Webology, Volume 15, Number 2, December, 2018

Micro Level Source Code Summarization of Optimal Set of Object Oriented Classes

Mrinaal Malhotra

Department of Computer Engineering, National Institute of Technology, Kurukshetra, Haryana, 136119, India. E-mail: mrinaalmalhotra@gmail.com

Jitender Kumar Chhabra

Department of Computer Engineering, National Institute of Technology, Kurukshetra, Haryana, 136119, India. ORCID. E-mail: jitenderchhabra@nitkkr.ac.in

Received May 10, 2018; Accepted December 25, 2018

Abstract

Maintenance of software is effectively possible only after proper understanding of its modules and components. Optimal summary of the different components can help a lot in understanding their internal logic. Hence source code summaries must include the description of the functionalities and the intent behind the existence of various source code entities. In this paper we propose a new methodology for source code summarization performed at class level. The proposed methodology identifies an optimal set of classes which are more beneficial for summarization. Our generated summary includes all important information such as its properties, architectural details, its dependencies on other classes, internal implementation details and inheritance etc related information. This entire summary is generated in natural language using an automated process. An empirical study of the proposed approach has been carried out over seven open source software and qualitative evaluation also has been carried out for certain important classes with help of experienced developers. In order to demonstrate the usefulness of our proposed approach, our generated summary is compared with the competitive commonly used summarization method. The comparison is done using pyramid approach which uses weights as a means to compare two or more summaries. The results clearly suggest that our approach generates summaries are quantitatively as well as qualitatively more useful from comprehension as well as maintenance view point.

Keywords

Summarization; Optimal set; Object oriented classes

Introduction

Program understanding is one of the primitive tasks of software maintenance. For understanding, software system maintainers need to read each and every line of code. This takes significant amount of time. The other alternate can be skimming the code. Skimming means skipping some lines of code which seem unimportant (Moreno et al., 2013). In this case only headers of modules are considered important and rest of the module is skipped. Sometimes the names of the modules are not relevant and do not help in understanding the module, and skimming can lead to misunderstanding and confusion (Haiduc et al., 2010). Hence, the need of the hour is some middle ground, i.e. summarization of source code in natural language.

Summarization is defined in the literature as "a reductive transformation of source text through content condensation by selection and/or generalization of what is important in the source" (Spark, 2007). The summaries, which are generated, should be precise, complete, useful, automated, and able to reveal the important terms, capture the structure and semantics of source code and should be shorter than the actual code (Haiduc et al., 2010). Summarization plays an important role in understanding and changing the software system during the maintenance phase of software development life cycle. As in the maintenance phase maintainers are not familiar with the source code and summaries help to get that familiarity with the software system. Summaries can be used in various fields. Summarization plays an important role in reusability (Parashar & Chhabra, 2011).

In case of reusability, developers need to know which of the features of some already designed software can be used in the new software and optimal summary of applicable components normally includes such features. Summarization is also useful for various maintenance related activities such as change impact analysis (Malhotra & Chhabra, 2018), restructuring (Rathee & Chhabra, 2017) etc. Summarization is also used in corrective as well as adaptive maintenance. While changing the software system, during maintenance, only a few of the source code entities are modified but due to the modifications some other source code entities which are related to the modified one are affected. Hence, we need to incorporate some changes in the effected entities as well. This phenomenon is change impact analysis (Malhotra & Chhabra, 2018). In order to find out which source code entities are affected by the introduced changes, understanding structural dependencies between different source code entities is mandatory which can be done easily with the help of summaries. If the summaries are made from implementation point of view then those summaries can provide the basic structure of the system and hence, this will be a great help for restructuring.

Broadly, summaries are of two types': extractive summaries and abstractive summaries. Extractive summaries are those in which contents of summary are obtained by selecting important information from document and abstractive summaries are those in which higher level of abstraction is considered for selecting information and it usually contains information which is not present in the original document (Kavitha & Ram, 2016) . In this paper we are focusing on extractive type of summaries. Summarization can be performed at various granularity levels but in this paper we are focusing on class level summarization for selected set of classes (optimal set of classes). The purpose of this summary will be to help maintenance team to carry out different maintenance activities such as corrective maintenance, adaptive maintenance etc.

Literature Review

Over a span of 10 years, lot of work has been done by researchers in the field of summarizing source code artifacts. Summarization has been attempted at four levels of granularity namely; system level, method level, class level and block level. Many researchers focused on method level summarization. Different techniques have been proposed in the past for summarizing methods. Content selection approach has been employed for automatic summarization of java methods. Here s_units (important lines of code of a method) are detected by predefined algorithm and included in the summary (Sridhara et al., 2010). Another approach is employed by Rodeghero et al. where eye tracking of experienced programmers is performed (Rodeghero et al., 2014) whereas in (Abid et al., 2015) method stereotypes have been used to find the summary of methods. Here primarily summarization is performed by focusing on the names of the methods. Another work is performed on summarizing methods where different types of messages are generated automatically with the help of identifiers and data and control dependencies (Mcburney & McMillan, 2016).

Apart from summarizing methods other source code artifacts have also been summarized. In (Hamou-Lhadj et al., 2006), execution traces have been summarized. Here, fan-in, fan-out metric is used for detecting utilities and in (Buse & Weimer, 2010), exceptions are summarized by an algorithm that locates all those statements which are throwing exceptions and tracks the flow of exceptions throughout the program. But in (Rastkar et al., 2010), bug reports are summarized using a machine learning approach.

In the literature, some work has been contributed towards summarizing the changes which are done to the software. . In (Buse & Weimer 2010), deltadoc named algorithm is used for documenting program changes. In (Cortes-coy et al., 2014), commit messages have been summarized. Commit messages describe what and why of any modification done in the system. This approach automatically generates a change set for each modification suggested by the developers. In (Shen et al., 2016) too, source code changes have been summarized. It describes what changes have been done and why all these changes are performed.

