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

esem 2015

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?

RQ2:

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

Study Design

Study Design

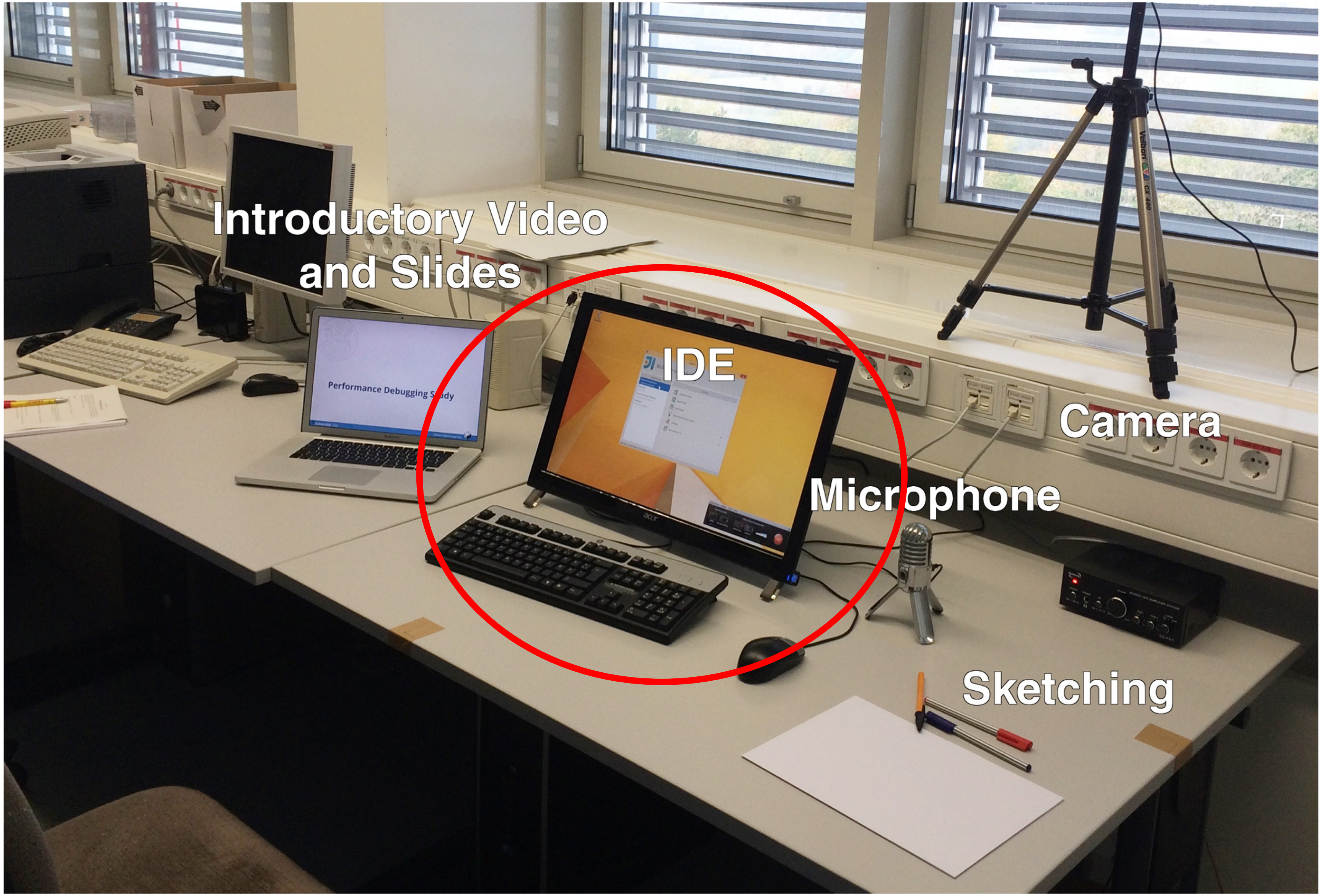
- Qualitative observation study
- Controlled setting
- 12 developers, pair programming
- Locate and fix four performance bugs in collection libraries
  (Apache Commons Collections and Google Guava Libraries)

Participants

Team	Participant	Current Occupation	Work Exp. (years)	Experience (no exp. = 0 to 4 = expert) OOP	Java	Collec.	IntelliJ	IDEs	Perf.Bugs	Our Tool	Profiling
T1	P1	Research assistant	5	4	4	3	3	3	1	1	0
	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
	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
	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
	P10	Research assistant, industry exp.	6	2	2	2	3	1	3	0	2
T6	P11	Software developer	15	3	1	3	0	3	2	1	2
	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

