APPLICATION OF THE KUSUMOTO COST-METRIC TO EVALUATE THE COST-EFFECTIVENESS OF SOFTWARE INSPECTIONS

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Narendar Reddy Mandala

In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

March 2012

Fargo, North Dakota

North Dakota State University

Graduate School

Title

Application of the Kusumoto Cost-Metric to Evaluate the Cost-Effectiveness of Software Inspections

By

Narendar Reddy Mandala

The Supervisory Committee certifies that this ***disquisition*** compiles with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Gursimran Singh Walia
Chair

Dr. Kendall Nygard

Dr. Hyunsook Do

Dr. Chao You

Approved by Department Chair:

03/27/2012
Date

Dr. Brian M. Slator
Signature

ABSTRACT

Inspection and testing are two widely recommended techniques for software quality improvement with a common goal of defect detection and removal in software products. While testing cannot be conducted until software is implemented, inspections can be applied in early stages of software development. In this way inspection enables saving of testing cost and time. To manage the quality of their software, project managers need objective information to make a trade-off between the testing costs saved by performing inspections against the testing cost that would otherwise be spent if no inspections were performed. Project managers also need to decide on the number of inspectors that would make it a cost-effective inspection process. To accomplish these research goals, we have analyzed the cost invested in the inspection process against the cost saved from the inspection process by applying the *Kusumoto* cost-metrics on an inspection data set with varying number of inspection team size.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my advisor Dr. Gursimran Singh Walia for his continued support throughout this paper. I appreciate his time, assistance and continuous guidance. I would also like to thank Dr. Kendall Nygard, Dr. Hyunsook Do, and Dr. Chao You for being a part of my graduate supervisory committee. Special thanks to the faculty and staff of the Computer Science Department for their unconditional support throughout my master's program. Finally, I am grateful to my parents who are the reason for all my achievements and have supported and motivated me throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The Success of a software development organization depends on their ability to deliver a quality end product. To ensure software quality, researchers and practitioners have devoted considerable effort to help developers find and fix problems early in the lifecycle. By identifying problems early (i.e., requirements and design documents), organizations reduce the likelihood that these will propagate to subsequent phase. In addition, finding and fixing problems earlier rather than later is easier, less expensive and reduces avoidable rework [37, 38]. It is estimated that 40-50% of development effort is spent on avoidable rework i.e., fixing problems that should have been fixed during the early stages of the development [7].

Among the various techniques used for identifying and repairing problems early in the lifecycle, software inspections have become an important part of the quality assurance effort for software products [39]. Inspections are a process whereby software artifacts are examined by a group of inspectors to ensure that they meet some set of quality constraints. Another common goal for inspections is to uncover defects in the artifact.

The main idea of the inspection as defined by Fagan is as follows. Once the author completes a software artifact, which could be a requirements document, design, or code, they submit it for inspection. The inspection consists of multiple steps. The inspection leader first chooses a team of people who will perform the inspection. Then, the document to be inspected is distributed to these team members. The author of the document gives a brief overview and background to provide some context. The team-members spend time individually reading over the document and defects are uncovered by the inspectors. The defects are collected in a list, which is returned to the document author. The author then fixes these defects, or explains why they are not defects. Since the initial definition of the inspection process, many variations

have been made on it. However, the main goal of software inspection is to remove the defects in the software work product right after their injection. In this way inspection enables; 1) saving of cost and time, which needs to be expended if the defects pass to later stages of software development life cycle; and 2) improving the quality of software product by enhancing its reliability, maintainability and availability [26].

Inspection [Fagan 1976 and 1986; Ackerman et al. 1989] and testing are two widely recommended techniques for software quality improvement. While both of them are used for defect detection and removal in software products, testing cannot be conducted until software is implemented, whilst inspections can be applied in early stages of software development process and help to avoid costly rework [10]. Fagan [12] a member of the IBM technical staff, claims that the inspection process finds 60-90% of all defects and the cost for inspections amounts typically up to 15% of the project cost.

To manager the quality of their software, project managers need objective information to make a trade-off between the testing costs saved by performing inspections during the early stages of development against the testing cost that would otherwise be spent if no inspections were performed. Furthermore, project managers also need to be able to decide when to stop the inspection process based on the number of defects found during an inspection. Additionally, it has been well documented that increasing the inspection team size helps uncover a larger number of defects present in the document. Therefore, project managers also need objective information to decide on the number of inspectors that would make it a cost effective inspection process. To accomplish these research goals, we have analyzed the cost invested in the inspection process against the cost saved from the inspection process by applying the *Kusumoto* [11] cost-metrics

on a diverse set of inspection data sets with varying number of inspection team size. Details of the cost-metrics and the *Kusumoto* cost model are presented in Section 2.1.

An additional challenge for project managers is that the total number of defects present in an artifact is not known prior to the inspection. Furthermore, inspections provide insights into the number of defects that remain post-inspection. Therefore, an accurate estimate of the actual number of defects in the software artifact can aid the project managers to help evaluate the cost efficiency of the inspections. Among the different approaches that are available for estimating the number of defects in the artifact (e.g., defect density, subjective assessment, historical data, capture-recapture method, curve-fitting method), the most appropriate and objective approach is the capture-recapture method [2, 4].

Capture-recapture (CR) is a statistical method originally developed by biologists to estimate the size of wildlife populations. To use CR, a biologist captures a fixed number of animals, marks them, and releases them back into the population. The biologist then captures the same number of animals again. If an animal that was 'marked' during the first trapping is caught again, it is said to have been recaptured. The process of trapping and marking can be repeated multiple times. The size of the population is estimated using: 1) the total number of unique animals captured across all trapping occasions, and 2) the number of animals that were re-captured. A higher percentage of recaptures indicates a smaller population [5, 23].

Using the same principle, the CR method can be used during the inspection process to estimate the number of faults in an artifact. During an inspection, each inspector finds (or captures) some faults. If the same fault is found by more than one inspector it has been re-captured [3, 10]. The total number of faults is estimated using the same process as in wildlife

research, except that the *animals* are replaced by *faults* and the *trappings* are replaced by *inspectors*. The inspection team can use this result along with the number of faults already detected to estimate the number of faults remaining in the artifact. More details of the capture-recapture models and estimators are provided in Section 2.2

Our earlier research have used capture recapture models for estimating defect count in the software product and are proved to be reliable with an appropriate number of inspectors used in inspection process [3,22,40]. In this research, we investigate whether the capture-recapture models can be used by the project managers to evaluate the cost effectiveness of inspections when the total defect count in an artifact is not known before.

To summarize, this paper evaluates the cost effectiveness of software inspections with varying number of inspection team size on diverse inspection data sets. We also evaluate the cost effectiveness of software inspections for the same data sets by using the estimates from CR models and compare both the results. These evaluations provide an insight to the project managers and developers in making a cost effective decision on when to stop the inspection process and the significance of using CR estimates when the fault count of the artifact is unknown.

The paper is structured as follows: Section 2 describes the software inspection metrics and proposes a best metric for evaluating the cost effectiveness of software inspections, the basic principles of capture-recapture models and their application to software inspections. Section 3 discusses the literature review useful for the evaluation study. Section 4 describes the design of the study for evaluating the cost effectiveness of software inspections. Section 5 describes the data analysis and results. Section 6 summarizes the evaluation results and future work.

2. BACKGROUND RESEARCH

This section provides background information regarding the software inspection cost models, the basic principles of the capture-recapture models, and a survey of the results from the empirical studies that have been conducted to evaluate the capture-recapture models and the cost-effectiveness of software inspections.

2.1 Inspection Cost Models

To evaluate the benefits of software inspection and to motivate their use, various models have been developed to capture the benefits of inspection. In this section we show some models that address various evaluation aspects like number of defects found during an inspection, inspection efficiency, and cost effectiveness. The following traditional cost model describes all costs related to inspections.

2.1.1 Traditional Cost Model

The traditional inspection cost model [11] consists of the following five components that are described in this section and shown in Figure 1:

a) D_{total} - total number of defects in the product existing before inspection;

b) C_r – cost spent for inspection;

c) D_r - number of unique defects found during inspection;

d) C_t – cost to detect remaining defects in testing;

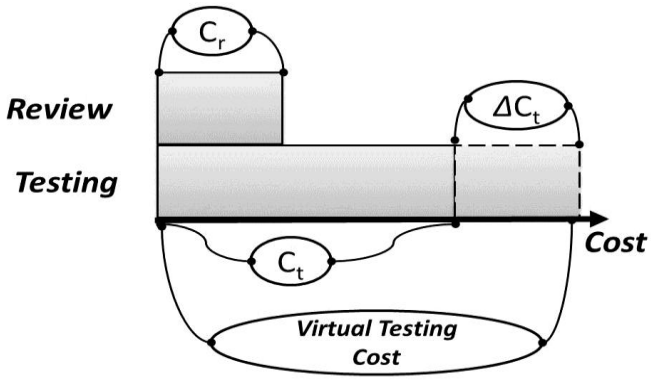


Figure 1. Traditional Cost Model

e) ΔC_t – testing cost saved by inspection;

The process of calculating each of these components is described as follows:

a) C_r – *Cost spent on Inspections*, is measure of the time taken to perform the inspection process in terms of man hours. This is the total cost invested on the inspection process and is calculated by adding the time taken by each inspector during inspection process.

$$C_r = \sum T_i \qquad (T_i \text{ , time taken for each inspector during inspection}) \ldots \text{Eq 2.1}$$

b) C_t – *Testing Cost*, is the cost required for detecting the remaining defects that are not detected during the inspection process. If we consider c_t as the average cost to detect a defect in the testing phase then from the figure 1, the testing cost is measured as the product of total number of undetected defects times the Average cost to detect a defect in testing.

$$C_t = (D_{total} - D_r) * c_t \quad \ldots\ldots\ldots \quad Eq\ (2.2)$$

c) D_{total} – *Total Defects*, can be determined by using an artifact that is seeded with a known number of defects. In case the total number of defects is not known at the beginning of the inspection, it can be determined using the capture recapture estimates.

d) c_t, – *Average cost to detect a defect in testing*, is measured as a factor of the average cost of detecting a defects during an inspection. To arrive on this measure, we surveyed the literature (described in section 3.2).

e) ΔC_t – *Cost saved by inspections*. By spending cost C_r during inspection, the cost ΔC_t is being saved during testing. It is calculated as the product of number of defects found in inspections (D_r) times the average cost to detect a defect in testing c_t.

$$\Delta C_t = Dr * c_t \ldots\ldots\ldots \text{Eq (2.3)}$$

(ΔC_t - C_r) Reduction of the total costs, It is calculated as the difference between the testing cost saved by inspections to the cost consumed by the inspections.

Virtual testing cost ($C_t + \Delta C_t$) is the testing cost if no inspections are performed. It is the sum of the testing cost and the cost saved by inspections.

$$\text{Virtual testing cost} = (C_t + \Delta C_t) \quad \ldots\ldots\ldots\ldots\ldots \text{Eq (2.4)}$$

2.1.2 Metrics to Evaluate Software Inspections

Several metrics have been proposed to evaluate the effectiveness of the inspections with respect to software development cost using some or all of the elements described in the cost model shown in Figure 1. In this section, we describe different metrics and discuss their limitations to select the most appropriate model for evaluating the cost-effectiveness of an inspection process. A brief description of metric follows:

a) ***Myers Metric (M_m)***: In order to evaluate the effectiveness of design reviews and code reviews, Myers proposed a metric that is equivalent to the number of defects detected by inspectors [Myers 1978] [8]. Myers metric would have been effective as a review metric if same number of defects is assumed to be in each product before their reviews, but different products have different number of total defects. Myers metric also does not account for the cost expended for inspections or cost that has been saved from inspections which is the main requirement in our current research.

$$M_m = Dr \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Eq (2.5)}$$

b) ***Fagans Metric (M_f)***: Fagan has evaluated the effectiveness of design reviews, code reviews, and unit test reviews using the metric M_f, called an Error Detection Efficiency. M_f

is defined to be the number of faults found by reviews over the total number of faults in the product before its reviews [6].

$$M_f = \text{Number of Faults Found} / \text{Total Number of Faults}$$

$$M_f = Dr / D_{total} \quad \text{…….. Eq (2.6)}$$

Fagans metric also does not take the cost factor into consideration. The proposed model may be effective but in real time software development the project managers would like to make a decision which is cost effective and gives a good return on investment. Our current topic of research is also in evaluating the cost effectiveness of inspections rather than just finding just the number of faults, hence this model fails in evaluating the cost effectiveness of software inspections.

c) **Collofello's Metric (M_c):** Collofello & Woodfield proposed a metric called *Cost Effectiveness* [Collofello & Woodfield 1989] [9] defined as the ratio of the "cost saved by the process" to the "cost consumed by the process".

$$M_c = \text{Cost saved by inspections} / \text{Cost consumed by inspections}$$

$$M_c = \Delta C_t / C_r \quad \text{…….. } \textit{Eq (2.6)}$$

Collofello metric is also inappropriate in evaluating the cost effectiveness of software reviews as M_c does not take into account the total cost to detect all faults from the software by reviews and testing. The models proposed by Collofello/Woodfield and Grady/van Slack might be problematic if we attempt to compare their results across projects. The following example illustrates why this is so. Suppose two projects with two defect detection activities: inspections and testing. Assume further, that in both projects, if inspections had not been performed, the costs of testing would be 1000 units. The first

project consumes 10 cost units for their inspections and saves 100 units. Thus, the total cost for defect detection is 910 units. In the second project inspections cost 60 units and save 600. Thus, the total cost for defect detection is 460 units, which is far smaller than the cost in the first project of 910. However, in both projects the value of collofello metric M_c of 10 would have been computed, which would prevent us from recognizing the economic advantage of inspections in the second project. This could have been eliminated if we had considered reduction of total cost in detecting faults by inspection rather than considering the cost saved by the inspection process.

d) **Kusumoto Metric M_k :** Kusumoto et al. proposed a metric for evaluation the *Cost Effectiveness* of inspection in terms of reduction of cost to detect and remove all defects from software product. It is a ratio of the reduction of the total costs to detect and remove all defects from the software product to the virtual testing cost (testing cost if no inspection is executed). The model proposed by Kusumoto normalizes the estimated savings by the potential defect cost. Hence, it can be compared across different types of inspections. This model proposed by Kusumoto is closest to our requirements.

