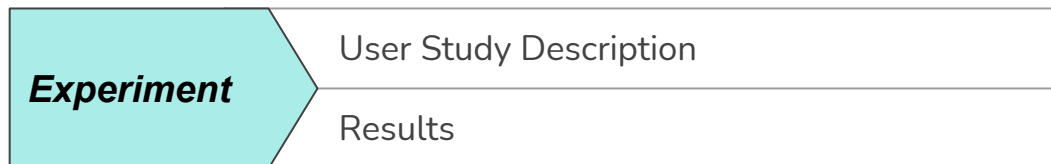
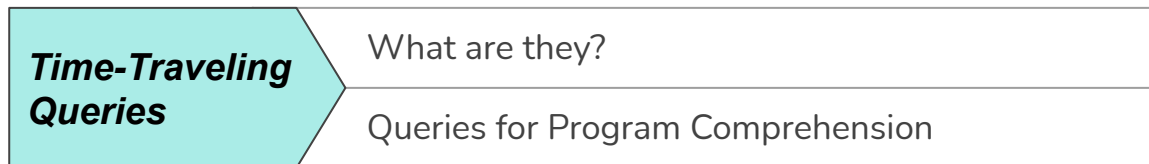
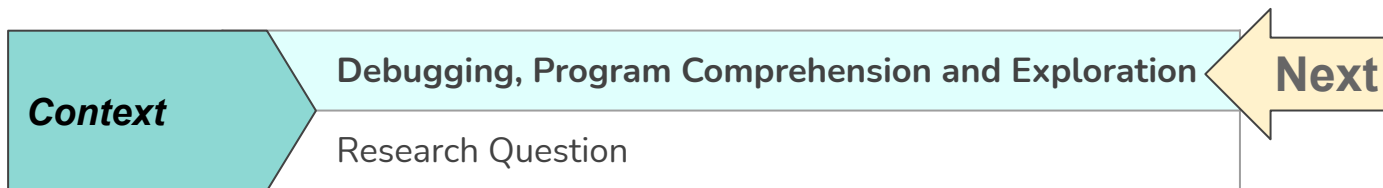


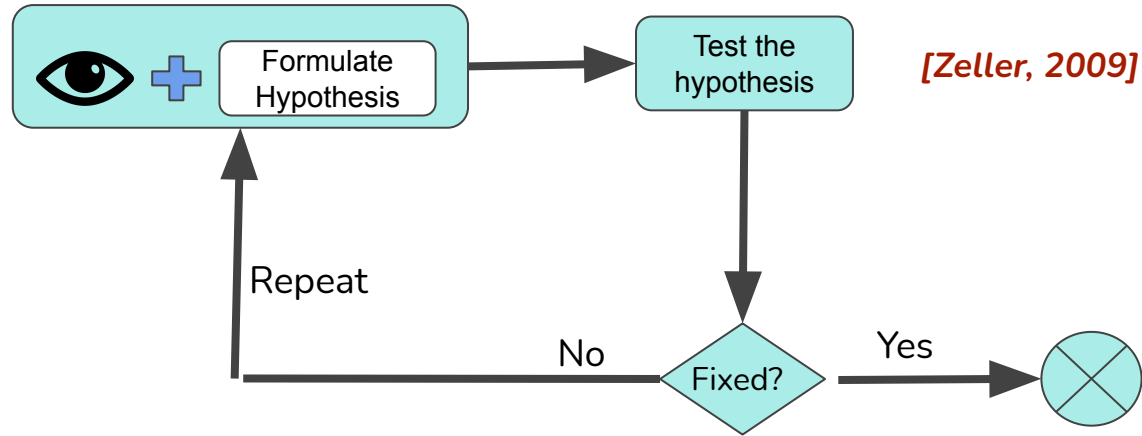
Time Traveling Queries for Faster Program Exploration

