
N. Kamaraj  
Assistant Professor and Head of B.Sc IT, Sri Ramakrishna Mission Vidyalaya College of Arts and Science, Coimbatore, Tamil Nadu, India  
nkamaraj17@gmail.com

A.V Ramani  
Associate Professor and Head of Computer Science, Sri Ramakrishna Mission Vidyalaya College of Arts and Science, Coimbatore, Tamil Nadu, India  
avramani@yahoo.com

Abstract: Bug localization is the task of determining the source code which entities are relevant to a bug report. It is an important task of classification in software data set resources. Manual bug localization is labour intensive since developers must consider thousands of source code entities. To builds bug localization classifiers, based on information retrieval models, to locate entities that are textually similar to the bug report. Current research, however, does not consider the effect of classifier configuration. Designers uses data to detect a bug located in the segment of the source code to correct the bug. They are several categories are used, i.e., code smells detection and pattern clustering. To identify the numerous semantic relations existing between two given words, a pattern clustering algorithm has been proposed. This survey proposes research analysis on the bug localization classifiers based on information retrieval models to locate entities that are textually alike to the bug report.

Keywords–Information retrieval, Code smells, Bug localization, Information extraction, Clustering, Classifiers

I. INTRODUCTION

Bug localization, a developer uses information about a bug to locate the portion of the source code to modify and to correct the bug. Some recent static techniques for automating bug localization have been built around modern Information Retrieval (IR) [1] models such as Latent Semantic Indexing (LSI) [2] and its probabilistic extension LSI [3]. IR-based static techniques aim to identify the elements of the software system that need to be modified to correct a bug.

From the standpoint of retrieval from large software libraries for the purpose of bug localization, five generic text models are compared: Unigram Model (UM), Vector Space Model (VSM), Latent Semantic Analysis Model (LSA), Latent Dirichlet Allocation Model (LDA), and Cluster Based Document Model (CBDM) [12].

Measuring the semantic [16] between words is an important component in various tasks on the web such as relation extraction, community mining, document clustering and automatic metadata extraction. The analysis is an empirical method to estimate semantic similarity using page counts and text snippets retrieved from a web search engine for two words.

Bug localization is the function of condition which source code operation is relevant to a bug report. Manual bug localization is labor in depth since developers must consider thousands of source code operations. Current research builds bug localization classifiers based on information retrieval models, to locate entities that are textually similar to the bug report [5].

II. METHODOLOGIES

2.1 Source Code Retrieval for Bug Localization

Source code retrieval techniques operate on models of the source code that are constructed from semantic information embedded in the code, including identifiers and comments and serve as the documents that are provided as input to an Information Retrieval (IR) technique [1].

IR-based static techniques aim to identify the elements of the software system that need to be modified to correct a bug. To perform Latent Dirichlet Allocation (LDA)-based bug localization on a given version of a software system, first an LDA model of the source code is built. Then, the created model is queried as often as necessary to localize bugs exiting in that version [4].

The goal of the case studies in bug localization is to measure how well an LDA-based system can predict the methods that likely need modification to correct a given bugs. There are two case studies was applied. The first case
study, the analysis of Rhino wills the LDA-based technique [3] perform well across all bugs in the system. The second case study, invokes the application of LDA-based technique to the Eclipse software system, an open source IDE written in Java.

LDA can successfully be activated to source code recover for the purpose of bug localization. Furthermore, the results suggest an LDA-based approach is more effective than approaches using LSI alone for this task. It is planned to examine more bugs in larger software systems, such as Eclipse and Mozilla, to determine how the approach works across a large number of bugs in those systems [1].

2.2 Retrieval from Software Libraries for Bug Localization

While most existing static and dynamic bug localization techniques that may be thought of as bug detection or prognosis tools, an IR-based bug localization technique is best considered as a post diagnosis (debugging) tool [12].

IR-based bug localization uses the VSM model to represent the source code. That contribution used the JEdit software to demonstrate that the patch files that were posted on the bug tracking system were a subset of the files extracted by the IR tool. Use of the Latent Dirichlet Allocation model for source code retrieval for bug localization was explored in [13].

The VSM model is the simplicity of the computations that go into model construction and the ease with which a query can be compared with the documents. Its main disadvantage is the generally large dimensionality and sparseness of the document vectors. The LSA (Latent Semantic Analysis) model attempts to reduce the large space spanned by the vectors of the VSM model.

The Cluster-Based Document Model (CBDM) is another relatively popular approach for text modeling [14]. CBDM is also workable to merge some of the generic models to form what are generally more powerful compound text models. A query is compared with the documents in a model for the purpose of retrieval. Each model has its own way of representing a query.

IR-based bug localization researchers have employed “rank of retrieved files” metric to analyze the performance of their retrieval algorithms [15]. IR-based bug localization tools with those that carry out dynamic bug localization could significantly improve the state of the art in bug localization.

