A Study of Project Description Inference Using Method Name Elements for Software Upcycling

Kohei Terakawa [1], Sinan Chen [2], Sachio Saiki [3], Masahide Nakamura [2]
[1]Graduate School of System Informatics, Kobe University, 1-1 Rokkodai-cho, Nada, Kobe, 657-8501, Japan
[2]Center of Mathematical and Data Sciences, Kobe University, 1-1 Rokkodai, Nada, Kobe, 657-8501, Japan
[3]School of Data and Innovation, Kochi University of Technology, 185 Miyanokuchi, Tosayamada, Kami, Kochi, Japan
Email: odajin@ws.cs.kobe-u.ac.jp, chensinan@gold.kobe-u.ac.jp, masa-n@cmds.kobe-u.ac.jp, saiki.sachio@kochi-tech.ac.jp

***Abstract*—In software development, there is often a situation where products developed in the past could be managed as more valuable assets. Conversely, such products may contain the reusable value. We are currently exploring utilizing existing project assets for new development, a concept we define as "software upcycling". To facilitate upcycling, the challenge lies in comprehending the software overview without documentation and making it easily referenceable. In a previous study, we proposed a methodology for inferring the system overview using a project corpus. We collected constituent words from class names and conducted a validation study to assess their utility in inferring the system's overview. In this research, we shift our focus to method names in order to further enhance accuracy. We attempt to understand the software's architecture efficiently by assigning weighted importance to words in the project corpus and visualizing them through Tag cloud images. We apply our proposed methodology to 136 Java projects managed within our laboratory and evaluate its effectiveness from two perspectives: the functions provided by the software and the technologies that enable the software to operate.**

***Index Terms*—software development, technical debt, upcycling, corpus, tag cloud, mining software repositories**

I. Introduction

In recent years, software products have been utilized in various places due to the development of digital technology. The environment surrounding software is changing rapidly, and software is gradually accumulating technical debt [1] due to technological progress and changes in business requirements. As a solution to this situation, we have been studying software upcycling [2]. Software upcycling is a concept that aims to improve development efficiency and utilize existing knowledge for new software development by applying asset values hidden in existing projects.

In contrast, repositories in the software development field are not maintained and managed as assets. Often, documentation related to the product, such as $README$, needs to be included or more. In such cases, developers need help utilizing the valuable design and implementation ideas in the product [3]. Hence, the challenge of software upcycling is to develop documentation for non-documented software and make it easy to refer to.

In our previous study, we proposed a system overview inference method using a project corpus ($corpus_{Pj}$) [4]. Assuming that class names are related to the purpose and function of the system, we created a corpus in the context of software by collecting class name component words. Using this, we conducted a subject experiment to confirm that $corpus_{Pj}$ reflects the system overview.

However, some issues still need to be solved in the previous studies. We found that the method needs to provide more information when class names do not contain behavior-related information or the method in suitable classes. Therefore, it is necessary to include more detailed internal information in $corpus_{Pj}$.

The goal of this study is to propose a new overview grasping method to improve the accuracy of the previous one. As a key idea, we focus on the method name component words and hypothesize that these words reflect the system's function. If we can effectively extract the meaning from method names alone, we can reduce the time spent understanding the software product overview and accelerate software upcycling. The proposed method consists of the following four approaches:

(A1) Mining software repositories
(A2) Get method name component words
(A3) Weight by tf-idf
(A4) Visualization of importance by tag cloud image

We conducted an experimental evaluation of 136 Java projects managed in our laboratory using the proposed method. The following two aspects are considered components of the project overview:

`What`: The function or value that the software provides
`How`: Technology to make the software work

As a result, we extracted 14604 methods and 1659 method name component words. We examined the method from the viewpoints of `What` and `How` and found that the elements appearing as nouns in the "noun + verb" structure of the method correspond to the things and events handled in the system and that this information is useful for inferring the outline from the viewpoint of `What`. However, we found getting higher-level information, such as design patterns, complicated from method names.

