Exploiting Code Search Engines to Improve Programmer Productivity and Quality

Suresh Thummalapenta

Advisor: Dr. Tao Xie

Department of Computer Science

North Carolina State University

sthumma@ncsu.edu

Student Research Competition

October 23, 2007

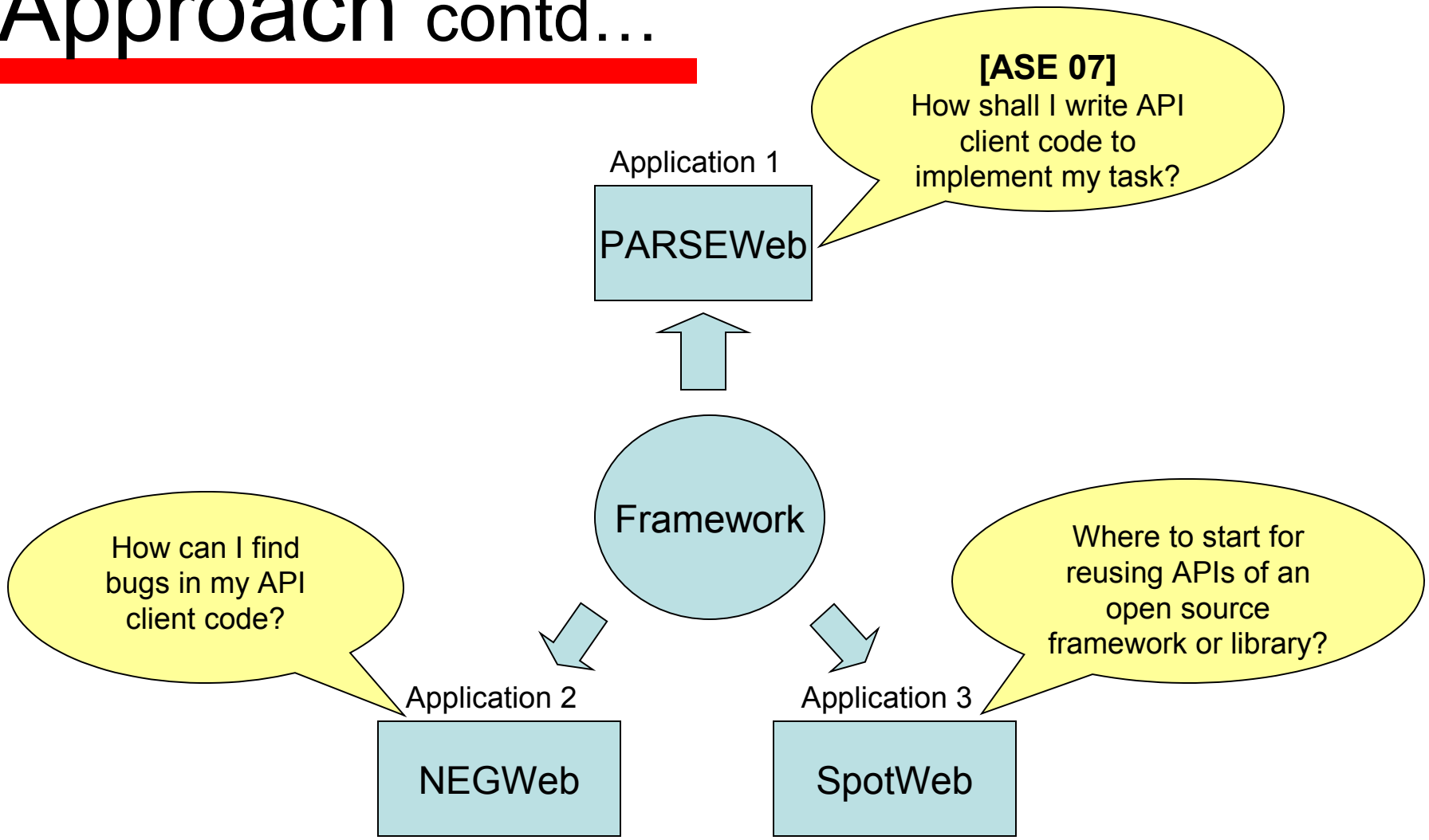
Motivation

- 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
    - Defects in the API client code

