New Frontiers of Mining Software Repositories

**Usability** and **Information Delivery**

Miryung Kim
University of California, Los Angeles
Why software specification inference?
An API is the interface to implemented functionality that developers can access to perform various tasks. APIs support code reuse, provide high-level abstractions that facilitate programming tasks, and help unify the programming experience (for example, by providing a uniform way to interact with list structures). However, APIs have grown very large and diverse, which has prompted some to question their usability.

It would be a pity if the difficulty of using APIs would nullify the productivity gains they offer. To ensure that this doesn't happen, we need to know what makes APIs hard to learn.

Common sense indicates that an API’s structure can impact its usability (see the “API Usability” sidebar). This intuition is reflected by efforts to flesh out sound design principles for APIs and empirical studies on the impact of design structure on API usability.

However, APIs don’t exist in isolation, and other factors can also affect how developers experience them. So, what exactly does make an API hard to learn?

To answer this question, I investigated the obstacles professional developers at Microsoft faced when learning how to use APIs. As opposed to previous API usability studies that focused on specific design aspects, I used an approach completely grounded in developers’ experience. By surveying and interviewing developers about the obstacles they faced learning APIs, I discovered many issues that complement those mentioned in API design textbooks and articles. In particular, I found that API learning resources are critically important when considering obstacles to learning the API, and as worthy of attention as the structural aspects of the API. I also elicited specific relationships between resources and API usage that API designers and documentation writers shouldn’t overlook when designing API documentation.

First, information about the high-level design of the API is necessary to help developers choose among alternative ways to use the API, to structure their code accordingly, and to use the API as efficiently as possible. Second, code examples can become more of a hindrance than a benefit when there’s a mismatch between the tacit purpose of the example and the goal of the example user. Finally, some design decisions can influence the behavior of the API in subtle ways that confuse developers.

Survey Design

In February and March 2009, I conducted a survey to gather information about developers’ experiences learning APIs. Specifically, I sought to identify areas of concern and themes worthy of attention.

Most software projects reuse components exposed through APIs. In fact, current-day software development technologies are becoming inseparable from the large APIs they provide. To name two prominent examples, both the Java Software Development Kit and the .NET framework ship with APIs comprising thousands of classes supporting tasks that range from reading files to managing complex process workflows.

What Makes APIs Hard to Learn? Answers from Developers

Martin P. Robillard, McGill University

A study of obstacles that professional Microsoft developers faced when learning to use APIs uncovered challenges and...
Response categories for API learning obstacles

<table>
<thead>
<tr>
<th>Main category</th>
<th>Subcategories/descriptions</th>
<th>Associated respondents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources</td>
<td>Obstacles caused by inadequate or absent resources for learning the API (for example, documentation)</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>General</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Content</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Task</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Format</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Design</td>
<td>8</td>
</tr>
<tr>
<td>Structure</td>
<td>Obstacles related to the structure or design of the API</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Design</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Testing and debugging</td>
<td>10</td>
</tr>
<tr>
<td>Background</td>
<td>Obstacles caused by the respondent’s background and prior experience</td>
<td>17</td>
</tr>
<tr>
<td>Technical environment</td>
<td>Obstacles caused by the technical environment in which the API is used (for example, heterogeneous system, hardware)</td>
<td>15</td>
</tr>
<tr>
<td>Process</td>
<td>Obstacles related to process issues (for example, time, interruptions)</td>
<td>13</td>
</tr>
</tbody>
</table>
Learning Barriers for Developers
Six Learning Barriers in End-User Programming Systems

Andrew J. Ko, Brad A. Myers, and Htet Htet Aung
Human-Computer Interaction Institute
Carnegie Mellon University, Pittsburgh, PA 15213 USA
ajko@cmu.edu, bam+@cs.cmu.edu, hha@cs.cmu.edu

Abstract
As programming skills increase in demand and utility, the learnability of end-user programming systems is of utmost importance. However, research on learning barriers in programming systems has primarily focused on languages, overlooking potential barriers in the environment and accompanying libraries. To address this, a study of beginning programmers learning Visual Basic.NET was performed. This identified six types of barriers: design, selection, coordination, use, understanding, and information. These barriers inspired a new metaphor of computation, which provides a more learner-centric view of programming system design.

