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



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



Universität Trier

Sebastian Baltes – How Developers Locate Performance Bugs (ESEM'15)



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)



Research Gap

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?



How do developers try to **understand** and **explain** the causes of performance bugs?







Study Design



Universität Trier

Sebastian Baltes - How Developers Locate Performance Bugs (ESEM'15)

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)



Universität Trier

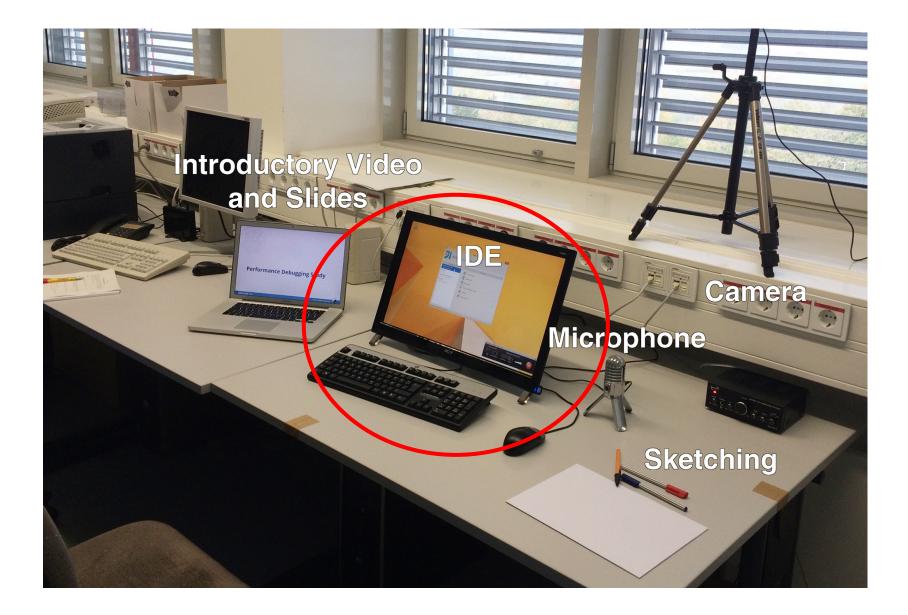


Participants

Team	Participant	Current Occupation	Work Exp.	Experience (no exp. $= 0$ to $4 =$ expert)										
	-	-	(years)	OOP	Java	Collec.	IntelliJ	IDEs	Perf.Bugs	Our Tool	Profiling			
T1	P1	Research assistant	5	4	4	3	3	3	1	1	0			
11	P2	Research assistant	5	4	4	4	1	4	2	1	1			
T2	P3	MSc student, industry exp.	1	3	3	2	0	3	1	0	2			
12	P4	MSc student, industry exp.	3	3	3	3	1	2	1	0	1			
T3	P5	Software developer	3	4	3	4	1	3	3	1	2			
	P6	Diploma student	4	3	3	3	4	2	1	1	0			
T4	P7	MSc student	0	3	2	3	1	2	1	0	0			
14	P8	MSc student	0	1	1	0	0	1	1	0	1			
T5	P9	Research assistant, industry exp.	10	3	2	3	0	4	4	0	3			
15	P10	Research assistant, industry exp.	6	2	2	2	3	1	3	0	2			
Т6	P11	Software developer	15	3	1	3	0	3	2	1	2			
10	P12	Software developer	1	3	3	2	2	2	1	0	1			
		mean values:	4.4	3.0	2.6	2.7	1.3	2.5	1.8	0.4	1.3			

