# Estimating the Number of Faults Remaining in Software Code Documents Inspected with Iterative Code Reviews

**A. Harel**
*Computer Science Dept., Technion*

assafh@cs.technion.ac.il

**Prof. E. Kantorowitz**
*Computer Science Dept., Technion*
*and*
*Software Engineering Dept.,*
*Ort Braude College of Engineering*
kantor@cs.technion.ac.il

## Abstract

*Code review is considered an efficient method for detecting faults in a software code document. The number of faults not detected by the review should be small. Current methods for estimating this number assume reviews with several inspectors, but there are many cases where it is practical to employ only two inspectors. Sufficiently accurate estimates may be obtained by two inspectors employing an iterative code review (ICR) process. This paper introduces a new estimator for the number of undetected faults in an ICR process, so the process may be stopped when a satisfactory result is estimated. This technique employs the Kantorowitz estimator for N-fold inspections, where the N teams are replaced by N reviews. The estimator was tested for three years in an industrial project, where it produced satisfactory results. More experiments are needed in order to fully evaluate the approach.*

## 1. Introduction

### 1.1. Code review as a form of inspection

The software inspection technique introduced by Fagan [1] is considered to be the most efficient method for eliminating software faults [2]. It is used extensively in all phases of a software-system life cycle [3]. Classical Fagan type inspection defines a number of roles for the participants of a software inspection meeting: moderator, organizer, inspector, author and recorder. Although the same person may assume multiple roles, the classic inspection meeting is usually attended by 3–6 participants [4]. Employing such large teams is quite expensive. Therefore, over the years, improved inspection techniques were introduced [5] [6] [7] [8].

Some Code reviews discussed in [9] involve only 2 participants. The first participant is the code writer who also acts as the coordinator, moderator, recorder, and inspector. The second participant acts only as an inspector [10]. The formal reading technique is usually ad-hoc, meaning the inspectors are not required to read the code document before the inspection meeting [11].

Code review may be an iterative process. Faults that are detected during a review are corrected, and then another review is scheduled. This process is repeated until all faults are believed to be eliminated [12]. The product of the code review process is a "bug free" software code document, and a log of all faults that were found during the code review iterations, along with their description and severity. This kind of ***iterative code review*** (ICR) is the subject of this study.

### 1.2. Using the products of inspection to estimate the number of remaining faults

A software inspection process is satisfactory when the number of undetected document faults is acceptably small. The number of undetected faults may be estimated with fault insertion (injection) techniques, which are labor intensive [13]. A number of methods that avoid fault insertion have therefore been developed [14] [15]. The most known method is "capture-recapture". The capture recapture technique is adapted from the domain of biological studies, where it is used to estimate the size of an animal population in a certain territory. Traps are set in the territory to capture the animals. Once an animal is caught it is tagged and released back into the wild. New traps are set after the tagged animals have had sufficient time to mix with the wild population. Assuming that the percentage of tagged animals in the wild and in the trapped populations is the same, the size of the animal population can be estimated from the known number of tagged animals. The CR method has several variants and estimation models [16] [17].

In order to apply the above techniques to the software inspection domain we replace the animals with software faults, and let the inspectors collect them independently. By labeling all logged faults as "captured" and all faults that were found by more then one inspector as "recaptured", the CR techniques can be used to estimate the total number of faults in the document [18] [19] [20].

An alternative approach called the Detection Profile Method (DPM) was proposed in [21]. The DPM procedure calculates the number of inspectors that detected each defect, and then sorts the defects according to this number. An estimate for the total number of defects is then calculated by fitting a decreasing exponential curve: $M_k = a \times e^{-bk}$, where $M_k$ indicates the total number of inspectors that found a defect $k$. The total number of defects is determined by the largest $k$-value for which this equation provides a result larger than or equal to 1/2. DPM is an intuitively appealing approach that can be easily explained graphically to non-specialists. A later study suggested a method for selecting between a CR model and DPM [22].

A further method is based on the subjective intuitive estimates of the inspectors on the number of defects left in the document [23]. Experienced inspectors were asked to estimate the number of remaining defects on the basis of the defects they discovered by inspection. The results suggest that such estimates are not too bad and may be employed as "a good starting point".

## 1.3. Using N-Fold metrics

Martin and Tsai introduced the N-Fold approach as a method of improving inspection results [24]. The method suggests using $N$ different inspection teams for inspecting the same document. They demonstrated that this approach could improve the results. Two teams found a factor of 1.5 more faults then a single team, and no single fault was found by all teams [25].

