CSNIPPEX: Automated Synthesis of Compilable Code Snippets from Q&A Sites

Valerio Terragni  Yepang Liu  Shing-Chi Cheung

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
{vterragni, andrewust, scc}@cse.ust.hk

ISSTA 2016
Social Network Revolution
Q&A sites for developers

Millions of high quality code snippets!

- Solutions of programming tasks
- Bug fixes
- API usage examples

12 Million Questions
19 Million Answers

http://stackexchange.com/sites?view=list#traffic June 2016
Usefulness of Q&A’s Code Snippets

- Developers often search code snippets in Q&A sites [Mao@RN2010, Stoole@TOSEM2014]

- Stackoverflow receives **500 Million** views per month

Usefulness of Q&A’s Code Snippets

code reuse and analysis

Dynamic/Static Analysis

- Collect API usage profiles
  - Regression testing
  - Mining temporal specifications

- Crowd debugging [Chen@FSE2015]

- Crowd bug fixing [Gao@ASE2015]
Many Code Snippets Do Not Compile

Written *concisely*, without implementation details [Naeshi@ICSM2012]

- Absence of import declarations or fully qualified names
- Dangling statements/methods
- Typos
- Place holders

```java
// regex for any sequence of non-comma, non-parenthesis characters that
// neither starts nor ends with whitespace:
Pattern p = Pattern.compile("[^,\s()]{1,}[^,\s()]{0,}\?");
Matcher m = p.matcher(textToMatch);
while (m.find()) {
    System.out.println(m.group()); // print entity
}
```

Many code snippets are **non-executable** and semantically **incomplete** for precise static analysis
Many Code Snippets Do Not Compile

Manual synthesis
- Tedious
- Requires familiarity with libraries
- Not scalable

Can we do it automatically?

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Answer9745185 {
    private static CharSequence textToMatch;

    public static void main(String[] args) {
        // regex for any sequence of non-comma, non-parentheses characters
        // that neither starts nor ends with whitespace:
        Pattern p = Pattern.compile("[^,\s]\)?([,\s]+\)?\s*\s*");
        Matcher m = p.matcher(textToMatch);
        while (m.find()) {
            System.out.println(m.group()); // print entire matched substring
        }
    }
}
```
Accepted answers or with score ≥ 2

491,906 Posts with Java code snippets

Baseline Synthesis

- Download external JARs from fully qualified names
- Create synthetic classes and methods for dangling statements/methods

Baseline Synthesis

https://mvnrepository.com/artifact/com.sun.org.apache/jaxp-ri/1.4

```java
import com.sun.org.apache.bcel.internal.classfile.JavaClass;
import com.sun.org.apache.bcel.internal.classfile.LocalVariable;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        ClassParser parser = new ClassParser("Main.class");
    }
}
Problem Understanding
baseline synthesis

Accepted answers
or with score >= 2

491,906 Posts with Java code snippets

- Download external JARs from fully qualified names
- Create synthetic classes and methods for dangling statements/methods

Baseline Synthesis

```java
public class Answer9745185{
    public static void main(String[] args){
        // regex for any sequence of non-comma, non-parenthesis characters that
        // neither starts nor ends with whitespace:
        Pattern p = Pattern.compile("[^,\s()]*[^,\s()]");
        Matcher m = p.matcher(textToMatch);
        while (m.find()) {
            System.out.println(m.group()); // print entire matched substring
        }
    }
}
```
Problem Understanding
baseline synthesis

Accepted answers
or with score >= 2

491,906
Posts with
Java code
snippets

Baseline
Synthesis

Java
Compiler

Only 8.41% (41,349)
successfully compile
# Missing Declarations

## Most common error type

3,905,444 compilation errors

<table>
<thead>
<tr>
<th>Top@</th>
<th>Error code</th>
<th>freq.</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 1</td>
<td>compiler.err.cant.resolve</td>
<td>1,485,626</td>
<td>38.04%</td>
</tr>
<tr>
<td>Top 2</td>
<td>compiler.err.expected</td>
<td>1,188,663</td>
<td>30.44%</td>
</tr>
<tr>
<td>Top 3</td>
<td>compiler.err.not.stmt</td>
<td>256,926</td>
<td>6.58%</td>
</tr>
</tbody>
</table>

Main reasons

1. Wrong inference of compilation-units
2. Missing external dependencies
3. Undeclared variables

Top Error code freq. %
Top 1 compiler.err.cant.resolve 1,485,626 38.04%
Top 2 compiler.err.expected 1,188,663 30.44%
Top 3 compiler.err.not.stmt 256,926 6.58%

Class                     950,324 (24.33%)
Variable                  484,035 (12.39%)
Method                     50,677 (1.30%)
Others                     590 (0.02%)
How To Infer Compilation Units?

