Polyglot Code Finder

Jan Ehmueller
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
jan.ehmueller@student.hpi.uni-potsdam.de

Alexander Riese
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
alexander.riese@student.hpi.uni-potsdam.de

Hendrik Tjabben
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
hendrik.tjabben@student.hpi.uni-potsdam.de

Fabio Niephaus
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
fabio.niephaus@hpi.uni-potsdam.de

Robert Hirschfeld
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
hirschfeld@hpi.uni-potsdam.de

ABSTRACT

With the increasing complexity of software, it becomes even more important to build on the work of others. At the same time, websites, such as Stack Overflow or GitHub, are used by millions of developers to host their code, which could potentially be reused.

The process of finding the right code, however, is often time-consuming. In addition, the right solution may be written in a programming language that does not fit the developer's requirements. Current approaches to automate code search allow users to search for code based on keywords and transformation rules, but they are limited to one programming language.

Our approach enables developers to find code for reuse written in different languages, which is especially useful when building polyglot applications. In addition to conventional search filters, users can filter code by providing example input and expected output. Based on our approach, we have implemented a tool prototype in GraalSqueak. We evaluate both approach and prototype with an experience report.

CCS CONCEPTS

• **Software and its engineering** → **Search-based software engineering**; **Reusability**; *Automatic programming*; *Integrated and visual development environments.*

KEYWORDS

code reuse, code search, polyglot, programming experience, GraalVM

ACM Reference Format:
Jan Ehmueller, Alexander Riese, Hendrik Tjabben, Fabio Niephaus, and Robert Hirschfeld. 2020. Polyglot Code Finder. In *Companion Proceedings of the 4th International Conference on the Art, Science, and Engineering of Programming (<Programming'20> Companion), March 23–26, 2020, Porto, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3397537.3397559

1 INTRODUCTION

In many cases programming is about finding the right building blocks and putting them together. Douglas Crockford underlines this by calling code reuse the "Holy Grail of Software Engineering" [1].

At the same time, the existing ecosystem of software, such as the underlying programming language, often limits the search space for potential building blocks. With the advent of polyglot runtime environments, such as GraalVM [11], mechanisms for language interoperability become more accessible. We believe that this opens up new possibilities for reusing code snippets between languages. Even if the user is not familiar with the programming language in which the desired functionality is written, we argue that experienced programmers can read and adjust the code to make it fit their needs. In combination with available code sources from the web, ranging from question-answering sites, such as Stack Overflow[1], to code hosting sites, such as GitHub[2] and GitLab[3], the user can choose from a large amount of reusable code to find a code snippet with the desired functionality.

To guide the user to the right code for reuse, we propose an approach for a tool that can be embedded into an Integrated Development Environment (IDE) and provides search options across different online code sources. The integration into the IDE workflow reduces context switches, for example, between the IDE and a web browser.

Our approach allows for keyword searches with different filter options that return the search result in the form of code snippets. Furthermore, those code snippets can be filtered by testing and evaluating them in an isolated sandbox. This evaluation step asserts that a code snippet produces a specific output value for a given set of input values, both provided by the user.

Contributions. In this work, we demonstrate the integration of a web-based code search tool into a polyglot IDE. Our contributions are as follows:

[1] https://stackoverflow.com (accessed 2020-01-20)
[2] https://github.com (accessed 2020-01-20)
[3] https://gitlab.com (accessed 2020-01-20)

(1) We present an approach that reduces context switches needed when searching for code on the web and thus increases a developer's productivity.
(2) We explain how we have built the Polyglot Code Finder, an implementation of our approach, and discuss challenges and its limitations.
(3) We evaluate our work in an experience report, in which we solve an example workflow. In this workflow, we demonstrate how our tool allows us to find reusable code for a data analysis task without having to leave the IDE.

Outline. First of all, we give an overview of the background and context in Section 2. Then, Section 3 describes the concept of web-based code search and polyglot development. Section 4 further presents an implementation of this concept for a polyglot IDE called Polyglot Code Finder. Section 5 demonstrates the Polyglot Code Finder's usage in an experience report and provides some limitations and challenges of polyglot programming that we encountered during our work. Afterwards, Section 6 gives an overview of related work in the field. Finally, Section 7 concludes this work and presents avenues for future improvements and tasks.

2 BACKGROUND/CONTEXT

Within the scope of this work, we developed a prototype on top of GraalVM. GraalVM is a virtual machine that provides interoperability between different programming languages and allows developers to combine libraries and frameworks across languages. Among others, Ruby [10], Python[4], R[5], and Squeak/Smalltalk [4] are supported languages. Each of these languages is implemented in Truffle, GraalVM's language implementation framework. This framework defines interfaces for language constructs that should be accessible across different languages. It further enables languages to send messages beyond language borders. To access the polyglot functionality within a language, special built-in methods are added. Besides running code within the context of a different language, access to a shared object store is provided as well.