The performance of novice and senior students employing N-Fold inspection of user requirement documents has been studied experimentally in [26]. Based on these observations Kantorowitz developed a model for N-Fold inspection. This model employs the metrics $P_{0,1}$ and ***FDRmax*** as well as an estimator ***FDR(j)*** of the ratio between the number of detected faults and the total number of faults as a function of the number of inspectors. The model was employed in a more recent research by the authors of this paper and L. Arzi [27]. The estimator employs the notation $P_{i,j}$ for the probability of fault $i$ to be found by a subset of $j$ of the $N$ inspection teams. $P_{i,1}$ for example, is the probability of fault $i$ to be detected by one team. Like the DPM method, the faults are ordered monotonically and numbered such that $P_{i,1} \geq P_{i+1,1}$. $P_{0,1}$ is therefore the probability of finding the easiest detectable fault by a single inspection team. $P_{i,j}$ can be recursively computed:

$$P_{i,j} = P_{i,j-1} + \left(1 - P_{i,j-1}\right) \times P_{i,1} \tag{1}$$

Or non-recursively:

$$P_{i,j} = 1 - (1 - P_{i,1})^{j} \tag{2}$$

It is possible to estimate the fault detection ratio ***(FDR)*** of a group of $j$ inspectors ($j$ inspection teams each of which is of size 1), by:

$$FDR(j) = \frac{\sum_{i=0}^{N-1} P_{i,j}}{N} \tag{3}$$

By assuming that there are many faults ($N \to \infty$):

$$FDR(j) \approx \int_{i/N=0}^{FDR_{\max}} P_{i,j}(x)\, dx \tag{4}$$

Inspectors with insufficient domain knowledge may not be able to detect certain types of faults. The ratio between the number of faults that these inspectors are able to find and the total number of faults in the document is denoted ***FDRmax.*** By assuming that $P_{i,1}$ varies linearly with $i$, ***FDR(j)*** may be expressed as a function of ***FDRmax*** and $P_{0,1}$:

$$FDR(j) \approx FDR_{\max}\left(1 - \frac{1-(1-P_{0,1})^{j+1}}{P_{0,1}(j+1)}\right) \tag{5}$$

A project manager, that has estimated the values of ***FDRmax*** and $P_{0,1}$ for a given group of inspectors, can employ the estimator (5) to estimate the number of inspectors $j$ required to detect the ratio ***FDR(j)*** of all the faults in the document.

## 2. Adapting the Kantorowitz estimator to Iterative Code Reviews

At first glance, the above algorithm does not seem appropriate for software code documents inspected by iterative code reviews. There are two issues, which must be addressed in order to employ the N-Fold metrics for ICR. First, the algorithm assumes that ***j*** inspectors are employed, whereas in ICR only two inspectors are employed. Second, ICR is an iterative process while the N-Fold algorithm was developed for a single iteration.

In order to adapt the Kantorowitz estimator to ICR we propose using ***j*** to represent the number of iterations instead of the number of inspectors as in the original model. This change of semantics allows the N-Fold model to be applied to the ICR setting.

In the iterative setting $P_{i,j}$ denotes the probability that fault ***i*** will be found by ***j*** iterations. $P_{0,1}$ of this new approach denotes the probability that the easiest detectable fault will be found at the first iteration. ***FDRmax*** now denotes the maximal proportion of faults that can be found by infinite iterations. ***FDR(j)*** is the proportion of all faults that can be found by ***j*** iterations.

Estimating $N_l$, the number of faults that were left in the document after ***j*** iterations, is done using formulas (6) and (7), where $N_j$ is the number of faults found after ***j*** iterations and ***Ne*** is the total number of faults in the document:

$$N_e = \frac{N_j}{FDR(j)} \tag{6}$$

$$N_l = N_j \times \frac{1 - FDR(j)}{FDR(j)} \tag{7}$$

## 3. Experiments

### 3.1. Industrial experiment

In order to test the performance of the modified Kantorowitz estimator, a software industry project was monitored for a period of three years. The project involved the development of a software driver for a hardware communication device, and its incorporation into a communication system (a terra-router system). The software driver was written in the ANSI-C programming language, and included over ten thousands lines of code. The developers of the project were three experienced engineers with more than five years of experience each.

The first six months were dedicated to finalizing the software requirements, specifications and design. All requirements, specifications and design documents were carefully reviewed, using a Fagan inspection. The next twelve months of the project were dedicated to coding and code review inspections. In the following eighteen months QA testers heavily tested the coded application, and detected faults that were undetected by inspection. In parallel, during these eighteen months, the application was integrated into software systems of five prospective clients. A Client standard integration process took between six to twelve months. Through the integration of the software driver with the clients' commercial systems faults that were not detected by the inspection process surfaced. The nature of the faults found by QA testers and by clients was rather similar. It is estimated, that after these eighteen months of thorough testing and usage by customers, all software faults have been detected. In other words, we have an estimation for the total number of faults ***Ne***.

