Source Code Summate using Semantics and Query

Chitti babu K\textsuperscript{1},Vaishnavi S\textsuperscript{2}, Anjana Devi J\textsuperscript{3} & Ramaneeswari T\textsuperscript{4}
\textsuperscript{1,2,3}Asst.Professor, dept.of.CSE, RMKCET
\textsuperscript{4}Scientist-SD, SDSC SHAR, ISRO.

Abstract: In the software evolution process a developer must analyze the source code in order to understand the entities in it. In general this analysis is done manually which takes lots of time and is a tedious task. The other option is to use automated source code summarization techniques. In most of the cases the developer will not analyse the total code he needs to analyse only one entity like packages or classes or methods or other entities in the source code. But all the Existing techniques provide the summary of the total code. Large software applications contain huge number of packages, classes, methods and other program entities so summarizing the total code is time consuming and tedious process. In this paper we propose a novel source code summarization technique which is based on the developers query and semantic content of the source code. Based on the query the summary is generated for the entities which the developer is interested.

1. Introduction

During software maintenance, developers often cannot read and understand the entire source code of a system and rely on partial comprehension, focusing on the parts strictly related to their task at hand. When creating software libraries, most developers use source code summarization. This technique makes it possible to convey information about functionality to other software developers using the library.

Source code summarization has potential for valuable applications in many software engineering tasks, such as:

(a) Understanding new code bases. Often developers need to quickly familiarize themselves with the core parts of a large code base. This can happen when a developer is joining an existing open source project, or when a developer is evaluating whether to use a new software library. (b) Code reviews. Reviewers need to quickly understand the key changes before reviewing the details. (c) Locating relevant code segments. During program maintenance, developers often skim code, reading only a couple lines at a time, while searching for a code region of interest.

Recent studies have shown that developers spend more time reading and navigating the code than writing it [1-2]. During these activities developers often only skim the source code [3] (e.g., read only the header of a method and maybe the leading comments when available). When the code is well documented internally (e.g., good preceding comments, meaningful names and parameters), this is often sufficient to determine if it is relevant or not.

The two activities (i.e., skimming and reading the whole implementation) are two extreme tactics; the former is very quick yet it can lead to misunderstanding, while the later is time consuming. An obvious option is something in the middle, i.e., offering developers a description of the source code, which can be read fast and leads to better understanding. With such descriptions (or summaries), developers could look over software entities fast and would make more informed decisions on which parts of the source code they need to analyze in detail. Two main challenges emerge in this context: determining what should be included in these summaries and how to generate them automatically.

When developers are searching for code with a specific functionality, textual clues can help them determine which parts of the code they need to investigate.

In this paper we propose to summarize those entities in which the developer or analyzer is interested but not all the entities in a source code. This improves the performance of the system because most of the software applications involve lots of code.
2. Text Summarization for Source Code

Automatic text summarization is concerned with the production of a brief but accurate representation, called a summary, of one or more source documents, with the help of a computer program. Summaries need to be significantly shorter than the original document, while preserving the most important information in the document. The summary of a source code entity we are investigating is a term-based summary, which contains the most relevant terms for the entity found in the code. Summaries can be divided in two main categories: extractive and abstractive. Extractive summaries are obtained from the contents of a document by selecting the most important information in that document. Abstractive summaries, on the other hand, are meant to produce important information about the document in a new way, at a higher level of abstraction, and usually include information which is not explicitly present in the original document. In this paper, we focus on extractive summaries.

3. Existing Techniques

A wide variety of techniques have been proposed for producing summaries in text summarization. Some of the most successful ones include techniques based on the position of words or sentences in the source document, and techniques based on text retrieval (TR). Among the techniques based on the position of terms, the lead Summaries are the most frequently used and most successful, being often selected as a baseline for assessing new techniques. Lead summaries are based on the idea that the first terms that appear in a document (i.e., the leading terms) are the most relevant to that document.

Statistical based TR techniques have also been successfully used for text summarization [4-6]. Different studies of program comprehension show that programmers rely on good software documentation. [9, 12, 14]. Unfortunately, manually-written documentation is notorious for being incomplete, either because it is very time-consuming to create [7, 11], or because it must constantly be updated [8, 10, 13]. One result has been the invention of the documentation generator. A documentation generator is a programming tool that creates documentation for software by analyzing the statements and comments in the software's source code. The key advantage is that they relieve programmers of many tedious tasks while writing documentation. They offer a valuable opportunity to improve and standardize the quality of documentation. Still, a majority of documentation generators are manual.

JSummarizer is an Eclipse plug-in for automatically generating natural language summaries of Java classes [15]. The tool uses a set of predefined heuristics to determine what information will be reflected in the summary, and it uses natural language processing and generation techniques to form the summary. The generated summaries can be used to re-document the code and to help developers to easier understand large and complex classes.

