Journal of Theoretical and Applied Information Technology
15th November 2017. Vol.95. No 21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SOURCE CODE ANALYSIS EXTRACTIVE APPROACH TO GENERATE TEXTUAL SUMMARY

[1]KAREEM ABBAS DAWOOD, [2]KHAIRONI YATIM SHARIF, [3]KOH TIENG WEI
[1,2,3]Department of Software Engineering and Information System
Faculty of Computer Science and Information Technology,
Universiti Putra Malaysia,
Selangor, Malaysia.
[1]Kareem.a.it@gmail.com, [2]khaironi@upm.edu.my, [3]twkoh@upm.edu.my

ABSTRACT

Nowadays, obtain program features becomes a hot issue in source code comprehension. A large amount of efforts spent on source code understanding and comprehension to develop or maintain it. As a matter of fact, developers need a solution to rapidly detect which program functional need to revise. Hence, many studies in this field are concentrating on text mining techniques to take out the data by source code analysis and generate a code summary. However, in this paper, we attempt to overcome this problem by propose a new approach (Abstract Syntax Tree with predefined natural language text Template (AST-W-PDT)) to generates human readable summaries for Java methods role. This paper describes how we developed a tool that the java source code can be summarized from the methods role. In evaluating our approach, we found that the automatically generated summary from a java class 1) is helpful to the developers in order to understand the role of the methods and will be useful, and 2) the automatically generated summary is precise**.**

Keywords: *Source Code Summarization, Program Comprehension, Source Code Maintenance, Abstract Syntax Tree*

1. INTRODUCTION

Software engineering can be defined as the process of analyzing the software in order to improve the efficiency. The outcome of the processing can be providing explanation and recommendation for improving certain system performance. To do so, intensive overall analysis should be considered on the important features of the source code. Thus, developers spend a lot of time for reading and exploring source code to understand it. However, program comprehension studies show that the developers would prefer to concentrate on specific part of source code during maintenance of the software, and try to keep away from comprehension of whole system. Therefore, analyzing features system provides worthy understanding of source code which simplify the process of reuse and modification that would be applied on certain code.

As a result the developers use skimming strategy on source code, for example by reading only the signatures or significant keywords in the methods in order to save their time. Skimming strategy will help programmers to grasp the source code, but the cons are the knowledge obtained cannot simply available to other developers. Reading a summary is an alternative of skimming source code. Summary including descriptive sentences that highlighting the most important source code functionality [1]. Thus, documentation is crucial for developers.

However, developers are lack of time to handle documentation so becomes outdated over the time, and it is expensive to produce and keep maintained. Therefore, automated solutions are required [2]. One solution is to use textual descriptive source code summary helps to grasp code semantics accurately [3]. As a consequence, developers can review software systems quickly and decide which entities to analyze and modify. A few works already proposed to generate code summaries by adapting text summarization techniques [2]. Recent research has been done towards automatic textual descriptive source code summary [4][5][6][7].

In specific, research by Sridhara (2010) that employ natural language summary of Java methods [6], then summaries can be aggregated to generate documentation of the code. Although these techniques are already enough to provide good

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

summaries by finding suitable keywords (lexical information), they my present some limitation related to support method role. This role is not these techniques focus mainly on lexical dimension (e.g., Latent Semantic Indexing [8], term frequency inverse document frequency(tf-idf) model , Vector Space Mode [4], etc.) to detect relevant terms.

In this paper, we hypothesize that the current summary generators tools would be effectiveness if they considered and included the information from methods that describe the methods role like (method name, parameters, variables, invocations, and what the method return). We define "effectiveness" in term of developers that find the generated summaries to be helpful to convey the most important aspects of its intended functions. Then, a new approach is proposed to automatically generate descriptive summary that consider both lexical and methods role information. This approach works by collecting data from methods and then using these data with predefined natural language template to describe the role of methods. ASTPrser is used to identify and extract the data that we need to include it in source code summaries. We thought the summary will be more readable, understandable, and accurate.

To test our hypothesis, we conduct a case study that aims at addressing mainly two research questions: (i) RQ1: Does the automatically generated summary from a java class helpful to the novice developer to understand the role of each method and will be useful?; (ii) RQ2: How well does the automatically generated summary in terms of preciseness, in having unnecessary information and in types of missing information?. This case study aims to compares source code summary that produced by our tool with perspective of an experienced programmers who is expected to perform some specific maintenance tasks.

The remainder of the paper is organized as follows. Section 2 describes the related works. Section 3 presents the research methodology, Section 4 presents the experimental procedure and setup and Section 5 conclusion and future work.

2. RELATED WORKS

Nowadays, researchers pay more attention in terms of applying information retrieval approaches for identifying feature location, and extracting identifiers from the source code. For instance, Marcus (2010) has proposed a Latent Semantic considered by adapting existing textual source code summarization technique. In fact

Indexing (LSI) method for software engineering applications. Such method aims to classify the portions of the source code by identifying the similarity among such portions. The authors have linked the concepts with each other in a matrix of similarity [14].

However, one of the challenging task that facing the mapping the between query typed by the developer and the relevant portion within the source code is the multi-word identifiers. Obviously, many identifiers are being declared with multiple words. Since the programming languages hinder the developer to separate the multi-word identifiers by a blank space therefore, developers tend to use multiple approaches for the separation whether using punctuation, digit or using CamelCase. Hence, there is a vital demand to accommodate a separation process in order to divide the multi-words identifiers into their original form. D. Lawrie (2011) has addressed this problem by proposing an approach for handling the process of dividing multi-word identifiers automatically. They have used regular expression approach in order to exploit the CamelCase and special characters such as 'underscore' [36]. Regular expression aims to examine the morphology of the word in terms of specific condition such as containing special characters or capitalization [12].