From the supported languages, Squeak/Smalltalk takes a unique position. Through the tight integration between language and IDE, it is particularly suitable for tool development. The programming system shipping with GraalSqueak, the Squeak/Smalltalk implementation for GraalVM, comes with different tools, which support multiple languages. This includes a polyglot code editor as well as a polyglot notebook system [6]. Thus, GraalSqueak can be used as a polyglot IDE.

3 APPROACH

We strived towards embedding a code search tool, which we call Polyglot Code Finder, into a polyglot IDE. Figure 1 shows that it can be accessed by other IDE-inclusive programming tools to query code snippets and develop polyglot applications. The code snippets are taken from online code sources. We formulate the following four criteria that an online code source must meet to be suitable for such a tool. A code source should be *well-maintained*, *accepted*, *universal*, and *queryable*:

[4]https://github.com/graalvm/graalpython (accessed 2020-01-20)
[5]https://github.com/oracle/fastr (accessed 2020-01-20)

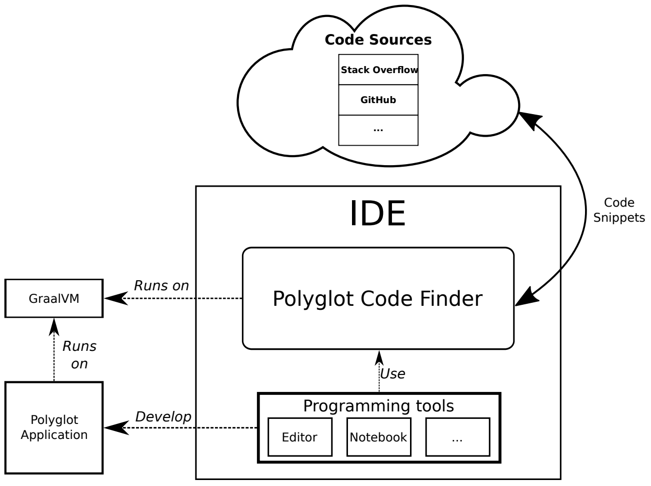


Figure 1: The Polyglot Code Finder's components and the embedment into its environment.

(1) A well-maintained code source increases the chance of finding executable code by promoting work that helps with solving a problem and rejecting unsuited solutions.
(2) A code source is accepted if it has been established in the community and people rely on its service, for example, because it hosts software dependencies.
(3) A universal code source is a service that supports content for different programming languages and paradigms.
(4) Queryable, in the context of code sources, means that the code source provides some form of public Application Programming Interface (API) that allows third parties to search for code snippets.

To further sift out solutions that actually solve a specific problem, retrieved code snippets are statically analyzed towards the following properties:

(1) Whether the code snippet is syntactically valid regarding the considered programming language.
(2) If the code snippet is unstructured or wrapped in some structure, such as a function or a class.
(3) The number of parameters.

We propose possibilities to search for code snippets based on specific method parameters and return values. We further aim at making the input and output instances freely selectable by users. In contrast to full text search, validating code snippets based on inputs and outputs allows users to find useful solutions more quickly. Therefore, every code snippet that passes the static analysis is executed and validated in a sandbox provided by the runtime. Using such a sandbox ensures security by limiting execution permissions and easing the process of monitoring the execution.

The execution of the code snippets is also monitored regarding performance. Thus, we allow the user to rank the results by different measures, such as the performance of the snippet or the complexity (i.e., lines of code).

4 IMPLEMENTATION

In this section, we describe the Polyglot Code Finder, an implementation of our approach. It is available on GitHub[6].

We chose GraalSqueak as an IDE for four reasons. First of all, because it is running on GraalVM, it allows the execution of code in a polyglot runtime environment. On the contrary, general-purpose IDEs like IntelliJ IDEA[7] and Eclipse[8] do not support a polyglot code execution out-of-the-box.

Secondly, GraalSqueak itself can be extended in a polyglot manner unlike, for example, IntelliJ IDEA, whose plugins are limited to JVM languages, such as Java or Kotlin. The polyglot approach allows us to separate tasks so that they can be implemented in the language that is most fitting for each task.

Thirdly, GraalSqueak already provides a polyglot notebook system and a polyglot code editor that can be integrated in the Polyglot Code Finder's workflow. The notebook system is similar to Jupyter[9] notebooks, but is implemented in Squeak/Smalltalk and comes with polyglot support. Our Polyglot Code Finder adds a web-based search to this notebook system and enables users to add snippets directly as cells into the notebook. This combines the existing block-based workflow of the notebook with the possibility to easily reuse existing blocks in any of the supported languages. The polyglot code editor supports a user in implementing polyglot programs by integrating GraalVM's polyglot API. For example, it generates boilerplate code necessary to import and export variables to and from the different GraalVM languages.