M_k = Reduction of total costs to detect all faults / Virtual testing cost.. Eq (2.7)

M_k is a ratio of the reduction of the total costs to detect and remove all defects from documents using inspections in a project to the virtual testing cost. The testing cost is reduced by (ΔC_t - C_r) compared to the virtual testing cost ($Ct + \Delta Ct$) if no inspection is executed. According to the above notations in the traditional cost model the kusumoto metric can be derived as

$$M_k = (\Delta C_t - C_r) / (C_t + \Delta C_t) \text{ .... } Eq\ (2.7)$$

$$= ((Dr * c_t) - (Dr * c_t)) / (D_{total} * c_t)$$

$$= (Dr / D_{total}) * (c_t - c_t) / c_t$$

$$= M_f \ (1- 1/M_c) \qquad \text{…….. Eq (2.7)}$$

From above equation (2.7) the kusumoto metric can be interpreted as a combination of Fagans Metric (M_f) and Collofello's metric (M_c). Kusumoto metric is more practical than the metrics Fagan's and Meyer's as it takes cost into consideration. When considered with collofello's metric, kusumoto model is intuitive as it can be interpreted as the percentage of defect rework costs that are saved due to inspections. With this definition of Kusumoto model, cost-effectiveness is stated in terms of effort savings and can be compared across different types of inspections (e.g., between inspections of different projects or in different phases of the life cycle). Hence in our experimental evaluation of cost effectiveness of software inspections we use the Kusumoto metric.

2.2 Use of Capture Recapture Models in Software Inspections

The use of the CR method in biology makes certain assumptions that do not always hold for software inspections. The assumptions made by CR method in biology include: 1) a closed population (i.e. no animal can enter or leave), 2) an equal capture probability (i.e. all animals have an equal chance of being captured), and 3) marks are not lost (i.e. an animal that has been captured can be identified) [23]. When using the CR in software inspections, the closed population assumption is met (i.e., all inspectors review the same artifact and it is not modified) and the assumption that marks are not lost is met (i.e. it can be determined if two people report the same fault). However, because some faults are easier to find than others and because inspectors have different abilities, the equal capture probability assumption is not met [3, 23].

Table 1. Capture-recapture Models

Model	Variation Source
$\mathbf{M_o}$	All inspectors have the same detection ability, and all defects are equally likely of being detected.
$\mathbf{M_t}$	Inspectors differ in their defect detection abilities, but all defects are equally likely of being found.
$\mathbf{M_h}$	Inspectors have the same detection ability, but defects differ in their probability of being found.
$\mathbf{M_{th}}$	Inspectors differ in their defect detection ability, and defects differ in their probability of being found.

Table 2. Capture-recapture Estimators

Models	Estimators
$\mathbf{M_o}$	Unconditional Maximum Likelihood Estimator (M_o-UMLE) [5]
	*Conditional Maximum Likelihood Estimator (M_o-CMLE) [30]
	*Estimating Equations (M_o-EE) [31]
$\mathbf{M_t}$	Unconditional Maximum Likelihood Estimator (M_t-UMLE) [5]
	*Conditional Maximum Likelihood Estimator (M_t-CMLE) [30]
	*Estimating Equations (M_t-EE) [31]
	Chaos Estimator (M_t-Ch) [32]
$\mathbf{M_h}$	Jackknife Estimator (M_h-JK) [29]
	*Sample Coverage (M_h-SC) [34]
	*Estimating Equations (M_h-EE) [31]
	Chaos Estimators (M_h-Ch) [33]
$\mathbf{M_{th}}$	*Sample Coverage (M_{th}-SC) [34]
	*Estimating Equations (M_{th}-EE) [31]

To accommodate these different assumptions, four different CR models are built around the two sources of variation: Inspector Capability and Fault Detection Probability. Table 1 shows the four CR models along with their source(s) of variation [28,23]. Each CR model in Table 1 has a set of estimators, which use different statistical approaches to produce the estimates. The estimators for each CR model used in this study are shown in Table 2. These estimators include estimators that have been evaluated in previous software inspection studies as well as new estimators from biology that have not previously been applied to software inspections (marked with an *).

The mathematical details of estimators are beyond the scope of this paper but can be found in provided references. The input data used by all the CR estimators is organized

as a matrix with rows that represent faults and columns that represent inspectors as shown in Figure 2. A matrix entry is 1 if the fault is found by the inspector and 0 otherwise.

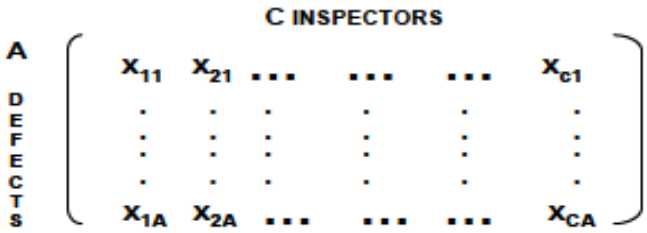


Figure 2. Capture Recapture Data Input Matrix

Our previous research on the use of capture recapture in software inspections has validated the ability of the CR estimators to accurately estimate the actual defect count of a software document based on the number of defects found by inspectors during an inspection process [41]. Our prior research has also shown a positive effect of the number of inspectors on the performance of the CR estimates. However, most of our prior CR research has been neglectful of the cost spent and cost saved by adding more number of inspectors during an inspection process. This research extends our prior work by evaluating the cost-effectiveness of inspection process with varying number of inspection team size. We have also evaluated the effect of the inspection team size on the CR estimator's ability to accurately predict the cost-effectiveness of an inspection process. The results from this research will provide guidance on how to appropriately use the cost metrics to evaluate the cost-effectiveness of software inspections and post-inspection decisions based on the CR estimates.

3. LITERATURE REVIEW

As mentioned earlier, in order to evaluate the cost effectiveness of software inspections, we need to know the number of defects found by the inspection process, the total time taken to complete an inspection, the number of defects left in the document after the inspection, and the testing cost saved from the inspection process (i.e. the cost that would have incurred during testing if there was no inspections).

The inspection data regarding the number of defects detected during an inspection and the time spent during the inspection are obtained at the end of the review process. However, the total number of faults still remaining in the document post-inspection can only be determined if the document was seeded with a known number of defects. In case the actual number of defects is not known, capture recapture estimators can help in estimating the remaining number of defects left which can be used for evaluating the cost effectiveness.

Similarly, the testing cost saved from inspections is not available at the end of an inspection process. In order to calculate the testing time that would have been expended if no inspection was performed, we survey the literature to investigate the amount of testing time that can be saved by the corresponding effort spent during inspections.

The following section briefly presents the results reported in various experimental settings that provide information on the costs and efforts expended on the inspections and the corresponding amount of effort or costs being saved during the testing. The result from the literature review provides an estimate on the amount of testing time saved by the corresponding effort spent during the inspections at various software organizations. We summarize these results to find a relation between the average cost to find a defect during inspections and the average

cost to find a defect in testing when no inspection occurs, which is then used in evaluating the cost effectiveness of inspections data sets later in the paper.

3.1 Testing Cost Saved by Performing Reviews during Early Stages of Software Development

Gilb and Graham [14] reported data from various sources in their discussion of the benefits and costs of inspections. Barbara Kitchenham also reported on experience at Imperial Chemical Industries (ICL) that the cost of finding one defect in design inspections was 1.58 work hours and the cost of finding one defect without inspection was more than five times more, and was 8.47 work hours. Boehm reports that the cost ratio for design inspections to testing at TRW (an American corporation involved in a variety of businesses, mainly aerospace, automotive, and credit reporting) was 1:6.25 [17].

Lionel Briand, in the industry study of one of his paper [18] reported a banking computer-services firm found that it took 4.5 hours to eliminate a defect by unit testing compared to 2.2 hours by inspection. Also an operating-system development organization for a large mainframe manufacturer reported that the average effort involved in finding a design defect by inspections is 1.4 staff-hours compared to 8.5 staff-hours of effort to find a defect by testing.

Weller [19] reports data from a project that performed a conversion of 1200 lines of C code to Fortran for several timing-critical routines. While testing the rewritten code, it took 6 hours per failure. It was known from a pilot project in the organization that they had been finding defects in inspections at a cost of 1.43 hours per defect. Thus, the team stopped testing and inspected the rewritten code detecting defects at a cost of less than 1 hour per defect. This shows that the defects are found at a cost 6 times less with inspections when compared to testing.

Cardiac Pacemakers Incorporated (Olson, 1995) [20] utilizes inspections to improve the quality of its life critical software. Olson estimates that if the cost to fix a defect during design is 1X, then fixing design defects during test is 10X and in post-release is 100X. He estimates the effort to fix and verify a defect once detected in the process is 0.25 to 0.5 hours during requirements and design, using the inspection process, versus 5-10 hours during systems integration and test, utilizing testing, which is a 10-20:1 ratio. Costs incurred were the cost of inspections (5-15% of the total project), startup costs and overhead. Cost reductions are the estimated cost by phase without inspections minus the actual costs by phase with inspections.

Madachy (1995) described utilization of inspections at Litton Data Systems [24]. Litton has experienced a 30% reduction in the number of errors found during systems integration and test. 400 people at Litton have been trained and 600 inspections have been performed. On one project they have experienced a 50% reduction in integration effort. Madachy estimates that 2.3 staff hours are saved in systems testing for every inspection hour. Which means the average cost to detect a defect in testing (when no inspection occurs) is 3.3 times the average cost to detect a defect in inspections.

Lionel Briand [18], Based on most of the data published in the literature has given an industry-wide benchmark, which intends to capture the current practice regarding inspection efficiency. He provided probability distribution parameters for the average effort using different defect detection techniques according to which the minimum, most likely and maximum value for design inspections are 0.58, 1.58 and 2.9 hours per defect respectively and for Testing are 4.5, 6 and 17 hours per defect. These values were summarized and derived from various literature studies on the cost and effort of finding defects in design, code reviews and testing. In constructing distributions, not all the data available in the literature was used and only

information resulting from precise data collection performed on actual defect detection activities was considered (i.e., they did not consider approximations or estimates, or data whose origins were unclear).

Table 3. Summary of Average Testing Cost Savings by Performing Early Inspections

S. No.	Author	Context	Ratio	Ref.
1.	Barbara Kitchenham at ICL	Reported their experience on cost of finding one defect in design inspections and cost of finding defect without inspections.	1.58 : 8.47 Or 1: 5.36	[14]
2.	B. Boehm at TRW	In his book reports on cost ratio of design inspections to testing.	1:6.25	[17, 18]
3.	Lionel Briand	Reports a banking computer services firm findings on the effort to eliminate the defects by inspection to unit testing.	2.2 : 4.5	[18]
4.	Lionel Briand	Reports the findings of an operating-system development organization for the effort to find a design defect by inspections to find a defect by testing	1.4 : 8.5 Or 1:6	[18]
5.	Weller	Reports the time taken in testing to inspecting the rewritten code while converting code from C to Fortran	1:6	[19]
6.	Olson T. at Cardiac Pacemakers Incorporated	Provided an industry standard cost ratio for fixing a defect during design and test	1:10	[20]
7.	Madachy at Litton Data Systems	Provided estimates for staff hours being saved in systems testing for every inspection hour.	1:3.3	[24]
8.	Lionel Briand	Provided probability distribution parameters for the average effort to detect a defect during inspections and Testing	Minimum- 0.58 : 4.5 Mostlikely- 1.58 : 6 Maximum- 2.9:17	[18]

Summarizing the various literature studies, experiment results, historical data and assumption made by different studies it proves to be very hard to give a definite estimate for average cost to detect a defect in testing when no inspection occurs as different literature study provides different evaluations because of the differences in software processes they follow, severity levels of the defects, review procedures and several other factors. A brief summary of the literature for the cost ratio of average cost to detect a defect in inspection to the average cost to detect a defect in testing is provided in the table 3. The result in table 3 shows that the cost ratio varies from 1:2 to 1: 10. In order to find a precise and most appropriate value we have

taken the median of the above cost ratio values and the median was found to be 1:5.93. Therefore, our research, we selected the cost ratio for inspections to testing to be 1:6. In other words, the average cost to detect a defect in testing is 6 times the average cost to detect a defect in inspection.

