Navigate, Understand, Communicate:
How Developers Locate Performance Bugs

Sebastian Baltes, Oliver Moseler, Fabian Beck, and Stephan Diehl

University of Trier, Germany
VISUS, University of Stuttgart, Germany

@s_baltes
s.baltes@uni-trier.de

9th International Symposium on
Empirical Software Engineering and Measurement
October, 22-23 - Beijing, China
Definitions

“A bug that affects speed or responsiveness.”
(Bugzilla@Mozilla)

“Defects where relatively simple source code changes can significantly speed up software, while preserving functionality.”
(Jin et al. - Understanding and Detecting Real-World Performance Bugs, PLDI’12)
Most existing debugging studies focused on how developers fix functional bugs.

But:

**Performance**
- is a non-functional requirement
- is difficult to measure (benchmarks?)

**Performance bugs**
- may corrupt user experience
- may waste resources (time, energy)
- can be difficult to reproduce and locate
- require knowledge of program state and runtime consumption

No study focusing on how developers locate (and fix) performance bugs.
Research Questions

**RQ1:**
How do developers **navigate** the source code and what **information and representation** is supportive for **locating** a performance bug?

**RQ2:**
How do developers try to **understand** and **explain** the causes of performance bugs?
Study Design
Study Design

• Qualitative observation study
• Controlled setting
• 12 developers, pair programming
• Locate and fix four performance bugs in collection libraries (Apache Commons Collections and Google Guava Libraries)
# Participants

### Table: Participant Information

<table>
<thead>
<tr>
<th>Team</th>
<th>Participant</th>
<th>Current Occupation</th>
<th>Work Exp. (years)</th>
<th>Experience (no exp. = 0 to 4 = expert)</th>
<th>Our Tool</th>
<th>Profiling</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>P1</td>
<td>Research assistant</td>
<td>5</td>
<td>OOP 4</td>
<td>Java 4</td>
<td>Collec. 3</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>Research assistant</td>
<td>5</td>
<td>OOP 4</td>
<td>Java 4</td>
<td>Collec. 4</td>
</tr>
<tr>
<td>T2</td>
<td>P3</td>
<td>MSc student, industry exp.</td>
<td>1</td>
<td>OOP 3</td>
<td>Java 3</td>
<td>Collec. 2</td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>MSc student, industry exp.</td>
<td>3</td>
<td>OOP 3</td>
<td>Java 3</td>
<td>Collec. 1</td>
</tr>
<tr>
<td>T3</td>
<td>P5</td>
<td>Software developer</td>
<td>3</td>
<td>OOP 4</td>
<td>Java 3</td>
<td>Collec. 4</td>
</tr>
<tr>
<td></td>
<td>P6</td>
<td>Diploma student</td>
<td>4</td>
<td>OOP 3</td>
<td>Java 3</td>
<td>Collec. 4</td>
</tr>
<tr>
<td>T4</td>
<td>P7</td>
<td>MSc student</td>
<td>0</td>
<td>OOP 3</td>
<td>Java 2</td>
<td>Collec. 3</td>
</tr>
<tr>
<td></td>
<td>P8</td>
<td>MSc student</td>
<td>0</td>
<td>OOP 1</td>
<td>Java 1</td>
<td>Collec. 1</td>
</tr>
<tr>
<td>T5</td>
<td>P9</td>
<td>Research assistant, industry exp.</td>
<td>10</td>
<td>OOP 3</td>
<td>Java 2</td>
<td>Collec. 3</td>
</tr>
<tr>
<td></td>
<td>P10</td>
<td>Research assistant, industry exp.</td>
<td>6</td>
<td>OOP 2</td>
<td>Java 2</td>
<td>Collec. 3</td>
</tr>
<tr>
<td>T6</td>
<td>P11</td>
<td>Software developer</td>
<td>15</td>
<td>OOP 3</td>
<td>Java 1</td>
<td>Collec. 3</td>
</tr>
<tr>
<td></td>
<td>P12</td>
<td>Software developer</td>
<td>1</td>
<td>OOP 3</td>
<td>Java 3</td>
<td>Collec. 2</td>
</tr>
<tr>
<td></td>
<td><strong>mean values:</strong></td>
<td></td>
<td><strong>4.4</strong></td>
<td><strong>OOP 3.0</strong></td>
<td><strong>Java 2.6</strong></td>
<td><strong>Collec. 2.7</strong></td>
</tr>
</tbody>
</table>