35.71% (175,653) stackoverflow posts contain multiple code snippets

Strategy 1 (baseline synthesis) each code snippet in a separate Java class/file

Example 1:
```java
public class C1{

    static void a(){
        [...]
    }
}
```

```java
public class C2{

    static void b(){
        [...]
    }
}
```

```java
public class C3{

    public static void main(String[] args){
        a();
        b();
    }
}
```

Example 2:
```java
public class C1{

    static void a(){
        [...]
    }
}
```

```java
public class C2{

    public void a(){
        [...]
    }
}
```

```
```
How To Infer Compilation Units?

35.71% (175,653) stackoverflow posts contain multiple code snippets

**Strategy 2**: always merge all code snippets in a post in the same Java class

Example 1:
```java
public class C1{
    static void a(){
        [...]
    }
}
```

Example 2:
```java
public class C1{
    public void a(){
        [...]
    }
}
```

Example 2:
```java
public class C1{
    public void a(){
        [...]
    }
}
```

Example 2:
```java
public class C1{
    public void a(){
        [...]
    }
}
```

```
```
How To Resolve Missing Dependencies?

Only 6.88% (33,833) posts contain import declarations

A simple name can match many fully qualified names in different libraries [Subramanian@ICSE2014]

```java
File input = new File(fileName);
Document doc = Jsoup.parse(input, "UTF-8");
String newTitle = doc.select("font.classname").first()
doc.title(newTitle);
PrintWriter writer = new PrintWriter(input,"UTF-8");
writer.write(doc.html());
writer.flush();
writer.close();
}
```

<table>
<thead>
<tr>
<th>File</th>
<th>IOException</th>
<th>PrintWriter</th>
<th>Document</th>
<th>Jsoup</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.specs.runner</td>
<td>com.sun.star.io</td>
<td>java.io</td>
<td>org.bson</td>
<td>org.jsoup</td>
</tr>
<tr>
<td>scala.reflect.io</td>
<td>java.io</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.io</td>
<td>net.kuuko.vertigo.io</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.....</td>
<td>......</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10 * 14 * 1 * 97 * 1 = 13,580

on average (in our experiments) # possible configurations for each Java file is $2.51 \times 10^{34}$
How To Declare Undeclared Variables?

Quick-fix

How to partition multiple code snippets in Java files?

How to recover external dependencies by simple names?

- Without the right JAR in the buildpath it can only suggest to mock declarative completeness

- Does not suggest which import declaration to generate
CSNIPPEX
Code SNIPPet Extractor

- Feedback-directed approach guided by compilation errors
- C-units inference and dependency resolution prepare the working environment for Eclipse Quick Fix
C-Unit Inference

Input Q&A post

Baseline Synthesis

Compilation Units

Java Compiler

C-Unit Inference

Dependencies Resolver

Code Completer

Stack Overflow

Java

Int a = 0;
a++;
</pre></code>

Public class

Import ...

Compilation errors

Output

Merge c-unit2

No errors

Quick Fix

Quick fixes available:
Follow the order of occurrence!

Example 1:
```java
public class C1{  
static void a(){  
[...]
}  
}
```

```
static void b(){  
[...]
}  
}
```

```
public static void main(String[] args){  
a();  
b();  
}
```

Example 2:
```
public class C1{  
}
```

```
public void a(){  
[...]
}
```

```java
public class C2{  
}
```

```
public void a(){  
[...]
}
```

```
public void a(){  
}
```

**C-Unit Inference**

**Example 1:**
- `public class C1{` is defined.
- `static void a(){` is defined.
- `static void b(){` is defined.
- `public static void main(String[] args){` is defined.