II. Previous Study: Experimental Evaluation of Inferring Software Description Using Project Corpus [4]

A. Challenges in Software Development

Not all products are managed in an asset-like manner in software development sites such as organizations and companies.

979-8-3503-2959-9/23/$31.00 ©2023 IEEE

There are many cases where product-related documentation such as $README$ needs to be included or increased. In contrast, such software may contain an asset value. Specifically, they are ideas for design and implementation. The value of existing software is expected to be utilized for new software development, which we call software upcycling [2]. In order to facilitate software upcycling, the challenge is understanding the project outline, with or without a description.

B. Proposed Method in Previous Study

In the previous study, we proposed a method to get an overview of an existing project without relying on $README$ or other explanatory documents. We create $corpus_{Pj}$ from the class name component words and use it to infer the system's functionality, technology, and overview. The experiments conducted in the previous study consisted of five Steps.

In Step 1, we select the project and subjects for the experiment. In the previous study, we used Java projects in Gitlab, managed in the author's laboratory.

In Step 2, we prepare the $corpus_{Pj}$. Using repository mining [5], we retrieve the class names contained in the project and split the strings word by word. In this way, we defined $corpus_{Pj}$ as the group of words extracted by this process (see Section II-C).

Step 3 is to design the questions. The following three questions are designed to determine the level of understanding of the project.

(Q1): Please describe in bullet points as many as you can think of this system's functions.

(Q2): Please describe in bullet points as many as you can think of what kind of technology and mechanism this system works with.

(Q3): Please describe in one or two lines an overview of the system as inferred from $corpus_{Pj}$.

In Step 4, the subject answers the questions. In addition to displaying the extracted words in lexical order, a Tag cloud image is presented, weighted by frequency of occurrence.

In Step 5, the responses are graded. A Gitlab administrator who knows the details of each project grades the responses.

C. Previous Key Idea: Defining Software Corpus

In the proposed method in the previous study, we defined $corpus_{Pj}$ as a group of words extracted by the class name segmentation process. Corpus [6] is generally used in linguistics. It refers to an extensive database of texts and utterances.

In this study, $corpus_{Pj}$ means a corpus in the context of software. Specifically, it is created by collecting class names contained in the project and splitting them into words. If the corpus reflects the feature of the project from the source project, it is easy to infer the project outline from it, even for projects without a $README$.

D. Results of Experiments in the Previous Study

Twelve subjects responded to the experiment. The results confirmed that two factors influenced project inferences.

The first is how well $corpus_{Pj}$ reflects the system's information. It depends on the developer how many functions and roles can be included in a class. By extracting the names of the methods inside a class, it is expected that the functions and roles of the class can be understood more accurately.

The second is the informational content of words. We confirmed instances of words that occur in large numbers but are not necessarily crucial to the software. Hence, weighting words for software characteristics would lead to a more practical overview of the software.

III. Proposed Method

A. Goal and Key Idea

The purpose of this study is to improve the accuracy of understanding by taking a different approach from the previous study. There are the following two key ideas:

First, the creation of $corpus_{Pj}$ using method names. Since method names are closely related to software functionality, it is expected to provide a more detailed understanding of system behavior than the approaches in the previous study.

The second is to weight $corpus_{Pj}$ according to word importance. In the previous study, we used the frequency of occurrence of words as a weighting index. However, we confirmed that a frequently appearing word does not necessarily represent an overview of the system. In this study, we use *TF-IDF* as an alternative measure to the frequency of occurrence to determine the importance of a word.

B. Approach

The proposed method uses a four-step approach. The outline of the approach is shown in Figure 1. The details of the approach are as follows, consisting of (A1) $\sim$ (A4).