The code review inspections were performed in a consistent manner with industry standards. For each code review iteration the programmer and his team leader met for approximately one hour, and read a code implementation of a specific software module. The programmer presented the code, and together they logged the faults that were found by both of them. Later on, the programmer corrected the code document according to the review results. Another review was scheduled usually about a week or two after the former review took place. The decision to re-inspect a software module was based on the amount of faults found in the code and the importance of the module in the system. In terms of schedule time delays each code review iteration costs approximately one full day of a programmer's time. This cost was taken into account in the project planning.

Five especially critical modules were chosen for our analysis. The criteria for being included in this study was that the module was large enough in terms of lines of code, and that there were several iterative code reviews.

Both values of ***FDRmax*** and $P_{0,1}$ for the inspectors were assumed to be **1.0**. These values were suggested for experienced and domain proficient engineers in [24]. Using these values reduces equation (5) to:

$$FDR(j) = \frac{j}{j+1} \tag{8}$$

For each module three curves were computed:

1. The ***Found*** curve denotes the number of faults actually found in each one of the different iterations.

2. The ***Expected*** curve denotes the number of detected faults expected for the different numbers of iterations ***j***. It is computed using equations (6), (7) and (8), where for each iteration ***j***, the expected value is:

$$Expected(j) = FDR(j) - \sum_{0<K<j} FDR(k) \qquad (9)$$

The ***Expected*** curve, for all code modules, was computed for ***j*** = 1…6, such that the curves for all the different code modules have the same abscissas 1-6. In most code modules, however only 3-5 iterations were done.

3. The ***Estimated*** total number of faults ***(Ne)*** computed from equation (6) with the hitherto found number of faults.

The $H_0$ hypothesis that the expected number of faults detected in each one of the ***j*** iterations fits the actual number of found faults was tested using the $X^2$ goodness-of-fit test. The results of the calculations are shown in tables 1-5 and figures 1-5.

**Table 1 – Module 1 code review results**

| Iteration # | Found by inspection | Expected by FDR(j) | Estimated # Faults |
|---|---|---|---|
| 1 | 14 | 14.00 | 28.00 |
| 2 | 5 | 4.67 | 28.50 |
| 3 | 3 | 2.33 | 29.33 |
| 4 | 1 | 1.40 | 28.75 |
| 5 | | 0.93 | |
| 6 | | 0.67 | |
| | | | |
| **Found by Testing:** | 5 | | |
| **Total # Faults *(Ne)*:** | 28 | | |

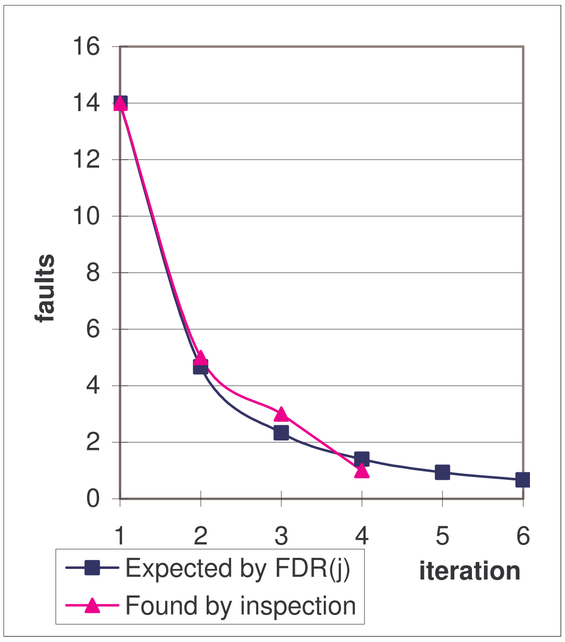

**Figure 1 – Module 1 code review curves**

**Module 1: 4 iterations** were employed, yielding **23 faults**. Later **5 more** faults were found by lab testing and through customer usage. The $H_0$ hypothesis that the expected distribution fits the data was tested using the $X^2$ goodness-of-fit test to produce $\boldsymbol{\alpha < 0.05}$.