Maximilian Willebrinck

Presentation Agenda



Context - The Debugging Process



Good Hypothesis



Less Iterations



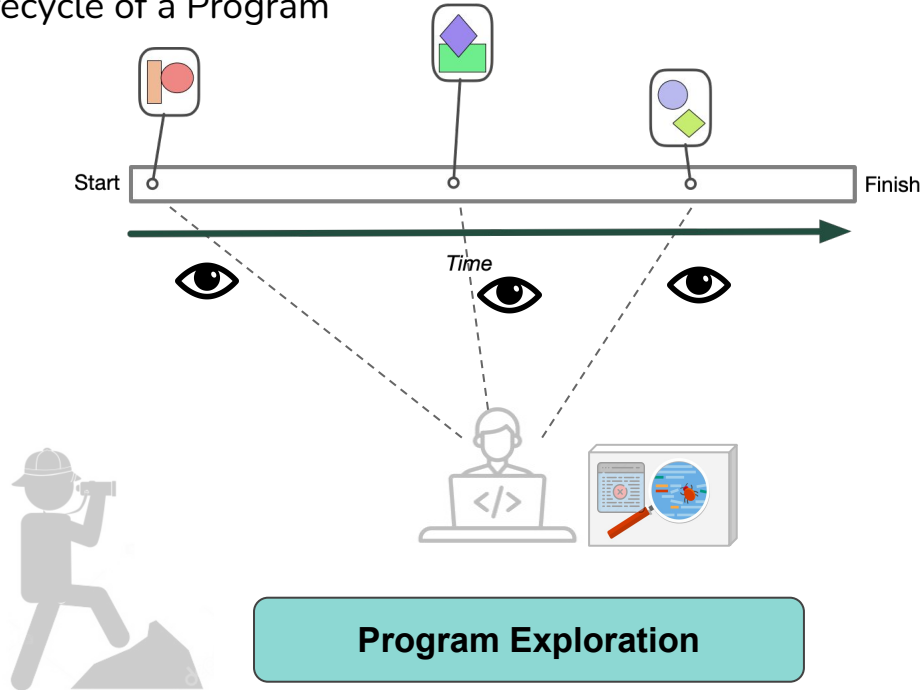
Good Hypothesis



Understand program behavior

Context - Understanding a program behavior

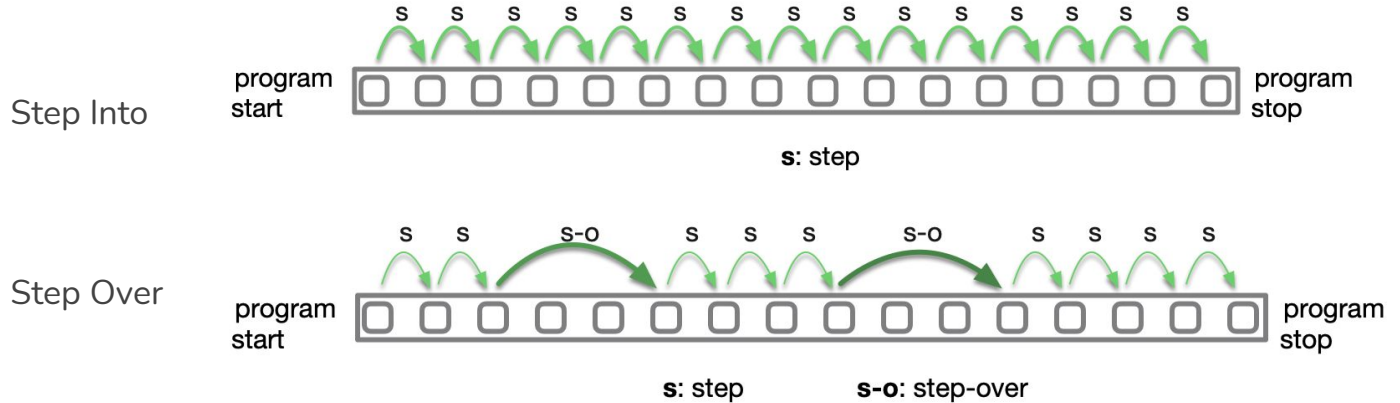
Lifecycle of a Program





Manually exploring a program execution

Stepping



Manual and tedious



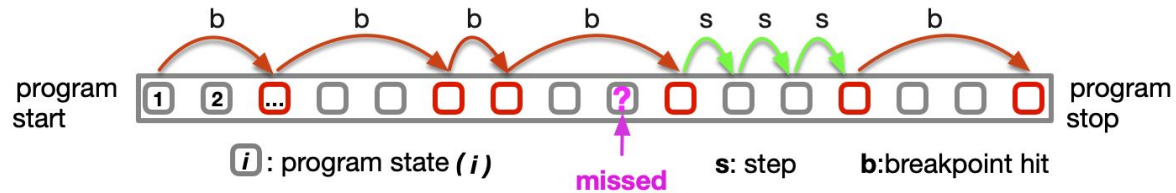
Traverse States Forward in Time





Manually exploring a program execution

Breakpoints



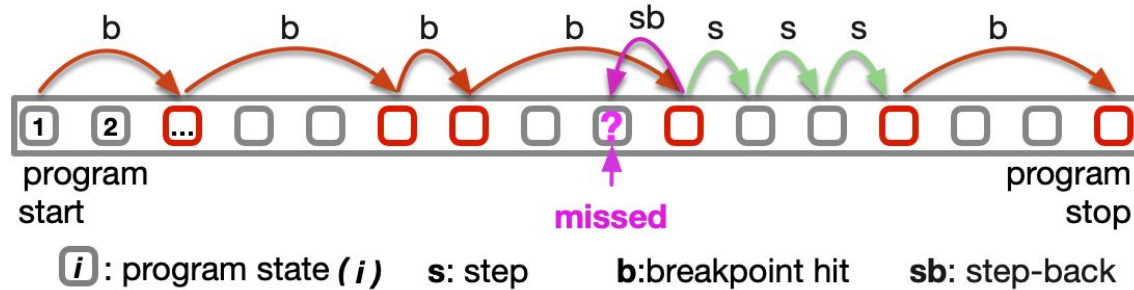
“Missing critical information” problem



Traverse States Forward in Time

Manually exploring a program execution

Time-Traveling Debuggers



Helps with the “Missing critical information” problem.