Finally, unlike many other GraalVM languages, most of GraalSqueak's Polyglot API is implemented on the language level rather than on the level of the VM. Therefore, many low-level primitives can be accessed directly, which allows the implementation of language-agnostic tools. At the same time, Squeak/Smalltalk is known to provide fast feedback loops and a framework for rapid tool building.

We decided to implement the Polyglot Code Finder in a polyglot manner, which enabled us to choose the best-suited language for each of the Polyglot Code Finder's tasks. This is especially important considering that some of GraalVM's language implementations are still under development and thus differ in terms of how complete they are. Additionally, we use some features that are not yet part of the polyglot API, which would make them available in each language, but that are already added to GraalVM and thereby usable in Java. We access those features in the languages that are most suited for them.

Figure 2 displays the modules of our Polyglot Code Finder, how they communicate on an abstract level, and in which languages they are implemented.

First, a user has to enter a plain-text query (e.g., "quick sort") and click on the search button. From this query we create an object that knows how to use the API of its code source. The response is parsed and stored in our code snippet data model. The resulting code snippets are then filtered in two steps. In the first step, each snippet is processed by a code highlighter to find syntax errors. If such an error is detected in the code snippet, it is excluded from the result

[6]https://github.com/hpi-swa-lab/px20-code-finder
[7]https://www.jetbrains.com/idea/
[8]https://www.eclipse.org/ide/
[9]https://jupyter.org/ (accessed 2020-01-20)

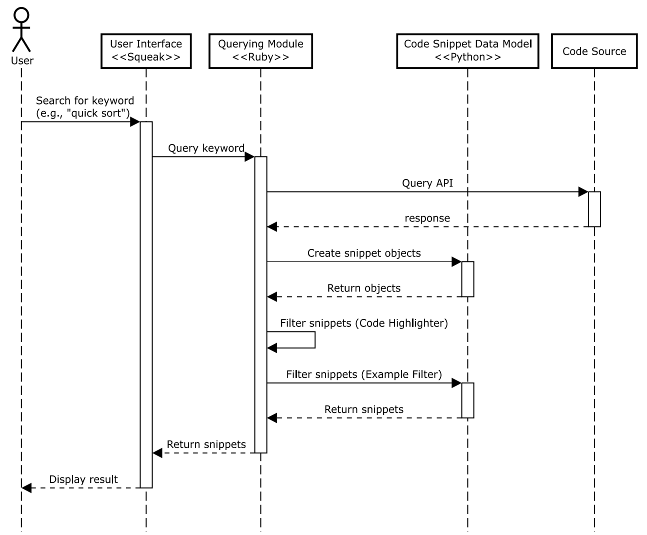

User
User Interface <<Squeak>>
Querying Module <<Ruby>>
Code Snippet Data Model <<Python>>
Code Source
Search for keyword (e.g., "quick sort")
Query keyword
Query API
response
Create snippet objects
Return objects
Filter snippets (Code Highlighter)
Filter snippets (Example Filter)
Return snippets
Return snippets
Display result

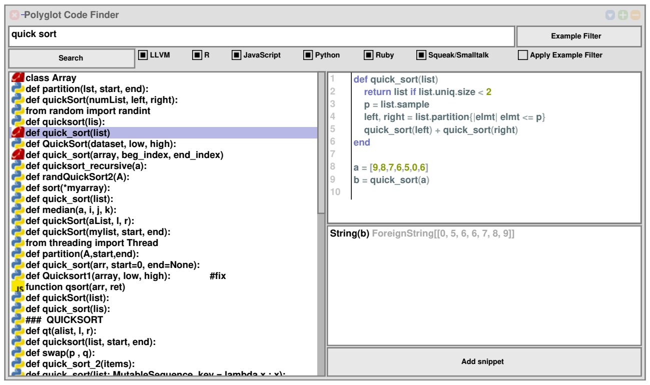
Figure 2: The modules of the Polyglot Code Finder and how they communicate.

and not shown to the user. In the second step, the code snippets are further analyzed statically. This allows us to remove responses that contain unwanted elements, such as output from an interactive shell session or just unstructured code. In an optional third step, potential code snippets are then executed with the given input values and validated against the expected output value if the user has set the Example Filter. For this, the code finder leverages GraalVM's sandboxing feature which is available through its `Context` API[10], for example, to disallow access to the file system. Finally, the remaining results are displayed in the user interface (UI).

