Empir Software Eng (2016) 21:17–42
DOI 10.1007/s10664-014-9344-6

An empirical study of the textual similarity between source code and source code summaries

Paul W. McBurney · Collin McMillan

Published online: 9 November 2014

Abstract Source code documentation often contains summaries of source code written by authors. Recently, automatic source code summarization tools have emerged that generate summaries without requiring author intervention. These summaries are designed for readers to be able to understand the high-level concepts of the source code. Unfortunately, there is no agreed upon understanding of what makes up a "good summary." This paper presents an empirical study examining summaries of source code written by authors, readers, and automatic source code summarization tools. This empirical study examines the textual similarity between source code and summaries of source code using Short Text Semantic Similarity metrics. We found that readers use source code in their summaries more than authors do. Additionally, this study finds that accuracy of a human written summary can be estimated by the textual similarity of that summary to the source code.

Keywords Source code summarization · Documentation · Textual similarity · Automatic documentation generation

1 Introduction

Programmers rely on good documentation in order to effectively understand source code (Forward and Lethbridge 2002). Documentation often consists of small *summaries* of source code. A "summary" is a brief description of the functionality and purpose of a section of source code, such as method summaries in JavaDocs (Kramer 1999). Summaries help programmers focus on small sections of code relevant to their efforts without needing to understand an entire system (Lakhotia 1993; Roehm et al. 2012; Singer et al. 1997). While

Communicated by: Massimiliano Di Penta

P. W. McBurney · C. McMillan (✉)
Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA
e-mail: cmc@nd.edu

P. W. McBurney
e-mail: pmcburne@nd.edu

popular, summaries are expensive to write (de Souza et al. 2005; Kajko-Mattsson 2005), are often incomplete (Lethbridge et al. 2003), and become outdated as code changes (Fluri et al. 2007; Ibrahim et al. 2012).

These difficulties in creating summaries have motivated researchers to design several automated summarization tools as alternatives to writing summaries manually (Burden and Heldal 2011; Mani et al. 2012; McBurney and McMillan 2014; Moreno et al. 2013; Sridhara et al. 2010, 2011a, b). These automated approaches follow one of two strategies. One approach presented by Haiduc et al. returns a list of the most important keywords from the method using *term frequency/inverse document frequency* (*tf/idf*) (Haiduc et al. 2010). A different approach is presented by Sridhara et al. (2010). Sridhara et al.'s approach selects statements from a Java method. Then, it creates natural language sentences from the selected statements by placing keywords from the statements into predefined templates (Sridhara et al. 2010). A related approach is presented in our own prior work (McBurney and McMillan 2014).

A key similarity of these approaches is that they produce summaries that influence the *reader's* understanding of the source code without input from the source code's *author*. *Authors* are individuals who wrote the source code. Meanwhile, *readers* are non-authors seeking to understand source code by examination. There is a mismatch between authors and readers. Authors translate the high-level concepts into low-level implementation, while readers must deduce the concepts and behaviors from the low-level details. The readers struggle to understand the author's intent (Biggersta et al. 1993), and inevitably make mistakes in their understanding. At the same time, authors who write documentation of their source code must choose which key concepts and details to communicate to readers via documentation. The concepts and details described in documentation are the ones that authors believe readers would need to know.

This mismatch between authors and readers is critical for source code summarization tools. The reason is that source code summarization tools generate documentation of code from the code itself—a process more similar to the reader's comprehension than the author's writing of documentation. Source code summarization tools are limited to the information in the code in the same way as readers. But, summarization tools are expected to produce documentation with the same information as documentation from human authors.

Unfortunately, current software engineering literature provides little guidance to designers of source code summarization tools. There is no consensus on what information from the source code that the tools should target. One frequent assumption is that summaries should be *textually similar* to the source code, meaning that the summaries should contain the same keywords as the source code (Khamis et al. 2010; Steidl et al. 2013). However, this assumption has not been empirically validated independent of tools that are based on the assumption. At the same time, it is not known whether readers and authors use the same keywords to describe the same concepts, nor even if readers use the same keywords as other readers. These issues are important because they affect researchers' understanding of what keywords should be used by automated summarization tools. At present, designers of these tools must "guess and check", in a time-consuming process of building multiple tools and selecting the best-performing configurations through expensive human-driven case studies (Haiduc et al. 2010; McBurney and McMillan 2014; Sridhara et al. 2010).

In this paper, we address this problem in an empirical study comparing summaries written by authors of source code to summaries written by readers of source code. We identify the similarity between source code and summaries written by the authors, and compare the similarity to the similarity between source code and summaries written by other readers.

Then, we use our analysis to draw guidelines for designers of source code summarization tools.

We compute similarity using Short Text Semantic Similarity (STSS) algorithms including overlap percentage, STASIS (Li et al. 2006), and Lightweight Semantic Similarity (Croft et al. 2013). We also examine the relationship between a summary's similarity to source code, and the reader perceived accuracy of the Java method summary. Each of these metrics is described in detail in Section 3.6. Our contributions in this paper are as follows:

- An empirical study comparing source code of Java methods to author and reader summaries of those methods. This study found that a method's source code is more similar to reader summaries than author summaries (See Section 4.1).
- An empirical study comparing reader summaries of Java methods to source code and to other reader summaries of the same Java methods. This study found mixed results (See Section 4.2).
- An empirical study examining the relationship between accuracy of author summaries of Java methods and the similarity of the summaries to the method's source code. Accuracy is based on data collected from a previous case study (McBurney and McMillan 2014), where readers were asked how accurately given summaries explained source code. This study found that high similarity between author summary and source code is correlated with accuracy (See Section 4.3).
- An empirical study examining the relationship between accuracy of automatically generated summaries and the similarity of those summaries to source code. This study found that the similarity to a method's source code of summaries generated using Sridhara et al.'s approach (Sridhara et al. 2010) is weakly positively correlated with reader perceived accuracy. However, the similarity of a method's source code to summaries generated using McBurney et al.'s approach (McBurney and McMillan 2014) is not correlated with accuracy (See Section 4.4).
- To ensure reproduceability, our results are publicly available through an online appendix.[1]

2 The Problem

We address the following gap in software engineering literature: there is no clear understanding of how readers understand source code or author summaries. Consider the diagram in Fig. 1. The areas labelled "Author" and "Reader" represent the keywords of the summaries written by authors and readers respectively, while the "Source Code" section represents the keywords in the source code being summarized. In this diagram, we are most interested in the areas of overlap between the separate groups. The overlapping areas represent the use of shared keywords. A large overlap between two entities would indicate those entities being similar.

For example, the overlap between Author and Source Code, labelled AS contains the keywords *authors* used in author summaries that come from the source code they wrote. AS represents keywords from the source code the author feels necessary to use in order to explain the source code to readers. A large AS would mean the author is using more keywords from source code.

[1] http://www.nd.edu/~pmcburne/sumalyze

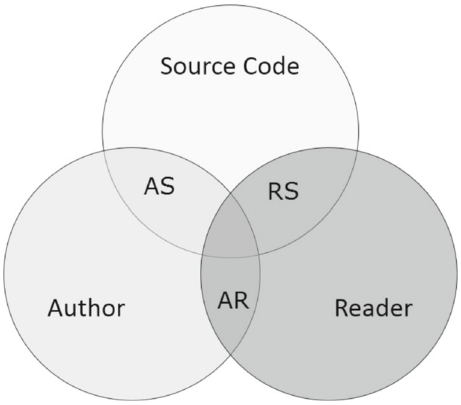


Fig. 1 Diagram illustrating keyword usage. Each *circle* represents a set of keywords present. "Source code" represents the keywords from the source code. "Author" and "Reader" represent the keywords in summaries written by the author and readers, respectively. Keywords from source code used in author summaries are represented by "AS". Keywords from source code used in reader summaries are represented by "RS". Keywords shared by author summaries and reader summaries are represented by "AR"

Another overlap area is the overlap between Readers and Source Code. Labeled RS, this section contains keywords that readers select from the source code to use in their summaries. The RS section represents which keywords from the source code readers feel are necessary to summarizing the source code as the reader understands it. A large RS would imply that readers use source code keywords frequently in their summaries.

The final overlap area in this diagram, AR, is the overlap between Authors and Readers. This overlap represents keyword selections shared between author-written summaries and reader-written summaries. A large AR in keyword selection between these two fields would imply a similar understanding from the source code, even if keywords in AR are not present in Source Code. A small AR would imply that there is a disconnect between how authors describe source code and how readers interpret source code.

So far, we have defined the size of the overlap between sections of the diagram in Fig. 1 as the number of keywords shared between each section. Another way to define the size of an overlap is by examining *semantic* similarity between source code, author summaries, and reader summaries. Semantic similarity is the similarity in the meaning of the keywords (Croft et al. 2013; Li et al. 2003). Using semantic similarity is useful because source code is often written with large identifiers that can decrease source code readability (Liblit et al. 2006). By this definition, a large AR section would mean that author and reader summaries are semantically more similar. That is, the author and reader summaries share similar meaning, even if the explicit similarity of keyword usage is different. A small AR would mean that author and reader summaries are not semantically similar.