1. Introduction
According to the U.S. Department of Labor, by 2012 30% of new jobs and nearly 8% of all U.S. jobs could require programming skills [1]. This is a dramatic shift for a skill that less than a million people had 10 years ago. Now, an increasing number of end-user programmers control manufacturing robots, create spreadsheets, and design interactive prototypes. Yet, for such growth to occur, millions of aspiring end-user programmers must overcome substantial learning barriers in programming systems. Do we know enough about these barriers to design systems that help these individuals? We know much about the learning barriers in programming languages [11], but little about the rest of a programming system, which includes its environment (the editor, debugger, help, etc.) and accompanying libraries. What barriers do these parts of a programming system pose, if any? In this paper, we answer this question both empirically and metaphorically. We begin by describing a study of Visual Basic.NET (VB), which identified six types of learning barriers. We then discuss several implications and describe a new metaphor of computation that facilitates a more learner-centric view of programming system design.

2. Prior Research on Learning Barriers
One way to understand learning barriers is to study the learner. For example, imagine Jill, a user interface designer who just began learning VB. Shortly after starting, she realizes that she must learn about event handlers to proceed. This poses a potential learning barrier. From an attention-investment perspective [2], she will weigh the cost, risk, and reward of overcoming the barrier, and if the risk of failure outweighs the reward, she is likely to abandon VB for other tools. Jill may also decide that progress is worth the risk of failure. We have proposed a framework that suggests she will make several simplifying assumptions about VB's language, environment, and libraries in trying to acquire the necessary knowledge [8]. If her assumptions are valid with respect to the programming system, she will make progress. If her assumptions are invalid—what we call knowledge breakdowns—she is likely to make an error. Within this framework, we define learning barriers as aspects of a programming system or problem that are prone to such invalid assumptions. These concepts are depicted in Figure 1.

Figure 1. In overcoming barriers, learners risk making invalid assumptions that often lead to error.
## Six Learning Barriers for Developers

<table>
<thead>
<tr>
<th>Barrier Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>I don’t know what I want the computer to do...</td>
</tr>
<tr>
<td>Selection</td>
<td>I think I know what I want the computer to do but I don’t know what to use</td>
</tr>
<tr>
<td>Coordination</td>
<td>I think I know what things to use but I don’t know how to make them work together...</td>
</tr>
<tr>
<td>Use</td>
<td>I think I know what to use, but I don’t know how to use it...</td>
</tr>
<tr>
<td>Understanding</td>
<td>I thought I knew how to use this but it didn’t do what I expected...</td>
</tr>
<tr>
<td>Information</td>
<td>I think I know why it didn’t do what I expected but I don’t know how to check</td>
</tr>
</tbody>
</table>

*Six Learning Barriers in End-User Programming Systems, Ko et al. 2004*
## Six Learning Barriers for Developers

<table>
<thead>
<tr>
<th>Barrier Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>I don’t know what I want the computer to do...</td>
</tr>
<tr>
<td>Selection</td>
<td>I think I know what I want the computer to do but I don’t know <strong>what to use</strong>...</td>
</tr>
<tr>
<td>Coordination</td>
<td>I think I know what things to use but I don’t know <strong>how to make them work together</strong>...</td>
</tr>
<tr>
<td>Use</td>
<td>I think I know what to use, but I don’t know how to use it...</td>
</tr>
<tr>
<td>Understanding</td>
<td>I thought I knew how to use this but it didn’t do what I expected...</td>
</tr>
<tr>
<td>Information</td>
<td>I think I know why it didn’t do what I expected but I don’t know how to check</td>
</tr>
</tbody>
</table>
Rise of Mining Big Code

May 5, 2014
DARPA Launches ‘Big Code’ Initiative
George Leopold

The U.S. Defense Advanced Research Projects Agency is attempting to take big data analytics to the next level through a “big code” project designed to improve overall software reliability through a large-scale repository of software that drives big data.

The DARPA “big code” initiative, formally known as Mining and Understanding Software Enclaves, or MUSE, seeks to leverage software analysis and big data analytics to improve the way software is built, debugged and verified.
Information Delivery and Usability

1. Comprehension
2. Interactive Navigation
3. Fit Developer Workflow
Information Delivery and Usability

1. Comprehension
Not Easy to Comprehend

Graph-based Mining of Multiple Object Usage Patterns, Nguyen et al. 2009
Not Easy to Comprehend

A Graph-based Approach to API Usage Adaptation, Nguyen et al. 2010
What Makes a Good Code Example?