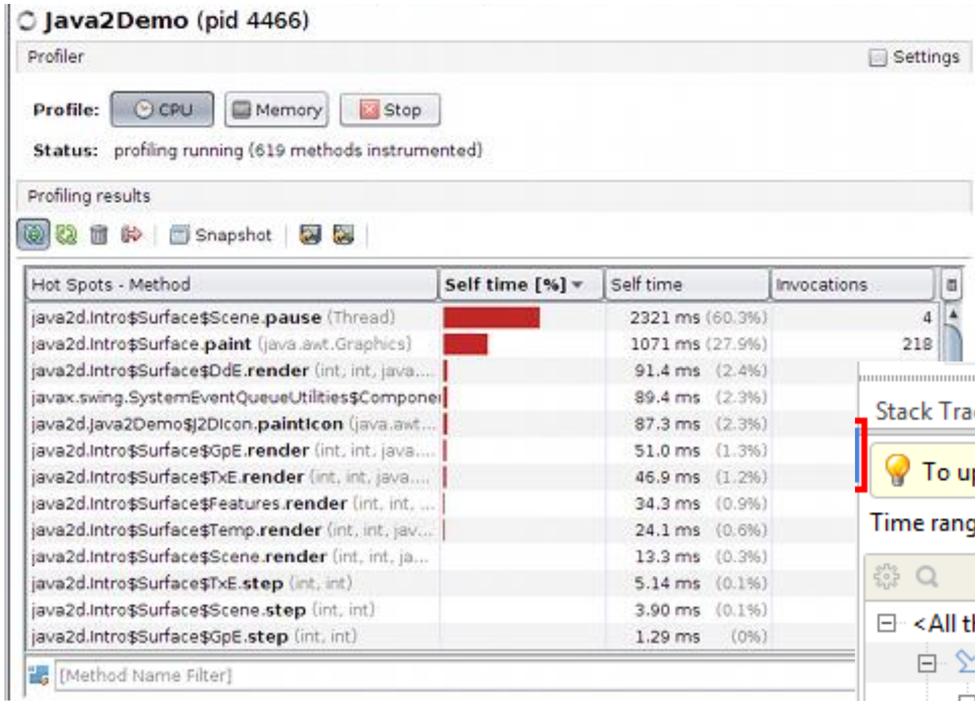
- All male
- Between 22 and 43 years old
- All except one team had industry experience
- Good level of expertise in OOP, Java, and data structures
- Lack of experience with IntelliJ IDE
- Not much experience fixing performance bugs (rare event)

Setup


Introductory Video
and Slides
Performance Debugging Study
IDE
Camera
Microphone
Sketching

