The Evaluation of an Approach for Automatic Generated Documentation

Nahla Abid[1], Natalia Dragan[2], Michael L. Collard[3], Jonathan I. Maletic[2]

[1]Department of Computer Science
Kent State University
Kent, Ohio 44242
{nabid, jmaletic}@kent.edu

[2]Department of Management and Information Systems
Kent State University
Kent, Ohio, USA
ndragan@kent.edu

[3]Department of Computer Science
The University of Akron
Akron, Ohio
collard@uakron.edu

***Abstract*— Two studies are conducted to evaluate an approach to automatically generate natural language documentation summaries for C++ methods. The documentation approach relies on a method's stereotype information. First, each method is automatically assigned a stereotype(s) based on static analysis and a set of heuristics. Then, the approach uses the stereotype information, static analysis, and predefined templates to generate a natural-language summary/documentation for each method. This documentation is automatically added to the code base as a comment for each method. The result of the first study reveals that the generated documentation is accurate, does not include unnecessary information, and does a reasonable job describing what the method does. Based on statistical analysis of the second study, the most important part of the documentation is the short description as it describes the intended behavior of a method.**

***Index Terms*— source-code summarization, program comprehension, method stereotypes, static analysis.**

I. INTRODUCTION

During the evolution of large software systems, developers spend a great deal of time trying to locate and understand the parts of the system that are related to their current task [1-3]. A typical practice for developers is to alternate between the source code and the associated internal documentation (aka comments) to understand the intent of a method or function. Furthermore, comments are found to be critical in assisting developers to understand code more quickly and more deeply. Unfortunately, comments are oftentimes outdated or incomplete [3] due to lack of either proper maintenance or adherence to proper documentation standards.

Several techniques have been proposed to automatically generate summaries directly from source code for methods [4-9] [10] and classes [4, 7, 11]. Recently, Abid et al. [12] proposed an approach that uses stereotypes in the automatic generation of documentation for methods. An example of the automatically generated documentation for a method is presented in Fig.1. The member function `calcWidthParm()` is in the class `BinAxsLinear` from the open-source system HippoDraw. The documentation starts off with a short description, (i.e., the first three lines in Fig.1). This emphasizes the computed value along with the data members and parameters used in the method. Following is a list of calls made from the method along with the corresponding stereotypes of the functions called. Here, we present two studies that are modeled after work to evaluate the documentation approach proposed by Abid et al. [12]. A common approach to evaluate automatic summarization is to ask developers to rate the accuracy and content adequacy of a set of generated summaries [4-8, 11, 13-16].

While the comments generated can be referred to as summaries, we choose to use the term *documentation* as they are more detailed and longer than a summary. The documentation starts with a short precise description of the main responsibility of the method. Following is information about external objects, properties modified, and a list of function calls and their stereotypes.

```
/**
calcWidthParm is a property that
   returns a computed value new_width
   based on data member: m_range and parameter: number.
Calls to- high()     get
          low()      get
 */
double BinAxsLinear::calcWidthParm (int number) const {

   double new_width = (m_range.high() - m_range.low ())
                    / (static_cast < double > (number));

   return new_width;
}
```

Fig.1. An example of a *property* method and its generated documentation. It is inserted into the code as a method comment

The contribution of this paper is threefold.

1) an evaluation study to judge the quality of the generated documentation [12].
2) improving the documentation approach by considering the result from the first evaluation study.
3) a follow-up study to evaluate the improved documentation approach and investigate what parts of the documentation programmers consider necessary.

Both studies are conducted using a set of C++ methods and the automatically-generated documentation, and involved 139 participants in total. The results of the studies show that the documentation generated from templates based on the stereotype accurately expresses what the method does. Furthermore, the results of the second study show that the most important part of the generated documentation is the short description.

The remainder of this paper is organized as follows. First, section II presents the related work on evaluating automatic source-code summarization approaches followed by the research questions in section III. Then, an overview of the approach is given in section IV. Section V demonstrates the result of the first evaluation. Section VI describes the improved documentation approach followed by the second evaluation in section VII. Finally, conclusion and future work is given in section D.

II. Related Work

Automated summarization of source code [5, 8, 9, 11, 12, 14-17] and other artifacts [18-21] has been a topic of recent interest for multiple researchers. This section summarizes the different summarization approaches and the evaluation studies design performed in the literature. Sridhara et al. proposed techniques using NLP [22] to automatically generate natural language comments for Java methods [5], sequences of statements [6], and formal parameters [14]. Using NLP, the technique first extracts and identifies the linguistic elements [22] of a method. Then, based on structural and linguistic clues in the method, the important lines of code from the method are identified. While the approach produced some good results, it does require high-level quality identifiers.

To evaluate the three different approaches [5, 6, 14], survey-based studies are performed. In particular, participants are presented with the automatic documentation and three evaluation questions regarding the accuracy and content adequacy of the presented automatic-generated natural language summaries.

McBurney and McMillan proposed generating documentation summaries for Java methods that include information about the context of methods [9]. The approach uses a call graph to find other methods that rely on the output of the method being summarized. They concluded that adding their documentation improves the quality of summaries generated by NLP techniques [5]. In a later study, they proposed an approach to evaluate a summary using textual similarity of that summary to the source code [23].

In the approach proposed by Abid et al. [12], static analysis is used instead. The technique uses method stereotypes to generate a standard documentation for each method. The stereotype type of a method is used to detect the important set of lines in a method. Then, the approach uses pre-defined template to generate the final documentation summary. Moreno et al. [11] use method stereotypes and class stereotypes [24] to generate natural-language summaries for Java classes. While Abid et al. [12] also use method stereotypes as applied in [11], its main contribution is to generate documentation summaries for methods, not classes.

The evaluation study of class summaries is designed as follow. First, Moreno et al. [11] asked participants to summarize the presented class; then, answer evaluation questions used by Sridhara et al. [5, 6, 14].

Wang et al. propose a model that defines the high-level action of loops by analyzing linguistic and structure clues [17]. To evaluate the model, a user study is performed. Developers are asked to indicate how much do they agree that a loop code implements a given action.

Haiduc et al. [4] and Eddy et al. [7] investigated the suitability of several text summarization techniques (TR) to automatically generate term-based summaries for methods and classes. According to their evaluation using a user study, it is concluded that such TR may produce a better result if combined with static analysis.