Summarization can be performed at various levels such as macro level, micro level and nano level. The figure 1 gives an overview of the summarization levels.

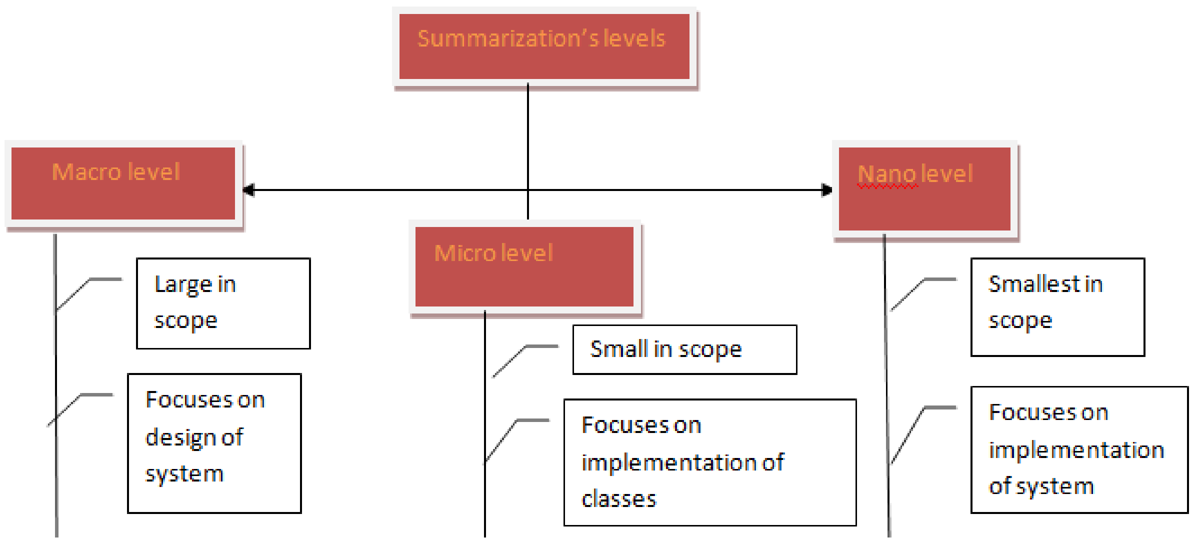


Figure 1. Summarization levels

Macro level:

In macro level a sub-system of software is summarized. Sub system consists of various methods and classes. These methods and classes are taken into consideration together as they belong to same sub-system. These modules and classes must be related to each other via a common link. In optimal macro level summarization a sub-system of any software is summarized with the help of design structures. This type of summarization has various applications like in recurrent neural networks, in news casting etc (Niu et al., 2017; Christie & Khodra 2016). Considerable amount of work is done by researchers in the past on macro level summarization (Choo et al., 2008) (Wang & Yang, 2006).Various techniques have been employed for performing summarization such as in (Sarkar et al., 2015), a graph based technique is employed for text summarization using enhanced sentence similarity measure, in (Britsom et al., 2015), data merging techniques have been used for generating summary, (Gulati & Sawarkar 2017), a novel technique is used for summarizing hindi text.

Different types of summaries are created in the literature like in (Tohalino & Amancio 2017), an extractive type of summary has been generated of a collection of modules or documents with the help of neural networks. In the paper, central sentences from a collection of multiple modules and documents (which have a connection with each other) have been extracted similarly (Ranjitha & Kallimani17) generated an abstractive type summary.

Micro level:

In micro level only one module or only one class is considered for summarization. Micro level is different from macro level as here only one particular method or a class is summarized instead of a complete set of methods and classes, micro level approach is useful from maintenance point of view where modifications are done to few classes/modules but due to those modifications some other entities are also affected and hence, needs to be understood such understanding can be more effective through natural language summaries. These cases do not need to understand the entire sub-system. This paper focuses on micro level summarization and we are considering classes of object oriented software for summarization.

In the past various researchers have worked on class level summarization. In (Haiduc et al., 2010), vector space model (VSM) is used for summarizing classes and methods. Here source code corpus is generated which is used to make decision matrices. These matrices decide whether a particular term comes in the summary or not. In the summary, lexical and structural information is included. Lexical information means general description based on name of class and structural information means purpose of the class. In (Haiduc et al., 2010), latent semantic indexing (LSI) is also presented. In only difference in LSI and VSM is that in LSI cosine similarity is used for filtering terms from the matrix instead of selecting top K values. In a research (Moreno et al., 2013), stereotypes are used for generating summaries. In (Kavitha & Ram 2016) too, a source code corpus is created just like (Haiduc et al., 2010) where a matrix is made having rows representing terms and columns representing documents. Each row contains two cells. One represents weight of those terms in that document and the other cell represents the relevancy ratio. Those terms which cross the threshold of relevancy ratio are included in the summary. Summary includes information about LOC of class and purpose of class. In this paper, we are focusing on micro level summarization of source code.

Nano level:

Unlike macro level and micro level, nano level summarization is done on small blocks of code present inside the source code. Nano level covers the basic implementation details of the source code in consideration. Nano level summarization is performed by considering various techniques such as in (Badre & Thepade 2016) summarization is performed using frame extraction i.e. it takes only special blocks into consideration. In (Nazar et al., 2016), a particular fragment is targeted for summarization, in (Kroening et al., 2008) loop summarization is performed using abstract transformers where loop is a particular block in consideration and (Kroening et al., 2013) performed the loop summarization using state transition invariants. Nano level is less explored by the researchers in the literature as this type of summarization has limited applications in the industry.

