ISSN:0975-9646

Investigating the Effect of Fault Category on Overall Capture Probability during Requirements Inspection

Tejalal Choudhary[1], Anurag Goswami[2]

Computer Science Department
Sushila Devi Bansal College of Technology, Indore, India

Computer Science Department
North Dakota State University, Fargo, USA

***Abstract*— Inspection helps software managers by eliminating faults in early phases of Software Development Lifecycle (SDLC). Capture-recapture is a technique which provides an estimates of faults remaining in a software artifact. This helps managers to make a decision whether a re-inspection is required or not. At a higher level, software requirements document consists of general faults, faults of omission and faults of comission. A common belief is that the faults that are visible in the artifact are easier to detect by the inspectors. The nature of faults that could be detected by inspectors has not been empirically studied. Using the inspection data from varying number of students, we analyzed the category (general or omission or comission) of faults that are easier to detect by the inspectors. The results from this study shows that faults of comission are significantly easier to detect by the inspectors.**

***Keywords*— Inspections, Requirements/Specifications, Review.**

I. INTRODUCTION

Requirements and design document are early artifacts that are developed during initial phases of *Software Development Lifecycle* (SDLC). A successful software organization thrives to build quality software product within allocated budget and time [1]. Evidence shows that faults are introduced in early artifacts (i.e. requirements and design documents) during their development. These faults are harder to find and fix if problems are left undetected and penetrate to later stages of development [2]. While 50-80% of development effort is spent at the testing stage, it is estimated that 40-50% of this is spent on fixing faults committed during the early stages [3]. To deliver a quality software product within the available resources, researchers have focused on validating methods of prevention and detection of early lifecycle faults [4].

Software requirements development is the first and most critical phase of SDLC. This phase involves gathering of requirements from different stakeholders (technical and non-technical both) and are recorded in a document known as *Software Requirements Specification* (SRS). SRS is often written in *Natural Language* (NL) which is a means of communication among different stakeholders, is specially defect prone due to its complex, imprecise, vague and ambiguous characteristics.

To eliminate early faults in software artifacts, software inspections are widely used [5] and are empirically validated [6]. Inspection process includes examining a software artifact by a group of inspectors to uncover faults in it. Inspections saves cost by avoiding costly rework and thereby, improving quality of software artifacts [7]. During an inspection, inspection team leader first selects a team of skilled individuals (as inspectors) who will perform inspections. Evidence in past shows that the effectiveness of an inspector during the individual review of software artifact significantly impacts the overall effectiveness of the inspection as a team [8].

Project manager needs an objective and reliable information which helps them to decide what category (general/omission/comission) of faults are easier to detect by the requirements inspectors. This would help them to concentrate better towards the training part of inspectors. To achieve this objective this research utilizes the most appropriate method of faults estimation known as capture-recapture [9, 10].

A controlled empirical study was conducted at Sushila Devi Bansal College of Technology (SDBCT). Participants performed an individual inspection of a NL requirements document using fault-checklist technique and logged the faults found during the inspection. By using the capture-recapture technique, faults detected, its category (general/omission/comission), and number of inspectors to estimate the category of faults that are easier to detect were provided as an input. The results from this study show that fault of comission significantly easier to detect by the inspectors as compared to faults of omission.

II. BACKGROUND

This section provides background information regarding fault-checklist that is used for inspections (Section II.A). Section II.B describes capture-recapture models with a brief description of use of capture-recapture models in SE.

A. *Fault-checklist Technique*

Fault-checklist is one of the most popular way of performing an inspection [11]. In this method inspectors are provided with a fault-checklist using which they can guide inspections [12]. Fault-checklist consists of fault category and types of faults lie in those categories. Below is a brief description of fault category and its types.

General (G)

- Are the goals of the system defined?
- Are the requirements clear and unambiguous?

