2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) | 978-1-6654-9590-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSE-SEIP55303.2022.9793550

A Software Impact Analysis Tool based on Change History Learning and its Evaluation

Haruya Iwasaki Tsuyoshi Nakajima
Department of Computer Science and Engineering
Shibaura Institute of Technology
Tokyo, Japan
{ ma20501,tsnaka}@shibaura-it.ac.jp

Ryota Tsukamoto Kazuko Takahashi Shuichi Tokumoto
Information Technology R&D Center
Mitsubishi Electric Corporation
Kamakura, Japan
{ Tsukamoto.Ryota@dy, Takahashi.Kazuko@dx,
Tokumoto.Shuichi@dr}.MitsubishiElectric.co.jp

ABSTRACT

Software change impact analysis plays an important role in controlling software evolution in the maintenance of continuous software development. We developed a tool for change impact analysis, which machine-learns change histories and directly outputs candidates of the components to be modified for a change request. We applied the tool to real project data to evaluate it with two metrics: coverage range ratio and accuracy in the coverage range. The results show that it works well for software projects having many change histories for one source code base.

1 INTRODUCTION

In case that a lot of small projects that make small changes to a large source code base continuously occur, the accuracy and efficiency of impact analysis on change requests are crucially important because it determines the quality and productivity of such projects. Impact analysis is the task of determining the extent to which a change request affects when implemented [1].

We developed an impact analysis tool, which machine-learns change histories to generate a list of candidates of program components to be modified from a change request. The change history to be machine-learned includes a pair of a change request in natural language and a list of the corresponding modified components. This tool was applied to real project data and evaluated with two metrics: coverage range ratio and accuracy. The result shows that the tool works well for the software projects that have change histories are accumulated for the same program base.

2 PROPOSED IMPACT ANALYSIS TOOL

ICSE-SEIP '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9226-6/22/05…$15.00
https://doi.org/10.1145/3510457.3519017

2.1 Characteristics of target projects

In software development of high-mix and continuously evolving products (called high-mix small-change development), thousands of changes are made to the relatively large-scale source code base in order to periodically add new functions to it and customize them for different hardware sets and various shipping destinations. Many small projects run in parallel, with shortage of programers having knowledge about the source code base enough to do impact analysis accurately and efficiently.

The target project group adopts the same derivative development, in which one change design document is created for every change request including a requirement in natural language and its implementation of changes on software design specification and source code. About thirty projects occur every year, each of which has ten change requests on average. Nearly a thousand change design documents have been created for three years. The source code base consists of 32 components.

2.2 Approach and its implementation

Because the target projects have a lot of change design documents, we thought it may be possible to infer modification candidates from the new change request by learning these documents. Based on this thought, we developed an impact analysis tool, which inputs a change request and outputs a list of modification candidates in descending order of likelihood of occurrence.

Figure 1 shows the configuration of the tool. The tool firstly extracts the text of the change request written in Japanese from the change design document and then translates the text into a vector (called requirement vector) using a word-embedding technique, and secondly extracts the names of the modified components and then convert it into the component vector, each element of which is 1 when the component is modified due to the change request / 0 when not modified). The machine-learning component inputs requirement vector and outputs the component vector.

To vectorize change requests, we use Doc2vec bundled in genism [2] and MeCab [3] as a Japanese morphological analyzer. To generate the corpus for them, we use not only Wikipedia in

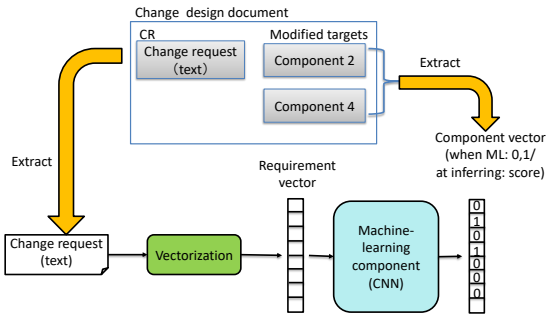

Figure 1 Configuration of the proposed tool

Japanese but all the change requests, so that they can handle all the words in change requests.

As a machine learning component, we used a convolutional neural network (CNN) with four hidden layers. The requirement vector has 100 dimensions, and the component vector has 32 dimensions corresponding to the number of the existing program components.

Considering CNN as a classifier, this problem comes down to the multi-label classification. In other words, it is a problem of selecting multiple labels for one input pattern. In this case, CNN can handle this type of problem using sigmoid as the output function and binary cross entropy error as the loss function. The following hyper parameters for the CNN are used: Number of epochs is 50, Batch size is 50, and Learning rate is 0.1.

We use the sigmoid values of the component vector as its score, and the modification candidates are sorted in descending order of the score to display. This allows developers to review components with a higher likelihood first.

3 EXPERIMENTAL EVALUATIONS

3.1 Target projects and their data

The target projects are for developing software embedded in mass-produced products that undergo model changes every year, which are typical high-mix small-change development ones. About 30 projects run annually on the same source code base, and each project implements about 10 change requests on average. Each project has multiple change requests, creates a change design document for each, and implements them by modifying the source code based on the documents.

3.2 Evaluation method

In this evaluation, 405 modified design documents are used, including 325 for training data, and evaluated with 80 test data. This testing was carried out 5 times in different training/test combinations. The evaluation is based on the average of the results.

3.3 Metrics for evaluation

Our tool needs to be evaluated from the following two points. One is for the modification candidates not to miss modification targets as much as possible. This is because overlooking some modification targets may result in incomplete or inconsistent implementation of the change request. The other is to reduce the number of modification candidates as much as possible. This is because reducing the number of components to review will make the task for determining the modification targets more efficient.

To evaluate from these two points, we use two metrics: coverage range ratio and accuracy in the coverage range.

The coverage ratio *Rc* is the average percentage of the components to be reviewed in the source code base for a change request, and the closer it is to 0, the better the evaluation.

$$R_c = Average\left(\frac{C}{W}\right) \quad (1)$$

C : rank of the last correct component in the list
W : length of the whole list

The accuracy in the coverage range *Ac* is the average percentage of the actually modified targets included in the coverage range, and the closer it is to 1, the better the evaluation.

$$A_c = Average\left(\frac{T}{C}\right) \quad (2)$$

T: number of components to be changed for the change request

3.4 Results of experiment

The results of the experiment are:

- *Rc* is 16.6%, which means that the modification targets can be covered by reviewing about 1/6 of the total number of components for one change request on average.
- *Ac* is 67.1%, which means about 2/3 of the reviewed components are right ones. Thus, it is found that our tool can significantly reduce the workload of impact analysis.

4 CONCLUSION AND FUTURE ISSUES

We developed a tool that machine-learns the histories and outputs a ranking list of the components to be modified from a change request. Furthermore, its performance was evaluated with the two metrics of the coverage range ratio and the accuracy in the coverage range. To evaluate this tool, we applied it to the data from a group of actual high-mix, small-change software development projects, and the results show its good performance. This indicates that the tool is likely to be applied with high accuracy to a group of software projects that record many change histories for the same source code base, such as OSS projects where many developers are constantly modifying using issues, and ticket-driven derivative development projects where a change request is handled as a ticket.

REFERENCES

[1] Sunil Sikka and Ankit Dhamija: Software Change Impact Analysis, BookRix (2020)

[2] Genism, https://radimrehurek.com/gensim/（Retrieved on Oct.15, 2021）

[3] Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. 2004. Applying conditional random fields to Japanese morphological analysis, In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2004), volume 2004.