By analyzing the overlap between each of the three groups in Fig. 1, we will be able to better understand the relationships that exist between source code and summaries written by authors and readers.

3 Empirical Study Design

In this section, we will describe our research questions for this study, and justify their use in solving the overall problem of keyword selection in source code summarization. Additionally, we will lay out our methodology for answering these questions, including brief explanations of the metrics we use.

3.1 Research Questions

This study seeks to identify what keywords are selected for source code summarization by both authors and readers of the source code. In doing so, we aim to examine the relationship between the three keyword sources as illustrated in Fig. 1. Therefore, we pose the following four Research Questions (*RQ*s):

RQ_1 Which is more similar to source code: author-written summaries or reader-written summaries?

RQ_2 Which is more similar to reader-written summaries: the source code being summarized, or the summaries of other readers?

RQ_3 Is there a correlation between the similarity of author-written summaries to source code and the reader defined accuracy of author-written summaries?

RQ_4 Is there a correlation between the similarity of automatically-generated source code summaries to the source code and the reader defined accuracy of automatically generated summaries?

The rationale behind RQ_1 is to determine whether authors or readers of source code more commonly use keywords from the source code when writing summaries. If readers are relying more on keywords in source code than the authors, it could imply that the author summary does not effectively communicate the source code to the reader. We do this so we can test the assumption that summaries should use the same keywords as the source code they summarize in order to most effectively communicate to source code readers (Khamis et al. 2010; Steidl et al. 2013). The rationale RQ_2 is to examine how multiple readers interpret source code. If reader summaries are more similar to each other than to source code, it could imply that readers are able to interpret a shared high level understanding from source code. However, if reader summaries are more similar to source code than each other, or are equally similar, it could imply that source code can often fail to communicate high level ideas. This could mean readers are simply translating the source code into natural language, rather than summarize the source code at a high level. This would indicate a larger problem in what readers think is important compared to what authors communicate in their documentation. RQ_3 explores the idea that an accurate author summary should have a high similarity to the source code. We define an accurate summary as one that correctly represents the actions of the source code. This definition is shared with the definition used by Sridhara et al. (2010) and by our previous work (McBurney and McMillan 2014). A positive correlation between similarity and accuracy would indicate that readers believe an accurate summary should reflect the source code. RQ_4 is similar to RQ_3. However, RQ_4 will specifically examine if a correlation between a summary's perceived accuracy by readers and that summary's similarity to the source code for automatically generated source code summaries. In this study, we examine two automatically generated techniques: Sridhara (Sridhara et al. 2010) and Sumslice (McBurney and McMillan 2014).

Table 1 The Java programs used in our empirical study

Program	Methods	KLOC	Java files
NanoXML	318	5.0	28
Siena	695	44	211
JTopas	613	9.3	64
Jajuk	5921	70	544
JEdit	7161	117	555
JHotdraw	5263	31	466

3.2 Methodology

The methodology we use to answer RQ_1 and RQ_2 is as follows: First, we collect author and reader summaries of six different Java applications (Table 1) (Section 3.3). Then, we preprocess those summaries using standard techniques such as stop word filtering (Section 3.4). Next, we use three different STSS metrics to compute the similarity among the author summaries, reader summaries, and source code (Section 3.6). Finally, we use statistical hypothesis tests to determine the statistical significance of the differences in the STSS metrics we calculated for the authors to source code, authors to readers, readers to source code, and readers to other readers for the same code (Section 3.8).

For RQ_3 and RQ_4, we collected data on reader perceived quality for author summaries and for summaries created by two different automatic source code summarization tools (Section 3.3). Then, we use the Pearson correlation metric to determine the correlation between this quality data and the STSS-calculated similarity to source code. Details of our analysis for all research questions are in the following sections, and throughout Section 4.

3.3 Data Collection

This empirical study uses data collected from a two case studies we conducted.[2] The first was a pilot study where we showed programmers summaries of source code and asked the participants to answer multiple-choice quality questions about those summaries. These questions are listed in Table 2. These questions were adapted from other automated source code summarization studies, including Haiduc et al. (2010) and Sridhara et al. (2010). The summaries came from three sources: *author* summaries via JavaDocs, a prototype automatic summarization approach, or a concatenation of the two. In addition to answering the multiple choice questions based on the summaries and source code, we asked the participants to write a brief summary for the code they read in their own words. The participant was able to click a link to open a source code browser which would initially focus on the method that was summarized. The participant was able to browse the entirety of the source code[3] in the source code viewer, but was asked to limit their summary to just the method in question. This pilot study was taken by 13 participants. Six were graduated students and

[2]Note that while we use data collected during these previous studies, this paper bears no other similarity to the previous studies, in that we study different research questions for a different purpose.

[3]The source code the participant viewed was stripped of all inline comments and Javadocs documentation to avoid biasing the participant.

Table 2 The questions we ask during the user studies

Q_1	Independent of other factors, I feel that the summary is accurate.
Q_2	The summary is missing important information, and that can hinder the understanding of the method.
Q_3	The summary contains a lot of unnecessary information.
Q_4	The summary contains information that helps me understand what the method does (e.g., the internals of the method).
Q_5	The summary contains information that helps me understand why the method exists in the project (e.g., the consequences of altering or removing the method).
Q_6	The summary contains information that helps me understand how to use the method.
Q_7	Write a summary for what the method does in your own words.

Questions Q1-Q6 were answerable by participants as “Strongly Agree”, “Agree”, “Disagree”, and “Strongly Disagree.” Q7 was an open answer question

three were undergraduate students at the University of Notre Dame Computer Science and Engineering Department. Four participants were affiliated with other organizations not listed due to privacy policies. These four participants were included professionals and graduate students.

The second study was the focus of our evaluation of two automated source code summarization tools (McBurney and McMillan 2014). This study was identical to the pilot study: participants were asked to answer six multiple-choice questions listed in Table 2 and briefly summarize the code they read. However, in the second study participants compared two automatic summary generation approaches: one by McBurney et al’s Sumslice approach (McBurney and McMillan 2014) and one by Sridhara et al.’s approach (Sridhara et al. 2010). Author summaries were not included as part of this study. Our second study surveyed 12 participants in total. Nine of the participants were graduate students from the University of Notre Dame Computer Science and Engineering Departments. The remaining three participants were professional programmers affilated with other organizations. These organizations are not listed due to our privacy policy. Two participants in our second study also participated in our pilot study. Thus, in total, both our studies combine to have 23 total unique participants.

Both studies used a cross-validation design. Each participant read an automatically generated summary and source and answered the questions in Table 2. The answer to Q7 is used in this empirical study as the reader summaries. Each participant saw six pages of methods to score and summarize. Each page consisted of four randomly selected methods from a single randomly selected project. For each method, the participant was asked to answer all questions in Table 2. Two of the pages would use our approach to generate a summary, two of the pages used the competing approach (either author summaries or state-of-the-art generated summaries), and the remaining two pages were summarized by concatenating Sumslice’s summaries with the competing summaries. We use the automated summaries generated by Sumslice and Sridhara to answer RQ_4. However, we do not use the concatenation of Sumslice and Sridhara, as our results in previous work imply that it is overly verbose (McBurney and McMillan 2014), making it a poor candidate for short text similarity metrics.

3.4 Pre-processing

All author summaries, reader summaries, automatically generated summaries, and source code are preprocessed to ensure consistency. We pre-process the summaries and source code in two steps. First, all special characters are removed. This includes operators in the source code, and punctuation in the summaries. All special characters are replaced by whitespace. Next, we *lexicalize* all the tokens in each input text. Lexicalization in source code often involves traslating an identifier into natural language text, such as interpreting `readFile` as "read file." To do this, we split all tokens on camelcase into multiple individual tokens separated by whitespace. This has the benefit of isolating individual words within tokens, which, as we explain in Section 3.5, improves the semantic analysis we perform later. For example, `readFile` does not have a natural language definition. However, "read" and "file" have natural language definitions.

3.5 WordNet

Two of our three metrics rely on WordNet (Miller 1995), a natural language knowledge base. This knowledge base is formatted as a hierarchical structure that groups words into sets of synonyms, called *synsets*. Words within the same *synset* have similar *semantics*, or meaning. WordNet is used to determine how similar two different words are with respect to semantics.

