesting questions involve how long it takes the author to resolve them and the effect of group reviewing.

Two important assumptions behind the likelihood equation (4) are that the reviewers are statistically independent and faults are probabilistically identical.[9] In practice these assumptions may be inaccurate because of *specialization*, *collusion*, and group *synergy*. In a review, reviewers specializing in different areas are intentionally chosen to ensure nothing is overlooked. Since each reviewer may focus on the parts of the document relating to his or her area of expertise, specialization may result in fewer common faults than would otherwise be expected. Collusion occurs when multiple reviewers work together and results in results in more common faults than would be expected. Finally, due to group synergy, some faults are discovered at the meeting. Group synergy is a measure of the effect of having multiple people simultaneously working on the same problem.

9 Two reviewers may review independently but find no faults in common because they focused on different parts of the document. In this case faults in different parts of the document are probabilistically different.

4.1 Fault Identification Analysis

Recall that in our sample review, FT7, there were a total of 24 faults found. The left and center panels of Figure 2 are histogram summaries of the marginal totals from Table 2. Of the 24 total faults 13 were found by one reviewer, 4 were found by a second, and 1 were found by a third. At the review 6 faults were discovered, that is no reviewer observed the fault in preparation. The right panel of Figure 2 is a plot of the log-likelihood (6) based on the overlap of reviewers finding the 18 faults discovered in preparation[10] for the review meeting. The likelihood function is peaked at about 28 suggesting that only about 4 faults (15%) remain undiscovered. The actual maximum is slightly larger that 28, but 28 is the maximizer over all integers. These estimates have high variance but are more precise than any other method known to the authors. We discussed our results with the design process owner and the reviewers. Based on experience, their intuition also suggested that there were few remaining faults.

Across all 13 reviews our estimates of the number of undetected faults ranged from 0 to 61 (0 to 60%) as shown in Figure 3. The number of remaining faults is the difference between the number predicted, 'P', and the total number found 'S'. For most of the documents we predict between 5 and 10 undiscovered faults (10 to 30%).

4.2 Review Meeting Synergy

Scheduling a review meeting is a difficult task. The meetings are often postponed because it is hard to find a two hour time slot when 8 or more people are available. We observed reviews being postponed several times over several weeks because of schedule conflicts. The average time from the completion of reviewer preparation until the review meeting could be scheduled has been about 10 working days. Features pass through multiple reviews—design, coding, testing, etc. Each one of these reviews potentially adds schedule delay into the process.

An alternative to the present documentation review meetings would be for each reviewer to individually give his or her faults to the author. This would eliminate the need to find a meeting time convenient to all reviewers. One measure of the value of conducting a common review meeting is the number of additional faults that are discovered at the review. These additional faults are due to reviewers taking another look at the document as well as group synergy. However, many factors other than group synergy contribute to the value of group reviews. For example, without review meetings reviewers may prepare less thoroughly.

We find a 5% to 50% further reduction in the number of undiscovered faults resulting from document review meetings, depending on the document quality. For most of the reviews the reduction is less than 10%. This is shown, for example, in Figure 3 where for most reviews the 'S' symbol is near the 'F'. In cases where many faults were found during the review meeting we investigated further and inevitability found an explanation. For example, for review FT7, 6 faults (25% of the total number found) appear to have been discovered during the review. However, 5 of these were found by the author (not a reviewer) prior to the review and therefore cannot be directly attributed to the group review.[11]

10. In our calculation of the likelihood function we ignore faults discovered at the meeting. This number is usually small, less than 10%.

11. This is an instrumentation problem that we had not observed in our trials for designing and testing the instrumentation. We have resolved this problem by