- Is a functional overview of the system provided?
- Is an overview of the operational modes provided?
- If assumptions that affect implementation have been made, are they stated?
- Have the requirements been stated in the terms of inputs, outputs, and processing for each function?
- Are all functions, devices, constraints traced to requirements and vice versa?
- Are the required attributes, assumptions and constraints of the system completely listed?

Omission (O)

- *Missing Functionality (MF)*
  - Are the desired functions sufficient to meet the system objectives?
  - Are all inputs to a function sufficient to perform the required function?
  - Are undesired events considered and their required responses specified?
  - Are the initial and special states considered (e.g., system initiation, abnormal termination)?
- *Missing Performance (MP)*
  - Can the system be tested, demonstrated, analyzed or inspected to show that it satisfies the requirements?
- *Missing Interface (MI)*
  - Are the inputs and outputs for all interfaces sufficient?
  - Are the interface requirements between hardware, software, personnel and procedures included?
- *Missing Environment (ME)*
  - Have the functionality of hardware or software interacting with the system been properly specified?

Comission (C)

- *Ambiguous Information (AI)*
  - Are the individual requirements stated so that they are discrete, unambiguous, and testable?
  - Are all mode transitions specified deterministically?
- *Inconsistent Information (II)*
  - Are the requirements mutually consistent?
  - Are the functional requirements consistent with the overview?
  - Are the functional requirements consistent with the actual operating system?
- *Inconsistent and Extra Functionality (EF)*
  - Are all desired functions necessary to meet the system objectives?
  - Are all inputs to a function necessary to perform the required function?
  - Are the inputs and outputs for all interfaces necessary?
  - Are all the outputs produced by a function used by another function or transferred across an external interface?
- *Wrong Section (WS)*
  - Are all the requirements, interfaces, constraints, etc. listed in the appropriate sections?

Inspectors can use these guidelines and can record fault with their appropriate fault category and type in a category.

B. *Capture-Recapture (CR) Overview*

CR is a technique originally developed by biologists to statistically estimate the size of wildlife populations. The following process is followed by biologists to estimate the wildlife population:

- A biologist captures fixed number of animals, mark them as captured, and release them back into the population.
- Next, another capture (trapping) occasion occurs after animals get chance to remix. If an animal marked in the first occasion is found again, then it is known as recaptured.
- This process is repeated multiple times. A large overlap of animals in different trapping indicates a smaller population [13, 14].

Using the same CR principle, faults are estimated for an artifact during inspections. During inspections, faults are detected by an inspector. If the same fault is detected (captured) by another inspector then it is known as to be recaptured [9, 10]. The whole process of estimating total faults are same as followed by biologists. *Animals* are replaced by the *faults* in an artifact and *trappings* are replaced by the *inspectors*.

There are some assumptions [9, 10] of CR in wildlife research that does not hold true for software requirements inspection. This could be understood more from Table I. A closed population (all inspectors detect faults in the same artifact without any modifications) and capture marks are not lost (faults are recorded by each inspector) assumptions are met. But in inspections, due to the different abilities of inspectors, equal capture probability assumption is not met [15, 16].

TABLE I. DIFFERENCE IN ASSUMPTIONS OF CR IN INSPECTIONS

S.No.	**Wildlife**	**Inspections**
1.	A closed population	A closed population
2.	*An equal capture probability*	*Some faults are easier to detect*
3.	Capture marks are not lost	Capture marks are not lost

Details of the models in CR estimation with CARE-4 tool has been derived by Huggins [17]. In this study model MM_{th} is taken. Which means, inspectors differ in their inspection ability as well as defects have different probability of being found. The mathematical implementations are not provided in this paper but can be found in the references provided. The input data used by CR estimators is organized as a matrix with rows representing faults and columns representing inspectors. Also, the number of attributes of faults (one in this study, i.e. fault category) is represented by last number of columns as shown in Fig. 1. A matrix entry is 1 if the ***fault (A)*** is detected by an ***inspector (C)*** and 0 otherwise. The column of ***fault attribute (D)*** follows the number attributes a fault has (in this case it is one i.e. fault category).