**Table 2 – Module 2 code review results**

| Iteration # | Found by inspection | Expected by FDR(j) | Estimated # Faults |
|---|---|---|---|
| 1 | 16 | 15.00 | 32.00 |
| 2 | 5 | 5.00 | 31.50 |
| 3 | 2 | 2.50 | 30.67 |
| 4 | | 1.50 | |
| 5 | | 1.00 | |
| 6 | | 0.71 | |
| | | | |
| **Found by Testing:** | 7 | | |
| **Total # Faults *(Ne):*** | 30 | | |

**Table 3 – Module 3 code review results**

| Iteration # | Found by inspection | Expected by FDR(j) | Estimated # Faults |
|---|---|---|---|
| 1 | 10 | 9.50 | 20.00 |
| 2 | 4 | 3.17 | 21.00 |
| 3 | 2 | 1.58 | 21.33 |
| 4 | 1 | 0.95 | 21.25 |
| 5 | 1 | 0.63 | 21.60 |
| 6 | | 0.45 | |
| | | | |
| **Found by Testing:** | 1 | | |
| **Total # Faults *(Ne):*** | 19 | | |

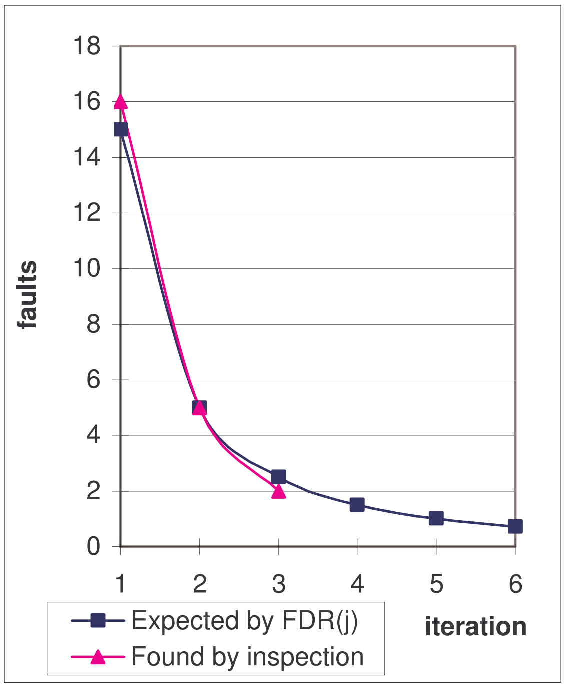

**Figure 2 – Module 2 code review curves**

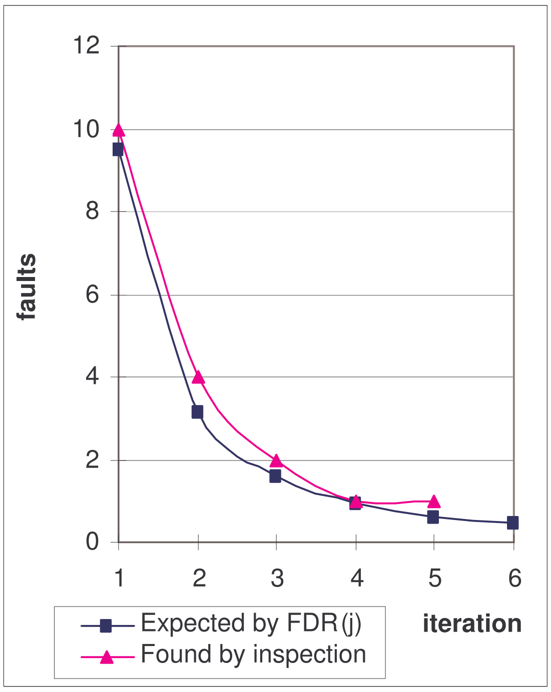

**Figure 3 – Module 3 code review curves**

**Module 2: 3 iterations** were employed, yielding **23 faults**. Later **7 more** faults were found by lab testing and through customer usage. The $H_0$ hypothesis that the expected distribution fits the data was tested using the $X^2$ goodness-of-fit test to produce **$\alpha < 0.08$.**

**Module 3: 5 iterations** were employed, yielding **18 faults**. Later **1 more** faults were found by lab testing and through customer usage. The $H_0$ hypothesis that the expected distribution fits the data was tested using the $X^2$ goodness-of-fit test to produce **$\alpha < 0.04$.**

**Table 4 – Module 4 code review results**

| Iteration # | Found by inspection | Expected by FDR(j) | Estimated # Faults |
|---|---|---|---|
| 1 | 17 | 17.00 | 34.00 |
| 2 | 5 | 5.67 | 33.00 |
| 3 | 3 | 2.83 | 33.33 |
| 4 | | 1.70 | |
| 5 | | 1.13 | |
| 6 | | 0.81 | |
| | | | |
| **Found by Testing:** | 9 | | |
| **Total # Faults *(Ne)*:** | 34 | | |