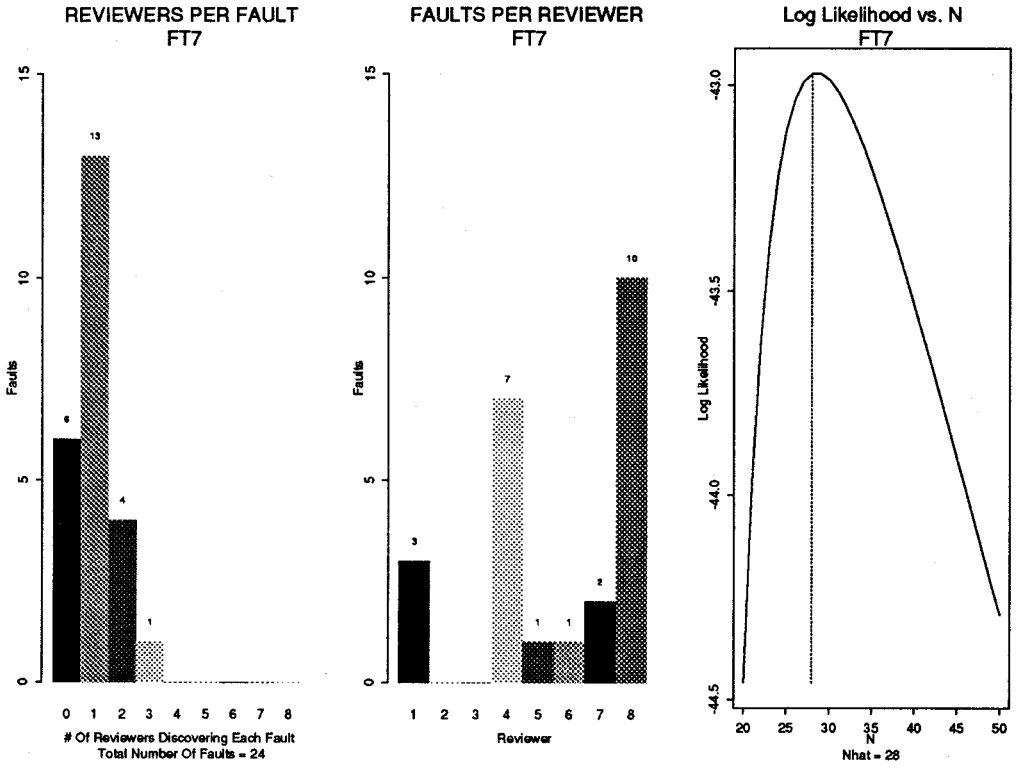

Figure 2: Different Fault-Reviewer Marginals and Likelihood Plot for FT7

The histogram on the left shows how many reviewers observed each fault. The middle histogram shows the number of faults that each reviewer observed. The plot on the right is the plot of the concentrated log-likelihood defined in equation (6) and the vertical line tags the maximum likelihood estimate for the total number of design document faults.

5. DISCUSSION

This paper describes a new technique to estimate the number of faults remaining in design documents after formal review meetings. The idea is to apply capture-recapture statistical techniques to the data collected at a document review meeting. Based on the amount of overlap in faults found by various reviewers we estimate the number of remaining faults. If the estimate is large the document is a candidate for reworking.

In software engineering it is important to detect and correct design faults as early as possible, particularly before coding. The reason is that if the design is changed after coding, say at the system test stage, large sections of the code may have to be rewritten. Our approach, in contrast to other software reliability engineering techniques [1], yields results and fault predictions at the design stage before code is written.

A current existing approach to measure document and design quality is to compare the total number of faults found at a review with historical norms. Previously a document would be reviewed a second time if the number of faults found was significantly different from the historical average. Too many faults were taken as evidence of a poor document, while too few were taken as evidence of a poor review. This approach implicitly assumes that variations among reviews are larger than variations among documents. If this is not the case then two problems arise. First, high quality documents must be re-reviewed, thus discouraging zero defect behavior. Second, low quality documents move easily on to the coding stage. We are hopeful that the capture-recapture approach will eliminate both of these problems.

Group reviews are valuable for motivation, training, discussion, networking, etc, but, surprisingly, most faults are found in preparation and not at the reviews. In the reviews we have monitored so far, about 10% of the faults are discovered at the review meeting, the other 90% are discovered in preparation. Our original impression was that a large percentage of the faults were discovered at the reviews. In the reviews we observed it took an average of 10 working days to schedule the meeting after all reviewers had completed their preparation.

The internal AT&T development community has been responsive to applying our methods to ongoing projects. One particular project applied the technique to a design document and found that our method predicted a large number of residual faults. The reviewers decided that even though the document met the historical conditions to be passed on to the next stage that it needed more work and another reviewed. They discussed

recording the authors' discovered faults as we do the reviewers in Table 2. Inspection of Figure 3 displays large group synergy components for FT12 and FT13. The reason is the same as for FT7.