$$\begin{array}{c|cccccccc}
 & 1 & 2 & \cdots & C & 1 & \cdots & & D \\
\hline
1 & X_{11} & X_{21} & \cdots & X_{C1} & Y_{11} & Y_{21} & \cdots & Y_{D1} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
A & X_{1A} & X_{2A} & \cdots & X_{CA} & Y_{1A} & Y_{2A} & \cdots & Y_{DA}
\end{array}$$

Fig. 1: CR data input matrix

Erick, et al. introduced CR in software inspections by applying it on the real AT&T data. His major recommendation was that an artifact should be reinspected if it contains more than 20% of undetected faults [9, 18]. Various research followed this study and evaluated the use of CR model in SE [10]. Prior work evaluated that the CR models usually underestimate the true fault count, but their estimation accuracy improves as number of inspectors are increased [19]. However, the prior CR studies neglected the effect of fault category on inspection. The existing CR studies in SE used artifacts with seeded number of defects but in this study, requirements document contains naturally occurring faults (i.e. faults manifested during development of SRS). This work extends CR estimates to fault attributes (i.e. fault category in this study), known as individual covariates.

III. EXPERIMENT DESIGN

The primary motivation for this study was to provide an evidence that one of the fault category (i.e. general, omission and comission) has a higher probability to be detected (i.e. captured) by requirements inspector during inspection. Data set was utilized where participants used the fault-checklist to guide inspections of the software requirements and reported faults.

A. *Research Goal*

To investigate the effect of various fault categories on overall capture probability of faults in inspection.

B. *Data Set*

Our study uses data from an inspection study conducted at SDBCT during 2013. The following subsection describes the data sets in terms of the artifacts inspected, the types of faults, the inspectors and the inspection process followed.

Data Set: Data set came from an in-class inspection study conducted at SDBCT where a NL requirement artifact was individually inspected by thirty participants over a period of ninety minutes.

Artifacts: participants in the study were given requirement document (developed internally by the students in 2010) that described the requirements for *Online Polling System* (OPS). The OPS system is responsible for registering users and enabling them to poll for a political party with the help of a web application. This system gave the users the ability to poll for their favourite candidate while being at either work place or at their homes, it also gives the polling administrator a better manageability over the candidates/people registered, and election results.

Faults: The faults in the artifact were natural faults that occurred during the development of requirements document. A master fault list was created which contained the faults detected by the participants as well as faults detected by one of the researcher. This data set consists a total of 48 naturally occurring faults.

Inspectors: The 30 inspectors for study were students enrolled in the Computer Science course during 2013. The subject (minor project) cover the understanding of the importance of *Software Engineering* (SE) and its processes. This subject covers the requirements and design development and the necessary skills for planning, analysis, and design of software system.

C. *Experiment Procedure*

Step 1: Training and Inspecting SRS for Faults: During the training, the participants received the SRS document for the OPS system, the fault-checklist and a list of different fault types. They were instructed on how to use the fault-checklist to record faults using a set of example requirements. All the participants were instructed by the same instructor.

Step 2: Inspection of OPS SRS: Next, the subjects individually inspected the OPS requirement document using the fault-checklist and log faults during the inspection.

In addition, the fault reporting forms required the subjects to classify the faults identified during the inspection into one of the following fault types: *General* (G), *Missing Functionality* (MF), *Missing Performance* (MP), *Missing Information* (MI), *Missing Environment* (ME), *Ambiguous Information* (AI), *Inconsistent Information* (II), *Extraneous* (EF), *Wrong Section* (WS), and *Others* (O).

The researcher validated that the fault reported by each participant were true positives. The researcher, who had knowledge of the system for which the requirements were developed, read through the faults reported by each participant to remove any false-positives before analyzing the data. Researcher also placed faults found in their required category (i.e. General, Omission, and Comission) as participants only marked fault type.