Still tedious



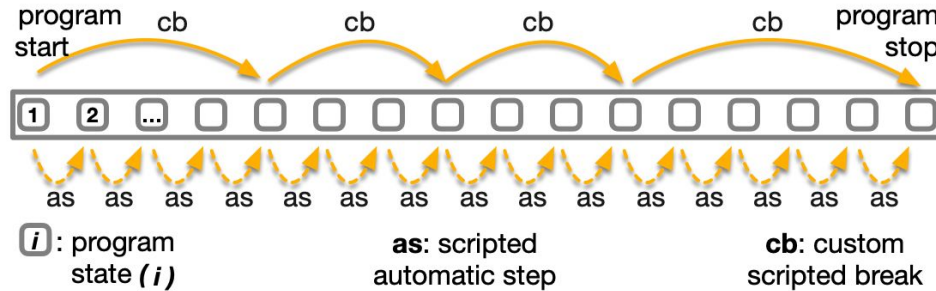
Traverse States Forward in Time





Exploring a program execution

Scriptable Debuggers



Help with tediousness but, requires prior knowledge of the execution



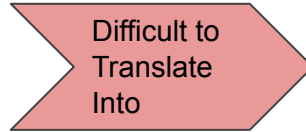


Exploring a program execution

Basic stepping, breakpoints, scriptable debugger, time-traveling, etc.



Debugging Questions



Sequence of Steps



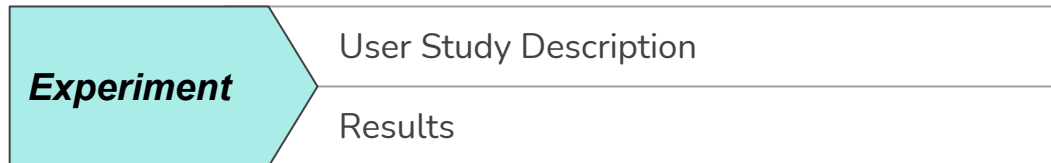
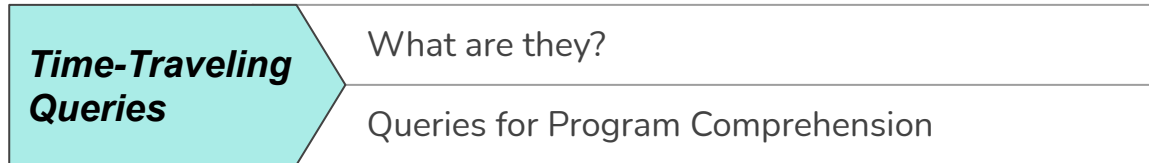
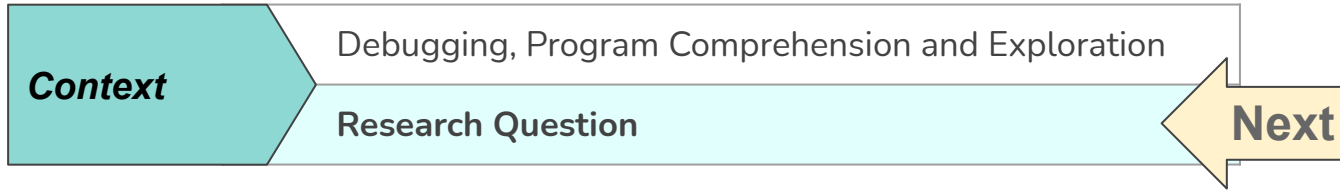


Problem Summary

Program Exploration is ...

- **Manual/Tedious**
- **Imprecise and miss critical information**
- **Translating debugging questions -> debugging actions is difficult**

Presentation Agenda





Research Question

“Can we **express** general program comprehension **questions** as **queries** over programs executions, and does that **improve** program exploration regarding developers' efforts, time spent and precision, **compared to standard debugging tools?**”



Proposed Solution

Time-Traveling Queries

Presentation Agenda

Context	Debugging, Program Comprehension and Exploration
	Research Question

Time-Traveling Queries	What are they?	Next
	Queries for Program Comprehension	

Experiment	User Study Description
	Results



Time Traveling Queries

→ Programmatic requests for execution data



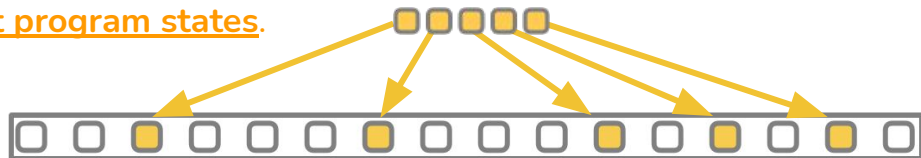
→ Automatically traverse program states...



→ Requesting and collecting relevant data.



→ Enabling direct time travel to relevant program states.