- 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



Software Engineering Group

8

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

🗘 Java2Demo (pid 4466)								
Profiler			Sett	ings				
Profile: CPU Memory Stop Status: profiling running (619 methods instrume) inted)							
Profiling results								
🞯 😢 🛅 🕪 📄 Snapshot 🔛 🔀								
Hot Spots - Method	Self time [%] 🔻	Self time	Invocations	a				
java2d.intro\$Surface\$Scene.pause (Thread) java2d.intro\$Surface.paint (java.awt.Graphics)		2321 ms (60.3%) 1071 ms (27.9%)	21	4 B		YourKit		
java2d.Intro\$Surface\$DdE.render (int, int, java		91.4 ms (2.4%) 89.4 ms (2.3%)						
javax.swing.SystemEventQueueUtilities\$Compone java2d.java2Demo\$j2Dicon.paintIcon (java.awt		87.3 ms (2.3%)	Stack	c Traces	CPU Usage Estimation			
java2d.Intro\$Surface\$GpE.render (int. int. java	1	51.0 ms (1.3%)		_				
java2d.Intro\$Surface\$TxE.render (int, int, java	1	46.9 ms (1.2%)		To updat	e this view, select range in	the graph above by dragging		
java2d.Intro\$Surface\$Features.render (int, int,		34.3 ms (0.9%)			_			
java2d.Intro\$Surface\$Temp.render (int, int, jav	1	24.1 ms (0.6%)	Time	range: 11	n 42m 22 sec 900 ms - 1h 4	2m 37 sec 600 ms Duration: 14 se	c Threads included: 13	of 13
java2d.Intro\$Surface\$Scene.render (int, int, ja		13.3 ms (0.3%)	[alla a	_		1	1	
java2d.intro\$Surface\$TxE.step (int, int)		5.14 ms (0.1%)	503 C	2	Call Tre	e	CPU Time (ms)	Samples
java2d.Intro\$Surface\$Scene.step (int, int)		3.90 ms (0.1%)		All threa	des		8,687 100 %	10 100 %
java2d.Intro\$Surface\$GpE.step (int, int)		1.29 ms (0%)						
[Method Name Filter]				∃ ∑ jav	/a.awt.EventDispatchThre	ad.run()	8,640 99 %	10 100 %
				E JC	omponent.java:889 🞽 De	moApp.paint(Graphics)	8,640 99 %	10 100 %
Vis	sualVN	1			DemoApp.java:91 🔰 De	moApp.drawDemo(Graphics2D)	8,640 99 %	10 100 %
					🗆 DemoApp.java:76 🖄	sun.java2d.SunGraphics2D.clip(8,640 99 %	10 100 %
				_∑ jav	/a.lang. Thread.run ()		46 1 %	10 100 %

Our Tool

ProfilingArtifactOverview
Filter artifacts (separate with comma)
callers method time callees
exclude color scale:
Methods Classes run () 0.53%
self time threads
22,87% com.intellij.rt.execution.application.AppMain
java.net.PlainSocketImpl.accept(java.net.SocketImpl)
java.net.ServerSocket.implAccept(java.net.Socket)
java.net.ServerSocket.accept()
<pre>> 22,75% java.net.DualStackPlainSocketImpl.socketAccept(java.net.SocketImpl)</pre>
<pre>> 22,75% java.net.AbstractPlainSocketImpl.accept(java.net.SocketImpl)</pre>
java.net.DualStackPlainSocketImpl.accept0(int, java.net.InetSocketAddress
21,81% com.intellij.rt.execution.application.AppMain.main(java.lang.String[])
<pre>> 21,75% java.lang.reflect.Method.invoke(java.lang.Object, java.lang.Object[])</pre>
21.75% sup reflect. Delegat
sun.reflect.NativeM private class Values extends AbstractCollection /18,35%
21.75% de.unitrier.TestCla
21.75%
18,12% de.unitrier.TestCla
<pre>> 17.63% de.unitrier.TestCla. public Iterator iterator ()) /18.35%</pre>
11,96% de.unitrier.TestCla
7.23% de.unitrier.A.runOn
3,63%
1,12% de.unitrier.MyThread final IteratorChain chain = new IteratorChain(); 0.13%
0,70% java.lang.ClassLoadClass(java.lang.String)
0,70% java.lang.ClassLoadClass(java.lang.String, boolean)
<pre>> 0,70% > sun.misc.Launcher\$AppClassLoadClass(java.lang.String, boolean)</pre>
0,65% java.net.URLClassLoader\$1.run()
0,65% java.net.URLClassLoader.findClass(java.lang.String)
<pre>> 0,65% > java.security.AccessController.doPrivileged(java.security.PrivilegedExcep)</pre>
0,57%
0,49% sun.misc.URLClassPath.getResource(java.lang.String, boolean)
0,43% de.unitrier.A.A()
0,43% de.unitrier.B.B()
0,41% de.unitrier.B.B()

