Reverse Engineering Method Stereotypes

Natalia Dragan
Michael L. Collard
Jonathan I. Maletic
Background

- Stereotypes are a mechanism (in UML) to extend and define semantics
- Simple documentation technique for methods
  - Role and semantics of the method
  - Common vocabulary (well known terms)
  - Predicate, mutator, accessor

- However, method stereotypes are rarely explicitly documented by developers
Motivation

- Support comprehension of method/class
- Changes to a method that change its stereotype is an indication of design change
- LCOM metrics are biased by certain types of methods (accessor & constructors)
- Good object abstraction requires good method abstraction
- Class stereotypes (e.g., boundary, control, entity) are based on the methods
Design Recovery

- Reverse engineering method stereotypes supports higher-level design recovery of object-oriented systems

- Little research has been conducted on method stereotypes in the context of reverse engineering or automatic identification
Objective

- Define a taxonomy of method stereotypes
- Automatically identify a method’s stereotype
- Re-document the method (annotate with a comment in the source)
- Scalable to large systems
Developing a Taxonomy

- Examined previous classifications
- Little work on the topic
- No formal in-depth studies
- Forward engineering view point
Research on Stereotypes

- **Method classifications with respect to the internal state of objects at the design level** [Fowler00], [Arevalo03]
  - Focused on how a method accesses its data members rather than a view on the primary purpose of the method

- **Method categorization with respect to collaborations between methods** [Lanza01], [Clarke03]
  - Limited to generalization relationships

- **Method stereotypes for assisting in program development of C++ and Java** [Workman02], [Riehle01]
  - No means for detection are given
Method Stereotypes

○ Behavioral characteristic
  ● Structural
    • Provide/support the structure of the class
  ● Collaborational
    • Communication between different objects
  ● Creational
    • Create/destroy objects

○ Type of data access to data members
  ● Read (accessors)
  ● Write (mutators)
### Stereotype Taxonomy

<table>
<thead>
<tr>
<th></th>
<th>Structural</th>
<th>Collaborational</th>
<th>Creational</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accessor</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mutator</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td></td>
<td></td>
<td>Constructor</td>
</tr>
<tr>
<td><strong>Predicate</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td></td>
<td></td>
<td>Copy Constructor</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Destructor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Factory</td>
</tr>
</tbody>
</table>
Structural Methods: Accessors

- A **get** method is an accessor that returns the value of a data member
  
  ```
  const string& getName() const;
  ```

- A **predicate** method is an accessor that returns a Boolean result
  
  ```
  bool isEmpty() const;
  ```

- A **property** method is an accessor that returns information about an object based on data member values
  
  ```
  int indexOfMinElement (int index) const;
  ```
Structural Methods: Mutators

- A **set** method is a mutator that changes the value of a data member
  
  ```
  void setName(const string& name);
  ```

- A **command** is a mutator that executes a complex change of the object’s state
  
  - The change may involve several data members
  
  - May change the data members either directly or indirectly using another mutator
    
    ```
    void draw(int x, int y);
    ```
Collaborational Methods

- A collaborator is a method that works other objects (different from its own class)
  - Parameters that are objects
  - Local objects
  - Objects accessed indirectly through a data member (e.g., a vector of object pointers)

```cpp
bool hasControlPoints
(const PlotterBase*plotter) const;
```
Creational Methods: Factory

A **factory** method is one that creates an object and returns it to the client (object creation method)

- Factory methods work outside of the class and change the state of the external objects with which they have relations

```cpp
PlotterBase* createDisplay
    (const string& name);
```
Approach

Source Code (Original) → srcML → StereoCode → XPath → XSLT → Source Code (annotated)
Implementation for C++

- Need to analyze:
  - Access to data members
  - Return type
  - Parameter type
  - Const-ness
  - Type of local variables
Example

STARTING POINT

```cpp
class DataSource
{
private:
    string m_ds_name;
    vector<string> m_labels;
public:
    const string& getName() const;
    bool isValidLabel(const string& label) const;
    void setLabels(const vector<string>& v);
};
```

RESULT

```cpp
class DataSource : public Observable
{
private:
    string m_ds_name;
    vector<string> m_labels;
public:
    /** @stereotype get */
    const string& getName() const;

    /** @stereotype predicate */
    bool isValidLabel(const string& label) const;

    /** @stereotype set */
    void setLabels(const vector<string>& v);
};
```
Evaluation

- Tools for stereotypes detection were applied to HippoDraw and Qt

- Each method was automatically labeled with a stereotype and the original source code was re-documented by our tools

- To assess the approach and the tools an experienced developer rated the automatically identified annotations in HippoDraw
HippoDraw and Qt

- HippoDraw - open-source application providing a data-analysis environment
  - 60 KLOC of source code in over 400 C++ files (200 classes with 2900 methods & free functions)

- Qt - cross-platform C++ GUI framework
  - 4.1.2. version - about 1000 KLOC (over 1000 classes with 20900 methods)

- The source for both is well written and follows a consistent object oriented style
# Method Classifications