Rodeghero et al. [15, 25, 26] proposed an improved version of Vector Space Model (VSM) by giving more weight to terms that developers look at more often during summarization. They concluded that this new weighting scheme outperforms existing state of the art VSM schemes when they are compared to human term-based summaries.

Mapping existing human descriptions to code segments is used to generate summaries for Java and Android systems [8, 16]. Such approach relies on the availability of human descriptions in discussion site such as StackOverflow.com.

As explained above, a user study is a commonly and successfully approach applied in the literature to evaluate automatic summarization [4-8, 11, 13-16]. Therefore, here, we present two evaluation studies that are modeled after this work.

III. Research Questions

We seek to answer the following research questions.

RQ1 To what extent does the automatically generated documentation accurately present what a method does without missing important information?

RQ2 If important information is missing, what kind of information is it?

RQ3 To what extent does information from conditional statements improves the automatically-generated documentation?

RQ4 To what extent does the absence of one section of the documentation improve/affect the results?

RQ1 and RQ2 are addressed in the first study where participants are asked to write their own documentation and evaluate the automatically-generated documentation. The second study is conducted to answer RQ3 by comparing the result of the evaluation ratings of the improved documentation approach to the ratings of original approach. Finally, RQ4 examines the value of the documentation's content without directly asking the question. Instead, we present set of examples where one section is randomly removed.

IV. Summary of The Approach

First an overview and definition are given for method stereotypes [27] as this is a key component of the approach. Method stereotypes are concise description that reflect the basic meaning and behavior of a method and include concepts such as *predicate*, *set*, and *factory*. Table I presents a summary of the stereotypes that were used in our approach.

Stereotypes that capture the method's responsibilities are utilized to select the content of the automatically generated documentation. In other words, the stereotype of the method is used to locate and determine the lines of code that reflect the main actions of a method. For example, the main action of a *property* method is to return a computed value whereas the main action for a *command* method is to modify more than one data member.

TABLE I. TAXONOMY OF METHOD STEREOTYPES AS GIVEN IN [27]

Stereotype Category	Stereotype	Description
Structural *Accessor*	predicate	Returns a Boolean value that is not a data member.
	property	Returns information about data members.
	void-accessor	Returns information through a parameter.
Structural *Mutator*	command	Performs a complex change to the object's state (this).
	non-void-command	
Creational	factory	Creates and/or destroys objects.
Collaborational	collaborator	Works with objects (parameter, local variable, and return object).
	controller	Changes an external object's state (not *this*).

The documentation summaries are produced using predefined fill-in-the-blank sentence phrases (templates) [12]. The templates are specifically constructed for different method stereotypes. The following demonstrates the steps and tools used:

1) Methods are automatically assigned stereotypes using stereoCode [27].
2) A list of all methods and their stereotypes are extracted. Fact extraction on the method is done using the srcML [13] infrastructure (www.srcML.org). During summary generation, the stereotypes of calls are looked up, which provides additional information about the actions taken upon the call.
3) After the appropriate templates are matched to the method, they are filled with data to form the complete documentation.

An example of the automatically generated documentation for a *command* method is presented in Fig.2. A *command* method executes a complex change of the object's state. The change involves more than one data member either directly or indirectly using another *mutator*. To identify whether a data member is modified by a call, we consider the stereotype of the call. When a *mutator* (e.g., a command) is called on a data member, we can conclude that the data member is modified by the call using the following template:

```
Data members modified- <data-member1>, …, <data-membern>
                    <data-member> via <stereotype>: <call>
```

In Fig.2, a method call `resize()` is invoked by four data members: `m_sum`, `m_num`, `m_sumsq`, and `m_sumwtsq`. Because the stereotype of the call `resize()` is *command,* we conclude that the data member is modified by the call. Therefore, the short description, (i.e., the first four lines in Fig.2) includes the four data members that are modified by a call inside the method and one that is changed via assignment statement. Following is a list of calls made from the method along with the corresponding stereotypes of the functions called.

```
/*
resize is a command that
  modifies 5 data member(s): m_sum, m_num, m_sumsq,
                             and m_sumwtsq via resize()
                             m_values_dirty
calls to- reset() coammnd
*/
void Bins1DProfile::resize ( int number )
{
  m_sum.resize( number + 2 );
  m_num.resize( number + 2 );
  m_sumsq.resize( number + 2 );
  m_sumwtsq.resize( number + 2);

  reset();

  m_values_dirty = true;
}
```

Fig.2. An example of a *command* method and its generated documentation.

This automatically-generated documentation has the potential to improve source-code comprehension and speed up maintenance in several ways. First, the short description assists developers in understanding the main behavior of the method and conclude its relevancy to their current task. Second, the documentation describes the main elements of a method (e.g., external objects, data members and parameters modified) in a structured format that can be easily mapped to the code. Third, because the main side effect of the method is often caused by one or multiple calls, including an abstract summary of calls (the stereotypes of calls) provides additional information about their roles. The final documentation uses the template shown in Fig. 3 to group the four sections just described.

```
/**
1 <Short Description>
2 Collaborates with- <obj1>, ... <objm>
3 Data members or parameters modified- <name1>, ... <namep>
4 Calls to-   <call1>        <stereotype1>
                         …
              <calln>        <stereotypen>
 */
```

Fig. 3. The template of the four sections of the documentation.

V. FIRST STUDY

In order for an automatically generated method documentation to be effective, it has to be accurate. It needs to express the underlying meaning of the method, while not missing any important information nor including any unnecessary information [11].

To evaluate the approach, we performed a survey study. Participants were asked to evaluate the automatically-generated documentation, answering questions that are modeled after questions (see Table II) used by McBurney and McMillan [9] in a very similar study. We feel that using these questions, instead of developing our own, mitigates some bias and allows for comparison to related work.

A Participant is asked to write his/her summary before evaluating the automatic summary. Although the automated summary is shown above on the same page, we believe that does not affect the accuracy of the result. This is further discussed in section D.

TABLE II. QUESTIONS USED IN THE STUDY. THE FIRST QUESTION IS OPEN-ENDED. QUESTIONS TWO TO FIVE ARE ANSWERABLE AS "STRONGLY AGREE", "AGREE", "DISAGREE", AND "STRONGLY DISAGREE."

#	Survey Questions
1	In a sentence or two, please summarize the method in your own words.
2	Independent of other factors, I feel that the summary is accurate.
3	The summary is missing important information, and that can hinder the understanding of the method.
4	The summary contains a large amount of unnecessary information.
5	The summary contains information that helps me understand what the method does.