• “It shouldn’t model something extremely specific”
• “It must be able to show multiple uses.”
• “a good example is easy to understand and read.”
• “less irrelevant, unrelated stuff in the example is better”
• “clear naming of variables”

Synthesizing API Usage Examples, Buse and Weimer 2012
What Makes a Good Code Example?

• “It shouldn’t model something extremely specific”
• “It must be able to show multiple uses.”
• “a good example is easy to understand and read.”
• “less irrelevant, unrelated stuff in the example is better”
• “clear naming of variables”

BufferedInputStream b; //initialized previously:
ObjectInputStream stream =
    new ObjectInputStream(b);
try {
    Object o = stream.readObject();
    //Do something with o
} catch (IOException e) {
} finally {
    stream.close();
}

Synthesized Code Example

Synthesizing API Usage Examples, Buse and Weimer 2012
Information Delivery and Usability

2. Interactive Navigation
Precision and Recall are Not Enough

Mining Version Histories to Guide Software Changes, Zimmermann et al. 2004

Are Code Examples on an Online Q&A Forum Reliable? Zhang et al. 2018
Not Easy to Navigate and Compare

Mining Succinct and High-Coverage API Usage Patterns from Source Code, Wang et al. 2013
1. It's Not Easy to Visualize

MAPO: Mining and Recommending API Usage Patterns, Zhong et al. 2009

- **API of interest**
- **Clicked sequence**
- **Contexts of methods**
- **Pattern**
- **Pattern rank**
- **Whether the API is invoked in this sample**
- **Methods with the clicked sequence**
- **Methods without the clicked sequence**
- **Similarity to the current programming context**

This definition is adapted from the classical definition of frequent sequences that is used by existing frequent subsequence miners. A frequent subsequence miner automatically mines the frequent sequences whose support values are greater than a threshold.

4.3 API Usage Recommender

This section presents the mechanism of MAPO to recommend associated snippets using the mined patterns as an index. Figure 5 shows MAPO's API usage recommender, which is a plug-in that integrates with the Eclipse IDE.

Instead of requiring programmers to check the snippets one by one, the recommender provides programmers with the capability to use the mined patterns as an index to locate snippets. For example, if a programmer wants to know the usages of `appendToGroup`, they need to type in "appendToGroup" in the method body under development. After that, the programmer selects "appendToGroup" and clicks "Query API patterns" of the context menu for the usages of "appendToGroup".

Figure 5 shows an annotated screen snapshot of the preceding query. The returned relevant patterns with the pattern ranks are shown in the pattern view on the right side of Figure 5. The rank of a pattern is the average similarity of the supporting snippets to the current programming task. Here, we use the method name and the class name to calculate the similarity. For example, supposing that the programmer is implementing a method named `m` in class `c`, `m` and `c` will be compared with the method name and the class name of each supporting snippet to calculate a similarity value.

The similarity definition is the same as the one in Section 4.2. From each pattern, MAPO lists its frequent
Abstract—Using static analysis tools for automating code inspections can be beneficial for software engineers. Such tools can make finding bugs, or software defects, faster and cheaper than manual inspections. Despite the benefits of using static analysis tools to find bugs, research suggests that these tools are underused. In this paper, we investigate why developers are not widely using static analysis tools and how current tools could potentially be improved. We conducted interviews with 20 developers and found that although all of our participants felt that use is beneficial, false positives and the way in which the warnings are presented, among other things, are barriers to use. We discuss several implications of these results, such as the need for an interactive mechanism to help developers fix defects.

Why Don’t Software Developers Use Static Analysis Tools to Find Bugs?