**Table 5 – Module 5 code review results**

| Iteration # | Found by inspection | Expected by FDR(j) | Estimated # Faults |
|---|---|---|---|
| 1 | 8 | 9.00 | 16.00 |
| 2 | 3 | 3.00 | 16.50 |
| 3 | 2 | 1.50 | 17.33 |
| 4 | 1 | 0.90 | 17.50 |
| 5 | 1 | 0.60 | 18.00 |
| 6 | | 0.43 | |
| | | | |
| **Found by Testing:** | 3 | | |
| **Total # Faults *(Ne)*:** | 18 | | |

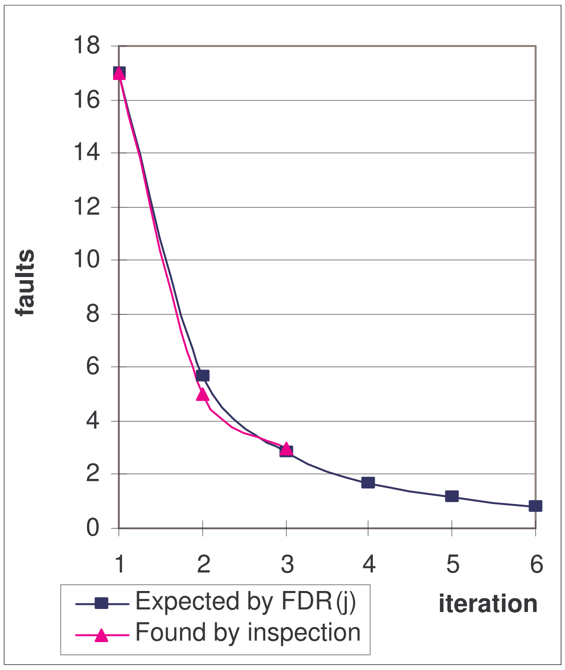

**Figure 4 – Module 4 code review curves**

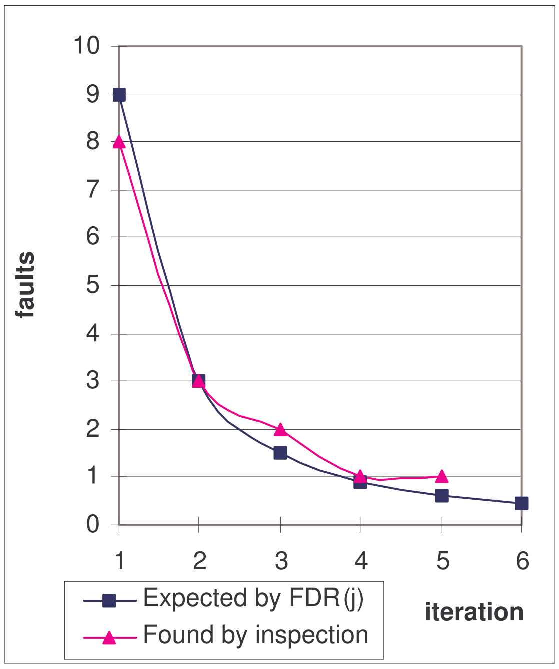

**Figure 5 – Module 5 code review curves**

**Module 4: 3 iterations** were employed, yielding **25 faults**. Later **9 more** faults were found by lab testing and through customer usage. The $H_0$ hypothesis that the expected distribution fits the data was tested using the $X^2$ goodness-of-fit test to produce $\boldsymbol{\alpha < 0.05}$.

**Module 5: 5 iterations** were employed, yielding **15 faults**. Later **3 more** faults were found by lab testing and through customer usage. The $H_0$ hypothesis that the expected distribution fits the data was tested using the $X^2$ goodness-of-fit test to produce $\boldsymbol{\alpha < 0.04}$.

Tables 1-5 and figures 1-5 demonstrate the accuracy of the Kantorowitz estimator in the analyzed ICR processes. In each module, starting from the first iteration, the estimator's error was less then 10%. The accuracy improved with each additional iteration. For four out of the five modules, the $X^2$ goodness-of-fit test showed that the prediction of the Kantorowitz estimator fits the collected data with significance greater than 95%. For the fifth module the significance was greater than 92%.

## 3.2. Using results from other studies

We found no evidence in the literature of any organization applying the ICR process as defined above. Ciolkowski et al. [28] in their thorough review of current industry practices, state that in 35% of the companies there is a second code review iteration, but it is only to make sure that faults found at the first iteration were corrected.