Universität Trier

Sebastian Baltes – How Developers Locate Performance Bugs (ESEM'15)

Navigation – IDE

		🖻 GravatarMain.java - SOUtils - [~/git/soutils]
▶ 🗄 💋 🛩 🏕 🗶 🗊 🗗 🔍 🚶 💠 🕂 🖻	GravatarHash 🚽 🕨 🗰 🎯	5 学 晶 音 単 ? 辞
📑 soutils 👌 🖿 src 🤇 🛅 com 👌 🖬 sbaltes 🖉 🖬 soutils 🗸 😋 Grava	itarMain	
ដ្ឋ 🖻 Project 👻	⊕ ‡ 🕸 🗄 😋 UnescapeMain.java ×	CravatarMain.java × C SelectQuery.java × C LinkedList.java × C DatabaseHelpers.java × C String.java ×
g □ Project g ▼ □ g ↓ □ idea ↓ □ lib ↓ □	<pre>import java.sql.Co import java.sql.SQ public class Grava</pre>	LException;
e ► Casel	-	
end sql ▼ □ src ▼ □ com.sbaltes.soutils	public static	<pre>void main(String[] args) {</pre>
 com.sbaltes.soutils c > DatabaseHelpers c > GravatarMain c > SelectQuery c > UnescapeMain gitignore SOUtils.iml External Libraries 	String dat String use String pas Connection try {	<pre>t = args[0]; abase = args[1]; r = args[2]; sword = args[3]; conn = null; DatabaseHelpers.connectToDatabase(host, database, user, password);</pre>
	Select	<pre>Query query = new SelectQuery(conn, "users");</pre>
	<pre>query. query. query. Databa } catch (S e.prin } finally DbUtil }</pre>	Paste 第V seHe Paste from History 企業V Paste Simple で合業V QLE> Column Selection Mode 企業■* tSta { Find Usages ℃F7
		Folding
	}	Go To Jump to Navigation Bar ℃ Generate ^N Declaration %B Compile 'GravatarMain.java' ☆ ೫FP Implementation(s) ℃ %B Run 'GravatarHash' ^ ☆ F1 Super Method %U © Debug 'GravatarHash' ^ ☆ F9 Tast O % 8T
Version Control: Local Changes Log		w Debug GravatarHash
Image: Specific definition Image: Specific definition VSS ★ Specific definition ★		Local History Git Compare with Clipboard
		File Encoding
 ⋈ ⋈		 Create Gist Diagrams
ie s		WebServices
Eavorites		



