Modeling the Relationship Between Identifier Name and Behavior

SCANL
Source Code Analysis And
Natural Language Laboratory

Hello, we are SCANL Lab!

We study the latent connection between source code behavior and the natural language elements used to describe that behavior

Members of the lab on this paper:

Christian D. Newman, Reem S. Alsuhaibani, Anthony Peruma

For more information on our tools, datasets, and research:

https://scanl.org/

Topic for today

- Identifier Names
  - Everywhere in source code
  - Atoms of comprehension
    - Semantics and Structure
- Name should concisely summarize role of correlating entity

For Example

```
public class ResultsWriter {

    private String outputFile;
    private FileWriter writer;

    public ResultsWriter() throws IOException {
        String time =  String.valueOf(Calendar.getInstance().getTimeInMillis());
        outputFile = MessageFormat.format("Output_{0}.csv", time);
        writer = new FileWriter(outputFile,false);
    }

    public void writeOutput(List<String> dataValues) throws IOException {
        writer = new FileWriter(outputFile,true);

        int i;
        for (i=0; i<dataValues.size(); i++) {
            writer.append(String.valueOf(dataValues.get(i)));

            if(i!=dataValues.size()-1)
                writer.append(",");
            else
                writer.append(System.lineSeparator());

        }
        writer.flush();
        writer.close();
    }
}
```

Sure, so what?

- Identifiers make up 70% of the code [Deissenboeck]
- Devs spend ~half their time on comprehension [Corbi]
  - Low quality identifiers increase this time by19+% [Hofmeister]
  - Low quality identifiers cause bugs and poor code quality [Butler]
- Automated approaches to analyze text in source code rely on good names
  - AI and ML approaches which may help in the IDE suffer [Jongeling]
    - Code generation: Generate maintainable, correct changes
  - Information retrieval based approaches suffer [Binkley]
  - Auto-generated documentation will never fully mature

What is a good name?

- Name should concisely summarize role of correlating entity
  - So the natural language and source code behavior must be synergistic
  - Cannot optimize a name without understanding code behavior, structure, and natural language synergy

- We need a model of this synergy

Two parts

- Natural language semantics
- Program semantics

...no wait three

- *One Ring to rule them all, One Ring to find them, One Ring to bring them all, and in the darkness bind them*

Grammar Patterns

- Identifiers are often a set of multiple terms
  - GetXMLHandler
- Terms together are more akin to a small sentence than a single word or concept
- Natural Language Part of Speech
  - Identify the role of a word within a sentence
    - Verb, noun, preposition, etc.
- Use NLP to study identifiers

Example - Grammar Ptrns

Grammar Pattern	Example Identifier
Verb-Adjective-Noun	GetParserHandler
Adjective-Noun	BitMask
Adjective-Adjective-Adjective-Noun	InPlaceSortHandler

Static Analysis

- Grammar patterns are the natural language data. Now we need to combine this with program semantics data
  - Recall that we cannot judge a name without understanding both natural language and code behavior
  - What source code behavior should we relate with natural language?
- Lexical categories [Newman]
  - Empirically derived taxonomy that categorizes identifiers based on their type information
- Stereotypes [Dragan]
  - Empirically derived taxonomy that categorizes methods based on their role

Lexical Categories

Lexical Categories for Source Code Identifiers

Christian D. Newman
Computer Science
Kent State University
Kent, OH 44240 USA
cnewman@kent.edu

Reem S. AlSuhaibani
Computer Science
Kent State University
Kent, OH 44240 USA
ralsuhai@kent.edu

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

Jonathan I. Maletic
Computer Science
Kent State University
Kent, Ohio, 44240 USA
jmaletic@kent.edu

***Abstract*—A set of lexical categories, analogous to part-of-speech categories for English prose, is defined for source-code identifiers. The lexical category for an identifier is determined from its declaration in the source code, syntactic meaning in the programming language, and static program analysis. Current techniques for assigning lexical categories to identifiers use natural-language part-of-speech taggers. However, these NLP approaches assign lexical tags based on how terms are used in English prose. The approach taken here differs in that it uses only source code to determine the lexical category. The approach assigns a lexical category to each identifier and stores this information along with each declaration. srcML is used as the infrastructure to implement the approach and so the lexical information is stored directly in the srcML markup as an additional XML element for each identifier. These lexical-category annotations can then be later used by tools that automatically generate such things as code summarization or documentation. The approach is applied to 50 open source**

