ITERATIVE CODE REVIEWS SYSTEM FOR DETECTING AND CORRECTING FAULTS FROM SOFTWARE CODE DOCUMENTS

Mr. Shinde Santaji Krishna¹, Dr. Shashank Dattatraya Joshi²

¹Research Scholar, Department of Computer Engineering, Shri Jagdish Prasad Jhabarmal Tibrewala University, Vidyaganj, Jhunjhunu, Rajasthan, India
²Professor, Department of Computer Engineering, Bharati Vidyapeeth Deemed University College of Engineering, Pune, Maharashtra, India

ABSTRACT

Iterative Code Review (ICR) is a better way for detecting faults in the software code document. Recent techniques of reviews require several inspectors i.e. manpower is required to perform the reviews. One person can’t detect all types of the faults in a single review. The process of using the number of peoples for code reviews is quite expensive process. Iterative Code Review Process needs only one inspector. Iterative Code review process contains the number of iterations. The use of the iterations is done for detecting and correcting the different types of faults. During every iteration, a particular type of fault is detected and corrected. We are considering Java language for ICR. The final result will be fault free java document.

Keywords: Code, Faults, Inspection, Iterative, Review.

1. INTRODUCTION

The technique introduced by Fagan is considered to be the most capable method for removing software bugs [1]. In all phases of a SDLC, it is used widely [2]. Traditional Fagan type inspection describes a number of roles for the members of a software inspection meeting: organizer, inspector, author, recorder and moderator. Even though the same person may assume multiple roles, in classical inspection usually, 3–6 participants attends the meeting. This technique is costly.

Therefore, in several years, better inspection techniques were introduced [3] [4] [5] [6]. There is involvement only two participants for code reviews that are discussed in [7]. The Code writer is the first member which is also performed as a coordinator, recorder, moderator, and an inspector.
The second member performs only as an inspector. Ad-hoc is a formal reading technique, which means before the inspection meeting; the inspectors are not necessary to read the code document [8].

Code evaluation may be an iterative process [11]. Correction of bugs is done after they are detected at the time of the reviews, and then another review is planned. The process of correction is continued till all faults are corrected. The output of the code review process is a “fault free “software code document. During the code review process reports of faults are generated. We are implementing iterative code review (ICR) as the subject of this study.

2. LITERATURE SURVEY

In the iterative code review (ICR) process, we are performing sequence of code reviews. The ICR process it may be employed with only one inspector (code executer). This research presents a new method for detecting and correcting the faults remaining in code document inspected by the ICR process. The ICR methods can be used to find the total number of errors in the document [12] [13].

The Iterative code review process not only detects the faults but also corrects it. Recursive procedure is applied for correcting the faults that are detected. The traditional methods for fault detection don’t consider the iterative procedure, i.e. in single iteration faults are detected and corrected. Single iteration method might not detects and corrects all the faults, so the accurate results are not generated which is the drawback of this method. Thus, ICR process is used for fault detection and correction. Iterative code review mechanism is useful where the code contains too many faults, and size of the code document is large. So scheduling the Iterative Code Review process can achieve the fault free code document. Fault detection and the fault correction is main focus of our work. Division of faults is done into different types.

During the different reviews, these faults are corrected. By scheduling code review process, the first iteration detects the simple faults from the given code document and then corrects those faults, then in the 2nd iteration it detects and corrects the medium faults. From the 3rd iteration, it detects and corrects the hard faults. Continuation of this procedure (detection and correction of simple, medium and hard faults) is done till all the faults are detected and corrected from the software code document.

Traditional method of code review process uses a single iteration for detecting and correcting the faults in the code document. Correction of all faults in a single iteration is not guaranteed. So, the accurate results might not be achieved. Traditionally, the number of peoples was required for performing the code review of the software code document. Use of one person as fault detector, and another as fault corrector is done in the traditional method of code review process. Requirement of training is necessary if these peoples don’t have sufficient knowledge of the code document. This is quite expensive process. Iterative behavior is also not possible with existing code review systems.

The current code review process doesn’t classify the faults into simple, medium and hard faults. It just detects faults and some of the methods corrects few faults. These methods do not produce fault free code document. When the number of unobserved faults is adequately small, then a checking of software process satisfactory. Projection of the number of unobserved faults is done with fault insertion methods, which are labor intensive. This is not competent method for fault detection and correction. A number of techniques that avoid fault inclusion have, therefore, been developed [9] [10]. The “capture-recapture” is the most known technique. From the domain of biological studies, capture recapture technique is adapted, where it is used to calculate approximately the size of an animal population in a certain area. To capture the animals, traps are set in the territory. After trapping the animal it is tagged and freed back into the wild. When the tagged animals join with the wild population, then new traps are set. Suppose that the proportion of trapped animals in the wild and the tagged animals is the same, then from the known number of tagged
animals, size of the animal population can be calculated approximately. The Capture Recapture method has several variants and estimation models. In order to apply the above methods to the software checking domain we replace the animals with software bugs, and let the inspectors bring them together independently. By tagging all logged faults as “captured” and all bugs that were found by more than one inspector as “recaptured”, this method can be used to calculate approximately the total number of bugs in the document.