A. Study Participants

This study included 64 students in computer science: 33 graduate students and 31 junior/senior undergraduate students from three universities.

The survey was given as a class assignment and students received credit for completion. Students were assigned the survey by their instructor and given one week to complete the survey. From preliminary questions on the participants' background, 70% of the participants have 2 to 5 years of programming experience.

B. Experimental Design and Procedure

The study includes methods from three open source C++ systems (systems 1, 2, and 4 from Table III). The systems are chosen from three different domains and are all examples of mature and well-written source code.

TABLE III. AN OVERVIEW OF THE SOFTWARE SYSTEMS USED IN THE TWO STUDIES. SYSTEM 1, 2, AND 4 ARE USED IN THE FIRST STUDY. SYSTEM 2-5 ARE USED IN THE SECOND STUDY.

#	System	Domain	Classes	Methods	KLOC
1	**SmartWin++**	GUI and SOAP library	572	2882	86
2	**HippoDraw**	data analysis environment	308	3315	125
3	**ClanLib**	SDK	765	4836	183
4	**CodeBlocks**	IDE	753	11586	436
5	**OpenOffice**	data processing and database	12851	100680	5650
	Total		**1633**	**17783**	**647**

The goal of the study is to gain understanding on the quality of the generated documentation across the categories of stereotypes. Therefore, for each stereotype, three to four methods are randomly selected (see Table IV), one method per system.

A total of 30 methods are selected, ten methods per system. Methods less than 6 LOC are not included. We also avoided methods greater than 30 LOC to avoid participant fatigue as we expected the survey to last approximately one hour. Each one of 64 participants evaluated 10 methods, with each method containing a different stereotype. We felt that more than 10 would cause the survey to be too long and hamper our ability to collect valid data. The study uses a well-known survey tool (qualtrics.com) and performed online. The survey has four sections which are:

1) an introduction describing the study;
2) a one-page tutorial on method stereotypes;
3) a 5-question quiz on method stereotypes;
4) the actual survey questions.

The purpose of the quiz is to ensure that the participants understood the definition of the different stereotypes. However, we did not expect participants to remember all the stereotype definitions. Therefore, Table I is provided during the entire study.

TABLE IV. THE STEREOTYPE OF THE METHODS USED IN THE STUDY.

Stereotype Category	Number of Methods	Stereotype
Accessor	3	(1) get collaborator and (1) property (1) property collaborator
	3	(1) predicate (2) predicate collaborator
	4	(1) void-accessor (3) void-accessor collaborator
Mutator	7	(1) set and (3) command (3) command collaborator
	5	(2) non-void-command (3) non-void-command collaborator
Creational	3	(3) factory
Collaborational	5	(1) controller predicate, (1) controller property,(3) controller command
	30	Total

For each method, the automatic documentation and the method body are presented. Prior to the start of the study, we estimated the time to evaluate ten methods to be 45-60 minutes with 5-10 minutes per method. The survey tool computed the time automatically. On average, participants completed the survey in 50 minutes (shortest 20 minutes, longest 3 hours). The average time spent on a method is 5 minutes. Therefore, we believe that participants spent adequate time to evaluate each. Finally, each documentation summary is evaluated by 14 to 20 participants.

C. Results and Discussion

Fig 6 (a) presents the responses concerning the automatically generated documentation. The results are promising as the majority of ratings for each evaluation question are, on average, positive.

Hence, we answer RQ1 as:

The automatic documentation is mostly accurate and describes the internals of the method, does not include a large amount of unnecessary information, but misses important information in (13/30) = 43% of the documentation cases.

The number of responses for “strongly agree” and “strongly disagree” are comparatively small, so we combined “agree” with “strongly agree” and “disagree with strongly disagree” for analysis of the results. However, some interesting cases of “strongly agree” and “strongly disagree” are discussed.

We use the terms agreement and disagreement for the discussion in this section.

We analyzed the responses given by participants for each of the three systems. The results indicate that the documentation for all three systems are found to be accurate by the majority (79%-81%) and to not include a lot of unnecessary information (80%-85%). Some methods are rated by 60%-50% of participants as “not properly expressing what the method does”. The percentage of these methods for CodeBlocks, HippoDraw, and SmartWin are 10%, 18%, and 11% respectively.

The responses for “missing important information” received the lowest ratings compared to the other evaluation questions. The percentage of methods that are determined from the survey to be missing important information for CodeBlocks, HippoDraw, and SmartWin is 30%, 36%, and 40%, respectively. As the ratings of all three systems are not significantly different, and all systems are from different domains, we believe that this indicates that the approach is generalizable to multiple application domains. We also did not find a significant difference between the graduate and undergraduate student ratings.

More detailed information about the results is now presented. The summaries written by the participants are compared to the generated documentation and participants' comments are analyzed to understand the reasons behind their ratings.

Since the generated documentation depends on the stereotype of a method, it is important to analyze the ratings of each stereotype. This information will now be discussed in terms of the four evaluation questions.

Accuracy. All method documentation is judged to be accurate by the majority of participants. The stereotypes that had the highest rated documentation (i.e., 80% or more agreement from participants) are: *get collaborator*, *property*, *property collaborator*, *controller property*, *command*, *non-void-command collaborator* and *factory*. In fact, on average, 25% of participants choose “strongly agree” to evaluate the accuracy of summaries.

Documentation of two *predicate collaborator* methods are rated as inaccurate by 25%-35% of the participants. In one method, the short description included information about the data members and calls that control the returned value. This information is included in one list, even though it is a part of two if-conditions. The summaries of the participants reported the two if-conditions separately in two steps. In the second *predicate collaborator* method, the returned value is modified twice (via call and via assignment) and the generated documentation reflected only one (modification by the call) that is interpreted as inaccurate by some of the participants.

Finally, the documentation of one *controller* method and one *command collaborator* method is rated as inaccurate by 36% and 38% of the participants, respectively. We did not have enough information to explain these ratings. However, a common observation about their short descriptions is that they only included the number of method calls from each stereotype category (as shown in Fig. 4) without any other details of method's complex behavior which could be part of the reason for the participants rating these as less accurate.

Missing/Containing Important Information. The results of the second and fourth questions (missing important information and expressing what the method does) are related and discussed together.