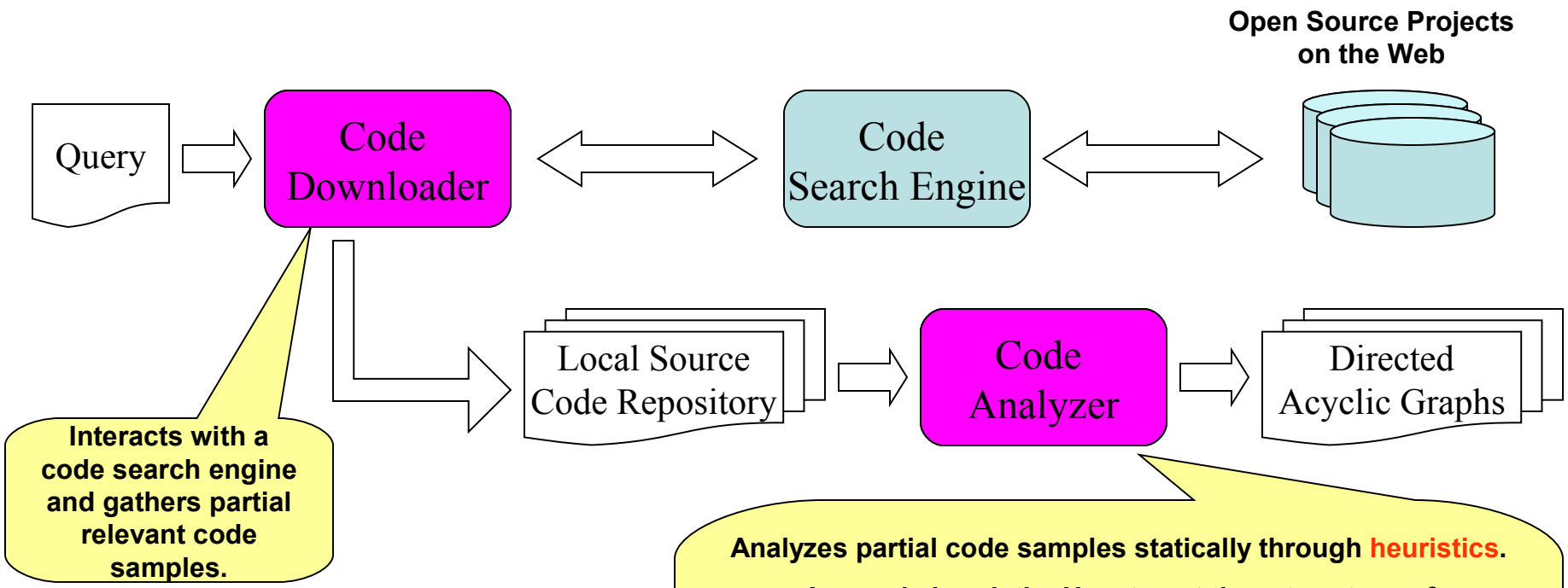
Approach

- Mine common patterns of API reuse from existing applications on the web by exploiting a code search engine
- How we address 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
- How we address quality?
  - Apply mined patterns of API reuse to detect violations in the API client code

Approach contd…



Framework



Analyzes partial code samples statically through heuristics.

A sample heuristic: How to get the return type of createQueueSession?

```
public QueueSession test() { ...
        return connect.createQueueSession(false,int);   }
```

The return is the same as or a subtype of enclosing method declaration *test*.

