Exploiting Code Search Engines to Improve Programmer Productivity

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

NC STATE UNIVERSITY Computer Science

An extensible framework that helps to exploit the strength of code search engines and addresses the major limitations in using code search engines for tasks that help improve the productivity of the programmer.

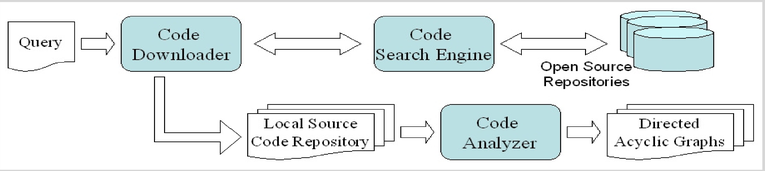
Motivation

Strengths of Code Search Engines (CSE):
- Powerful resources of open source code.
- Can be exploited for several tasks like searching for relevant code samples, identifying framework hotspots, and finding bugs.

Major limitations in exploiting CSEs:
- Returned code samples are too many.
- The code samples returned by CSEs are often partial.

Framework: Overview

Major components of our framework:
- Code Search Engine
- Code downloader
- Code analyzer : Uses heuristics for analyzing partial code samples statically and builds Directed Acyclic Graph.



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.*

Transformation Process

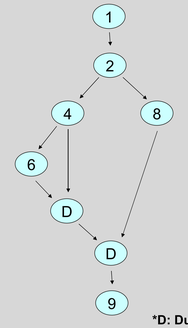
Sample code:

```
1 public QueueSender test()   { ...
2     queueFactory=(QueueConnectionFactory)context.lookup("Book");
3     if (i == 1){
4         queueConnection = queueFactory.createQueueConnection();
5         while(k <= 2){
6             queue = (Queue)context.lookup("BookJMSFifoQueue");}
7     } else {
8         queueConnection =queueFactory.createQueueConnection();
9     } return queueSession.createSender(queue);
10 }
```

Textual form of the graph:

```
1 public QueueSender test()   { ...
2     javax.naming.Context,lookup(java.lang.String)
          DownCasted_ReturnType:javax.jms.QueueConnectionFactory
3     IF-THEN{
4         javax.jms.QueueConnectionFactory,createQueueConnection()
              ReturnType:javax.jms.QueueConnection
5         WHILE/FOR{
6             javax.naming.Context,lookup(java.lang.String)
                  DownCasted_ReturnType:javax.jms.Queue}
7     } IF-ELSE {
8         javax.jms.QueueConnectionFactory,createQueueConnection()
              ReturnType:javax.jms.QueueConnection
9     }javax.jms.QueueSession,createSender(javax.jms.Queue)
          ReturnType:javax.jms.QueueSender
10 }
```

Transformed graph:



*D: Dummy nodes inserted by our framework.

Application 1: PARSEWeb

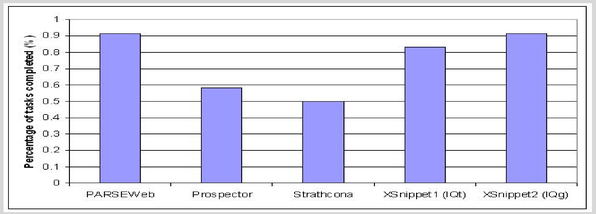
Problem:

Programmers often know what type of object that they need, but do not know how to get that object with a specific method sequence.

Query: "Source → Destination"

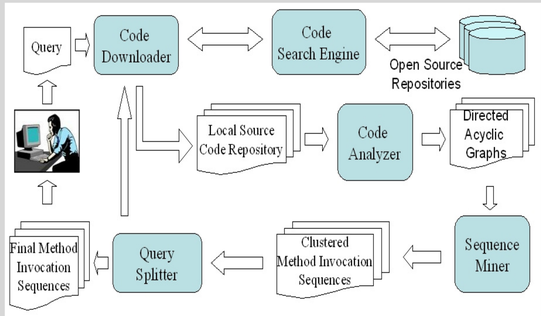
Results:

PARSEWeb results with related existing tools Prospector, Strathcona, and XSnippet.



Solution:

Identify nodes in the Directed Acyclic Graph that contain Source and Destination object types, and calculate the shortest path from Source node to Destination node to get a solution method sequence.



Application 2: Hotspotter

Problem:

Identify framework hotspots, weakspots, and deadspots

- *Hotspots: More used APIs, helpful for users.*
- *Weakspots and Deadspots: Less and never used APIs, helpful for developers.*

Solution:

Identify usages of each API of the framework.

*Usage Percentage (UP) = (No. of usages of API / Total no. of usages of all APIs) * 100*

An API is Hotspot if UP > Upper Threshold

An API is Weakspot if UP < Lower Threshold & UP != 0

An API is deadspot if UP = 0

Results:

S.No.	Subject Name	Total No. of APIs	Hotspots No.	Hotspots %	Weakspots No.	Weakspots %	Deadspots No.	Deadspots %
1	JUnit	379	22	5.8	83	21.9	124	32.7
2	Log4j	1334	21	1.6	180	13.5	746	55.9
3	BCEL	2691	18	0.7	486	18.1	1867	69.4
4	Struts	4679	28	0.6	0	0	4332	92.6

Total: Total number of APIs, No.: Number of APIs identified as hotspots, weakspots, and deadspots and their percentage (shown as %) among the total number of APIs.

Results summary: Framework users often use only a small subset of all APIs.

Sample hotspots in JUnit:
- Constructor of junit.framework.TestSuite class.
- Method: void junit.framework.TestSuite.addTest(junit.framework.Test).

Sample hotspot in Log4j:
- Method: org.apache.log4j.Logger org.apache.log4j.Logger.getLogger(String).

Conclusion

Our extensible framework helps to use the strength of CSEs and addresses the major limitations in using CSEs. We showed the effectiveness of our framework with two tools developed based on our framework. In future work, we plan to extend our framework for detecting bugs by mining specification patterns from the code samples returned by CSEs.