To illustrate the idea of a knowledge base, we provide a brief example not related to software engineering. Consider a hypothetical hierarchical knowledge base shown in Fig. 2. If we wanted to find the similarity between the words "evergreen" and "deciduous", we would first find the distance, which is 2, and the depth of the shared ancestor node "trees", which is 3. The depth is relevant, because the lower the depth, the less similar words at equal length are. "plants" has the same distance to "rocks" as "evergreen" had to "deciduous." However, because the shared ancestor "trees" is deeper, it is a more specific identifier than "things", the shared ancestor of "rocks" and "plants". Thus, the larger the depth of the

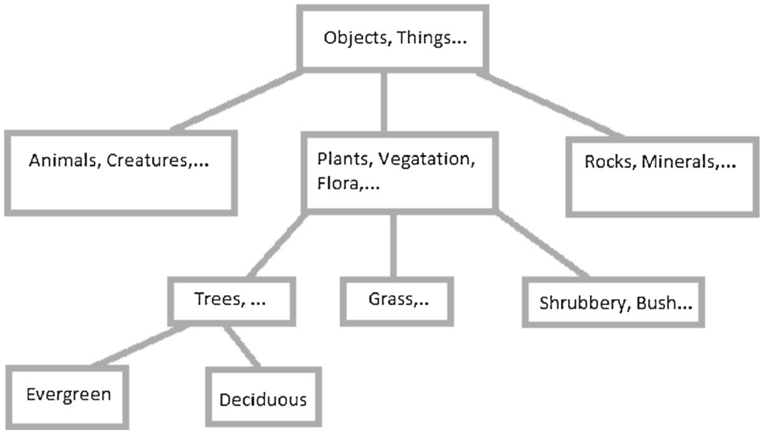


Fig. 2 A hypothetical knowledge base. The similarity between two words, such as "evergreen" and "deciduous" is a function of the distance between the nodes and the depth of the lowest shared ancestor. The distance from "evergreen" and "deciduous" is 2, going through "trees", and the depth of the common ancestor "trees" is 2 where the root of the knowledge base tree is 0

shared ancestor, the more similar the two words are. Further, since like objects are grouped together, the larger the distance between two nodes, the less related they are.

3.6 Metrics

In this section, we will descibe in detail the three metrics of calculating Short Text Semantic Similiarity (STSS) we use in this empirical study.

3.6.1 Overlap

The overlap metric is a percentage reflecting the number of keywords present in one body of text that are present in a second body of text. This metric gives us the intersection between summaries and source code. There are two different overlap metrics, overlap with *stop words*, and overlap without stop words. Stop words are common words, such as “a” and “an” which add little semantic information. For overlap without stop words, we remove the same stop words removed by LSS and STASIS for consistency.

3.6.2 STASIS

Li et al.’s short text similarity approach STASIS (Li et al. 2006) is a metric that considers *word semantic similarity*, *sentence semantic similarity*, and *word order similarity*. By using this semantic similarity, this metric allows us to examine the meaning of keywords selected, rather than just relying on specific keyword selection. Sentence semantic similarity, and word order similarity are calculated using word semantic similarity. Sentence semantic similarity, and word order similarity are then combined to calculate the overall similarity. The use of word order similarity sets STASIS apart from most other STSS metrics (Croft et al. 2013). Complete details of this metric are in related work (Li et al. 2006), though we briefly describe the three types of similarity here for clarity and reproduceability.

Word Semantic Similarity is the measure of how similar two individual words are. Word semantic similarity is used to calculate sentence semantic similarity and word order similarity. It requires the use of a hierarchical knowledge base. Li et al. used WordNet (Miller 1995) in their approach. Thus, our approach will also use WordNet. The word semantic similarity is calculated by comparing the relative word positions within a given knowledge base’s hierarchical structure. Specifically, we focus on the distance between two terms in the hierarchy, and the depth of the shared ancestor between the two words within the hierarchy.

It is assumed that the smaller the distance between two words in a knowledge base, the more similar they are. Additionally, the larger the depth of the shared ancestor between two words, the more specific and, by extension, the more similar the words are. Li et al. propose the following formula to generate the numeric similarity between two words:

$$s(w_1, w_2) = e^{-\alpha l} \cdot \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}}$$

In Formula 3.6.2, l represents the distance between two terms in the hierarchical structure of WordNet, and h represents the depth of the shared ancestor. α and β are constants whose value will vary between different hierarchical knowledge bases. For WordNet, Li et al. found the optimal constants to be $\alpha = 0.2$ and $\beta = 0.45$ (Li et al. 2003).

Sentence semantic similarity is the measure of how similar two sentences are. This metric calculates a similarity measure between two different sets of words. It is the first component of the overall similarity calculation.

First, T_1 and T_2 are combined with a union operation into one join word set T. Then, similarity vectors s_1 and s_2 are made for T_1 and T_2 respectively. For each word w_i in T, if that word appears in T_j, then the i-th element of s_j is set to 1. If the word does not appear in T_j, then the *word semantic similarity* is found between w_i and every word in T_j, and the maximum similarity is found. If the maximum similarity is above a certain threshold then the i-th element of s_j is set to that maximum similarity. Otherwise, the i-th element of s_j is set to zero. This threshold exists to prevent noisy data.

This generates two similarity vectors, s_1 and s_2. Each vector represents the similarity of one input set of words to the union of both input sets T. The sentence semantic similarity is the cosine similarity between s_1 and s_2.

$$S_s = \frac{s_1 \cdot s_2}{||s_1|| \cdot ||s_2||}$$

Word order similarity is the measure of how similar the word order is between two text bodies. It is the second component in the overall similarity calculation. This process is similar to sentence semantic similarity. T_1 and T_2 are again combined with a union operation into a joint word set T. Then, we create word order vectors r_1 and r_2 for T_1 and T_2 respectively. For each word w_i in T, if w_i exists in T_j, then the corresponding i-th element in r_j is set to the position of w_i in T_j. If w_i does not exist, we find the most similar word in T_j using word semantic similarity. If the similarity is above a certain threshold, the the i-th value of r_j is set to the position of the most similar word in T_j. Otherwise, the i-th value of r_j is set to zero. Finally, to calculate the word order similarity, the following formula is used.

$$S_r = 1 - \frac{||r_1 - r_2||}{||r_1 + r_2||}$$

Overall similarity is the measure of how similar two text items are. This metric combines sentence semantic similarity and word order similarity to calculate the overall similarity metric of two text items.

$$S(T_1, T_2) = \delta S_s + (1 - \delta) S_r$$

In formula 3.6.2, δ is a constant that weights the relative importance between sentence semantic similarity and word order. Li et al. used $\delta = .85$, to ensure that sentence semantic similarity S_s was given more weight than word order similarity S_r. In our study, we examine STASIS *with* word order and *without* word order. To calculate STASIS with word order, we use $\delta = .85$ to match Li et al. To calculate STASIS without word order, we use $\delta = 1$. This means that word order is nullified, as it is multiplied by 0.

3.6.3 LSS

Croft et al. (2013) proposed an STSS metric called Lightweight Semantic Similarity (LSS). LSS addressed a lack of sentence structure within some short text items; the paper specifically addresses titles of photographs. Because source code does not have a sentence structure, this metric would appear an ideal approach. LSS has two stages. The first is a text pre-processing stage, and the second is calculating the textual similarity between the two text items.

Initially, LSS pre-processes the two input text items to be compared. Pre-processing for LSS is very similar to the preprocessing described in Section 3.4. The key difference is that the WordNet synsets of each word are identified as an additional step. Using synsets instead or the raw input word allows the keywords to be normalized, identifying the word's semantic purpose without respect to verb tense or singular/plural nouns. In cases where a word has no synsets, it is maintained in a set for character-based String comparison. This is particularly useful with source code, which may contain unique names that do not have easily interpreted natural language analogues (such as Jajuk).

After pre-processing, each input text item has an associated word set, A and B. The union A and B, C contains all terms within A and B. Note that if two words being compared are in the same synset, their similarity is equal to 1. Then a similarity matrix is constructed by finding the similarity between all elements in C and all other elements in C. A and B are then used to create a weighted term vector using this similarity matrix. Taking A as an example, $\mathbf{A}$ is weighted vector that has the same number of elements as C. Each element i in $\mathbf{A}$ is set to the sum of all elements in A compared to C_i, using the similarity matrix. The overall similarity is then determined by taking the cosine similarity of those two weighted vectors.

$$S_s = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \cdot ||\mathbf{B}||}$$

3.7 Automatic Summary Generation

This section will briefly explain the two automatic summary generation tools we use in this study to answer RQ_4. Each of these approaches were used in our previous work (McBurney and McMillan 2014), and the reader scores come from that case study.

3.7.1 Sridhara

Sridhara et al. developed an automatic source code summarization tool to describe the internal workings of a Java method (Sridhara et al. 2010). Sridhara's approach relies on generating a Software Word Usage Model (SWUM). SWUM is used to identify, within the method's signature and body, identifiers within the code. These identifiers are classified by part of speech, such as noun, verb, etc. SWUM creates a natural language representation of the Java method. Sridhara's approach constructs summaries using natural language *templates*. These *templates* are natural language sentences describing a particular type of Java statement. The template is "filled in" with keywords generated by the SWUM Model selected from the summarized statement.