Motivation

Summaries are a backbone in the maintenance phase of software development life cycle. During maintenance, summarization helps in building the basic knowledge about the implementation and functionality provided by the software system. A number of important factors are missing from the summaries in the literature. These factors can improve the summaries to a great extent. The factors which are missed by researchers in the past are taken as a motivation for our work. A number of techniques described in section 2, are analyzed for discovering the missing issues of summarization which are stated as follows:

- Almost all of the summaries generated by previous techniques do not include any low level implementation details. Implementation details must be present in the summary because during maintenance different classes require changes in the source code for either correcting the previous implementational bugs or for adding new features. In all such cases the maintainers need to understand the logic and implementation details of the classes and thus such kind of information is highly desirable to be included in the summary. Implementation details (number of mutable fields in a class, number of final fields of a class, whether the class is self referencing or not, which methods of the parent class are over ridden by the class, whether can be re-written as interface or not etc) are not considered by any of the previous techniques while generating the summary. These details can be revealed using micro patterns (Gil & Maman 2005). Here, we propose to use micro patterns as a basis for fetching the implementation details of classes. Micro patterns are recurring schema which helps in capturing static and dynamic structure of the class (Gil & Maman 2005). By using micro patterns as an aid in summarization, we can get clear understanding of the implementation details. It helps significantly in re-structuring the system, helps in deciding the reusability of any class also is useful in change impact analysis (Malhotra & Chhabra 2018b).

- Different types of interdependencies between the source code entities are not included in the existing generated summaries. Inter dependencies between different source code elements are highly desirable in the summary as these help in identifying more prominent classes (prominent class will have more number of inter-dependencies), These also help in finding the set of affected classes if some changes are done to the system, help in designing the class diagram of any old system which has been brought in for the maintenance (From class diagram we can re-draw the basic structure of the system and this will be a great help in restructuring the system). Interdependencies can also help in finding the returned objects of the class and also the variables which are returned by the functions declared in the class. Thus, some information about dependencies must be present in the summaries. In this paper we propose to include details of dependencies between classes.

- While generating summary of any class, the high level design or the overall architecture of the class should also be considered. The basic design of class will include all the mandatory variables and objects that are supposed to be a part of the respective class. All this information is obtained with help of the design patterns that capture the static design structure at the class level. In this paper we are considering the design patterns as an aid in software documentation because it helps describing in the evolution of the software system (Douglas et al. 2013).

- Apart from all these major gaps, some other information is also in the summaries generated in the past like information about inheritance and information about the packages contains the class under consideration. Paper proposes to include these details in the summary. This general description is also covered in this paper. This general information should be present in the summary because it helps in computing the reusability, and change impact analysis.

All the above mentioned research gaps are filled in this paper.

Proposed methodology

Software consists of many classes, sometimes hundreds of classes. If summarization of all classes is targeted, it will take lot of time and resources, but many of these summaries might not get used at all because of less importance of classes. It has been observed that some classes are more relevant than others. Relevancy means that those classes which are more utilized in the source code i.e. those classes whose objects have more frequency of occurrence than other objects. Summarization of each and every class is a time consuming process. So, in order to save resources and time, an optimal set of classes should be identified.

- For optimal summarization only relevant classes should be considered. Relevant classes are found out by setting a threshold value. Only those classes which cross the threshold of relevancy will be considered for summarization.

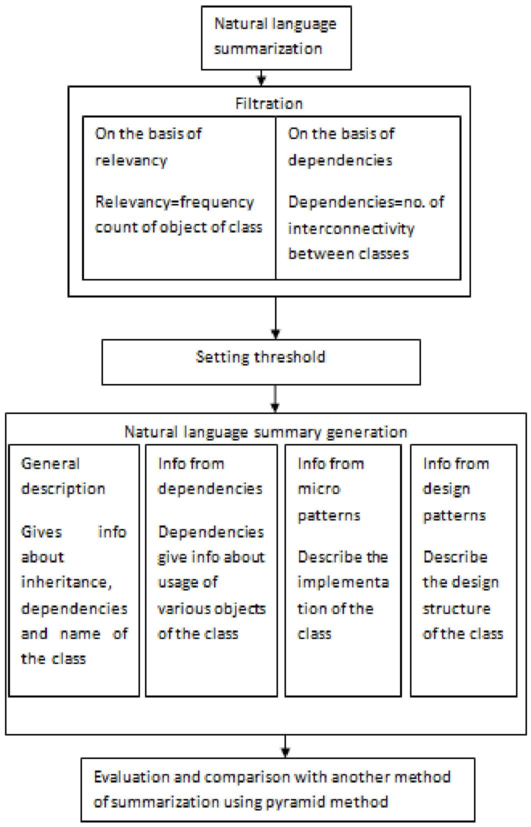
- Apart from relevancy, the dependencies between classes should also be considered for filtration. Dependencies inform us about the usage of objects of various classes i.e. dependencies tell us where the objects or data members of the class are used in the entire source code. If a class has been used in many functions then that class is more important from maintenance point of view. Hence, second filtration is performed on the basis of dependencies. Similar to relevancy, a threshold is set. Only those classes are summarized which cross the threshold value.

The summary is generated in natural language thus it is easy to understand by everyone. Our automatically generated summary contains the following information:

- General information of the class: Information about name of the class, information about parent class, total number of dependencies and dependents present in the system.

- Information obtained through micro patterns: Information about the name, category and definition of those micro patterns which are tagged with the class.
- Information obtained through design patterns: Information about those design patterns which are tagged with the class.
- Information obtained through dependencies: Information about the usage of the objects of the class.

For providing the above mentioned information about the class, a template has been designed. After designing templates, information extracted from source code and the respective template is filled. In this paper, we are only considering class level summarization. The main steps of our proposed methodology are explained in Figure 2 below.



