Automatically Redocumenting Source Code with Method and Class Stereotypes

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

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

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

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

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

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

Abstract—The tool implements an approach that automatically derives and redocuments source code with the corresponding method and class stereotypes. The stereotype of each method is first computed via static analysis and a set of definitions. Then the class stereotype is computed based on the distribution of method stereotypes. The approach is fully automatic and highly scalable. It uses the srcML infrastructure to do the analysis and insertion of the stereotype information into the code.

Keywords—Documentation, Method & Class Stereotype

I. INTRODUCTION

The automated documentation approach presented here concerns the labeling of methods and classes with a stereotype [1][2]. A stereotype is a concise description of the basic behavioral aspects of a method or class. A simple example of a method stereotype is a getter or setter. Methods either get or set information about the state of a class. Well-known class stereotypes include boundary, control and entity. Stereotypes have been used to support a number of program comprehension and documentation activities [3][4][5][6][7][8][9][10][11][12]. They form the basis for such things as constructing natural language method [5] and class [13] summaries.

While stereotypes are a simple form of documentation, they form the basis for further generation of more detailed and sophisticated documentation. The automated labeling and re-documentation of source code with accurate stereotype information is thus very important prerequisite for many approaches to documentation generation.

Here we present our tool, StereoCode, which does just that. It takes existing source code and re-documents all methods and classes with one or more predefined stereotypes. This information is added as a comment (e.g., JavaDoc) to the code. StereoCode is built using the srcML [14][15] infrastructure (srcML.org) and supports both the Java and C++ programming languages. Additionally, StereoCode produces summary information for the class in the form of a histogram of the method stereotypes contained in the class shown in Fig. 1. This gives a quick overview of the makeup of the class.

Fig. 1. Example of method stereotype histogram

II. DATA SOURCES USED

The approach uses only the source code file(s). However, the source code need not be complete nor in a state that can compile. That is, srcML can accurately parse a single file or code fragments. As such, the entire system is not required to run StereoCode and produce stereotypes for a class or method.

III. APPROACH

Our approach starts by automatically identifying and labeling all methods in a class with their stereotype. An example of the stereotype labeling can be seen in Fig. 2. This information is then collected and a distribution of method stereotypes for the class is calculated. Class stereotypes are derived from this distribution via a set of rules that map method stereotype distribution characteristics to the class stereotype taxonomy.

Fig. 2. Example of redocumented method with stereotype.

We now provide a brief introduction to the method stereotypes followed by a description of the class stereotypes. The work of Dragan et al. [1] introduces a taxonomy for method stereotypes, which we show in TABLE I. A method stereotype concisely represents behavioral aspects of a method. Method stereotypes are separated into five broad categories: Structural Accessors – query the state of an object on which it is called; Structural Mutators – modify the state of an object on which it is called; Creationals – create/provide new objects;
Collaborational – work on objects pertaining to classes other than itself; and Degenerate – no use of the object’s state and often no statements. The individual stereotypes indicate a refinement of the broad behavior described by the category. As an example, stereotypes in the general category Structural Accessor query an object’s state, while the stereotype predicate more specifically returns a computed Boolean value. This Boolean characterizes some information about the state of the object on which the predicate method is called. Methods may be labeled with one or more stereotypes. That is, methods may have a single stereotype from any category and may also have additional stereotypes from other categories. For example, a get collaborator is a get (accessor) method that uses an object of another class (e.g., return type).

### Table I. Method Stereotype Taxonomy. Each Stereotype Category is Listed with its Set of Stereotypes.

<table>
<thead>
<tr>
<th>Stereotype Category</th>
<th>Stereotype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Accessor</td>
<td>get</td>
<td>Returns a data member.</td>
</tr>
<tr>
<td></td>
<td>predicate</td>
<td>Returns Boolean value that is not a data member.</td>
</tr>
<tr>
<td></td>
<td>property</td>
<td>Returns info about data members.</td>
</tr>
<tr>
<td></td>
<td>void-access</td>
<td>Returns information via a parameter.</td>
</tr>
<tr>
<td>Structural Mutator</td>
<td>set</td>
<td>Sets a data member.</td>
</tr>
<tr>
<td></td>
<td>command</td>
<td>Performs a complex change to the object’s state.</td>
</tr>
<tr>
<td></td>
<td>non-void-command</td>
<td></td>
</tr>
<tr>
<td>Creational</td>
<td>constructor</td>
<td>Creates and/or destroys objects.</td>
</tr>
<tr>
<td></td>
<td>copy-constructor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>destroyer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>factory</td>
<td></td>
</tr>
<tr>
<td>Collaborational</td>
<td>collaborator</td>
<td>Works with objects (parameter, local or return value).</td>
</tr>
<tr>
<td></td>
<td>controller</td>
<td>Changes only an external object’s state (not this).</td>
</tr>
<tr>
<td>Degenerate</td>
<td>incidental</td>
<td>Does not read/change the object’s state.</td>
</tr>
<tr>
<td></td>
<td>stateless</td>
<td>Does not read/change the object’s state.</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>Has no statements.</td>
</tr>
</tbody>
</table>

Briefly, the creation of the taxonomy for class stereotypes (e.g., entity, boundary, factory, controller, degenerate, small class) started with an empirical investigation that led to the formulation of the rules for the identification of each stereotype. The rules are based on the distribution of method stereotypes within that class. They were validated on open source systems that led to the rules refinement and further validations of the class’s taxonomy. The actual class names are not used in the categorization. While the name can be a good source of information, it can also be misleading; an aspect of the investigation we leave for future work. We refer readers to Dragan [2] for a complete discussion of class stereotypes. StereoCode is freely available (i.e., licensed under GPL) as part of the srcML infrastructure (see [www.srcML.org](http://www.srcML.org)).

### IV. RESULTS

The results for the class XSSFWorkbook are available at [http://www.sdml.cs.kent.edu/dysdoc3/stereotypes/](http://www.sdml.cs.kent.edu/dysdoc3/stereotypes/). At the top is a visualization of the method stereotype distribution. For each method, the method comment is redocumented with the method stereotype. The class comment is redocumented with the class stereotype, a brief description, and a text form of the method stereotypes distribution.

**StereoCode** is extremely fast on a single file (less than one second). To redocument the entire Apache POI project, using **StereoCode**, takes under 2 minutes on a normal laptop. This includes generating srcML (~1.5 minutes).

### REFERENCES