- All male
- Between 22 and 43 years old
- All except one team had industry experience
- Good level of expertise in OOP, Java, and data structures
- Lack of experience with IntelliJ IDE
- Not much experience fixing performance bugs (rare event)
Setup

- Introductory Video and Slides
- IDE
- Camera
- Microphone
- Sketching
Visual Performance Analysis Tools

- **Profiling tools** record program runs and assign measured performance values to code entities (e.g. runtime or memory consumption)
- We focus on **runtime consumption** and **Java** programs
- Standard user interface: **Lists**

![VisualVM and YourKit](image)

- **VisualVM**
- **YourKit**
Our Tool

callers \downarrow \quad method time \downarrow \quad callees

run() \quad 0.53% \quad self time \quad threads

color scale:
0% \quad 100%
Navigation – IDE

[Image of an IDE interface with navigation tools highlighted]
Navigation – Profiling Tool

```java
package performancetests;

import ...

public class PerformanceTest {
    public static void main(String[] args) {
        int size = 20000; // Number of elements to store in the multi value map
        MultiValueMap multi = new MultiValueMap();
        for (int i = 0; i < size; i++) { // Insert values
            multi.put(i, 1); // 0.2%
        }
        List<Integer> toContain = new ArrayList<>(); // A list of elements to check
        for (int i = size - 1; i > -1; i--)
            toContain.add(i); // 8.0%
        // Get all values of the multi value map
        Collections.multimapLikes = multi.values();
        // ContainsAll Collection
        Collections.containsAll(toContain); // 99.7%

        long start = System.currentTimeMillis(); // Start time measuring
        Collections.containsAll(toContain);
        System.out.println("Time is " + (stop - start) + " ms"); // Print elapsed time
        System.out.println("99.73% java.util.AbstractCollection.containsAll(java.util.Collection)");
    }
}
```
Data Collection
Available Data

Course of a study session:

- Tutorial
- Warm-up task
- Performance Bug
- Structured Interview
- Questionnaire

Available Data:

- Tutorial
- Warm-up task
- Performance Bug
- Structured Interview
- Questionnaire

Audio recording
Screen capture
IDE navigation log
Sketching video
Interview transcripts
Coding of interaction
Visualization

Demographic Data
Results – RQ1
RQ1:
How do developers **navigate** the source code and what **information and representation** is supportive for **locating** a performance bug?

**Methods (RQ1)**

**Interview transcripts (bug 1-4)**

**Cross-case analysis** [Seaman99]

**Navigation visualization (bug 3)**

**Pattern search**

**TABLE II. PROPOSITIONS BASED ON CROSS-CASE ANALYSIS OF INTERVIEW ANSWERS RELATED TO RQ1.1 (TOP) AND RQ1.2 (BOTTOM).**

<table>
<thead>
<tr>
<th>No.</th>
<th>Proposition</th>
<th>Teams</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The dynamic instance of a method call and connected runtime information are important for navigation.</td>
<td>T1, T3, T4, T5</td>
</tr>
<tr>
<td>1.2</td>
<td>Following high quantities of runtime in the dynamic method call graph is helpful as a navigation strategy.</td>
<td>T1, T2, T3, T6</td>
</tr>
<tr>
<td>1.3</td>
<td>The more complex the performance bug is, the less helpful the provided tool support and information becomes.</td>
<td>T1, T3, T5, T6</td>
</tr>
<tr>
<td>2.1</td>
<td>The integration into the code view provides additional context for the profiling visualization.</td>
<td>T1, T2, T4, T6</td>
</tr>
<tr>
<td>2.2</td>
<td>The overview (list view) was not needed in this setting.</td>
<td>T1, T4, T5</td>
</tr>
<tr>
<td>2.3</td>
<td>The overview (list view) could be used as a starting point for further analyses.</td>
<td>T1, T2, T4</td>
</tr>
</tbody>
</table>

**Team 6**

<table>
<thead>
<tr>
<th>PerformanceTest_03</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractCollection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>containsAll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iterator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>containsAll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MultiValueMap$Values</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ValuesIterator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getCollection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MultiValueMap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getEntry</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hash</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RQ1: Navigation

RQ1.1: How was information from the profiling tool or other parts of the IDE used to locate the performance bug?