Time-Traveling Queries

Key supporting components

1. Time-Traveling Debugger

Advances or restores an execution to any point in time

2. ProgramStates

An iterable collection of all the program states

3. Query

A programmatic request of execution data

1. Time-Traveling Debugger

- As an **extension** of Pharo 9.0 debugger
- Allows to **reverse** a program's execution (step backwards)
- **Replay-based** Implementation

The screenshot displays the Pharo 9.0 debugger interface. The top window shows the execution of a test case: `[self setUp. self performTest] in RecursionExamples(TestCase)>>runCase`. The stack pane lists the current execution context: `RecursionExamples` (Method: `testDirectRecursionConcatenationSeeker`), `RecursionExamples (TestCase performTest)` (Package: `SUnit-Core`), and `RecursionExamples (TestCase [self setUp. self performTest]) SUnit-Core`.

The main editor shows the source code of `testDirectRecursionConcatenationExample`:

```
1 testDirectRecursionConcatenationExample
2 | stringCollection separator result|
3 stringCollection := { 'One' . 'Two' . 'Three' . 'Four' }.
4 separator := ','.
5 result := self directForwardConcatenation: stringCollection separator: separator .
6 self assert: result equals: 'One,Two,Three,Four'
```

The right pane shows the `Statement Stepping` window, which is currently on line 33: `seeker stepper stepForwardToStatementImmediatelyAfterMethodReturn: (RecursionExamples>>#directFibonacci:).`

The bottom pane shows the `A StDebuggerContext (Recurs...)` window, displaying the current state of the debugger. The `Variable` pane shows the following state:

Type	Variable	Value
implicit	self	RecursionExamples>>#testDirectRecursionConcatenationExample
temp	{ }	stringCollection
temp	separator	an Array [4 items] ('One' 'Two' 'Three' 'Four')
temp	result	One, Two, Three, Four,
attribute	testSelector	testDirectRecursionConcatenationExample
attribute	expectedFails	an Array [0 items] ()
implicit	stackTop	One, Two, Three, Four

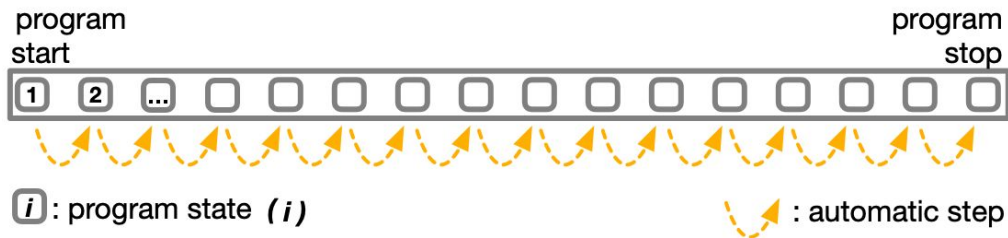
The `Raw` pane shows the current value of `self` as `nil`.



2. ProgramStates

→ A generator of ProgramState

- ◆ Iterable object that exposes an API to retrieve execution data from every state of the program (during its iteration)
- ◆ Every iteration of the loop advances execution by one step
- ◆ No trace(recording) is required to answer queries.





3. Query

→ Declared like this

```
allReturnValuesQuery := Query
  from: programStates
  select: [ :cs | cs isMethodReturn ]
  collect: [ :cs |
    Dictionary newFrom: {
      (#receiverClass -> cs receiverClass).
      (#methodSelector -> cs methodSelector).
      (#returnValue -> cs methodReturnValue) } ].
```

From where to collect?

Which states are relevant?

How to Collect the data?

What to Collect?

Time-Traveling Query Usage

Stack

Class	Method	Package
GLineTest	testAsGLine	Geometry-Tests
GLineTest (TestCase)	performTest	SUnit-Core
GLineTest (TestCase)	[self setUp]. self performTest	SUnit-Core
FullBlockClosure (BlockClosure)	ensure	Kernel

Stepping Control

Query Scripting

No Query executed

Case sensitive filter... (Press enter to apply)

```
1 testAsGLine
2   link Do it
3   set Debug it
```

2 and: 3 , 5. equals: line

Showing 0 results.

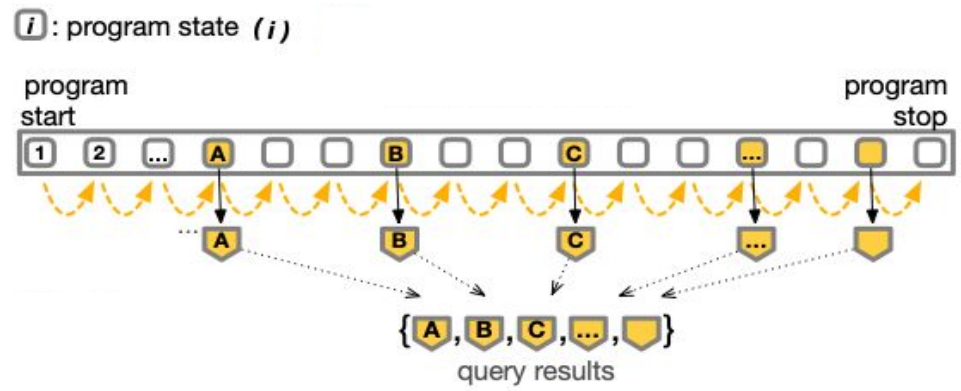
ExecutedBytecode: 203

Queries the execution, from start to finish, for every <instance creation message send>. Every listed result corresponds to the moment before performing the object instantiation. (Sender's point of view)

