# Exploiting Code Search Engines to Improve Programmer Productivity & Quality

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

NC STATE UNIVERSITY Computer Science

## Motivation

**Programmers commonly reuse APIs of existing frameworks or libraries**

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

## Our Approach

- **Mine common patterns of API reuse from existing applications on the web by exploiting a code search engine (CSE)**
- **Productivity:**
  - **Suggest mined common patterns that help programmers in reusing APIs of a framework**
  - **Identify a framework's hotspots that can serve as starting points in reusing APIs of a framework**
- **Quality:**
  - **Apply mined patterns of API reuse to detect violations in the API client code**

## Application 1: PARSEWeb [1]

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

### Example

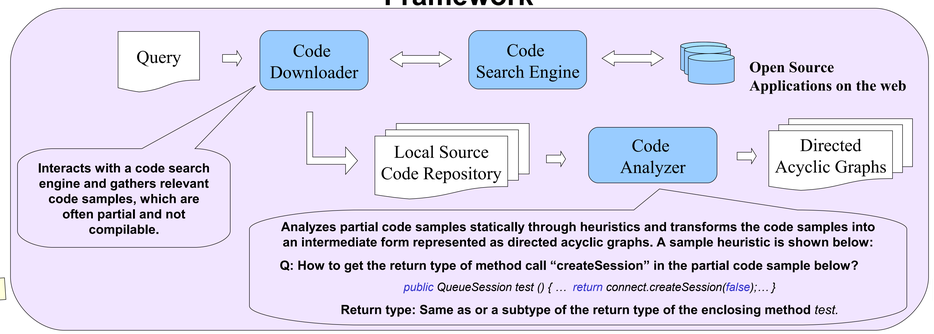
**Question:** How to parse the code in a dirty editor of Eclipse IDE?

**Query:** IEditorPart -> ICompilationUnit

**Solution sequence:**

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

## Framework



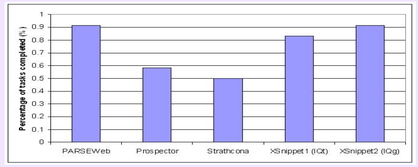

## Results

### PARSEWeb

**Real programming problem from Jakarta BCEL user forum, 2001**

- **Question: "How to disassemble Java byte code"**
- **Query: "Code → Instruction"**
- **Solution Sequence:**

```
Code code;
InstructionList il = new InstructionList(code.getCode());
Instruction[] ins = il.getInstructions();
```




**Comparison with related tools Prospector, Strathcona, and XSnippet**

### NEGWeb

- **Mined patterns of 5 open source applications with a low percentage of false positives, confirmed 3 defects in the literature, and found 3 new defects in Columba**
- **Mined condition patterns of Java Util package: Rules: 56.25% (36/64), Usage patterns: 26.56% (17/64), and False positives: 17.18% (11/64)**
- **Highlights of Columba case study: All 3 defects are detected by top 10 patterns and no false positives among the violations detected by top 10 patterns**

**A real defect detected in Columba**

```
private String removeDoubleEntries(String input) {
   Pattern sP = Pattern.compile("s*(<[^s<>]+>)s*");
   ArrayList entries = new ArrayList();
   Matcher matcher = sP.matcher(input);
   while (matcher.find()) {
         entries.add(matcher.group(1));
   }
   Iterator it = entries.iterator(); ...
   String last = (String) it.next();
}
```

- **Violated pattern: "next PRE_METHOD BOOLEAN hasNext"**
- **Support: 0.929**
- **Description: The code results in NullPointerException, if the input string does not match with the regular expression.**

### SpotWeb

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

## Application 2: NEGWeb [2]

- **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 by using dominance and data-dependency relationships
  - Apply mined condition patterns to detect violations in the API client code

### Example

**Sample Code**

```
01:public static void verifyBCEL(String cName) {
04: Verifier verf = …
05: if(verf != null) {
06:     vr0 = verf.doPass1();
07:     if(vr0 != VerificationResult.VR_OK)
08:         return;
09:     vr1 = verf.doPass2();
10:     if (vr1 == VerificationResult.VR_OK) {
11:         …
12:         for(mId=0; mId<jc.getMethods().length; mId++) {
13             vr2 = verf.doPass3a(mId);
14:            vr3 = verf.doPass3b(mId);
15:            …
16:         }
17:     }
18:}
```

**Extracted Condition Patterns**

| Method | Pattern Type | Condition Type | Additional Info. |
|---|---|---|---|
| doPass1 | RECEIVER | NULLITY | |
| doPass1 | RETURN | CONST_EQUAL | VerificationResult, VR_OK |
| doPass2 | PRE_METHOD | CONST_EQUAL | doPass1 |
| doPass3a | ARGUMENT | GEN_EQUALITY | 0 < Arguments < # methods |
| doPass3a | PRE_METHOD | CONST_EQUAL | doPass2 |

## Application 3: SpotWeb [3]

- **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 class and method, and analyze the metrics to identify starting points (referred as hotspots) of the given framework
- **Example:** Computed usage metrics for the *TestSuite* class of the JUnit framework

**Class: junit.framework.TestSuite**
**Invocations: 165, Extensions: 32, Implements: 0**

**Methods:**
**void,addTestSuite(java.lang.Class)**
**Invocations: 42, Overrides: 1, Implements: 0**

## Future Directions

- Latitudinal Program Analysis: Analyze code bases locally and exploit a code search engine to gather similar code locations elsewhere to validate/ improve local analysis/inference
- Develop a new application 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. "*NEGWeb: Static Defect Detection via Searching Billions of Lines of Open Source Code*", **NCSU TR-2007-24, 2007**

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