Recently, McBurney (2016) work on contextual information to generate code summary by using the algorithm of Page Rank to compute the call graph of the program in addition with SWUM is used to present novel approach that use java method to generate the summary, this approach is differ from the other by summarizing the context surrounding a method rather than using details from the internals of the method, then NLG system is used to create text of natural language [7].

In the same scene, Y. Liu (2014) focus on the linguistic information, latent semantic indexing (LSI) and clustering to group the source code artifacts with similar vocabulary is used to generate the summary based on the analyze the composition of each package in a program. Latent semantic indexing is a standard technique in information retrieval to index, retrieve and analyze the textual information; the authors locate the linguistic topics in a software document by applying LSI, then clustering the source artifacts based on their similarity. They use the identifier names and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

comments to generate the term by document matrix in LSI. The splitting identifier names techniques is used to split the different identifier convention. The authors generate the source code summary at a package level by helping of using Minipar and a natural language parser [11].

While Chitti babu K (2016) proposed novel Entity based source code summarization technique (EBSCS) which is based on classes, methods, and comments entities in the target code, the generated description of entities and comment lines are used to generate summary for the target code. The proposed technique consist two phases, the first one is the Extraction Phase which is all entities including comment lines are extracted from the target code, while Summary Generation phase is the other one, which is rely on the semantic content extracted from previous phase to create a text document[13].

P. Rodeghero (2015) mentioned that selecting a subset of statements and keywords is the current technique that used to generate code summarization; therefore they focus on improving the process of those selections. The authors present ten professional Java programmers eye-tracking study, in which those professional programmers wrote English summaries based on reading Java methods. The findings are applied to build a novel summarization tool. To identify the keywords and statements that the programmers focus on P. Rodeghero et al. analyzed the programmers' eye movements and looks fixations [3].

Haiduc (2010) has presented a leading work for source code summarization; they proposed an automatic generation approach based on the extractive summaries of the source code, they obtain the extractive summary by selecting the most important information from the contents of document. The lexical and structural information from the source code are the base of their approach [5].

In the same sense Haiduc (2010) has been combine several text summarization techniques, based on text retrieval (e.g., LSI, Vector Space Model, etc.), to generate source code summaries by finding the suitable relevant terms. The authors mentioned that a combination of automated text summarization techniques is more reliable for source code and helps in better program comprehension. They focused on investigating the suitability of several summarization techniques, mostly based on text retrieval methods, to capture source code semantics in a way similar to how developers understand it [16].

Another similar work was proposed by Moreno (2013) they proposed a technique to automatically generate natural language descriptions for java classes, presuming no documentation of the code exists. The tool determines the class and method stereotype and uses them in conjunction with heuristics to select which information to be included in the summary. The tool takes a Java project as input, and for each class, it outputs a natural-language summary. The authors considered that summary is based on the stereotype of the class; they proposed J Summarizer, which is an Eclipse plug-in that automatically generates natural language descriptions of Java classes [8]

Hill (2009) presented a novel approach that automatically extracts natural language phrases from source code identifiers and organizes them in a hierarchy. They proposed an algorithm to automatically extract and generate noun, verb, and prepositional phrases from method and field signatures, capturing word context of natural language queries. These phrases naturally form a hierarchy that allows the developer to quickly identify relevant program elements by reducing the number of relevance judgments, while the phrases help the developer to formulate effective queries [15].

Sridhara (2010) has been presented an automatic technique for identifying code fragment that implement high level abstraction of actions and expressing them as natural language description, their approach was the first for identifying code fragments of statement sequences, conditionals and loops that can be abstracted as a high level action[9].

3. RESEARCH METHODOLOGY

This section provides the illustration of the research methodology that has been used in this study in order to accomplish the objectives. As shown in Figure 1, the research methodology is composed of four main phases. These phases will be tackled one by one. Section 3.1 discusses the first phase of the research methodology which is 'Problem identification'; Whereas, Section 3.2 illustrates the second phase which is 'Design'. Section 3.3 concentrates on the third phase which is 'Implementation'. Finally, Section 3.5 focuses on the fourth phase which is 'Evaluation'.

Journal of Theoretical and Applied Information Technology
15th November 2017. Vol.95. No 21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

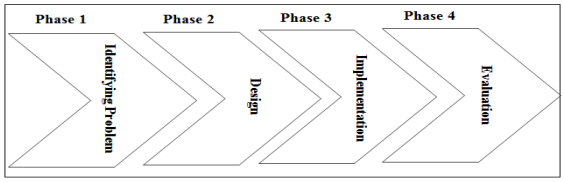

Figure 1: Research methodology phases

3.1 Phase 1: Problem Identification

This phase is associated with identifying the problem statement of the research in which the literature review will be investigated in order to identify gaps and ongoing challenging issues. Basically, an extensive literature review has been performed where the Feature Location task and analyzing the functions in order to provide summary for the source code has been illustrated in detail. Consequentially, splitting the identifiers which is a sub-task of a feature location has been illustrated in detail as a task where it requires recognizing the identifiers and then splitting such identifiers. A review for the existing approaches and techniques have been conducted which leads this study to figure out that there is still room for improvement in terms of the accuracy of source code summary. Hence, the problem has been formulated.