D. *Evaluation Criterion*

This section explains the procedure used to evaluate the research goal (Section III.A). A tool was used known as CARE-4 developed by Hsin-Chou Yang and Anne Chao from Institute of Statistics, National Tsing Hua University, Hsin-Chu, Taiwan. The tool can be downloaded from the link below:

http://chao.stat.nthu.edu.tw/blog/software-download/care/

This tool uses the concept of capture-recapture technique to estimate the total population as well as impact of various attributes/properties of population (known as individual covariates in CARE-4) [20, 21]. In one of the capture-recapture experiments [19], one author have used the previous version of this tool known as CARE-2 which only estimates the total population without taking their attributes (i.e. individual covariates) into account. CARE-4 takes the input in the form of matrix (described in Section II.B).

The general logistic model MM*tbh is:

logit(P_ij)=a + c_j + v * Y_ij + beta * W_i + r * R_j

where,

i : refers to the ith individual;
j : refers to the jth sample or jth capture occasion;
a : baseline intercept;
c_j : the unknown time or occasional effect of the jth capture occasion (set c_t=0, where t: the number of capture occasions;
v : (behavioral response) the effect w.r.t. the past capture history indicator Y_ij;
beta : the effect of individual covariates W_i;
r : the effect of occasional covariate R_j;

The tool automatically computes all the estimates with two types of covariates: *individual covariates* of population (i.e. faults) and *occasional covariates* which are the properties of any external factor (e.g., capture during day or night) [20]. The covariates could be discrete (e.g., gender) or continuous (e.g., weight in kilograms). In this experiment, there are not any occasional covariates. Below (Fig.2) is the actual fault matrix created for the tool as an input for 30 participants as inspectors and 48 faults along with one individual covariate as fault category. In Fig. 2, I_1, I_2… I_{30} are inspectors and IC_1 is *Individual Covariate 1* (i.e. fault categories) which are marked *as General: 1, Omission: 2, and Comission: 3* in the input matrix.

Following are the steps performed in CARE-4 tool to estimate the probability of fault category detected by 30 inspectors:

- Select analysis with covariates in options.
- Input the number of distinct individuals (48 faults) and sampling occasions (30 inspectors).
- Input number of individual covariates (1 i.e. fault category).
- Input number of continuous individual covariates (0).
- Input the fault matrix created (Fig. 2).
- Input the number of occasional covariates (0).
- Input the number of continuous occasional covariates (0 again).
- Input unknown time effects y or n? (‘n’)
- Provide the output file path which saves the results of CR estimation with covariates (i.e. includes fault properties).

$$\begin{array}{c|ccccccccc}
\text{Faults} \backslash \text{Inspectors} & I_1 & I_2 & I_3 & I_4 & I_5 & \cdots & \cdots & I_{30} & IC_1 \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & \cdots & \cdots & 0 & 2 \\
2 & 0 & 0 & 0 & 0 & 0 & \cdots & \cdots & 0 & 1 \\
3 & 1 & 0 & 0 & 1 & 0 & \cdots & \cdots & 0 & 2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
48 & 0 & 0 & 0 & 0 & 0 & \cdots & \cdots & 0 & 2
\end{array}$$

Fig. 2: Actual input matrix for CARE-4 CR tool

IV. Analysis and Results

The analysis of impact of fault category on the probability of detecting faults focuses on research goal in section III.A. This section evaluates the effect of fault categories on the inspection effectiveness. As stated earlier all 30 participants perform individual inspections of OPS document using fault-checklist technique which turns out to be individual inspection data, shown in Fig. 3. Individual results are combined with fault category to form CR input matrix for CARE-4 tool. Fault categories comprises of G: general, O: omission, and C: comission faults.