- **Dynamic runtime information** important for navigation (Prop. 1.1)

  ```java
  values.containsAll(toContains); 99.73%
  long stop = System.currentTimeMillis();
  System.out.println("Time is " + (stop - start) + "ms"); // Print elapsed time
  ```

- **Helpful strategy:** Following high quantities of runtime in dynamic call graph (Prop. 1.2)

- **But:** The more complex the performance bug is, the less helpful the provided information becomes (Prop. 1.3)

Beside runtime information, the **dynamic call graph** is important, but it can become too complex. (→ future work)
RQ1: Navigation

RQ1.2: Is the in-situ visualization of the profiling data beneficial compared to a traditional list representation?

- Integration into code view provides **additional context** for the profiling data (Prop. 2.1)

  ![Code Example](image)

  ```java
  // Get all values of the multi value map
  Collection<?> values = multi.values();

  // Call containsAll on the values
  values.containsAll(toContain); // Start time measuring
  long start = System.currentTimeMillis();

  // Call containsAll on the values
  System.out.println("Time is " + (stop - start) + ",\n"");
  ```

  **VS.**

  ```java
  System.out.println("Time is " + (stop - start) + ",\n"");
  ```

- **List view not needed** in this setting (test cases) (Prop. 2.2)

- **But:** List view could be good starting point for further analyses (Prop. 2.3)

**Integrating source code and performance information** is a promising approach; list and in-situ visualization seem to complement each other.
RQ1.3: What navigation strategies do developers pursue to locate a performance bug?

- About 70% of navigation through IDE, 30% with our tool
- Navigation with method call visualization dominant (in-situ)
- List view never used for bug 3
- Identified two navigation strategies:

  **Strategy 1 (Toggle):** Frequent switching between test class and and other classes related to bug (IDE navigation).

  **Strategy 2 (Path Following):** Follow dynamic method calls with high runtime consumption (In-situ visualization).
RQ1: Navigation

**Team 4**

PerformanceTest_03

MultiValueMap

getCollection

Collection

AbstractCollection

containsAll

contains

MultiValueMap$Values

iterator

MultiValueMap$ValuesIterator

ValuesIterator

HashMap

get

getEntry

hash

---

**Team 6**

PerformanceTest_03

AbstractCollection

containsAll

contains

MultiValueMap$Values

iterator

containsAll

MultiValueMap$ValuesIterator

ValuesIterator

MultiValueMap

getCollection

HashMap

get

getEntry

hash

Collection

---

**Strategy 1**

(Toggle)

**Strategy 2**

(Path Following)
Results – RQ2
RQ2: How do developers try to understand and explain the causes of performance bugs?

- Interview transcripts (bug 1-4)
- Coding of interaction (bug 3)
- Cross-case analysis
- Descriptive statistics
RQ2:
How do developers try to **understand** and **explain** the causes of performance bugs?

Sketching video (bug 3)
RQ2: Understanding and Communicating

RQ2.1: How do developers communicate with each other when locating a performance bug?

- 4 of 6 teams expressed first hypothesis about cause of bug in the first half of session
- Driver and navigator mostly worked on same level of abstraction
- 3 teams had very active navigator (e.g. asking questions about code, prompting driver to navigate to certain methods)
- 2 teams had very passive navigator (mostly observed)
- Different levels of expertise can be reason for active/passive role

Driver and navigator work on same level of abstraction; interaction could be affected by different levels of expertise.
RQ2.2: Could sketches help to understand and communicate a performance bug?

- Four teams spontaneously created a sketch while locating bug 3
- All sketches created by navigator
- Sketching **static structure** (e.g. `MultiValueMap`)
- Sketching **dynamic aspects** (execution of method `contains(...)`)  
- Keeping track of **alternative hypotheses**

Sketches considered mostly positive as an aid for explaining a performance bug (in a PP setting).
• **Unusual setting** for participants (laboratory, libraries, IDE, tool, etc.) → Tutorial phase, focus on third bug

• Teams **did not know** each other before → Focus on third bug

• We **helped participants** if they got stuck → Prepared hints beforehand, same order for all groups

• A part of the analysis (coding, cross-case analysis) **conducted by two researchers alone** → Discussed the results in group, went back to raw data if required
Conclusion

• First study focusing on how developers locate performance bugs

• **Input for improving profiling tools:**
  • In-situ visualization of performance data helpful
  • Dynamic call graph important (but: complexity needs to be considered)
  • Tools should support different strategies (toggle and path following)

• **Future work:**
  • Trying to replicate results in industry context
  • Coding of developer interactions for all bugs, searching for patterns

Data and supplementary material:
http://st.uni-trier.de/study-debugging

@s_baltes
s.baltes@uni-trier.de