Messages

- All Instances Creation
- Assignments - Object Centric
- Assignments - General

Query from: programStates
select: X
collect: Y



Query results

→ Shown in UI like this

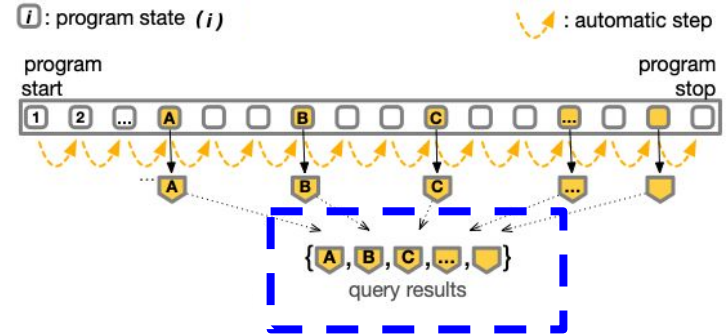
The screenshot shows an IDE window titled "GLineTest->testAsGLine". The main editor displays the following code:

```
1 testAsGLine
2   line := GLine through: 1, 2 and: 3, 5.
3   self assert: line asGLine equals: line
```

The "Seeker" window is open, showing a "Query for All Instances Creation" table. The table has columns for "ExecBC", "About to instantiate a:", and "Sender's Class".

ExecBC	About to instantiate a:	Sender's Class
1 1862	GPoint [Geometry]	GPoint class
2 1874	G2DCoordinates [Geometry]	G2DCoordinates class
3 1935	GPoint [Geometry]	GPoint class
4 1947	G2DCoordinates [Geometry]	G2DCoordinates class
5 2002	GLine [Geometry]	GLine class
6 2182	OrderedCollection [Collections-Sequence...	OrderedCollection class
7 2270	UndefinedObject [#undefined]	Array class
8 2273	WriteStream [Collections-Streams]	WriteStream class
9 2391	OrderedCollection [Collections-Sequence...	OrderedCollection class
10 2479	UndefinedObject [#undefined]	Array class
11 2482	WriteStream [Collections-Streams]	WriteStream class
	OrderedCollection [Collections-Sequence...	OrderedCollection class

Below the table, it says "Showing 0 results. from a total of 35 unfiltered results, fetched in: 571ms." and "ExecutedBytecode: 2038 (23.30% of known execution)".



Time-Traveling from Results

Stack

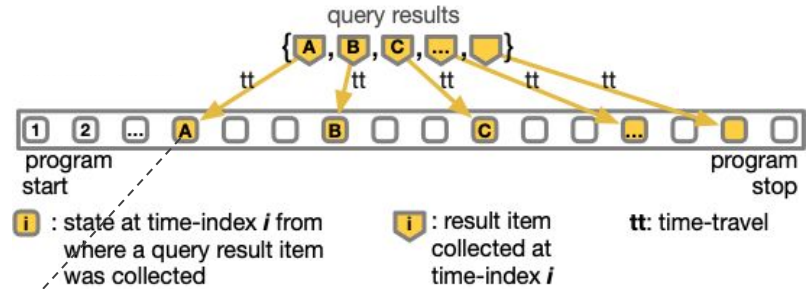
Class	Method	Package
GLineTest	testAsGLine	Geometry-Tests
GLineTest (TestCase)	performTest	SUnit-Core
GLineTest (TestCase)	[self setUp. self performTest SUnit-Core	SUnit-Core
FullBlockClosure (BlockClosure)		Kernel

```

1 testAsGLine
2   line := GLine through: 1, 2 and: 3, 5.
3   self assert: line asGLine equals: line
  
```

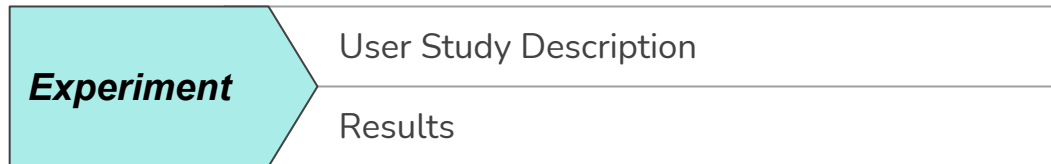
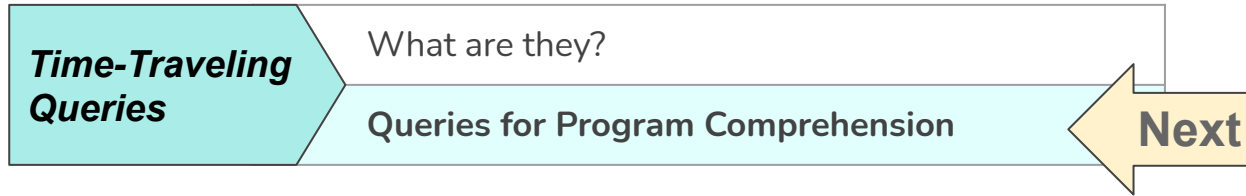
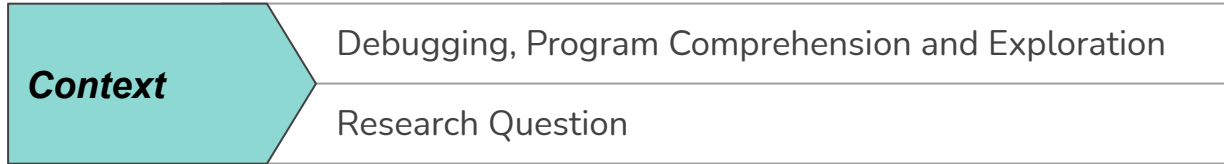
ExecBC	About to instantiate a:	Sender's Class
1	1862	GPoint [Geometry]
2	1874	G2DCoordinates [Geometry]
3	1935	GPoint [Geometry]
4	1947	G2DCoordinates [Geometry]
5	2002	GLine [Geometry]
6	2182	OrderedCollection [Collections-Sequence-...
7	2270	UndefinedObject [#undefined]
8	2273	WriteStream [Collections-Streams]
9	2391	OrderedCollection [Collections-Sequence-...
10	2479	UndefinedObject [#undefined]
11	2482	WriteStream [Collections-Streams]

