On the Use of Automated Text Summarization Techniques for Summarizing Source Code

Sonia Haiduc[1], Jairo Aponte[2], Laura Moreno[2], Andrian Marcus[1]

[1]Department of Computer Science
Wayne State University
Detroit, MI, USA

[2]Dept. de Ing. de Sistemas e Industrial
Universidad Nacional de Colombia
Bogotá, Colombia

sonja@wayne.edu; jhapontem@unal.edu.co; lvmorenoc@unal.edu.co; amarcus@wayne.edu

Abstract— **During maintenance developers cannot read the entire code of large systems. They need a way to get a quick understanding of source code entities (such as, classes, methods, packages, etc.), so they can efficiently identify and then focus on the ones related to their task at hand. Sometimes reading just a method header or a class name does not tell enough about its purpose and meaning, while reading the entire implementation takes too long. We study a solution which mitigates the two approaches, i.e., short and accurate textual descriptions that illustrate the software entities without having to read the details of the implementation. We create such descriptions using techniques from automatic text summarization. The paper presents a study that investigates the suitability of various such techniques for generating source code summaries. The results indicate that a combination of text summarization techniques is most appropriate for source code summarization and that developers generally agree with the summaries produced.**

Keywords-text summarization, program comprehension

I. INTRODUCTION

During software maintenance, developers often can not 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 [1]. Recent studies have shown that developers spend more time reading and navigating the code than writing it [2-3]. During these activities developers often only skim the source code [4] (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. However, more often than not, good comments are missing and method headers contain words that the developer is not familiar with. In such cases, developers have little choice but to read the implementation of the artifact and even more than that, other related artifacts. This takes significant effort and time.

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 than the header alone. 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.

The domain semantics of a source code fragment (i.e., its purpose) are often encoded in comments and identifiers. 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. We advocate for including such textual clues in source code summaries. To that end we propose adapting and using techniques from the field of *automated text summarization* [5], in order to automatically generate source code summaries.

In this paper we are investigating the suitability of several summarization techniques, mostly based on text retrieval (TR) methods, to capture source code semantics in a way similar to how developers understand it. We present a case study that aims at determining how each technique impacts the quality of the summaries produced.

II. AUTOMATIC TEXT SUMMARIZATION FOR SOURCE CODE ENTITIES

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

1095-1350/10 $26.00 © 2010 IEEE
DOI 10.1109/WCRE.2010.13

and we also investigate a light variant of abstractive summaries.

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 [6-8]. We developed a framework that allows us to use these techniques to generate automated source code summaries. While obtaining the lead summaries is straight-forward, the framework for the generation of TR summaries is more complex, having two main components, with the following goals:

1. Extract the text from the source code of the system and convert it into a corpus.
2. Determine the most relevant terms for documents in the corpus and include them in the summary.

Each of the two components can be implemented in various ways, which we discuss here.

A. Source code corpus creation

Source code contains a lot of text, yet it is not entirely natural text. In order to use text retrieval techniques on source code, 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, which is optional, is splitting the identifiers according to common naming conventions. For example, "setValue", "set_value", and "SETvalue", are all split into "set" and "value". If identifiers are split, there is a choice whether to keep the original (unsplit) version of the identifier or not. The last step (typical in all TR applications) 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.).

B. Generating source code summaries using text retrieval

Once a corpus a created, several TR techniques can be used to generate the summaries. We implemented two variants: one based on the *Vector Space Model (VSM)* [9], and the second on *Latent Semantic Indexing (LSI)* [10]. Both techniques have been shown to work well for the summarization of natural language documents [6-8, 11-12]. They have also been used to index software corpora in a variety of comprehension applications [13-16].

While the two techniques generate summaries differently, the first step is the same in each case: represent the terms and documents in the corpus in a matrix where each row represents a term and each column represents a document. The content of a cell in this matrix represents the *weight* of a term (the row) with respect to a document (the column). These weights are generally a combination between a *local weight* and a *global weight*. The local weight captures the relationship between a term and a particular document and it is usually derived from the frequency of the term in the document. The global weight refers to the relationship of a term with the entire corpus and it is usually derived from the distribution of the term in the corpus. There are several options to choose from for both the local and global weights, resulting in numerous combinations. We identified the three combinations that perform the best in natural language summarization: *log*, *tf-idf*, and *binary-entropy*, for both LSI [6-7, 17] and VSM [8, 12] and used them in our study.

For generating a summary using VSM, the terms in the document (i.e., class or method) are ordered according to the chosen weight and the top K terms are included in the summary. LSI does not operate on the original term by document matrix, but it projects it to a smaller one which approximates it, corresponding to the highest Eigen values in the original matrix. Thus, LSI allows the representation of terms and documents in the same coordinates. In consequence, it allows computing similarities between terms and documents. These similarities are used to obtain the list of the most similar, i.e., most relevant terms to a document, which are then included in the summary. First, the cosine similarities between the vectors of each term in the corpus and the vector of the document to be summarized are computed. Then the terms are ordered based on this similarity and the top K terms, having the highest similarity to the document are included in the summary. One particular feature of the top K list of terms generated in this manner with LSI is that it can contain also terms that do not appear in the summarized method or class, but appear somewhere else in the corpus. When a summary contains such terms, it can be interpreted as a lightweight *abstractive summary*, as it produces information about the original document in a new form, i.e., which is not contained in the document.

III. CASE STUDY

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 (e.g., corpus creation, TR technique, term weight used, summary length, etc.) on the summary quality.