It is noteworthy that documentation of 25 out of 30 methods is rated to properly express what the method does by the majority of participants. The documentation of stereotypes consistently rated to properly express what the method does are: *property*, *predicate*, *command*, *non-void-command*, *factory*, *and void-accessor* (3 out of 4 methods rated properly), *controller command* (2 out of 3 methods rated properly), and *non-void-command collaborator* (2 out of 3 methods rated properly).

In addition, 17 out of above 25 methods are rated as “not missing important information” by the majority of participants. The documentation of stereotypes that consistently rated to include all of the important information are: *command*, *non-void-command*, *void-accessor, property*.

```
/**
setLog is a command that collaborates with 6 objects and
calls 4 mutators, a controller and a collaborator
…
*/
```

Fig. 4. The short description generated for a *command collaborator* method.

Of the generated documentation cases, 5 out of 30 methods are rated by the majority to not properly describe “what the method does” and to “miss important information”. Note, all are collaborational methods.

The first in this group is a *void-accessor collaborator* method where 21% of participants choose “strongly agree” to evaluate the question of missing important information”. The generated documentation for this method included a list of objects, data members, and a list of calls without their stereotypes. The stereotypes of the calls are not determined (method calls are not originally stereotyped) by the automated approach (a rare case). This method is from CodeBlocks and only 0.01% of the methods are not stereotyped.

The next two methods of these five - *controller command* and *command collaborator* - have similar behavioral characteristics where 12%-22% of participants choose “strongly disagree” regarding the question “expressing what the method does”. Their short descriptions included the number of method calls from each stereotype category (as shown in Fig. 4). This information might be considered insufficient to reflect the method's primary behavior, as these methods are somewhat complex.

The fourth method documentation that missed important information is for a *command collaborator* method. The documentation included the data member modified along with the call that modifies it. However, the modification is done iteratively within a while-loop. Participants' documentation that did not agree with the automatically generated documentation described how the loop iterates through one local variable and modified the data member.

The last method documentation judged to "not express what the method does" is a *non-void-command collaborator* method. The automatically generated documentation correctly specifies the two method calls that are returned using an "or" conjunction. However, a pointer to an external object used to call the two methods is initialized using another call. According to participants' summaries, information about "how the local pointer is changed before the call" is considered to be critical. The same observation applied to one *controller command* method that requires further investigation.

Three cases of the automated documentation are judged to express what the method does but miss some important information. The first is for a *factory* method where the short summaries emphasized the object created and returned. Participants considered the condition that checks the success of creating the object to be important and that it should be included in the documentation. We plan to improve the template for *factory* in the future to address this issue. Similarly, in case of the *set* method, information about "under what condition the data member is modified" is expected by the participants. We plan to improve that template also.

The third method is a *property collaborator*. Its documentation included the name of two calls used to compute the returned value. However, participants' summaries emphasized the type of the returned value (i.e., a string). This information is felt to be missing by some participants. We plan to enhance the template to reflect the type of the returned value without increasing the length of the documentation (e.g., for a string, we can use "returns a computed string" instead of "returns a computed value").

The stereotype consistently rated as "missing important information" is *predicate* (2 out of 3 methods). Participants wanted information about the action performed by the method (checking the existence of an element, evaluating a value for being in a certain range, or other actions). We plan to study and improve the rules of the *predicate*'s documentation to include information from control-flow statements.

Including Unnecessary Information. All 30 methods are judged by the majority of participants to not include a lot of unnecessary information.

Only 4 out of 30 documentation received ratings (23%-32%) that indicated they included unnecessary information: two *command collaborator*, one *controller predicate*, and one *non-void-command collaborator*. We do not have enough evidence to fully explain these ratings but can make some observations. In the case of the two *command collaborator* and one *controller predicate* methods, the list of calls in the documentation included multiple *set* and *get* methods, which may be unnecessary. In one *non-void-command collaborator* method, the stereotypes of 3 out of 5 calls are not specified. Participants might consider including any calls without stereotypes to be unnecessary. Based on the above discussion, we answer RQ2 as the following:

Control-flow statements (if/switch) are found to be important for including in the method summary especially for predicates. In addition, related work suggested that local variables were judged by programmers to be "useless" to include in the summary [10]. However, we observed that information about local variables that are objects appear to be important to a number of participants.

D. Threats to Validity

A number of issues could affect the results of the survey study we performed and so may limit the generalizability of the results. The quality of the generated documentation might be influenced by the stereotype of the method. To control this, we have included three or four methods from each stereotype.

A learning effect might occur during the study. As soon as participants evaluated the first method, they knew what would be asked later. We did not try to mitigate or control this factor. However, the last section of the survey asked the participants to judge the overall documentation. We compared this rating with the participants' responses for all methods (see Fig 6 (a) and we found no significant difference.

Furthermore, the participants are graduate and undergraduate (junior and senior) students. We cannot conclude whether the study with more expert programmers will reveal the same result.

In the study when a participant writes his/her summary, he/she has access to the automated summary. Therefore, the participant's summary might be influenced by our summary. We seek to mitigate this threat in the second study. In particular, participants are not asked to write their own summaries. Instead, when a participant gives an answer of disagree on a question, a fill-in-the-blank text box appears that asks about what aspect that she/he disagrees with.

E. Summary of the results

Some *collaborator* and *controller* methods are more difficult to summarize compared to the other stereotypes. They usually perform complex tasks that involve multiple external objects. In these methods, the stereotype alone is not sufficient to determine what the method does.

The documentation approach of these methods might be better handled using by combining static analysis (method stereotypes) and part-of-speech tagging of identifiers [28] to refine the summarization templates. Such analysis is out the scope of this paper.

Of the generated summaries, 25 out of 30 were half the size of the original method or smaller. Only five summaries were about the same size as the method (in LOC). This case only occurred for short methods. However, the majority of participants consider the included information to be somewhat necessary. Therefore, we can conclude that the structured documentation summaries were found to be valuable by programmers

Finally, conditional statements are found to be important for including in the method summary. The majority of participants stated that they were missing and the lack of them decreased the overall rating of the summaries. Here, we enhance the automatic summarization by analyzing the stereotype of conditional statements.

VI. IMROVING THE AUTOMATED APPROACH

The results of the first evaluation indicated that the automatically generated documentation accurately expresses what the method does, and does not include a large amount of unnecessary information but sometimes misses important information. For example, control-flow statements are found to be important for the method documentation. The majority of participants stated that they were missing and the lack of them decreased the overall rating of the documentation.