3.2 Phase 2: Design

This phase aims to identify a solution for the problem formulated in the previous phase in order to set it as the research objective. This can be performed by examining the existing approaches for code summarization in terms of the techniques that could be used for such task. In this vein, the techniques that have been used for summarizing code are being reviewed in terms of accuracy, readability, and understandability of summery that provide. This is to facilitate selecting an appropriate technique as an objective of this study. The reviews of these techniques have been conducted in Chapter II, and a conclusion has been attained. Such conclusion implied that the Java language parser for creating abstract syntax trees (ASTParser) technique tend to be the most appropriate technique for a detailed tree representation of the Java source code in order to generate the source code summary. However, Eclipse's Java Development Tools (JDT) is a powerful set of libraries. Of particular interest is the Abstract Syntax Tree (AST) API that it has which is extremely robust and full-featured. We can generate an AST representation of existing code (for modification or analysis) [17].

3.3 Phase 3: Implementation

This phase aims to carry out the research objectives produced from the previous phase. In order to do so, multiple sub-phases should be applied to accomplish the objective. As shown in Figure 2, there are three main phases: java source code, preprocessing, and summary generator, and those phases broken out to five sub-phases consisting of Source code under java source code phase, Transformation (AST maps java source code to tree form), Obtaining Information from an AST Node by ASTVisitor, Splitting Identifier under preprocessing phase, and Summary generator under summary generator phase. Source code phase discusses the source code that used in the experiment. Whereas, Transformation (AST maps java source code to tree form) sub-phase discusses the preparation tasks that have been conducted in order to turn the source code into an appropriate form for representation. Obtaining Information from an AST Node by ASTVisitor sub-phase discusses the obtaining information task is that have been conducted in order to extract the names of class, method , variables, method invocation, and return value by method to enhance the process of generating summary. While splitting identifier sub-phase has been use CamelCase mechanism to separate the multi-word identifiers. Finally, Summary generator phase is associated with the predefined natural language templet to generate accurate summary.

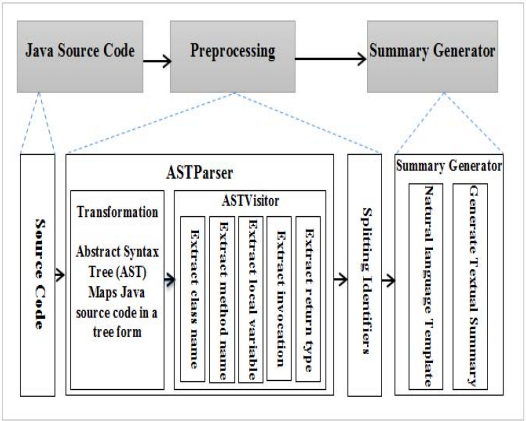

Figure 2: Implementation phases for AST-W-PDT Approach

3.3.1 Java source code

Java projects consist of a set of packages. Each package mainly contains a set of classes. In software design, each package is considered as a

Journal of Theoretical and Applied Information Technology
15th November 2017. Vol.95. No 21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

subsystem that has related set of classes that consist to set of methods to provide a certain functions. In a general view, the proposed approach automatically extracts a textual summary for the methods provided by a java class. So, the source code of each method is analyzed to extract a textual summary about its role. The summary of a method will generate from its contents. The contents include method name, method parameters, local variables, class variables (data field), methods' invocations, and return value types.

This section aims to determine the steps of acquiring the data (java source code). The java source code file (.java file) has been selected from the location that is saved in pc.

3.3.2 Preprocessing

This sub-phase include the following:

3.3.2.1 Transformation (AST maps java source code to tree form)

As mentioned earlier, the source code should be transformed into an internal and appropriate representation in order to facilitate the feature extraction. This can be represented by Abstract Syntax Tree (AST) technique that is used to map a tree model that entirely represents the source code provided as a tree of AST nodes. This tree is more convenient and reliable to analyses and modify programmatically than text-based source. Each Java source element is represented as a subclass of the ASTNode class. Every subclass of ASTNode contains specific information for the Java element it represents. E.g. a Method Declaration will contain information about the name, return type, parameters, etc. The information of a node is referred as structural properties. The example and Figure 3 below shows a closer look at the characteristics of the structural properties and the properties of the method declaration [17].

```
public void start(BundleContext context) throws
Exception {
        super.start(context);}
```

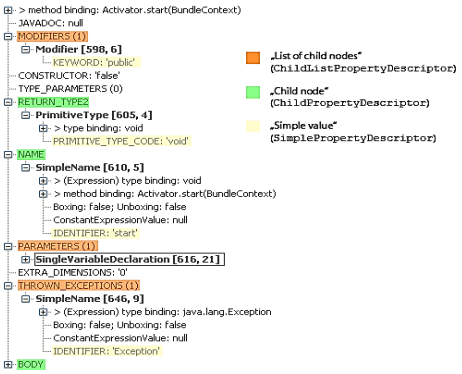

Figure 3 Structural properties of a method declaration

As shown in Figure 3.3 the structural properties are grouped into three different kinds: (i) properties that hold simple values, (ii) properties which contain a single child AST node and (iii) properties which contain a list of child AST nodes. Figure 4 show the Structural Property Descriptor and subclasses[17].