A. Subjects and objects of the study

In this study we asked four computer science students (coded as D1-D4) to evaluate code summaries automatically generated using VSM and LSI for methods and classes from two Java software systems. Each student has several years of programming experience.

The two Java systems we used in the case study are both available for download from Sourceforge. The first system, ATunes[1] (version 1.6), is a full-featured media player and manager having 221 classes, 1,852 methods, and 25,000 non-blank lines of text in its source code (including 5,000 lines of comments). The second system, Art of Illusion[2] (version 2.7.2), is a 3D modeling, rendering, and animation studio. It has 597 classes, 7,057 methods, and almost 100,000 non-blank lines of text (including 15,000 lines of comments).

For each system, we chose a set of 10 methods and their corresponding classes from the source code. The methods were chosen in such a way that they form a heterogeneous set for each system, based on the following characteristics of their implementation: length in lines of text (we chose methods from three categories: less than 20 lines, between 20 and 50 lines, and more than 50 lines), presence of comments, presence of parameters, type of class they belong to (entity, control, boundary), the presence of a return statement, the number of actions the method implements (one or more than one), and the stereotype of the method (accessor, mutator, creational, other). Detailed statistics about the selected methods and classes are available on our website[3].

B. Types of summaries

We generated summaries for each class and method using four different summarization techniques: lead, VSM, LSI, and a baseline technique, i.e., random summaries. The random summaries are formed by randomly selecting *K* terms from the identifiers and comments of a method or class and including them in the summary. For each of these techniques, we varied a series of parameters and observed the effect that these variations have on the quality of the generated summaries.

For all techniques we varied the number of terms (*K*) included in the summary. We generated summaries with 5 and 10 terms for each technique, as we believe that fewer than 5 terms capture too little information and having more than 10 terms pushes our short term memory limits [18].

For VSM and LSI we used the three weighting schemes mentioned before (i.e., *log*, *tf-idf*, and *binary-entropy*) for the terms in the term-by-document matrix. The terms and order of these terms in the summaries depend on the weight used. For the lead and random summaries there is no need to use such weights.

We also used three different ways to generate the corpora: (1) splitting identifiers and discarding the original form; (2) splitting identifiers and keeping the originals; (3) not splitting the identifiers.

Applying all these variations lead to the generation of 66 different summaries for each of the 40 methods and classes in the two systems, for a total of 2,640 summaries. These summaries were then analyzed and evaluated by the developers.

C. The evaluation of summaries

To measure the quality of the automatic summaries, we performed an intrinsic online evaluation, which is one of the standard evaluation approaches used in the field of text summarization [5]. 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.

In a first phase, the developers spent three days to familiarize themselves with the two software systems, focusing specifically on the methods and classes that were going to be summarized. Three of them had previous experience with adding features and fixing bugs in one of the two systems.

In the next phase, the four developers were presented with the 66 summaries for each method and each class and asked to answer the question *"How much do you agree with this automated summary of the method/class?"* In order to limit the possibility of a placebo effect on the participants, they were not presented with any information that would disclose the summarization technique or the parameters used to produce any of the summaries.

The developers had four options to choose from to answer the question, each being assigned a *score: strongly agree* (assigned a score of 4), *agree* (score of 3), *disagree* (score of 2), and *strongly disagree* (score of 1). This set of possible answers represents a four level Likert scale [19]. We decided not to use a five level scale with a middle option (i.e., *"neither agree nor disagree"*) in order to exclude the situation of central tendency in the answers due to non-committal answers (i.e., users choose the middle option because they do not want to choose a side).

In order to be able to understand the results better, in addition to choosing one of the above options for rating each summary, we also asked the developers to tag each of the terms that appear in a summary as either relevant or irrelevant for the method or class under analysis. The developers were provided with a form for each method or class, which contained the summaries, each followed by the list of terms in the summary. Next to each term in this list there was an empty box where developers were asked to write a 0 when they considered the term irrelevant, and a 1 when the term was relevant. The developers had access to the source code of both systems at all times and they were not given a time limit to finish the evaluation. To minimize the evaluation bias, developers were not given any other instructions on assessing the summaries or the relevance of the terms.

D. Follow-up questionnaire

In order to understand how the developers performed the evaluation and get more insight into the results of the study, we asked the developers to answer three follow-up questions. We report here on the answers we received from developers in the follow-up questionnaire in order to better understand the results that follow.

[1] http://sourceforge.net/projects/atunes
[2] http://sourceforge.net/projects/aoi
[3] http://www.cs.wayne.edu/~severe/wcre2010

The first question we asked was: *What was the process you followed when rating the summaries?*

Two of the developers, D1 and D2, followed very similar processes when rating summaries. They started by extracting a list of relevant words from the source code, i.e., the words they considered should appear in the summary. Then, they rated all the summaries based on the terms they had in mind and at the end they marked the terms as relevant or irrelevant.

D3 and D4 did not extract the relevant terms from the source code. D3 analyzed each summary and its terms individually, rating the summary first and then marking each of the terms as relevant or irrelevant. D4, on the other hand, first compiled the list of unique terms found in all summaries of a class/method and marked in this list the ones that were relevant. Based on this list, D4 then marked all terms in all summaries as relevant or irrelevant and then rated the summaries. All developers reported changing the initial ratings of some of the summaries when they compared summaries between them.

We observed that there is no connection between the process followed by developers to rate summaries and the scores they gave. For example, the pairs of developers (D2, D4), and (D1, D3), who followed different processes, had more similar ratings than (D1, D2), who followed a similar process for rating summaries.