- **(A1) Mining Software Repository:** Get program files that exist in the target project (Pj) by mining software repository.
- **(A2) Get method name component words:** Extract method names contained in the file using a parsing tool. Then, the extracted method names are decomposed into words.
- **(A3) Weight by TF-IDF:** Weight the words using *TF-IDF*.
- **(A4) Visualization of importance using Tag cloud images:** Visualize the weight of words for each project using Tag cloud. Words with high importance are displayed in a large size and highlighted.

C. (A1) Mining Software Repositories

Get class files from the target project group. Mining is performed only for class files that describe the process, excluding files such as configuration files and $README$.

The following example is based on the service named **MP3PlayService**, which plays MP3 format audio files in our laboratory. Table I shows the results of repository mining. MP3PlayService is used as an example in the following steps as well.

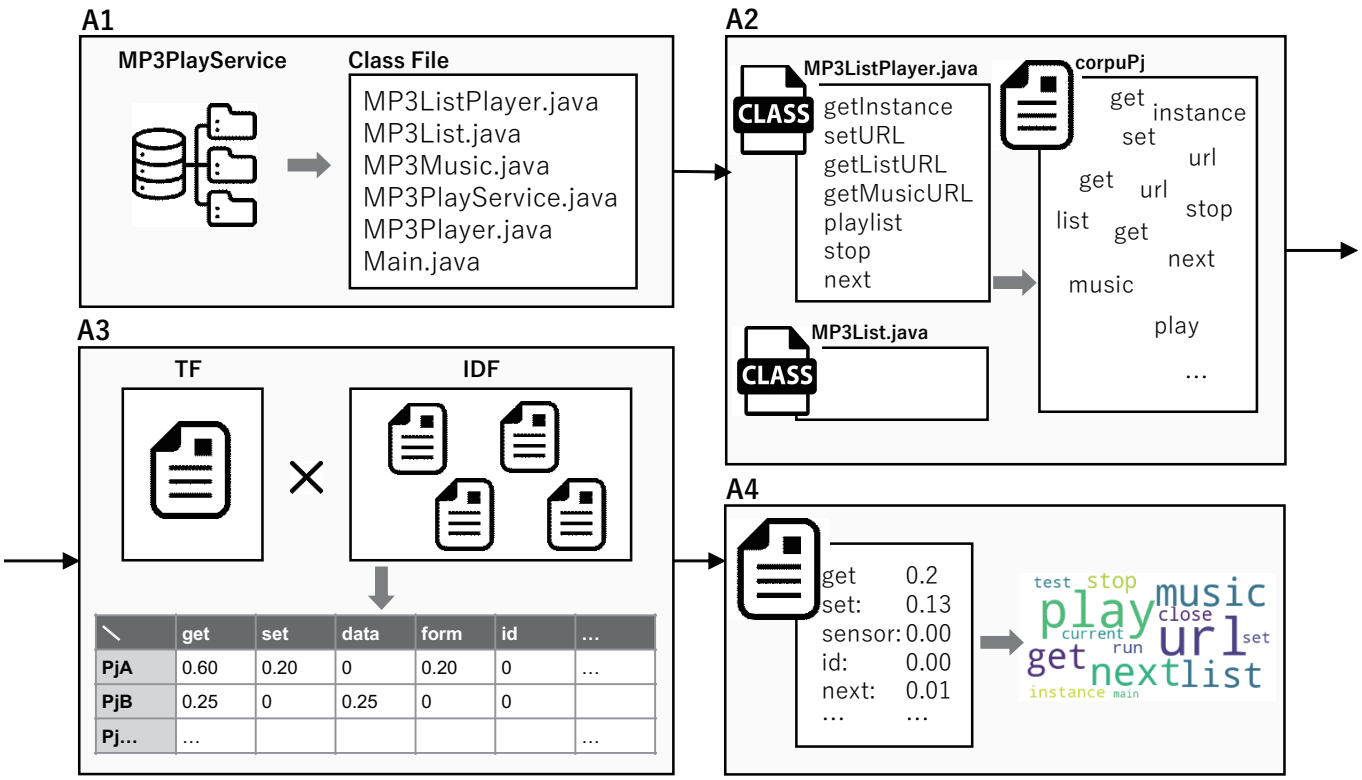


Fig. 1. Flow of proposal approach.