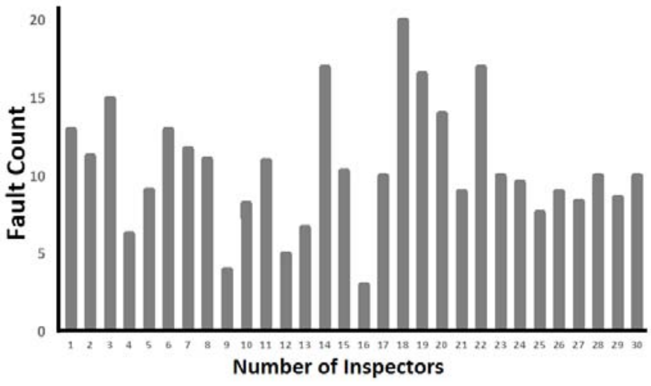

Fig. 3: Individual inspection data

To provide an overview of the results, Fig. 4 compares the *number of faults found* in each category by 30 inspectors. Results are arranged by increasing fault category number. Solid bar represents actual number of faults that exists for a category and stripped bar represents the number of times a fault category is detected by 30 inspectors. To understand the impact of different fault categories, results are organized by the increasing number of fault categories (General: 1, Omission: 2, and Comission: 3).

The results from Fig 4 shows that the ratio between number of times a fault category is detected to number of faults of a category present is highest for general faults (3.68), then it is for comission faults (2.73) and lest one is for omission faults (1.9). Which means general faults have higher tendency to be detected during an inspection but due to less number of general faults present in OPS this cannot be validated. For comission faults results could hold true that their detection probability is highest among all three fault categories.

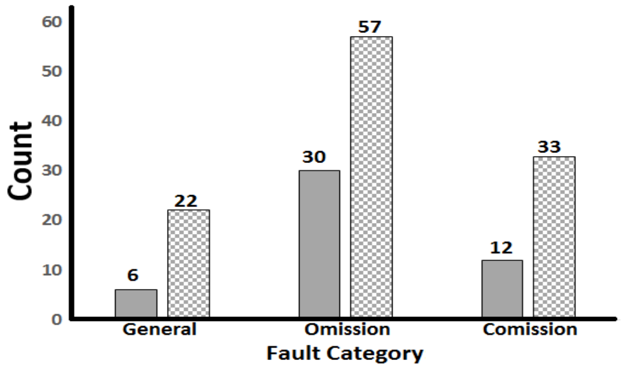

Fig. 4: Comparison of different fault categories on inspection

To evaluate research goal, input matrix was provided to CARE-4 tool which outputs the regression coefficient along with Standard Error. In this study we have selected MM_{th} model for results (as faults does not have any behavior like animals). Results show that omission faults have strong negative *(-0.78)* and general faults have positive but weak *(0.36)* regression coefficient *(with standard error 0.14 and 0.18)* with reference to comission faults.

Therefore, based on these results, during an inspection guided by fault checklist method, faults of comission does appear to have higher probability of getting detected (captured) during a requirement inspection. While the result is positive for general faults but they are not significant.

V. Threats to Validity

In this experiment some of the validity threats were addressed. Same SRS for all the participants was used which handled the heterogeneity effect. Participants were selected from the same course (i.e. same level of educational background). To address the training bias, all the participants obtained training from one trainer. Fatigue effect was also handled because participants had enough time to perform requirements inspections where they can work in their comfortable environment and can take breaks. However, participants were students from academic settings and are likely not represent professionals in an industry setting with practical experience. Actual number of faults present were not known, as SRS document has natural faults.

VI. Discussion of Results

A major focus of this study is to find out the effect of different fault categories on inspection. Research goal focused on understanding whether probability of a particular fault category detection is higher during requirements inspection as compared to other categories. The results from Section IV showed that the faults of comission has a higher probability of getting detected during inspection. This means, faults that are not visible in the requirements document are not easy to be detected by the inspectors in inspection. When the result was tested statistically, there was a significant negative regression coefficient for fault of omission. Therefore, it is easy to detect mistakes as compared to detect content which has to be in requirements document.

VII. Conclusion and Future Work

Based on the results provided in this paper, concept of CR along with covariates (population/individual properties) can help to manage the quality of software by laying more stress on detecting fault of omission during inspections. CARE-4 tool can be used to extend various CR studies which does not include covariates of individuals. This concept can also be extended to other phases of software development (e.g., design review, code review etc). These results motivate us for further investigation. Immediate future works includes replicating the studies for larger data sets which would also include investigation of different fault types. Another future work is to replicate the research studies during the inspection of the design documents and the code and the test plan reviews.

