Tackling Lack of Software Specifications
A Sustained, Sustainability and Productivity Crisis

Hridesh Rajan

Collaboration with Hoan A. Nguyen, Tien Nguyen, Gary Leavens, Samantha Khairunnesa, John Singleton, Hung Phan, Robert Dyer, and Vasant Honavar
Sustainability and productivity challenge

- To produce critical software infrastructure so it is:
  - of highest quality and free of defects,
  - produced ethically and within budget, and
  - maintainable, upgradeable, portable, scalable, secure.

- Pervasiveness of software infrastructures in such critical areas as power, banking and finance, air traffic control, telecommunication, transportation, national defense, and healthcare need us to address this challenge.
Software specifications* can help achieve this sustainability and productivity challenge.

* Software specifications: formal, often machine readable, description of software’s intended behavior, e.g. \{Pre\} S \{Post\} behavioral specifications
Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.
  - Maintenance of code can become easier
  - Lower cost of code understanding & total lifecycle cost
  - Specification-guided code optimization
  - Prevent introducing new bugs during maintenance
  - Code reuse
  - Specification-guided synthesis
  - Modular analysis and verification, scalable tools
Sustainability and productivity challenge

• If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

- Maintenance of code can become easier, because engineers will not need to spend time reverse engineering code.
- Lower cost of code understanding, lower total lifecycle cost.
- Optimization of code will be greatly facilitated
- Prevent introducing new bugs during maintenance
- Code reuse
- Synthesis of code
- Modular analysis and verification leading to scalable tools

Despite these benefits useful, non-trivial specifications aren’t widely available
Sustainability and productivity challenge

• If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

- Maintenance of code can become easier, because engineers will not need to spend time reverse engineering code.
- Lower cost of code understanding, lower total lifecycle cost.
- Optimization of code will be greatly facilitated.
- Prevent introducing new bugs during maintenance.
- Code reuse.
- Synthesis of code.
- Modular analysis and verification leading to scalable tools.

Why aren’t software specifications widely available?
Sustainability and productivity challenge

• If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

- Maintenance of code can become easier, because engineers will not need to spend time reverse-engineering code.
- Lower cost of code understanding, lower total lifecycle cost.
- Optimization of code will be greatly facilitated.
- Prevent introducing new bugs during maintenance.
- Code reuse.
- Synthesis of code.
- Modular analysis and verification leading to scalable tools.

Cost
Education
Tools
Libraries
Sustainability and productivity challenge

• If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

  - Maintenance costs will be reduced
  - Lower cost of code understanding, lower total lifecycle cost.
  - Optimization of code will be greatly facilitated
  - Prevent introducing new bugs during maintenance
  - Code reuse
  - Synthesis of code
  - Modular analysis and verification leading to scalable tools

Cost
Education
Tools
Libraries
Sustainability and productivity challenge

• If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

  - Maintenance of code can become easier, because engineers will not need to spend time reverse engineering code.
  - Lower cost of code understanding, lower total lifecycle cost.
  - Optimization of code will be greatly facilitated
  - Prevent introducing new bugs during maintenance
  - Code reuse
  - Synthesis of code
  - Modular analysis and verification leading to scalable tools

Unspecified libraries are root cause
  - increase cost of specification
  - make education harder
  - make tool support difficult
  - make specifying libraries harder
Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

How to Solve it? Specify key libraries
- decrease cost of specification
- make education easier (examples)
- make tool support easier
- make specifying libraries easier
Sustainability and productivity challenge

• If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.
  - Maintenance of code can become easier, because engineers will not need to spend time reverse engineering code.
  - Lower cost of code understanding, lower total lifecycle cost.
  - Optimization of code will be greatly facilitated
  - Prevent introducing new bugs during maintenance
  - Code reuse
  - Synthesis of code
  - Modular analysis and verification leading to scalable tools

How to Solve it? Specify key libraries

Challenge #1: lower manual cost of specifying libraries, infer most
Challenge #2: infer rich, but practical specifications, allow code evolution
Mining Preconditions of APIs in Large-scale Code Corpus, FSE’14.

Preconditions can be mined from **guarded conditions** at the call sites of the code using the APIs.

Preconditions mined from **multiple projects** in a **large-scale code corpus** can be used to filter out chaff.

```java
void m(...) {
    ...
    if (pred)
        lib.api();
    ...
}
Preconditions can be mined from guarded conditions at the call sites of the code using the APIs.

Preconditions mined from multiple projects in a large-scale code corpus can be used to filter out chaff.

```java
public boolean setPathFragmentation(int servletPathStart, int extraPathStart) {
    if (servletPathStart < 0 || extraPathStart < 0 ||
        servletPathStart > completePath_.length() ||
        extraPathStart > completePath_.length() ||
        servletPathStart > extraPathStart)
        return false;
    if (servletPathStart == completePath_.length()) {
        ... return true;
    }
    if (completePath_.charAt(servletPathStart) != '/')
        return false;
    if (extraPathStart == completePath_.length()) {
        ... return true;
    }
    if (completePath_.charAt(extraPathStart) != '/')
        return false;
    contextPath_ = completePath_.substring(0, servletPathStart);
    servletPath_ = completePath_.substring(servletPathStart, extraPathStart);
    ... return true;
}
```

Client code of API `String.substring(int,int)` in project SeMoA at revision 1929
Key Ideas

Preconditions can be mined from guarded conditions at the call sites of the code using the APIs.

Preconditions mined from multiple projects in a large-scale code corpus can be used to filter out chaff.