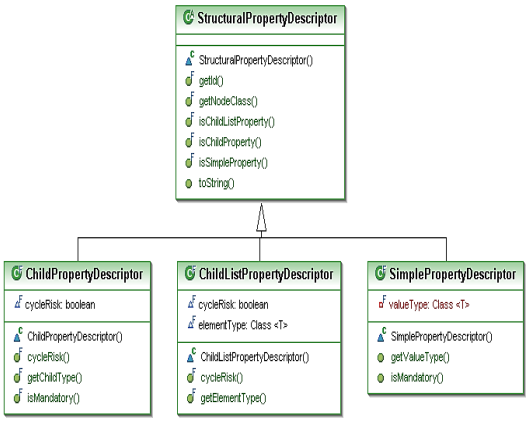

Figure 4: Structural Property Descriptor and subclasses

- Simple Property Descriptor: The value will be a String, a primitive value wrapper for either Integer or Boolean or a basic AST constant.

- Child Property Descriptor: The value will be a node, an instance of an ASTNode subclass.

- Child List Property Descriptor: The value will be List of AST nodes.

3.3.2.2 Obtaining information from an AST node by ASTVisitor

Basically, features can be defined as the characteristics and properties of each instance where specific description of the instance can be

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

depicted [18]. Features play an essential role in terms of the summarization where the significant feature that has the ability to accurately describe the instance would definitely improve the accuracy of the summarization. For this purpose, every ASTNode allows querying for a child node by using a visitor(ASTVisitor). There will find for every subclass of ASTNode two methods, one called visit(), the other called endVisit(). Further, the ASTVisitor declares these two methods: preVisit(ASTNode node) and postVisit (ASTNode node). The subclass of ASTVisitor is passed to any node of the AST. The AST will recursively step through the tree to provide the information that extracted from the methods to describe the role of each method. The information that provided by ASTVisitor are:

- Class name.
- Method name and parameters (method signature based).
- Method local variables.
- Class variables (data field) that used by the method
- Invoked methods.
- Method returns type.

3.3.3 Splitting identifier

Since all the programming languages do not allow the developer to declare an identifier with blank space such as “Compute Area", the developer tend to use multiple mechanisms to avoid such limitation. Sometimes, developers use special characters such as underscore in order to separate multi-word identifiers for instance, turning ‘Compute Area’ into ‘Compute_Area’. Furthermore, the developers may use CamelCase mechanism to separate the multi-word identifiers. This mechanism aims to capitalize the first letter of the first word, as well as, capitalize the first letter in the second word, and leaving the rest letters in a lower-case for instance, turning ‘Compute Area’ into ‘ComputeArea’ [10]

3.3.4 Summary generator

In this phase, the predefined natural language templet is used with ASTParser technique to generate the summarization of the source code. The process starts by reading the java source code, in the second step, the source code will transformed to a tree model that entirely represents the source code provided. The result will parsed to extract names of class, methods, data fields, local variables, and invoked method in addition to what the method return. Then, all identifier will be splitted. Finally a summary will be generate for each method by filling the predefine natural language template [9]. Finally, all methods summaries will refined and integrated as one. The following subsections detail the proposed approach and illustrate the idea with examples.

As shown in the code below, the class contains two methods. This means that it provides two different roles. The first method named getArea is split into “get Area” and the second method named getperimeter is split into “get perimeter”. Both methods, in the code, use data fields from the class one with local variable and the other is not. The two methods also do not invoke any other methods.

```
public class Circle {
  double radius = 1;
  double getArea() {
     double  area=radius * radius * Math.PI;
     return area;
  }
  double getPerimeter() {
     double Perimeter= 2 * radius * Math.PI;
     return Perimeter;
  }
}
```

For each method, its local variables and method invocations will included in the generated text. The generated summaries for the two methods in code above are:

- The role of the method is: get area for Circle. The method uses local data: area the method uses the attribute of Circle: radius. The method returns area.
- The role of the method is: get perimeter for circle. The method uses the attributes of circle: radius. The method returns perimeter.

The bold texts are templates that included in all generated summaries. The name of the method is followed by the words of the class. The names of used data fields and local variables are also included in the summary. The data fields used in the method are preceded with "the attributes of" in the summary followed by the class name. The class name is also split into words based on camel case naming convention. On the other hand, local variables are preceded with “local data” in the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

summary and method return type types are preceded with the word "types".

```
public class LeftArrowIcon implements Icon {
 public void paintIcon(Component      c, Graphics g,
int x, int y){
int  w=getIconWidth(),h=getIconHeight();
  g.setColor(Color.black);
  Polygon p = new Polygon();
   p.addPoint(x + 1,y + h/2+1);
   p.addPoint(x + w, y);
   p.addPoint(x + w, y + h);
   g.fillPolygon(p); } }
```

The above code shows another example for method paintIcon that is defined in class LeftArrowIconin.

The method uses local variables, returns type and invokes some other methods. The generated summary for the method, based on the proposed approach, will be as follows:

•The method role is: paint Icon for Left Arrow Icon. The method uses local data: w, h, x, y. The method uses types: Icon, Component, Graphics, Color, Polygon. The method get Icon Width, get Icon Height, set Color, add Point, fill Polygon.

The summary includes the name of the method at the beginning. It also lists both local variables (w, h, and y) and used return types (Icon, Component, Graphics, and Polygon). Local variables are distinguished by the words “local variable". Finally, the names of invoked methods are included after the words "The method". Including invoked methods enhance the generated summary and make it more meaningful. Actually, invoked methods participate in shaping the behavior of the method.

