Creating and Evolving Software by Searching, Selecting, and Synthesizing Relevant Source Code

Denys Poshyvanyk¹, Mark Grechanik²,³, and Collin McMillan¹
¹Computer Science Department, College of William & Mary, ²Accenture Technology Labs, ³Computer Science Department, University of Illinois, Chicago

Abstract
When programmers develop or maintain software, they instinctively sense that there are fragments of code that other developers implemented somewhere, and thus these code fragments could be reused if found. In this paper we propose a novel solution that addresses the fundamental questions of Searching, Selecting, and Synthesizing (S³) software based on the analysis of Application Programming Interface (API) calls as units of abstractions that implement high-level concepts (e.g., the API call EncryptData implements the cryptographic concept).

Background
The three main problems inhibiting mainstream software reuse practices are how to search source code effectively, how to select retrieved code snippets from relevant retrieved applications, and how to bridge the abstraction gap between a design and low level implementations.

State-of-the-art code search engines, such as Google Code Search, match words from search queries to the identifiers or comments in open-source projects. Unfortunately, these engines provide no guarantee that found code snippets implement concepts or features described in queries.

Programmers use third-party API calls to implement high-level requirements. Rather than attempting to directly map user queries to source code elements, we aim at connecting queries to usage documentation, then documentation to API calls, and then those calls to the relevant parts of source code based on which elements contain those calls.

We propose unifying searching, selecting, and synthesizing into a single framework (the S³ architecture) based on the common abstraction and behavior-specific compositional mechanisms of software systems (e.g., API usage).

Figure 1. An overview of the S³ architecture. The S₁ component searches for relevant applications from source code repositories, S₂ selects relevant fragments of code from those applications (at varying granularity), and S₃ synthesizes those relevant fragments into the user’s code at his or her discretion.
**S\textsuperscript{2} Essentials**

As a preliminary step, we tested an implementation of the S\textsuperscript{2} component.

We focused on the official Java API documentation and 40 publicly available Java examples\textsuperscript{1}. By using the given example descriptions as an oracle for mapping the user queries to source code fragments, we were able to compute accuracy, discovery, and their harmonic mean.

\[
\text{accuracy} = 100 - n \times (\text{rank of correct result}) \\
\text{discovery} = \frac{(\text{queries with positive accuracy})}{(\text{total number of queries})}
\]

\textsuperscript{1}http://www.java2s.com/

---

**S\textsuperscript{3} Walkthrough**

In S\textsuperscript{1}, help pages are processed to associate text documentation to API calls. These are then linked to user queries with a natural language processing technique. A ranking engine combines this information with programs retrieved from a code search engine. The progress on indexing open-source software is presented in Table 1.

The code search engine chooses relevant applications from the index created by our source code crawler using the same user queries. In this way, structural and textual search methods are combined. Relevant applications are then statically analyzed to retrieve metadata. Metadata contain dataflow and dependencies among API calls.

The ranking engine melds metadata with the lists of relevant API calls. Outputted is a list of the relevant applications which use the relevant calls.

Given this set of relevant applications, the S\textsuperscript{2} component selects portions implementing functionality described by user queries.

---

**Source Code Crawler**

<table>
<thead>
<tr>
<th>Items</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Projects</td>
<td>21,934</td>
</tr>
<tr>
<td>Files</td>
<td>38,330</td>
</tr>
<tr>
<td>Files Downloaded (*.zip, etc.)</td>
<td>31,371</td>
</tr>
<tr>
<td>Files Skipped (*.exe, *.pdf, etc.)</td>
<td>6,959</td>
</tr>
<tr>
<td>GB Downloaded</td>
<td>105.62 GB</td>
</tr>
<tr>
<td>GB Skipped</td>
<td>45.71 GB</td>
</tr>
<tr>
<td>Files Indexed by Lucene</td>
<td>10,897</td>
</tr>
<tr>
<td>Java docs in index</td>
<td>100,866</td>
</tr>
</tbody>
</table>

---

**Further Information**

Visit Semeru: [http://www.cs.wm.edu/semeru/](http://www.cs.wm.edu/semeru/)

Email Denys Poshyvanyk: denys@cs.wm.edu

Email Mark Grechanik: drmark@uic.edu