2.3 Web Search Engine-based approach to Semantic Similarity

Danushka Bollegala is proposed an automatic method to estimate the semantic similarity between words or entities using web search engines. Because of the vastly numerous documents and the high growth rate of the web, it is time-consuming to analyses each document separately [16].

Given taxonomy of words, a straightforward method to calculate similarity between two words is to find the length of the shortest path connecting the two words in the taxonomy [17]. A word is polysemous and then multiple paths might exist between the two words. In such cases, only the shortest path between any two senses of the words is review for calculating sameness.

Cilibrasi and Vitanyi proposed a distance metric between words using only page counts retrieved from a web search engine. The snippets get back by a search engine for the conjunctive query of two words supply useful clues related to the semantic relations that exist between two words [18].

Typically, a semantic relation can be expressed using more than one pattern. Identifying the different patterns that express the same semantic relation enables the user or researcher to represent the relation between two words accurately. According to the distributional hypothesis [19], words that occur in the same context have similar meanings. The distributional hypothesis has been used in various related tasks, such as identifying related words, and extracting paraphrases [20].

By sorting the lexical patterns in the descending order of their frequency and clustering the most frequent patterns first, it form clusters for more common relations first. This enables us to separate rare patterns which are likely to be outliers from attaching to otherwise clean clusters [21].

A semantic similarity measure is evaluated using the correlation between the similarity scores produced by it for the word pairs in a benchmark data set and the human ratings. Both Pearson correlation coefficient and Spearman correlation coefficient have been used as evaluation measures in the previous work on semantic similarity. Spearman correlation is more appropriate for evaluating semantic similarity measures, which might not be necessarily linear. In fact, most semantic similarity measures are nonlinear.

2.4 GUI and Web Applications

A GUI is the front-end to software’s underlying back-end code. An end user interacts with the software via events [22]; the software responds by changing its state, which is usually reflected by changes to the GUI’s widgets.
Researchers have shown that this type of GUI testing finds faults related not only to the GUI and its glue code, but also in the underlying business logic of the application [23]. Present techniques used in implementation to test such GUIs are mostly manual. The most popular tools used to test GUIs are capture/replay tools such as WinRunner that provide very little automation [24], especially for creating test cases.

Web application testing is defined as exercising the entire application code by generating URL based inputs with the intent of finding failures that manifest themselves in output response HTML pages. Testing of Web program code to locate liability in the program is mostly a manual task. Capture-replay tools capture tester interactions with the application and are then replayed on the Web application [25].

GUI and Web systems consistently support changes as part of their maintenance process. New versions of the applications are often created as a result of bug fixes or requirements modification. Web application GUI concern act differently. That is, some widget actions are handled at the client (e.g., in the form of JavaScript code in the browser) [22].

In spite of GUI and Web applications were modeled in previous research. These two classes of applications have many sameness. These similarities to generate the single model for test suite order of both GUI and Web applications.

A test case is modeled as a chain of actions. For each action, a user sets a value for one or more variables. Prioritization function takes as input a set of test cases to be ordered and returns a sequence that is ordered by the prioritization criterion, because they have developed a unified model of GUI and Web applications [23].

One feature of event-driven software and test cases which also belongings for other types of systems is their province on parameters and values for execution. Relation between multiple parameter values make the program follow a decided execution path and are probable to expose faults in the system. This basic premise led to the development of the first set of prioritization criteria, which are based on giving higher priority to test cases with a large number of parameter values interactions.

One way and Two way parameter value interaction coverage techniques, select tests to systematically enclose parameter value interactions between windows. The final set of criteria gives higher priority to test cases that cover windows that are perceived to be important to the EDS from a testing perspective [26].

Frequency based criteria, importance of a window as the number of times the window is accessed in the test cases. Because of the user-basic design of event-driven software, these windows are probable to contain more code functionality (and likely to contain more faults), and thus test cases that cover.[26].

In all of the implementations, in case of a tie between two or more tests that meet the prioritization criterion, a random tie-breaking strategy is implemented using the RANDOM() function. In addition, a greedy optimal ordering (G-Best) uses a greedy step to select the next test case that detects the most yet-undetected faults and repeat this process until all the tests are selected. A greedy worst ordering (G-Worst) criterion, where in each iteration, the algorithm selects the next test case that covers the least uncovered faults. It repeats this until all of the tests are selected. Ordering by Random selects a next test uniformly at random [26].

2.5. Information Retrieval Model on Bug Localization

Information retrieval is the study of querying for text within a collection of documents [4]. IR-based bug localization classifiers use IR models to find textual similarities between a bug reports (i.e., query) and the source code entities (i.e., documents). IR-based classifiers contain several parameters that control their behavior. Specifying a value for all parameters fully defines the configuration of a classifier [5].

The Vector Space Model (VSM) is a simple algebraic model based on the term-document matrix of a corpus [6]. The term-document matrix is an m*n matrix whose rows represent individual terms, i.e., words and columns represent individual documents. Latent Semantic Indexing (LSI) is an extension to VSM in which singular value decomposition (SVD) is used as a means to project the original term-document matrix into three new matrices: a topic-document matrix D, a term topic matrix T, and a diagonal matrix S of eigenvalues [7].