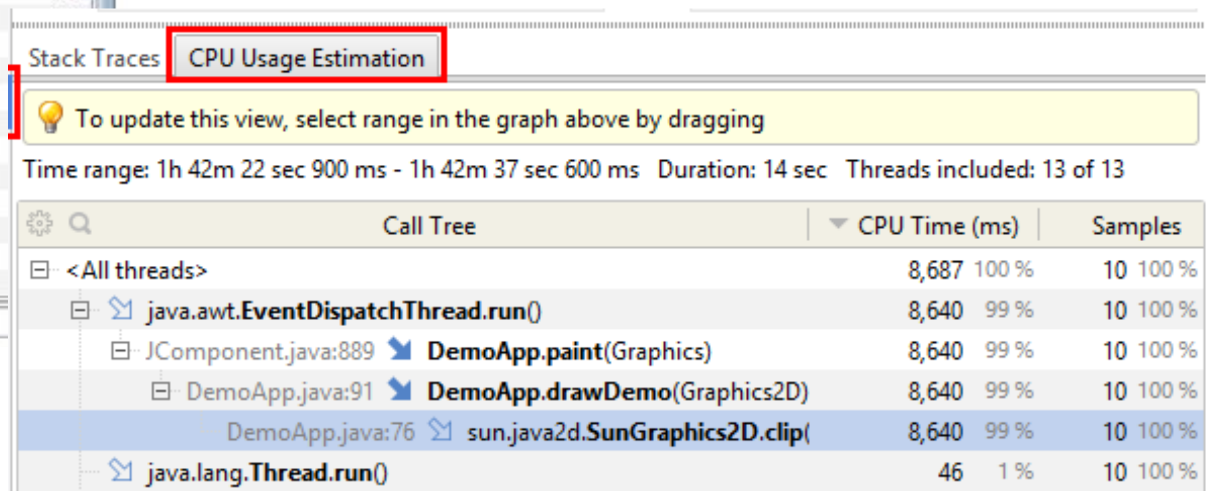
Visual Performance Analysis Tools

- **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**

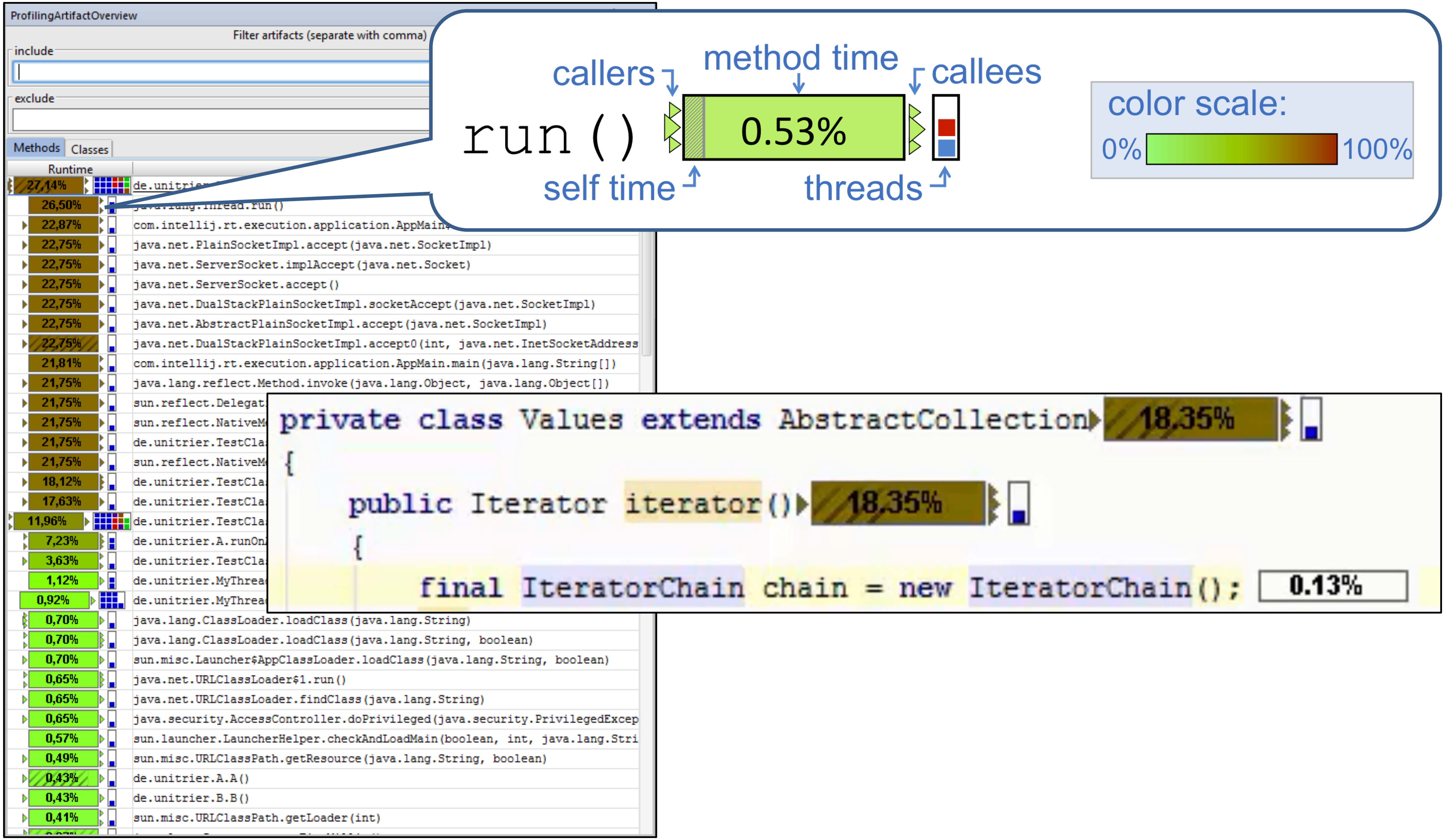


VisualVM

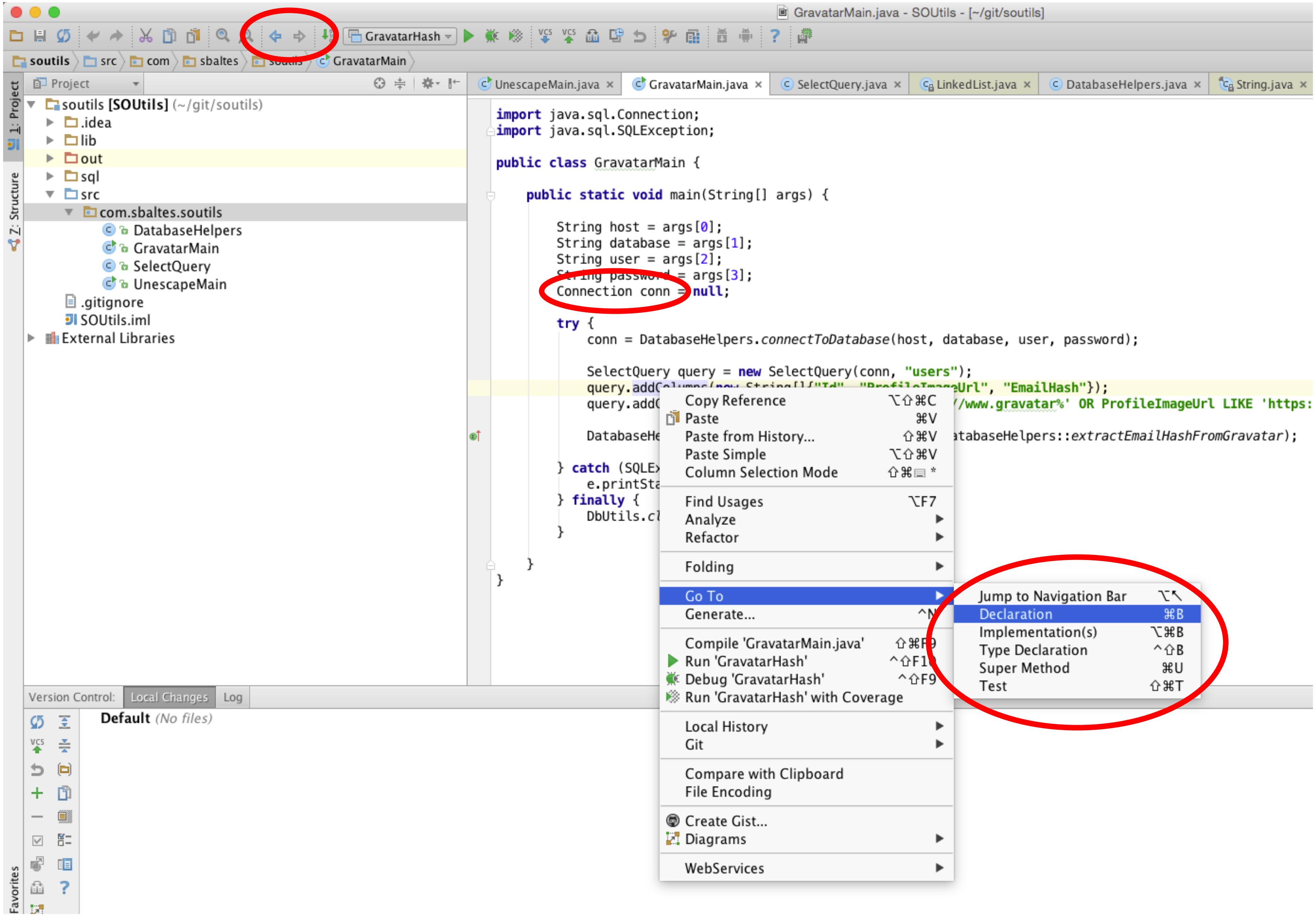
YourKit



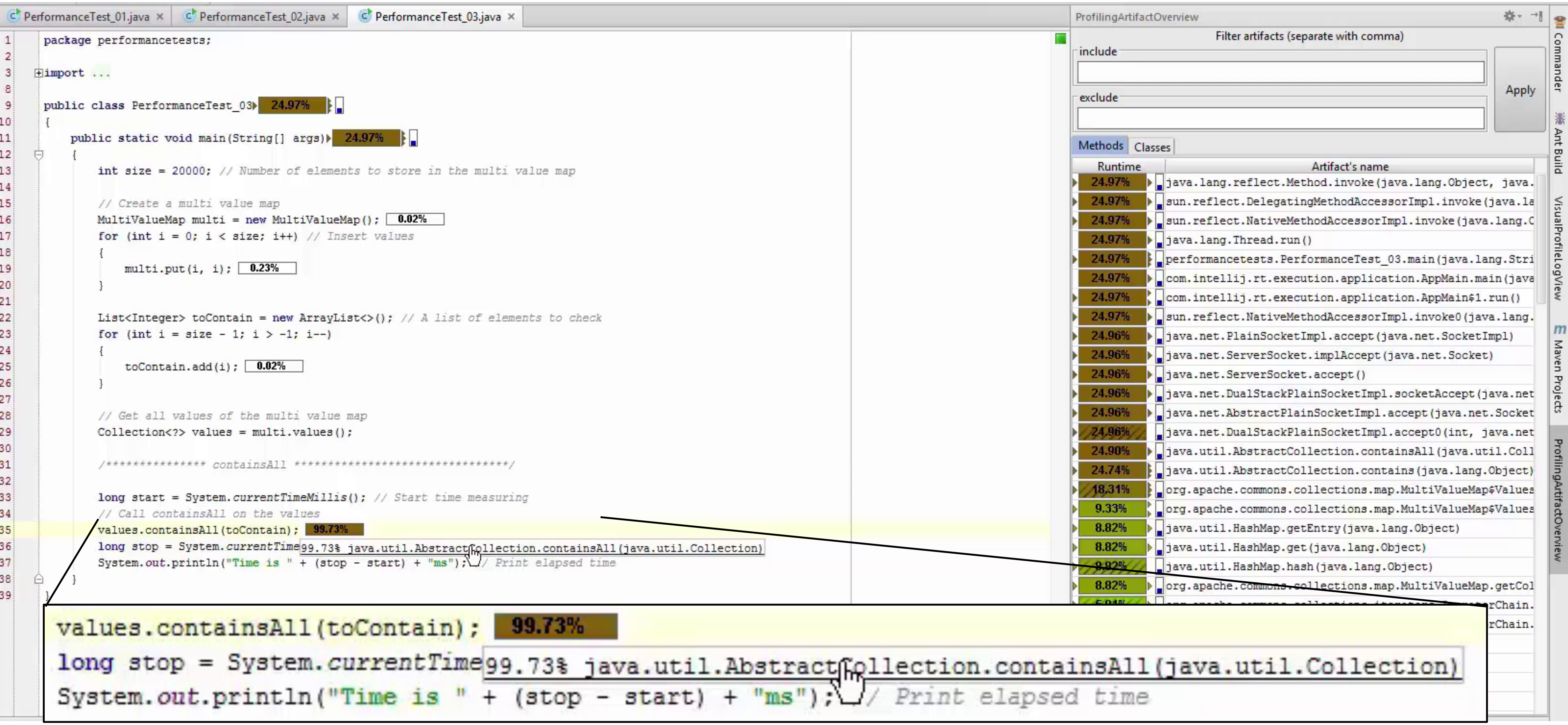
Our Tool


ProfilingArtifactOverview
Filter artifacts (separate with comma)
include
exclude
Methods
Classes
Runtime
27,14%
26,50%
22,87% com.intellij.rt.execution.application.AppMain
22,75% java.net.PlainSocketImpl.accept(java.net.SocketImpl)
22,75% java.net.ServerSocket.implAccept(java.net.Socket)
22,75% java.net.ServerSocket.accept()
22,75% java.net.DualStackPlainSocketImpl.socketAccept(java.net.SocketImpl)
22,75% java.net.AbstractPlainSocketImpl.accept(java.net.SocketImpl)
22,75% java.net.DualStackPlainSocketImpl.accept0(int, java.net.InetSocketAddress
21,81% com.intellij.rt.execution.application.AppMain.main(java.lang.String[])
21,75% java.lang.reflect.Method.invoke(java.lang.Object, java.lang.Object[])
0,70% java.lang.ClassLoader.loadClass(java.lang.String)
0,70% java.lang.ClassLoader.loadClass(java.lang.String, boolean)
0,70% sun.misc.Launcher$AppClassLoader.loadClass(java.lang.String, boolean)
0,65% java.net.URLClassLoader$1.run()
0,65% java.net.URLClassLoader.findClass(java.lang.String)
0,65% java.security.AccessController.doPrivileged(java.security.PrivilegedExcep
0,57% sun.launcher.LauncherHelper.checkAndLoadMain(boolean, int, java.lang.Stri
0,49% sun.misc.URLClassPath.getResource(java.lang.String, boolean)
0,43% de.unitrier.A.A()
0,43% de.unitrier.B.B()
0,41% sun.misc.URLClassPath.getLoader(int)
callers
method time
callees
run()
0.53%
self time
threads
color scale:
0%
100%
private class Values extends AbstractCollection 18,35%
{
public Iterator iterator() 18,35%
{
final IteratorChain chain = new IteratorChain(); 0.13%

