Automatically Generating Natural Language Documentation for Methods

Christian D. Newman
Rochester Institute of Tech
New York, USA
cnewman@se.rit.edu

Natalia Dragan
Kent State University
Ohio, USA
ndragan@kent.edu

Michael L. Collard
The University of Akron
Ohio, USA
collard@uakron.edu

Jonathan I. Maletic
Kent State University
Ohio, USA
jmaletic@kent.edu

Michael J. Decker
Bowling Green State University
Ohio, USA
mdecke@bgsu.edu

Drew T. Guarnera
Kent State University
Ohio, USA
dguarner@kent.edu

Nahla Abid
Taibah University
Saudi Arbia
nabid@kent.edu

Abstract— **A tool to automatically generate natural language documentation summaries for methods is presented. The approach uses prior work by the authors on stereotyping methods along with the source code analysis framework srcML. 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 for each method. This summary is automatically added to the code base as a comment for each method. The predefined templates are designed to produce a generic summary for specific method stereotypes.**

Keywords—Documentation, Stereotype, Method Summarization

I. Introduction

There are many techniques that automatically generate summaries/documentation directly from source code [1][2][3][4][5][6][7][8]. While these approaches can produce high-quality results, their performance depends upon high-quality identifiers and method names due to reliance on Natural Language Processing (NLP) techniques, which are not always reliable on source code [12], [13]. Consequently, if identifiers and methods are poorly named, the approach may fail to generate accurate comments or any reasonable comments at all.

Recently, Abid et al. [9][10] proposed an approach that uses stereotypes [11] in the automatic generation of documentation for methods. This approach does not depend on NLP at all and is the one we will apply in this work. Instead, the summaries (for methods) are generated using predefined fill-in-the-blank sentence phrases, which are known as templates. We constructed the templates specifically for different method stereotypes. Stereotypes reflect the basic meaning and behavior of a method and include concepts such as *predicate*, *set*, and *factory*. In previous work by the authors on method stereotypes, a fully automated approach to assign stereotypes to methods was developed and evaluated [11].

Our tool, *MethodMan*; short for Method Manual, leverages stereotype information and custom template phrases. We constructed a custom template phrase for each stereotype to form the summaries. After the appropriate templates are matched to the method, they are filled with data to form the complete summary. To fill in the templates, static analysis and fact extraction on the method is done using the srcML [14][15] infrastructure (www.srcML.org).

The generation of the summaries is fully automated. The summaries start with a short and precise description of the main responsibility of the method. Also included is additional information about external objects, properties modified, and a list of function calls along with their stereotypes.

II. Data Sources Used

The approach uses only the source code file(s). However, it also leverages srcML and the stereotype information. srcML can accurately parse a single file or code fragments. As such the entire system is not required to compile to produce documentation for one class or method.

III. Approach

MethodMan takes source code as input and for each method automatically generates a documentation summary that is placed as a comment block above the method. An example of the automatically generated documentation summary for one method is presented in Fig. 1. The member function `calcWidthParm()` is in the class `BinAxsLinear` from the open-source system HippoDraw. The summary starts off with a short description, (e.g., the first three lines in Fig. 1). This emphasizes the computed value along with the data members and parameters used in the computation. Following that is a list of calls made from this method along with the corresponding stereotypes of the calls.

To automatically construct a summary, two issues must be addressed. The first is determining what information should be included in the summary. The next is to present this information efficiently. Previous studies [7][8] on source-code summarization have investigated the former issue in depth and this information is used as a set of guidelines for building our automatic source-code summarization tool. First, the method summary should include at least a verb (i.e., describes the action performed by the method) and an object (i.e., describes on which the action is performed). Second, the name of the method is considered to be the most useful part of a method to use in the summary. Moreover, full identifiers are preferable to split identifiers, as they assist in mapping the summary to the source code. Calls within the method often produce side effects and the recommendation is to include them in the summary. Finally, local variables do not provide very useful information, and should not be included.

```
/**
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