The second question was: *What criteria did you use for rating the summaries?*

D1 considered the number of irrelevant terms contained in a summary as an important factor in the evaluation, whereas D2, D3, and D4 did not. This is corroborated by the data we collected, which reveals that D1 gave higher scores more often to 5-term summaries than to 10-terms summaries (10 times more often than D2, 12 times more often than D3, and 4 times more often than D4), as the latter contained more irrelevant terms. The other developers were less influenced by the noise in the summaries and focused on the relevant terms hence they rarely gave higher ratings to 5-terms summaries than to 10-terms summaries. The fact that D1 considered the irrelevant terms in his ratings led to a decrease in the average score, mode and sum of scores of his ratings for all summaries. For example, even if he gave the highest scores to the lead summaries among all summarization techniques, the highest average score for classes was 2.47, compared to 3, 3.06 and 3.59, as given by the other developers. In average, he did not consider any type of summary very good. For methods, his average rating of the lead summaries was 2.75, compared to 3.25, 3.10, and 3.70, as given by the other developers. More details on these scores are presented in the next section.

D4 reported that the order of the terms in the summary was important at times, as the summary was clearer when the relevant terms followed one another and when they were positioned at the beginning of the summary. D4 used the position of the relevant terms to differentiate between good and very good summaries. The other developers said the order of the terms in the summary did not impact their ratings.

When analyzing method summaries, all developers considered the method name to be the most relevant piece of information and ranked summaries having this idea in mind.

When scoring class summaries, D2 and D4, considered that the class name and attribute names were the most relevant information for a class and therefore should be present in the summary of a class. The other two developers considered that the names of the methods defined in the class are more relevant. By analyzing the scores given by each developer to the methods and classes in the case study, we found that D2 and D4, agreed the most in their ratings among all pairs of developers, for both classes and methods.

All developers reported noticing changes in their rating criteria between different scoring sessions and attributed them to changes in mood and fatigue. Also, D2 reported that once during the same session, when he did not order the summaries according to their first terms, he sometimes assigned slightly different scores to the same summary, when this appeared more than once among provided summaries (this happens when two different techniques generate the same summary for a method or class). For example, he remembered scoring the same summary with both a score of 3 and a score of 4 before ordering the summaries according to their content, but then realizing his inconsistency and synchronizing the scores.

The third question we asked was: *What was missing (if anything) from the summaries you analyzed?*

This question helps us understand what should be included in a summary, besides what was already mentioned as important in Question 2.

D1, D2, and D4 reported that they would always include good comments that appear before the class/method definition in the summaries. D3 thinks that if the identifiers are good, the comments are not needed. Even if this type of comments were mostly included in the lead summaries when available, they were generally not found in the other types of summaries. The fact that developers appreciate the leading comments could be one of the reasons for the good scores obtained by lead summaries, as discussed in the next sections.

All developers reported that they would always include at least a verb and an object in method summaries, which describe the action performed by the method and the object on which it is performed. Our future work on automatic summarization of source code will take these observations into account, as it will try to produce summaries as close as possible to what developers would like to see.

E. Results and discussion

The six lead summaries scored higher than any other individual technique (see Table I). This indicates that the terms found at the beginning of a class or method are generally relevant terms for the artifact. The maximum average score obtained by lead summaries when considering both methods and classes is 3.09, which is achieved for 10 term-summaries and unsplit identifiers. The minimum score obtained by lead summaries for both methods and classes is 2.68, and is achieved for summaries having 5 terms and

where the identifiers were split and the originals were not kept.

TABLE I. THE 24 HIGHEST RANKED TYPES OF SUMMARIES

The format of the technique name: AaNnBbbCc
Aa=technique (Le=Lead, Ls=LSI, Vs=VSM)
NN=number of terms (5 or 10)
Bbb=corpus splitting technique (Spn=Split No Originals, SpO= Split Keep originals, NSp=No Split)
Cc=weighting (Tf=tf-idf, Lg=log, Be = binaryEntropy)
M = methods, C= classes, MC = methods and classes

#	Technique name	Average score			St. deviation			Median		
		M	*C*	*MC*	*M*	*C*	*MC*	*M*	*C*	*MC*
1	Le10Nsp	3.25	2.94	3.09	0.77	0.97	0.89	3	3	3
2	Le10Spn	3.28	2.90	3.09	0.73	1.04	0.91	3	3	3
3	Le10Spo	3.21	2.86	3.04	0.76	1.00	0.90	3	3	3
4	Le5Nsp	3.08	2.45	2.76	0.88	1.01	0.99	3	2	3
5	Le5Spo	2.98	2.40	2.69	0.98	0.95	1.00	3	2	3
6	Le5Spn	2.98	2.39	2.68	0.94	0.93	0.98	3	2	3
7	Vs10SpoBe	2.43	2.34	2.38	0.82	0.99	0.91	2	2	2
8	Vs10SpoLg	2.40	2.31	2.36	0.89	0.92	0.91	2	2	2
9	Vs10NspTf	2.35	2.36	2.36	0.89	0.83	0.86	2	2	2
10	Vs10SpnLg	2.35	2.31	2.33	0.96	0.80	0.88	2	2	2
11	Vs10NspLg	2.33	2.28	2.30	0.85	0.86	0.85	2	2	2
12	Vs10SpnBe	2.39	2.19	2.29	0.83	0.90	0.87	2	2	2
13	Vs10SpoTf	2.30	2.24	2.27	0.88	0.83	0.85	2	2	2
14	Vs10SpnTf	2.26	2.23	2.24	0.81	0.84	0.82	2	2	2
15	Vs10NspBe	2.25	2.19	2.22	0.95	0.99	0.97	2	2	2
16	Ls10SpoTf	2.38	2.06	2.22	1.00	0.99	1.00	3	2	2
17	Ls10NspLg	2.39	2.01	2.20	1.05	0.91	1.00	2	2	2
18	Ls10NspTf	2.33	1.96	2.14	1.02	0.95	1.00	2	2	2
19	Ls10NspBe	2.29	1.95	2.12	1.01	0.90	0.97	2	2	2
20	Ls10NspLg	2.10	2.04	2.07	0.92	0.91	0.91	2	2	2
21	Ls10SpoLg	2.24	1.84	2.04	1.07	0.88	1.00	2	2	2
22	Vs10SpoBe	2.03	2.03	2.03	0.73	0.95	0.85	2	2	2
23	Ls10SpoTf	1.95	2.09	2.02	0.84	1.00	0.92	2	2	2
24	Ls10SpoBe	2.21	1.79	2.00	0.95	0.84	0.92	2	2	2

