Exploiting Code Search Engines to Improve Programmer Productivity & Quality

Suresh Thummalapenta
(Advisor: Dr. Tao Xie)
sthumma@ncsu.edu

NC STATE UNIVERSITY Computer Science

An extensible framework that exploits the strength of a code search engine (CSE) and addresses the major limitations in using the CSE for tasks that help improve the productivity and quality.

Motivation

Programmers commonly reuse APIs of existing frameworks or libraries

- **Advantages: Low cost and high efficiency**
- **Challenges: Complexity and lack of documentation**
- **Consequences: Low programmer productivity and defects in the API client code**

Existing Approaches

- **Gather a few client applications and mine common patterns of reuse**
- **Improve productivity by suggesting mined patterns of reuse**
- **Detect violations of mined patterns**

Problem

Major issues:
- Large number of False positives (fp) and False negatives (fn)

Examples:

- **JADET [ESEC/FSE 07] with AspectJ produced 87.84% fp violations**
- **PRMiner [FSE 05] with Linux, PgSql, and Apache produced 40%, 75%, and 86% fp violations, respectively**

Primary reason: Limited data scope

Application 1: PARSEWeb [1]

- **Problem:** Programmers often know what type of object that they need (destination) but do not know how to get that object with a specific method invocation sequence (MIS) from a known object (source)
- **Solution:** Gather relevant code samples with source and destination objects from a CSE and mine frequent MIS from the samples that accept the source object as input and yield the destination object.

Example

Question: How to parse the code in a dirty editor of Eclipse IDE?

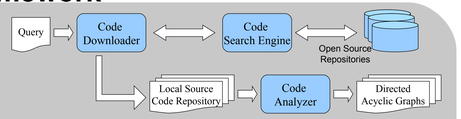
Query: IEditorPart -> ICompilationUnit

Solution sequence:

```
IEditorPart iep;
IEditorInput editorInp = iep.getEditorInput();
IWorkingCopyManager wcm =
JavaUI.getWorkingCopyManager();
ICompilationUnit icu = wcm.getWorkingCopy(editorInp);
```

Base Framework

- **Our approach: Extend data scope to billions of lines of open source code by leveraging a code search engine (CSE)**
- **Challenges: Code samples returned by a CSE are often partial and not compilable**
- **Solution: Use heuristics for type resolution**



A sample heuristic:

How to get the return type of method call "createSession" in the partial code sample below?

```
public QueueSession test () {
…  return connect.createSession(false);… }
```

Return type: Same as or a subtype of the return type of the enclosing method *test.*

Application 2: SpotWeb [2]

- **Problem:** Programmers unfamiliar to a given framework often face challenges in identifying where to start for reusing APIs of the framework
- **Solution:**
  - Compute usage metrics for each method of the given framework from the code samples gathered from a CSE; usage metrics represent how often a method is reused by code samples such as the method is invoked for 10 times or overridden for 15 times
  - Analyze computed usage metrics to identify starting points (referred as hotspots) of the given framework

Application 3: NEGWeb [3]

- **Problem:** Neglected conditions (missing conditions that check the receiver, arguments, or return objects) are quite common among the defects in the API client code
- **Solution:**
  - Mine condition patterns of receiver, arguments, or return objects around the call sites of a given API among the code samples gathered from a CSE
  - Apply mined condition patterns to detect violations in the API client code

Results

PARSEWeb:
With related tools Strathcona [ICSE 05] and Prospector [PLDI 05] for 12 tasks of Eclipse IDE programming

- **PARSEWeb: 91% (11 / 12)**
- **Strathcona: 50% (6 / 12)**
- **Prospector: 58% (7 / 12)**

SpotWeb:

- **Log4j: Recall 100%**
  Precision 16.21% (11 out of 12 classes in top 16)
- **JUnit: Recall 100%**
  Precision 26.08% (5 out of 6 classes in top 7)

NEGWeb:

- **Mined patterns of 5 open source applications, confirmed 3 defects in the literature, and found 3 new defects in Columba**
- **Condition patterns of Java Util package: Rules: 56.25% (36/64), Usage Patterns: 26.56% (17/64), and False positives: 17.18% (11/64)**
- **Violations in Columba application: Defects: 4.28% (3/70), Code smells, wrappers, and hints: 75.71% (53/70), False positives: 20% (14/70)**

Future Directions

- **Latitudinal Program Analysis: Analyze code bases locally. Then exploit a code search engine to gather similar code locations elsewhere to validate/ improve local analysis/inference**
- **Extend the base framework to detect exception related errors while reusing the APIs of a given framework**

References

[1] S. Thummalapenta and T. Xie. "*PARSEWeb: A Programmer Assistant for Reusing Open Source Code on the Web*", **ASE 2007**

[2] S. Thummalapenta and T. Xie. "*SpotWeb: Characterizing Framework API Usages Through a Code Search Engine*", **NCSU TR-2007-34, 2007**

[3] S. Thummalapenta and T. Xie. "*NEGWeb: Static Defect Detection via Searching Billions of Lines of Open Source Code*", **NCSU TR-2007-24, 2007**