In order to test the Kantorowitz estimator, we applied it to experimental data provided by Wohlin et al.'s research on CR estimators [29]. Wohlin et al. used five ANSI-C code modules named 3A, 4A, 5A, 6A and 7A, defined in [30]. The inspection team was comprised of eight inspectors: four PhD students in computer science, three experienced software engineers and one graduate student. Every inspector reviewed three code documents. There was only a single code review iteration performed in this research.

Using equation (5), we calculated $P_{0,1}$ and ***FDRmax*** values for each inspection team. The results are shown below in table 6.

**Table 6 – Details of inspected programs**

| | No. of Inpsectors | Lines of code | No. of faults | $P_{0,1}$ | FDRmax |
|---|---|---|---|---|---|
| 3A | 5 | 190 | 22 | 0.56 | 1.00 |
| 4A | 5 | 113 | 16 | 0.86 | 1.00 |
| 5A | 5 | 85 | 16 | 0.84 | 1.00 |
| 6A | 5 | 304 | 35 | 0.83 | 1.00 |
| 7A | 4 | 208 | 20 | 0.98 | 1.00 |

In order to verify the Kantorowitz estimator in the ICR setting (i.e. two inspectors, several iterations), we tested the ability of two inspectors to discover faults. Since a team of four or five inspectors inspected each document, we formed all the different possible teams of two inspectors in the given team of four / five inspectors. For each such team of two we computed the number of faults that they detected. We then calculated the average number of faults detected by a team of two inspectors. Afterwards, equation (6) was used to estimate ***Ne***. The results of the comparison between the estimated ***Ne*** and the known number of faults are presented in Table 7. The table also shows the average minus the standard deviation (STD).

**Table 7 – *Ne* ($P_{0,1}$ & *FDRmax* known)**

| | Ne | Ne(avg. minus STD) | Ne (avg.) | Ne (avg. plus STD) |
|---|---|---|---|---|
| 3A | 22 | 14.026 | 19.101 | 24.177 |
| 4A | 16 | 10.873 | 14.997 | 19.121 |
| 5A | 16 | 12.071 | 14.881 | 17.691 |
| 6A | 35 | 23.634 | 32.314 | 40.993 |
| 7A | 20 | 15.279 | 18.691 | 22.102 |

Table 7 demonstrates that when $P_{0,1}$ and ***FDRmax*** are known the Kantorowitz estimator produced reasonably accurate results after a single iteration (with an error of less then 10%). However, in this case we were lucky to have the results of earlier inspections with four or five inspectors, so $P_{0,1}$ and ***FDRmax*** could be estimated beforehand. A more relevant case to the ICR setting would be the use of the Kantorowitz estimator without such prior knowledge. In such cases, as in the industrial experiment above, we suggested using $P_{0,1}$ = ***FDRmax*** = **1.0**. Table 8 presents the results using these values.

**Table 8 – *Ne* ($P_{0,1}$ & *FDRmax* unknown)**

| | Ne | Ne(avg. minus STD) | Ne (avg.) | Ne (avg. plus STD) |
|---|---|---|---|---|
| 3A | 22 | 12.776 | 17.4 | 22.024 |
| 4A | 16 | 13.34 | 18.4 | 23.46 |
| 5A | 16 | 14.601 | 18 | 21.399 |
| 6A | 35 | 28.378 | 38.8 | 49.222 |
| 7A | 20 | 20.165 | 24.667 | 29.169 |

Table 8 demonstrates that even when $P_{0,1}$ and ***FDRmax*** are unknown the Kantorowitz estimator can produce reasonably accurate results after a single iteration (with an error of less then 20%). As shown in our industrial test case, accuracy may improve when applying more iterations. Notice that with a single code review iteration, the estimator tends to over estimate when the inspectors are good (as in module 7A, where $P_{0,1}$ = 0.98 and ***FDRmax*** = 1.0), and tends to under estimate when the inspectors are not so good (as in module 3A).

## 4. Discussion

The experimental results suggest the applicability of the Kantorowitz estimator for experienced engineers inspecting a familiar kind of code documents, i.e. when ***FDRmax* = 1.0, and $P_{0,1}$ = 1.0**. Applying the estimator on experimental data from the literature also suggests its applicability to a single iteration ICR process. The estimator was also applicable on published data for a number of single iteration ICR processes.

### 4.1. Estimating *FDRmax* and $P_{0,1}$

Equation (5) requires estimates of $P_{0,1}$ and ***FDRmax***. Three different approaches may be applied to obtain these estimates:

1. Test the inspectors beforehand with a software document injected with faults. This technique was employed in [27].
2. Employ the values observed in [27]. We employed in the analysis of our experiments the values observed in [27] for experienced engineers.
3. Use $P_{0,1}$ and ***FDRmax* = 1.0** in order to estimate the lower bound on the number of required iterations. The required number of iterations is higher or equal to this estimated lower bound.