Another method used for detecting faults is Detection Profile Method (DPM) which was proposed in [14]. The DPM finds the number of inspectors that detects each fault, and then classifies the faults according to this number. The approximation of calculation for the total number of faults is done by fitting decreasing exponential curve: \( M_k = a \times e^{-bk} \), where \( M_k \) indicates the total number of inspectors that found a fault \( k \). Calculation of total number of faults is done by the largest \( k \)-value for which this equation provides a result larger than or equal to 1/2. DPM is an unthinkingly appealing technique that can be effortlessly clarify graphically to non-experts. A further study suggested a method for selecting between a DPM and CR model [10].

Another method is based on the subjective spontaneous calculation of the inspectors on the number of remaining faults in the document [15]. Skilled inspectors were asked to calculate the number of remaining faults on the basis of the faults they found by inspection. The results recommend that such estimates are not too bad or satisfactory. Result recommend that such estimates can be improvised and employed as a good opening point. N-Fold approach is introduced by Tsai and Martin is of improving inspection results [16]. The method recommends using N different inspection panels for inspecting the same document. They showed that this method could improve the results. This method is quite expensive because it requires the number of inspection teams for inspecting a code document.

3. OUTLINE OF PROPOSED WORK

The Iterative code review process consists of the following steps:

Factor \( P(i,j) \): This is nothing but probability of fault \( i \) that can be detected by \( j \) iterations.

a. \( P(0,1) \) is the simple faults that can be detected by first iteration. \( P(1,2) \) is probability of medium faults detected after second iteration. \( P(2,3) \) is probability of hard faults detected by third iteration.

\[
P_{i,j} = P_{i,j-1} + (1 - P_{i,j-1}) \times P_{i,1}
\]

(1) [22]

b. Factor Fault Detection Ratio-FDR(\( j \)): Quantity of all errors that can be found by \( j \) iterations.

\[
FDR(\( j \)) \approx FDR_{\text{max}} \left( 1 - \frac{1 - (1 - P_{0,1})^{(j+1)}}{P_{0,1}(j+1)} \right)
\]

(2) [22]

c. \( FDR(\text{max}) \): Ratio between number of faults found after \( j \) iterations and total number of faults in document.

d. Correcting the faults: The faults detected by the above factors are corrected in proposed algorithm.

e. Generation of reports is done
4. ALGORITHM

Proposed algorithm:

1. By scanning a source code document simple faults are detected.
2. On the detected faults, fault correction process will be applied.
3. If (faults present (medium, hard) after step 2 then repeat step number 1 to 2 of the proposed algorithm).
4. Repetition of steps 1 to 3 of the proposed algorithm is done till all faults are corrected from the document.
5. Else generation of review reports is done after every successful review.

5. METHODOLOGY COMPRISING

a. Methods of data collection: The different files of code documents are taken from the user as input code document. All input files are stored in directory.
b. Methods of data analysis: In analysis faults and number of iterations are considered for review. The review results generated by our system is compared with Traditional REVIEW.

![Fig. 1: Proposed framework for fault detection and correction](image)

The above block diagram is for Iterative Code Review process

Review Schedule: In review schedule process the code document is given input. With this process the different reviews can be scheduled.

Fault Detection: In this process different kinds of faults (simple, medium and hard) detected.

Fault Correction: Correction fault in the fault detection process is done.

Iterative Code Review Process: Combination of fault detection process and fault correction process forms Iterative

Code Review Process: In the first iteration of the detection phase, simple faults are detected. The simple faults detected are given to the fault correction process for correction. In the 2nd iteration, another review is scheduled. So the code document is again given for schedule review process, in
this review ICR Process detects and corrects the medium faults. In the 3rd iteration, another review is scheduled for detection and correction of the hard faults.

Continuation of this process is done till all faults in the code documents are corrected.

Review Reports: Review reports are generated after every iteration and stored in the database. Faults free code document generation: Generation of fault free code document is done Not only detect the faults but corrects it.

6. RESULT DISCUSSION

We found no proof in the literature of any organization applying iterative code review process for detecting and correcting faults from the software code documents. Our system requires less time as compare to traditional systems for the fault detection and correction.

Table 1: Time required for finding faults by traditional vs. proposed system

<table>
<thead>
<tr>
<th>Sample number</th>
<th>Traditional System (ms)</th>
<th>Proposed System (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>152</td>
<td>37</td>
</tr>
<tr>
<td>2</td>
<td>140</td>
<td>38</td>
</tr>
<tr>
<td>3</td>
<td>149</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>145</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>144</td>
<td>27</td>
</tr>
</tbody>
</table>

Fig. 2: Graph of time required for finding faults by traditional vs. proposed system

The above graph shows proposed system is more effective than traditional system. In proposed system, source code document when given for Iterative Code Review Process takes less time as compare to traditional system.

Table 2: Faults detected by traditional system vs. proposed system

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Input file name</th>
<th>Traditional system</th>
<th>Proposed system</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sample 1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Sample 2</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Sample 3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Sample 4</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>Sample 5</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>
7. CONCLUSION

Traditional N Fold Method code review system for code inspection does not give accurate results after inspecting a code document. When size of the code document is large, Fagan inspection method does not provide efficient results. The Iterative Code Review system works well for the java source code document. Iterative code review system divides the faults into different categories and then corrects the faults. The review reports are generated after each Iterative Code Review Process. In this way we developed Fault Tolerant System. We have improved accuracy of the Software document.

REFERENCES