We improve the automated approach proposed by Abid et al. [12] to include information from if and switch statements. In particular, if the main action of a method is entirely located in an if-statement, the condition is added to the short description using the following:

```
depending on data member(s): <data-mem_1>, …, <data-mem_n>
and/or parameter(s): <parameter_1>, …,  <parameter_m>
and/or value(s) computed from <call_1>, .., <call_k>
```

Fig 5 shows the improved short description of a *void-accessor* method. The main action of the method is to change the parameter `color` via the *command* method: `setColor.` Then, the documentation is improved by adding the phrase `depending on parameter(s): index`.

Note, we do not handle nested if statements at this point.

```
/* doubleToColor is a void-accessor that
modifies a parameter: color via command setColor()
        depending on a parameter: index
*/
void BinToColorMap::doubleToColor(double value,Color&
color)const
{
  double tmp = (value - m_vmin) / m_dv;
  int index = static_cast <int> (255. * std::pow(tmp,
                                        m_gamma));
  if ( index < 256 )
      color.setColor(m_reds[index],m_greens[index],
                                    m_blues[index]);
}
```

Fig 5. An example of a void-accessor method and its improved summary.

VII. SECOND STUDY

The goal of this study is to evaluate the improved approach. Furthermore, it examines the importance and necessity of the four documentation sections: short description, list of external objects, list of data members read, and list of calls and their stereotypes.

To accomplish this, each one of these sections are randomly removed from a subset of selected automatically generated documentation. The study also investigates what type of information programmers desire in a summarization.

A. Participants

The study includes 75 students in computer science: 51 undergraduate students and 24 graduate students. The survey was given as an extra credit assignment for their classes and they received credit for completion. None of the participants in the second study were involved in the first evaluation.

Students were assigned the survey by their instructor and given one to two weeks to complete the survey. From preliminary questions on the participants' background, 10% of the participants have more than 5 years of programming experience and about 50% have two to five years of programming experience. Finally, 20% of the participants have worked in industry.

B. Experimental Design and Procedure

The study includes 56 methods from four open source C++ systems (systems 2-5 from Table III). Similar to the previous study, the selected methods are from different stereotype categories. The percentages of these methods per stereotype category are: 14% *accessors*, 25% *accessor collaborators*, 8% *mutators*, 18% *mutator collaborators*, 25% *controller*, and 10% *creational methods*. The size of methods ranges from 16 to 50 LOC.

We describe the characteristic of the 56 different documentation cases generated as the following: 18 include all sections, 10 without the list of objects, 10 without the short description, 9 without the list of data members read, and 9 without the list of calls and their stereotypes. Through the analysis of the data, we refer to the above five groups as documentation categories.

The selection of the category to remove is random, with three conditions:

1) the documentation has at least one section to present.
2) removing the list of objects is required only for collaborator or controller methods.
3) the method name and its stereotype are present in all cases.

For a fair comparison, the set of methods with full documentation is twice the size of other documentation categories. The 9 full documentation cases of *collaborational* methods are compared to the 10 documentation cases without a list of objects. The 11 full documentation cases for methods that are not *controller* are compared to the 9 documentation cases without a list of data members.

This study uses the same four survey sections as in the first study. Each participant evaluated 6 documentation cases where the first two cases include all four sections to familiarize participants with the documentation structure. The other four cases include all sections except one as previously explained. Finally, participants are not aware that some sections are missing from the displayed documentation.

In addition to the questions asked in the related survey (questions 2 to 5 from Table II), we ask participants to evaluate the structure of the documentation using the following question.

"The structure of the documentation is easy to read and map to the code."

Finally, when a participant gives an answer of disagree on a question, a fill-in-the-blank text box appears that asks about what aspect that she/he disagrees with. On average, participants completed the survey in 50 min. (shortest 15 min., longest 1 hour). Each documentation summary is evaluated by 6 to 11 participants. Overall, 450 records of 75 participants are analyzed.

C. Results and Discussion

In the second study, 75 participants evaluated 18 documentation full summaries. Fig 6 presents a comparison of the first and second evaluation ratings. The rating of all the four evaluation questions improved in the following ways: 1) accuracy increases by 7%; 2) missing important information decreases by 10%; 3) expressing what the method does increases by 15%; 4) containing a large amount of unnecessary information decreases by 9%.

This change in the overall ratings suggests that information from control statements may enhance the automatic documentation. To further examine this, we conduct Mann-Whitney Test [29] to compare documentation of each method in the first study and the second study. Mann- Whitney Test is a non-parametric test that does not assume normality of the data [29] which is the suitable test to use since the our data may not be normally distributed.

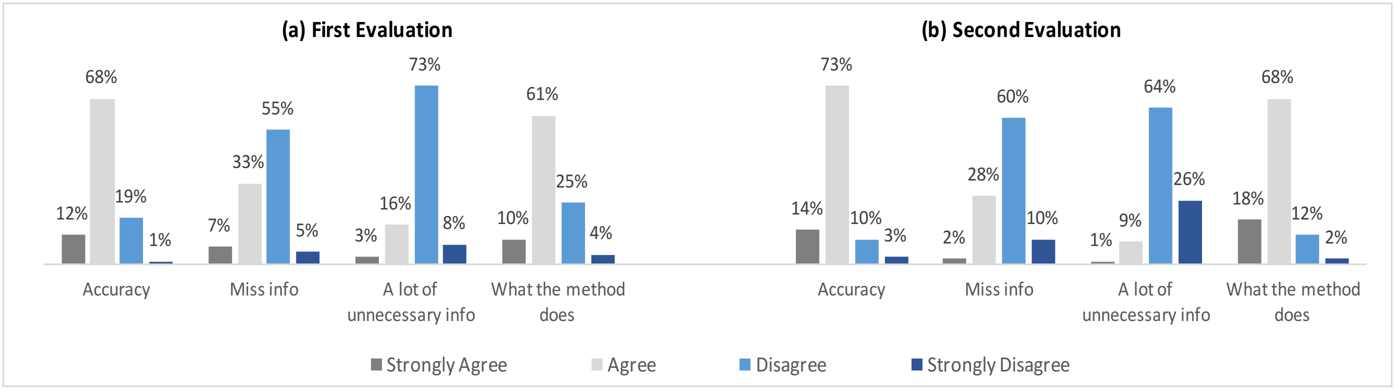

Fig 6. A comparison of the first and second evaluation

We combined "agree" with "strongly agree" and "disagree" with "strongly disagree" for analysis of the results. This is done because "strongly agree" and "strongly disagree" do not notably differ when a section of documentation is removed. Some critical increment or decrement of these values is reported through the discussion.

