Change Impact Analysis by Concept Propagation

Zeinab Mahzoon\textsuperscript{a,}\textsuperscript{*}, Omid Bushehrian\textsuperscript{b}

\textsuperscript{a}Shiraz university of technology,\textsuperscript{b}Shiraz university of technology.

\textbf{A R T I C L E I N F O.}

Article history:
Received: 14 July 2018
Revised: 29 December 2018
Accepted: 14 January 2019
Published Online: 25 June 2019

Keywords:

\textbf{A B S T R A C T}

Software maintenance is an important phase of the software life cycle. An important task in this phase is to locate code fragments affected by user change requests. However, performing this task manually is costly and requires prior knowledge of the software structure. In previous studies, Latent Semantic Indexing (LSI) has been applied to map the user change queries to the relevant code segments automatically. However, due to the lack of domain knowledge embedded in the source code, LSI might be unable to perform this mapping accurately. In this paper, we have proposed a domain knowledge propagation method to obtain more relevant impact set for each change request. This method spreads the user interface originated domain knowledge to the program classes according to the program dependency graph. The proposed method has been applied to ArgoUML case-study which is an open-source project associated with its change requests. It was observed that applying the concept propagation resulted in 5\% increase in the accuracy of the plain LSI method.

\textsuperscript{*}Corresponding author.

Email addresses: m.z.mahzoon@gmail.com (Z. Mahzoon), Bushehrian@stu.teh.ac.ir (B. Bushehrian)

https://dx.doi.org/10.22108/jcs.2019.111708.1007

ISSN: 2322-4460 © 2019 JComSec. All rights reserved.

1 Introduction

Software maintenance is an important phase of the software life cycle and Impact Analysis is a vital task in this phase\textsuperscript{1}. Impact Analysis is the task of locating particular program features, documented in the change requests or bug reports, in the source code\textsuperscript{2}. Later, Information Retrieval (IR) methods have been employed by many studies to map change request queries to the code parts\textsuperscript{2,3,4}. This method is applied to the raw source code files containing language keywords, identifiers and probably some textual information such as programmer inserted comments and produces a set of prepared documents represented in a vector space format. Moreover, the user change request is also represented as a vector in order to be matched against the documents. Finally, a ranked list of documents (program classes) is reported by calculating the similarity between the user change request and the prepared documents\textsuperscript{5}. Here, LSI\textsuperscript{5}\textsuperscript{2,2} method is used to produce the ranked list of documents based on a similarity order.

The success of the LSI technique is highly dependent on the amount of domain knowledge associated with the code parts provided by the programmers, however, in a real development environment, there is no guarantee that this information exists. In this paper, we proposed a method to enrich the code by propagating the domain concepts extracted from user interface classes among program classes. This concept propagation increases the accuracy of the LSI method in finding the correct impact set.

In this study, textual change requests and program source code are the main inputs. By applying IR
method the parts of the code that are most likely to be changed during the fixing the reported bug are located. However, original source files usually lack the adequate domain knowledge to be used by IR method and adding the user interface information to the source files can effectively increase the accuracy of the matching algorithm. The rest of the paper is organized as follows: The related works are summarized in Section 2. Section 3 provides details of the proposed approach while Section 4 presents the evaluation and discusses obtained results. Finally, Section 5 concludes the paper with the final remark and future works.

2 Related works

Change impact analysis algorithms are widely studied in the following three categories [1]. The first category is dependency-based algorithms that are known as the oldest and the easiest to use impact analysis methods applying the program dependency graph to determine which parts of the source code are dependent [7, 8]. The idea is that when a code segment is changing, all the relevant parts are also changed. Traditional dependency-based algorithms have improved by using Bayesian Network Model [9] or network decisions [10] models. In some cases, powerful tools have been used to generate dependency graph efficiency [11, 12]. Although these algorithms have been proved to be efficient, they are unable to solve the feature location problem when the inputs are in natural language format [13, 14].

The second category is mining repository approaches which are the most popular and most efficient impact analysis methods. These algorithms mine previously applied change reports to extract the patterns and determine which parts of the code are always changed together [15, 16]. To improve the accuracy of the impact analysis phase, association rules are extracted from the history of past changes [17], however these methods are highly data-dependent and are useless in the cases that historical data are not available. These algorithms have very high accuracy in the presence of rich historical data but are not effective in finding parts of code that may change when another part changes [18].

The third category applies the Information Retrieval (IR) algorithms. In this category, the textual change requests reported by the users with a limited level of information about the system are focused. Hence, the main focus of the IR-based algorithms is on the detection of similarities among textual change requests and the code fragments.

In [6] an Information Retrieval method in text processing is used to find relationships between a set of textual documents by measuring their similarity considering the terms in the documents. The subject of concept location in impact analysis was explained in 2004 by Andrian Marcus et al. [2]. The authors used the LSI method to map user change request represented in natural language to the part of source code and then measured cosine similarity of the prepared documents to determine a ranked list of relevant parts. Since their method relied mostly on the internal comments of the source code when applying the LSI method, the main shortcoming was the lack of common terms between user queries and code parts in text processing which resulted in inefficient matchmaking in LSI method.