Natural language summarization
Filtration
On the basis of relevancy
Relevancy=frequency count of object of class
On the basis of dependencies
Dependencies=no. of interconnectivity between classes
Setting threshold
Natural language summary generation
General description
Gives info about inheritance, dependencies and name of the class
Info from dependencies
Dependencies give info about usage of various objects of the class
Info from micro patterns
Describe the implementation of the class
Info from design patterns
Describe the design structure of the class
Evaluation and comparison with another method of summarization using pyramid method

Figure 2. Flow Chart of Algorithm

```
    Pseudo code:
    Main (){
Input: source code, jar file
Consider a set S of selected classes for summarization.
        Filtration_on_basis_of_relevancy()
        Filtration_on_the_basis_of_dependencies()
        Natural_language_summary_generation()
        Evaluation
    }
    Filtration_on_basis_of_relevancy(){
        for all  i, i belongs to set of all classes of software.
        p= $\sum o_i$ where $o_i$ is occurrence of class I in source code.
        set threshold (Φ ) for filtration
        j=1.
        while j<i
            if $p_i$>= Φ
                S= S U i where S is the set of selected classes for summarization.
            else if   $p_i$< Φ
                discard i for summarization
    }
    Filtration_on_the_basis_of_dependencies(){
        for all i,  i belongs to set S.
        D =$\sum d_i$ where $d_i$ dependencies ofclass i.
        set threshold  (Φ ) for filtration
        j=1
        while j<n(S)
            if $D_i$ < Φ
                S= S –{i}
            else if $D_i$>= Φ
                keep i for summarization
    }
    Natural_langugae_summary_generation(){
        Design templates
        for all i, where i belongs to S
        Analyze the source code
        Fill templates with appropriate info
        Join together all templates to form one summary.
    }
```

A. Finding the optimal set on the basis of relevancy

This step finds the optimal set of relevant classes. For finding the optimal set and filtering the less important classes, relevancy of each class of software system is calculated. For that we are using the frequency of objects of the class. If a class’s object is occurring more number of times, then it means that the class is more important and is useful for much functionality. The following formula is used for calculation:

$$p_i = \sum O_i \text{ in LOC}$$

Where O_i is occurrence of object of class i in lines of code.

p_i is relevancy of class i.

The relevancy of a class is directly proportional to the frequency of occurrences of that class. After calculating the relevancy of all classes of the system, a threshold is obtained by following the given approach:

All the classes are sorted according to frequency of occurrence in decreasing order. A number for threshold is decided in such a way that at most 45 percent of total classes are included in the first draft of the optimal set. Those classes which cross the threshold of the relevancy, are considered for summarization and rest are discarded.

B. Finding the optimal set on the basis of dependency

After filtration on the basis of relevancy, second filtration is performed on the basis of dependency. Dependency of each class is calculated with the help of a tool called structural analysis of java (Sangal et al. 2005). The dependencies are calculated using the following relationships:

$$D_i = Accesses_i + Calls_i + Contains_i + Extends_i + Implements_i + Instantiates_i + Uses_i$$

After calculating the dependencies and dependents of every class, a threshold is obtained by following the same approach described above. Only those are considered for summarization which will cross the threshold value. Those classes which are selected after second filtration are going to be summarized.

C. Generating natural language summary

After finding the optimal set of classes which are to be summarized, we need to design templates for each and every type of information to be provided by the summary.

i) The template which is designed to represent the information obtained by dependencies relationship is as follows:

“*The objects of <name of class> have been used at <name of functions/classes>*”.

Apart from usage tags we also need to add some general information in the summary which will give description about name of the class, inheritance and total number of dependencies and dependents.

ii) The template which is defined for this purpose is shown as:

“*This class encapsulates data and behavior related to <name of class>. This class inherits features from <name of parent classes comma separated>. This class has <no> of dependencies and <no> of dependents*.”

iii) We need to provide information about the implementation of the software system which is generated with the help of the micro patterns. As already explained, micro patterns are recurring schema which capture the lower level implementation of the software system. Micro patterns describe the basic structure of the class under inspection. We can search for finding out which micro patterns can be tagged in a class. A class can be tagged with more than one micro pattern. For tagging, the entire source code of the class is scanned. After scanning pattern is identified then pattern is tagged with the definition of the class. Some micro patterns are more common and easy to locate in various classes. The percentage of classes which are tagged with a particular micro pattern is termed as the prevalence of that micro pattern. For some micro patterns, prevalence of 0.5% is enough as they are quite rare to locate. On the other hand, some micro patterns prevalence is 30-40%. The Templates designed for information obtained through micro patterns is :

“*<name of class> has<definition of micro patterns>”*

iv) Apart from lower level implementation, higher level of abstraction is equally important to describe. Design patterns are used to describe the higher level view of abstraction. Design patterns describe the structure of the basic architecture of the class. Design patterns help to identify some parts of design which can be reused in some other software system. After identifying the design problem, one does not need to solve the problem from scratch. Form the past experience of solving those problems, new problems which are coming with exactly same design can be solved quite easily by following the same procedure which was used last time. Hence, design patterns become a great aid in documentation. The templates designed for this purpose is as follows:

“*<name of class> has<definition of design pattern>”*

v) After designing the templates, each and every class of the set of optimal classes is analyzed thoroughly and information to be filled in these templates is obtained. After filling of each template, the complete summary is obtained by joining together individual templates one after the other.

D. Evaluation of the summary generated

After generation of the summary, the summary must be evaluated and the results must be compared with some commonly used methods of finding the summary. Various methods of evaluation have been proposed in the past like precision, recall, cosine similarity but we are using pyramid method (Nenkova & Passonneau 2004) for evaluation purposes. In this method, weight of each summary is calculated with the help of SCUs. Method of summary generation described in (Moreno et al., 2013) is a competitive method by many researchers and is used by us for comparison purposes.