First, for each documentation summary, the average ratings of each evaluation question are computed. Therefore, we have 30 samples from the first study and 18 samples from the second study. Then, for each evaluation question, a Mann-Whitney Test is applied. We conclude that the result of an evaluation question is significant if and only if $p < .05$ and $|Z|>= 1.96$.

Mann-Whitney Test is applied for the four evaluation questions to assess the following hypothesis:

H_n: the difference between the rating of the first study and the second study is not significantly different for [accuracy / missing important information / including a lot of unnecessary information/ expressing what the method does].

In case of accuracy, we found a significant difference in the scores for the first study (M=74.51, SD=11.00) and second study (M=85.92, SD=9.42); U = 120, Z = -3.19, p = 0.001. These results suggest that including control-flow statements enhances the accuracy of the documentation.

Additionally, we found a significant difference in the scores for the first study (M=80.11, SD=12.18) and second study (M=85.886, SD=12.614); U = 108, Z = -3.45, p = 0.001 for expressing what the method does.

There is not a significant difference in the scores for the first study (M=41.8.56, SD=11.74) and second study (M=30.55, SD=22.21); U = 181, Z = -1.89, p = 0.06 for missing important information.

There is not a significant difference in the scores for the first study (M=16.65, SD=9.06) and second study (M=11.33, SD=10.79); U = 180, Z = -1.9, p = 0.056 in the case of containing a large amount of unnecessary information.

By comparing the average ratings of the first study to second study, we noticed improvements in all evaluation questions. The improvement in two evaluation questions: "accuracy" and "expressing what the method does" ratings are statistically significant. This suggests that control-flow statements add additional value to the generated summaries. Therefore, RQ3 is answered as:

Including information from conditional statements (if and switch) improve the accuracy and the quality of content of automated documentation approach.

To understand the effect of the absence of one section, we compute the total number of responses for each of the five documentation categories per evaluation question. To study the significance of the absence of each section for each evaluation question, we use the Mann-Whitney Test [29].

We found no statistical difference of the ratings of the last evaluation question, i.e., *structure of the documentation*, in all documentation categories. Furthermore, for all documentation categories, 82% or more of the ratings agree that the structure is easy to read and map to the code. This result suggests that the proposed structure of the documentation sections is appropriate.

No short description. A total of 75 records are collected for 10 documentation cases without short description, and compared to 150 records collected for 18 of the full documentation cases.

Comparing to full documentation, the ratings of documentation that is missing the short description decrease visibly in accuracy (13%) and what the method does (20% where half of them are *strongly disagree*). However, when participants are asked what aspect is inaccurate, they state that more information is needed. Furthermore, the ratings of no short description category increase by 23% to be missing important information. These drops in ratings indicate that the short description plays an important role in the documentation.

To further examine this, we conduct the Mann-Whitney Test. First, for each documentation case that omits short description, the average ratings of each evaluation question are computed. Second, for each full-documentation case, the average ratings of each evaluation question are computed. Finally, the Mann-Whitney Test is used to compare the two documentation categories as shown in Table V. The same steps are also repeated for the rest of the documentation categories in Table V- Table VIII.

To examine the significance of the absence of short description, we posit following hypothesis:

H_n: the difference between the rating of full documentation and documentation without short description is not significantly different for [accuracy / missing important information / including a lot of unnecessary information/ expressing what the method does / easiness of the structure].

From Table V, we reject hypothesis H1 and H2 only. Therefore, we conclude that there is statistical difference in the following evaluation questions: accuracy and missing important information. These results indicate that the short description is essential to the automatically generated documentation.

TABLE V. MANN-WHITNEY TEST FOR DOCUMENTATION WITHOUT SHORT DESCRIPTION.

H	Evaluation question	Category	N	μ	SD	U	Z	P
1	Accuracy	Full	18	85.9	9.42	45	-2.14	.03
		No short	10	71.96	18.05			
2	Miss info	Full	18	30.5	22.21	44	-2.21	.02
		No short	10	52.73	22.02			
3	A lot of unnec. info	Full	18	11.3	10.79	81	-.446	.68
		No short	10	8.33	9.41			
4	What the method does	Full	18	85.8	12.61	51	-1.87	.06
		No short	10	65.6	28.49			
5	Structure	Full	18	81.15	18.8	87	-.146	.90
		No short	10	80.37	18.94			

No calls and their stereotypes. A total of 75 records are collected for 9 documentation cases without list of calls and their stereotypes, and are compared to the ratings of 18 of the full documentation cases.

Compared to full documentation, the ratings of documentation missing the list of calls and stereotypes slightly improve in three evaluation questions. This improvement appears in case of accuracy 4%, contains unnecessary information 6%, and structure 5%. It seems that the list of calls and their stereotypes might add additional overhead for a few participants.

Mann-Whitney Test is applied for the five evaluation questions to answer the following hypothesis:

H_n: the difference between the rating of full documentation and documentation without the list of calls is not significantly different for [accuracy / missing important information / including a lot of unnecessary information/ expressing what the method does / easiness of the structure].

TABLE VI. MANN-WHITNEY TEST FOR DOCUMENTATION WITHOUT LIST OF CALLS.

H	Ques.	Category	N	μ	SD	U	Z	P
1	Accur.	Full	18	85.9	9.42	59	-1.12	.26
		No list of calls	9	90.06	11.38			
2	Miss info	Full	18	30.5	22.21	76	-.232	.81
		No list of calls	9	30.27	12.52			
3	A lot of unnec. info	Full	18	11.3	10.79	47	-1.83	.06
		No list of calls	9	3.83	5.89			
4	What the method does	Full	18	85.8	12.61	69	-.60	.54
		No list of calls	9	82.50	15.05			
5	Structure	Full	18	81.15	18.8	70	-.57	.56
		No list of calls	9	86.64	8.88			

We cannot reject hypothesis (H1-H5) from Table VI. Therefore, for all the evaluation questions, we found no statistical significant difference between full documentation and documentation that omits the list of calls and stereotypes (see Table VI).