References

[1] Nalbant, S.: ‘An Evaluation of the Reinspection Decision Policies for Software Code Inspections’, MIDDLE EAST TECHNICAL UNIVERSITY, 2005

[2] Boehm, B., and Basili, V.R.: ‘Software defect reduction top 10 list’, Foundations of empirical software engineering: the legacy of Victor R. Basili, 2005, 426

[3] Perry, W.E.: ‘Effective Methods for Software Testing: Includes Complete Guidelines, Checklists, and Templates’ (John Wiley & Sons, 2006. 2006)

[4] Leszak, M., Perry, D.E., and Stoll, D.: ‘A case study in root cause defect analysis’. Proceedings of the 22nd international conference on Software engineering2000 pp. 428-437

[5] Fagan, M.E.: ‘Design and code inspections to reduce errors in program development’: ‘Pioneers and Their Contributions to Software Engineering’ (Springer, 2001), pp. 301-334

[6] Doolan, E.: ‘Experience with Fagan's inspection method’, Software: Practice and Experience, 1992, 22, (2), pp. 173-182

[7] Ackerman, A.F., Buchwald, L.S., and Lewski, F.H.: ‘Software inspections: An effective verification process’, IEEE software, 1989, 6, (3), pp. 31-36

[8] Porter, A., Siy, H., Mockus, A., and Votta, L.: ‘Understanding the sources of variation in software inspections’, ACM Transactions on Software Engineering and Methodology (TOSEM), 1998, 7,(1),pp.41-79

[9] Briand, L.C., El Emam, K., Freimut, B.G., and Laitenberger, O.: ‘A comprehensive evaluation of capture-recapture models for estimating software defect content’, Software Engineering, IEEE Transactions on, 2000, 26, (6), pp. 518-540

[10] 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, 2004, 72, (2), pp. 249-264

[11] Thelin, T., Andersson, C., Runeson, P., and Dzamashvili-Fogelstrom, N.: ‘A replicated experiment of usage-based and checklist-based reading’. Software Metrics, 2004. Proceedings. 10th International Symposium on2004 pp. 246-256

[12] Thelin, T., Runeson, P., and Wohlin, C.: ‘An experimental comparison of usage-based and checklist-based reading’, Software Engineering, IEEE Transactions on, 2003, 29, (8), pp. 687-704

[13] Lee, S.M., and Chao, A.: ‘Estimating population size via sample coverage for closed capture-recapture models’, Biometrics, 1994, pp. 88-97

[14] White, G.C.: ‘Capture-recapture and removal methods for sampling closed populations’ (Los Alamos National Laboratory, 1982. 1982)

[15] Boehm, B.W.: ‘Software engineering economics’, 1981

[16] Sabaliauskaite, G.: ‘Investigating defect detection in object-oriented design and cost-effectiveness of software inspection’, Citeseer, 2004

[17] Yang, H.C., and Chao, A.: ‘Modeling animals' behavioral response by Markov chain models for capture–recapture experiments’, Biometrics, 2005, 61, (4), pp. 1010-1017

[18] Eick, S.G., Loader, C.R., Long, M.D., Votta, L.G., and Vander Wiel, S.: ‘Estimating software fault content before coding’. Proceedings of the 14th international conference on Software engineering1992 pp. 59-65

[19] Walia, G.S., and Carver, J.C.: ‘Evaluation of capture-recapture models for estimating the abundance of naturally-occurring defects’. Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement2008 pp. 158-167

[20] Huggins, R.: ‘Some practical aspects of a conditional likelihood approach to capture experiments’, Biometrics, 1991, pp. 725-732

[21] Huggins, R.: ‘On the statistical analysis of capture experiments’, Biometrika, 1989, 76, (1), pp. 133-140