Case study

In order to explain the above methodology, consider the subject program gson consisting of total 40 classes. These classes are interconnected with each other through dependencies. For evaluation purposes (Moreno et al., 2013) is used for comparison with our methodology. Firstly, these 40 classes need to be filtered on the basis of relevancy and on the basis of dependencies. The data obtained after filtration is shown in Table 1.

Table 1. Data obtained after filtration

Sr. no	Filtration criteria	Threshold	# of remaining classes
1	On the basis of relevancy	8	15
2	On the basis of dependencies	20	11

After both filtrations the final optimal set classes consists of eleven classes. These 11 classes are going to be summarized and the result is compared with (Moreno et al., 2013).

Consider the class gson as an example to explain the concept of evaluation. For gson the following summary is generated with our method (named as summary 1):

"*This class encapsulates data and behavior related to gson. This class inherits features from typetoken, constructorconstructor and fieldnamingpolicy. This class has 46 of dependencies and 23 of dependents. The objects of gson have been used at jsonadapterannotation,typeadatpterfactory and gsonbuilder. Gson has only static final fields and fields are assigned only during instance construction. Gson has encapsulated function and data provided by existing non-object oriented APIs.* "

Similarly the summary generated by (Moreno et al., 2013) method is given below (named as summary 2):

"*GSON is an entity and final class that encapsulates data and behavior. This boundary class communicated with jsonadapterannotation, typeadapterfactory and gson builder. It provides access to excluder and fieldnamingstrategy. It declares helper classes treeadapterfactory and objecttypeadapter*."

After the generation of both the summaries, we are comparing the efficiency of both summaries using pyramid method (Nenkova & Passonneau 2004). This method has been used by many other researchers also for comparing purposes (Hovy et al., 2006; Gillick & Favre 2009; Louis & Nenkova 2009; Haiduc et al., 2010; Rastkar et al., 2010). In this method, both summaries are compared with each other on the basis of a weight. This weight is calculated by analyzing the summaries and collecting the important lines in both the summaries. These important lines of both summaries are called summarization content units or SCUs. The weight of each SCU is calculated by finding those summaries which contain the SCU. In our case the maximum weight of SCU is 2 because maximum of only two summaries can contain the SCU. After finding the weight of SCUs, weights of summaries are calculated by adding the weights of all SCUs which belong to that summary.

Following SCUs are created for the evaluation of the summaries: i) Encapsulates data and behavior, ii) Provides access, iii) Declares helper classes, iv) A boundary class that communicates, v) Inherits features vi) Has only static final fields, vii) Fields are assigned during instance construction.

Figure 3 describes the constructed pyramid while evaluating these summaries.

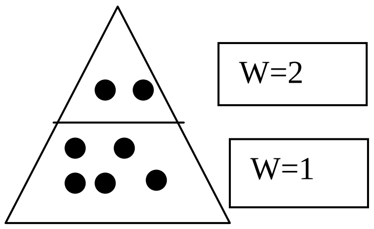


Figure 3. Pyramid constructed

Dots represent the number of SCUs of a particular weight. As per this method, the weight of summary 1 is: 7 and the weight of summary 2 is: 6.

Graphically it can be represented as:
It can be stated that on average the complete structure might look something like this:

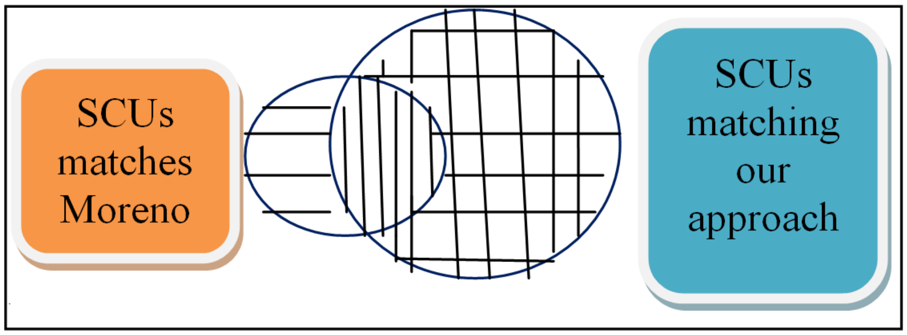


Figure 4. Venn diagram representing matching of SCUs with summaries

▤ These SCUs matches with Moreno et al. (2013). The SCUs which matches are: declare helper classes and a boundary class that communicates,

▥ These SCUs matches with our approach only. The SCUs are Inherits features, Has only static final fields and Fields are assigned during instance construction

▦ These SCUs matches with both the summaries. The SCUs are Encapsulates data and behavior and provides acsess.

Our approach do not produce the SCUs described first as they are less important from maintenance viewpoint. Percentage of improvement is: 16.67%. Hence, it can be concluded that our method improves the final result of summarization.

Evaluation and Results

The proposed methodology is applied to seven software systems for empirical evaluation. Following subject programs are taken into consideration. The source codes of all subject programs are taken from github.com. Table 2 describes the information about subject programs.