Latent Dirichlet Allocation is a popular statistical topic model that provides a means to automatically index, search, and cluster documents that are unstructured and unlabeled [8].

Stephen W. Thomas and Meiyappan Nagappan proposed a configuration framework for analyzing the various configurations of a classifier. They used categorical and numerical parameters for their configuration. [32].

They analyzed the performance of the configuration with two different goals. One is to determine the best and worst configuration and the other is to determine which parameter values are most effectively [10].

Their research aims at measuring features of the source code, such as lines of code (LOC), past bug proneness, change proneness, and logical coupling between classes, to predict which source code entities contain bugs [9].
The best individual IR-based classifier uses the Vector Space Model, with the index built using tf-idf term weighting on all available data in the source code entities (i.e., identifiers, comments, and past bug reports for each entity), which has been stopped, stemmed, and split, and queried with all available data in the bug report (i.e., title and description) with cosine similarity [10].

2.6. Code-Smell Detection

Code-smells are also called design anomalies or design defects, refer to design situations that adversely affect the software maintenance. As stated by, bad-smells are unlikely to cause failures directly, but may do it indirectly [27].

The code-smells’ detection process consists of finding code fragments that violate structure or semantic properties such as the ones related to coupling and complexity. In this setting, internal attributes used to define these properties, are captured through software metrics and properties are expressed in terms of valid values for these metrics [28].

Optimization problems are often hard and expensive from a CPU time and/or memory viewpoint [29]. The use of Meta heuristics, such as Evolutionary Algorithms and Particle Swarm Optimization (PSO), allows reducing the computational complexity of the search process. Evolutionary algorithm is based on

- Genetic Programming (GP) to generate code-smells detection rules and also executed in parallel is Genetic Algorithm (GA) that generates detectors (code smells examples) from well-designed code examples [30].
- GP evaluates detection rules from code-smell examples (input) and GA evaluates in parallel to generate detectors from well-designed code (input) using global and local alignment techniques [31]. In the initialization of the Parallel Evolutionary Algorithm (P-EA), one system from the base of examples is selected randomly to calculate the intersection function [33].
- GP studies that have newly intense on detecting code-smells in software using different techniques. These techniques range from fully automatic detection to guided manual inspection.

There are seven broad categories: manual approaches, symptom-based approaches, rule-based approaches, probabilistic approaches, visualization-based approaches, search-based approaches and cooperative based approaches [34].

### III. ANALYSIS

Table I: Analysis of various literature surveys as shown in the below table.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Author</th>
<th>Year</th>
<th>Techniques</th>
<th>Methods</th>
<th>Findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stacy K. Lukins, Nicholas A. Kraft, Letha H. Etzkorn</td>
<td>2008</td>
<td>Latent Semantic Indexing (LSI)</td>
<td>Information retrieval</td>
<td>Improve the LDA</td>
</tr>
<tr>
<td>2</td>
<td>Shivani Rao, Avinash Kak</td>
<td>2010</td>
<td>Vector Space Model (VSM)</td>
<td>Bug localization</td>
<td>Rank of Retrieval files</td>
</tr>
<tr>
<td>3</td>
<td>Danushka Bollegala, Yutaka Matsuo, Mitsuru Ishizuka</td>
<td>2011</td>
<td>Lexical Pattern Extraction</td>
<td>Web Text analysis</td>
<td>Information Extraction</td>
</tr>
<tr>
<td>5</td>
<td>Stephen W Thomas, Dorothea Blostein, Ahmed E Hassan</td>
<td>2013</td>
<td>Singular Value Decomposition (SVD)</td>
<td>Configuration Framework</td>
<td>Classify the bugs</td>
</tr>
<tr>
<td>6</td>
<td>Wael Kessentini, Marouane, Houari Sahraoui, Slim Bechikh</td>
<td>2014</td>
<td>Code smells detection</td>
<td>Optimization process</td>
<td>New search based approach</td>
</tr>
</tbody>
</table>

Comparison of various models - Literature Survey

### IV. CONCLUSION

Resolving the bug localization problem has major implications for programmers or developers, because it can reduce the time and effort required to maintain or manage software. Classifier combination helps in almost all cases. No matter the underlying classifiers are used or the specific combination technique is used.

The IR-based classifier uses the Vector Space Model, with the index built using tf-idf term weighting on all available data in the source code entities (i.e., identifiers, comments, and past bug reports for each entity), which has been stopped, stemmed, and split, and queried with all available data in the bug report (i.e., title and description) with cosine similarity

© www.ijarcsse.com, All Rights Reserved
Future work for Lexical Pattern Extraction Algorithm has been used to extract numerous semantic relations existing between two words. Moreover, a Sequential Pattern Clustering Algorithm is to identify different Lexical Patterns that describe the same semantic relation. Both the page counts-based co-occurrence measures and Lexical Pattern Clusters would be used to define features for a word pair.

REFERENCES