4. STUDY DESIGN

Previous research in software inspections has validated their effectiveness in detecting defects present in the requirements and design document. However, there is a lack on research on the how much testing cost can be saved by performing inspections early in the software development process and how many inspectors should be involved during the inspection of early software products in order to maximize the savings by avoiding costly rework during the later stages.

Additionally, previous empirical studies of CR in software inspections have validated the ability of the CR estimators to accurately estimate the total fault count present in a software product based on the defects found during an inspection process. The common finding from these evaluation studies is that the CR models generally underestimate the true fault count, but accuracy improves with more number of inspectors (or captures). Furthermore, the studies have validated that ability of the CR estimators to accurately predict the need of a re-inspection. However, the CR research (including our previous research efforts) has been neglectful of the improvement in the costs effectiveness of an inspection by adding more number of inspectors when using the CR estimators to guide the re-inspection of software artifacts.

This paper provides results from the application of the Kusumoto cost metric on diverse inspection data sets in order to evaluate the cost effectiveness of software inspections with varying number of inspection team size when using the actual fault count and when relying on the CR estimates. In addition, we compared the values obtained from applying the Kusumoto cost-metric on software inspections when using the estimates from CR models, against the values obtained when using the actual fault count present in the document. This analysis was performed to gain insights into the ability of the CR estimators to help manage the inspection process. Also,

the results from this analysis provided insights into the relative performance of different CR estimators and provide recommendations on which estimators should be used when making re-inspection decisions during the software development.

4.1 Research Goal(s)

This study has two main goals. The first goal is to evaluate the effect of the number of inspectors on the cost effectiveness of an inspection process using the actual fault count of software documents.

Evaluate the cost effectiveness of software inspections

For the purpose of characterizing the impact of the number of inspectors

From the point of view of project managers and inspectors

In the context of requirements document with seeded and naturally occurring defects

Also, we want to analyze how the performance of the CR estimators (in the context of their ability to accurately predict the cost-effectiveness of an inspection process) improves when increasing the number of inspectors. Stated more formally, the second goal is to evaluate the effect of the number of inspectors on the cost effectiveness of software inspections when using the estimate of the total fault count relative to the results from Goal 1.

Analyze the capture-recapture estimators

For the purpose of evaluating the cost-effectiveness of software inspections and characterizing the improvement in their performance with increasing inspection team size

From the point of view of project managers and inspectors

In the context of a requirements document with unknown number of total fault count

4.2 Data Set

The CR uses data from six inspection data sets (with varying number of inspection team size). These data sets are described in Table 4 and were drawn from earlier inspection studies conducted at Microsoft Research, data set 1, and Mississippi State University (MSU), data sets 2-6. Each dataset was drawn from a study with other goals (i.e. the goal was not to evaluate CR models). The details and findings of the original studies are reported elsewhere [36]. Only the information that is relevant to the CR analysis appears in this section. The following three subsections describe the data sets grouped by similarity in artifacts, defects and inspectors.

Table 4. Capture-recapture Data Sets

Data Set	Artifact Name	Description	Number of Inspectors	First Inspection Defects	Total Defects
1	Loan Arranger Financial System	Grouping loans into bundles based on user-specified characteristics	73	NA	30
2	Starkville Theatre System	Management of ticket sales and seat assignments for the community theatre	8	30	55
3	Management of Apartment and Town properties	Managing apartment and town property, assignment of tenants, rent collection, and locating property by potential renters	8	46	105
4	Conference Management	Helping the conference chair to manage paper submission, notification of results to authors, and other related responsibilities	6	52	94
5	Conference Management	Same as Above	6	64	118
6	Data Warehouse Functional Requirements	The functional, and other (e.g., security, performance, interface) requirements of Data Warehouse	17	169	253

4.2.1 Data Set 1

Data Set 1 was drawn from an earlier inspection study that was conducted at Microsoft Research to investigate the impact of educational background on the effectiveness of an inspector.

Artifacts: The artifact inspected during this study was a generic (i.e., non-Microsoft) requirements document describing the requirements for the Loan Arranger financial system. The Loan Arranger system is responsible for grouping loans into bundles based on user-specified characteristics. These loan bundles are then sold to other financial institutions.

Defects: For use in previous studies, researchers seeded the document with thirty realistic defects. The authors of this paper were not involved in the defect seeding process The defects were seeded prior to the design of the CR study. Therefore the seeded defects should not provide be biased in any way for the goals of the current study.

Inspectors: The 73 inspectors who drawn from an internal training course taught by the Microsoft Engineering Excellence group. One of the main goals of the course was to teach participants about inspections and their use at Microsoft. The participants came from all major product groups within Microsoft. About 70% had bachelor's degrees with the other 30% having Master's degrees. On average, the participants had about two years of experience working in the field.

Inspection Process: First, the participants received training on the basic concepts involved in an inspection process. Then, the participants inspected the Loan Arranger requirements document. To guide their review of the document, the participants used a standard fault-checklist. During the inspection, each participant worked alone to identify and record as many defects as possible. The participants had 70 minutes to complete the inspection task. At the

conclusion of the inspection, the 73 defect lists were collected and processed. The processing involved determining which of the 30 seeded defects were found by each participant. This information is raw data input to the CR study described in the remainder of this paper.

4.2.2 Data Sets 2, 3, 4, and 5

Data sets 2, 3, 4, and 5 were drawn from earlier inspection studies conducted at MSU. The original goal of these studies was to investigate how the use of error information impacted requirements inspections.

Artifacts: The artifacts used in these studies were real requirement documents developed by senior-level undergraduate students\enrolled in the Software Engineering Senior Design Course at MSU during the Fall 2005 and Fall 2006 semesters. The 16 participants in the Fall 2005 semester were divided into two 8-person teams that developed the requirement documents for different systems (i.e., *Starkville Theatre System* and *Management of Apartment and Town Properties*) as shown in Table 4. Similarly, the 12 participants in Fall 2006 semester were divided into two 6-person teams that each developed their own requirement document for the *Conference Management* system. The course required student teams to interact with real customers to elicit and document requirements that they would later implement. So, even though the developers were students, the artifacts are realistic for a small project. A brief description of the requirement artifacts belonging to each of these four data sets is provided in Table 4.

Defects: The requirements documents used in data sets 2 through 5 included naturally occurring defects that were made by developers during the development of these artifacts. The defects were not seeded by the researchers.

Inspectors: The same developers who created each requirement document also inspected that requirements document. Table 4 shows the number of inspectors for each artifact.

Inspection Process: Each requirement document was inspected twice by the same inspectors. During the first inspection, the participants received training on the use of a fault checklist. Then, each inspector individually inspected the requirements using the fault checklist and logged any faults identified. After the first inspection, the inspectors met as a team to consolidate their faults into a team fault list. Then, the participants were trained on how to abstract errors from faults, how to classify the errors, and how to use the errors to re-inspect the requirements document. Then, each inspector re-inspected the requirements using the errors to find the additional faults. The requirements were not modified between inspections (i.e., the same requirements document was re-inspected).

The number of faults found during the first inspection and the total number of faults found after both inspections for each artifact is shown in the last two columns of Table 4. For example, for the requirements used in data set 2 (i.e., *Starkville Theatre System*), 8 inspectors found 30 distinct faults during the first inspection and found 25 additional faults during the second inspection for a total of 55 faults (as shown in the last column).

For the purpose of the CR evaluation in Goal 1 and Goal 2, only the data from the first inspection is used to calculate the CR estimates. Because the requirements were created by the study participants and the faults were naturally occurring and of unknown number (unlike in data set 1 where the faults were seeded), the data from the second inspection is used to calculate the total number of faults present in the document. We assume this total to represent all faults present in the document. Therefore, we use it to evaluate the performance of the CR models. Using the data only from first inspection as input to the CR models also helps control the variability of the inspection technique employed, since all four data sets use the fault checklist during the first inspection.

4.2.3 Data Sets 6

Data set 6 came from another inspection study conducted at MSU.

Artifacts: The requirements document inspected during this study was a natural language requirements specification document of a data warehouse system. The requirements were developed by professional developers at the Naval Oceanographic Office. The document was 30 pages long and included an overview (scope and purpose of the system), functional requirements, and other requirements (e.g., security, performance, interface).

Defects: Similar to Data sets 2-5, this data set also included naturally occurring defects made by developers during the creation of the requirements.

Inspectors: Eighteen graduate students enrolled in the Software Verification and Validation (V&V) course or the Empirical Software Engineering (ESE) course at MSU inspected the requirement document. These participants did not develop the requirements document, nor did they have access to any of the developers of the requirement document.

Inspection Process: Similar to the data sets 2 through 5, each participant inspected the requirements twice. Data from the first inspection (during which the participants individually used the fault checklist method to log defects) is used as input to the CR analysis and the number of unique faults found at the end of both inspections is assumed to be the total fault count.

4.3 Evaluation Procedure

This section describes the evaluation procedure relative to the research goals presented in Section 1.1. To summarize, we evaluated the a) effect of the *number of inspectors* on the cost-effectiveness of software inspection with the actual fault count known (i.e., Goal 1); and b) effect of the *number of inspectors* on the cost-effectiveness of software inspection using the CR estimates (i.e., Goal 2) and compared it against the cost-effectiveness values obtained in Goal 1.

The evaluation procedure for these analyses is discussed below.

A. *Goal 1 - Effect of the number of inspectors on the cost-effectiveness of software inspection using the actual fault count*: The cost-effectiveness of software inspections with increasing inspection team size was evaluated by: a) creating a fixed number of virtual inspection teams for each inspection team size (e.g., for Data Set 1 we varied the inspection team size from 1 to 73), and b) calculating the Kusumoto cost metric (measure of the cost-effectiveness) for each virtual inspection data set at all inspection team sizes. The process of creating virtual inspections and calculating the cost-effectiveness of the virtual inspections for all the six data sets is described in the following subsections.

   a. *Process of Creating Virtual Inspections*: This process consisted of randomly selecting the appropriate number of inspectors from the overall pool of inspectors. For example, to create the fifteen member inspection teams in data set 1, fifteen inspectors were randomly selected from a pool of seventy-three inspectors. Then, a matrix of the inspection data (containing 15 columns representing the inspectors and 30 rows representing the total defects) from these fifteen inspectors was created by keeping the fault count constant. Using this approach, 10 virtual inspection teams were created for each team size, i.e. 10 virtual inspection teams of size two, another 10 virtual inspection teams of size three, and so on. For Data Set 1, this process resulted in the creation of 10 inspection teams for each inspection team size (1- 72) and one team that combines all the seventy-three inspectors. Similar process for varying the inspection team size was performed for all the other data sets (i.e., Data Sets 2, 3, 4, 5, and 6) shown in Table 4.

      After creating the virtual inspections, we calculated the cost-effectiveness of all ten virtual inspections for each inspection team size as explained below.

b. *Calculating the Cost-Effectiveness of Virtual Inspections*: The following costs and savings were calculated to compute the cost-effectiveness of each virtual inspection (using the data from Step a):

i. ***Average cost to detect a defect in inspection (c_r)***: Adding all the defects found by all the inspectors, the average number of defects found by an inspector is calculated. From the available values of time taken by each inspector and the average number of defects found by an inspector, c_r is calculated using the following equation:

$c_r = C_r / D_r$ (i.e; Cost spent on inspection / Total defects found during an inspection)

The "**C_r - *Inspection Cost***", is calculated by adding the total time spent by all the inspectors employed during the inspection. For Data set 1, inspectors were given seventy-minutes to complete the inspection, whereas for Data sets 2-6, the inspectors were not asked to complete the inspection in a given time period. Instead, they used the fault reporting forms to log the start and end times of the inspection, the time they found each fault, and any breaks they took. From these fault forms, we calculated the total time spent by each inspector to perform the inspection.

The "**D_r**", is the total number of unique faults found by all the inspectors during an inspection cycle.

ii. ***Virtual Testing Cost (C_{vt})***: Virtual testing cost is the testing cost that would have been spent if no inspection were performed. It is calculated as the product of the average cost to detect a fault in testing (i.e., c_t) and the total number of faults in the product existing before inspection (i.e., D_{total}). That is, ***C_{vt}*** **$= c_t * D_{total}$**

The "***c_t - Average cost to detect a defect in testing",*** is calculated as 6 times of average cost to detect a defect during the requirements inspection (**c_r**) (based on the

findings from the literature survey described in section 3). But the average cost to detect a defect during inspection (**c_r**) varies with the inspection team size as it is dependent on the time taken by inspection and defects found ($c_r = C_r / D_r$). Since Testing process is independent of the inspection and inspection team size, and considering all the defects to be of the same severity, we calculated the average cost to detect a defect in inspection (**c_r**) for an individual inspector and multiplied it by 6 to get the average cost to detect a defect during testing (**c_t**) (based on section 3). The average cost to detect a defect in testing (**c_t**) thus calculated.is used for all the evaluations regardless of the inspection team size.