Navigation – Profiling Tool

🔮 PerformanceTest_01.java 🗙 🕐 PerformanceTest_02.java 🛪 🕑 PerformanceTest_03.java 🗴	ProfilingArtifactOverview 🕸 - 🚽
package performancetests;	Filter artifacts (separate with comma)
⊕import	
	Apply
public class PerformanceTest_03	exclude
{ public static void main(String[] args) 24.97%	
	Methods Classes
int size = 20000; // Number of elements to store in the multi value map	Runtime Artifact's name
	24.97% java.lang.reflect.Method.invoke(java.lang.Object, java.
// Create a multi value map	24.97% sun.reflect.DelegatingMethodAccessorImpl.invoke(java.la
MultiValueMap multi = new MultiValueMap(); 0.02%	24.97% sun.reflect.NativeMethodAccessorImpl.invoke(java.lang.C
<pre>for (int i = 0; i < size; i++) // Insert values </pre>	24.97% java.lang.Thread.run()
multi.put(i, i); 0.23%	24.97% performancetests.PerformanceTest_03.main(java.lang.Stri
	24.97% com.intellij.rt.execution.application.AppMain.main(java
	24.97% com.intellij.rt.execution.application.AppMain\$1.run()
List <integer> toContain = new ArrayList<>(); // A list of elements to check</integer>	24.97% sun.reflect.NativeMethodAccessorImpl.invoke0(java.lang.
for (int $i = size - 1; i > -1; i$)	24.96% java.net.PlainSocketImpl.accept(java.net.SocketImpl)
	24.96% java.net.ServerSocket.implAccept(java.net.Socket)
toContain.add(i); 0.02%	24.96% java.net.ServerSocket.accept()
3	24.96% java.net.DualStackPlainSocketImpl.socketAccept(java.net
// Get all values of the multi value map	24.96% java.net.AbstractPlainSocketImpl.accept(java.net.Socket
Collection values = multi.values();	24.96% java.net.DualStackPlainSocketImpl.accept0(int, java.net
	24.90% java.util.AbstractCollection.containsAll(java.util.Coll
/**************** containsAll ***********************************	24.74%
	18,31% org.apache.commons.collections.map.MultiValueMap\$Values
<pre>long start = System.currentTimeMillis(); // Start time measuring</pre>	9.33%
// Call containsAll on the values values.containsAll(toContain); 99.73%	<pre>8.82%</pre>
long stop = System.currentTime99.73% java.util.Abstractfiollection.containsAll(java.util.Collection)	8.82% Java.util.HashMap.get(java.lang.Object)
System.out.println("Time is " + (stop - start) + "ms"); // Print elapsed time	3.32% java.util.HashMap.hash(java.lang.Object)
0.700	rChain.
values.containsAll(toContain); 99.73%	rChain.
long stop = System.currentTime99.73% java.util.Abstractfollec	tion.containsAll(java.util.Collection)
System.out.println("Time is " + (stop - start) + "ms"); / Pr	int alanged time
System. out. printin("lime is " + (stop - start) + "ms"); ()/ Pr	THE STAPSED FIME

Data Collection



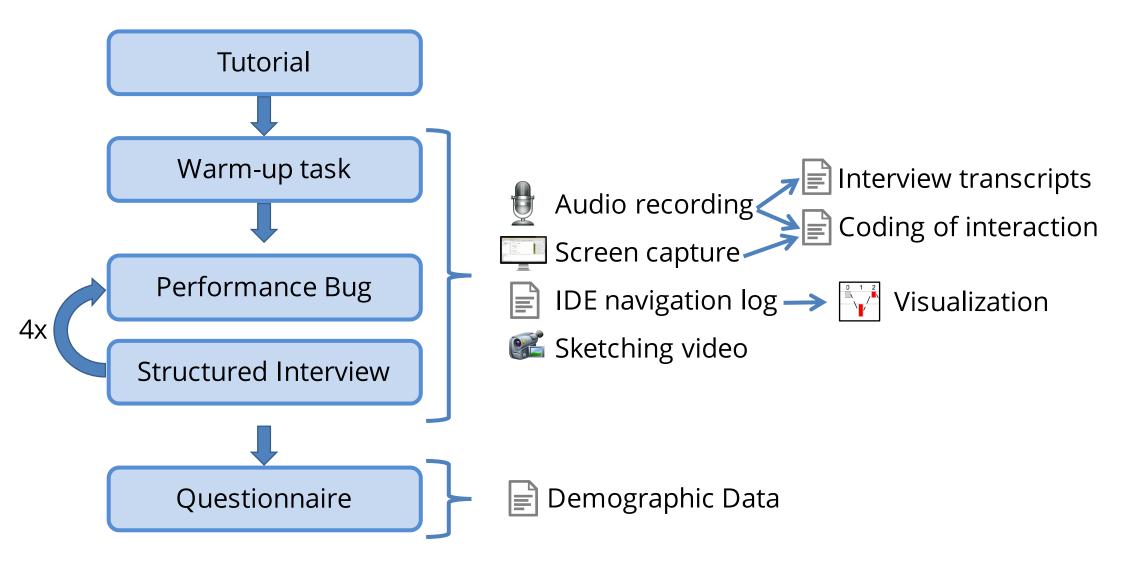
Universität Trier

Sebastian Baltes – How Developers Locate Performance Bugs (ESEM'15)