After analyzing the summaries and the developers' answers to the post-experiment questionnaire, we concluded that the high scores of the lead summaries are due to the terms in the headers of methods, the names of classes, and sometimes good leading comments, which are included in these summaries and considered important by developers. Among the TR techniques, the top 9 average scores (for both methods and classes) are obtained by the 10-term VSM generated summaries with a variety of options (see rows 7 to 15 in Table I).

For analyzing the difference in performance of the techniques over the entire data, we averaged the scores over all artifacts, developers, and parameter configurations. Lead summaries got an average of 2.89, VSM summaries 2.11, LSI summaries 1.75, and the random summaries 1.56 (remember that 1 corresponds to the *strongly disagree* rating and 4 to the *strongly agree* rating). We applied a statistical test to compare the means of the different techniques. Based on the characteristics of the arrays of values for the different summarization techniques (normal distribution, dependent values), we used a two-tailed paired t-test for this purpose. The resulted p-values were 0.02 for the pair of LSI-random summaries and 0.00 for all the other pairs, indicating that the differences between summarization techniques are statistically significant.

We also looked at the differences in the ratings given by developers between the two software systems. We found that the differences in ratings were minor and that there was a very high correlation between the scores assigned to artifacts in the two systems (0.98).

1) Combining the lead and VSM summaries

While the scores of the lead summaries are good, they also indicate there is room for improvement, i.e., there is relevant information not captured by the lead summaries. The TR summarization technique with the highest scores, VSM, is also far from perfect. However, the data support the observation that the VSM summaries contain relevant information valued by the developers, even if not as much and not the same as lead summaries do. One obvious missing term, for example, is the method name. VSM favors terms with high frequency in the document and in most cases method names, for example, appear only once in a method. It is likely that a VSM based summary will not include them.

We analyzed the VSM and lead summaries to see if any of the relevant terms they capture are not included in the other type of summaries. More specifically, we computed the intersection and union of the relevant terms between each VSM configuration and each lead configuration (see Table II) across all the summarized methods and classes in the two systems. We found that the number of relevant terms in the union is always greater than in each of the two techniques considered individually (in average, twice as many relevant terms as the lead summaries) and the intersection of relevant terms between VSM and lead summaries is empty in 1/3 of the cases. This fact strongly suggests that lead and VSM summaries capture different relevant information about the methods and classes.

TABLE II. THE AVERAGE NUMBER OF RELEVANT TERMS IN LEAD AND VSM SUMMARIES, THEIR UNION, AND INTERSECTION

M = methods, C= classes, MC = methods and classes

	Lead	VSM	Lead U VSM	Lead ∩ VSM
M	6.26	5.68	10.14	1.80
C	5.50	5.53	9.81	1.22
MC	5.88	5.60	9.98	1.51

As the number of relevant terms in summaries is strongly correlated with the scores obtained by the summaries (0.82 Spearman rank correlation), we expect that if the number of relevant terms in a summary increases, so does its score.

Based on the observations we made, our hypothesis was that combining the lead and VSM summaries will lead to better summaries and higher rankings from the developers compared to the individual summaries. In order to study if this is indeed the fact, we combined the lead and VSM

summaries by concatenating to the lead summaries the terms in the VSM summaries which did not already exist in the lead one. We obtained 18 new summaries for each artifact, as we performed every combination between lead and VSM summaries having the same number of terms. The same four developers ranked each of the new summaries using the same procedure they used for ranking the individual summaries. The results (see Table III) show that our expectations were met and that the combined summaries obtained higher scores than the individual summaries, some coming close to the maximum score. The highest average score across all methods and classes was 3.54, achieved for the union of the 10-term lead and VSM summaries, without splitting the identifiers and using tf-idf as the weight for VSM. The improvement in score was almost half a point, which made the transition from a good summary to an excellent summary.