Figure 3 shows the results of the query "quick sort". On the top, the query field, the language selection to select which languages should be considered for the results, the search button, and a button to open the Example Filter are located. On the left, the result code snippets are listed with their respective language and their first line of code. On the right, the full source code of the selected code snippet is displayed. In this view, the code snippet can also be edited. Below that, there is a workspace in which the selected snippet can be used and executed with custom code. In the example case, Ruby code is displayed which prints the result from the example that is already given in the selected code snippet. At the lower edge is a button that allows the user to export the snippet. If the Polyglot Code Finder was opened via a tool like the polyglot notebook, this button adds the source code, the answer URL, the author URL, and the license information of the selected code snippet as a new notebook cell. If, on the other hand, the Polyglot Code Finder was opened by itself, the button copies that information to the clipboard instead.

Figure 4 displays the results of the same query as before, but this time with the Example Filter enabled as well. With this query the number of results is much smaller. However, all of these code snippets were validated to perform the desired transformation,

[10]https://git.io/JvUCq (accessed 2020-01-20)



quick sort

[✓ LLVM] [✓ R] [✓ JavaScript] [✓ Python] [✓ Ruby] [✓ Squeak/Smalltalk] [ ] Apply Example Filter

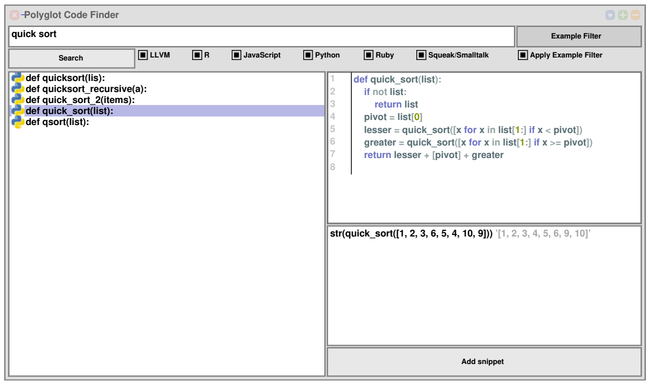
```
class Array
def partition(lst, start, end):
def quickSort(numList, left, right):
from random import randint
def quicksort(lis):
def quick_sort(list)
def QuickSort(dataset, low, high):
def quick_sort(array, beg_index, end_index)
def quicksort_recursive(a):
def randQuickSort2(A):
def sort(*myarray):
def quick_sort(list):
def median(a, i, j, k):
def quickSort(aList, l, r):
def quickSort(mylist, start, end):
from threading import Thread
def partition(A,start,end):
def quick_sort(arr, start=0, end=None):
def Quicksort1(array, low, high):            #fix
function qsort(arr, ret)
def quickSort(list):
def quick_sort(lis):
###  QUICKSORT
def qt(alist, l, r):
def quicksort(list, start, end):
def swap(p , q):
def quick_sort_2(items):
def quick_sort(list: MutableSequence, key = lambda x : x):
```

```
def quick_sort(list)
  return list if list.uniq.size < 2
  p = list.sample
  left, right = list.partition{|elmt| elmt <= p}
  quick_sort(left) + quick_sort(right)
end

a = [9,8,7,6,5,0,6]
b = quick_sort(a)
```

```
String(b) ForeignString[[0, 5, 6, 6, 7, 8, 9]]
```

Figure 3: Results of an example query in the Polyglot Code Finder.



quick sort

[✓ LLVM] [✓ R] [✓ JavaScript] [✓ Python] [✓ Ruby] [✓ Squeak/Smalltalk] [✓ Apply Example Filter]

```
def quicksort(lis):
def quicksort_recursive(a):
def quick_sort_2(items):
def quick_sort(list):
def qsort(list):
```

```
def quick_sort(list):
  if not list:
    return list
  pivot = list[0]
  lesser = quick_sort([x for x in list[1:] if x < pivot])
  greater = quick_sort([x for x in list[1:] if x >= pivot])
  return lesser + [pivot] + greater
```

```
str(quick_sort([1, 2, 3, 6, 5, 4, 10, 9])) '[1, 2, 3, 4, 5, 6, 9, 10]'
```

Figure 4: Results of the example query filtered with the Example Filter.

which, in this case, was sorting a list. Hence, all displayed results will work as expected, at least for the provided example. Additionally, it becomes evident that the input and output values were provided in Python, since we can only see Python results. The reason for this is discussed in Section 5.

5 EXPERIENCE REPORT

This section describes our experience when using the Polyglot Code Finder to enhance the experience of polyglot notebooks. In our example workflow, we analyze data from the Stack Overflow Developer Survey 2019[11] and visualize how many participants there are from each country. This example is also recorded in a video available on YouTube[12].