Showing 0 results. ExecutedBytecode: 2098 (23.30% of known execution)



time-index

Presentation Agenda





Queries for program comprehension

List of Time-Traveling queries

I Messages.

- I.1 Find all messages sent during the execution.
- I.2 Find all messages, with a given selector, sent during the execution.
- I.3 Find all received messages.

II Instances Creation.

- II.1 Find all instance creations.
- II.2 Find all instance creations of a class with a given name.
- II.3 Find all instance creations of exceptions.

III Assignments - Object Centric.

- III.1 Find all assignments of instance variables for the receiver of the currently executed method.
- III.2 Find all assignments of instance variables for a particular instance.
- III.3 Find all assignments of a given instance variable for the receiver of the currently executed method.

IV Assignments - General.

- IV.1 Find all assignments of variables with a given name.
- IV.2 Find all assignments of any variable.
- IV.3 Find all assignments of instance variables for instances of a given class.



Queries for program comprehension

Based on questions from the literature

[Sillito, 2006]

[Kubelka, 2014]

- [13.] When during the execution is this method called?
- [14.] Where are instances of this class created?
- [15.] Where is this variable or data structure being accessed?
- [19.] What are the values of these arguments at run time?
- [20.] What data is being modified in this code?
- [32.] Under what circumstances is this method called or exception thrown?

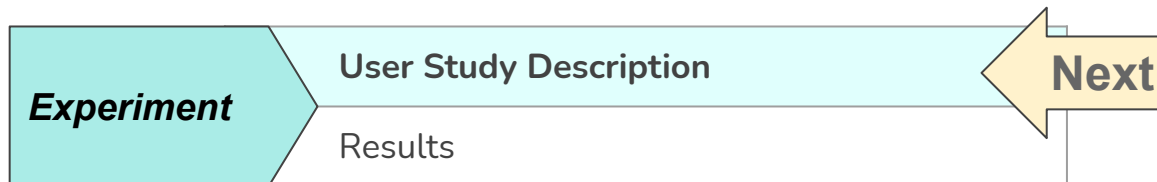
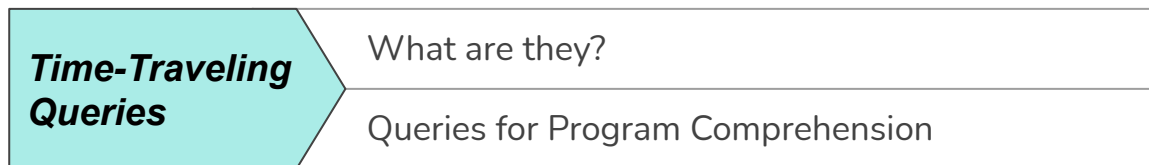
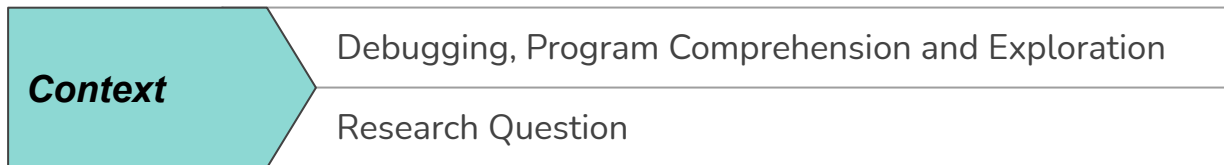


How they help?

List of queries  Translation of questions into scripts/steps

Time-Traveling from results  Less “manual and tedious” traversal

Presentation Agenda





User Study - Queries for Debugging

Evaluated our **Queries** approach vs **Standard** Debugging Techniques, for **program comprehension tasks**:

Do Time Traveling Queries ...