The "**D_{total}** - ***Total Faults Count***", is the total number of faults present in a document. The artifact used in data set 1 was seeded with 30 realistic faults, which is the total defect count. The artifacts used in data sets 2-6 were not seeded and contained naturally occurring faults. For that reason, the total number of unique defects that were found in these artifacts at the end of two inspection cycles is assumed to be actual fault count for the purpose of the evaluation in this research.

iii. ***Testing Cost saved from an Inspection (***ΔC_t): The testing cost saved by spending cost during the inspection process is calculated as the product of number of defects found during an inspection (D_r) and the average cost to find a defect during the testing (c_t). That is, $\Delta \boldsymbol{C_t} = \mathbf{Dr} * \mathbf{c_t}$.

The difference in the testing cost saved by performing an inspection and the cost spent during an inspection provides the reduction of the total costs (i.e., $\Delta C_t - C_r$).

Kusumoto Metric (M_k) then, calculates the cost-effectiveness of software inspections using the calculations performed in steps I through III as shown below:

$$M_k = \frac{\text{Reduction of total costs to detect all faults (i.e., } \Delta C_t - C_r)}{\text{Virtual testing cost (i.e., } C_{vt})} \quad \text{..... Eq (4.3)}$$

By computing the cost (in hours) of inspection, cost saved from inspection and virtual testing cost in similar fashion as described above; the kusumoto metric (M_k) can thus be determined for each of the 10 virtual inspections at different inspection team sizes for all the six data sets. As an example, the calculations of the cost-effectiveness of 10 virtual inspections on Data set 1 for an inspection team size of 15 inspectors is shown in Table 5.

Table 5. Calculation of Kusumoto Metric (M_k) for Goal 1

Calculations based on "known" fault count

Virtual Inspection #	Total Defect Count (D_{total})	Defects Found (D_f)	Cost Of Inspection (C_r) (mins)	Avg. Cost to detect a defect in Testing (c_t) (mins)	Testing Cost Saved by Reviews ($\Delta C_t = c_t * D_r$)	Virtual Testing Cost C_{vt} (mins)	Kusumoto Metric $M_k = (\Delta C_t - C_r)/C_{vt}$
1	30	21	1050	105	2205	3150	0.366666667
2	30	21	1050	105	2205	3150	0.366666667
3	30	22	1050	105	2310	3150	0.4
4	30	22	1050	105	2310	3150	0.4
5	30	24	1050	105	2520	3150	0.466666667
6	30	21	1050	105	2205	3150	0.366666667
7	30	20	1050	105	2100	3150	0.333333333
8	30	20	1050	105	2100	3150	0.333333333
9	30	25	1050	105	2625	3150	0.5
10	30	20	1050	105	2100	3150	0.333333333

Similar process was followed for deriving the M_k values for all the ten virtual inspection by varying the inspection team size for all the six data sets (i.e., 1-73 for data set 1, 1-8 for data sets 2 and 3, 1-6 for data sets 4 and 5, and 1-17 for data set 6).

B. Goal 2 - Effect of the number of inspectors on the cost-effectiveness of software inspection using the CR estimates: We followed the same process (as used in Goal 1) to evaluate the effect of the number of inspectors on the cost-effectiveness of software inspections assuming we don't know the actual fault count prior to the inspection (which is a more realistic situation). The same virtual inspections created in Goal 1 were used in this

analysis. The only difference is that for each virtual inspection data set, rather than using the actual defect count (D_{total}) (as used in Goal 1), we used the capture recapture estimators to estimate the defect count to calculate the cost-effectiveness of that virtual inspection. In order to estimate the defect count to calculate the Kusumoto Metric (M_k) for each virtual inspection in Goal 2, automated tools namely CAPTURE [23] and CARE-2 [35] was used in this study to calculate the estimates of total defect count for each virtual inspection. Each of these tools has different set of estimators. So, using these tools, the virtual inspections for each inspection team size (and for all the six data sets) were used as input to the capture-recapture estimators to produce estimates of the total number of defects. For each virtual inspection, using the estimate of the total fault count, the virtual testing cost (in hours) is then calculated. The kusumoto metric (M_k) values are then determined for virtual inspections of each team size and each estimator combination using the same process as described above. The calculations of the cost-effectiveness of same 10 virtual inspections (as shown in Table 5) of Data set 1 for an inspection team size of 15 inspectors and using estimates from a particular CR estimator (M_h-SC) is shown in Table 6. The columns whose values are different in Goal 2

Table 6. Calculation of Kusumoto Metric (M_k) for Goal 2

Calculations based on "**estimated**" fault count

Virtual Inspection #	Estimated Defect Count (D_{total})	Defects Found (D_r)	Cost Of Inspection (C_r) (mins)	Avg. Cost to detect a defect in Testing (c_t) (mins)	Testing Cost Saved by Reviews ($\Delta C_t = c_t * D_r$)	Virtual Testing Cost C_{vt} (mins)	Kusumoto Metric $M_k = (\Delta C_t - C_r)/C_{vt}$
1	25.3	21	1050	105	2205	2656.5	0.434782609
2	26.7	21	1050	105	2205	2803.5	0.411985019
3	27.3	22	1050	105	2310	2866.5	0.43956044
4	27.5	22	1050	105	2310	2887.5	0.436363636
5	32.8	24	1050	105	2520	3444	0.426829268
6	24.3	21	1050	105	2205	2551.5	0.452674897
7	23.7	20	1050	105	2100	2488.5	0.421940928
8	23.5	20	1050	105	2100	2467.5	0.425531915
9	35.7	25	1050	105	2625	3748.5	0.420168067
10	23.4	20	1050	105	2100	2457	0.427350427

analysis from the values calculated for Goal 1 analysis is shaded in Table 6.

4.4 Evaluation Criterion

For each inspection team size (i.e.,, 1-73 for data set 1, 1-8 for data sets 2 and 3, 1-6 for data sets 4 and 5, and 1-17 for data set 6), ten possible virtual inspections were used to evaluate the effect of the inspection team size on the cost-effectiveness of software inspections for Goal 1 and Goal 2 separately.

For Goal 1, the Kusumoto Metric (M_k) was computed for all the ten virtual inspections for each inspection team size using the actual fault count. These ten values were then used to calculate the median and the variability in the M_k values for each inspection team size and for all the data sets. This scenario is illustrated in Figure 3 with example of an inspection team size of two inspectors.

The kusumoto metric (M_k) value ranges from -1 to +1. The M_k value of 1 means the most

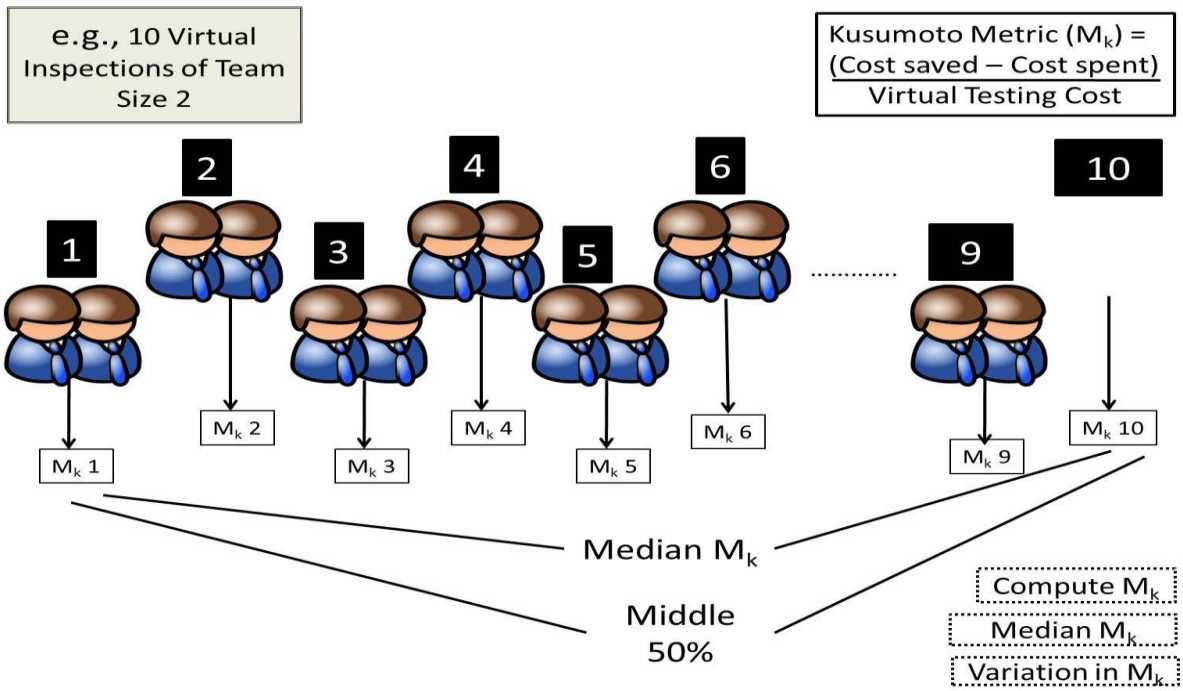


Figure 3. Evaluation Criteria

cost effective inspection process. A positive value of the kusumoto metric value is an indication that it is a cost effective process (i.e., cost (in hours) saved by performing the inspection outweighs the costs (in hours) spent during the inspection). A negative value of kusumoto metric is an indication of a cost ineffective process, as a negative value of M_k means the cost invested in inspection is greater than the cost saved from inspections. An M_k value of 0 means that the cost saved from inspections is equal to the cost invested in inspections.

For Goal 2, the estimates from the CR estimators were used to compute the Kusumoto Metric (M_k) as opposed to the actual fault count. Similar to the evaluation criteria for Goal 1, we calculated the median and the variability in the M_k values for each inspection team size and for all the data sets. In addition, the CR estimators are then evaluated on their performance using three parameters: accuracy (bias), precision (variability), and failure rate by comparing the estimates from the CR estimators with the results from Goal 1 (i.e., when using the actual fault count):

- The **accuracy (bias)** is measured as the relative error (R.E) in the ability of an estimator to calculate the cost-effectiveness of an estimate. It is calculated relative to the M_k values based on the actual fault count :

$$\textit{Relative error} = \frac{(M_k \textit{ based on the estimated fault count} - M_k \textit{ based on the actual fault count})}{M_k \textit{ based on the actual fault count}} \quad \text{Eq (4.4)}$$

A R.E of zero means absolute accuracy (i.e., M_k valued based on estimated fault count is same as the M_k value based on the actual fault count), a positive R.E. means an underestimation, and a negative R.E means an overestimation. The accuracy of the estimator is measured by calculating the median relative error for each inspection team size. Furthermore, the error in the estimated cost-effectiveness is calculated relative to each artifact to allow for combination of the results from all the artifacts.

According to Eick et al. and Briand et al., the accuracy of an estimate is considered satisfactory when the R.E. is within +/- 20% of the actual value [3, 22]. In this paper, we evaluated the accuracy of the estimators at different levels of R.E (e.g., +/- 20%, +/- 10%, +/- 5%, 0% etc.).

- The **precision** of an estimator is measured by calculating the variability of the R.E. estimates for each input size (e.g., 1-73). R.E variability around the central tendency i.e. (median value) is measured using the inter quartile range of the 25^{th} percentile to the 75^{th} percentile.
- The **failure rate** of an estimator is defined as the number of time an estimator fails to produce any result. Because each estimator makes different assumptions about the data and they all operate on the same data matrix, some estimators can fail if the actual data fails to meet some of its basic assumptions.

5. ANALYSIS AND RESULTS

This section analyzes of the cost-effectiveness of software inspections and is organized around the three research goals described in Section 1.1. In order to reduce duplication, we have grouped the results from all the six data sets for each research goal. Section 5.1 evaluates the cost-effectiveness of software inspection as a factor of increasing inspection team size when considering the actual fault count. Section 5.2 evaluates the performance of CR estimators and the relative errors in their ability to accurately predict the cost-effectiveness of software inspections with increasing inspection team size.

5.1 Effect of Inspection Team Size on the Cost-Effectiveness of Software Inspections Using Actual Fault Count

Our first research goal deals with evaluating the effect of inspection team size on the cost-effectiveness of the inspection process when the actual defect count of an artifact is known beforehand. To provide an overview of the results, Figure 4 shows the median M_k values (computed using 10 virtual inspections data sets) across all team sizes for Data Set 1 (with inspection team size varying from one through seventy-three) that was seeded with 30 realistic defects prior to the inspection.

The major results in Figure 4 are summarized as follows:

a) There is a consistent improvement in the cost-effectiveness (i.e., median M_k) of an inspection process with increase in the inspection team size up to 11 inspectors;

b) The inspection process was most cost-effective with inspection team size of 11 inspectors (with a median M_k value of 0.46). Beyond this point (i.e., team size of 12 and more), adding more number of inspectors did not increase the cost-effectiveness of software inspection process any further;

c) The inspection process is still cost-effective (i.e., a positive median M_k value) for inspection team size varying from 12 up to 37 inspectors. However, there is a consistent decrease in the level of cost-effectiveness for inspection team size varying from 12 up to 37 inspectors.

d) For inspection process involving 39 and more number of inspectors, the cost (in man hours) spent on the inspection process outweighs the testing cost (in hours) saved from performing it. Furthermore, there is a consistent decrease in the median M_k value (going from -0.03 for an inspection with 39 inspectors to -0.62 for an inspection with 73 inspectors) with the increase in the number of inspectors.