First, we need to download the data. Using the Polyglot Code Finder to look up how to download files from the web, we found a well-suited solution in Ruby. Having done that, we see that the downloaded file is a zip archive. We use our tool to find a way to unzip that archive in-memory, which also happens to be Ruby.

In the decompressed archive we detect not only contains CSV-files, but also other files. We want to extract the right CSV-file and simultaneously use this step to clean that data using regular expressions. At this point, we only keep the first 1000 lines of the data file for demonstration purposes. Then, not remembering the exact syntax for regular expressions in Ruby, we use our tool to look it up. Having cleaned the files, we now want to parse the CSV-file containing the actual data.

We use the Polyglot Code Finder to find out how to do exactly that and find a working solution in Python. Since we are working in a polyglot context, we can easily switch programming languages after exporting the CSV-files' contents with the polyglot API. Afterwards, we are switching back to Ruby to use its powerful collection API to extract the columns that we need.

We select the "Country" column and aggregate on it to find out how many participants there are for each country. For the visualization we only want to see the top ten countries, since there are too many countries otherwise.

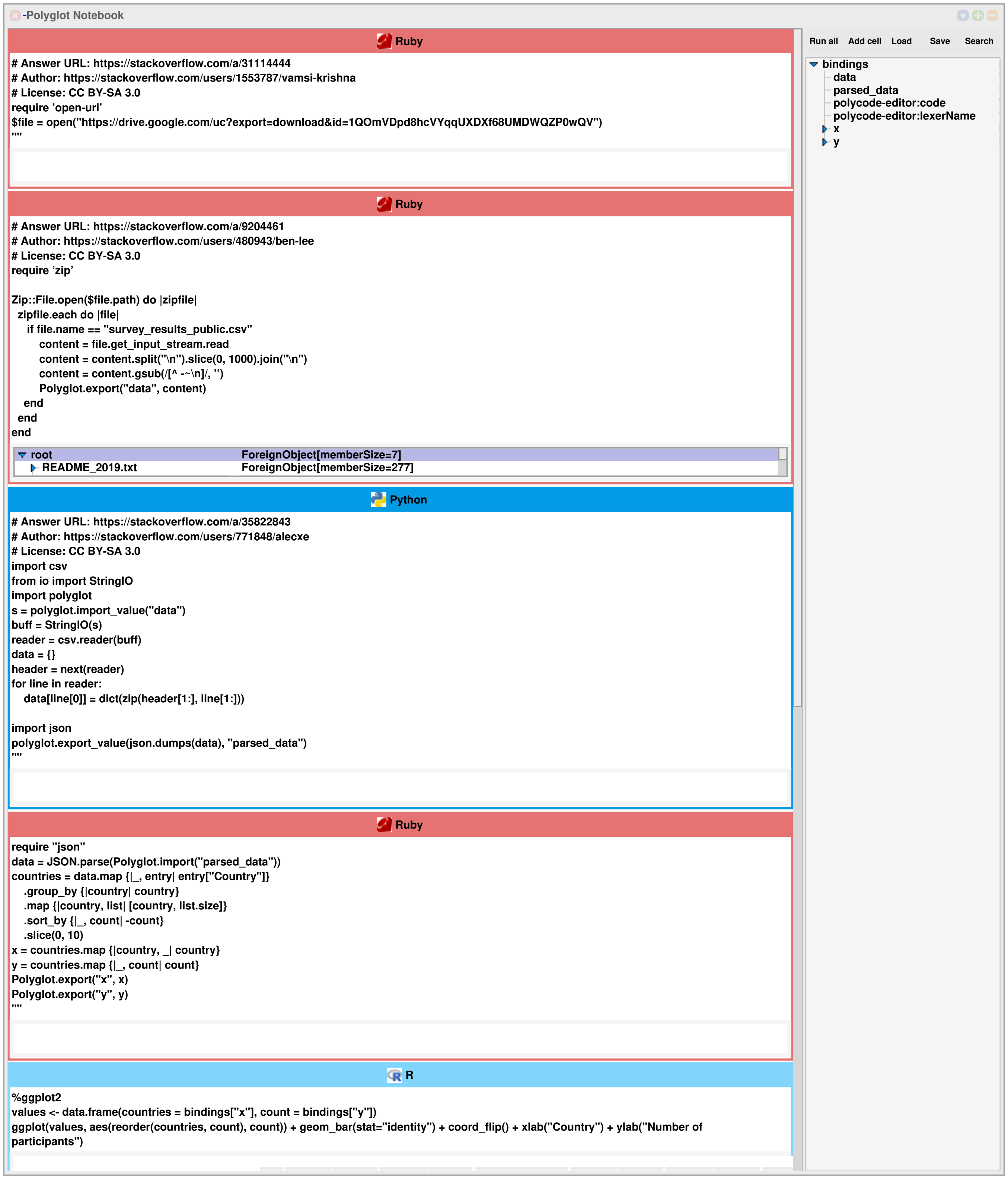
The last step is the data visualization, which we decided to do in R with ggplot2. Thus, we need to export the data via the polyglot API again. The polyglot notebook system supports a `%ggplot2` command, which handles the module import and shortens the syntax to display the plot. We use this command as well as the Polyglot Code Finder to find out how to visualize our data and how to tune the fine details, such as flipping the coordinates and setting the label for an axis. Figure 5 shows the finished notebook that we created in this report.