TABLE I
JAVA FILES INCLUDED IN MP3PLAYSERVICE.

class name
MP3List.java
MP3ListPlayer.java
MP3Music.java
MP3PlayService.java
MP3Player.java
Main.java

TABLE II
METHOD NAMES INCLUDED IN `MP3ListPlayer.java`.

method name
getInstance
setURL
getListURL
getMusicURL
playList
stop
next

D. (A2) Obtaining Method Name Component Words

The parser tool is used to extract the method names in the file. The method name is cut out as a string and extracted using an analyzer corresponding to the programming language. The string is divided into words using regular expressions, etc., according to the naming conventions used in the programming language.

Table II shows the results of retrieving the method names contained in `MP3ListPlayer.java` using ANTLR [7]. ANTLR is a parser generator for generating parsers, and various programming languages are supported.

Table III shows the result of splitting the methods extracted in Table II. Since the naming convention for method names in Java is camelCase, the method is split using the position where the capital letter appears as the delimiter. In this case, MP3ListPlayer.java generates a corpus of nine methods: get, Instance, set, URL, List, URL, Music, play, and stop.

TABLE III
RESULT OF METHOD NAME SPLITTING.

method name	$corpus_{PJ}$		
getInstance	get	Instance	
setURL	set	URL	
getListURL	get	List	URL
getMusicURL	get	Music	URL
playList	play	List	
stop	stop		
next	next		

E. (A3) Weighting by TF-IDF

The weight of each word extracted in A2 is determined using *TF-IDF* [8]. Considering a software project as a document, the word frequency for each project is calculated as `TF` (term frequency), and the word frequency for all projects in the repository is calculated as `IDF` (inverse document frequency). The definition is shown below.

$$tf(w, d) = \text{Number of word } w \text{ present in project } p$$

$$idf(w) = \log \frac{\text{Total number of projects}}{\text{Number of projects where the word } w \text{ exists}}$$

F. (A4) Visualization of Importance by Tag cloud image

The importance calculated for each word is visualized by using Tag cloud [9] images. Tag cloud displays words with higher importance in a larger size, visualizing the words important to the system.

IV. Experimental Evaluation

A. Purpose of Experiment

In this experiment, the proposed method is applied to 136 Java projects managed in the Gitlab of our laboratory. In addition, these projects are contributed by a total of 42 developers. We confirm the effectiveness of the proposed method by creating $corpus_{Pj}$ using the method and discussing it.

B. Experimental Procedure

The proposed method creates $corpus_{Pj}$ from the class name component words. For repository mining in A1, we used GitlabAPI in python. For method name component word acquisition in A2, ANTLR was used as the analysis tool. Segmentation was performed using regular expressions based on the position of uppercase letters in the camelCase notation used in Java. In the TF-IDF weighting of A3, the python library scikit-learn [10] was used to calculate the TF-IDF values. For the generation of the Tag cloud image in A4, we used the python library wordcloud [11].

C. Results

As a result of the experiment, 14604 method names and 1659 method name component words were extracted. Words varied in number of occurrences, revealing the existence of words that are commonly used in method names and words that are not.

Figure 2 shows the top 20 most frequently occurring method name component words. The most frequently appearing word is get. This was followed by verbs such as set, create, and update. Adjunctions and adverbs were also common. Conjunctions and adverbs were also common. In particular, words such as and, is, not, than, and so on were frequently used in naming.

However, the most frequently occurring nouns were found to be rare. System-specific object names appear as nouns in method names, and most $corpus_{Pj}$ is composed of low-frequency proper nouns.