The main drawback of this tool is that it works for only Java and it generates summary based on the stereotype. It work for only object oriented source code.

4. Proposed System

In this paper we propose a novel summarization technique uses simple mechanism which is based on the semantic of the source code and the query items as shown in Fig-1.

![Fig-1: Proposed System Sample View](image)

A. Query Extraction Phase

In this phase the details of entity are extracted including comment lines are from the source code. The extraction of comments involve the identification of the tokens // and /*. The extraction of the class is done by identifying the key word class. The extraction of method is based on the syntactical structure of method. The first line of a method definition "return type method name (argument list)".

B. Summate Phase

In this phase a text document is created from the semantic content extracted in the previous phase. This text file is used as input in this phase. This phase involves two main steps:

1. Convert the extracted text into a corpus.
2. Determine the most relevant terms for documents in the corpus and include them in the summary.
1. Source code corpus creation

Source code contains a lot of text, yet it is not entirely natural text, we need to convert it into a document collection. What is a document? For OO software, the obvious choices are methods and classes, though one can think of files and packages as well. In each case, the identifiers and comments in the source code entities are extracted. The next step is using a stop-words list to filter out terms which do not carry specific meaning. Such terms are conjunctions, prepositions, articles, common verbs, pronouns, etc. (e.g., “should”, “may”, “any”, etc.) and programming language keywords (e.g., if, else, for, while, float, char, void, etc.).

2. Determine the most relevant terms

The generated document contains many terms from this the most relevant terms are extracted and added to summary. The most relevant terms are identified by using a special matrix called relevancy matrix (RM) which represents the terms in rows and it has two columns the first column represents frequency count and the second column represents the relevancy ratio. The relevancy ratio is fixed to a threshold value based on which the terms are added to summary. For example if the threshold value for relevant ratio is 0.03, then all the terms whose relevant ratio is less than 0.03 are not added to summary.

The relevant ratio of each term is calculated using the below formulae

$$\text{Relevant ratio} (i) = \frac{1}{\text{LOC} - \sum_{i=1}^{N} \text{Frequency Count}(i)}$$

Where LOC stand for Lines of Code, N indicates the total no of terms and i indicates the term no for which we want to find the relevant factor.

For example LOC is 320, summation of all frequencies is 180 and threshold value is 0.03, suppose we want to find the relevant ration for third term in the document, the frequency count (i) is 10. Then relevant ration (3)=10/320-180.=0.07.since 0.07 > 0.03 the term is added to summary.

For example consider three words add, two and num, the RM Matrix contents will look as

<table>
<thead>
<tr>
<th></th>
<th>Frequency count</th>
<th>Relevant ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>10</td>
<td>0.02</td>
</tr>
<tr>
<td>two</td>
<td>30</td>
<td>0.08</td>
</tr>
<tr>
<td>num</td>
<td>5</td>
<td>0.01</td>
</tr>
</tbody>
</table>

The whole process involved in this technique is depicted in Fig.2. The Proposed system involves three main steps in the first step extraction of entities like classes, methods and control statements and semantics is done. In the second step the LOC is identified for each entity and in the last step the semantic content is stored in a text file, corpus is created and most relevant terms are identified. Based on these most relevant terms the summary is generated.

The summary generated by proposed system will be very useful for the developers to identify the entities in the program and to understand the logical aspect of the program. It makes the task of the developer easier by giving the line numbers of each entity and a formal description of the entities.

In general the summary generated by proposed system will look like as in Fig-3. In the first line the total LOC of the source code is displayed, the first tables speaks about the classes in the code, the second tables reveals the methods in the code and the third table gives the idea about the loops in the source code. At last the summary is display in the form of two to three lines.
Here CLOC stands for class lines of code and MLOCM indicates the Method Lines of Code. This parameter helps to identify the significance of the methods. Every entity is displayed in the form of a table giving the line no and it purpose. This makes the developer task easy in analyzing the code, the developer need not search the whole code for a particular method or loop he can directly go to the particular code by using line no, this reduces the time.

To evaluate the quality of the automatically produced source code summaries and to measure their capability to capture the developers’ understanding of the code, we performed a study in which developers judged the quality of a large set of summaries for methods and classes. The study was particularly aimed at assessing the impact of the various factors affecting the generation of code summaries on the summary quality.

5. Conclusion and Future Work

In this paper we have presented a novel approach for source code summarization which is useful for software developers and testers. This approach is based on the entities in which the developer is interested in the code. We found that our summaries were superior in quality and that our generated summaries providing contextual information.

We also plan to investigate multi-document approaches for the summarization of source code packages and classes. We plan to study how the generated summaries impact program comprehension by running studies where developers make use of such summaries during their daily tasks.

6. References