Brittany Johnson, Yoonki Song, and Emerson Murphy-Hill
North Carolina State University
Raleigh, NC, U.S.A.
bijohnso,ysong2@ncsu.edu, emerson@csc.ncsu.edu

Robert Bowdidge
Google
Mountain View, CA, U.S.A.
bowdidge@google.com

There are many situations where a developer may consider using a static analysis tool to find defects in their code. Let us consider a developer, Susie. Susie is a software developer at a small company. She wants to make sure that she is following the company’s standards while maintaining quality code. She needs a way of checking her code in her IDE, before submitting it to the general code repository, without worrying about any outside dependencies that she has no control over. Susie decides that her best bet is to install a static analysis tool. She decides to install FindBugs because she likes the quality of the results and the fact that bugs can be found as she types; at first, she is very happy with her decision and
Disuse of Static Analysis Tools

- **Large volumes of false positives and warnings** outweigh true positives in volumes

- **Custom Navigation and Filter** Users would like to configure the ways that you see and filter results

- **Actionable Understandability** A developer not being able to understand what the tool is telling her is a barrier to use

- **Quick Fixes** “if you can tell me it’s an error, you should be able to tell me how to fix it.”

*Why Don’t Software Developers Use Static Analysis Tools to Find Bugs?* Johnson et al. 2013
Information Delivery and Usability

3. Fit Developer Workflow
Must Fit Developer Workflow

• Developers opportunistically interleave web foraging of online resources, learning, and writing code.

• Programmers search for code very frequently, conducting an average of 5 search sessions with 12 queries each workday.

• 7% of respondents reused or modified code examples from Stack Overflow daily, 40% did at least weekly, and 62% did at least monthly.

How Developers Search for Code: A Case Study, Sadowski et al. 2015
How do developers utilize source code from stack overflow? Wu et al. 2018
Information Delivery and Usability

1. Comprehension

2. Interactive Navigation

3. Fit Developer Workflow
Part 1

API Usage Mining from GitHub
API Misuse Detection in Stack Overflow [ICSE 2018]

Part 2

Visualization of Code Examples at Scale [CHI 2018]
“How do I write data to a file using FileChannel?”

You're probably interested in a `FileChannel`. Channels were designed to perform bulk IO operations to and from `Buffers`.

Ex:

```java
FileChannel fileOut = new FileOutputStream(file).getChannel();
fileOut.write(ByteBuffer.wrap("Whatever you want to write".getBytes()));
```

Actually, I want to maintain a large buffer (whose size I can mention) and periodically flush it. – Arpsss Apr 8 ’12 at 19:43

@Arpsss You can maintain a `ByteBuffer` and periodically write it to the file system. You don't have to create your `ByteBuffer` inline like that. – Jeffrey Apr 8 ’12 at 19:47

Thanks. Is it possible to use charbuffer of NIO? – Arpsss Apr 8 ’12 at 20:05

`FileChannel` cannot write a `CharBuffer`. You can, however, use `ByteBuffer#putChar` to put characters.
“How do I write data to a file using FileChannel?”

You’re probably interested in a `FileChannel`. Channels were designed to perform bulk IO operations to and from `Buffers`.

Ex:

```java
FileChannel fileOut = new FileOutputStream(file).getChannel();
fileOut.write(ByteBuffer.wrap("Whatever you want to write".getBytes()));
```

This example forgets to close the FileChannel object properly.
"How do I write data to a file using FileChannel?"

Somewhat like this:

```java
short[] payload = {1,2,3,4,5,6,7,8,9,0};
ByteBuffer myByteBuffer = ByteBuffer.allocate(20);
myByteBuffer.order(ByteOrder.LITTLE_ENDIAN);

ShortBuffer myShortBuffer = myByteBuffer.asShortBuffer();
myShortBuffer.put(payload);

FileChannel out = new FileOutputStream("sample.bin").getChannel();
out.write(myByteBuffer);
out.close();
```

This example forgets to handle potential exceptions such as IOException and FileNotFoundException.
API Usage Mining from GitHub

- We mine API usage patterns from 380K GitHub projects.

```
1
Code Search  Program Slicing  Call Sequence Extraction
380K Java Repositories on GitHub

2
Structured API call sequences
Frequent Sequence Mining

3
SMT-based Guard Condition Mining

API usage patterns
```
Insight 1: Mining a Large Code Corpus

- Our code corpus includes 380K GitHub projects with at least 100 revisions and 2 contributors.

Insight 2: Removing Irrelevant Statements via Program Slicing

- We perform backward and forward slicing to identify data- and control-dependent statements to an API method of interest.