Table 2. Subject programs

Sr. No.	Name of subject program	Description	# of classes	# of packages
1	Gson	Open source java library	40	8
2	Google collections	Set of common libraries	100	5
3	Monte screen reader	Provides assistance visually impaired	75	19
4	Javax.ws.rs.api	High-level interfaces and annotations used to create RESTful service resources	108	6
5	Junit	Unit testing framework	122	31
6	Extent report	HTML reporting library	45	11
7	Facebook messanger	Instant messaging Service	49	12

A. Results after filtration

Table 3 describes the number of classes cross the threshold value after filtration:

Table 3. Results after filtration

Sr. No	Name of subject program	Threshold value for relevancy	Remaining no. of classes after relevancy	Threshold value for dependency	Remaining no. of classes after dependency
1	Gson	8	15	20	11
2	Google collections	15	31	10	19
3	Monte screen reader	13	32	11	20
4	Javax.ws.rs.api	20	30	15	21
5	Junit	22	35	18	25
6	Extent report	5	21	4	11
7	Facebook messenger	3	19	5	15

After generation of our summary, its efficiency should be compared with some other method of summarization. For that purpose a method which works very efficiently and is used very commonly is picked. This method gives the summary of classes in a precise form which is useful for maintainers as a summary is supposed to be precise. For comparison purposes (Moreno et al., 2013) is used. In this paper class level summarization is performed with the help of stereotypes. Stereotypes describe the intent of the class. It describes the reason of the existence of the class. Stereotypes are identified by scanning the entire source code of classes and finding which stereotype is going to fit best with the class. The content of the summary includes general description based on the name, information obtained through scanning and tagging stereotypes, information about methods of the class and information about inner classes (if any). A tool is also made by the authors of this paper is our tool plug-in/online available.

The validation of our approach is carried out by testing the usefulness of our proposed approach for maintainers. For evaluation purposes, pyramid method (Nenkova & Passonneau 2004) is used. This method of evaluation is already explained in the previous section. Following Table 4 depicts the average improvement in summary for all software under consideration and these results are graphically shown in Figure 4 below.

Table 4. Results after evaluation

Sr. no	Subject program	Total Weight of summary generated by our method	Total Weight of summary generated by (Moreno et al., 2013)	% of improvement
1	Gson	100	68	47.05
2	Google collections	152	114	33.33
3	Screen reader	161	123	30.89
4	Javax.ws.rs.api	167	125	33.6
5	Junit	202	153	32.02
6	Extent report	85	69	23.18
7	Facebook messenger	123	94	30.85

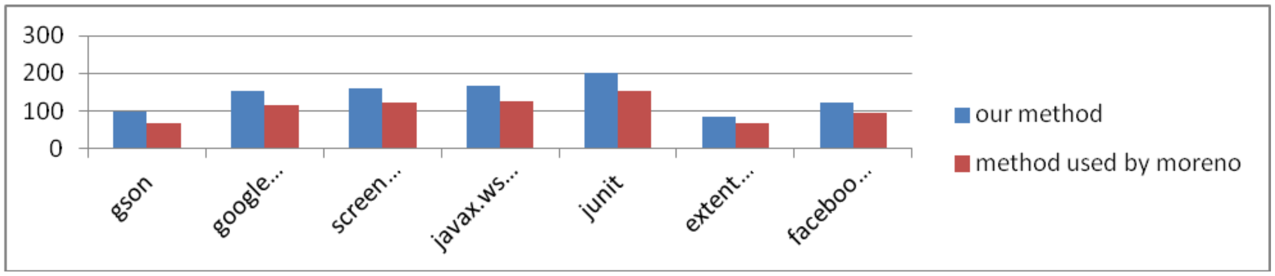


Figure 5. Evaluation results

Figure 4 as well as table 4 describes the results after implementing pyramid method on both the methods of summarization. According to the results, it is clearly evident that our approach is more useful as the weights are more in our approach. This suggests that our approach is better from maintenance viewpoint.

The summaries which are generated by our approach are also evaluated qualitatively with help of some experienced developers of java software systems. Five main classes were identified for subjective evaluation. The summary of these 5 classes which belongs to optimal set are analyzed by developers and the following findings are reported:

1. Class **typeadapter** from subject program gson: After analyzing the source code of the class type adapter, developers came to a conclusion that more stress should be present on the code level as well as high level implementation details in the summary. These details have been skipped in the previous summary but are present in the summary which is generated by our approach.
2. Class **FMProvider** from subject program facebook messanger: Developers felt that for this class more importance should be provided to low level implementational details as per the source code. This type of information is more important from corrective maintenance point of view and this information gets new added in our summary, but was missing in the previous summaries generated.
3. Class **extentreoprts** from subject program extent report: In this class also, low level implementation details play a major role and according to developers, this info should be present in the summery but was missing in the previous summaries. This information is very helpful from corrective maintenance point of view.
4. Class **junitcore** from subject program junit. After analyzing the source code and the design structure of the class, it was observed by the developers that in this class overall architecture of the system is more important and should be present in the summary. This information is present in our summary and it was missing in the summary generated by other competitive methods. This type of information is very much needed for restructuring.
5. Class **rational** from subject program screen reader. In this class also more stress should be given to high level implementation details and design structures. This new info is included in our summary and is very useful for restructuring during maintenance.

Above information is depicted in the following pie-charts where each chart represents one chosen class by the developers and it describes the distribution of the information provided by each patterns and dependencies. The diagram also describes the newly added information which was missing in the summary generated by competitive methods.