This approach then selects *s_units*, or statements, from the Java method. These *s_units* are selected based upon their interpreted importance across multiple criteria. For each selected *s_units*, an associated *template* is selected and a natural language summary constructed using information from SWUM. The set of these summaries, each summarizing a different *s_unit* is then combined and smoothed to create the method summary.

3.7.2 Sumslice

Sumslice, our own previous work (McBurney and McMillan 2014), is an automatic source code summarization tool that primarily focuses on the interaction between function, rather than the internal source code that Sridhara's approach focused on. Sumslice also uses a Software Word Usage Model (SWUM) to identify keywords and their parts of speech in the message signature, but does not use a SWUM for the method body. The SWUM of the method signature is used to create a natural language summary for the *purpose* of the method, that is, what action the method performs. How this action is performed is not summarized by our approach. Instead, we choose to use information about the external interactions of a method by examining a *call graph*. A call graph is a graph created where the nodes, or vertices, are the Java methods, and the edges connecting vertices are directed edges showing which functions are called by which. By constructing this graph, we can learn, given a particular method, which other methods are called, and which other methods call this particular method. Using this information, we create a natural language summary explaining to the reader the method's purpose, what methods it is called by, and we can directly cite a source code example showing how the method is used.

3.8 Statistical Tests

The results of each research question form a set of scores, one for each metric (see Section 3.6). In RQ_1 and RQ_2 we use the Mann-Whitney statistical test (Mann and Whitney 1947) to determine significance in results. We selected Mann-Whitney because we cannot guarantee our results will be distributed normally, so a non-parametric test must be used. Additionally, our results are unpaired due to the random nature of which readers received which methods to summarize. Thus, there is no one-to-one relationship between reader summaries and source code, nor is there a one-to-one relationship between reader summaries and other reader summaries. We use Pearson correlation in RQ_3 and RQ_4 to determine if the similarity metrics of the source code to a summary is correlated with reader perceived accuracy of the summary.

3.9 Threats to Validity

The key source of threats to validity in our study is the data we collect. Because this study used human experts, it is susceptible to the same threats to validity as other human driven case studies. Participants may have varying programming experience, or may have experienced fatigue during the study, among other uncontrollable factors. We mitigated the threats within the case study by seeking diversity among our 23 participants, including students and professionals. However, we cannot say with certainty that different participants in a similar study would not result in different findings.

Another source of a threat to validity is that source code often relies on abbreviations and shorthand versions of longer words. While WordNet may contain common abbreviations and acronyms, such as "radar" and "ZIP", as in ZIP Code, source code abbreviations may often be highly specified to a technical domain, or may be specific to the source code itself. It is likely, given a large source code project, that several abbreviations will simply not be found in WordNet. This represents a threat to validity when examining semantic similarity. This is somewhat mitigated in each of the three approaches we examine, in that all three approaches also consider string matching, even when a word cannot be found in WordNet. However, we cannot guarantee a similar study with source code and documentation that relies heavily on acronyms would produce the same results.

There is a potential bias from the fact that the programmers saw summaries from one of our three main sources (author, Sridhara, and Sumslice). This is mitigated by our cross-validation design, ensuring participants saw different summary sources alongside source code. Therefore, biases resulting from the summary source a participant was given would be countered by other participants who received a different summary source. However, we cannot guarantee that a study conducted without the participants being given a summary to score would produce the same results.

Our results for RQ_4 are largely dependent on the two automatic summarization tools we use. Different automatic summarization tools are very likely to produce different results. Given this, our results will strictly apply to the summarization tools use. However, the process of determining the correlation between reader perceived accuracy and a summary's similarity to source code would be applicable to other automatic summarization tools not used here.

Finally, as with any case study that uses existing source code as a corpus, the limitation of Java programs selected in this study (shown in Table 1) are a source of threats to validity. It is possible that our results would be unique among the selected six projects, and would not be repeated on a random selection of other projects. Further, the methods with the projects selected could present a threat to validity for the same reasons. This threat was mitigated in the case study by selecting 20 methods randomly from each of the six programs, which combined had over 19,000 Java methods. The random selection mitigates the introduction of bias on behalf of the authors of the study.

3.10 Reproducibility

For the purposes of reproduceability and independent study, we have made all evaluation data and tools for this study available via an online appendix.[4]

4 Empirical Results

This section presents the results of our empirical study. Our answers for each research question are presented and supported by our data and interpretation.

4.1 RQ_1: Source Code Similarity

This study found strong evidence indicating that source code is statistically more similar to reader summaries than author summaries.

Figure 3 shows histograms for Overlap percentages (without stop words), LSS scores, and STASIS scores (without word order). The white bars in the histograms represent similarity between author summaries and source code. The black bars represent similarity between reader summaries and source code. The overlap histogram shows the similarity between author summaries and source code having a large number of low overlap percentages. 23 of 82 author summaries have 0 % overlap with the source code. Just 26 of 360 reader summaries have 0 % overlap with the source code. The reader summary overlap percentages peak in the 40 %–50 % range, while more than half of author summaries have an overlap percentage less than 20 %. Only 8 % of reader summaries have an overlap percentage less than 20 %.

The LSS and STASIS (without word order) histograms in Fig. 3 also indicate that reader summaries are more similar to source code than author summaries. In the LSS histogram, 70 % of reader summaries, when compared to source code, score higher than .9, and 29 % score higher than .95. Only 58 % of author summaries, when compared to source code, score higher than .9, and just 10 % score higher than .95. In the STASIS (without word order) histogram, 54 % of STASIS scores comparing author summaries to source code score less than .6. Only 46 % of STASIS scores comparing reader summaries to source code score less than .6.

We use the Mann-Whitney statistical analysis to test the null hypotheses:

[4]http://www.nd.edu/~pmcburne/sumalyze/

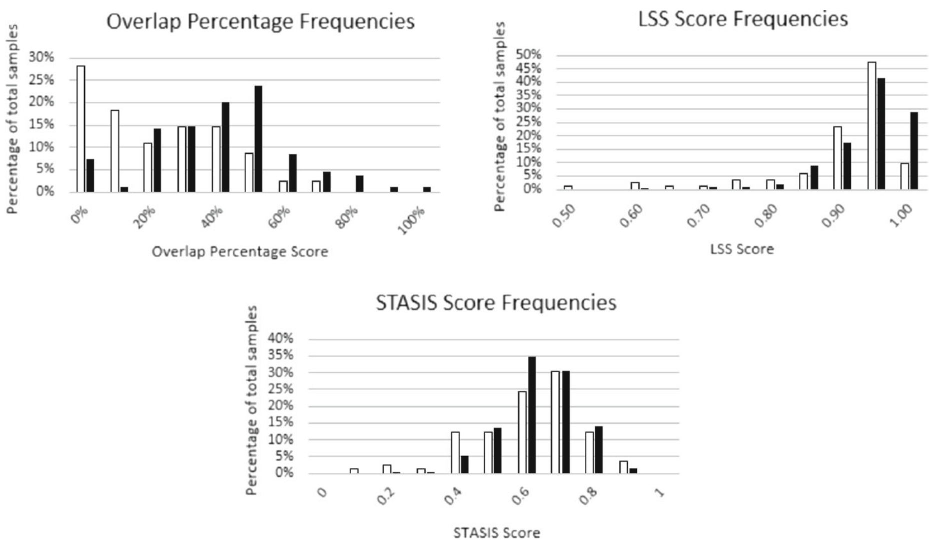


Fig. 3 These histograms illustrate the distribution of similarity scores. *White bars* are scores from comparing author summaries to source code. *Black bars* are scores from comparing reader summaries to source code. The overlap percentage without stop words histogram (*top left*) clearly shows that reader summaries have more overlap with source code than author summaries. LSS (*top right*) and STASIS without word order (*bottom*) also show reader summaries being more similar to source code than author summaries

H_0 Source code is not significantly more or less similar to author summaries as it is to reader summaries, when similarity is measured using *metric*.

Metric is a placeholder for Overlap with and without stop words, LSS, and STASIS with and without word order. As a result of this, we have five null hypotheses to test.

Our Mann-Whitney tests in Table 3 confirm what our histograms suggest. Reader summaries are statistically more similar to source code using Overlap with stop words, Overlap without stop words, and LSS than author summaries. STASIS, however, gives mixed results. For STASIS (with word order) we cannot reject the null hypothesis the difference between the similarity of reader summaries to source code and the similarity of author summaries and source code is not statistically significant, as the p-value is greater than $\alpha = .05$. However, the Mann-Whitney test for STASIS (without word order) indicates, with a p-value $<.0001$, that reader summaries are statistically more similar to source code than author summaries.