Of the documentation that omits the list of calls and their stereotypes, 3/9 are evaluated by the majority to miss important information. Most of the participants complained that calls are not explained in the documentation. In fact, a few of them asked why the stereotypes of calls are not included. These observations suggest that participants value the inclusion of the list of calls and their stereotypes.

No data members read. A total of 75 records are collected for 9 documentation cases without list of data members, and are compared to the ratings of 11 of the 18 full documentation cases. Seven methods are excluded from this analysis as they are controller methods that by definition have no data members.

The ratings of documentation that omits data members read are slightly improved in three evaluation questions. This improvement appears in case of missing important information, what the method does, and structure by 6%, 10%, and 11%, respectively. We do not have enough evidence to explain why such improvement occurs. It could be that shorter documentation (one line shorter) happens to be more preferable.

Mann-Whitney Test is applied for the five evaluation questions to answer the following hypothesis:

H_n: the difference between the rating of full documentation and documentation without the list of data members read is not

significantly different for [accuracy / missing important information / including a lot of unnecessary information/ expressing what the method does / easiness of the structure].

We cannot reject hypothesis (H1-H5) from Table VII; therefore, no statistical difference is found in all of the evaluation questions. Consequently, we conclude that the absence of the list of data members read is not as critical.

No list of objects. A total of 75 records are collected for 10 documentation cases without list of objects, and are compared to the ratings of 9 of the 18 full documentation cases of *collaborational* methods. Nine methods that are *non-collaborational* are excluded from this analysis because by definition they do not have the list of objects.

TABLE VII. MANN-WHITNEY TEST FOR DOCUMENTATION WITHOUT LIST OF DATA MEMBER READ.

H	Evaluation question	Category	N	μ	SD	U	Z	P
1	Accuracy	Full	11	88.7	10.23	49	.00	1.0
		No list of DM	9	88.6	10.29			
2	Miss info	Full	11	26.29	21.53	46	-.229	.82
		No list of DM	9	21.53	12.72			
3	A lot of unnec. info	Full	11	9.49	10.98	45	-.359	.72
		No list of DM	9	11.92	33.33			
4	What the method does	Full	11	83.11	13.46	30	-1.47	.14
		No list of DM	9	92.11	8.08			
5	Structure	Full	11	80.5	19.40	35.5	-1.09	.27
		No list of DM	9	90.67	10.37			

TABLE VIII. MANN-WHITNEY TEST FOR DOCUMENTATION WITHOUT LIST OF OBJECTS.

H	Evaluation question	Category	N	μ	SD	U	Z	P
1	Accuracy	Full	9	84.54	8.5	31	-1.15	.24
		No list of objs	10	80.72	8.17			
2	Miss info	Full	9	35.93	21.31	36	-.695	.48
		No list of objs	10	42.1	23.3			
3	A lot of unnec. info	Full	9	13.26	9.4	40	-.374	.70
		No list of objs	10	11.7	13.88			
4	What the method does	Full	9	88.9	10.1	41	-.297	.76
		No list of objs	10	88.28	18.04			
5	Structure	Full	9	80.3	17.54	34	-.88	.37
		No list of objs	10	84.1	21.85			

Except of "missing of important information", none of the evaluation questions seem to be affected by the absence of list of objects. Compare to full documentation, the ratings of documentation that does not include list of objects increase by 5% in case of missing important information.

Mann-Whitney Test is applied for the five evaluation questions to answer the following hypothesis:

H_n: the difference between the rating of full documentation and documentation without the list of objects is not significantly different for [accuracy / missing important information / including a lot of unnecessary information/ expressing what the method does / easiness of the structure].

We cannot reject hypothesis (H1-H5) in Table VIII. Therefore, there is no statistical difference is found in all the evaluation questions. Consequently, we conclude that the absence of the list of objects is not as critical.

The average ratings of documentation summaries may slightly improve or decrease when one section is eliminated. These changes may be related to participants' preferences of the content to be included. However, in the case of removing the short description, the accuracy and missing important information questions statistically decreased compared to the full documentation. Therefore, RQ4 is answered as the following:

Based on the statistical analysis, the most important section of the documentation is "short description".

D. Threats to Validity

As the design and procedure of the second study is similar to the first study, the same threats to validity can be encountered. These issues are discussed in section VD.

When documentation that eliminated one section is compared to full documentation, we observe small reduction or improvement in participants' ratings. These small changes may not be strong evidence of the importance of a section. To mitigate this threat, we use the Mann-Whitney Test to study if the difference is statistically significant.

VIII. CONCLUSION

This paper presents two evaluation studies of an automatic approach for generating documentation for C++ methods. The goal of the first study is to evaluate the effectiveness of using method stereotypes to solve the summarization problem. The results indicated that the automatically-generated documentation accurately expresses what the method does. While the majority of the participants positively rate the automatically-generated documentation, some improvements regarding including conditional statements are needed.

We enhance the approach using static analysis of control-flow statements of the main action of the method. According to our second evaluation study, this improvement increased the accuracy and content adequacy of the documentation. The second evaluation also studies the necessity of each section of the automatically generated documentation. Based on the analyzed ratings from 75 programmers, we conclude that the most important part of the automatically generated documentation is the short description, as their absence significantly reduce the documentation ratings. In addition, removing the list of data members read or the list of calls and their stereotypes seems to be more preferable for some of participants. Therefore, we feel that allowing programmers to choose the desired level of details of the documentation is the best approach.

The results from the two evaluation studies demonstrate that method stereotypes can directly support the process of automatically generating useful method documentation in an efficient manner. The approach also demonstrates that a template-based technique, specialized for each stereotype, combined with simple static analysis is a practical approach for automatically generated documentation.

ACKNOWLEDGMENTS

This work was supported in part by a grant from the National Science Foundation CNS 13-05292/05217. We thank Dr. Bonita Sharif, Dale Haverstock, Dr. Gregory DeLozier, and Michael Decker for their assistance in the surveys.

References

[1] 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, 2006, 32, (12)pp. 971-987.

[2] T. D. LaToza, G. Venolia and R. DeLine: "Maintaining mental models: a study of developer work habits",28th International Conference on Software Engineering (ICSE), New York, NY, May 20-28, 2006, pp. 492-501.

[3] B. Fluri, M. Wursch and H. C. Gall: "Do code and comments co-evolve? on the relation between source code and comment changes",14th Working Conference on Reverse Engineering (WCRE), Washington, DC, USA, October 28-31, 2007, pp. 70-79.