Part of speech tagging is a method of categorizing words with similar grammatical properties. It has been used in a number of software maintenance tasks including identifier naming [5], code summarization [6, 7], reformulation of source-code queries [8], concept location [9], traceability-link recovery [10], and bug fixing [11]. In all of these, the underlying need for PoS is to provide information about what the word(s) that make up the identifier mean in English, so that this meaning can be correlated with the identifier's use in code. The assumption made is that the words that make up the identifier contain information that is relevant to the identifier's meaning within its source-code context. PoS tagging is beneficial to these approaches because it is able to compute a portion of the meaning of a word and this meaning partially correlates to how the word is used in code as part of an identifier. Hence, PoS tagging provides useful information to

S-lexical categories sample

Definition s-adjectives: s-adjectives, like adjectives in English, modify (i.e. describe) an s-noun or proper s-noun. Their role in the system is to provide ways to check (i.e. describe) system state such as the size of a container, the current color of a texture, an employee's given name, etc.

Definition s-pronouns: s-pronouns, like pronouns in English, can reference different entities depending on context (e.g., the last thing they were assigned to). Their role is to allow the system the flexibility to reference or keep track of entities which have no name or whose name has fallen out of scope

Method Stereotypes

Reverse Engineering Method Stereotypes

Natalia Dragan, Michael L. Collard, Jonathan I. Maletic
Department of Computer Science
Kent State University
Kent Ohio 44242
{ndragan, collard, jmaletic}@cs.kent.edu

Abstract

An approach to automatically identify the stereotypes of all the methods in an entire system is presented. A taxonomy for object-oriented class method stereotypes is given that unifies and extends the existing literature to address gaps and deficiencies. Based on this taxonomy, a set of definitions is given and method stereotypes are reverse engineered using lightweight static program

deal with change can also be envisioned. Changes in a method's stereotype due to modification may indicate major design changes to the class rather than a simple fix.

As such, we feel this is a very important, yet unexamined, area of OO design recovery. This work has three main contributions. The first is a taxonomic description of axiomatic object-oriented method stereotypes. This is the first comprehensive investigation on this topic with respect to reverse engineering and

Example - Stereotypes

Stereotype Category	Stereotype	Description
Structural Accessor	get	Returns a data member.
	predicate	Returns Boolean value that is not a data member.
	property	Returns info about data members.
	void-accessor	Returns information via a parameter.
Structural Mutator	set	Sets a data member.
	command	Performs a complex change to the object's state.
	non-void-command	
Creational	constructor, copy-constructor, destructor, factory	Creates and/or destroys objects.
Collaborational	collaborator	Works with objects (parameter, local or return value).
	controller	Changes only an external object's state (not *this*).
Degenerate	incidental	Does not read/change the object's state. No calls to other class methods.
	stateless	Does not read/change the object's state. One call to other class methods.
	empty	Has no statements.

Usage Patterns

	Identifier - tile_list_head usages	Identifier – tile_list_tail usages
1	static GList * tile_list_head = NULL;	static GList * tile_list_tail = NULL;
2	list = tile_list_head;	if (list == tile_list_tail)
3	**g_assert (tile_list_head != tile_list_tail);**	**g_assert (tile_list_head != tile_list_tail);**
4	tile_list_head = g_list_remove_link (tile_list_head, list);	tile_list_tail = g_list_last (g_list_concat (tile_list_tail, list));
5	while (tile_list_head && …){…}	tile_list_tail = g_list_append (tile_list_tail, tile);
6	gimp_tile_cache_flush (tile_list_head->data);	g_hash_table_insert (tile_hash_table, tile, tile_list_tail);
7	if (! tile_list_head)	tile_list_tail = g_list_last (tile_list_tail);
8	**tile_list_head = tile_list_tail;**	**tile_list_head = tile_list_tail;**
9	tile_list_head = g_list_remove_link (tile_list_head, list);	if (list == tile_list_tail)
10	if (! tile_list_head)	tile_list_tail = tile_list_tail->prev;

Altogether

- Grammar patterns gives us natural language semantics
- Stereotypes, lexical categories provide static-based program semantics for all identifiers at class, method, and function-local levels
- Usage patterns allow us to discriminate identifiers with similar name and type-based semantics

We will open source all code to generate this model

https://scanl.org/

SCANL

Source Code Analysis And
Natural Language Laboratory