Limitations. Some limitations of the Polyglot Code Finder become clear in the example workflow. First, we are currently only using Stack Overflow as a code source since the APIs of other sites, such as GitHub and GitLab, do not provide a global search and thus do not meet our criteria described in Section 3. Thus, our results depend on the richness of Stack Overflow, which provides a lot of examples for commonplace languages like Python and JavaScript, but leads to limited results for languages such as Smalltalk and R. Additionally, results are not always pure functions, but sometimes written as unstructured code or entire classes. This makes the results harder to test and reuse in a different context. In our example workflow, this was not a problem due to the nature of notebooks, but in a more conventional software architecture this would be more problematic. Unstructured code, which was, for instance, recorded from an interactive shell session, is more difficult to test with our Example Filter, because there are no well-defined inputs and outputs. Furthermore, some queries will not produce function results that can be plugged into a different project immediately.

Another limitation are language specific details that affect the validity of code snippets. For instance, various results for Python might be written in different language versions which complicates interoperability. In our example workflow, we observed a CSV-parsing code snippet written in Python 2 trying to import a class that is in a different module in Python 3. Since GraalPython is a Python 3.7 implementation, that snippet did not work even though it is a valid solution for Python 2. This is not immediately obvious from the metadata, but becomes obvious when trying to use the code.

Finally, the current interoperability between the different GraalVM languages is still under active development. This means, that implementations for a sorting algorithm in Ruby do not yet necessarily work with list objects from other languages, such as Python. One

[11] https://insights.stackoverflow.com/survey/2019 (accessed 2020-01-20)
[12] https://www.youtube.com/watch?v=r_xesEExHno



Ruby

```
# Answer URL: https://stackoverflow.com/a/31114444
# Author: https://stackoverflow.com/users/1553787/vamsi-krishna
# License: CC BY-SA 3.0
require 'open-uri'
$file = open("https://drive.google.com/uc?export=download&id=1QOmVDpd8hcVYqqUXDXf68UMDWQZP0wQV")
""
```

Ruby

```
# Answer URL: https://stackoverflow.com/a/9204461
# Author: https://stackoverflow.com/users/480943/ben-lee
# License: CC BY-SA 3.0
require 'zip'

Zip::File.open($file.path) do |zipfile|
  zipfile.each do |file|
    if file.name == "survey_results_public.csv"
      content = file.get_input_stream.read
      content = content.split("\n").slice(0, 1000).join("\n")
      content = content.gsub(/[^ -~\n]/, '')
      Polyglot.export("data", content)
    end
  end
end
```

```
root                    ForeignObject[memberSize=7]
  README_2019.txt       ForeignObject[memberSize=277]
```

Python

```
# Answer URL: https://stackoverflow.com/a/35822843
# Author: https://stackoverflow.com/users/771848/alecxe
# License: CC BY-SA 3.0
import csv
from io import StringIO
import polyglot
s = polyglot.import_value("data")
buff = StringIO(s)
reader = csv.reader(buff)
data = {}
header = next(reader)
for line in reader:
    data[line[0]] = dict(zip(header[1:], line[1:]))

import json
polyglot.export_value(json.dumps(data), "parsed_data")
""
```

Ruby

```
require "json"
data = JSON.parse(Polyglot.import("parsed_data"))
countries = data.map {|_, entry| entry["Country"]}
    .group_by {|country| country}
    .map {|country, list| [country, list.size]}
    .sort_by {|_, count| -count}
    .slice(0, 10)
x = countries.map {|country, _| country}
y = countries.map {|_, count| count}
Polyglot.export("x", x)
Polyglot.export("y", y)
""
```

R

```
%ggplot2
values <- data.frame(countries = bindings["x"], count = bindings["y"])
ggplot(values, aes(reorder(countries, count), count)) + geom_bar(stat="identity") + coord_flip() + xlab("Country") + ylab("Number of
participants")
```

Figure 5: A polyglot notebook that analyzes the Stack Overflow Developer Survey from 2019 and plots the top ten countries with the most participants.

of the reasons is that such an algorithm might use Ruby specific methods, such as `uniq`, to sort the list. If that method is called on a Python list, that object would not understand the message, because the method does not exist in the Python world. One possible solution for this could be polyglot adapters [5]. These would map, for example, the `uniq` method to a Python method that does the same thing on the Python list. This limitation is one of GraalVM itself and not of the Polyglot Code Finder, however it becomes apparent when using the tool.