```
 /**
 1 <Short Description>
 2 Collaborates with- <obj_1>, ... <obj_m>
 3 Data members or parameters modified- <name_1>, ... <name_p>
 4 Calls to-  <call_1>        <stereotype_1>
                        …
            <call_n>        <stereotype_n>
  */
```

Fig. 2. The template of the four sections of the documentation summary.

The template for the four sections of the documentation summary is shown in Fig. 2. If a section is not applicable, it is omitted. For example, the automated summary of the *property* method in Fig. 1 omits lines 2 and 3. The individual templates for each section of the summarization are explained below.

The first section of the documentation summary (line 1 in Fig. 2) is an English description that explains the main responsibility of the method in terms of the stereotype(s). The template for the short description is composed of two phrases. The first phrase is standardized for all stereotypes while the second phrase is customized for each specific stereotype. The first phrase of the template for the short description presents the method stereotype in the following form:

```
<method> is a <stereotype> [that collaborates with <obj>]
```

The second part in the above template is optional and is used if the secondary stereotype of the method is *collaborator* or *controller*. For example, the generated summary of the method `validate()` that works with the object `Range` gives:

```
validate is a void-accessor that collaborates with Range
```

We avoid long lists of objects. Therefore, if the method is collaborating with more than one object, then the number of objects is included in the short description and the list of names is moved to the second part of the summary (line 2 Fig. 2). The second phrase in the short description explains the main actions of the method. The particular template used depends on the stereotype of the method. We have developed specialized templates for all the different stereotypes. These are detailed in the original works by abid et al [9][10].

IV. Results

The results for the class XSSFWorkbook are available at http://www.sdml.cs.kent.edu/dysdoc3/methodman/.

MethodMan is quite fast and runs on a class in a few seconds (including running srcML and computing the stereotypes). Generating srcML and computing the stereotypes for the ***entire*** Apache POI project takes less than 2 minutes.

References

[1] 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, 2010, pp. 35-44.

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

[3] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards Automatically Generating Summary Comments for Java Methods" the International Conference on Automated Software (ASE), Antwerp, Belgium, 2010, pp. 43–52.

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

[5] E. Wong, J. Yang, and L. Tan, "AutoComment: Mining Question and Answer Sites for Automatic Comment Generation", the nternational Conference on Automated Software Engineering (ASE), Palo Alto, CA, USA, 2013, pp. 577–582.

[6] P. McBurney and C. McMillan, "Automatic Documentation Generation via Source code Summarization of Method Context", the International Conference on Program Comprehension (ICPC), Hyderabad, India, 2014, pp. 279–290.

[7] L. Moreno and J. Aponte, "On the Analysis of Human and Automatic Summaries of Source Code", *CLEI Electron. J.*, vol. 15, no. 2, 2012.

[8] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for Java classes", the nternational Conference on Program Comprehension (ICPC), San Francisco, CA, USA, 2013, pp. 23–32.

[9] 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", the IEEE International Conference on Software Maintenance and Evolution (ICSME), 2015, pp. 561–565.

[10] N. J. Abid, N. Dragan, M. L. Collard, and J. I. Maletic, "The Evaluation of an Approach for Automatic Generated Documentation", the International Conference on Software Maintenance & Evolution (ICSME), Shanghai, China, 2017, p. 11 pages.

[11] N. Dragan, M. L. Collard, and J. I. Maletic, "Reverse Engineering Method Stereotypes", the 22nd IEEE International Conference on Software Maintenance (ICSM'06), 2006, pp. 24–34.

[12] W. Olney, E. Hill, C. Thurber, and B. Lemma, "Part of Speech Tagging Java Method Names", *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 483–487.

[13] D. Binkley, D. Lawrie, and C. Morrell, "The need for software specific natural language techniques", *Empir. Softw. Eng.*, vol. 23, no. 4, pp. 2398–2425, Aug. 2018.

[14] M. L. Collard, M. Decker, and J. I. Maletic, "srcML: An Infrastructure for the Exploration, Analysis, and Manipulation of Source Code", the 29th IEEE International Conference on Software Maintenance (ICSM'13) Tool Demonstration Track, 2013, pp. 1–4.

[15] M. L. Collard, M. J. Decker, and J. I. Maletic, "Lightweight Transformation and Fact Extraction with the srcML Toolkit", 11th IEEE International Conference on Source Code Analysis and Manipulation, 2011, pp. 173–184.