Navigation – IDE



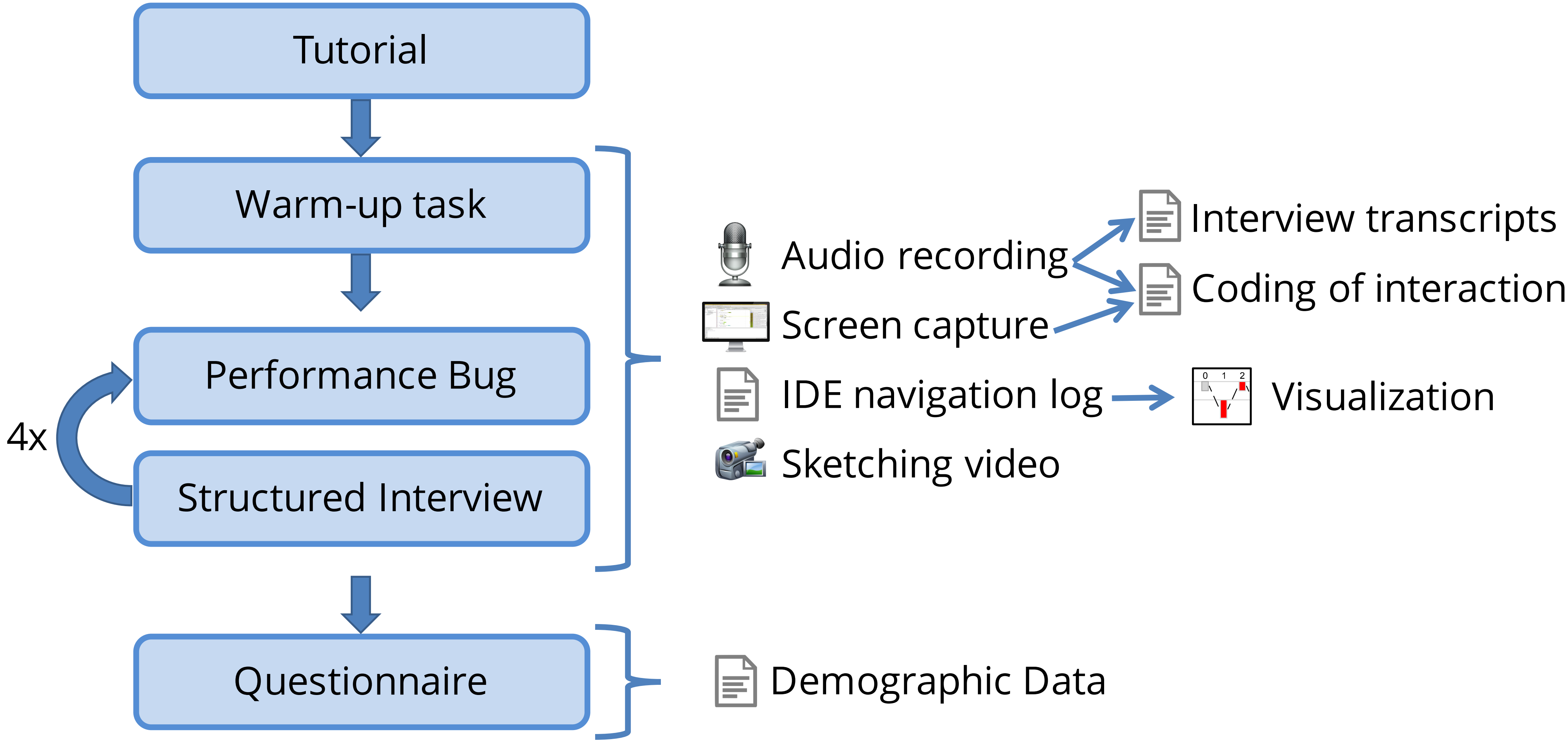
Navigation – Profiling Tool



Data Collection

Available Data

Course of a study session:



Results – RQ1

Methods (RQ1)

RQ1:

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

Interview transcripts (bug 1-4)