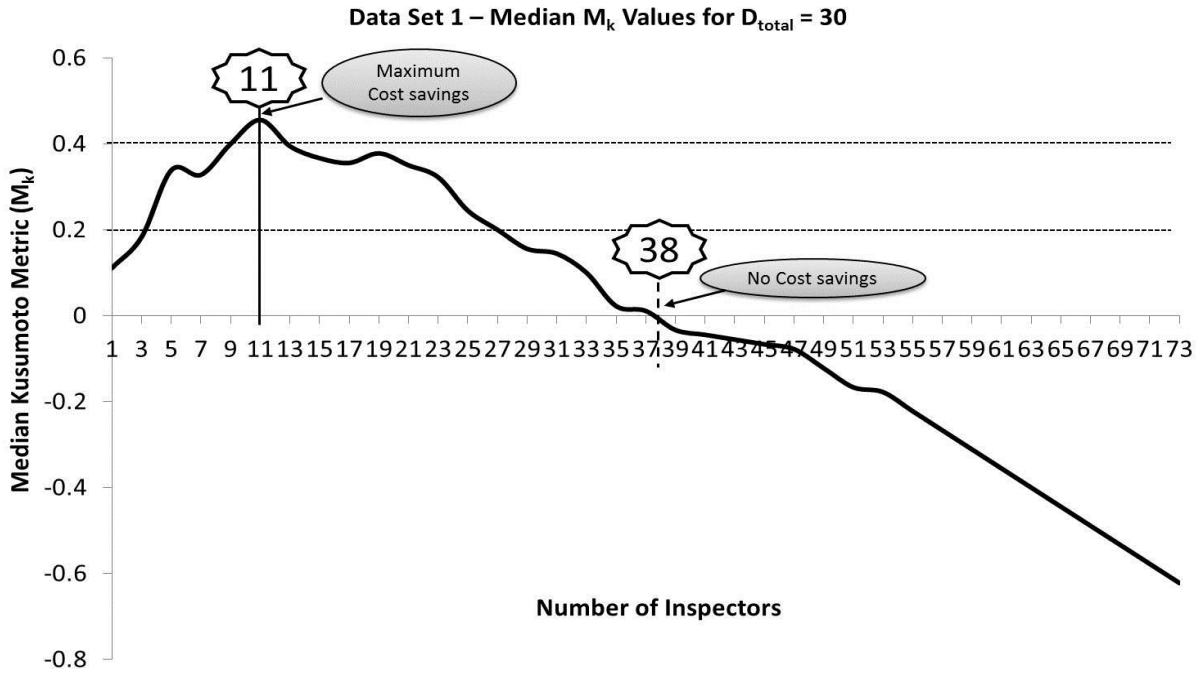


Figure 4. Median Mk Values for Data Set 1 at all Inspection Team Sizes

Therefore, based on the results from Figure 4, an inspection process of 11 inspectors was most cost-effective (i.e., median M_k value of 0.46), and an inspection process of 39 and more

inspectors were not cost-effective at all (i.e., a negative median M_k value). Therefore, adding more number of inspectors only increases the cost-effectiveness of inspection process (and consequently the testing cost savings) to a certain point beyond which additional improvement in software quality comes at the expense of cost (in hours) spent in adding more number of inspectors.

While, the above results are based on the median M_k values, we also examined the variability in the M_k values (across an array of 10 values) at each inspection team size to gain additional insights. An approach for combining the analysis of median and variability of Kusumoto metric (i.e., M_k values) is to calculate the three different values for each inspection team size (from 1 -73). These following three values are calculated from an array of 10 values:

a) The *median* M_k value (50^{th} percentile),

b) The *seventh largest* value (75^{th} percentile), and

c) The *third largest* value (25^{th} percentile).

Together b) and c) define the interquartile range (IQR) and is essentially the range of the middle 50% of the M_k values. Figure 5 shows these three values with the median M_k value appearing between the upper (75^{th} percentile) and lower bound (25^{th} percentile) at all inspection team sizes. To quantify these results, we analyzed the *median*, the *seventh largest*, and the *third largest* M_k values (as shown in Figure 5) to determine how many inspectors are required to achieve varying levels of cost-effectiveness in the inspection process. The inspector count for varying levels of cost-effectiveness was determined so that beyond that point, all the three M_k values (median, 75^{th} percentile, and 25^{th} percentile) are greater than the given cost saving percentage.

Figure 5. Median and Variance in the Mk Values for Data Set 1

Table 7 shows the number of inspectors required to achieve cost savings greater than 40%, 30%, 20%, 10% and less than 0%. For example, an inspection process of 7 to 22 inspectors are required to achieve all the three M_k values greater than 0.3 (i.e., cost savings greater than 30%). Similarly, 5 -26 inspectors achieved cost savings greater than 20% (i.e., all three M_k values are greater than 0.2).

Table 7. Cut-off Points for Varying Levels of Cost Savings for Data Set 1

Cost Savings	> 40%	>35%	>30%	>25%	>20%	>15%	>10%	>5%	>0%	< 0%
Inspector Count	Never	9 to 17	7 to 22	5 to 23	5 to 26	5 to 29	3 to 31	1 to 33	1 to 35	37 to 73

Based on the results shown in Figure 5 and the inspector count shown in Table 7, some general observations are as follows:

a) Performing software inspections of early software products helps save testing cost (in hours) that would be otherwise spent to find and fix problems later in the development process. Performing an inspection with even a single inspector saved some testing cost (i.e., value of Kusumoto Metric is greater than zero).

b) Adding more number of inspectors increases the testing cost savings (and the cost-effectiveness) upto a certain point. Formally stated,

    a. A minimum of 3 inspectors and maximum of 31 inspectors were required to achieve cost savings more than 10% (i.e., all the three M_k values greater than 0.10);

    b. A minimum of 5 inspectors and maximum of 29 inspectors were required to achieve cost savings greater than 15% (i.e., M_k value greater than 0.15).

    c. A minimum of 7 inspectors and 9 inspectors were required to achieve cost savings greater than 30% and 35% respectively.

c) Adding more number of inspectors yields positive cost savings up to 36 inspectors. The inspection data showed that an inspection process of 36 inspectors found an average of 84% of total defects (across 10 virtual inspections) and the cost (in hours) spent by adding more number of inspectors is more than the cost saved by them. Therefore, performing an inspection with 37 and more number of inspectors does not save any testing cost (in hours) (i.e., a negative value of Kusumoto metric).

Overall, these results showed that performing software inspections with even few numbers inspectors (e.g., 9 to 17) save testing costs in excess of 35%. The inspector count for varying level of cost savings is dependent on the cost-spent during the inspection and the number

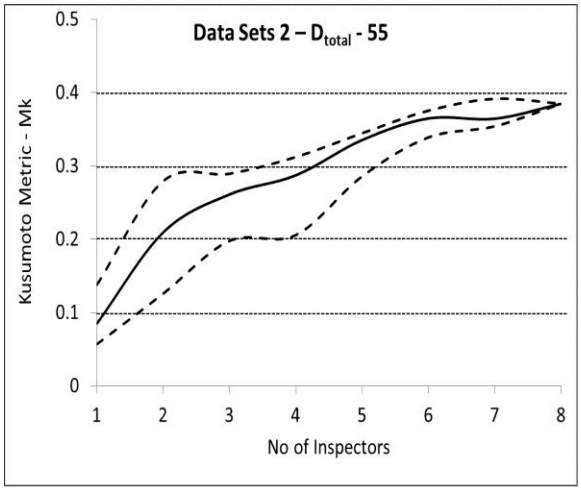
of additional defects found by employing additional inspectors. The project managers can employ these metrics in their organizations to understand the cost-effectiveness of the software reviews and the number of inspectors to be employed in the inspection process.

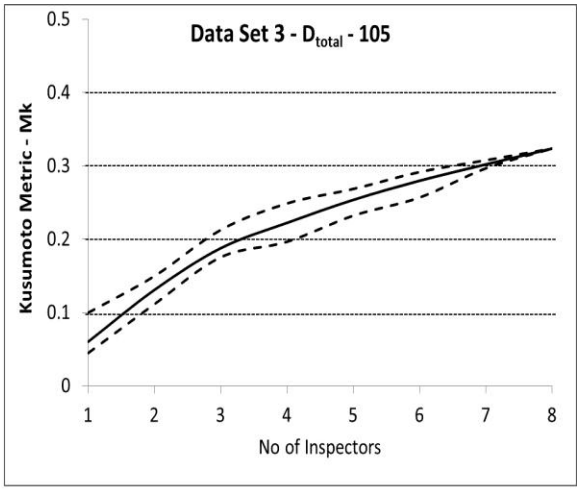
While the above results demonstrate the application of *Kusumoto Metric* on inspection data sets, the data set used in the above analysis (i.e., Data Set 1) was seeded with defects and the inspectors were required to complete inspection within certain time period. To understand these results better, we replicated the same analysis with the data sets 2-6 (shown in Table 4) that contain naturally occurring defects and the inspectors used varying amount of time to complete the inspection process.

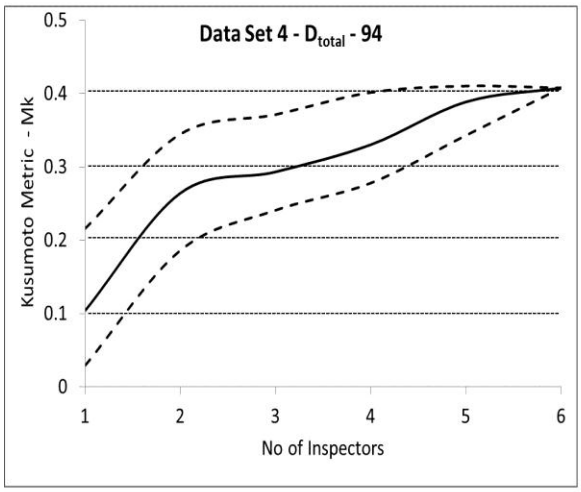
We performed the same analysis (as used for Data Set 1) to evaluate the effect of the inspection team size on the cost-effectiveness of inspection process for data sets 2, 3, 4, 5 and 6. Figure 6 shows the median, 75^{th} percentile, and 25^{th} percentile of the M_k values (computed using 10 virtual inspection data sets) for each inspection team size for all the five data sets. As mentioned earlier, we only used the data from the first inspection to calculate the cost-effectiveness at each inspection team size. The number of defects found at the end of first inspection for each data set is shown in Table 4. Also, we used the total number of defects found at the end of two inspection cycles (also shown in Table 4) as actual defect count (D_{total}) to evaluate the cost-effectiveness of software inspections.

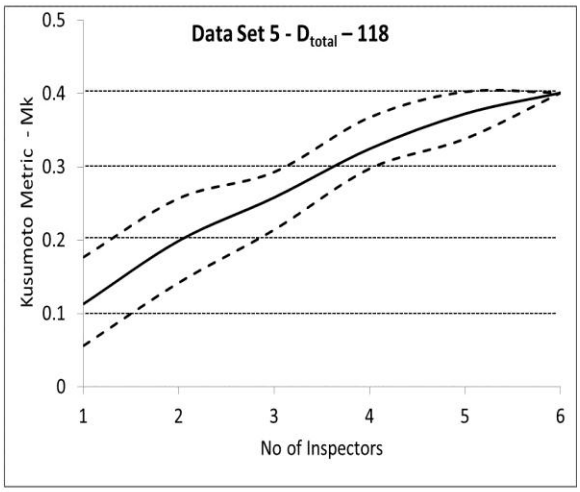
From Figure 6, the following are some of the common results across all the five data sets:

a) A minimum of one inspector is needed to achieve a positive cost savings by performing an inspection process (i.e., M_k value greater than zero). This is consistent with the results from Data Set 1.









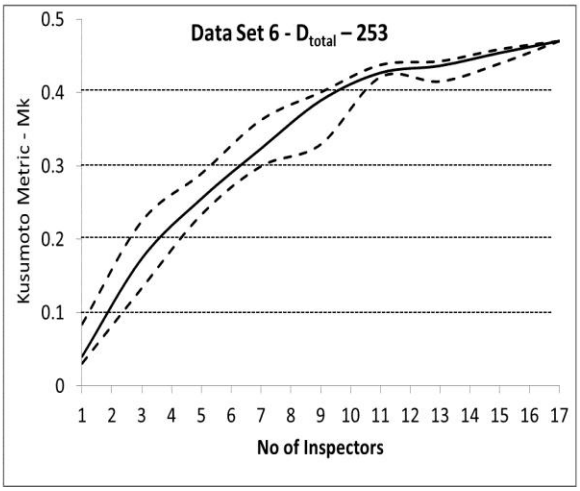


Figure 6. Median M_k Values for Data Sets 2, 3, 4, 5 and 6.

b) The inspection cost-effectiveness increases with an increase in the number of inspectors in a lin ear fashion. All the three M_k values (Median, 25^{th} and 75^{th} percentile) increases with an increase in inspection team size.

c) Unlike the result from Data Set 1 (that had a maximum cost savings with inspection team of 11 inspectors), the cost-effectiveness of software inspection in Data set 6 increases beyond inspection team of 11 inspectors. This could be due to the fact that the number of faults captured by inspection team of 11 inspectors in Data Set 6 is only 56% of the actual faults (and more number of defects are found by adding more number of inspectors) as compared to the 70% of the total faults that had been captured by an inspection team of 11 inspectors in Data Set 1.

d) For Data Sets 2 and 3 (that varied the inspection team size from 1 to 8), the cost effectiveness at the end of first inspection cycle (i.e., with inspection team of 8 inspectors) is similar for Data Set 2 (M_k value of 0.38) and Data Set 3 (M_k value of 0.33). Similarly, for Data Sets 4 and 5 (that varied the inspection team size from 1 to 6), the cost-effectiveness at the end of first inspection cycle (i.e., with inspection team of 6 inspectors) is similar for Data Set 4 (M_k value of 0.41) and Data Set 5 (M_k value of 0.40).