3.4 Phase 4: Evaluation

This phase aims to evaluate the proposed approach in which the results obtained from the code summarization process will be assessed. To measure the quality of the automatic summaries, we performed an intrinsic evaluation, which is one of the standard evaluation approaches used in the field of text summarization[19]. This evaluation involves the active participation of human judges, who rate each of the automatic summaries based on their own perception of its internal quality.

4. EXPERIMENTAL PROCEDURE AND SETUP

In this section, we perform the evaluation of the code summary which is generated by tool which is implemented as described in Chapter 3, evaluation has been highly important because it allows researchers to assess the results of a summarization approach, identify and understand the drawbacks of a particular summarization process. This chapter reports on a preliminary case study that was our first step towards the utilization of text summarization tool for generating summaries of source code. We designed an experiment where we investigated for the first time the use of a tool for generating extractive summaries.

4.1 Study Design

The main goal of this study is to investigate to what extent the generated code summaries meet the perspective of a novice developer who is expected to perform some specific maintenance tasks, and how impactful will these summaries in real situations. The quality focus is about the readability, understandability, and accuracy of the summaries that generated by code summary tool. The effectiveness of the summary for a java class is evaluated when it is useful to explain the class methods role to a novice developer who is expected to perform some specific maintenance tasks practically.

We perform the study with a questionnaire, where a Likert scale is used to determine the answers. We used the Likert scale with 5 scale rating: Strongly agree, Agree, Neither agree nor disagree, Disagree and Strongly disagree. This study is conducted in the form of an interview. We designed research questions that are answered during the empirical study.

4.2 Research Questions

This study aims at addressing mainly two research questions: (i) RQ1: Does the automatically generated summary from a java class helpful to the novice developer to understand the role of each method and will be useful? We want to assess to what level the novice developer can understand a part of system role from these automatically generated summaries, it meant the usefulness of the summary to novice developer ; (ii) RQ2: How well does the automatically generated summary in terms of preciseness, in having unnecessary information and in types of missing information? We want to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

assess if the automatically generated summary contains unnecessary information, and if it is missing any type of information when compared to source code, which will help in improving the understandability of the summary, it meant closeness of the summary to the source code.

As mentioned previously, the above questions have a scale to measure opinion of the participants. Since, we are interviewing the participants some questions are open-ended, which does not have a scale but the participants can answer the question openly. Therefore, we prepared the below questions to know more about the automatically generated summary:

• **Q1**: What improvements can be made on these summaries to attain better understandability about the method role? We want to know if there is any possibility to increase the understandability about the role of method by adding extra information.

• **Q2**: Will you use this tool to automatically generate summaries in the future and why? We want to know if the generated summary tool is useful for the developers.

4.3 Study Context

The context of this study contains of (i) object i.e., a Java test class extracted from Java open-source project and (ii) participants who will test the specified object. The participants recruited for this study are the postgraduate students from the Faculty of Computer Science and Information Technology at University Putra Malaysia (UPM) are consulted to know their interest to participate in the study. We request 15 participants asking their interest to participate in our survey. The information about their work or education was collected during the interview. Of them, 3 were developers from industry experience and 12 were having different experience as developers. Out of 15 employees, 3 participants have 6 to 10 years' experience, 3 participants have 3 to 5 years' experience, 6 participants has less than two years of experience, and 3 participants has less than 1 year experience . Table 1 shows the participants with their work experience.

Table 1: Participant Working Experience

Working Experience	Number of participants
0	3
1-2	6
3-5	3
6-10	3

4.4 Experimental Procedure

The experiment was organized by conducting a face-to-face interview with the questionnaire. An example of the survey can be found in the appendix. The actual survey document each participant received has three parts: (i) introduction about survey and instructions to perform, (ii) questionnaire about participants work or education and (iii) questionnaire. Before the survey, we explained to participants what we expected them to do during the survey: they were asked to read and understand the code that we are going to automatically generated summary, then read and understand automatically generated summary to answer the questionnaire then answer the questions which followed it. The survey had 6 questions about the automatically generated summary, are described in table 2 the question 1 belong to RQ1, questions 2 to 4 belong to RQ2 and remaining are open-ended questions.

Each participant was asked a pre-study questionnaire about their work experience, designation and programming experience. The total duration of the interview was between 15-20 minutes on average. So the total time duration of the entire survey depends on the participant.

Table 2: Survey Questions

Question number	Survey Question
Q1	Does the automatically generated summary of a single test class help the novice developer to understand the role of each method?
Q2	Is any kind of information missing in the automatically generated summary when compared to a test class?

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Q3	Is this automatically generated summary precise when compared to the test class?
Q4	Is this automatically generated summary containing unnecessary information when compared to the test class?
Q5	What improvements can be made on the automatically generated summary to attain better understandability about the method role?
Q6	Will you use this tool to automatically generate summaries in the future and why?

4.5 Results And Discussions

In this section, we report the results of our survey, and answer the research questions formulated in section 4.2.