```java
public boolean setPathFragmentation(int servletPathStart, int extraPathStart) {
    if (servletPathStart < 0 || extraPathStart < 0 ||
        servletPathStart > completePath_.length() ||
        extraPathStart > completePath_.length() ||
        servletPathStart > extraPathStart)
        return false;
    if (servletPathStart == completePath_.length()) {
        ...    
        return true;
    }
    if (completePath_.charAt(servletPathStart) != '/')
        return false;
    if (extraPathStart == completePath_.length()) {
        ...
        return true;
    }
    if (completePath_.charAt(extraPathStart) != '/')
        return false;
    contextPath_ = completePath_.substring(0, servletPathStart);
    servletPath_ = completePath_.substring(servletPathStart, extraPathStart);
    ...
    return true;
}
```
Preconditions can be mined from guarded conditions at the call sites of the code using the APIs

Preconditions mined from multiple projects in a large-scale code corpus can be used to filter out chaff: a. infer, b. filter and rank

Key Ideas

Client method $M_1$ → Build CFG → Extract and Normalize → Infer → Preconditions

Client method $M_2$ → Build CFG → Extract and Normalize → Infer → Preconditions

... → Build CFG → Extract and Normalize → Infer → Preconditions

Build CFG

Extract and Normalize

Infer

Filter and Rank

Key Ideas

Preconditions can be mined from guarded conditions at the call sites of the code using the APIs

Preconditions mined from multiple projects in a large-scale code corpus can be used to filter out chaff: a. infer, b. filter and rank
Evaluation – Accuracy

Data collection

<table>
<thead>
<tr>
<th>SourceForge</th>
<th>Apache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projects</td>
<td>3,413</td>
</tr>
<tr>
<td>Total source files</td>
<td>497,453</td>
</tr>
<tr>
<td>Total classes</td>
<td>600,274</td>
</tr>
<tr>
<td>Total methods</td>
<td>4,735,151</td>
</tr>
<tr>
<td>Total SLOCs</td>
<td>92,495,410</td>
</tr>
<tr>
<td>Total used JDK classes</td>
<td>806 (63%)</td>
</tr>
<tr>
<td>Total used JDK methods</td>
<td>7,592 (63%)</td>
</tr>
<tr>
<td>Total method calls</td>
<td>22,308,251</td>
</tr>
<tr>
<td>Total JDK method calls</td>
<td>5,588,487</td>
</tr>
</tbody>
</table>

Almost 120 millions SLOCs

Ground Truth

www.jmlspecs.org

Extracted preconditions from published formal specification for JDK APIs on JML website
- 797 Methods
- 1155 preconditions

```java
/*@ public normal_behavior
@ requires 0 <= beginIndex
@ && beginIndex <= endIndex
@ && endIndex <= length();
@ */

/*@ public behavior
@ ...
@ signals (NoSuchElementException) isEmpty();
@ */
```
Accuracy of Preconditions Mining

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceForge</td>
<td>84%</td>
<td>79%</td>
<td>17h35m</td>
</tr>
<tr>
<td>Apache</td>
<td>82%</td>
<td>75%</td>
<td>34m</td>
</tr>
<tr>
<td>Both</td>
<td>83%</td>
<td>80%</td>
<td>18h03m</td>
</tr>
</tbody>
</table>

Performance
- ~ 1 minute/condition
- 5 preconditions are newly found for the JDK API methods that has already had JML specifications
- Effective for new specs

Performance

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Suggest</th>
<th>Accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringBuffer</td>
<td>delete(int,int)</td>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>replace(int,int,String)</td>
<td>2</td>
<td>Y*</td>
</tr>
<tr>
<td></td>
<td>setLength(int)</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>subSequence(int,int)</td>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>substring(int,int)</td>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td>LinkedList</td>
<td>add(int,Object)</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>addAll(int,Collection)</td>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>get(int)</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>listIterator(int)</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>remove(int)</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>set(int,Object)</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>2 classes</td>
<td>11 methods</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>
Accuracy by size

SourceForge

Apache
Usefulness Evaluation
Web-based Survey
http://boa.cs.iastate.edu/jml

Correctness
- Correct: 63%
- Good Starting Point: 18%
- Incorrect: 19%

Usefulness
- Strongly Agree: 33%
- Agree: 48%
- Disagree: 13%
- Strongly Disagree: 6%

Problem: Sparse labels in mined code corpus

Key Ideas

- Additional labels can be mined from **implicit beliefs** at the call sites of the code using the APIs

```
void m(...) {
    ...
    O o = new O();
    ...
}
```

- Implicit beliefs mined from **multiple projects** in a **large-scale code corpus** can be used to strengthen explicit labels
An Algorithm and Tool to Infer Practical Postconditions, Ongoing work.

Problem: Using extant work, e.g. strongest postcondition (sp), for postcondition inference produces impractical specs.

Strongest postcondition inference produces implicitly parallel formulas

\[ \text{sp} \left( \text{IF } B \text{ THEN } S_1 \text{ ELSE } S_2 \right) P = (\text{sp} \ S_1(P \land B)) \lor (\text{sp} \ S_2(P \land \neg B)) \]

Key Ideas

Flattening, and recombining parallel formulas can lead to much simpler inferred specifications.
Impact: 84% of specifications < ¼ page in length
We are overcoming lack of software specifications, a critical hurdle for high assurance SE, by combining program analysis and data mining.