### 4.2. Comparing N inspectors to N inspections

The original N-fold inspection method works because different inspectors see different faults, so together they detect more faults than a single inspector. Our experiments showed that the ICR inspection also works well in spite of using the same inspectors in different reviews. One possible explanation is that in later reviews they inspect code documents where faults detected earlier have already been corrected. They thus inspect a better document, which facilitates the detection of further faults. Other possible explanations are the time lapse of 1-2 weeks between the inspections and the better insights in the document gained through repeated inspections.

### 4.3. Deciding whether to re-inspect

The estimator introduced in this paper may be used in order to decide whether or not to continue the review process. In our case study modules 3 and 5 were the most important modules of the project as their functionality lay at the heart of the system and all other modules needed to use their facilities. Therefore, both modules were reviewed with the goal of eliminating all software faults. It was found (Table 3 and 5) that iterations 4 and 5 of both modules produced only one new fault each and that our estimator predicts that a 6th inspection would detect only 0.43 and 0.45 faults respectively. This resulted in a decision to terminate the review process. The cost of a 6th iteration, i.e. 2 engineer hours, was not considered worthwhile. It was estimated that possible remaining faults would be found later by other QA means.

### 4.4. Comparison with other methods

A number of methods have been developed to estimate the number of undetected faults left in software code documents [14] [31] [32]. However, none of these methods addresses the ICR procedure, which is the subject of this paper. The methods published hitherto require a large, e.g. 5-6, number of inspectors. These methods may not be practical in small industries where it is difficult to organize large inspection teams. Our model provided estimates sufficiently accurate for determining the number of required iterations. This was achieved by a two-person iterative code review (ICR) process. We observed that the ICR process integrates well in the software development process. It is noted that Porter et al. [8] also considers team of two to be especially advantageous.

El-Emam and Laitenberger conducted a comprehensive evaluation of capture-recapture techniques for code documents inspected with two inspectors [33]. Their study used the common capture-recapture technique with the information provided by a single code review iteration, in order to decide whether to re-inspect. They observed that their estimator re-inspection decision was accurate in less then 70% of the cases, and suggested therefore further research.

### 4.5. Taking fault severity into account

Large-scale projects usually maintain a list of faults with their respective severity. It is possible to create different ***FDR(j)*** curves for each severity in each inspected module, in order to fine-tune the re-inspection decision. For instance, a reasonable decision may be to continue the review process, until no faults of severity one and few faults of severity two are left in the code. Such a decision may be based on a comparison of the economic penalty of an unfound fault of severity one with the further inspection costs.

## 5. Conclusions

This research considers the iterative code review (ICR) process, which involves a sequence of reviews. The advantage of ICR is that it may be employed with only two inspectors. Its disadvantage is that it takes more time to perform a sequence of iterative reviews with two inspectors than a single review with a large number of inspectors. This research presents a new method for estimating the number of faults remaining in code documents inspected by the ICR process. In the industrial case study reported in this paper, the estimator was sufficiently accurate for determining when to stop the ICR process. The case study involved experienced inspectors working on system code. Applying the estimator on data published in the literature also suggested its applicability. When working with experienced and domain proficient inspectors, the simple equations (6), (7), (8) and (9), may provide sufficient accurate estimates.

Further experiments are required in order to evaluate the applicability of ICR in additional situations. It is especially important to check its validity with other types of code and inspectors. Possible future research may also attempt to take the severity of different faults into account

## 6. References

[1] M.E. Fagan: "Design and Code Inspections to Reduce Errors in Program Development", In IBM Systems Journal, 15(3), p. 182-211, 1976.

[2] T. Gilb, D. Graham: "Software Inspection", Addison-Wesley Publishing, 1993.

[3] O. Laitenberger: "A Survey of Software Inspection Technologies", Handbook on Software Engineering and Knowledge Engineering, World Scientific Publishing, 2002.

[4] A.F. Ackerman, L.S. Buchwald, F.H. Lewsky: "Software Inspections: An Effective Verification Process", IEEE Software, 6(3), p. 31-36, 1989.

[5] J.C. Knight, E.A. Myers: "Phased Inspections and their Implementation", ACM SIGSOFT Software Engineering Notes, 16(3), p. 29-35, 1991.

[6] S. Rifkin, L. Deimel: "Applying Program Comprehension Techniques to Improve Software Inspection", proceedings of the 19th annual NASA software eng. laboratory workshop, NASA, 1994.