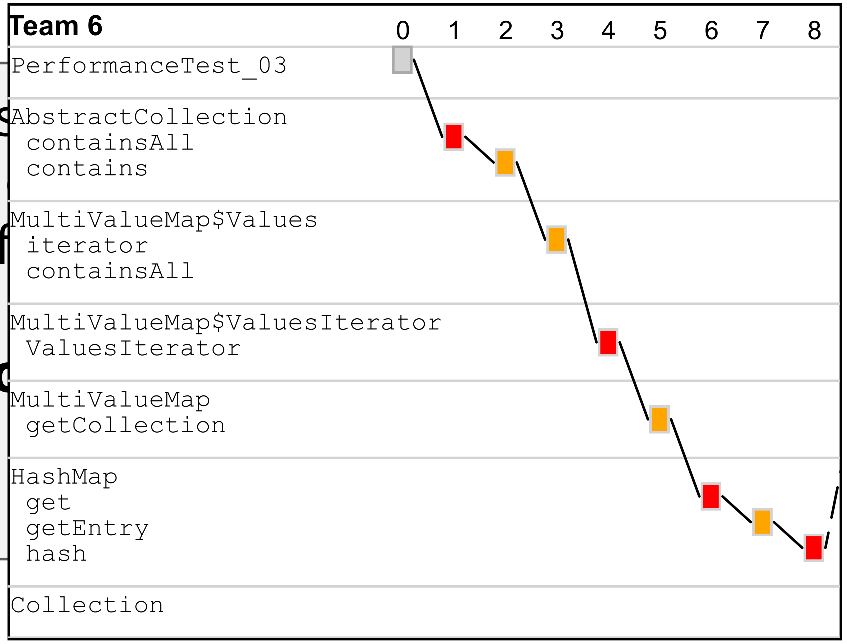
↳ Cross-case analysis [Seaman99]

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).

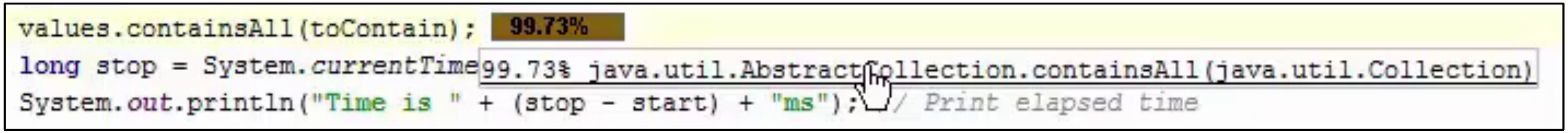
No.	Proposition	Teams
1.1	The dynamic instance of a method call and connected runtime information are important for navigation.	T1, T3, T4, T5
1.2	Following high quantities of runtime in the dynamic method call graph is helpful as a navigation strategy.	T1, T2, T3, T6
1.3	The more complex the performance bug is, the less helpful 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



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)



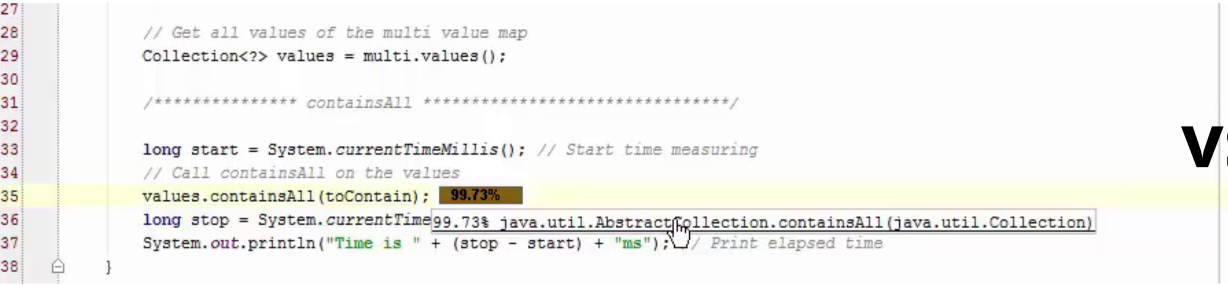
- **Helpful strategy:** Following high quantities of runtime in dynamic call graph (Prop. 1.2)
- **But:** The more complex the performance bug is, the less helpful the provided information becomes (Prop. 1.3)

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)



vs.

- **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: Navigation

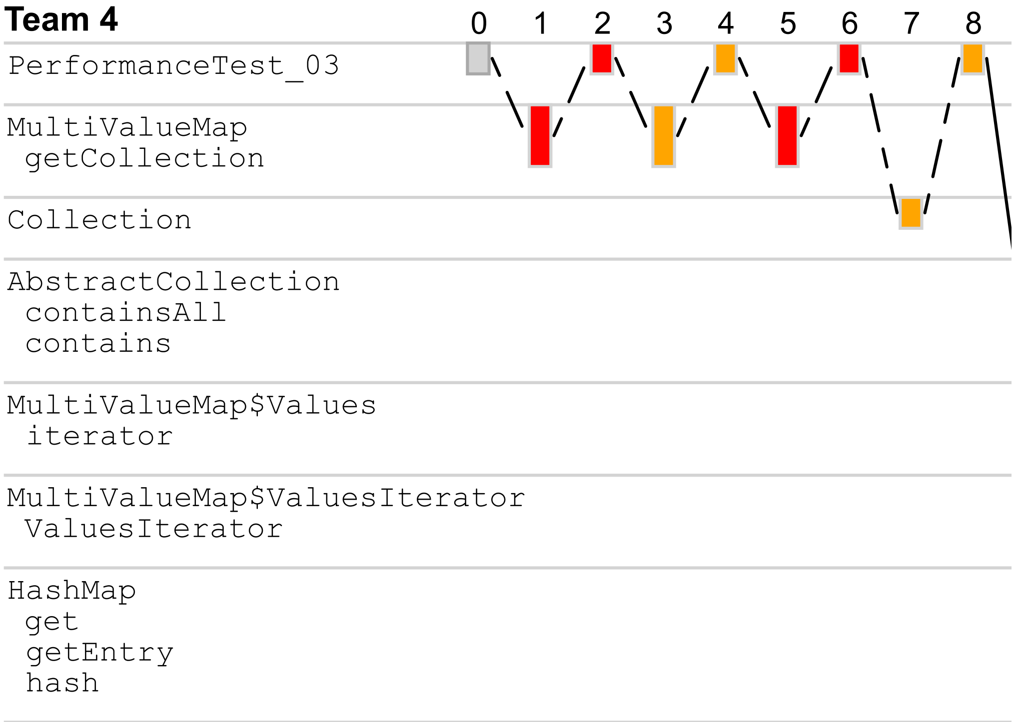
RQ1.3: What navigation strategies do developers pursue to locate a performance bug?