[4] S. Haiduc, J. Aponte, L. Moreno and A. Marcus: "On the Use of Automated Text Summarization Techniques for Summarizing Source Code",17th Working Conference on Reverse Engineering (WCRE), Beverly, MA, October 13-16 2010, pp. 35-44.

[5] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock and K. Vijay-Shanker: "Towards Automatically Generating Summary Comments for Java Methods",IEEE/ACM International Conference on Automated Software (ASE), Antwerp, Belgium, September 20-24 2010, pp. 43-52.

[6] G. Sridhara, L. Pollock and K. Vijay-Shanker: "Automatically Detecting and Describing High Level Actions within Methods",33rd International Conference on Software Engineering (ICSE), Honolulu, HI, USA, May 21-28 2011, pp. 101-110.

[7] B. P. Eddy, J. A. Robinson, N. A. Kraft and J. C. Carver: "Evaluating Source code Summarization Techniques: Replication and Expansion",21st International Conference on Program Comprehension (ICPC), San Francisco, CA, USA, May 20-21, 2013, pp. 13-22.

[8] E. Wong, J. Yang and L. Tan: "AutoComment: Mining Question and Answer Sites for Automatic Comment Generation",28th International Conference on Automated Software Engineering (ASE), Palo Alto, USA, November 11-15, 2013, pp. 577-582.

[9] P. McBurney and C. McMillan: "Automatic Documentation Generation via Source code Summarization of Method Context",22nd International Conference on Program Comprehension (ICPC), Hyderabad, India, June 2–3, 2014, pp. 279-290.

[10] L. Moreno and J. Aponte: "On the Analysis of Human and Automatic Summaries of Source Code",CLEI Electronic Journal, 2012, 15, (2)pp.

[11] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock and K. Vijay-Shanker: "Automatic generation of natural language summaries for Java classes",21st International Conference on Program Comprehension (ICPC), San Francisco, CA, USA, May 20-21, 2013, pp. 23-32.

[12] N. J. Abid, N. Dragan, M. L. Collard and J. I. Maletic: "Using Stereotypes in the Automatic Generation of Natural Language Summaries for C++ Methods",Software Maintenance and Evolution (ICSME), Bremen, Germany, Sept. 29-Oct. 1 2015, pp. 561 - 565.

[13] X. Zou, R. Settimi and J. Cleland-Huang: "Improving automated requirements trace retrieval: A study of termbased enhancement methods",Empirical Softw. Engg., 2010, 15, (2)pp. 119-146.

[14] G. Sridhara, L. Pollock and K. Vijay-Shanker: "Generating Parameter Comments and Integrating with Method Summaries",19th International Conference on Program Comprehension (ICPC), Kingston, Ontario, Canada, June 22-24, 2011, pp. 71 - 80.

[15] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch and S. D'Mello: "Improving Automated Source code Summarization via an Eye-Tracking Study of Programmers",36th International Conference on Software Engineering (ICSE), India, May 31- June 7, 2014, pp. 460-471.

[16] C. Vassallo, S. Panichella, M. Di Penta and G. Canfora: "CODES: mining source code Descriptions from developers discussions",22nd International Conference on Program Comprehension (ICPC), Hyderabad, India, June 2-3, 2014, pp. 106-109.

[17] X. Wang, L. Pollock and K. Vijay-Shanker: "Developing a Model of Loop Actions by Mining Loop Characteristics from a Large Code Corpus ",31st International Conference on Software Maintenance and Evolution (ICSME) Bermen, Germany, Sep 29 - Oct 1, 2015, pp. 51-60.

[18] A. T. T. Ying and M. P. Robillard: "Selection and Presentation Practices for Code Example Summarization",22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering, Hong Kong, China, November 16–21, 2014, pp. 460-471.

[19] H. Burden and R. Heldal: "Natural Language Generation from Class Diagrams",8th International Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa), New York, NY, USA, October 17, 2011.

[20] S. Rastkar, G. C. Murphy and A. W. J. Bradley: "Generating Natural Language Summaries for Crosscutting Source Code Concerns", 27th IEEE International Conference on Software Maintenance (ICSM), Williamsburg, VA, September 25-30, 2011, pp. 103-112.

[21] S. Rastkar, G. C. Murphy and G. Murray: "Summarizing Software Artifacts: A Case Study of Bug Reports",32nd International Conference on Software Engineering (ICSE), Cape Town, South Africa, May 2-8 2010, pp. 505-514.

[22] E. Hill, D. Muppaneni, L. Pollock and K. Vijay-Shanker: "Automatically Capturing Source Code Context of NL-Queries for Software Maintenance and Reuse",31st International Conference on Software Engineering (ICSE), Vancouver, British Columbia, Canada, May 16-24, 2009, pp. 232 - 242.

[23] P. W. McBurney and C. McMillan: "An Empirical Study of the Textual Similarity between Source Code and Source Code Summaries",Empirical Software Engineering, 2016, 21, (1)pp. 17-42.

[24] N. Dragan, M. L. Collard and J. I. Maletic: "Automatic Identification of Class Stereotypes",IEEE International Conference on Software Maintenance (ICSM), Timisoara, Romania, September 12-18 2010, pp. 1 -10.

[25] P. Rodeghero, C. Liu, P. McBurney and C. McMillan: "An Eye-Tracking Study of Java Programmers and Application to Source Code Summarization",IEEE Trans. Software Eng., 2015, 41, (11)pp. 1038-1054.

[26] P. Rodeghero and C. McMillan: "An Empirical Study on the Patterns of Eye Movement during Summarization",International Symposium on Empirical Software Engineering and Measurement (ESEM) Beijing, China, October 22-23, 2015, pp. 11-20.

[27] N. Dragan, M. L. Collard and J. I. Maletic: "Reverse Engineering Method Stereotypes",22nd IEEE International Conference on Software Maintenance (ICSM), September 24-27, 2006, pp. 24 - 34.

[28] C. D. Newman, R. S. AlSuhaibani, M. L. Collard and J. I. Maletic: "Lexical Categories for Source Code Identifiers",the 24 IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER), Klagenfurt, Austria, 2017

[29] H. B. Mann and D. R. Whitney: "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other",Annals of Mathematical Statistics, 1947, 18, (1)pp. 50-60.