TABLE III. THE LEAD+VSM SUMMARIES AND THEIR SCORES

The format of the technique name: AaNnBbbCc
Aa=technique (Le=Lead, Ls=LSI, Vs=VSM)
NN=number of terms (5 or 10)
Bbb=corpus splitting technique (Spn=Split No Originals, SpO= Split Keep originals, NSp=No Split)
Cc=weighting (Tf=tf-idf, Lg=log, Be = binaryEntropy)
M = methods, C= classes, MC = methods and classes

#	Technique name	Average score			St. deviation			Median		
		M	*C*	*MC*	*M*	*C*	*MC*	*M*	*C*	*MC*
1	LeVs10NspTf	3.66	3.43	3.54	0.59	0.74	0.68	4	4	4
2	LeVs10NspLog	3.61	3.4	3.51	0.58	0.81	0.71	4	4	4
3	LeVs10NspBe	3.59	3.38	3.48	0.61	0.82	0.73	4	4	4
4	LeVs10SpoBe	3.55	3.28	3.41	0.61	0.83	0.74	4	3	4
5	LeVs10SpoTf	3.55	3.26	3.41	0.61	0.82	0.74	4	3	4
6	LeVs10SpoLg	3.48	3.29	3.38	0.64	0.78	0.72	4	3	4
7	LeVs5NspBe	3.35	3.04	3.19	0.66	0.79	0.74	3	3	3
8	LeVs10SpnLg	3.26	3.08	3.17	0.63	0.74	0.69	3	3	3
9	LeVs10SpnTf	3.25	3.08	3.16	0.65	0.78	0.72	3	3	3
10	LeVs10SpnBe	3.28	3.05	3.16	0.64	0.78	0.72	3	3	3
11	LeVs5NspTf	3.36	2.96	3.16	0.68	0.75	0.74	3	3	3
12	LeVs5NspLg	3.28	2.91	3.09	0.71	0.75	0.75	3	3	3
13	LeVs5SpoBe	3.19	2.81	3	0.62	0.80	0.74	3	3	3
14	LeVs5SpoTf	3.16	2.8	2.98	0.63	0.74	0.70	3	3	3
15	LeVs5SpoLg	3.1	2.8	2.95	0.63	0.70	0.68	3	3	3
16	LeVs5SpnLg	3.01	2.74	2.88	0.58	0.71	0.66	3	3	3
17	LeVs5SpnTf	3.01	2.71	2.86	0.61	0.68	0.66	3	3	3
18	LeVs5SpnBe	3.01	2.63	2.82	0.63	0.75	0.72	3	3	3

In all cases, the combined summaries obtained higher average scores than each of the individual summaries, and at times the improvement was substantial. The lead summaries benefitted from the addition of the terms from the VSM summaries and the improvement averaged 0.28 over all artifacts and all six types of lead summaries. The highest improvement was in the case of the 10-terms summaries without identifier splitting, which had previously scored the highest among the individual summaries. In this case, the score of the new, combined summary indicates that developers agree highly with the new summary, suggesting that lead+VSM summaries are a good baseline for the automatic summarization of software artifacts.The lead+VSM summaries gained most over the lead summaries in the cases of classes and less in the case of methods (see Table III). This can be explained by the fact that the method lead summaries already contain parts of the header of a method and/or explanatory leading comments, which are all considered very relevant by developers. In the case of classes, on the other hand, the lead summaries contain the name of the class and/or leading comments and the first few attributes in the class, which are not always the most relevant for the class. Adding extra relevant terms by the addition of the VSM summaries increases the quality of the summaries.

Theoretically, the learning resulted from the first summarization session could lead in a difference in the judging and rating of summaries in the second summarization session and impact the results. However, the time passed between the session when they rated the individual summaries and the one when they rated the combined summaries was significant (almost four months), which was enough to mitigate the effect of learning. This was corroborated also by the fact that two of the developers reported having some difficulties understanding one class and one method, respectively, before the second summarization session, even though they had no problems understanding them before the first session. They eventually understood the class and method well, but they spent considerably more time understanding them in the second session than in the first one. This indicates that the learning effect, if any, was neutralized between the two sessions.

2) Methods vs. classes

Lead summaries received better scores for methods than for classes (see Table I). This can be explained by the fact that for the lead method summaries, the four developers agreed more on the type of terms that should be included in the summary. In the case of classes, however, besides the name of the class, which was considered important by all four developers, there were different opinions on what should be in the summary. D2 and D4 put emphasis on the name of class attributes, while D1 and D3 considered the name of methods to be more important. The name of the methods, though, were rarely included in the class lead summaries, as opposed to the attribute names, which led to the lower rating of these summaries by D1 and D3. For example, for the class AmazonService in ATunes, we found that D2 and D4 gave a score of 3 to all the lead summaries having 5 terms, which contained the name of the class and one or two of the names of attributes in the class. On the other hand, developers D1 and D3 both assigned a score of 2 to all these summaries.

The other summarizing techniques also performed better on methods than on classes, even though the differences between methods and classes were not as high as in the case of lead summaries, where this difference was 0.47 on average. In the case of VSM, the difference was 0.08 on

average, for LSI this difference was 0.17, and 0.34 for random summaries.

For the lead+VSM summaries we observed the same phenomenon: the summaries for methods received higher scores than the summaries for classes, for all combinations of lead and VSM summaries. This is not surprising, due to the higher scores for methods received by each of the techniques independently. We also noticed that the difference between the scores of the method summaries and the scores of the class summaries are higher for the union of the short summaries, i.e., 5-term summaries. This is because, in the case of methods, 5 terms are often enough for the lead summaries to include most of the header of a method, which contains relevant terms according to the developers. In the case of classes, however, the first 5 terms are often not enough to capture the most relevant information, hence the lower scores.

3) 5-term vs. 10-term summaries

By construction, the 10 terms summaries always contain the 5 terms summaries, followed by other 5 terms. When comparing two summaries obtained using the same technique and the same parameters, the 10 terms summaries are always ranked higher than the 5 terms summaries. Table IV shows an aggregated view of these results, presenting the average scores for all techniques across all artifacts for 5 and 10 terms, respectively. The last column shows the results of the union between lead and VSM summaries, when each of them contains 5 or 10 terms.