Also, similar to Data Set 1, we analyzed the *median*, the *seventh largest*, and the *third largest* M_k values for Data sets 2, 3, 4, 5 and 6 (as shown in Figure 6) to determine the minimum number of inspectors required to achieve varying levels of cost-effectiveness in the inspection process. Table 8 shows the number of inspectors required to achieve cost savings greater than 50%, 40%, 30%, 20%, 10%, 0% and less than 0% for all the six data sets.

Table 8. Cut-off Points for Varying Levels of Cost Savings for Data Sets 1-6

	Data Set #					
Cost-Effectiveness Level	***1 (1-73)***	***2 (1-8)***	***3 (1-8)***	***4 (1-6)***	***5 (1-6)***	***6 (1-17)***
> 50%	Never	Never	Never	Never	Never	Never
> 40%	Never	Never	Never	6	6	11
> 30%	7	6	7	5	5	8
> 20%	5	4	4	3	3	5
> 10%	3	2	2	2	2	3
> 0%	1	1	1	1	1	1
< 0%	37-73	Never	Never	Never	Never	Never

Based on the results shown in Figure 6 and the inspector count shown in Table 8, the Cost savings up to 40% can be achieved with as few as six inspectors (as in Data Sets 4 and 5) and with maximum of eleven inspectors (as in Data Set 6). Also, a minimum of 2 to 3 inspectors, and a minimum of 3 to 5 inspectors are required to achieve cost-savings greater than 10% and cost-savings greater than 20% respectively.

An interesting observation is that the cost-effectiveness improves linearly if a large percentage of faults remain to be detected. Project managers can vary the inspection team size and apply the Kusumoto metric at the end of the inspection process to gain insights into whether or not adding more number of inspectors to perform an inspection would save further testing cost (in hours). The project managers can also use these results prior to performing an inspection to decide the minimum number of inspectors required to achieve particular level of cost-savings required while building the software products in their organizations.

5.2 Effect of Inspection Team Size on the Cost-Effectiveness of Software Inspections Using Capture Recapture Estimators

The cost-effectiveness results presented in Section 5.1 were calculated using the actual fault count of software documents. In real settings, we don't know the actual fault count prior to performing the inspection. To that end, an accurate estimate of the actual fault count (D_{total}) can be used to calculate the cost-effectiveness of inspection process. Our prior research in software inspections has evaluated the use of the Capture Recapture (CR) models to estimate the fault count in a software product using the number of unique defects and the overlap of defects found by inspectors. In this section, we compare the cost-effectiveness results based on the CR estimates (from all the CR estimators) at all inspection team sizes against the results based on the actual fault count (as shown in Section 5.1). We also analyze the relative errors in the M_k values based on the estimated fault count against the M_k values based on the actual fault count.

To provide an overview of the results, Figure 7 shows the median M_k values (computed across 10 virtual inspections) at all inspection team sizes (varying from 1 to 73 for Data Set 1) using the estimated total fault count from all the CR estimators as shown by solid gray lines. Figure 7 also shows the median M_k values at all inspection team sizes using the actual fault count of 30 defects that were seeded into the document prior to the inspection (as shown by dotted line and highlighted). The actual median M_k values for data set 1 are also shown in Figure 4.

The general observations from Figure 7 are discussed as follows:

a) For all the CR estimators with a small number of inspectors, there is a huge difference in the estimated median M_k values as compared to the actual median M_k values. This is because the CR estimators generally underestimate the actual fault count with a small number of inspectors which reduces the true virtual testing cost (in hours) (i.e., C_{vt}, the

demoniator of Kusumoto metric formula), and thereby returns a higher value of the test cost-savings.

b) Also, for all the CR estimators, the estimated median M_k values are closer to the actual Mk values with increase in the number of inspectors. This is because the estimation accuracy of the CR estimators improve (i.e., the estimated fault count is closer to the actual fault count) with the increase in the number of inspectors.

c) The median M_k values for some of the CR estimators is closer to the actual M_k values with fewer number of inspectors as compared to the other CR estimators. This is becase some CR estimators (i.e., estimators belonging to the M_h and M_{th} models) are more accurate with fewer number of inspectors than the other CR estimators (i.e., estimators belonging to M_o and M_t models).

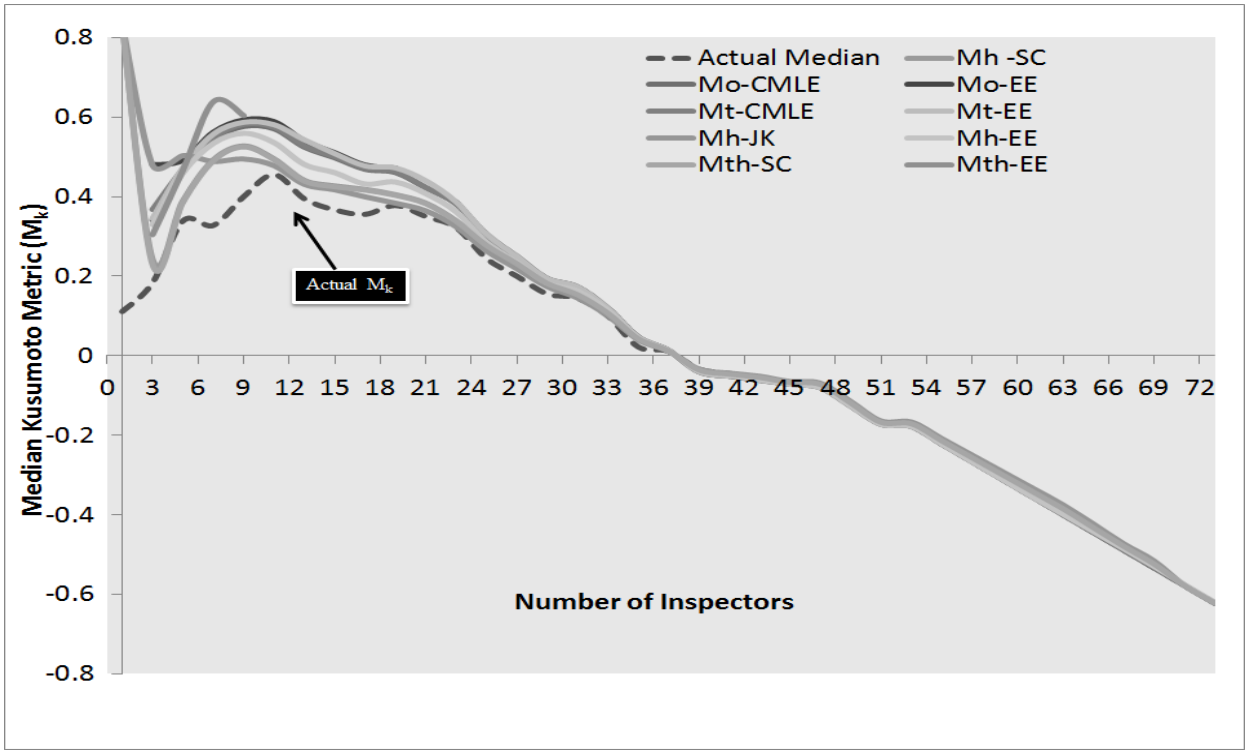


Figure 7. Median M_k for all the Capture Recapture Estimators

To quantify the results provided in Figure 7, Figure 8 shows the relative error (R.E) percentage in the median M_k values produced by each CR estimator at each inspection team size. The dashed lines in Figure 8 show the region of +/- 20% within which the estimation results are considered satisfactory [3,22]. Also, as mentioned earlier in Section 4.4 (under evaluation criteria), the following formula was used to calculate the error in the M_k values based on the CR estimates relative to the M_k values based on the actual fault count:

$$\textit{Relative error} = \frac{(M_k \textit{ based on the estimated fault count} - M_k \textit{ based on the actual fault count})}{M_k \textit{ based on the actual fault count}} \quad \text{Eq (5.2)}$$

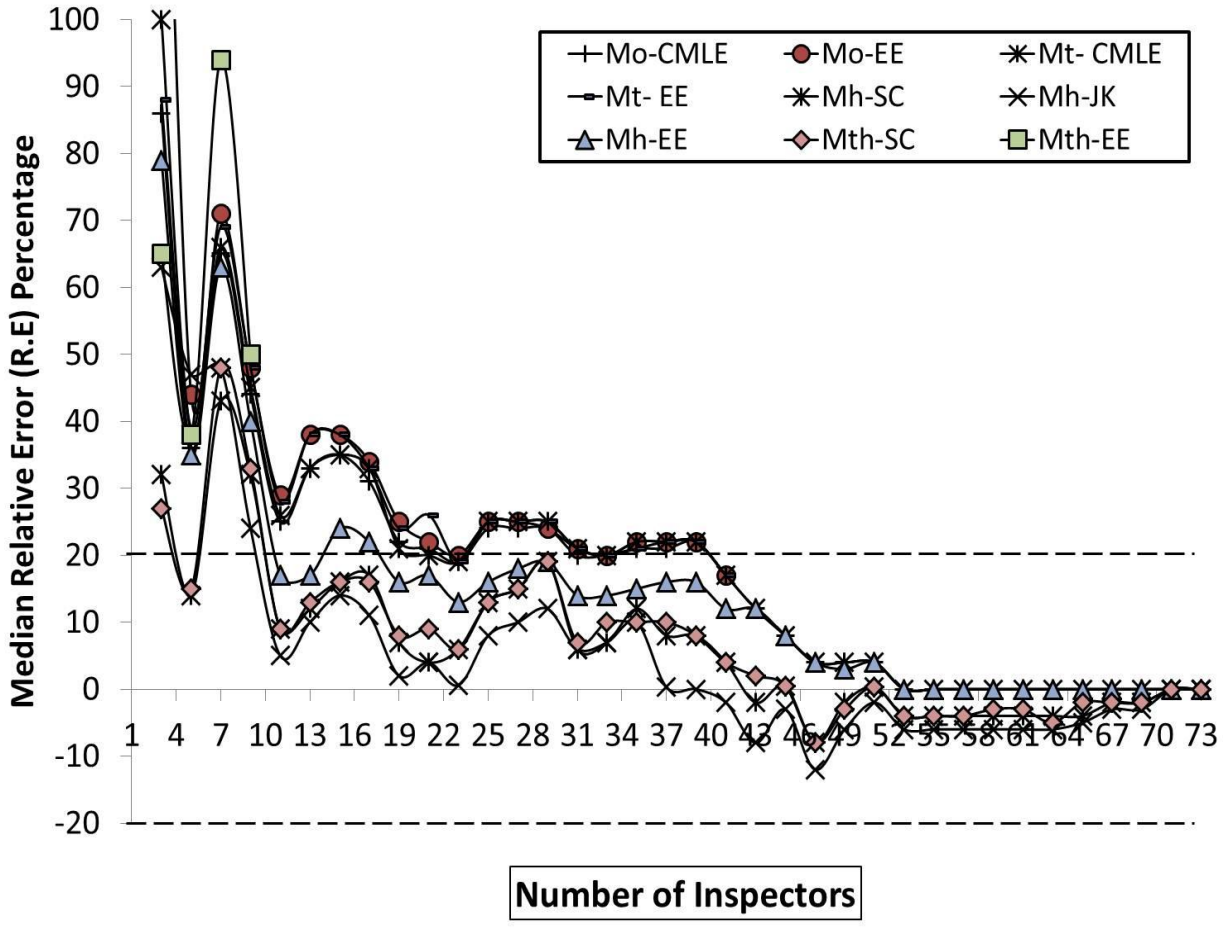


Figure 8. Relative Error in Median Mk for all the CR Estimators for Data Set 1

The major observations from Figure 8 are discussed as follows:

a) The relative error in the median M_k values is greater than 20% for all the CR estimators for inspection team size of 1 upto 9 inspectors. Therefore, the CR estimators can not be relied upon to determine the cost-effectiveness of inspection process with less than ten inspectors;

b) In general, the CR estimators belonging to M_h and M_{th} models improve faster, and consequently the relative error in the median M_k values obtained from these CR estimators is less than 20% with fewer number of inspectors (11 inspectors) than the CR estimators belonging to Mo and Mt models (41 inspectors).

c) The same trend is true for relative error in the M_k values at level +/- 10%, where the SC estimators for M_h and M_{th} models are better than all the remaining estimators;

d) The EE estimators for all models (M_o-EE, M_t-EE, M_h-EE, and M_{th}-EE) fail even for the larger number of inspectors. Among all the EE estimators, M_h-EE exhibits the lowest failure rate. Therefore, we don't recommend any of these EE estimators.