- 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:

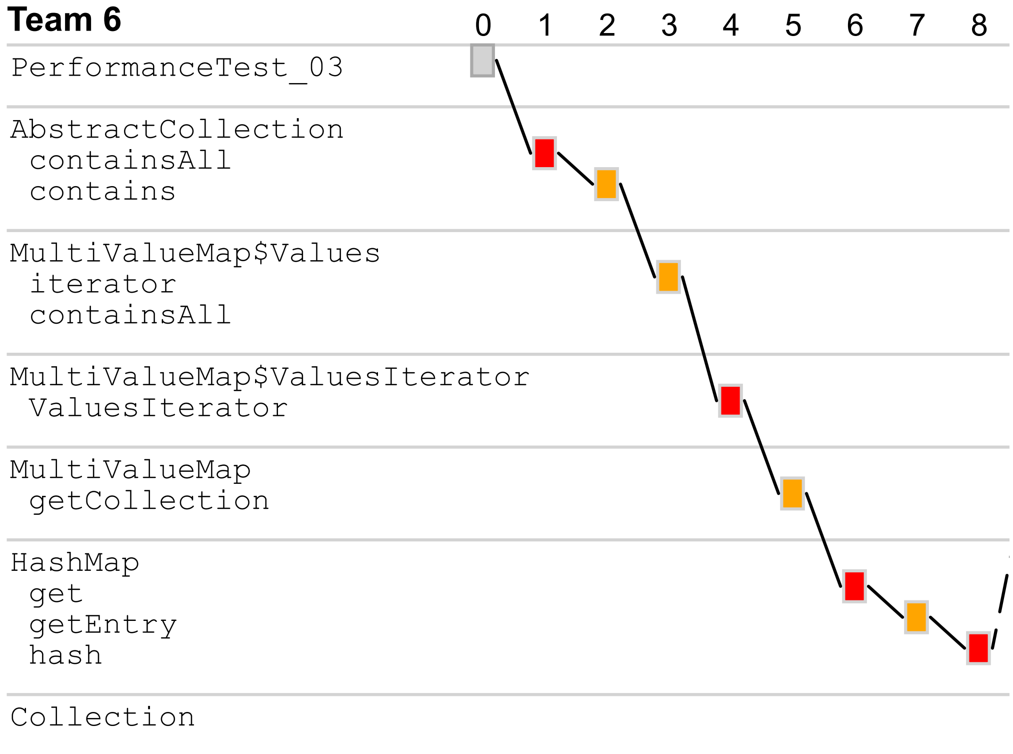
Strategy 1 (Toggle): Frequent switching between test class and and other classes related to bug (IDE navigation).

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

RQ1: Navigation



Strategy 1
(Toggle)



Strategy 2
(Path Following)

Results – RQ2

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

No.	Proposition
3.1	Sketches are a useful tool mance bug, but context in understand them afterwards.
3.2	Sketches are a suitable d "polished" enough).
3.3	If and how much sketching sketching experience of the
3.4	A common sketch vocabular
3.5	More complex problems or likely to be sketched.
3.6	Sketches can be used to exp a program.

TABLE IV. INTERACTIONS WHILE LOCATING PERFORMANCE BUG 3
(D: DURING, A: AFTER LOCATING BUG, *: NAVIGATOR TOOK OVER ROLE OF DRIVER, CODES: SEE TABLE V)

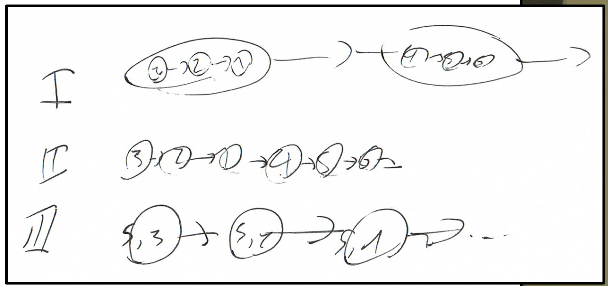
Team	Time (min.)	Success	Driver	Navigator	Codes: Total	DC+HC	DR+HR	QC+QR	PN+PI	CO	RD+RC+RE	Other	First Strategy	Sketch
T1	30	✓	P2	P1	165 45% 55%	46 57% 43%	11 55% 45%	28 21% 79%	5 0% 100%	10 20% 80%	11 55% 45%	54 54% 46%	1	D
T2	30	✓	P4	P3	112 57% 43%	21 67% 33%	19 58% 42%	24 54% 46%	9 11% 89%	6 33% 67%	9 56% 44%	24 75% 25%	1	A
T3	24	✓	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
T4	35	✓	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
T5	20	○	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
T6	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