13

Available Data

Course of a study session:



Results – RQ1



Universität Trier

Sebastian Baltes – How Developers Locate Performance Bugs (ESEM'15)

Software Engineering Group

Methods (RQ1)

RQ1:

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



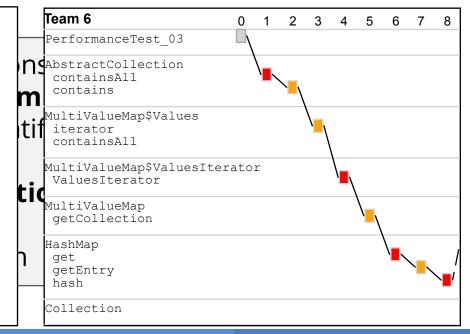
Cross-case analysis [Seaman99]

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).

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



Universität Trier

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)

values.containsAll(toContain); 99.73% long stop = System.currentTime99.73% java.util.Abstractfipllection.containsAll(java.util.Collection) 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. (\rightarrow future work)

17

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)



- 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.

Universität Trier

RQ1.3: What navigation strategies do developers pursue to locate a performance bug?

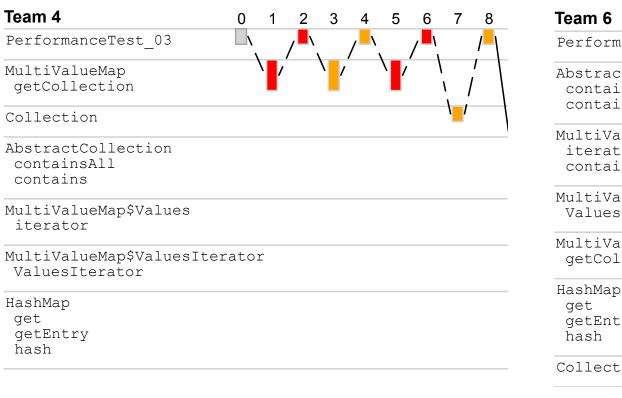
- About 70% of navigation through IDE, 30% with our tool
- Navigation with method call visualization dominant (in-situ)
- List view never used for bug 3
- Identified two navigation strategies:

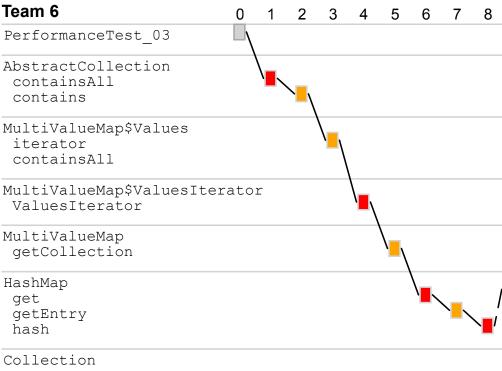


Strategy 1 (Toggle): Frequent switching between test class and and other classes related to bug (IDE navigation).



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





Strategy 1 (Toggle)

Strategy 2 (Path Following)

Software Engineering Group

Results – RQ2



Universität Trier

Sebastian Baltes – How Developers Locate Performance Bugs (ESEM'15)

Software Engineering Group 👔 21

Methods (RQ2)

RQ2:

How do developers try to **understand** and **explain** the causes of performance bugs?