4.5.1 Results

RQ1: Usefulness of the summary to novice developer

This research question is answered using sub question Q1 described in table 2. Figure 5 depicts the bar graph with the number of opinion counts given by the participants. The first impression we get when we look at the results is that the number of participants agreeing that the summary is explaining the role of the method, is high. There are few participants who do not completely agree, and no participants who disagree or strongly disagree with the given statement. Figure 6 depicts the pie graph with the number of opinion counts given by the participants for RQ1, there are 6 participants who completely agree that given summary provides role of the method while 8 accept the statement. Remaining 1 have a neutral opinion about the statement. If we calculate the mean value of participants who agree for the statement, it is 0.933 i.e., 93.3% in total, while the value for disagreement is 0.0 i.e., 0% in total, value of neutral opinion is 0.067 i.e., 6.7 %. We then compared the working experience of the participant and the choice they made. The mean value of the participants working and agreed for Q1 is 0.80. The mean of the participants not working and agreed for Q1 is 0.20. This means that participants irrespective of their work experience agree that the summary will be useful. Therefore, we conclude that the automatically generated summary from a test class is helpful to the novice developer in order to understand the role of the methods and will be useful.

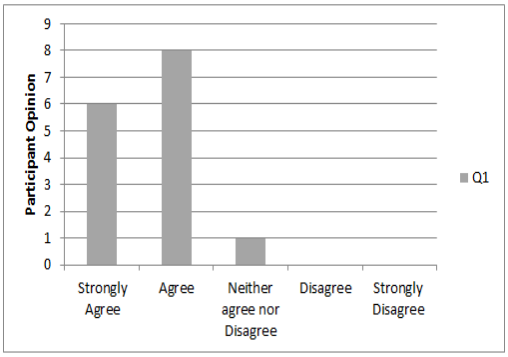

Figure 5: participants answer for Q1

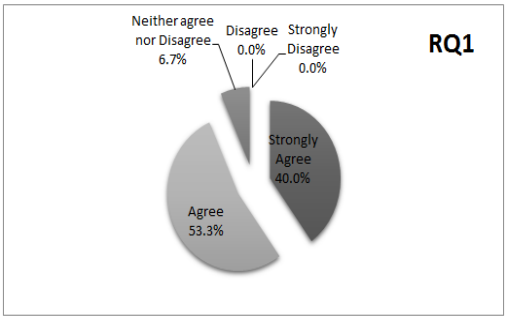

Figure 6: RQ1 usefulness of the summary based on Q1

RQ2: Closeness of the summary to the source code

To answer the question RQ2, we need the results of Q2, Q3, and Q4 from table 2. The results for those questions are depicted in the Figure 7 with a bar graph with the number of opinion counts given by the participants, determined from (i) missing information, (ii) preciseness and (iii) unnecessary information present in the automatically generated summary. A positive response for question Q3 and a negative response for questions Q2, Q4 describes that our generated summary is closer to source code and methods role.

For question Q2, from the Figure 8, the number of participants who agree and neither agree nor disagree are same with 1 each, and remaining 13 disagreed. This says that, 6.7% believe that there is some information missing from the automatically generated summary, and 6.7% has neutral opinion, while 86.6% believe that there is no information missing from the automatically generated summary.

For Q3, by looking at the graph in figure 7, we can say that more participant agree with the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

statement. 8 participants agree and 6 participants are completely agreed while rest has neutral opinion. This means, 93.3% of the participants accept that our automatically generated summary is detailed, and 6.7% has neutral opinion as shown in figure 9.

For Q4, we can find that 7 participants disagree, 6 participants completely disagree and remaining had neutral opinion. These shows, 86.7% of the participants say that our automatically generated summary does not contain any unnecessary information, and 13.3% has neutral opinion. The mean value to agree that RQ2 is 0.867, for disagree is 0.0 and for neutral opinion is 0.133 as shown in figure 10. Therefore, we conclude that the automatically generated summary is precise.

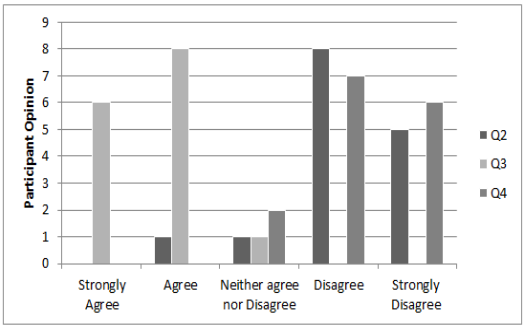

Figure 7: participants answer for Q2, Q3, and Q4

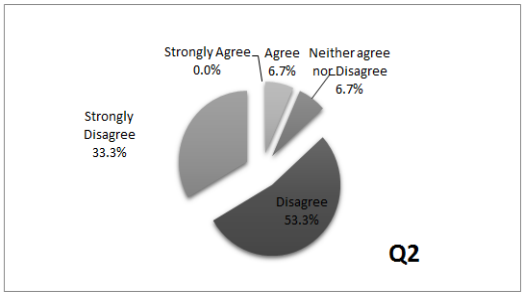

Figure 8: Q2: Is any kind of information missing in the automatically generated summary when compared to a test class?

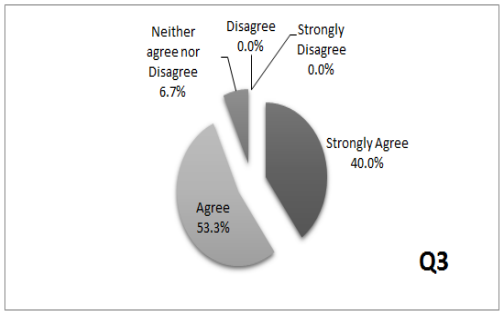

Figure 9: Q3- Is this automatically generated summary precise when compared to the test class?