Four of our five metrics show statistically via Mann-Whitney test that reader summaries are more similar to source code than author summaries are to source code. The lack of significance for STASIS (with word order) cannot justify disregarding the other four metrics being significantly different. Therefore, our answer for RQ_1 is that source code is more similar to reader summaries than author summaries.

4.2 RQ$_2$: Reader Similarity

Our results for RQ_2 are inconclusive: our statistical tests provide conflicting results, so we are unable to claim with certainty whether reader summaries are more similar to other readers or source code.

The results of our histogram analysis are conflicting. The histogram for LSS scores in Fig. 4 suggests a higher similarity between reader summaries and source code than reader

Table 3 Statistical summary for RQ_1 and RQ_2

RQ	Source	Comp.	Metric	n	Median	Mean	Var	U	Uexpt	Uvari	Z	Zcrit	p	Decision	Winner
RQ1	Source	Author	Overlap (w/	81	0.143	0.193	0.032	10544	14540	1067238	3.87	1.96	<1e-3	Reject	Reader
		Reader	stop words)	359	0.259	0.273	0.027								
		Author	Overlap (w/o	81	0.121	0.187	0.034	7322	14540	1065203	6.99	1.96	<1e-3	Reject	Reader
		Reader	stop words)	359	0.375	0.372	0.203								
		Author	LSS	81	0.912	0.876	0.009	10836	14540	1068653	3.58	1.96	<1e-3	Reject	Reader
		Reader		359	0.926	0.914	0.003								
		Author	STASIS (w/	81	0.600	0.574	0.018	14415	14540	1068653	0.12	1.96	0.905	Not	N/A
		Reader	word order)	359	0.590	0.585	0.013							Reject	
		Author	STASIS (w/o	81	0.598	0.561	0.026	12267	14539	1068653	2.20	1.96	0.028	Reject	Reader
		Reader	word order)	359	0.617	0.611	0.014								
RQ2	Reader	Source	Overlap (w/	359	0.296	0.273	0.027	153932	112726	18548100	9.57	1.96	<1e-3	Reject	Readers
		Readers	word order)	628	0.375	0.385	0.039								
		Source	Overlap (w/o	359	0.375	0.372	0.041	128152	112726	18598956	3.58	1.96	<1e-3	Reject	Source
		Readers	stop words)	628	0.300	0.330	0.044								
		Source	LSS	359	0.926	0.914	0.003	65259	56363	6331443	3.54	1.96	<1e-3	Reject	Source
		Readers		314	0.914	0.893	0.006								
		Source	STASIS (w/	359	0.590	0.585	0.013	72763	56363	6331443	6.52	1.96	<1e-3	Reject	Readers
		Readers	word order)	314	0.658	0.641	0.013								
		Source	STASIS (w/o	359	0.617	0.611	0.014	67645	56363	6331444	4.48	1.96	<1e-3	Reject	Readers
		Readers	word order)	314	0.650	0.668	0.016								

The Source column is the item we are comparing our two "Comp." elements using the specified metric. For example, in row 1, the we are comparing the Author to the Source using the Overlap (with stop words) metric. U, U_{expt}, and U_{vari} are Mann-Whitney test values. Decision criteria are Z Z_{crit}, and p. The Winner column states which comparison item was statistically more similar to the source for a given metric, except when there is no statistical difference

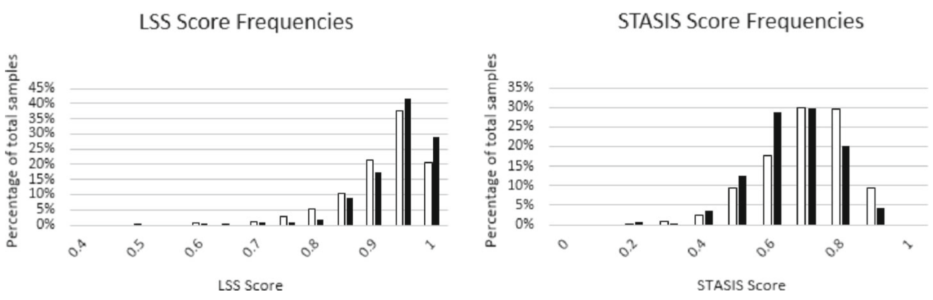


Fig. 4 These histograms illustrate the distribution of similarity scores. *White bars* are scores from comparing reader summaries to other reader summaries. *Black bars* are scores from comparing reader summaries to source code. Here, we have conflicting results. The LSS scores (*left*) seem to show there is more similarity between reader summaries to source code. However, the STASIS scores (*right*) shows readers being more similar to other reader summaries than to source code

summaries compared to other reader summaries. 70.2 % of comparisons between reader summaries and source code using LSS scored greater than 90 %. Just 58.3 % of LSS scores comparing reader summaries to source code were greater than 90 %. However, the STASIS histogram gives the opposite suggestion. Here, the reader summary similarity to other reader summaries seems shifted rightward compared to the similarity distribution between reader summaries and source code. Nearly 40 % of comparisons from reader summaries to other readers scored above .8, compared to just 24 % of comparisons from reader summaries to source code scoring above .8, implying that reader summaries are more similar to summaries written by other readers than source code.

We use Mann-Whitney to test the following null hypotheses, with *metric* being a placeholder for Overlap with and without stop words, LSS, and STASIS with and without word order:

H_0 Reader summaries are not significantly more or less similar to other reader summaries than they are to source code, when similarity is measured using *metric*.
There is no significant difference between the similarity of reader summaries to source code and reader summaries to other reader summaries using *metric*.

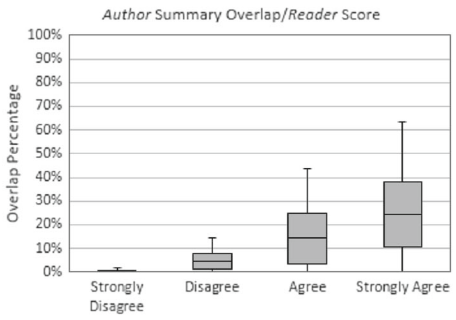
Our Mann-Whitney tests in Table 3 directly conflict one another. All five tests reject the null hypothesis that reader summaries are equally similar to source code and other reader summaries. However, the tests suggest conflicting outcomes. Two of our five tests, LSS and Overlap (with stop words) had p-values less than .0001 indicating reader summaries are more similar to source code than to other reader summaries. However, STASIS (with word order), STASIS (without word order), and Overlap (without stop words) indicate the opposite. Each of the three tests produced a p-value less than .0001 and indicate that reader summaries are more similar to other reader summaries than to source code. Because of these conflicting results, our findings are inconclusive. We cannot make a claim as to whether reader summaries are more similar to other reader summaries or more similar to source code.

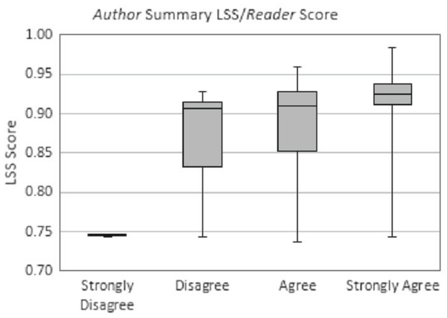
4.3 RQ$_3$: Author Similarity

Our study found evidence to support a moderate to strong positive correlation between the similarity from author summary to source code and the reader perceived accuracy of the author summary.

The Overlap Percentage chart in Fig. 5 illustrates the distribution of the Overlap percentages (without stop words) from author summary to source code broken down for each reader response regarding accuracy. The median overlap percentage when the reader “Strongly Disagrees” that the author summary was accurate was 0.0 %. This means more than half of the cases where the reader “Strongly Disagrees” that a summary is accurate, the summary shared no keywords with the source code. By contrast, when the reader “Strongly Agrees” the summary is accurate, the median overlap percentage was 24.2 %. The quartile chart illustrates that the median, as well as the lower and upper quartiles, increases as reader perceived accuracy increases.

The LSS quartile chart in Fig. 5, also shows an increase in the median similarity score as reader perceived accuracy increases. The median LSS similarity when the reader “Strongly Disagrees” that the author’s summary is accurate is .746. By contrast, when the reader “Strongly Agrees” that the author’s summary is accurate, the median similarity is .924. The median STASIS score (without word order) also increases as reader perceived accuracy increases. When the reader “Strongly Disagrees” that the author summary is accurate, the median STASIS score without word order is .341. When the reader “Strongly Agrees” that the author is accurate, the median score is .543. However, the median STASIS score (without word order) for “Agree” is .547, slightly higher than the median for “Strongly Agree.”