1. Code Search
2. Program Slicing
3. Call Sequence Extraction

380K Java Repositories on GitHub

Structured API call sequences

Frequent Sequence Mining

SMT-based Guard Condition Mining

API usage patterns
void initInterfaceProperties(String temp, File dDir) {
    if (!temp.equals("props.txt")) {
        log.error("Wrong Template.");
        return;
    }
    // load default properties
    FileInputStream in = new FileInputStream(temp);
    Properties prop = new Properties();
    prop.load(in);
    ... init properties ...
    // write to the property file
    String fPath = dDir.getAbsolutePath() + "/interface.prop";
    File file = new File(fPath);
    if (!file.exists()) {
        file.createNewFile();
    }
    FileOutputStream out = new FileOutputStream(file);
    prop.store(out, null);
    in.close();
}

Data dependency up to one hop, i.e., direct dependency
void initInterfaceProperties(String temp, File dDir) {
    if (!temp.equals("props.txt")) {
        log.error("Wrong Template.");
        return;
    }
    // load default properties
    FileInputStream in = new FileInputStream(temp);
    Properties prop = new Properties();
    prop.load(in);
    ... init properties ...
    // write to the property file
    String fPath = dDir.getAbsolutePath() + "/interface.prop";
    File file = new File(fPath);
    if (!file.exists()) {
        file.createNewFile();
    }
    FileOutputStream out = new FileOutputStream(file);
    prop.store(out, null);
    in.close();
}

The focal API method
Insight 3: Capture the Semantics of API Usage

• It is important to capture the temporal ordering, enclosing control structures, and appropriate guard conditions of API calls.

```
new File (String); try { new FileInputStream(arg0.exists()); } catch (IOException); }
```
Insight 3: Capture the Semantics of API Usage

• It is important to capture the temporal ordering, enclosing control structures, and appropriate guard conditions of API calls.

```java
new File (String); try { new FileInputStream(File)@arg0.exists(); } catch (IOException); }
```
Insight 3: Capture the Semantics of API Usage

- It is important to capture the temporal ordering, enclosing control structures, and appropriate guard conditions of API calls.

```
new File (String); try { new FileInputStream(@arg0.exists()); } catch (IOException); }
```
Insight 4: Variations in Guard Conditions

- GitHub developers may write the same predicate in different ways.

Two equivalent guard conditions for `substring(int)`:

\[
arg0 \geq 0 \land arg0 \leq rcv.length() \iff arg0 > -1 \land arg0 < rcv.length() + 1
\]
Yet Another API Usage Mining Tool?
API Misuse Detection in Stack Overflow

- We examine 220K SO posts with 180 confirmed patterns.
- => 31% of SO posts contain API usage violations!

Highly-voted posts are not necessarily more reliable in terms of correct API usage.
ExampleCheck [FSE’18 Demo]
ExampleCheck
Offered by: Tianyi Zhang

Potential API Misuse

You may want to check whether the receiver of `getAsString()` is not equal to null. You may also want to handle the potential `Exception` thrown by `getAsString()` by using a `try-catch` block here. 117 GitHub code examples also do this.

```java
try {
    if (match_number!=null) {
        match_number.getAsString();
    }
} catch (Exception e) {
    // handle exception
}
```

See this in a GitHub example:
ybonnel/json
Aleks-Yahh/java-api
extery/TTRSS_android_extery
Part 1
API Usage Mining from GitHub API Misuse Detection in Stack Overflow [ICSE 2018]

Part 2
Visualization of Code Examples at Scale [CHI 2018]
Examplore: Visualizing Examples at Scale

Focal API

Many code examples using this call

label and collate into code skeleton

Interactive visualization showing common usage and frequency

Demo: http://examplore.cs.ucla.edu:3000
Mining API Usage from a Large Corpus

API call of interest

new FileInputStream()

crawl

380K GitHub repositories

Many code examples using this call

if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…
}

File file = new File(_basePath + "/" + path);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}

File propertiesFile = getPropertiesFile();
try {
    InputStream in = new FileInputStream(propertiesFile);
    workspaceProperties.load(in);
} catch (IOException e) { }
Program Slicing and Labeling