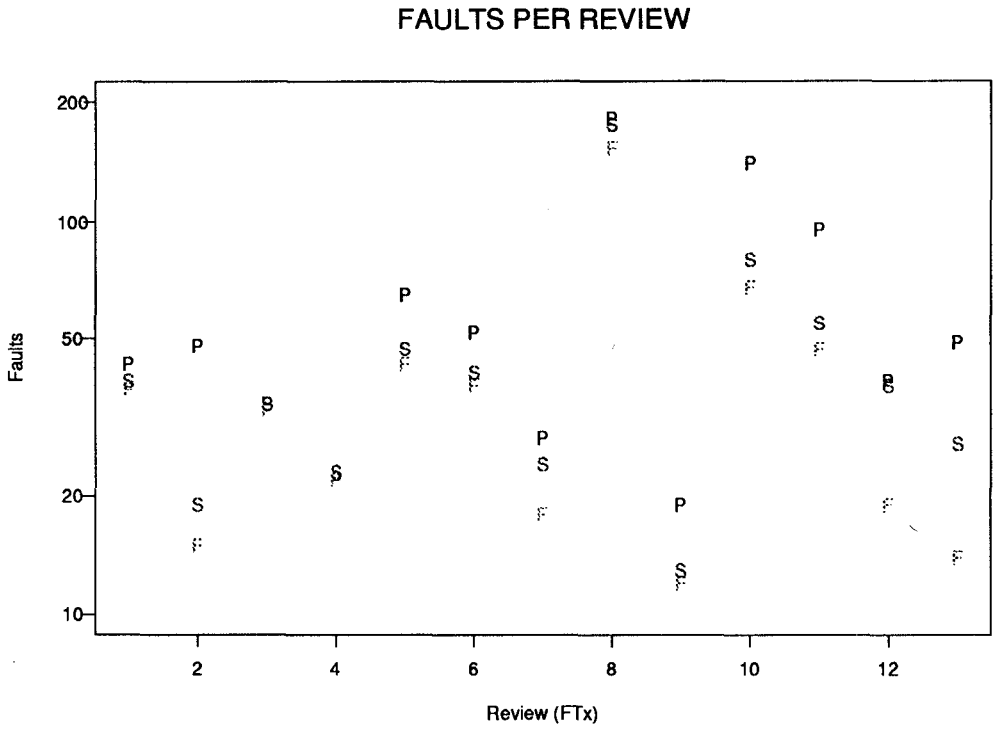

Figure 3: Predicted and Discovered Number of Faults"
For each of the 13 reviews we show the predicted number of faults 'P', number found during preparation 'F', and total number found including those discovered at the meeting 'S'. We display the result in log scale. The review we have been discussing is FT7.

this with project management and the document was reworked.

References

1. John Schneidawind, "Software flaws take costly toll," *USA Today*, August 29, 1991.
2. John D. Musa, Anthony Iannino, and Kazuhira Okumoto, *Software Reliability Measurement, Prediction, Application*. New York, New York: McGraw-Hill Book Company, 1987.
3. Y. Levendel, "Improving Quality With A Manufacturing Process," *IEEE Software*, pp. 13-25, March 1991.
4. Siddharta R. Dalal and Collin L. Mallows, "When Should One Stop Testing Software?" *Journal of the American Statistical Association*, vol. 83, no. 403, pp. 872-879, September 1988.
5. Watts S. Humphery, *Managing the Software Process*, Reading, Massachusetts: Addison-Wesley Publishing Co., 1989.
6. Kenneth H. Pollock, "Modeling Capture, Recapture, and Removal Statistics for Estimation of Demographic Parameters for Fish and Wildlife Populations: Past, Present, and Future," *Journal of the American Statistical Association*, vol. 86, no. 413, pp. 225-238, March 1991.
7. Victor R. Basili and Barry T. Perricone, "Software Errors and Complexity: an Empirical Investigation," *Communications of the ACM*, vol. 27, no. 1, pp. 42-52, January 1984.
8. Dewayne E. Perry and Michael Evangelist, "An empirical Study of Software Interface Faults—An Update," *Proceedings of the Twentieth Annual Hawaii International Conference On System Sciences*, January 1987, Volume II, 113-126.
9. David T. Otis, Kenneth P. Burnham, Gary C. White, and David R. Anderson, "Statistical Inference for Capture Data On Closed Animal Populations," *Wildlife Monographs*, no. 62, pp. 1-135, October 1978.
10. Gary C. White, David R. Anderson, Kenneth P. Burnham, and David T. Otis, "Capture-Recapture and Removal Methods for Sampling Closed Populations," Los Alamos National Laboratory, LA 8787-NERP, Los Alamos, NM, 1982.
11. J. N. Darroch, "The Multiple-Recapture Census 1: Estimation of a closed population," *Biometrika*, vol. 45, pp. 343-359, 1958.
12. E. D. Le Cren, "A note on the history of mark-recapture population estimates," *Journal of Animal Ecology*, vol. 34, pp. 453-454, 1965.
13. F. C. Lincoln, "Calculating waterfowl abundance on the basis of banding returns," U.S. Department of Agriculture Circular 118, 1930.