**Example 2:**
- `public class C1{` is defined.
- `public void a(){` is defined.
- `public class C2{` is defined.
- `public void a(){` is defined.
- `public void a(){` is defined.

**Result:**
- `YES` for `already.defined`?
  - Yes
  - No

**Keep/Unmerge:**
- Yes
- No
Dependencies Resolver

Input Q&A post

Baseline Synthesis

Compilation Units
package a
package a
import ....
public class

Java Compiler

Compilation errors
no errors

C-Unit Inference

Dependencies Resolver
import org..
c-unit1 merge c-unit2

Code Completer

Quick Fix

int a = 0;
a++;
</code></pre>

Baseline Synthesis

Compilation Units
package a
package a
import ....
public class

Java Compiler

Compilation errors
no errors

C-Unit Inference

Dependencies Resolver
import org..
c-unit1 merge c-unit2

Code Completer

Quick Fix
Clustering Hypothesis

Import declarations in the same compilation unit likely form clusters **naturally**, each of which refers to a package or sub-package.

**Correct import declaration**

```java
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
```

**Why?**

Types from the same package more likely interact with one another than with those from other packages.

Is the clustering hypothesis often valid?
Validating the Clustering Hypothesis

Import declarations in the same compilation unit likely form clusters naturally, each of which refers to a package or sub-package.

- **Distance between two packages**
  \[ d(p_A, p_B) = \text{the length of the longest uncommon suffix} \]

- **Partition** \( P_T^\tau \) of \( I \) such that each pair of packages in the same subset (cluster) has a distance less than a threshold \( \tau \)

- **Heterogeneity Degree**
  \[ HD^2_I = \frac{2}{5} \cdot 100 = 40 \]

```java
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Document;
```

- \( \tau = 2 \)
  - \( d(\text{java.util, java.util}) = 0 \)
  - \( d(\text{org.jsoup, java.util}) = 2 \)
  - \( d(\text{org.jsoup, org.jsoup.nodes}) = 1 \)
  - \( d(\text{java.util, org.jsoup.nodes}) = 3 \)
Validating the Clustering Hypothesis

~31 Million complete (compilable) Java files
~198 Million import declarations

http://boa.cs.iastate.edu/stats/index.php  Dyer@ICSE2013

<table>
<thead>
<tr>
<th></th>
<th>HD¹</th>
<th>HD²</th>
<th>HD³</th>
<th>HD⁴</th>
</tr>
</thead>
<tbody>
<tr>
<td>average</td>
<td>62.00</td>
<td>44.44</td>
<td>34.64</td>
<td>27.68</td>
</tr>
<tr>
<td>median</td>
<td>60</td>
<td>40</td>
<td>28.57</td>
<td>22.22</td>
</tr>
</tbody>
</table>

Average heterogeneity degree

# import declarations per class
Dependencies Resolver

- A solution with low HD is more likely to be the correct one
- Too expensive to enumerate all possible solutions and compute HD
- We propose a greedy algorithm

STEP1: compute the global frequency for each package

<table>
<thead>
<tr>
<th>File</th>
<th>IOException</th>
<th>PrintWriter</th>
<th>Document</th>
<th>Jsoup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 org.specs.runner</td>
<td>1 com.sun.star.io</td>
<td>3 java.io</td>
<td>1 org.bson</td>
<td>1 org.jsoup</td>
</tr>
<tr>
<td>1 scala.reflect.io</td>
<td>3 java.io</td>
<td></td>
<td>1 org.jdom</td>
<td></td>
</tr>
<tr>
<td>3 java.io</td>
<td>1 net.kujo.vertigo.io</td>
<td></td>
<td>1 org.jsoup.nodes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>......</td>
</tr>
</tbody>
</table>