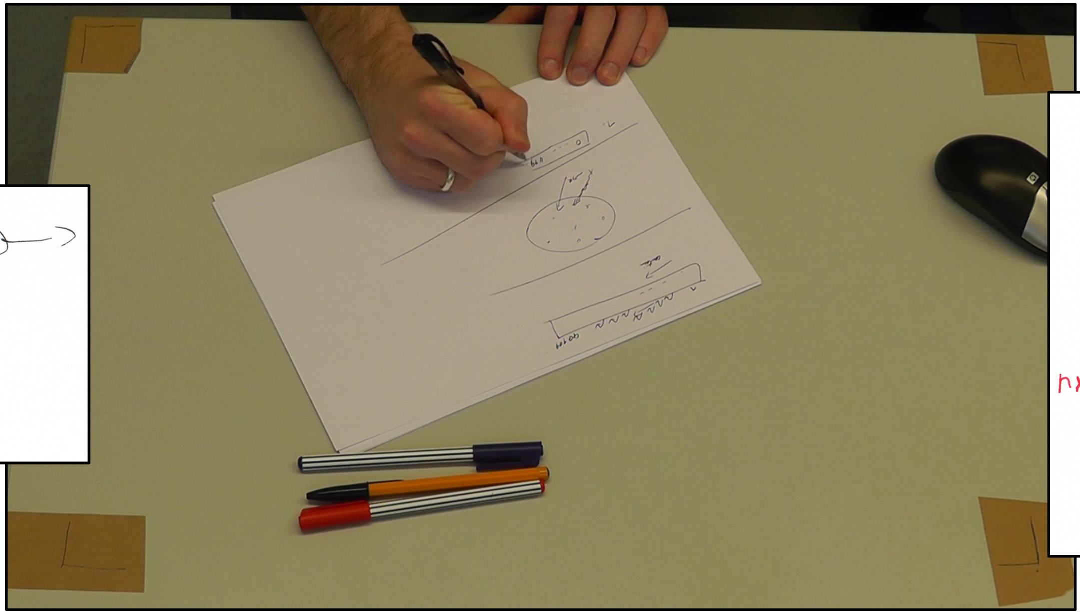
Methods (RQ2)

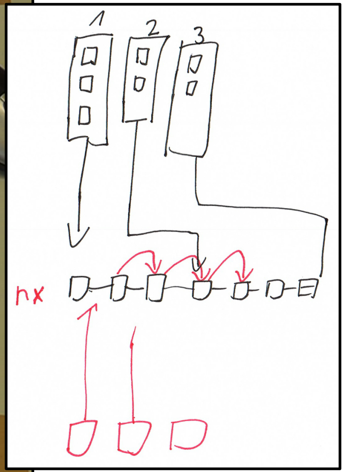
RQ2:

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?

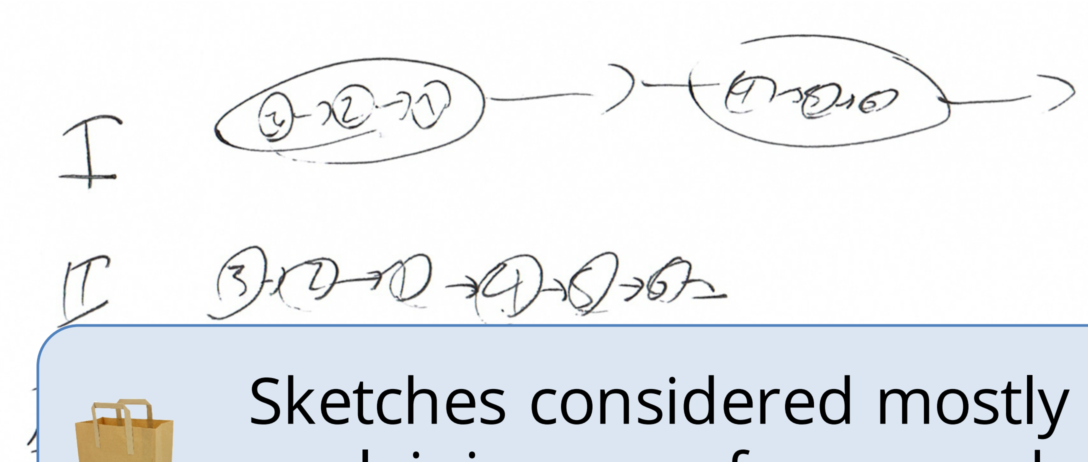
- 4 of 6 teams expressed first hypothesis about cause of bug in the first half of session
- Driver and navigator mostly worked on **same level of abstraction**
- 3 teams had very **active navigator** (e.g. asking questions about code, prompting driver to navigate to certain methods)
- 2 teams had very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

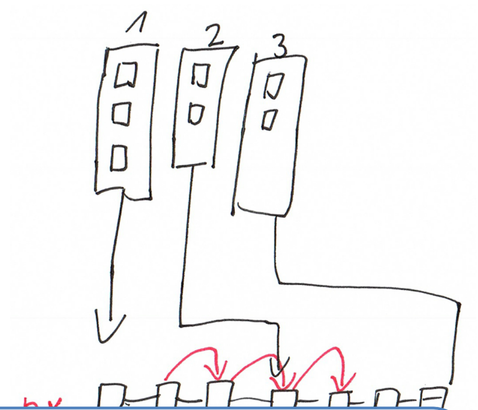
Driver and navigator work on **same level of abstraction**; interaction could be affected by different levels of expertise.

RQ2: 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**
  → 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