1. Improve **correctness**?
2. Reduce the **employed time**?
3. Reduce the **number of debugging actions**?

(Versus Standard Debugging Tools)



User Study - Experiment Design

- Quantitative experiment
- Repeated Measures Design (Within-subject)
- **34 Participants.**
- Session of **90 minutes**, solving program comprehension tasks, using:
 - ◆ Time-Traveling Queries.
 - ◆ Standard Debugging Tools
- Measure the effect of: “TTQs”
 - ◆ On: Participant **Score, Time, Debugging Actions**
- Followed by Qualitative Survey



Experiment Tasks. From “simpler” ...

- How many times is the method #atEnd of the object 'generator' is called? and from which methods?

testAtEnd

```
| generator |  
generator := self numbersBetween: 1 and: 3.  
self deny: generator atEnd.  
generator next.  
self deny: generator atEnd.  
generator next.  
self deny: generator atEnd.  
generator next.  
self assert: generator atEnd
```



Experiment Tasks. To less “simple”...

- What are the different values of the
`pc` instance variable of the first
`newContext` object during this
test?

testSteppingReturnSelfMethod

```
|newContext|
aMethodContext := Context
  sender: thisContext
  receiver: SimulationMock new
  method: (SimulationMock >>#exampleSelfReturnCall)
  arguments: #().

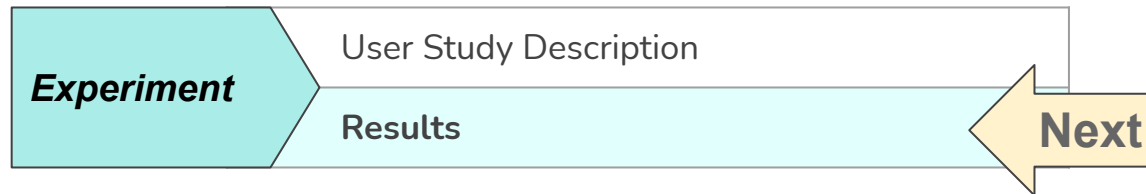
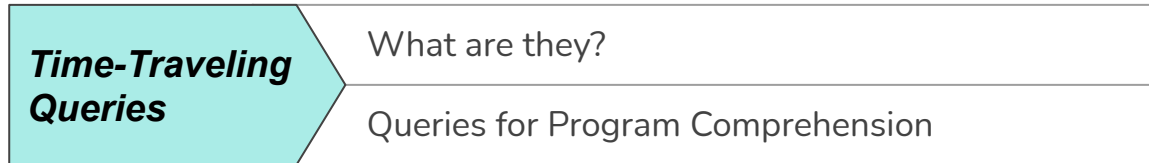
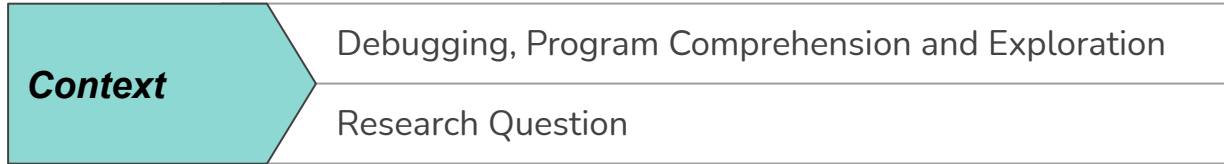
aMethodContext step.
aMethodContext stepIntoQuickMethod: true.
newContext := aMethodContext step.

"We're in the quick method now, it should be steppable"
self assert: newContext sender identicalTo: aMethodContext.
self assert: newContext willReturn.

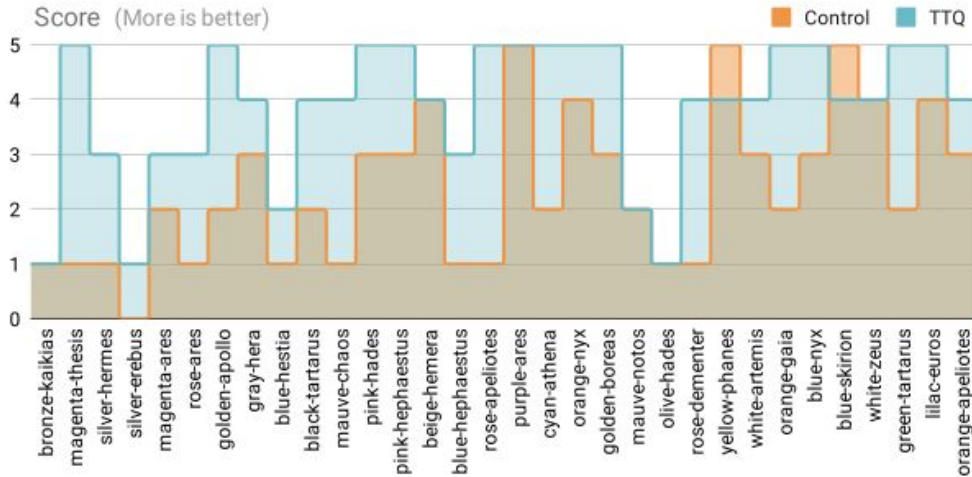
newContext := newContext step.
self assert: newContext identicalTo: aMethodContext
```