6 RELATED WORK

In the following, we discuss related work.

How Developers Search Code. Sadowski et. al conducted a case study at Google on when and how developers search for code online [8]. Their results show that searching for information on the web is a key software development activity and an essential part of a programmer's every-day life. On average, developers perform 12 search queries every weekday. Based on these insights, various requirements for future code search techniques and tools are formulated. One such requirement is that in order to facilitate the search process, tools should be integrated into the development environment and provide minimal yet informative results.

Internet-Scale Code Search. Gallardo-Valencia et al. claim that developers like to build software composed of code that they have found online in open-source software repositories [2]. The authors aggregate the process of searching for software development solutions online under the term "Internet-Scale Code Search". They argue that Internet-Scale Code Search has many similarities with other research areas such as software reuse or code search, but note that there are currently no novel solutions to facilitate such a search.

IDE-embedded web browsers. Developers often use a web browser for searching code online. For this reason, some IDEs, such as Eclipse[13] or IntelliJ IDEA[14], come with an embedded web browser. Unlike our approach, an integrated web browser allows users to browse the web freely, but is usually not optimized for finding reusable code.

Method Finder. Squeak/Smalltalk's Method Finder [9] is a tool that helps to find the right method in a Squeak/Smalltalk image. It supports a keyword and an example-based search. While the keyword search allows users to find methods based on their name, the example-based search allows them to find methods based on how they transform a specific input to a specific output. However, the Squeak/Smalltalk internal Example Filter is limited to methods that are written in Smalltalk and that are present in the current Squeak/Smalltalk image. Our approach searches for code written in arbitrary programming languages online using both keyword-based and example-based search.

[13]https://git.eclipse.org/c/platform/eclipse.platform.ui.git/tree/bundles/org.eclipse.ui.browser (accessed 2020-01-20)
[14]https://plugins.jetbrains.com/plugin/10750-embedded-web-browser-for-idea (accessed 2020-01-20)

Stack Overflow Importer. Stack Overflow Importer[15] is a Python module that can be used to import certain algorithms directly from Stack Overflow. Similar to our approach, Stack Overflow Importer implements this algorithm by querying Stack Overflow's public API and by searching for implementations written in Python. Afterwards, the retrieved code snippets are ranked by their number of upvotes and are validated based on syntactic correctness. This validation process, however, does not ensure that a retrieved code snippet provides the desired result. Our Polyglot Code Finder, with the integrated Example Filter, goes one step further and validates that each code snippet transforms an input into the desired output.

StackInTheFlow. StackInTheFlow by Greco et al. is a tool that queries code snippets by utilizing Stack Overflow's API [3]. Queries can either be manually constructed or automatically assembled based on the current source code context. For example, the tool automatically detects compiler and runtime errors and queries Stack Overflow to find solutions for these errors. A recommendation system, which is based on Stack Overflow tags, personalizes results over time. The tool is available as a plugin for the IntelliJ IDE family. Similar to our approach, StackInTheFlow is integrated into the working environment and queries Stack Overflow' API. It further applies mechanisms for automatically detecting errors and personalizing results. StackInTheFlow also provides static analysis metrics, such as the post's publishing date and number of votes. However, our Polyglot Code Finder, and more specifically its Example Filter, can execute code snippets in a sandbox to test them for expected behavior. Additionally, this execution can also be used to generate additional metrics, for example to assess their runtime performance.

Reiss presents an approach for finding and validating solutions for development problems [7]. It consists of the following steps. First, a keyword based search is done to get possible solutions. Afterwards, each possible solution is validated with regard to static specifications, such as a natural language description, and dynamic checks, such as contracts and test cases. Contracts are defined as preconditions and postconditions, while test cases are provided in the form of input-output pairs. Finally, each potential solution is examined regarding further dimensions such as security, complexity, or performance. Our approach follows the steps proposed by Reiss to facilitate Internet-Scale Code Search. But in contrast to Reiss' web tool, our approach aims to embed a code search tool directly into a polyglot IDE.