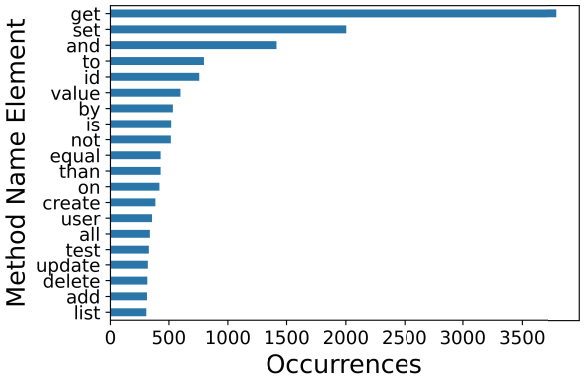


Fig. 2. Top 20 in frequency of occurrence.

V. Discussion

A. Perspectives for Consideration

Several perspectives are used to describe the project overview. In this study, a project overview is described from the following two perspectives.

`What`: The behavior or value provided by the software. It is distinguished from the use cases of the system since the way the software is used varies from user to user. There can be multiple use cases in a project.

`How`: Technology that makes the system work. Tools for system construction, such as libraries and design patterns.

We will evaluate the proposed method's effectiveness using two perspectives. For illustration purposes, we will consider MP3PlayService. The "what" in this service includes playing MP3 sound sources, list playback, etc. The "how" includes JavaLayer for playing MP3 files, URLEncoder for encoding HTML files, etc. Table reffig:projects shows an overview of the projects used in the discussion, $corpus_{Pj}$, and Tag cloud images.

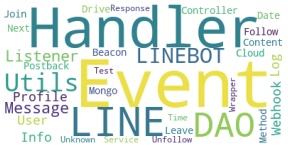
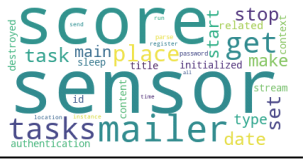
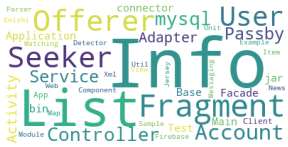
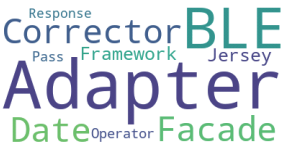
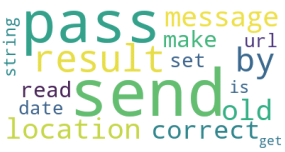
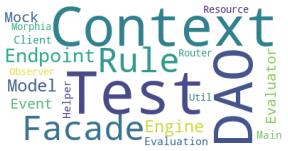
B. Perspective of What

Weighting using TF-IDF allowed us to discriminate words representing objects involved in the system's behavior. As an example, we show `LifeActivityMailer`. In the Tag cloud image, words such as score and sensor are weighted to highlight objects involved in the system's behavior.

Compared to the method in the previous study, which used class names, the proposed method provides more detailed information about the system. In contrast, the Tag cloud image of the proposed method shows specific information such as location, message, date, etc., which are handled by the sensor. Therefore, the proposed method is effective for systems with many methods in a class.

However, there are examples where unimportant words are weighted. Focusing on ENISHI as an example, we can see the high importance of words such as "get" and "set." ENISHI is a web application built using *SpringBoot*, and the reason for this is that the system has many descriptions for routing data acquisition in the Controller layer classes in the MVC model.

TABLE IV
PART OF THE EXPERIMENTAL DATA.

Overview	previous study	proposed method
NewLINEAgentService A service that sends LINE messages using LINE Bot		
LifeActivityMailer Email notification service when there is a change in environmental sensors		
ENISHI Enishi's back-end services Functionality is very limited for mocks		
MP3PlayService MP3 playback service based on a given URL Playback by playlist is also available.		
BLEAdapter Adapter that formats BLE (Bluetooth Low Energy) data and conforms to the passing-by framework		
thin-cas Lightweight command line application to perform Context Aware Service (CAS) with ECA rules		

As shown above, we can see verbs frequently occurring in the implementation pattern of web applications, such as the getter and setter methods provided in each class as entities.