TABLE IV. AVERAGE SCORES FOR DIFFERENT NUMBERS OF TERMS

	Lead	VSM	LSI	Random	Lead+VSM
10 terms	3.07	2.30	1.93	1.76	3.24
5 terms	2.71	1.91	1.57	1.37	3.11

One reason for the fact that 10-term summaries get higher scores is that they contain more relevant terms than the 5-term ones. Since three of the four developers reported in the post-experiment questionnaire that they put emphasis on the relevant terms rather than the irrelevant ones when ranking summaries, the 10-term summaries get higher scores. This observation is supported also by the high correlation we found between the number of relevant terms contained in the summaries and the scores of they received (0.82 Spearman rank correlation).

According to these results, one could say that summaries should have as many terms as possible, since the scores of the summaries increase with the number of relevant terms, and the number of relevant terms increases with the total number of terms in the summary. However, the purpose of the summaries is to offer developers a short description of the software artifact, which would take less to read than the original artifact.

4) Splitting techniques

When analyzing the results for individual summaries, we observed that the versions containing only full identifiers obtained the highest score in 60% of the cases, followed by the summaries containing both the terms resulted after splitting the identifiers and the original identifiers, in 40% of the cases. The summaries containing only the terms by splitting the identifiers never had the highest score and in 80% of the cases they were assigned the lowest scores.

We found the same tendency for the lead+VSM summaries, where the version of the summaries having the full identifiers always received the highest scores (on average the score was 3.33), followed in all cases by the version containing both the full identifiers and the words resulted after splitting the identifiers (on average they got a score of 3.19). The summaries containing only the terms after splitting the identifiers were always the ones developers assigned the lowest score to, i.e., 3.01 on average.

These findings were confirmed also by the developers, which reported that they preferred the summaries containing the full identifiers. All developers reported that the summaries containing only the terms resulted after splitting the identifiers were last in their preferences. They explained their choice by the fact that it is harder to understand what the method or class is doing when the phrases appearing in identifiers are split, especially when these are phrases containing a verb and its direct object.

5) Weights

For both TR techniques we used three of the most popular weighting schemes used in text summarization, namely binary-entropy, tf-idf, and log (see [20] for the formulas and more information about each weight). In order to determine which weight is most appropriate for source code summarization, we computed the averages for each weighting scheme for methods, classes and over all the artifacts. The aggregated results for all summarization techniques using a particular weight are presented in Table V. The weighting scheme with the lowest scores for VSM across all data is *td-idf.* This weight obtained the lowest score for both 5-term summaries and 10-terms summaries, in the case of methods, classes, and overall. *Binary-entropy* was the weight which resulted in the highest ratings for VSM over the entire data, and for methods as well. However, in the case of classes, it was surpassed by *log.*

For LSI, the situation was the opposite. *Tf-idf* obtained the highest scores among the weights for classes and over all artifacts. However, for methods, *log* performed better.

One curious thing was that, even though *tf-idf* received the lowest scores for VSM summaries, the lead+VSM summaries got the highest scores when this weight was used. Sometimes, however, *binary-entropy* or *log* were on the same level as *tf-idf.*

TABLE V. THE AVERAGE SCORES FOR DIFFERENT WEIGHTS

Avg. scores	VSM	LSI	Lead+VSM
Tf-Idf	2.07	**1.81**	**3.19**
Log	2.12	1.78	3.16
Binary-entropy	**2.13**	1.67	3.18

Even though there are slight differences between the performances of TR summarization techniques using different weighting schemes, this difference proved to be not statistically significant. It is hard to draw any clear conclusion on this issue at this point. Future studies

involving larger and more varied corpora are most likely needed in order to reach a conclusion.

6) Agreement between developers

Subjectivity is unavoidable in this type of evaluations. It is desirable to estimate its effect on the results and it is common to measure the agreement of the evaluators in order to understand the results better. Light variations in rating between developers are normal and expectable, for example one developer agreeing highly with a summary (score 4) and a second one still agreeing, but less intensely (score 3). However, situations when a developer agrees or highly agrees with a summary, and another one disagrees, or highly disagrees with it can indicate misunderstandings of the class or method. In consequence, we focused on finding such cases and analyzing their causes.

The agreement measure we used for a pair of developers is defined as the number of times the two developers either both agreed with the summary to some degree (they assigned it a score of 3 or 4) or they both disagreed with it (they assigned it a score of 1 or 2). Note that if a developer gives a summary a score of 1 and a second developer gives the summary a score of 2, we consider the two developers to be in agreement, as they both disagreed, even though to a different degree. We found that the pairs of developers agreed on average in 76.8% of the cases. This is similar to the level of agreement found in studies involving the assessment of natural language summaries [21]. The agreement for each type of summary and number of terms is shown in Table VI.

For all summaries not involving the lead summaries, the 5-term summaries had a higher agreement than the 10-term summaries. Among these, the highest agreement was in the case of the random summaries, which had a 92.8% agreement for the 5-term summaries. These observations are explained by the fact that in the case of VSM, LSI, and random summaries, the 5-term summaries were often of poor quality (see Table IV), containing few relevant terms, thus making it easy for developers to assign them a low score and to agree on this. One could also say that since the 5-term summaries contain less terms, there is less to disagree about.