7 CONCLUSION AND FUTURE WORK

In this work, we propose an approach and implementation of a web-based code search tool that is integrated into a polyglot IDE. By using this tool, developers can find reusable code from online resources that fit their needs. The code search can be done without leaving the IDE and across programming languages. To enhance the search experience, we provide both a keyword-based search and a search based on input to output transformation rules. Latter is achieved by automatically executing and validating candidate code snippets in a secure sandbox environment by specifying expected input and output values. The main advantages of this approach

[15]https://github.com/drathier/stack-overflow-import (accessed 2020-01-20)

are that code snippets can be queried from within the IDE and automatically filtered based on the developer's needs.

As part of this work, we used our tool to solve an example workflow and reported the experience.

For further research, a more sophisticated evaluation approach, such as an empirical user study, is possible. In addition, some code snippets need to be adjusted to specific use-cases. This can be achieved by automatically encapsulating unstructured code snippets in functions, which could be done by analyzing a code snippet's abstract syntax tree. Additionally, the number of results when using the Example Filter can be increased by adding interface mapping, such as polyglot adapters [5]. This mapping would allow code snippets that use language-specific methods to properly work with objects of different languages.

Furthermore, the Polyglot Code Finder could support specific querying for different user groups. For example, for students that are learning how to code, only parts of a code snippet could be displayed. This would help students in solving a problem, but still require them to come up with the missing parts themselves. Another example would be domain experts using the Polyglot Code Finder to query a domain-specific code source, such as machine learning specific code.

Finally, the integration of statically-typed languages, such as languages based on LLVM, provides challenges. Although it is possible to execute LLVM bytecode on the GraalVM, relevant code snippets are provided in the corresponding programming languages and require further processing.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of HPI's Research School[16] and the Hasso Plattner Design Thinking Research Program[17].

[16]https://hpi.de/en/research/research-school.html
[17]https://hpi.de/en/dtrp/

REFERENCES

[1] Douglas Crockford. 2005. *The Elements of JavaScript Style.* https://crockford.com/javascript/style1.html

[2] R. E. Gallardo-Valencia and S. Elliott Sim. 2009. Internet-Scale Code Search. In *2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation.* 49–52. https://doi.org/10.1109/SUITE.2009.5070022

[3] Chase Greco, Tyler Haden, and Kostadin Damevski. 2018. StackInTheFlow: Behavior-Driven Recommendation System for Stack Overflow Posts. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (Gothenburg, Sweden) *(ICSE '18).* Association for Computing Machinery, New York, NY, USA, 5–8. https://doi.org/10.1145/3183440.3183477

[4] Fabio Niephaus, Tim Felgentreff, and Robert Hirschfeld. 2019. GraalSqueak: Toward a Smalltalk-based Tooling Platform for Polyglot Programming. In *Proceedings of the 16th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes* (Athens, Greece) *(MPLR 2019).* ACM, New York, NY, USA, 14–26. https://doi.org/10.1145/3357390.3361024

[5] Fabio Niephaus, Tim Felgentreff, and Robert Hirschfeld. 2019. Towards Polyglot Adapters for the GraalVM. In *Proceedings of the Conference Companion of the 3rd International Conference on Art, Science, and Engineering of Programming* (Genova, Italy) *(Programming '19).* Association for Computing Machinery, New York, NY, USA, Article 1, 3 pages. https://doi.org/10.1145/3328433.3328458

[6] Fabio Niephaus, Eva Krebs, Christian Flach, Jens Lincke, and Robert Hirschfeld. 2019. PolyJuS: A Squeak/Smalltalk-based Polyglot Notebook System for the GraalVM. In *Proceedings of the Conference Companion of the 3rd International Conference on Art, Science, and Engineering of Programming* (Genova, Italy) *(Programming '19).* ACM, New York, NY, USA, Article 24, 6 pages. https://doi.org/10.1145/3328433.3328434

[7] Steven P. Reiss. 2009. Semantics-based Code Search. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09).* IEEE Computer Society, Washington, DC, USA, 243–253. https://doi.org/10.1109/ICSE.2009.5070525

[8] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How Developers Search for Code: A Case Study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015).* Association for Computing Machinery, New York, NY, USA, 191–201. https://doi.org/10.1145/2786805.2786855

[9] Squeak/Smalltalk Community. 2019. *Method Finder.* https://wiki.squeak.org/squeak/1916

[10] Thomas Würthinger, Christian Wimmer, Christian Humer, Andreas Wöundefined, Lukas Stadler, Chris Seaton, Gilles Duboscq, Doug Simon, and Matthias Grimmer. 2017. Practical Partial Evaluation for High-Performance Dynamic Language Runtimes. *SIGPLAN Not.* 52, 6 (June 2017), 662–676. https://doi.org/10.1145/3140587.3062381

[11] Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko. 2013. One VM to Rule Them All. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Indianapolis, Indiana, USA) *(Onward! 2013).* ACM, New York, NY, USA, 187–204. https://doi.org/10.1145/2509578.2509581