Labeled Code Examples

```java
private void getLatestVersion() {
    // TODO Auto-generated method stub
    File temp = new File(System.getenv("PRIVATE_FOLDER") + "/pdTemp/");
    try {
        List<File> listMain = IoUtil.getZipResource(new FileInputStream(pdzZipPath), temp, true);
        if (listMain.size() != 0) {
            for (File f : listMain) {
                if (f.isDirectory()) folderName = f.getName();
                if (f.getAbsolutePath().toLowerCase().contains("droidparty_main.pd")) {
                    foundMainPd = true;
                    dpMainfileName = f.getName();
                    InputStream in = new FileInputStream(f);
                    BufferedReader reader = new BufferedReader(new InputStreamReader(in));
                    String line;
                    while ((line = reader.readLine()) != null) {
                        if (line.contains("version: ")) {
                            Log.d("LatestVersionLine", line);
                            version = line.substring(line.lastIndexOf("=") + 1, line.length() - 1);
                            this.latestVersion = Float.parseFloat(version);
                            break;
                        } else {
                            version = "0.0";
                            this.latestVersion = Float.parseFloat(version);
                        }
                    }
                    reader.close();
                    Log.d("LatestVersion", latestVersion + "");
                    break;
                } else {
                    closePd();
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

[Ko et al. 2004, Duala-Ekoko & Robillard 2012]
Code Canonicalization

API Skeleton

```java
class APISkeleton {

    declarations

    try {
        pre method call
        if (...) {
            focus
            if (...) {
                post method call
            }
        }
    }

    catch (...) {
        exception handling call
    }

    finally { ... }
}
```

Labeled Code Examples

```java
File file = new File("String" + ");
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

File file = new File(_basePath + "/" + path);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
File propertiesFile = getPropertiesFile();
try {
    InputStream in = new FileInputStream(propertiesFile);
    workspaceProperties.load(in);
} catch (IOException e) {
    throw new IOException(e);
}
```

```java
new FileInputStream()
```

crawl
380K
Github repositories

Many code examples using this call
Examplore Interface

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(...);
}

File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}

File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```plaintext
InputStream stream = new FileInputStream(file);
```
```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}

File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e)
}

File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

---

**Examplore Interface**

**Abstraction**

**API Skeleton**

```
declarations
try {
    pre method call
    if ( ... ) {
        focus
    }
}
stream = new FileInputStream(file);
if ( ... ) {
    post method call
}
} catch ( ... ) {
    exception handling call
}
finally { ...
```
```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```

```java
File file = getPropertiesFile();
try {
    InputStream stream = new FileInputStream(file);
    workspaceProperties.load(stream);
} catch (IOException e) {
}
```

```java
if (file != null) {
    return new FileInputStream(file);
} else {
    return new ByteArrayInputStream(…)
}
```

```java
File file = new File(String);
try {
    return new FileInputStream(file);
} catch (FileNotFoundException e) {
    throw new IllegalArgumentException(e);
}
```

```java
try {
    if (file != null) {
        return new FileInputStream(file);
    } else {
        return new ByteArrayInputStream(…)
    }
}
```
Examplore Interface
Examplore Interface

Tool is available at [http://examplore.cs.ucla.edu:3000/](http://examplore.cs.ucla.edu:3000/)
Lab Study Results

- Examplore users investigated many relevant examples.
- Baseline users often answered based on one example or by guessing.

Average # of correct answers on Q1-7

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Examplore</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/7</td>
<td>0/7</td>
<td>7/7</td>
</tr>
</tbody>
</table>

Mean difference is statistically significant (paired t-test: t=3.02, df=15, p-value<0.01)
Lab Study Results

For **Q8**, 88% of participants gave valid comments about the StackOverflow answer.

The majority of participants’ critiques...

- (Using the baseline) were about style and the mechanics of adaptation
- (Using Examplore) were about safety

• **Q8.** How might you modify this code example on Stack Overflow if you were going to copy and paste it into your own solution to the original prompt?
Lab Study Results

“I am confident in my answers to the API usage questions.”

<table>
<thead>
<tr>
<th>Participant Count</th>
<th>Strongly disagree</th>
<th>Neutral</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

Median: 4
Median: 5
Summary

It’s time to go beyond measuring precision and recall of software specification inference techniques.

1. Comprehension

2. Interactive Navigation

3. Fit Developer Workflow
Collaborators: Tianyi Zhang, Elena Glassman, Bjoern Hartmann, Gangesha Upadhyaya, Hridesh Rajan, Anstasia Reinhart
ExampleCheck and Examplore

Tool is available at http://examplore.cs.ucla.edu:3000/