......
A solution with low HD is more likely to be the correct one

Enumerate all possible solutions and compute HD is too expensive

We propose a greedy algorithm

<table>
<thead>
<tr>
<th>File</th>
<th>IOException</th>
<th>PrintWriter</th>
<th>Document</th>
<th>Jsoup</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io</td>
<td>java.io</td>
<td>java.io</td>
<td>org.bson</td>
<td>org.jsoup</td>
</tr>
<tr>
<td>org.specs.runner</td>
<td>com.sun.star.io</td>
<td></td>
<td>org.jdom</td>
<td></td>
</tr>
<tr>
<td>scala.reflect.io</td>
<td>net.kuuko.vertigo.io</td>
<td></td>
<td>org.jsoup.nodes</td>
<td></td>
</tr>
</tbody>
</table>

**STEP1**: compute the global frequency for each package

**STEP2**: For each simple name order packages by their frequency
A solution with low HD is more likely to be the correct one

Enumerate all possible solutions and compute HD is too expensive

We propose a **greedy** algorithm

TOP solution has the biggest cluster

<table>
<thead>
<tr>
<th>File</th>
<th>IOException</th>
<th>PrintWriter</th>
<th>Document</th>
<th>Jsoup</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>org.specs.runner</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>scala.reflect.io</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td></td>
<td>.....</td>
</tr>
</tbody>
</table>

**STEP1:** compute the global frequency for each package

**STEP2:** For each simple name order packages by their frequency

```java
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
```
A solution with low HD is more likely to be the correct one

Enumerate all possible solutions and compute HD is too expensive

We propose a **greedy** algorithm

### Dependencies Resolver

#### STEP 1: compute the global frequency for each package

<table>
<thead>
<tr>
<th>File</th>
<th>IOException</th>
<th>PrintWriter</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>org.specs.runner</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>scala.reflect.io</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>......</td>
<td>....</td>
<td></td>
</tr>
</tbody>
</table>

#### STEP 2: For each simple name order packages by their frequency

<table>
<thead>
<tr>
<th>Document</th>
<th>Jsoup</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.bson</td>
<td>1</td>
</tr>
<tr>
<td>org.jsoup</td>
<td>1</td>
</tr>
<tr>
<td>org.jdom</td>
<td>1</td>
</tr>
<tr>
<td>org.jsoup.nodes</td>
<td>......</td>
</tr>
</tbody>
</table>

#### STEP 3: Refining the top solution by compilation errors
A solution with low HD is more likely to be the correct one

Enumerate all possible solutions and compute HD is too expensive
- We propose a **greedy** algorithm

### Dependencies Resolver

#### STEP 1: compute the global frequency for each package

#### STEP 2: For each simple name order packages by their frequency

#### STEP 3: Refining the top solution

- by compilation errors
- by higher density threshold

```java
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
```

<table>
<thead>
<tr>
<th>File</th>
<th>IOException</th>
<th>PrintWriter</th>
<th>Document</th>
<th>Jsoup</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>org.specs.runner</td>
<td>1</td>
<td>com.sun.star.io</td>
<td>1</td>
<td>org.jsoup</td>
</tr>
<tr>
<td>scala.reflect.io</td>
<td>1</td>
<td>net.kuujo.vertigo.io</td>
<td>1</td>
<td>org.bson</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>
Code Completer

Baseline Synthesis

Compilation Units
package a
package a
import ....
public class

Java Compiler

no errors
Output

C-Unit Inference
c-unit1
merge
merge
c-unit2

dependencies Resolver

Code Completer

Quick Fix

tvar cannot be resolved to a variable
4 quick fixes available:
Code Completer

Systematic exploration of suggested quick-fixes

Occam’s razor

‘The simplest answer is most often correct!’

Breadth First Search (BFS)

Evaluation
RQ1 synthesis effectiveness

- Download 3,000 popular jars from
- 242,175 posts with at least one `compiler.err.cant.resolve` error

242,175 Q&A posts

<table>
<thead>
<tr>
<th>Error code</th>
<th>freq.</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 1: compiler.err.cant.resolve</td>
<td>1,485,626</td>
<td>38.04%</td>
</tr>
<tr>
<td>Top 2: compiler.err.expected</td>
<td>1,188,663</td>
<td>30.44%</td>
</tr>
<tr>
<td>Top 3: compiler.err.not.stmt</td>
<td>256,926</td>
<td>6.58%</td>
</tr>
</tbody>
</table>