Application 1: PARSEWeb

- Problem: Programmers often
  - » know what type of object they need
  - » but don't know how to write code to get that object
- Example task: How to parse code in a dirty editor in Eclipse IDE programming?
- Query: IEditorPart → ICompilationUnit
- PARSEWeb Solution:

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

- Difficulties:

  a. Needs an instance of *IWorkingCopyManager*

  b. Needs to invoke a static method of *JavaUI* for getting the preceding instance

Application 1: PARSEWeb

0_JavaFileMerger.java

```
// delete Abator generated stuff, and collect imports
astParser.setSource(icu);
CompilationUnit cu = (CompilationUnit) astParser.createAST(null);
AST ast = cu.getAST();

ExistingJavaFileVisitor visitor = new ExistingJavaFileVisitor();

cu.recordModifications();
```

Problems | Javadoc | Declaration | PARSEWeb Search Results

org.eclipse.jdt.core.ICompilationUnit -> org.eclipse.jdt.core.dom.CompilationUnit

File Name and Method Sequences	Rank	Frequency	Length	Confidence	U...
File: 0_JavaFileMerger.java Method: getMergedSource	1	3	2	High	1
org.eclipse.jdt.core.dom.ASTParser,setSource(org.eclipse.jdt.core.ICompilationUnit) ReturnType:void					
org.eclipse.jdt.core.dom.ASTParser,createAST(null) DownCasted_ReturnType:org.eclipse.jdt.core.dom.CompilationUnit					
File: 5_BugResolution.java Method: runInternal	2	1	1	High	1
this,createWorkingCopy(org.eclipse.jdt.core.ICompilationUnit) ReturnType:org.eclipse.jdt.core.dom.CompilationUnit					

Snapshot of the PARSEWeb Eclipse plugin

Application 2: NEGWeb

Problem: Neglected conditions (missing conditions on receiver, arguments or return objects of an API) are often in the API client code

Solution: Mine condition patterns for APIs and detect violations of patterns

```
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:         …
16:     }
17:   }
18:}
```

Conditions extracted by NEGWeb:

```
01: doPass1 RECEIVER NULLITY
02: doPass2 PRE_METHOD CONST_EQUAL doPass1
03: doPass1 RETURN CONST_EQUAL VerificationResult.VR_OK
```

Application 3: SpotWeb

- Problem: Programmers unfamiliar to a framework face challenges in identifying where to start for reusing APIs of the framework
- Solution: Computes usage metrics for all classes and methods and uses the computed 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

Instances: 165, Extensions: 32, Implements: 0

Methods:

void,addTestSuite(java.lang.Class)

Invocations: 42, Overrides: 1, Implements: 0

Results

- PARSEWeb with related tools Strathcona [Holmes and Murphy ICSE 05] & Prospector [Mandelin et al. PLDI 05] for 12 tasks of Eclipse programming:
  - PARSEWeb: 91% (11/12)
  - Strathcona: 50% (6/12)
  - Prospector: 58% (7/12)

- 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)

Results contd…

- NEGWeb
  - Summary
    - Mined patterns of 5 open source applications with a low percentage of false positives
    - Confirmed 3 defects in the literature
    - Found 3 new defects in Columba
  - Mined condition patterns of Java Util Packages
    - Rules: 56.25% (36/64)
    - Usage Patterns: 26.56% (17/64)
    - False positives: 17.18% (11/64)
  - Highlights of Columba case study
    - All 3 defects are detected by top 10 patterns
    - No false positives among the violations detected by top 10 patterns

Conclusion

- Designed a generic framework by exploiting a code search engine. Developed three applications and showed that the applications can perform better than existing related tools
  - PARSEWeb: mines API usage patterns for type queries
  - NEGWeb: mines condition patterns around APIs
  - SpotWeb: identifies starting points of a framework
- In future work, we plan to develop a new application to detect exception-related errors while reusing the APIs of a given framework

Questions?

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**