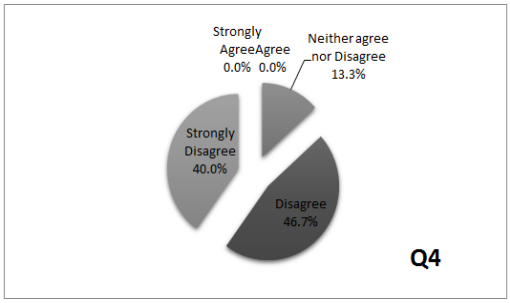

Figure 10: Q4- Is this automatically generated summary containing unnecessary information when compared to the test class?

4.5.2 Discussion

In the following, we provide qualitative results reported in Section 4.5.1. At the end of each question in the survey, all the 15 participants were asked the reason for the given Likert scale opinion. The reason for each participant's opinion is discussed here.

4.5.2.1 Usefulness of the summary: The participants who agree (93.3%) that the summary provides description about the methods role can be formulated as "the summary gives the sense of methods role". While the response of the participants who had a neutral opinion (6.7%) can be said as "the summary should more elaborated ". Hence, we can say that the summary will be useful for novice developer in understanding the system behaviour.

4.5.2.2 Closeness of the summary: The total percentage of participant who agrees that the summary was missing some information is 6.7%, and the total percentage of participant who has neutral opinion is 6.7%. While the 86.6% who say that summary is not missing important information were all outcomes are present where necessary; and explanation is well enough for developers. The participants who agree (93.3%) that the summary is precise as the summary has: "brief overview of what is happening in a system is present in an understandable way; and presented in a simple and compact way with necessary information". There were 86.7% of the participants who agree that the summary does not contain unnecessary information as the summary is containing useful information to understand the system. Remaining 13.3% had neutral opinion.

From above discussed points, we can say that most of the participants feel that summary is more favourable in terms of accuracy and information

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

present in it for the novice developer. Hence we conclude that our automatically generated summary is closer to a source code.

4.6 OPEN QUESTIONS

As mentioned in the section 4.2, we have prepared two open-ended questions which are answered after the research questions.

Question Q1 from Section 4.2 is questioned as Q5 in the survey and was about improvements that can be done to the summary. Very few participants were answered this question, and they suggested the following:

- Few participants were concerned expected more elaborate explanation for each line in the source code.

- Participants with work experience recommended producing summary for different programming language rather than java programming language only.

- Few participants recommended considering package base as input, rather than class base.

Question Q2 from Section 4.2 is questioned as Q6 in the survey and was about if the participants are willing to use our tool to generating summaries in the future. All participants said that they will use it in future, and it will be more useful if the suggested improvements are added to it.

4.7 THREATS TO VALIDITY

This section describes the possible threats to validity of our study and how we solved them.

4.7.1 Construct Validity

Threats to construct validity mainly concern on how we set up the study. Due to the fact that all the participants involved in our study need to have a prior knowledge about the code summarization. Lack of knowledge about code summarization will produce incorrect results. To handle this, before starting the survey we present a brief overview on code summarization that would be sufficient to do our survey. Then before starting the survey, we ask the participant if they understand the concept of code summarization to continue the survey.

4.7.2 External Validity

Threats to external validity concern the generalization of our results. It is important to point that the object i.e., test class which is summarized could influence the results of our survey. The evaluation here is limited to the summary of a single class only. Another threat is the size of the participants used for this study, as larger set of participants would increase the confidence about the survey results. Therefore, the results should be taken only as guides for further user studies.

5. CONCLUSION AND FUTURE WORK

This section provides the conclusion of the study in which Section 5.1 shows the final conclusion of the research, Section 5.2 shows the research contribution and Section 5.3 discusses the future work that could be proposed.

5.1 Conclusions

Current software systems must be continually changed to meet new requirements and adapt them to changing conditions in their operating environment. Additionally, it is widely accepted that effort and time spent understanding parts of a software system are a significant proportion of the resources needed to maintain and evolve existing code. Developers responsible for maintenance tasks are faced every day with software systems with thousands lines of code. This situation is particularly problematic when developers have to deal with large systems developed by others and the code is the only source of information that is available and up to date.

Different maintenance tasks require different levels of code comprehension. Most developers strive to understand as much as it is needed to perform a given task – no more and no less. The amount of code understood at one time is often limited by the ability to memorize, recognize, and recall textual tokens from the source code and their semantics, which makes tool support essential for most software comprehension tasks. Modern IDEs, together with searching and navigation tools, recommendation systems and data mining tools, all help developers minimize their effort in identifying parts of code relevant to their task. A common feature of these tools is that they provide a list of source code elements (such as methods) to the user, which he still has to read and understand in order to make final decision on their relevancy to the task at hand. When the code is well documented internally

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(for example, a method has good leading comments, meaningful name and parameters), it is frequently sufficient to see such comments and the method header to determine if it is relevant or not. However, more often than not, comments are missing, or out of date, and method headers contain words that the developer is not familiar with. In such cases, developers have little choice but to read the implementation and sometimes more than that.

We propose to provide help to developers in such situations. The goal is to supply them with a description of the code (such as the abstract of an article), which is more informative than the header and the leading comments, yet much shorter than the implementation, while capturing the essential information from it. Such descriptions will not replace reading the code when it needs to be understood, but they can save unnecessary effort spent reading irrelevant parts of it.

This paper explored the use of text summarization technology for automatically generating such descriptions of source code. Specifically, we studied how people describe code artifacts using term-based and sentence-based approaches, we adapted extractive technique for automatic code summarization, and also, we used evaluation to assess the usefulness of automatic summaries.