In [3] a method that used the previous history of change requests as input LSI documents has been proposed. Authors have proposed to rank past change requests based on their similarity to the new request to assign the impact sets to the new request. This method improved traditional LSI accuracy but it is a data-dependent approach which is unable to find the impact set in absence of historical information. In [1] historical information extracted from software repositories (including Control Version System (CVS) logs and past Change Requests (CRs) descriptions) has been combined with source code and internal comments to improve concept location phase but this method only succeeds when historical data is available. In [20] a method to improve feature location in lack of historical information (only by software source code) has been presented. Authors used Stereotypes to decrease the gap between user queries and source code parts. Stereotypes are the terms in natural language that describe a method’s functionality in a class. Preparing method stereotypes are easy for a small number of classes and methods however it is costly to do in complex systems and needs some preprocessing on the source code. This method is programming language dependent as well.

In summary, the previous studies mostly ignore the fact that programmers typically do not add enough comments and semantic information to the program. Accordingly, the gap between code segments and user queries which is inevitable in traditional LSI algorithms has to be handled by new approaches. Hence, we proposed a method to decrease such a gap without requiring any historical data and only relying on the domain knowledge latent in the internal structure of the program code.

3 Concept Propagation

In this paper, we have proposed a method to improve traditional change impact analysis techniques by aug-
menting domain concepts to the source code. This augmented knowledge is extracted from the software user interface which mostly contains text in natural language which refer to domain terms. There are three phases in the proposed method as shown in Figure 1: (1) Preprocessing (2) Applying LSI (3) and Similarity Measurement. The novelty of this study is the proposed preprocessing method that extracts UI concepts and propagates them among the relevant classes prior to applying the LSI method and measuring the similarity of documents.

### 3.1 Preprocessing and Propagation

In this phase, first, the source code and associated comments are divided into documents at class granularity level. The created text files are refined by performing textual and conceptual pre-processing in which UI concepts are extracted from UI class and added to relevant classes and its internal comments by propagating the concepts throughout the class dependency graph.

#### 3.1.1 Textual and Conceptual Pre-Processing

In this step the irrelevant words and symbols including special characters such as (!?<>;|‘#“@!/+ - . . ), programming-language keywords (public, void, main, private new,) and famous English stop words (this, there, that, too, and, or) are removed. The next filtering step is to split the multi-parts words such as camelCase, CamelCase, CAMELCase or camelCase and add the resulting parts to the document as recommended in [2]. Subsequently, domain terms are extracted from the UI classes by extracting the component labels such as buttons, labels, warning messages and other texts displaying to the users during their interaction with the software.

#### 3.1.2 Propagation

In this phase, the domain concepts extracted from a UI class are propagated to the classes which are part of invocation chains starting from that class. The aim is to enrich the classes with domain terms based on their relevancy to a specific software use-case. To achieve this goal, extracting the program class dependency graph is essential. A class dependency graph CDG(V,E), in which V denotes the set of program classes and E denotes the set of invocation or object creation relations among classes, is created by scanning the source code and locating method calls and object creation statements. In the dependency graph, CDG the user interface classes identified in the previous step are considered as starting nodes for concept propagation. The propagation algorithm is listed in Listing 1.

As shown in Listing 1, the `propagateConcept` function (line 1) is called for all UI nodes in the given CDG to propagate each node domain terms along the subsequent classes reachable from the UI node (lines 2-3). The `addConcept2Node` function (line 5) recursively traverses the CDG nodes starting from a given UI node and adds the domain terms extracted initially from the UI node to each visited node during the traversal (lines 8-9). The density parameter controls the number of times that a domain term has to be added to a class and is decreased gradually along the invocation chain based on the presumption that the further a class is from a UI class, the less relevant is to that UI class (lines 10-11).

![Figure 1](image-url). The steps of the proposed method.

Listing 1: The Concept Propagation Algorithm.

```
1. propagateConcept(CDG g, int density){
2.    for each UI node N in g{
3.        propagateConcept(g, density)
4.    }
5.    addConcept2Node(CDG Node N, int density, String DomainTerms){
6.        if (N.visited || N is null)
7.            return;
8.        for (int n=1; n<=density; n++)
9.            N.addDomainInfo(DomainTerms);
10.        for (int j=0; j<N.neighbors.size(); j++)
11.            addConcept2Node(N.neighbors[j], density-1, DomainTerms);
12.    }
13.    addConcept2Node(CDG Node N, String DomainTerms){
14.        for (int j=0; j<N.neighbors.size(); j++)
15.            N.neighbors[j].addDomainInfo(DomainTerms);
16.    }
17.}
```

Figure 2 illustrates an example of concept propagation in a hypothetical CDG starting from two UI nodes A and B. The intensity value of domain terms initialized to 3 is shown in the Figure as the domain term labels. The `propagateConcept` function starts from nodes A and B respectively. Corresponding to each UI node the CDG nodes reachable from that node are visited recursively using the `addConcept2Node` function and the concepts are added to the visited node considering the current intensity value.