[7] P.M. Johnson, D. Tjahjono: "Does Every Inspection Really Need a Meeting", Journal of Empirical Software Engineering, 3(1), p. 9-35, 1998.

[8] A. Porter, H. Siy, C. Toman, and L.G. Votta: "An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development", IEEE Transactions on Software Engineering, 23(6), p. 329-346, 1997.

[9] R. Jeffery, T. Stalhane, C. Kutay and H. Al-Kilidar: "Teaching the Process of Code Review", Australian Software Engineering Conference, p. 271, 2004.

[10] D. Bisant and J. R. Lyle: "A Two-Person Inspection Method to Improve Programming Productivity", IEEE Software Engineering, 15, p.1294-1304, 1989.

[11] E.P. Doolan: "Experience with Fagan's Inspection Method". Software Practice and Experience, 22(3), p. 173-182, 1992.

[12] E. Yourdon: "Structured Walkthroughs". Prentice Hall, 4th edition, N.Y, 1989.

[13] J.D. Musa, A. Ianino, K. Okumoto: "Software Reliability: Measurement, prediction, application", McGraw-Hill, Inc., New York, NY, 1987.

[14] P. Runeson, C. Wohlin: "An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections", Journal of Empirical Software Engineering, 3(4), p. 381-406, 1998.

[15] L. Briand, K. El Emam, B. Freimut: "A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method", in Proceedings of the 9th International Symposium on Software Reliability Engineering, p. 32-41, 1998.

[16] D. Otis, K. Burnham, G. White, D. Anderson: "Statistical Inference from Capture Data on Closed Animal Populations", Wildlife Monographs, 62: p. 1-135, 1978.

[17] G. White, D. Anderson, K. Burnham, D. Otis: "Capture-Recapture and Removal Methods for Sampling Closed Populations", Technical Report LA-8787-NERP, Los Alamos National Laboratory, 1982.

[18] J. Duran, J. Wiorkowski: "Capture-Recapture Sampling for Estimating Software Error Content". IEEE Transactions on Software Engineering, 7(1): p. 147-148, 1981.

[19] S.G. Eick, C.R. Loader, M.D. Long, L.G. Votta, S.V. Wiel: "Estimating Software Fault Content before Coding", In Proceedings of the 14th International Conference on Software Engineering, p. 59–65, 1992.

[20] J. Miller: "Estimating the Number of Remaining Defects After Inspection", Software Testing, Verification and Reliability, 9, p. 167-189, 1998.

[21] C. Wohlin, P. Runeson: "Defect Content Estimations From Review Data", The 20th International Conference on Software Engineering, p. 400-409, 1998.

[22] L.C. Briand, K. El Emam, B.G. Freimut, "A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method", International Software Engineering Research Network, ISERN-98-11, 1998.

[23] K. El Emam, O. Laitenberger, H. Harbich: "The Application of Subjective Effectiveness to Controlling Software Inspections", Journal of Systems and Software, 54(2), 2000.

[24] J. Martin, W.T. Tsai: "N-Fold Inspection: A Requirements Analysis Technique", Comm. ACM, 33, p. 225–232, 1990.

[25] G. M. Schneider, J. Martin, W.T. Tsai: "An Experimental Study of Fault Detection In User Requirements Documents", ACM Trans. on SW Eng. and Methodology, 1, p. 188-204, 1992.

[26] E. Kantorowitz, L. Arzi, A. Gutmann: "The profile of N-fold requirements method", Requirement Engineering Journal, 2, p. 152-164, 1997.

[27] E. Kantorowitz, L. Arzi and A. Harel: "A Method for Evaluation of the Performance of Requirements Inspections", Proceedings of the 11th European Software Control and Metrics Conference, p. 355-358, 2000.

[28] M. Ciolkowski, O. Laitenberger, S. Biffl: "Software Reviews The State of the Practice", IEEE Software, 20(6), p. 46-51, 2003.

[29] P. Runeson, C. Wohlin: "An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections", Journal of Empirical Software Engineering, 3 (4), p. 381-406, 1998.

[30] Humphrey: "A Discipline for Software Engineering", Addison-Wesley, 1995.

[31] J. Barnard, K. El Emam, D. Zubrow: "Using Capture-Recapture Models for the Reinspection Decision", Software Quality Professional, 5 (2), p. 11-20, 2003.

[32] L. Briand, K. El Emam, B. Freimut, O. Laitenberger: "A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content", IEEE Transactions on Software Engineering, 26 (6), 2000.

[33] K. El Emam, O. Laitenberger: "Evaluating Capture-Recapture Models with 2 Inspectors", IEEE Transactions on Software Engineering, 27 (1), p. 851-864, 2001.