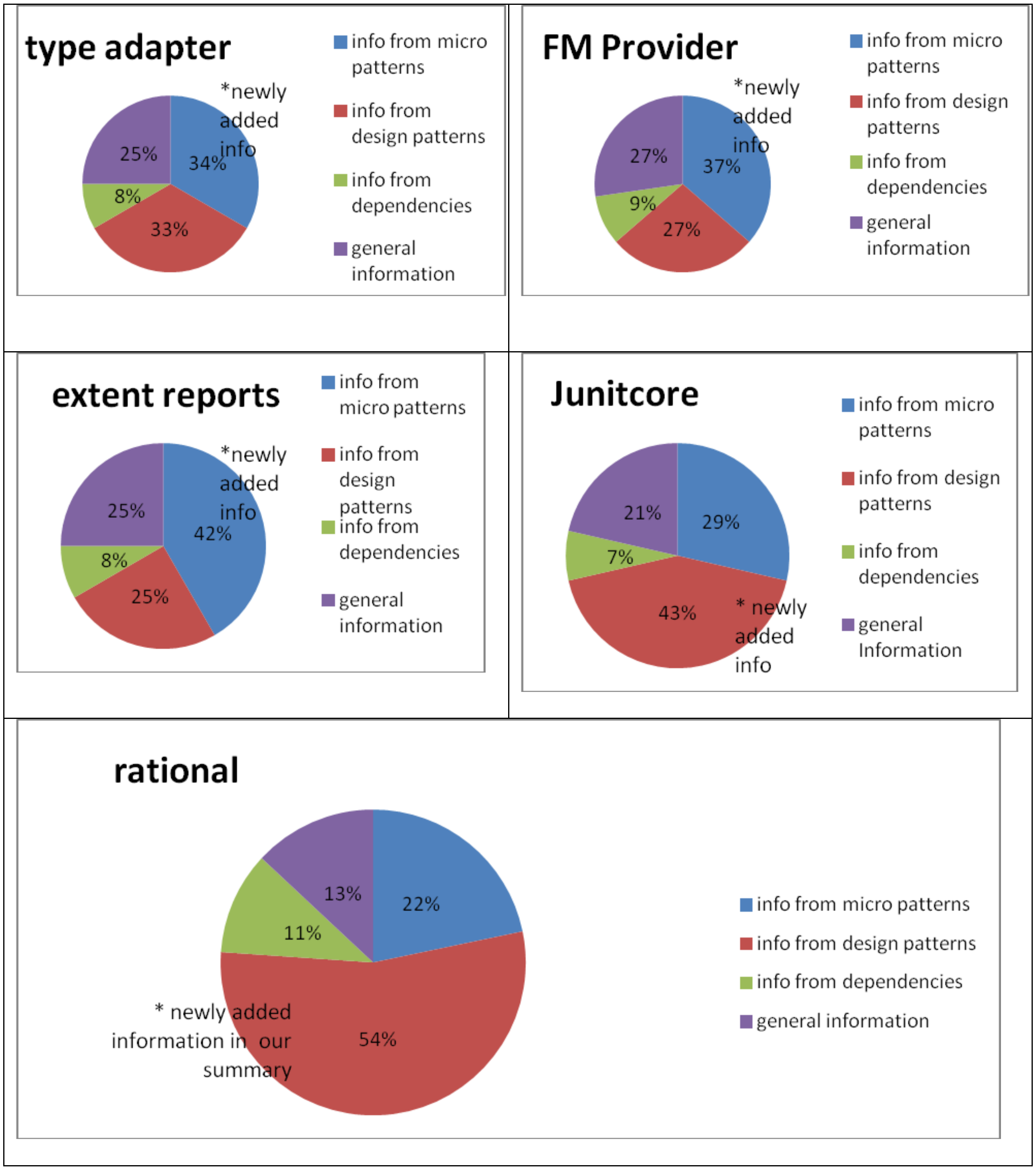


Figure 6. Pie charts describing the distribution of information

In Figure 5 it is evident that new information is added in the summary generated through our approach. Significant amount of code level information is added in the summaries of type adapter, FMprovider and extent reports. Figure 5 also communicates that remarkable amount of information about design patterns in included in the summary of Junitcore and rational.

Conclusions

In this paper, we presented a new methodology for micro level source code summarization of classes of an object oriented software. Such types of summaries are needed for comprehension which hels in effective and efficient maintenance. The proposed methodology first of all extracts an optimal set of classes and subsequently their summaries are generated automatically by including the desired information. Our generated summary includes lot of useful information such as external and internal properties, architectural as well as micro-pattern based details, inter-class dependencies details alongwith special details such as inheritance, mutable data etc. The empirical results clearly demonstrate the usefulness of our approach. The quantitative as well as qualitative evaluation of the proposed approach has also been carried out and the proposed approach has been found to perform better than a commonly used existing approach. Hence it can be concluded that our proposed can be very useful in generating a comprehensive summary of optimal set of classes and can become a useful aid for maintainers.

References

Abid, N. J., Dragan, N., Collard, M. L., & Maletic, J. I. (2015). Using stereotypes in the automatic generation of natural language summaries for C++ methods. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pp. 561-565.

Badre, S. R., & Thepade, S. D. (2016). Summarization with key frame extraction using thepade's sorted n-ary block truncation coding applied on haar wavelet of video frame. In *Advances in Signal Processing (CASP), Conference on*, pp. 332-336.

Britsom, D.V., Bronselaer, A., & Tré, G.D. (2015). Using data merging techniques for generating multidocument summarizations. *IEEE Transactions on Fuzzy Systems* 23(3). pp. 576-592.

Buse, R.P. L., & Weimer, W. R. (2010). Automatically documenting program changes. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pp. 33-42.

Choo, K., Woo, Y., & Min, H. (2008). Two-Level Clausal Segmentation System Based on Semantic Information for Automatic Summarization. In *Artificial Intelligence, 2008. MICAI'08. Seventh Mexican International Conference on*, pp. 42-47.

Christie, F., & Khodra, M. L. (2016). Multi-document summarization using sentence fusion for Indonesian news articles. In *Advanced Informatics: Concepts, Theory And Application (ICAICTA), 2016 International Conference On*, pp. 1-6.

Cortés-Coy, L.F., Linares-Vásquez M., Aponte, J., & Poshyvanyk, D. (2014). On automatically generating commit messages via summarization of source code changes. In *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*, pp. 275-284.

Douglas, C.S., Stal, M., Rohnert, H., & Buschmann, F. (2013). *Pattern- Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, Vol. 2. John Wiley & Sons.