TABLE VI. AVERAGE AGREEMENT BETWEEN ALL PAIRS OF DEVELOPERS

Avg. agreement	5 terms	10 terms	All
Lead	63.8%	73.6%	68.7%
VSM	76.2%	61.8%	69%
LSI	86.6%	75.2%	80.9%
Random	92.8%	80.4%	86.6%
Lead+VSM	70.2%	81.6%	75.9%

The lead and lead+VSM summaries, on the other hand, had a higher agreement for the 10-term summaries. In this case the situation is exactly the opposite: the 10-term summaries were often very good, especially in the case of methods. It was easier for developers to agree on assigning those good scores, as opposed to the lead 5-terms summaries, which contained fewer relevant terms.

Overall, however, for the individual summaries developers agreed more on rating the bad summaries than the good ones. The random summaries got the highest agreement, followed by the LSI summaries, then VSM, and lead as last. This is the reverse order of the types of summaries according to their score. In fact, we found a strong negative correlation (-0.75) between the scores received by a summary and the agreement of developers in scoring it.

7) Relevant terms

Table VII presents the number and ratio of relevant terms (number of relevant terms/total number of terms in a summary) in the 5-term and 10-term summaries for the summaries with the highest scores, i.e., lead, VSM, and lead+VSM. As we can observe, the lead+VSM summaries contain always a higher number of relevant terms than the lead or the VSM summaries alone, even though the ratio of relevant terms decreases compared to the highest ratio in the two individual summaries. However, the lead+VSM summaries always received a higher rating from developers than any other summarization technique alone. This indicates that the number of relevant terms in a summary is more important than the ratio of relevant/irrelevant terms, i.e., relevant terms bare more weight than the irrelevant ones in rating a summary. This was confirmed by three of the developers, i.e., D2, D3, and D4 during the post-experiment questionnaire. D1 was the only developer which considered the number of irrelevant terms as an important factor in his ratings.

TABLE VII.THE NUMBER AND RATIO OF RELEVANT TERMS IN LEAD, VSM AND LEAD+VSM SUMMARIES

	Rel. terms	Methods			Classes			All		
		L	*V*	*LV*	*L*	*V*	*LV*	*L*	*V*	*LV*
5T	***No.***	4.70	4.07	7.58	3.82	4.02	7.14	4.19	4.04	7.36
	Ratio	**0.91**	**0.81**	**0.86**	**0.76**	**0.80**	**0.78**	**0.84**	**0.81**	**0.82**
10T	***No.***	7.95	7.29	12.40	7.18	7.05	12.32	7.57	7.17	12.36
	Ratio	**0.80**	**0.73**	**0.77**	**0.72**	**0.71**	**0.70**	**0.76**	**0.72**	**0.74**

From the same table we can notice that for methods the lead summaries always have a higher number and ratio of relevant terms than the VSM summaries. For classes, however, the lead summaries have a lower number of relevant terms for the 5-term summaries, and a comparable number of terms for the 10-terms summaries. This is because often few of the first 5 or 10 terms in a class are relevant terms for the class. For methods, however, the lead summaries often contain the header of the method, among other things, which contains terms considered relevant by the developers. Even though in the case of classes the VSM summaries have more or almost the same number of relevant terms as the lead summaries, the lead always received a higher rating. This indicates that not only the number of relevant terms is important, but also which these terms are. For example, the lead summaries contain often the names of classes, considered very important by all developers, which tend to give them high ratings.

F. Threats to validity

As with any case study, generalization of the results has to be done with care. Several factors influence our ability to produce analytical generalizations.

Due to the high time demand, we were not able to involve more developers at this stage. With other developers providing the ratings the results may be somewhat different. We plan to perform studies involving significantly more developers in our future work.

The developers did not agree in some cases about the ratings of the summaries. However, the level of disagreement was not higher than observed during the assessment of natural language text summaries [21]. We expect that similar agreement levels would be observed among any set of developers.

During the evaluation sessions, developers had to rate several summaries of the same artifacts. In consequence, the presence of a learning effect is possible. We did not try to measure it or mitigate it.

We selected only ten methods and ten classes from each of the systems. While we tried to vary their properties, they may not necessarily be the most representative of the two systems and even less so of other systems. More than that, the two systems have high quality, self explanatory identifiers. It is hard for us to estimate what the results would be for systems with poor identifier naming.

IV. Related work

The automatic summarization of natural language text has been widely investigated by researchers [5]. The summarization of software artifacts, however, is only at the beginning. Rastkar et al. [22] studied recently the summarization of bug reports and used an approach based on machine learning to create summaries of bug discussions. In [23], the same authors proposed a process for summarizing software concerns. The TR based approaches for source code summarization, first introduced in [24], which we evaluate in this study, focus on summarizing whole software artifacts, with the purpose of aiding developers in comprehension tasks. In [23], however, the goal is to summarize concerns, which can be spread across many artifacts, and can occupy only parts of each artifact. TR based summarization approaches are more lightweight than the ones adopted by Rastkar et al., as they are mostly extractive techniques and do not require the use of an ontology, as in [23], nor training data, as in [22].

A form of structural summarization of source code has also been proposed in [25], which presented two techniques, i.e., the software reflection model and the lexical source model extraction for a lightweight summarization of software. These two techniques are complementary to the approaches we investigated and we envision combining them in the near future.

Other related work includes [26], where TR techniques are used to cluster source code and relevant terms from each cluster are extracted to form labels. A similar approach is used in [27], where TR is also used to extract the most relevant set of terms to a group of methods returned as results to a search. These terms are treated as attributes used to cluster the methods. In each case, the labels and attributes can be considered as (partial) summaries.

Another related research thread is on source code tagging and annotations [28]. These mechanisms could support developers to create and represent manual summaries of the code (in addition to comments).