With regard to automatic generation of summaries, the paper describes new approach for creating short and accurate textual descriptions for the method role. We proposed, present, and validated an approach for summarizing methods that uses a combination between abstract syntax tree parser (ASTParser) and predefined natural language text template.

We conducted the survey by interviewing the participants and using a Likert scale to answer the survey questions. The survey was conducted to assess to what level the novice developer can understand a part of system role from these automatically generated summaries, and assess if the automatically generated summary contains unnecessary information, and if it is missing any type of information when compared to source code, which will help in improving the understandability of the summary.

The results of the evaluation this approach indicate that the automatically generated summary from a java class is helpful and precise to the novice developer in order to understand the role of the methods, it achieving 93.3% of participant agreement. Thus, these summaries can be useful for improving software comprehension processes, which usually occur during software maintenance tasks. In conclusion, it can be said that the automatically generated summary is precise without unnecessary information and is helpful to the developers in order to understand the role of each java method.

5.2 Future Work

This section discusses the possible works that would be inspired for future trends. Such discussion will be conducted by providing suggestions and recommendations in terms of enhancing the source code summary. Such suggestions can be described as follows:

- Enhance the automatically generate source code summary approach to produce summary for different programming language rather than java programming language only.

- Enhance the automatically generate source code summary approach to considering a package base input.

- Develop benchmark for evaluation is crucial.

In the long term, we expect that the use of source code summaries will reduce the developers' cognitive effort during comprehension activities. This should positively impact the time of development, as well as the quality of the software produced. More than that, the summaries could be consumed not just by developers, but also by other tools. For example, we envision using the source code summaries to support tools for automatic reverse engineering of legacy code, software ontologies extraction, re-documentation, etc. We expect the summaries to be used by existing searching and navigation tools.

ACKNOWLEDGMENT

This research work has been funded by the Fundamental Research Grant Scheme (FRGS) under the Malaysian Ministry of Education (MOE) for the project no. 08-01-16- 1850FR (5524957). Sincere gratitude extends to the lecturers, students and other individuals who are either directly or indirectly involved in this project.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES

[1] B. Du Bois, "Towards an Ontology of Factors Influencing Reverse Engineering," 2005.

[2] S. W. Thomas, "Mining software repositories using topic models," *Proceedings of the 33rd International Conference on Software Engineering*. pp. 1138–1139, 2011.

[3] P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan, "An Eye-Tracking Study of Java Programmers and Application to Source Code Summarization," *IEEE Trans. Softw. Eng.*, vol. 41, no. 11, pp. 1038–1054, 2015.

[4] P. W. McBurney, "Automatic Documentation Generation via Source Code Summarization," *Proc. - Int. Conf. Softw. Eng.*, vol. 2, pp. 903–906, 2015.

[5] S. Haiduc, J. Aponte, and A. Marcus, "Supporting program comprehension with source code summarization," *2010 {ACM}/{IEEE} 32nd {International} {Conference} {Software} {Engineering}*, vol. 2, no. May 2016, pp. 223–226, 2010.

[6] C. Science and M. Studies, "J-Summarizer," vol. 7782, pp. 59–62, 2016.

[7] P. W. McBurney and C. McMillan, "Automatic Source Code Summarization of Context for Java Methods," *IEEE Trans. Softw. Eng.*, vol. 42, no. 2, pp. 103–119, 2016.

[8] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for Java classes," *IEEE Int. Conf. Progr. Compr.*, pp. 23–32, 2013.

[9] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for Java methods," *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng. - ASE '10*, p. 43, 2010.

[10] G. Sridhara, L. Pollock, and K. Vijay-Shanker, "Generating parameter comments and integrating with method summaries," *IEEE Int. Conf. Progr. Compr.*, pp. 71–80, 2011.

[11] Y. Liu, X. Sun, X. Liu, and Y. Li, "Supporting program comprehension with program summarization," *2014 IEEE/ACIS 13th Int. Conf. Comput. Inf. Sci. ICIS 2014 - Proc.*, pp. 363–368, 2014.

[12] E. Enslen, E. Hill, and L. Pollock, "Mining Source Code to Automatically Split Identifiers for Software Analysis ∗," pp. 71–80, 2009.

[13] Chitti babu K, Kavitha C., and SankarRam N, "Entity based source code summarization (EBSCS)," *2016 3rd Int. Conf. Adv. Comput. Commun. Syst.*, pp. 1–5, 2016.

[14] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An Information Retrieval Approach to Concept Location in Source Code," *Proc. 11th Work. Conf. Reverse Eng. (WCRE '04)*, pp. 214–223, 2004.

[15] E. Hill, L. Pollock, and K. Vijay-Shanker, "Automatically capturing source code context of NL-queries for software maintenance and reuse," *Proc. - Int. Conf. Softw. Eng.*, pp. 232–242, 2009.

[16] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 35–44, 2010.

[17] "Abstract Syntax Tree." [Online]. Available: http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/. [Accessed: 09-Apr-2017].

[18] D. Freitag, "Machine Learning for Information Extraction in Informal Domains," *Mach. Learn.*, vol. 39, no. 2/3, pp. 169–202, 2000.

[19] K. Spärck Jones, "Automatic summarising: The state of the art," *Inf. Process. Manag.*, vol. 43, no. 6, pp. 1449–1481, 2007.