Many errors are outside the scope of the paper
Evaluation
RQ2  precision of the dependencies resolving

Golden set: 13,444 compilable code snippets with import declarations

We removed the user-specified import declarations to evaluate to what extent CSNIPPEX is able to recover them

```java
public class HtmlParser {
    public static void main(String[] args) {
        modifyTitleForAllFilesInFolder(new File("c:/Test"));
        System.out.println("Done");
    }
}
Evaluation
RQ2  precision of the dependencies resolving

Golden set: 13,444 compilable code snippets with import declarations

<table>
<thead>
<tr>
<th>solution</th>
<th>Top@</th>
<th>% compile</th>
<th>% equivalent</th>
<th>average time each post (ms)</th>
<th>median time each post (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top1</td>
<td>76.87%</td>
<td>76.30%</td>
<td></td>
<td>66</td>
<td>32</td>
</tr>
<tr>
<td>Top10</td>
<td>89.66%</td>
<td>87.35%</td>
<td></td>
<td>103</td>
<td>47</td>
</tr>
<tr>
<td>Top100</td>
<td>91.04%</td>
<td>88.27%</td>
<td></td>
<td>4,454</td>
<td>1,889</td>
</tr>
</tbody>
</table>

Clustering hypothesis is effective. Refinements are effective. Compilation is a good proxy for correctness.
Evaluation

RQ2: comparison with Baker [Subramanian@ICSE2014]

Golden set: 13,444 compilable code snippets with import declarations

BAKER state-of-the-art in API link recovering

It found unique matches of external class types for 36.71% code snippets
Conclusion

Social Network Revolution
Q&A sites for developers

 millones of high quality code snippets!

- Solutions of programming tasks
- Bug fixes
- API usage examples


Problem Understanding
baseline synthesis

Accepted answers
or with score >= 2

491,906
Posts with
Java code
snippets

Baseline Synthesis
Java Compiler

Only 8.41% (41,349)
successfully compile

Evaluation
RQ2 precision of the dependencies resolving

Golden set: 13,444 compilable code snippets with import declarations

<table>
<thead>
<tr>
<th>solution</th>
<th>% compile</th>
<th>% equivalent</th>
<th>average time/medium time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top1</td>
<td>76.83%</td>
<td>76.30%</td>
<td>66</td>
</tr>
<tr>
<td>Top10</td>
<td>89.66%</td>
<td>87.35%</td>
<td>103</td>
</tr>
<tr>
<td>Top100</td>
<td>91.04%</td>
<td>88.27%</td>
<td>4,454</td>
</tr>
</tbody>
</table>
Future Work

- Focus on other types of error (place holders, broken code snippets etc..).

<table>
<thead>
<tr>
<th>Top@</th>
<th>Error code</th>
<th>freq.</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 1</td>
<td>compiler.err.cant.resolve</td>
<td>1,485,626</td>
<td>38.04%</td>
</tr>
<tr>
<td>Top 2</td>
<td>compiler.err.expected</td>
<td>1,188,663</td>
<td>30.44%</td>
</tr>
<tr>
<td>Top 3</td>
<td>compiler.err.not.stmt</td>
<td>1,236,926</td>
<td>6.58%</td>
</tr>
</tbody>
</table>

- Compilability is only a necessary but not a sufficient condition to obtain executable code
  - Automated Synthesis of **Executable Code Snippets from Q&A Sites**
    - Feedback-directed approach guided by **runtime exceptions**

- Use the [Stack Overflow](https://stackoverflow.com) code snippets for **regression testing** API libraries
CXNIPPEX tool &
dataset of 93,092 compilable code snippets are available at

http://sccpu2.cse.ust.hk/csniппекс/
BACKUP SLIDES
Dependencies Resolver

- A solution with low HD is more likely to be the correct one
- Enumerate all possible solutions and compute HD is too expensive
  - We propose a **greedy** algorithm

  Why temporary?
  
  Example
  
  ```
  ```
  is undefined.

  Either
  - `java.io.PrintWriter`
  - `scala.io.File`
  could be wrong

STEP1: compute the global frequency for each package

STEP2: For each simple name order packages by their frequency

STEP3: Refining the top solution by compilation errors