V. Conclusions and future work

We investigated the use of automatic text summarization techniques to generate source code summaries. We found that a combination between techniques making use of the position of terms in software and TR techniques capture the meaning of methods and classes better than any other of the studied approaches. Developers generally agree with the summaries produced using this combination.

When analyzing the parameters that impact the production of source code summaries, we found that the weights used for the TR techniques do not impact the produced summaries considerably. On the other hand, longer summaries seem to be preferred by developers over short ones. However, the summaries have to remain short, as a developer should be able to read them faster than the source code. Developers also preferred the summaries containing the full identifiers, as opposed to the ones where the identifiers are split.

The results we obtained represent a very good starting point for the summarization of source code entities and our future work will focus on developing more complex techniques which will also include structural information from the source code. We also plan to investigate multi-document approaches for the summarization of source code packages and classes. Last but not least, 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.

Acknowledgments

We are grateful to Fabio Navarrete for his help with the evaluation. This work was supported in part by grants from the US National Science Foundation: CCF-1017263, CCF-0845706, and CCF-0820133.

References

[1] A. Lakhotia, "Understanding Someone Else's Code: An Analsis of Experience," *The Journal of Systems and Software,* vol. 23, pp. 269-275, 1993.

[2] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in *28th IEEE International Conference on Software Engineering*, 2006.

[3] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks," *IEEE Transactions on Software Engineering,* vol. 32, pp. 971-987, 2006.

[4] J. Starke, C. Luce, and J. Sillito, "Searching and skimming: An exploratory study," in *25th IEEE International Conference on Software Maintenance*, Edmonton, Alberta, Canada, 2009, pp. 157-166.

[5] K. Sparck-Jones, "Automatic summarising: The state of the art," *Information Processing and Management: An International Journal,* vol. 43, pp. 1449-1481, 2007.

[6] Y. Gong and X. Liu, "Generic text summarization using relevance measure and latent semantic analysis," in *24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, Louisiana, United States, 2001, pp. 19-25.

[7] J. Steinberger and K. Ježek, "Update Summarization Based on Latent Semantic Analysis," in *Text, Speech and Dialogue*, ed: Springer Berlin / Heidelberg, 2009.

[8] G. Salton, *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*: Addison-Wesley, 1989.

[9] G. Salton, A. Wong, and C. S. Yang, "A Vector Space Model for Automatic Indexing," *Communications of the ACM,* vol. 18, pp. 613-620, 1975.

[10] T. K. Landauer, P. W. Foltz, and D. Laham, "An Introduction to Latent Semantic Analysis," *Discourse Processes,* vol. 25, pp. 259-284, 1998.

[11] K. Kireyev, "Using Latent Semantic Analysis for Extractive Summarization," in *Proceedings of Text Analysis Conference*, 2008.

[12] X. Liu, J. J. Webster, and C. Kit, "An Extractive Text Summarizer Based on Significant Words," in *Computer Processing of Oriental Languages. Language Technology for the Knowledge-based Economy*, ed: Springer Berlin / Heidelberg, 2009, pp. 168-178.

[13] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia, "Information Retrieval Models for Recovering Traceability Links between Code and Documentation," in *IEEE International Conference on Software Maintenance*, San Jose, CA, 2000, pp. 40-51.

[14] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the Use of Relevance Feedback in IR-Based Concept Location," in *IEEE International Conference on Software Maintenance*, 2009, pp. 351-360.

[15] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic, "An Information Retrieval Approach to Concept Location in Source Code," in *11th IEEE Working Conference on Reverse Engineering*, 2004, pp. 214-223.

[16] A. Marcus and J. I. Maletic, "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing," in *25th IEEE/ACM International Conference on Software Engineering*, Portland, OR, 2003, pp. 125-137.

[17] G. Salton, A. Singhal, M. Mitra, and C. Buckley, "Automatic text structuring and summarization," *Information Processing and Management: An International Journal,* vol. 33, pp. 193-207, 1997.

[18] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *Psychological Review,* vol. 63, pp. 81-97, 1956.

[19] R. Likert, "A Technique for the Measurement of Attitudes," *Archives of Psychology,* vol. 140, pp. 1-55, 1932.

[20] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management: An International Journal,* vol. 24, pp. 513-523, 1988.

[21] I. Mani, "Summarization Evaluation: An Overview," in *NAACL Workshop on Automatic Summarization*, 2001.

[22] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing Software Artifacts: A Case Study of Bug Reports," in *International Conference on Software Engineering*, Cape Town, South Africa, 2010, pp. 505-514.

[23] S. Rastkar, "Summarizing software concerns," in *International Conference on Software Engineering, Student research competition* Cape Town, South Africa, 2010, pp. 527-528.

[24] S. Haiduc, J. Aponte, and A. Marcus, "Supporting Program Comprehension with Source Code Summarization," in *32nd ACM/IEEE International Conference on Software Engineering - NIER track*, Capetown, South Africa, 2010, pp. 223-226.

[25] G. Murphy, "Lightweight Structural Summarization as an Aid to Software Evolution," PhD Thesis, University of Washington, 1996.

[26] A. Kuhn, S. Ducasse, and T. Girba, "Semantic Clustering: Identifying Topics in Source Code," *Information and Software Technology,* vol. 49, pp. 230-243, 2007.

[27] D. Poshyvanyk and A. Marcus, "Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code," in *15th IEEE International Conference on Program Comprehension*, 2007, pp. 37-46.

[28] M. A. Storey, L. T. Cheng, I. Bull, and P. Rigby, "Shared Waypoints and Social Tagging to Support Collaboration in Software Development," in *Computer Supported Collaborative Work*, 2006.