To better understand the relative performance for different CR estimators with respect to varying level of relative error (R.E) in the M_k values, Table 9 shows the number of inspectors required by each estimator to obtain median M_k within 0%, +/- 5%, and so on up to +/- 40% of the actual M_k value. The inspector count for each R.E percentage shown in Table 9 is the minimum number of inspectors for which the estimate falls within the given R.E. range and never goes outside the range as the number of inspectors increases. For example, the number 19 in the M_o-CMLE row in the +/- 30% column means that for all inspection team sizes greater-than or equal to 19, the median R.E. is never greater than + 30% and for at least one inspection team

size less-than 19, the median R.E. exceeds + 30%. The estimators that achieved satisfactory estimates with fewest number of inspectors is highlighted in Table 9 and is discussed as follows:

a) Across all CR estimators, 11 to 41 inspectors are required to achieve a satisfactory estimate of the cost-effectiveness of an inspection process (i.e., with R.E. of +/- 20%);

b) The estimators M_h–SC, M_h-JK, the M_{th}– SC require fewest inspectors to obtain an estimate within 20% R.E. (11 inspectors) compared with the estimators for the M_o and M_t models (19 or 41 inspectors);

c) Only the SC estimators (i.e., M_h-SC and M_{th}-SC) achieved results within -10% and at 0% relative error with fewest number of inspectors;

d) EE estimators for all models (M_o-EE, M_t-EE, M_h-EE, and M_{th}-EE) fail even for the larger number of inspectors.

Table 9. Number of Inspectors Required to Achieve Different Levels of Relative Error in Median M_k Values for Data Set 1

Estimators	+/- 40%	+/- 30%	+/- 20%	+/- 10%	0%
Mo-CMLE	11	19	41	45	53
Mo-EE	11	19	41	*Failed*	
Mt-CMLE	11	19	41	45	53
Mt-EE	11	19	41	*Failed*	
Mh-SC	**9**	**11**	**11**	**31**	**43**
Mh-JK	**9**	**11**	**11**	49	71
Mh-EE	11	19	19	45	53
Mth-SC	**9**	**11**	**11**	**31**	**45**
Mth-EE	*Failed*				

Based on these results, the accuracy of the SC estimators (for the M_h and M_{th} models) are most positively affected by increasing the inspection team size compared with the other CR

estimators. Therefore, based on the median R.E values, SC are the best estimators to be used when evaluating the cost-effectiveness of software inspections.

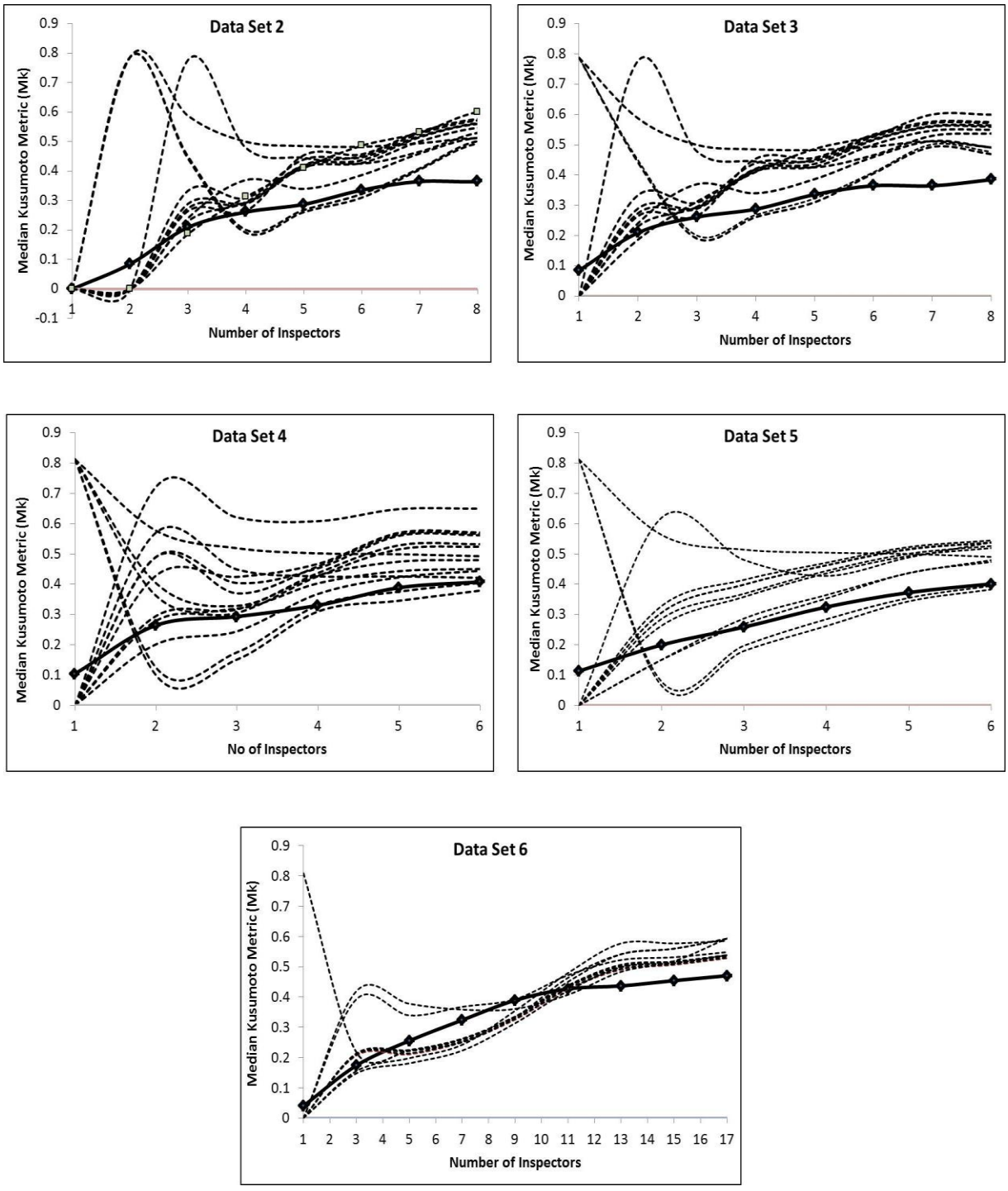


Figure 9. Median M_k Values for CR Estimators vs. Actual M_k Values for Data Sets 2 - 6.

We replicated the same analysis process as described above to evaluate the cost-effectiveness of software inspection using the artifacts that contain naturally occurring defects. To that end, we calculated the relative error in the median M_k values for all CR estimators for the data sets 2, 3, 4, 5, and 6 against the actual M_k values (shown in Figure 6).

Figure 9 shows the median M_k for each CR estimator across all team sizes for Data Sets 2 and 3 (with inspectors varying from 1 to 8), for Data Sets 4 and 5 (with inspectors varying from 1 to 6), and for Data Set 6 (with inspectors varying from 1 to 17). The median M_k values produced using the CR estimators (as shown in dotted lines) is compared against the actual median M_k values using the actual fault count (as shown with a solid line).

This figure illustrates some interesting observations as listed follows:

a) Across all the six data sets, there is a big difference in the estimated median M_k values and the actual median M_k values for small number of inspectors. This difference becomes smaller with the increase in the inspection team size. This is consistent with the results from Data Set 1 (as shown in Figure 7).

b) The SC estimators showed lower relative error in the median M_k values as compared to the other estimators even with small number of inspectors. This is also consistent with the earlier results.

Similar to analysis performed for Data Set 1, Figure 10 shows the relative error (R.E) percentage in the median M_k values produced by each CR estimator at each inspection team size for all the six data sets. The relative error in the cost-effectiveness is calculated relative to the M_k values based on the actual fault count. The major observations from Figure 10 are as follows:

a) Regarding Data Set 2, the results confirm the earlier findings that the CR estimators severely underestimate the actual fault count with eight inspectors or less. The median M_k estimates for the CR estimators never reached within the 20% R.E. range.

b) Regarding Data Set 3, the estimation results are same as Data Set 2 results. The only exception is that the SC estimators for the M_h and M_{th} models achieved median M_k

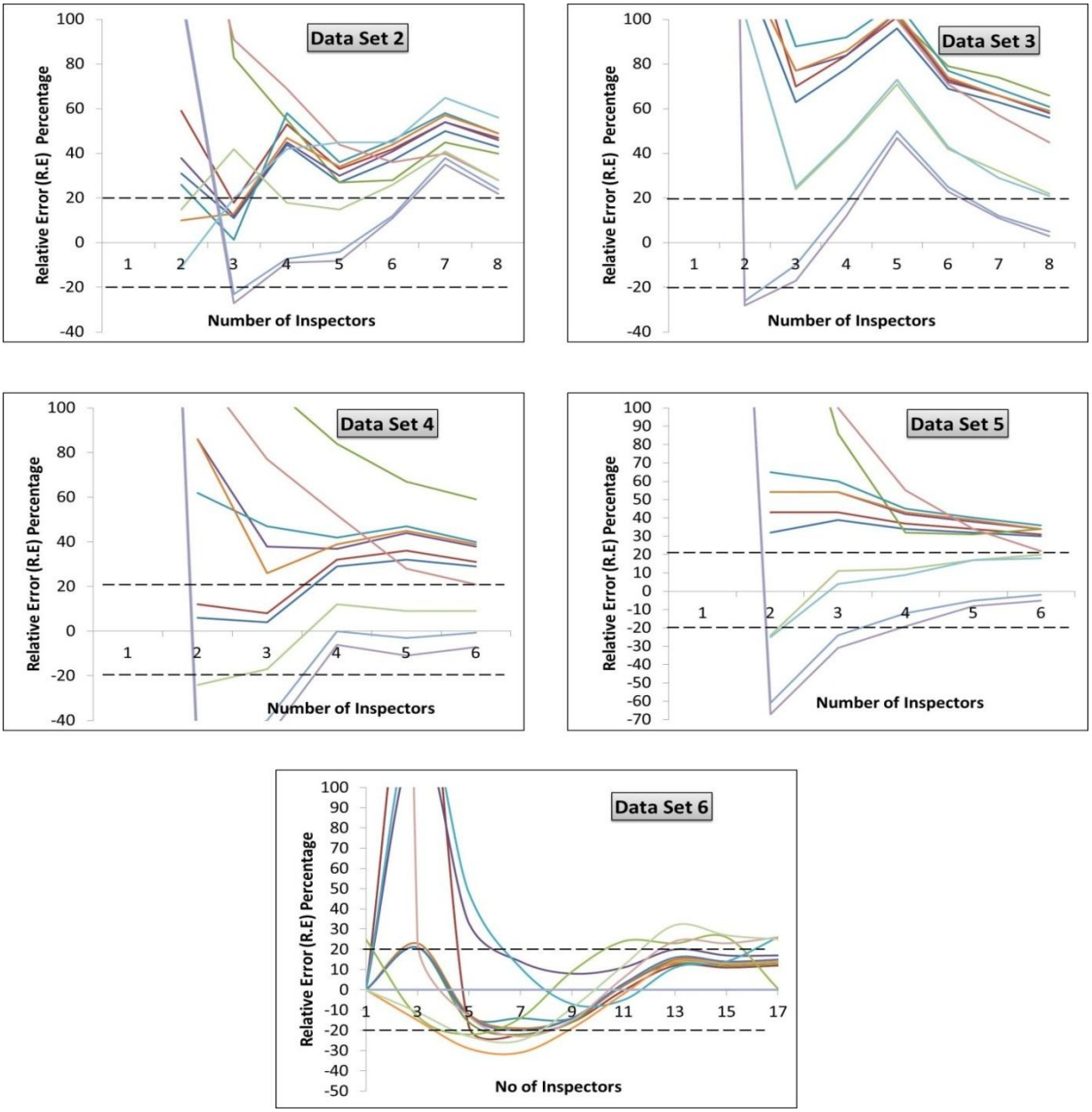


Figure 10. Relative Error in Median M_k for all the Estimators for Data Sets 2 to 6

results within 20% of the actual value with an inspection team size of seven and eight inspectors.

c) Results from Data Sets 4 and 5 showed a similar trend. That is, only the SC estimators for the M_h and M_{th} models achieved satisfactory median M_k values (i.e., within +/- 20% of the actual M_k values). Rest of the CR estimators never reached the 20% relative error range.

d) Results from Data Set 6 showed that, most of the CR estimators achieved satisfactory results for inspection team size of more than 9 or 11 inspectors (depending on the type of the estimator).

Furthermore, we analyzed the relative improvement in the performance for different CR estimators with increase in the inspection team size at varying level of relative error (R.E) in the M_k values (in similar fashion as done for data set 1 and described in Table 9). Table 10 provides insights into the minimum number of inspectors required by each CR estimator to obtain median M_k value within +/- 20% of the actual M_k value for all the 5 data sets (i.e., data sets 2 - 6). The

Table 10. Number of Inspectors Required for +/- 20% Relative Error in M_k for Data Set 2-6

	Data Set#	Mo Model			Mt Model			Mh Model			Mth Model	
		UMLE	CMLE	EE	UMLE	CMLE	EE	SC	JK	EE	SC	EE
Inspector Count for R.E within +/_ 20%	2	Never within +/- 20%										
	3	Never within +/- 20%						7	Never		7	Never
	4	Never within +/- 20%						4	Never		4	Never
	5	Never within +/- 20%						4	Never		4	Never
	6	9	9	15	11	9	NA	9	Never		9	Never

inspector count for each R.E percentage shown in Table 10 is the minimum number of inspectors for which the estimate falls within the given R.E. range and never goes outside the range as the number of inspectors increase.

