Exploiting the Correspondence between Micro Patterns and Class Names

Jeremy Singer and Chris Kirkham

University of Manchester, UK

Thanks to Mark Harman for presenting
What are Micro Patterns?

- Simple single-class properties
- Detect with efficient static analysis
- Invented by Gil and Maman [OOPSLA ’05]
Example Micro Pattern

```java
public class List {
    Object head;
    List tail;
}
```

exhibits the recursive micro pattern, since at least one instance field has the same type as the class itself.
Another Example Micro Pattern

```java
public class Point {
    public int getX() {
        return this.x;
    }
    public int getY() {
        return this.y;
    }
}
```

exhibits the data manager micro pattern, since all methods are data accessors.
Existing Applications of Micro Patterns

- Used for static program analysis and optimization
  - detect bugs in development of software project (as classes change MPs)
  - predict object lifetimes for garbage collection (some MPs live longer)
Existing Applications of Micro Patterns

- Used for static program analysis and optimization
  - detect bugs in development of software project (as classes change MPs)
  - predict object lifetimes for garbage collection (some MPs live longer)
- our new technique: correlate MPs with class names
Java Class Names

- Camel Case: multiple words run together, capital letter marks new word
- Descriptive: adjectives and nouns
- Example: `ByteArrayBuffer`
Java Class Names

- Camel Case: multiple words run together, capital letter marks new word
- Descriptive: adjectives and nouns
- Example: `ByteArrayBuffer`
- focus on last word in name: `suffix`
- e.g. `Buffer`
Using the Semantic Information

- In program comprehension, we often use natural language information
- Not generally the case for static program analysis/optimization
- We show a relationship between class names and micro patterns
Using the Semantic Information

- In program comprehension, we often use natural language information
- Not generally the case for static program analysis/optimization
- We show a relationship between class names and micro patterns
  - this allows us to use class names for analysis/optimizations!
Our hypothesis

Class name suffix is often an indicator of micro patterns exhibited by that class.
Our investigation

- Large corpus of open-source Java applications
- around 30,000 classes
- around 4,000 distinct class name suffixes
Our investigation

- Large corpus of open-source Java applications
- around 30,000 classes
- around 4,000 distinct class name suffixes
- $\Rightarrow$ suffix re-use is common practice for Java developers
• 50% of suffixes (2000/4000) unique to a single class
• 5% (200/4000) shared between 20+ classes
Examine each of the $N$ classes with suffix $S$. If all $N$ classes exhibit micro pattern $p$
   ▶ create a rule that associates $S$ with $p$
Rule generation

- Examine each of the \( N \) classes with suffix \( S \).
- If all \( N \) classes exhibit micro pattern \( p \)
  - create a rule that associates \( S \) with \( p \)
- If 90% of \( N \) classes exhibit \( p \)
  - create a rule that associates \( S \) with \( p \), with confidence level 90%.
Rule generation

- Examine each of the $N$ classes with suffix $S$.
- If all $N$ classes exhibit micro pattern $p$
  - create a rule that associates $S$ with $p$
- If 90% of $N$ classes exhibit $p$
  - create a rule that associates $S$ with $p$, with confidence level 90%.

- Statistical significance issues
  - Over all the classes, for the most popular micro pattern, there is only a 4% chance that two randomly selected classes will share that micro pattern.
For suffixes with at least two classes, from at least two programs *(see paper for more graphs with different parameters)*

- Around 70 rules at 100% confidence
Comparable

suffix shared between 10 classes. 100% of these classes exhibit the *PureType* micro pattern, i.e. they only contain abstract methods, they have no fields or static members.
Example Rules

**Comparable**

suffix shared between 10 classes. 100% of these classes exhibit the *PureType* micro pattern, i.e. they only contain abstract methods, they have no fields or static members.

**Exception**

suffix shared between 839 classes. 88% of these exhibit the *Sink* micro pattern, i.e. their methods do not propagate calls to any other methods (leaf methods).
Example Rules

**Comparable**

suffix shared between 10 classes. 100% of these classes exhibit the *PureType* micro pattern, i.e. they only contain abstract methods, they have no fields or static members.

**Exception**

suffix shared between 839 classes. 88% of these exhibit the *Sink* micro pattern, i.e. their methods do not propagate calls to any other methods (leaf methods).

*Possible optimizations / bug checks for these rules are presented in paper.*
Applications of these rules

- build or download a database of such (suffix,pattern) rules
- apply at code development time
  - to get auto-complete hints
- apply at code review time
  - to identify possible bugs
Auto-complete hints

- developer types class name in IDE
- automatic wizard analyses the suffix
  - suggests possible micro patterns for this class
  - links to documentation
  - fills in skeleton source code
Development time tool: Eclipse wizard
Review time tool: Lint-like checker

Given complete source code for a class, check to see if it violates the micro pattern rules for that suffix. Warn user of potential problems:

Example

*Violation* of Recursive micro pattern! Class TreeNode, declared in file:TreeNode.xml, line 9, does not contain any instance fields of type TreeNode. This rule has confidence 75%
Conclusions

- *Class name suffix is often an indicator of micro patterns exhibited by that class.*

- **Why is this useful?**
  - formalizing the instinctive behaviour of Java programmers
    - suffix/pattern rules
  - exploiting rules for program analysis and optimization
    - prototype tools presented