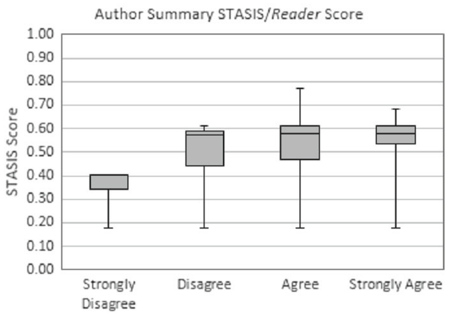


Fig. 5 These charts show the quartile distribution of Overlap percentages (without stop words) (*top*), LSS scores (*bottom left*), and STASIS scores (without word order) (*bottom right*) for summaries broken down by reader perceived accuracy of author summaries. The *box* shows the upper and lower quartiles, with the *black line* showing the median. The *small lines* above and below the box show the minimum and maximum. Note that the LSS scores chart uses a different scale. This was done to more precisely illustrate the differences in scores for each accuracy rating. All three charts show increases in similarity scores as reader perceived accuracy improves

We perform a Pearson correlation statistic test for the following null hypothesis. As before, *metric* is a placeholder for our five similarity metrics:

H_0 There is no statistical correlation between reader perceived accuracy of an author summary and similarity between that author summary and source code when similarity is measured using *metric*.

To calculate Pearson correlation, numbers are assigned to the reader accuracy scores. "Strongly Disagree" is given a value of 1, "Disagree" is given a value of 2, "Agree" is given a value of 3, and "Strongly Agree" is given a value of 4. In this way, a positive correlation would mean high similarity scores are correlated with high accuracy. The results of our calculations for Pearson correlation are shown in the author row of Table 4. LSS, Overlap (with stop words), and Overlap (without stop words) each had a Pearson score greater than .4, indicating a strong positive correlation. STASIS (with word order) and STASIS (without word order) both had a Pearson correlation score less than .4 but greater than .3, indicating a moderate positive correlation.

Our findings provide evidence to suggest that reader perceived accuracy of author summaries is positively correlated with the similarity between author summaries and source code. Two of our three metrics suggest a strong positive correlation, with the third showing a moderate correlation. The implications of these findings are discussed in Section 5.

4.4 RQ$_4$: Automated Summary Similarity

Two automatic summarization tools, Sridhara and Sumslice, were studied to see if the accuracy of an automatically generated summary was correlated with the similarity of that summary to the source code. Sridhara and Sumslice performed differently when examined for correlation between source code similarity and summary accuracy. Summaries generated using Sridhara have a weak positive correlation between perceived reader accuracy and summary similarity to source code in all five metrics. Summaries generated using Sumslice did not have a significant correlation between accuracy and similarity to source code for any of the five metrics.

The distribution of automated summary similarity for Sridhara generated summaries is shown in Fig. 6. There is an increase in the LSS similarity score from "Strongly Disagree"

Table 4 This table shows the Pearson correlation score between accuracy and the similarity of summary to source code

Pearson correlation scores

Summary source	Overlap (w/ stop words)	Overlap (w/o stop words)	STASIS (w/ word order)	STASIS (w/o word order)	LSS
Author	0.510	0.478	0.377	0.354	0.515
Sridhara	0.238	0.224	0.227	0.245	0.228
Sumslice	.042	.041	.168	.174	.035

The similarity of author summaries to source code has a moderate to strong positive correlation with reader perceived accuracy of a summary for all metrics. The accuracy of summaries generated using Sridhara has weak positive correlation with accuracy for all metrics. For Sridhara, all five metrics had a Pearson correlation score greater than .2, but less than .3, indicating a weak positive correlation. The similarity of Sumslice summaries was not correlated with accuracy for any of the five metrics. The Pearson correlation scores for all five metrics with Sumslice, while positive, were less than .2, and are therefore not statistically significant

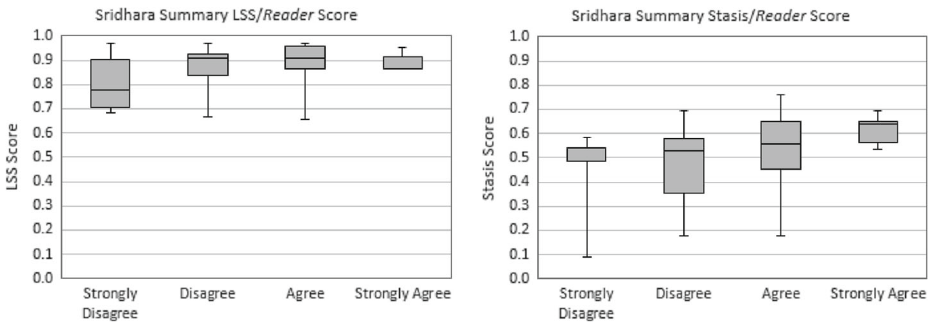


Fig. 6 This charts shows the distribution of LSS scores (*left*) and STASIS without word order scores (*right*) of Sridhara generated summaries broken down by reader perceived accuracy. The *box* shows the upper and lower quartiles, with the *black line* showing the median. The *small lines* around the box show the minimum and maximum. Both charts show an upward trend in similarity as accuracy improves, though there is a slight visible dip in the LSS scores between "Agree" and "Strongly Agree". The visual evidence suggests a positive correlation between source code similarity and accuracy for summaries generated by Sridhara

to "Disagree", and again from "Disagree" to "Agree". However, there is a decrease in the median between "Agree" and "Strongly Agree". A possible reason for this is that readers "Strongly Agreed" that only 5 of 59 Sridhara generated summaries were accurate, meaning there was not enough data for a stable conclusion. STASIS (without word order) trends slightly upward as accuracy improves.

We perform a Pearson correlation test using Sridhara generated summaries to test the following null hypotheses:

H_0 There is no statistical correlation between reader perceived accuracy of an automatically generated summary and similarity between that automatically generated summary and the source code when similarity is measured using *metric*.

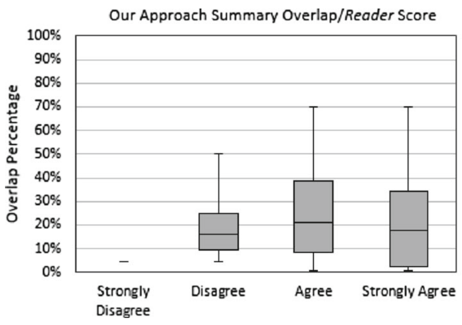
We found that all five similarity metrics on Sridhara had a Pearson correlation score between .2 and .3 (see Table 4). This indicates a weak positive correlation between the similarity of a summary generated by Sridhara and that summary's accuracy in describing the source code.

Figure 7 shows the quartile charts for the Overlap (without stop words) and LSS metrics. The source code similarity of the summary generated by Sumslice is not clearly correlated to accuracy in these charts. Note that there was only one "Strongly Disagree" accuracy rating among the 65 Sumslice generated summaries readers were given.

We perform a Pearson correlation test to examine if the reader perceived accuracy of Sumslice generated summaries is correlated with the similarity of those summaries to source code. We test the following null hypotheses:

H_0 There is no correlation between reader perceived accuracy of an automatically generated summary using Sumslice and similarity between that summary and source code with respect to *metric*.

All five metrics, including both variations of Overlap and STASIS, have a positive Pearson correlation score. However, because all of these scores are less than .2, no statistical statement of correlation can be made. Thus, we cannot reject the null hypothesis that the similarity of Sumslice generated summaries to source code is not correlated with reader perceived accuracy of the summary in describing the source code.



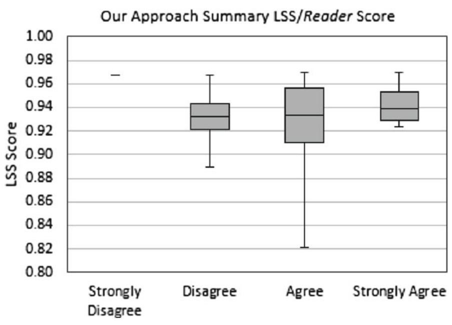


Fig. 7 This charts shows the distribution of Overlap (without stop words) percentages (*left*) and LSS scores (*right*) for Sumslice generated summaries broken down by reader perceived accuracy. The *box* shows the upper and lower quartiles, with the *black line* showing the median. The *small lines* around the box show the minimum and maximum. There was only one "Strongly Disagree" score. There is no clear visual trend linking accuracy to similarity in either chart

Our results suggest automatically generated summaries and their similarity to source code can be correlated with reader perceived accuracy, such as in Sridhara. However, there may also be no correlation, such as in Sumslice. Based on our findings, we can only suggest that each summarization tool must be investigated individually. Further discussion of the impacts of these findings can be found in Section 5.

5 Discussion

In this section, we discuss the impact of our findings. We draw three key conclusions to guide work in automated source code summarization:

1. Source code documentation should use keywords from source code. We have independently validated the assumption on which other approaches have been based.
2. Short text similarity metrics can be used to estimate accuracy for human written summaries.
3. However, short text similarity metrics do not estimate accuracy for automatically generated summaries.