The results in Table 10 suggest some general observations as follows:

a) For data sets 2-5, only the SC estimators for the M_h and M_{th} models were able to estimate the cost-effectiveness of inspection process at level within 20% of the actual M_k value for inspection team size more than 4 or 7 inspectors (depending on the data set). No other CR estimator produced median M_k value that was within acceptable relative error (i.e., +/- 20% of the actual value);

b) For Data Set 6, most of the estimators were able to produce median M_k value that was with a relative error of less than 20% of the actual M_k value. This was true for inspection team size of 9 or 11 or 15 inspectors (depending on the estimator). Also, an exception to the earlier results, M_k values from the JK estimator never consistently stayed within +/- 20% R.E range with the increase in the number of inspectors.

Therefore, based on the median R.E values, SC are the best estimators to be used when evaluating the cost-effectiveness of software inspections.

6. THREATS TO VALIDITY

The following threats to validity are present in our study:

C**onclusion Validity.** The threat due to the heterogeneity of participants was not controlled across all the data sets. The inspectors in Data Set 1 were Microsoft professionals whereas the inspectors in Data Set 2 through 6 were undergraduate and graduate students.

External Validity. Data sets 2 through 5 were obtained in a course setting where the participants worked with a real client to develop requirements for a system that they later implemented. However, there remains a threat because the participants were all undergraduate students in an educational setting and likely do not represent professional developers. Also, the nature of faults made by students during development can differ from the faults made by software professionals. To mitigate this validity threat, Data Sets 1 and 6 were industrial strength requirement documents that contained realistic defects. In Data Set 1, the realistic defects were seeded into the document rather than being naturally occurring (as in Data Set 6). But, the defects were seeded by researchers who had no knowledge that results would be used for a capture-recapture study. Therefore, the defects were not seeded in such a way to specifically benefit a capture-recapture analysis.

Construct Validity. The actual number of defects present in Data Sets 2 through 6 is not known and might actually be higher than the assumed defect count (i.e., the total number of faults found after two inspections). Also, we did not collect any data regarding faults that might have occurred during implementation. During the original inspection studies (from which data sets 2 through 6 were analyzed), the CR models were not used, the inspectors' subjective opinion regarding the remaining faults after the second inspection (which was all that was available) was collected. The inspectors agreed that they had located all the faults present in the artifact during

second inspection, ruling out any need of further inspection. So, the inspection process was stopped.

Internal Validity. To reduce the threat of using a small number of inspectors, the number of inspectors used in Data Set 1 is the largest used in any previous study of this type. Additionally, to control the variability of the inspection techniques, we used the data from first inspection for all the six data sets in which all the inspectors used the same inspection technique (i.e., fault checklist) to detect defects.

7. DISCUSSION OF RESULTS

This section discusses the results from all the six data sets in light of the original research goals. We discuss the major finding and recommendation about the minimum number of inspectors required for to maximize the cost-effectiveness of software inspections; and the number of inspectors required by the CR models and estimators to achieve estimates of the cost-effectiveness of inspection process within satisfactory relative error range.

Effect of Inspection Team Size on Cost-Effectiveness: The results demonstrate that performing inspections during the early stages of software development returns significant cost savings (in hours). Even an inspection conducted by just one inspector returns a positive test cost savings. The increase in cost savings is positively correlated with the *number of inspectors* up to certain team size (i.e., when majority of the defects have been detected) beyond which the cost savings does not increase with further increase in the number of inspectors. This point of maximum cost-effectiveness and the level of cost savings are dependent on the number of defects present in the document prior to the inspection, and can vary depending on the product and the defect detection abilities of inspectors. Project managers can vary the inspection team size (using the same process as described in this paper) to evaluate the improvement in the cost-effectiveness of inspection results to decide if additional inspectors need to be employed for saving further testing cost. Our results also provided the minimum number of inspectors that are required to achieve varying amount of cost-savings. This information can also help project managers to decide on the baseline number of inspectors to use when planning an inspection process and then decide if more number of inspectors needs to be added to increase the cost savings.

Cost-Effectiveness Using the Capture Recapture Estimators: The results showed that the SC estimators for the M_h and M_{th} models can be used with a minimum of four to nine inspectors

(depending on the data set) to accurately predict the cost-effectiveness of software inspections within +/- 20% relative error level. Our results also provide information about the minimum number of inspectors to be used for achieving more accurate (i.e., at +/- 10% or at 0% relative error level) cost-effectiveness results. Since, the total fault count is never known beforehand prior to the inspection, the CR estimates from the SC estimators can be used to make cost-effective re-inspection decisions.

8. CONCLUSION AND FUTURE WORK

Based on the results provided in this paper, project managers can apply the kusumoto metric to evaluate the cost-effectiveness of the inspection process in their organizations. Results from the application of kusumoto metric can help them decide if more number of inspectors is needed to achieve larger cost-savings. Furthermore, the CR estimate of the total fault count can be used while deciding on a need of re-inspection. Software organizations can use the results in this paper about the number of inspectors required for achieving varying levels of cost-effectiveness of the inspection process. The result will also help software project managers make objective post-inspection decisions on whether using more inspectors to perform additional inspection will increase the further cost-savings.

9. **REFERENCES**

[1] Ackerman, A., Buchwald, L., and Lewski, F., "Software Inspections: An Effective Verification Process". *IEEE Software*, 1989, **6**(3): 31-36.

[2] Briand, L.C, Emam, K.E, and B.G.Freimut, "A Comparison and Integration of Capture-Recapture". In *Proceedings of the 9th International Symposium on Software Reliability Engineering*, 1998, Paderborn, Germany: 32-41.

[3] Briand, L.C, Emam, K.E, Freimut, B.G, and Laitenberger, O, "A Comprehensive Evaluation of Capture Recapture Models for Estimating Software Defect Content". *IEEE Transactions on Software Engineering*, 2000. **26**(6): 518-539.

[4] El-Emam, K., Laitenberger, O., and Harbrich, T., "The Application of Subjective Estimates of Effectiveness to Controlling Software Inspections". *Journal of Systems and Software*, 2000, **54**(2): 119-136.

[5] Otis, D., Burnham, K., White, G., and Anderson, D., "Statistical Inference from Capture Data on Closed Animal Population*". Wildlife Monograph*, 1978, **64**: 1-135.

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

[7] Boehm, B., and Basili, V.R., "Software Defect Reduction Top 10 List". IEEE Computer, 2001, 34(1): 135-13.

[8] Meyer, G, "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections". In *Communications of the ACM*, September 1978, 21(9):760-768.

[9] Collofello, J.S, Woodfield, S.N, "Evaluating the Effectiveness of Reliability-Assurance Techniques". *Journal of Systems and Software, 1989*, 9 (3) 191-195.

[10] SABALIAUSKAITE, G, "Investigating Defect Detection in Object-Oriented Design and Cost-Effectiveness of Software Inspection". January 2004.

[11] Kusumoto, S, Matsumoto, K, Kikuno, T, Torii, K, "A New Metrics for Cost Effectiveness of Software Reviews". *IEICE Transactions on Information and Systems*, 1992, E75-D (5) 674-680.

[12] Fagan, M. E., "Advances in Software Inspections". *IEEE Transactions on Software Engineering*, July 1986, Vol. SE-12, No. 7, pp. 744-751.

[13] Kelly, J. C., Sherif, J. S., Hops, J., "An Analysis of Defect Densities Found During Software Inspections". *Journal of Systems Software*, 1992. Vol. 17, pp. 111-117

[14] Gilb, T, Graham, D, "Software Inspection". *Addison-Wesley*, 1993.

[15] Biffl, S, Freimut, B, Laitenberger, O, "Investigating the Cost-Effectiveness of Reinspection in Software Development". *Proceedings of the 23rd International Conference on Software Engineering*, 2001. pp. 155-164.

[16] Franz, L, and Shih, J, "Estimating the Value of Inspections and Early Testing for Software Projects". In *Hewlett-Packard Journal*, December 1994, pp. 60-67.

[17] Boehm, B, "Software Engineering Economics". *Prentice-Hall*, 1981.

[18] Briand, L, Emam, K. El, Laitenberger, O, and Fussbroich, T, "Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects"., In *Proceedings of the 20th International Conference on Software Engineering*, 1998. pp. 340-349.

[19] Weller, E, "Lessons from Three Years of Inspection Data". In *IEEE Software*, September 1993, 10(5):38–45.

[20] Olson, T., "Piloting Software Inspections to Demonstrate Early ROI," *Notes from Presentation given at the 1995 SEPG Conference.*

[21] McGibbon, T, "A Business Case for Software Process Improvement". *A DACS State-of-the-Art Report*, September 1996.

[22] Eick, S., Loader, C., Long, M., Votta, L., and Weil, S.V. "Estimating Software Fault Content Before Coding*"*. In *Proceedings of the 14th International Conference on Software Engineering*, 1992, Melbourne, Australia: ACM Press: 59-65

[23] White, G.C., Anderson, D.R., Burnham, K.p., and Otis, D.l., "Capture-Recapture and Removal Methods for Sampling Closed Populations". *Los Alomos National Laboratory, 1982.*

[24] Madachy, R., "Process Improvement Analysis of a Corporate Inspection Program," *Seventh Software Engineering Process Group Conference*, Boston, MA, May 23, 1995.

[25] Miller, J., "Estimating the Number of Remaining Defects after Inspection". *Software Testing, Verification, and Reliability, 1999*, Vol. 9, No. 3, pp. 167-189.

[26] Serkan Nalbant., "An Evaluation of the Reinspection Decision Policies for Software Code Inspections". *In Thesis Submitted to The Graduate School of Natural and Applied Sciences of Middle East Technical University*, Jan 2005.

[27] Wiegers, K.E., "Peer Reviews in Software: A Practical Guide". *Addison-Wesley Information Technology Series*, 2002.

[28] Petersson, H., Thelin, T., Runeson, P., and Wohlin, C., "Capture-Recapture in Software Inspections after 10 Years Research - Theory, Evaluation and Application". *Journal of Systems and Software*, 2003.

[29] Burnham, K.P. and Overtom, W.S., "Estimation of the Size of a Closed Population When Capture Probabilities Vary Among Animals*"*. *Biometrica*, 1978, **65**: 625-633.

[30] Darroch, J.N., "The Multiple-Recapture Conensus 1: Estimation of a Closed Population". *Biometrika*, 1958, **45**: 343-359.

[31] Yip, P.S.F., "A Martingale Estimating Equation for a Capture-Recapture Experiment in Discrete Time*"*. *Biometrics*, 1991, **47**: 1081-1088.

[32] Chao, A., "Estimation the population Size for Capture-Recapture Data with Unequal Catchability". *Biometrics*, 1987, **43**(4): 783-791.

[33] Chao, A., "Estimating Animal Abundance with Capture Frequency Data*"*. *Journal of Wildlife Management*, 1988, **52**(2): 295-300.

[34] Lee, S.M. and Chao, A., "Estimating Population Size via Sample Coverage for Closed Capture-Recapture Models". *Biometrics*, 1994, **50**: 88-97.

[35] Chao, A. and Yeng, H.C., "Program CARE-2 (for Capture-Recapture Part.2)". *http://chao.stat.nthu.edu.tw, 2003.*

[36] Walia, G., and Carver, J., "Using Error Abstraction and Classification to Improve Requirement Quality: Conclusions from a Family of Four Empirical Studies". *Empirical Software Engineering: An International Journal*, 2012. DOI: 10.1007/s10664-012-9202-3

[37] Brykczynski, B., "A Survey of Software Inspection Checklists". *ACM SIGSOFT Software Engineering Notes*, 1999, 24(1): 82-89.

[38] Bernardez, B., Genero, M., Duran, A., and Toro, M. "A controlled Experiment for Evaluating a Metric-Based Reading Technique for Requirement Inspection". In *Proceedings of the 10th International Symposium on Software Metrics (METRICS'04), 2004: IEEE Computer Society*: pp. 257-268

[39] Carver, J., "The Impact of Background and Experience on Software Inspections", *PhD Thesis, Department of Computer Science, University of Maryland College Park,* MD, 200

[40] Walia, G., Carver, J. and Nagappan, N. "The Effect of the Number of Inspectors on the Defect Estimates Produced by Capture-Recapture Models". *Proceedings of the 30th*

International Conference on Software Engineering, May 10-18, 2008, Leipzig, Germany, pp. 331-340.

[41] Walia, G., and Carver, J. "Evaluation of Capture-Recapture Models for Estimating the Abundance of Naturally Occurring Defects". *Proceedings of the 2nd ACM-IEEE International Symposium of Empirical Software Engineering and Measurement*, October 9-10, 2008. Kaiserslautern, Germany. pp. 158-167