Gillick, D., & Favre, B. (2009). A scalable global model for summarization. In *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*, pp. 10-18.

Gulati, A. N., & Sawarkar, S. D. (2017). A novel technique for multidocument Hindi text summarization. In *Nascent Technologies in Engineering (ICNTE), 2017 International Conference on*, pp. 1-6.

Haiduc, S., Aponte, J., & Marcus, A. Supporting program comprehension with source code summarization. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pp. 223-226.

Hamou-Lhadj, A., & Lethbridge, T. (2006). Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system. In *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on*, pp. 181-190.

Hovy, E., Lin, Chin-Yew, Zhou, L., & Fukumoto J. (2006). Automated summarization evaluation with basic elements. In *Proceedings of the Fifth Conference on Language Resources and Evaluation (LREC 2006)*, pp. 604-611.

Gil, J.Y., & Maman I. (2005). Micro patterns in java code. *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 97–116,

Kavitha, C., & Ram N. S. (2016). Entity based source code summarization (EBSCS). In *2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS)*, Vol. 1, pp. 1-5.

Kroening, D., Sharygina, N., Tonetta, S., Tsitovich, A., & Wintersteiger, C.M. (2008). Loop summarization using abstract transformers. In *Springer's International Symposium on Automated Technology for Verification and Analysis*, pp. 111-125.

Kroening, D., Sharygina, N., Tonetta, S., Tsitovich, A., & Wintersteiger, C.M. (2013). Loop summarization using state and transition invariants. *Formal Methods in System Design*, 42(3), pp. 221-261.

Louis, A., & Nenkova, A. (2009). Automatically evaluating content selection in summarization without human models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1*, pp. 306-314.

McBurney, P. W., & McMillan C. (2016). Automatic source code summarization of context for java methods. *IEEE Transactions on Software Engineering* 42(2). pp. 103-119

Moreno, Laura, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and K. Vijay-Shanker. "Automatic generation of natural language summaries for java classes." In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pp. 23-32. IEEE, 2013.

Malhotra M., & Chhabra, J.K. (2018). Class Level Code Summarization Based On Dependencies and Micro patterns. *International conference on Inventive Communication and Computational Technologies, Coimbatore, accepted and to appear in IEEE Explore*.

Malhotra M., & Chhabra, J.K. (2018). Improved Computation of Change Impact Analysis in Software using all Applicable Dependencies. *Springer's International Conference* on *Futuristic Trends in Network and Communication Technologies*, Waknakghat, Springer CCIS series.

Nazar, N., Jiang H., Gao, G., Zhang, T., Li, X., & Ren, Z. (2016). Source code fragment summarization with small-scale crowdsourcing based features. *Frontiers of Computer Science* 10(3). 504-517.

Nenkova, A., & Passonneau, R. (2004). Evaluating content selection in summarization: The pyramid method. In *Proceedings of the human language technology conference of the North American* *chapter of the association for computational linguistics: HLT-NAACL 2004*.

Niu, J., Chen, H., Zhao, Q., Su, L., & Atiquzzaman, M. (2017). Multi-document abstractive summarization using chunk-graph and recurrent neural network. In *Communications (ICC), 2017 IEEE International Conference on*, pp. 1-6.

Parashar, A., & Chhabra, J.K. (2011). Clustering dynamic class coupling data to measure class reusability pattern. *Springer's High Performance Architecture and Grid Computing*, pp.126-130.

Ranjitha, N. S., & Kallimani, J.S. (2017). Abstractive multi-document summarization. In *Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on*, pp. 1690-1694.

Rastkar, S., Murphy, G.C., & Murray, G. (2010). Summarizing software artifacts: a case study of bug reports. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp. 505-514.

Rathee, A., & Chhabra, J.K. Software. (2017). Remodularization by Estimating Structural and Conceptual Relations Among Classes and Using Hierarchical Clustering. In *Springer's Advanced Informatics for Computing Research*, pp. 94-106.

Rodeghero, P., McMillan, C., McBurney, P.W., Bosch, N., & D'Mello, S. (2014). Improving automated source code summarization via an eye-tracking study of programmers. In *ACM's Proceedings of the 36th International Conference on Software Engineering*, pp. 390-401.

Sangal, N., Jordan, E.V., Sinha, V., & Jackson, D. (2005). Using dependency models to manage complex software architecture, *ACM Sigplan Notices*, 40(10), pp. 167-176.

Sarkar, K., Saraf, K., & Ghosh, A. Improving graph based multidocument text summarization using an enhanced sentence similarity measure. (2015). In *Recent Trends in Information Systems (ReTIS), 2015 IEEE 2nd International Conference on*, pp. 359-365.

Shen, J., Sun, X., Li, B., Yang, H., & Hu, J. (2016). On Automatic Summarization of What and Why Information in Source Code Changes. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 1, pp. 103-112.

Spark, J.K. (2007). Automatic summarizing : The state of art. *Information Processing & Management*, 43(6). pp.1449 1481.

Sridhara, G., Hill, E., Muppaneni, D., Pollock, L., & Vijay-Shanker, K (2010). Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM international conference on automated software engineering*, pp. 43-52.

Tohalino, J.V., & Amancio, D.R. (2017). Extractive Multi Document Summarization using Dynamical Measurements of Complex Networks. *arXiv preprint arXiv:1708.01769*.

Wang, F.L., & Yang, C.C. Impact of document structure on hierarchical summarization. (2006). In *Springer's International Conference on Asian Digital Libraries*, pp. 459-469.

Bibliographic information of this paper for citing:

Malhotra, Mrinaal, & Chhabra, Jitender Kumar (2018). "Micro level source code summarization of optimal set of object oriented classes." *Webology*, 15(2), Article 175. Available at: http://www.webology.org/2018/v15n1/a175.pdf