Documentation that describes source code should contain the same keywords as the source code. This suggest that readers believe a "good summary" should use source code keywords and semantics. The accuracy of author summaries is higher when the summaries are similar to the source code, as shown in RQ_3. Our results showed a strong to moderate positive correlation between accuracy and the similarity of author summaries to source code. RQ_1 however, shows a disparity where author summaries are less similar to source code than reader summaries in 4 of our 5 metrics. Authors, therefore, can improve their documentation through the use of keywords from source code in their documentation. This finding also impacts automatically generated summary tools. Future automatically generated summary tools should make use of source code word choice and semantics in order to best communicate the source code readers.

Short text similarity metrics can estimate the accuracy of source code summaries when those summaries are written by humans. RQ_3 demonstrates that high accuracy of author, or human, written summaries was strongly correlated with high scores in 3 of our 5 similarity metrics: LSS, Overlap (with stop words), and Overlap (without stop words). High STASIS (with word order) and STASIS (without word order) scores were moderately correlated with high accuracy. Automatic comment quality assessment tools, such as Javadoc Miner (Khamis et al. 2010) would benefit from our findings. Using the similarity of source code to author summaries can improve Javadoc Miner's analysis of code/comment consistency.

However, accuracy should not be conflated with overall summary quality. While high accuracy is considered a component in a good summary (McBurney and McMillan 2014; Sridhara et al. 2010; Steidl et al. 2013)), other conditions have been suggested. Both our previous work McBurney and McMillan (2014) and Sridhara et al. (2010) examine a summary with respect to conciseness and content adequacy. A concise summary is one that contains little or no unnecessary information. A content adequate summary is one that contains all necessary information. Sridhara et al., noticed that the relationship between conciseness and content adequacy is similar to the relationship between precision and recall. Steidl et al. (2013) use their own set of conditions to assess comment quality. Some of these conditions cannot be automated by relying on short text similarity metrics.

For an example, consider the Java method shown in Fig. 8 with the given author summary. The author summary focuses on the high-level view of the method. This summary accurately describes the high-level idea behind the method, which is to determine if an entity object is external. A reader may write a summary such as "Set obj equal to the entity 'name'. If obj is an instance of a String, return false." This summary is accurate, but does not communicate high-level functionality. This low-level summary is more textually similar to the method than the author summary. This is an example of when higher textual similarity does not result in a higher quality summary. Both summaries are accurate, but the author summary would be more useful to a reader. This shows that accuracy, while important, cannot substitute for overall comment quality. Nevertheless, textual similarity can inform method summary quality, as shown in our findings.

Short text similarity metrics do not effectively estimate accuracy for automatically generated summaries. In RQ_4, we found that the similarity of Sridhara summaries to source code was only weakly correlated to accuracy across all 5 metrics. Contrast this with RQ_3, where accuracy was moderately or strongly correlated with the similarity of author summaries to source code across all 5 metrics. The similarity of source code to Sumslice summaries was not significantly correlated with accuracy in any of the 5 metrics. One possible explanation is that readers took issue with grammar or writing style in both Sridhara and Sumslice. Participants in our previous study frequently mentioned this for both summarization tools in our previous work (McBurney and McMillan 2014). Errors in natural language generation for automatically generated summaries may more significantly affect reader perception than

```
\**
 * Returns true if an entity is external.
 *\

public boolean isExternalEntity(String name)
{
   Object obj = this.entities.get(name);
   return ! (obj instanceOf java.lang.String);
}
```

Fig. 8 This is the source code and author summary for the isExternalEntity() method in the XMLEntityResolver class of NanoXML. The authors give a high level summary that describes the methods functionality. However, the summary is only somewhat textually similar to the source code. This poses a unique problem to our approach

the summary's content. However, we cannot be sure if improving natural language would result in a strong correlation between accuracy and the similarity of source code to automatically generated summaries. Examining this question is beyond the scope of this paper. The lack of correlation between accuracy and source code similarity to Sumslice in particular may be caused by Sumslice summaries focusing on the external interaction between Java methods, rather than focusing on how a method works internally. However, we cannot say for certain that this is the reason for the lack of correlation for Sumslice summaries.

6 Related Work

There are three areas of software engineering literature that are related to our work: studies of program comprehension, source code comment quality metrics, and automated source code summarization. We present the most-related of those approaches in this section.

6.1 Program Comprehension

Different studies have been published in program comprehension over several decades. These studies consistently show that programmers follow either a "systemic" strategy for understanding source code, or an "opportunistic" one (Ko and Myers 2005; Littman et al. 1987; Brandt et al. 2010; Kotonya et al. 2008; Davison et al. 2000; Lakhotia 1993; Holmes and Walker 2013; Ko et al. 2006). The difference between these two strategies is that in a systemic strategy, programmers try to understand how the components of a program interact, while in an opportunistic strategy, programmers only seek to understand how to modify a small section of code. These studies are relevant to our work in that they provide clues about how programmers will read source code, but they do not provide guidance on how the programmers summarize source code. One recent study does suggest that programmers only turn to documentation after they have attempted face-to-face communication (Roehm et al. 2012). This study would seem to suggest that summarization tools mimic the types of information developers seek during face-to-face conversations. Some studies indicate that this information tends to be about the interactions between different components in source code (Stylos and Myers 2006; Holmes and Walker 2013). Meanwhile, other studies point to evidence that documentation should contain primarily high-level concepts about source code (Lethbridge et al. 2003). Studies of note-taking among programmers suggest a mixed view, that summaries should contain details about code sections and high-level concepts (Guzzi 2012). Our work adds to this body of knowledge by targetting the problem of summarization, and presenting guidance for designers of source code summarization tools.

6.2 Source Code Comment Quality

Two recent studies highlight the problem of quality in source code comments. Steidl et al. (2013) present an automated approach for measuring comment quality based on four factors: coherence, usefulness, completeness, and consistency. The approach estimated these factors by calculating the textual similarity between the comments and the code, and also the length of each comment. The idea is that long comments which are textually similar to the code are considered "high quality" comments. Short comments that are not textually similar are not considered to have lower quality. That approach is similar to work by Khamis et al. (2010), in which textually similar comments are considered to have higher quality. Our work is related to these approaches in that it tests the same underlying assumption: that high quality comments are similar to source code. Unlike previous work, we empirically test this assumption for summaries written by human experts, as well as summaries written by two different automatic source code summarization tools.

6.3 Source Code Summarization Tools

In addition to the tools use in our study (see Section 3.7), a small number of automatic source code summarization tools have been proposed. Haiduc et al., in work that has been independently verified (Eddy et al. 2013), suggest using a Vector Space Model approach to extract important keywords from source code (Haiduc et al. 2010). Moreno et al. propose an approach, based on work by Sridhara et al. (2010, 2011b), that matches Java classes to Java class stereotypes, and uses different templates depending on the stereotype (Moreno et al. 2013). Earlier work by Buse et al. has focused on Java Exceptions (Buse and Weimer 2008) and change logs (Buse and Weimer 2010). An assumption common to these approaches is that the source code summaries can be created from the source code itself. Our work is related to these approaches in that we test this assumption and provide guidance for future development of these tools.

7 Conclusion

We have presented an empirical study exploring the similarity between author summaries, reader summaries, and source code. Our study is novel in that we use Short Text Semantic Similarity (STSS) metrics to examine these similarities, specifically Overlap, LSS, and STASIS. We found that readers use source code in their summaries more than authors. We also found that STSS between source code and human written summaries can estimate the accuracy of summaries as perceived by readers. This addresses a gap in literature by showing that a “good summary” written by an author should have high semantic similarity to source code. Automatically generated summaries, however, were inconsistent, and their similarity to source code could not consistently estimate reader perceived accuracy.

Acknowledgments The authors would like to thank David Croft for furnishing an LSS implementation. We would also like to thank the 23 participants of the case studies on which this paper is based for their time and efforts.