Interview transcripts (bug 1-4)

Cross-case analysis

Coding of interaction (bug 3)

Descriptive statistics

TABLE VI. PROPOSITION INTERVIEW ANS		TABLE IV. INTERACTIONS WHILE LOCATING PERFORMANCE BUG 3 (D: DURING, A: AFTER LOCATING BUG, *: NAVIGATOR TOOK OVER ROLE OF DRIVER, CODES: SEE TABLE V)														
		Team	Time (min.)	Success	Driver	Navigator	Total	DC+HC	DR+HR	QC+QR	odes PN+PI	CO	RD+RC+RE	Other	First Strategy	Sketch
No.	Proposition	T1	30	~	P2	P1	165 45% 55%	46 57% 43%	11 55% 45%	28 21% 79%	5 0% 100%	10 20% 80%	11 55% 45%	54 54% 46%	1	D
3.1	Sketches are a useful tool mance bug, but context in understand them afterwards.	T2	30	~	P4	Р3	112 57% 43%	21 67% 33%	19 58% 42%	24 54% 46%	9 11% 89%	6 33% 67%	9 56% 44%	24 75% 25%	1	Α
3.2 3.3	Sketches are a suitable de "polished" enough). If and how much sketching	Т3	24	\checkmark	P5	P6	78 63% 37%	18 83% 17%	13 85% 15%	10 90% 10%	6 0% 100%	7 0% 100%	3 100% 0%	21 52% 48%	2	A
3.4	sketching experience of the A common sketch vocabular	T4	35	\checkmark	P7	P8	136 46% 54%	24 58% 42%	22 68% 32%	20 20% 80%	15 0% 100%	7 29% 71%	10 20% 80%	38 68% 32%	1	D
3.5	More complex problems or likely to be sketched.	Т5	20	0	P10*	P9*	48 35% 65%	14 21% 79%	9 44% 56%	10 30% 70%	2 0% 100%	0 0% 0%	2 100% 0%	11 45% 55%	-	D
<u> </u>	3.6 Sketches can be used to exp a program.	Т6	24	×	P12	P11	40 63% 38%	15 73% 27%	13 77% 23%	1 0% 100%	2 0% 100%	3 0% 100%	0 0% 0%	6 67% 33%	2	D

Universität Trier

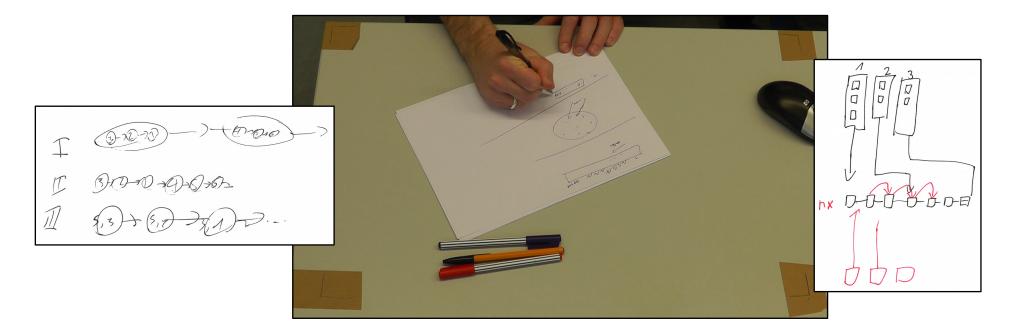
22

Methods (RQ2)

RQ2:

How do developers try to **understand** and **explain** the causes of performance bugs?





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



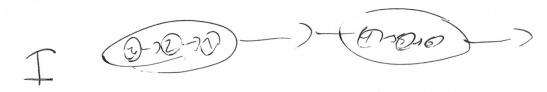
Driver and navigator work on **same level of abstraction**; interaction could be affected by different levels of expertise.

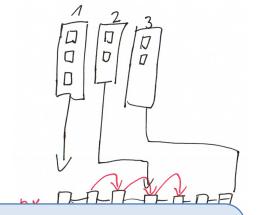


RQ2: Understanding and Communicating

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).

Threats to Validity

- 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

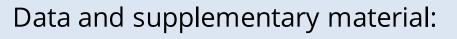


- We helped participants if they got stuck
 → Prepared hints beforehand, same order for all groups
- A part of the analysis (coding, cross-case analysis) conducted by two researchers alone

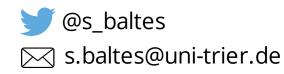
 \rightarrow Discussed the results in group, went back to raw data if required

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



http://st.uni-trier.de/study-debugging



Universität Trier