### 3.2 Running LSI

The LSI method[6] consists of the following three steps (1) Creating a term-document matrix(TDM) (2) Applying singular-value decomposition(SVD) and
(3) Query formulation. The line of codes, internal comments and UI concepts documented in a class form a document in LSI and the set of documents are referred to as LSI corpus. Corpus is tokenized and all words are extracted to create TDM. In this matrix element TDM[i][j] stores the repetition value of ith term in the jth document. Then the LSI vector space is optimized using the SVD algorithm which is a statistical algorithm for deriving a set of uncorrelated indexing variables or factors to reduce the size of TDM [6]. The aforementioned steps are also repeated for the user requests to create the query vectors.

3.3 Measuring Similarity

The last phase is the measurement of the similarity between user queries and documents of the corpus. The output of this phase is a ranked list of documents based on their relevance to the query.

\[
\text{Similarity}(y_q, y_i) = \cos(v_q, v_i) = \frac{v_q \cdot v_i}{|v_q| \times |v_i|} \tag{1}
\]

Where \(v_q\) and \(v_i\) are user query and the document against which the similarity is measured. \(v_q \cdot v_i\) denotes the inner product of two vectors and \(|v_q|\) denotes the length of the vector [2]. Accordingly, relevant classes are ranked based on their similarity measure to the user query to create the final impact set.

4 Experimental Results

All the explained steps have been implemented in Java language to automate the impact analysis process. We used the SEMERU\[1\] dataset consists of five software projects [21]. Each project in this dataset is associated with user change requests and their actual impact sets corresponding to each request. Among them, ArgoUML which is a drawing tool for UML diagrams written in Java that contains 1421 classes has been selected to evaluate the effectiveness of the proposed method.

The value of \textit{intensity} parameter in the proposed propagation algorithm was chosen three experimentally. It was observed by this value the best results are obtained. This parameter controls the propagation depth along any given UI path. The higher the value of this parameter the concepts stemmed from UI nodes reach further classes in the CDG.

We applied the LSI algorithm to create the ranked list of classes corresponding to each change request of ArgoUML first without concept propagation (base method) and then with concept propagation (proposed method). To compare the accuracy of resulting ranked lists, the rank of each class was looked up in the generated ranked list of both “base” and “proposed” methods corresponding to each user request. To compare the results, the following accuracy formula was defined:

\[
\text{Accuracy} = (1 - p/n) \tag{2}
\]

Where \(p\) denotes the rank of the first actual impact set class in the generated ranked lists and \(n\) denotes to the number of documents in the ranked list. We computed the accuracy for all the 56 queries and the results are illustrated in Figure 3. It was observed that in 73% of queries, the proposed method produces the more accurate list of classes and in 16% of queries, both methods are identical. It was observed that the average accuracy in this case study was 0.90 for the proposed method and 0.852 for the base method.

Though the proposed method succeeded to obtain higher accuracy in average, it was not able to outperform the base method in all bug reports. This observation implies that this assumption that: “the more similar classes to the bug report are the ones that really are changed by the programmer in the production environment” is not necessarily true. For instance an abstract class which becomes very similar to the bug report after concept propagation is probably not the class that needs to be fixed with good chance. Hence, as the future work we intend to tackle this issue and consider other factors that may affect the chance of modifying a class in addition to the similarity factor.

5 Conclusions and Future Works

In this paper, an LSI based impact analysis method was proposed that augments domain knowledge to the program source code in the preprocessing phase to increase the accuracy of the resulting impact set for each user query. The domain concepts were extracted

---

\[1\] Software Engineering Maintenance and Evolution Research Unit
from the knowledge-rich parts of the source code which are UI classes and were propagated to all relevant classes according to the dependencies in the program dependency graph. The proposed method was applied to the ArgoUML case study. It was observed that the LSI method when the source code documents were extended by the UI domain terms generates more accurate impact sets.

As the future work, we aim to increase the accuracy of the impact set by first applying the past change history of the software and second limiting the number of classes to be matched against a user query by filtering out the irrelevant classes during the preprocessing phase.

References


Zeinab Mahzoon received the B.Sc in IT engineering from Shiraz University of Technology, Iran, the M.Sc degree in software engineering from the Shiraz University of Technology, Iran. She is currently a software developer.

Omid Bushehrian received the B.Sc in software engineering from AmirKabir University of Technology (Tehran Polytechnics), Iran, the M.Sc and Ph.D degrees in software engineering from the Iran University of Science and Technology, Iran. He is currently a faculty member at Shiraz university of Technology, Iran. His main research interest is cloud computing and software quality.