References

Biggersta TJ, Mitbander BG, Webster D (1993) The concept assignment problem in program understanding. In: Proceedings of the 15th international conference on soft-ware engineering, ICSE ’93. IEEE Computer Society Press, Los Alamitos, pp 482–498. http://dl.acm.org/citation.cfm?id=257572.257679

Brandt J, Dontcheva M, Weskamp M, Klemmer SR (2010) Example-centric programming: integrating web search into the development environment. In: Proceedings ofthe 28th international conference on human factors in computing systems, CHI ’10. ACM, New York, pp 513–522. doi:10.1145/1753326.1753402

Burden H, Heldal R (2011) Natural language generation from class diagrams. In: Proceedings of the 8th international workshop on model-driven engineering, verication and validation, MoDeVVa. ACM, New York, pp 8:1–8:8. doi:10.1145/2095654.2095665

Buse RP, Weimer WR (2008) Automatic documentation inference for exceptions. In: Proceedings of the 2008 international symposium on software testing and analysis, ISSTA ’08. ACM, New York, pp 273–282. doi:10.1145/1390630.1390664

Buse RP, Weimer WR (2010) Automatically documenting program changes. In: Proceedings of the IEEE/ACM international conference on Automated software engineering, ASE '10. ACM, New York, pp 33–42. doi:10.1145/1858996.1859005

Croft D, Coupland S, Shell J, Brown S (2013) A fast and efficient semantic short text similarity metric. In: Proceedings of the 13th UK workshop on computational intelligence, UKCI '13, pp 221–227

Davison JW, Mancl DM, Opdyke WF (2000) Understanding and addressing the essential costs of evolving systems. Bell Labs Tech J 5(2):44–54

de Souza SCB, Anquetil N, de Oliveira KM (2005) A study of the documentation essential to software maintenance. In: Proceedings of the 23rd annual international conference on design of communication: documenting & designing for pervasive information, SIG-DOC '05. ACM, New York, pp 68–75. doi:10.1145/1085313.1085331

Eddy B, Robinson J, Kraft N, Carver J (2013) Evaluating source code summarization techniques: replication and expansion. In: Proceedings of the 21st international conference on program comprehension, ICPC '13

Fluri B, Wursch M, Gall HC (2007) Do code and comments co-evolve? On the relation between source code and comment changes. In: Proceedings of the 14th working conference on reverse engineering, WCRE '07, IEEE Computer Society, Washington, DC, pp 70–79. doi:10.1109/WCRE.2007.21

Forward A, Lethbridge TC (2002) The relevance of software documentation, tools and technologies: a survey. In: Proceedings of the 2002 ACM symposium on document engineering, DocEng '02. ACM, New York, pp 26–33. doi:10.1145/585058.585065

Guzzi A (2012) Documenting and sharing knowledge about code. In: Proceedings of the 2012 international conference on software engineering, ICSE 2012. IEEE Press, Piscataway, pp 1535–1538. http://dl.acm.org/citation.cfm?id=2337223.2337476

Haiduc S, Aponte J, Moreno L, Marcus A (2010) On the use of automated text summarization techniques for summarizing source code. In: Proceedings of the 2010 17th working conference on reverse engineering, WCRE '10. IEEE Computer Society, Washington, DC, pp 35–44. doi:10.1109/WCRE.2010.13

Holmes R, Walker RJ (2013) Systematizing pragmatic software reuse. ACM Trans Softw Eng Methodol 21(4):20:1–20:44. doi:10.1145/2377656.2377657

Ibrahim WM, Bettenburg N, Adams B, Hassan AE (2012) Controversy corner: on the relationship between comment update practices and software bugs. J Syst Softw 85(10):2293–2304. doi:10.1016/j.jss.2011.09.019

Kajko-Mattsson M (2005) A survey of documentation practice within corrective maintenance. Empir Softw Eng 10(1):31–55. doi:10.1023/B:LIDA.0000048322.42751.ca

Khamis N, Witte R, Rilling J (2010) Automatic quality assessment of source code comments: the JavadocMiner. In: Hopfe CJ, Rezgui Y, Métais EM, Preece AD, Li H (eds) 15th international conference on applications of natural language to information systems (NLDB 2010). Lecture Notes in Computer Science (LNCS), vol 6177/2010. Springer, Cardi, pp 68–79. doi:10.1007/978-3-642-13881_27. http://www.springerlink.com/content/n67470n270mt61m1/fulltext.pdf

Ko AJ, Myers BA (2005) A framework and methodology for studying the causes of software errors in programming systems. J Vis Lang Comput 16(12):41–84. doi:10.1016/j.jvlc.2004.08.003. http://www.sciencedirect.com/science/article/pii/S1045926X04000394

Ko AJ, Myers BA, Coblenz MJ, Aung HH (2006) An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Trans Softw Eng 32(12):971–987. doi:10.1109/TSE.2006.116

Kotonya G, Lock S, Mariani J (2008) Opportunistic reuse: lessons from scrapheap software development. In: Proceedings of the 11th international symposium on component-based software engineering, CBSE '08. Springer, Berlin, pp 302–309. doi:10.1007/978-3-540-87891-9_20

Kramer D (1999) Api documentation from source code comments: a case study of javadoc. In: Proceedings of the 17th annual international conference on computer documentation, SIGDOC '99. ACM, New York, pp 147–153. doi:10.1145/318372.318577

Lakhotia A (1993) Understanding someone else's code: analysis of experiences. J Syst Softw 23(3):269–275. doi:10.1016/0164-1212(93)90101-3

Lethbridge T, Singer J, Forward A (2003) How software engineers use documentation: the state of the practice. IEEE Softw 20(6):35–39. doi:10.1109/MS.2003.1241364

Li Y, Bandar ZA, McLean D (2003) An approach for measuring semantic similarity between words using multiple information sources. IEEE Trans Knowl Data Eng 15(4):871–882. doi:10.1109/TKDE.2003.1209005

Li Y, McLean D, Bandar ZA, O'Shea JD, Crockett K (2006) Sentence similarity based on semantic nets and corpus statistics. IEEE Trans Knowl Data Eng 18(8):1138–1150. doi:10.1109/TKDE.2006.130

Liblit B, Begel A, Sweeser E (2006) Cognitive perspectives on the role of naming in computer programs. In: Proceedings of the 18th annual psychology of programming workshop. Psychology of Programming Interest Group, Sussex

Littman DC, Pinto J, Letovsky S, Soloway E (1987) Mental models and software maintenance. J Syst Softw 7(4):341–355. doi:10.1016/0164-1212(87)90033-1. http://www.sciencedirect.com/science/article/pii/0164121287900331

Mani S, Catherine R, Sinha VS, Dubey A (2012) Ausum: approach for unsupervised bug report summarization. In: Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering, FSE '12. ACM, New York, pp 11:1–11:11. doi:10.1145/2393596.2393607

Mann H, Whitney D (1947) On a test of whether one of two random variables is stochastically larger than the other. Ann Math Stat 18(1):50–60

McBurney PW, McMillan C (2014) Automatic documentation generation via source code summarization of method context. To appear in proceedings of the 22nd international conference on program comprehension, ICPC '14. New York, pp 1–10

Miller GA (1995) Wordnet: a lexical database for english. Commun ACM 38:39–41

Moreno L, Aponte J, Giriprasad S, Marcus A, Pollock L, Vijay-Shanker K (2013) Automatic generation of natural language summaries for java classes. In: Proceedings of the 21st international conference on program comprehension, ICPC '13

Roehm T, Tiarks R, Koschke R, Maalej W (2012) How do professional developers comprehend software? In: Proceedings of the 2012 international conference on software engineering, ICSE 2012. IEEE Press, Piscataway, pp 255–265. http://dl.acm.org/citation.cfm?id=2337223.2337254

Singer J, Lethbridge T, Vinson N, Anquetil N (1997) An examination of software engineering work practices. In: Proceedings of the 1997 conference of the centre for advanced studies on collaborative research, CASCON '97. IBM Press, p 21. http://dl.acm.org/citation.cfm?id=782010.782031

Sridhara G, Hill E, Muppaneni D, Pollock L, Vijay-Shanker K (2010) Towards automatically generating summary comments for java methods. In: Proceedings of the IEEE/ACM international conference on automated software engineering, ASE '10. ACM, New York, pp 43–52. doi:10.1145/1858996.1859006

Sridhara G, Pollock L, Vijay-Shanker K (2011a) Automatically detecting and describing high level actions within methods. In: Proceedings of the 33rd international conference on software engineering, ICSE '11. ACM, New York, pp 101–110. doi:10.1145/1985793.1985808

Sridhara G, Pollock L, Vijay-Shanker K (2011b) Generating parameter comments and integrating with method summaries. In: Proceedings of the 2011 IEEE 19th international conference on program comprehension, ICPC '11. IEEE Computer Society, Washington, DC, pp 71–80. doi:10.1109/ICPC.2011.28

Steidl D, Hummel B, Juergens E (2013) Quality analysis of source code comments. In: Proceedings of the 21st international conference on program comprehension, ICPC '13

Stylos J, Myers BA (2006) Mica: a web-search tool for finding api components and examples. In: Proceedings of the visual languages and human-centric computing, VL-HCC '06. IEEE Computer Society, Washington, DC, pp 195–202. doi:10.1109/VLHCC.2006.32

Paul W. McBurney is a graduate research assistant at the University of Notre Dame. He completed his Master's Degree at West Virginia University in 2012. His focus is on source code summarization and program comprehension. He received the Best Paper Award at ICPC 2014. He is a GAANN fellow.

Collin McMillan is an Assistant Professor at the University of Notre Dame. He completed his Ph.D. in 2012 at the College of William & Mary, focusing on source code search and traceability technologies for program reuse and comprehension. Since joining Notre Dame, his work has focused on source code summarization and efficient reuse of executable code. Dr. McMillan's work has been supported by SimVentions Inc. and the Virginia Space Grant Consortium, and recognized by the Stephen K. Park Award.