<table>
<thead>
<tr>
<th>Stereotype(s)</th>
<th>Occurrences</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hippo</td>
<td>Qt</td>
</tr>
<tr>
<td>Number of methods labeled with only one stereotype</td>
<td>1357</td>
<td>6410</td>
</tr>
<tr>
<td>Number of methods labeled with two stereotypes</td>
<td>1099</td>
<td>14067</td>
</tr>
<tr>
<td>unclassified</td>
<td>220</td>
<td>386</td>
</tr>
<tr>
<td>empty_method</td>
<td>30</td>
<td>8</td>
</tr>
<tr>
<td>Overall Total</td>
<td>2706</td>
<td>20869</td>
</tr>
</tbody>
</table>
# Single Stereotypes

<table>
<thead>
<tr>
<th>Stereotype(s)</th>
<th>Occurrences</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hippo</td>
<td>Qt</td>
</tr>
<tr>
<td>command</td>
<td>439</td>
<td>1281</td>
</tr>
<tr>
<td>property</td>
<td>361</td>
<td>1098</td>
</tr>
<tr>
<td>collaborator</td>
<td>239</td>
<td>3707</td>
</tr>
<tr>
<td>get</td>
<td>133</td>
<td>109</td>
</tr>
<tr>
<td>predicate</td>
<td>99</td>
<td>54</td>
</tr>
<tr>
<td>set</td>
<td>84</td>
<td>161</td>
</tr>
<tr>
<td>factory</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>Number of methods labeled with only one stereotype</strong></td>
<td><strong>1357</strong></td>
<td><strong>6410</strong></td>
</tr>
</tbody>
</table>
## Multiple Stereotypes

<table>
<thead>
<tr>
<th>Stereotype(s)</th>
<th>Occurrences</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hippo</td>
<td>Qt</td>
</tr>
<tr>
<td>collaborator,command</td>
<td>623</td>
<td>8546</td>
</tr>
<tr>
<td>collaborator,factory</td>
<td>296</td>
<td>889</td>
</tr>
<tr>
<td>collaborator,property</td>
<td>90</td>
<td>2806</td>
</tr>
<tr>
<td>collaborator,set</td>
<td>30</td>
<td>819</td>
</tr>
<tr>
<td>collaborator,predicate</td>
<td>23</td>
<td>471</td>
</tr>
<tr>
<td>collaborator,get</td>
<td>22</td>
<td>378</td>
</tr>
<tr>
<td>collaborator,empty_method</td>
<td>14</td>
<td>156</td>
</tr>
<tr>
<td>property,empty_method</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Number of methods labeled with two stereotypes</td>
<td>1099</td>
<td>14067</td>
</tr>
</tbody>
</table>
Assessment Methodology

- A subset of 19 classes and 365 methods (~14% of the system) with a wide diversity of stereotypes was selected from Hippodraw

- A subject (an experienced developer) was given the taxonomy with the definitions of each stereotype

- Rating of a method’s stereotype on a Likert scale: Very Good, Good, Fair, and Poor
## Subject Assessment

<table>
<thead>
<tr>
<th>Hippodraw</th>
<th>Very good</th>
<th>Good</th>
<th>Fair</th>
<th>Poor</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject’s Assessment</td>
<td>289 (79%)</td>
<td>40 (11%)</td>
<td>15 (4%)</td>
<td>21 (6%)</td>
<td>365 (100%)</td>
</tr>
<tr>
<td>Errors due to poor design of methods</td>
<td>1</td>
<td></td>
<td>5</td>
<td></td>
<td>6 (1.6%)</td>
</tr>
<tr>
<td>Errors due to incomplete analysis</td>
<td>19</td>
<td></td>
<td>5</td>
<td></td>
<td>24 (6.6%)</td>
</tr>
<tr>
<td>Errors due to differences in interpretation</td>
<td>7</td>
<td></td>
<td>5</td>
<td>11</td>
<td>23 (6.3%)</td>
</tr>
</tbody>
</table>

- **Total**
  - 90%
  - 10%
Scalability

- Translating to srcML
  - ~15 seconds for HippoDraw
  - ~3 minutes for Qt
- The identification and annotation
  - ~30 seconds for HippoDraw
  - ~6 minutes for Qt
- Converting back to raw C++
  - ~2 seconds for HippoDraw
  - ~30 seconds for Qt
Contributions

- Defined a taxonomy of method stereotypes and rules for the automatic identification in C++
- Developed a tool, *StereoCode*, to automatically identify the stereotypes and re-document source code
- Evaluation servers as a benchmark for further studies and investigations
Conclusions

- Our stereotype classification along with the tool StereoCode for automatically identifying and re-documenting method stereotypes is sound and efficient.
- Our results were very good as an experienced developer agreed 90% of the time with our classification.
- StereoCode tool, based on a lightweight static program analysis approach, is efficient and usable.
Future work

- Construct design-quality metrics based on stereotype classification

- Extend this approach to automatically reverse engineer class stereotypes