Based on the above, the noun elements in the method correspond to the things handled in the system. We consider this valuable information for inferring the outline of the system.

C. Perspective of How

We consider whether the technologies and frameworks used in the system can be inferred. As an example, we focus on `NewLINEAgentService`.

LINE Bot [12] is used in this project, and since it exchanges data and AccessToken with the LINE Bot in the `LINEBOTController.java` class, these words can be seen in the Tag cloud image. However, these words are relatively small in terms of importance value and are relatively small in the Tag cloud image. Thus, it isn't easy to efficiently get multiple functional overviews using a single importance index in Tag cloud visualization.

Also, information about the system's design, such as design patterns, is difficult to infer. Thin-case implements the DAO pattern and the Facade pattern. These are mainly manifested in naming directories and classes, but such information cannot be read from the method names. Thus, getting higher-level information, such as design philosophy, from method names takes much work.

VI. CONCLUSION

In this study, we proposed a method focusing on method name component words to improve the accuracy of understanding project outlines in the previous study. Then, we weighted $corpus_{Pj}$ using TF-IDF and calculated the importance of the words. As a result, we got more information on things related to the internal operation of the project than in the previous study. Moreover, we found the possibility of using

this method to get a general understanding of the functions provided by the project. However, it has some disadvantages compared to the previous studies. Getting information related to high-layer design, such as design patterns, is difficult. As a challenge, we found that it was inferior to the previous studies regarding the technology used. In future work, we plan to create $corpus_{Pj}$ from multiple approaches and consider methods of summary inference according to the purpose.

ACKNOWLEDGMENT

This research was partially supported by JSPS KAKENHI Grant Numbers JP19H01138, JP20H05706, JP20H04014, JP20K11059, JP22H03699, JP19K02973, and Young Scientists (No.23K17006).

REFERENCES

[1] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *Ieee software*, vol. 29, no. 6, pp. 18–21, 2012.

[2] T. Nakata, S. Chen, S. Saiki, and M. Nakamura, "Succeed: Sharing upcycling cases with context and evaluation for efficient software development," *Information*, vol. 14, no. 9: 518, September 2023.

[3] C. Berenguer, A. Borges, S. Freire, N. Rios, R. Ramač, N. Taušan, B. Pérez, C. Castellanos, D. Correal, A. Pacheco, G. López, M. Mendonça, D. Falessi, C. Seaman, V. Mandić, C. Izurieta, and R. Spínola, "Investigating the relationship between technical debt management and software development issues," *Journal of Software Engineering Research and Development*, 02 2023.

[4] K. Terakawa, S. Chen, and M. Nakamura, "Design and evaluating a method using project corpus for inferring software description," in *2023 20th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2023, pp. 285–290.

[5] J. Oliveira, M. Viggiato, D. Pinheiro, and E. Figueiredo, "Mining experts from source code analysis: An empirical evaluation," *Journal of Software Engineering Research and Development*, vol. 9, 02 2021.

[6] T. McEnery, *Corpus linguistics*. Edinburgh University Press, 2019.

[7] "Antlr," https://www.antlr.org/, (Accessed on 20 February 2023).

[8] L.-P. Jing, H.-K. Huang, and H.-B. Shi, "Improved feature selection approach tfidf in text mining," in *Proceedings. International Conference on Machine Learning and Cybernetics*, vol. 2, 2002, pp. 944–946 vol.2.

[9] S. Bateman, C. Gutwin, and M. Nacenta, "Seeing things in the clouds: the effect of visual features on tag cloud selections," in *Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*, 2008, pp. 193–202.

[10] "scikit-learn machine learning in python," https://scikit-learn.org/stable/, (Accessed on 20 February 2023).

[11] "Wordcloud for python documentation," http://amueller.github.io/word_cloud/, (Accessed on 20 February 2023).

[12] "Line developers messaging api," https://developers.line.biz/en/docs/messaging-api/, (Accessed on 20 February 2023).