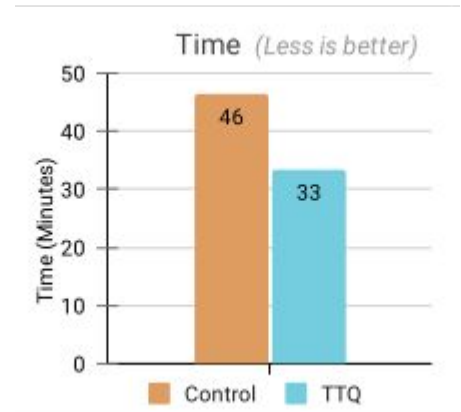
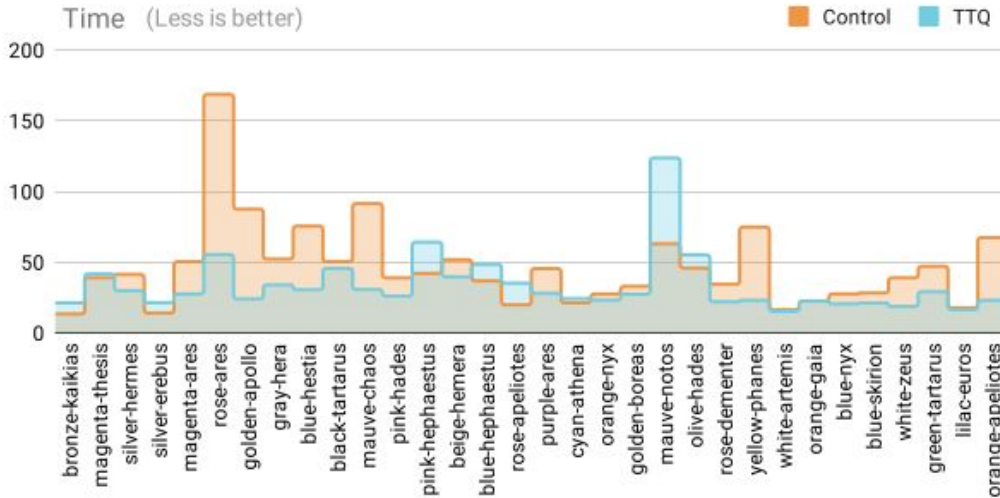
Presentation Agenda



Participants Score

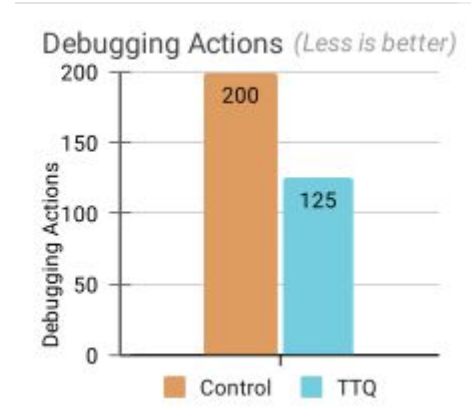
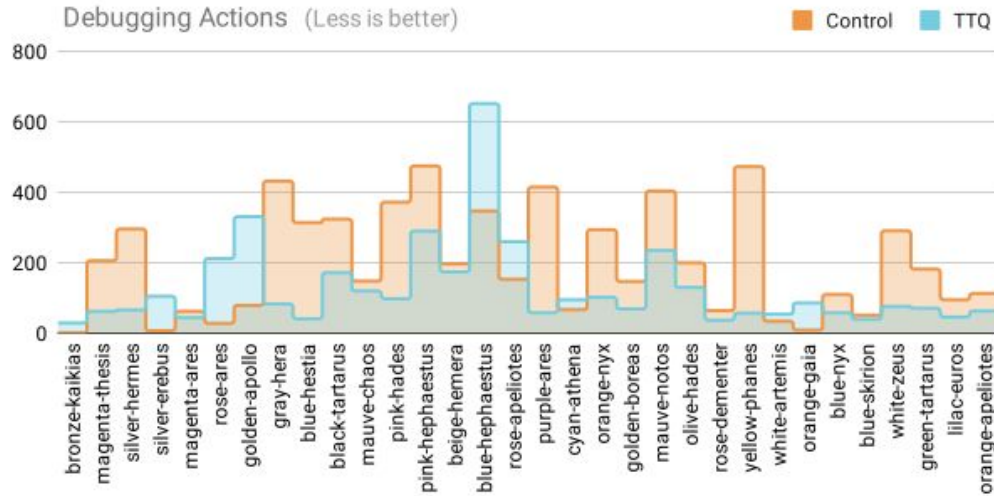


Participants Time



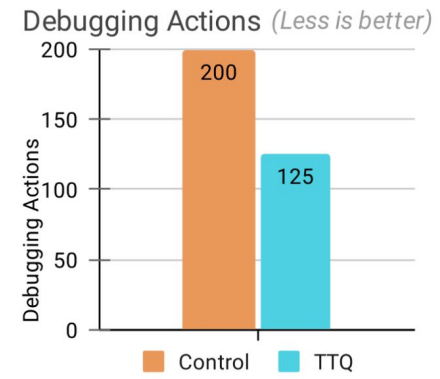
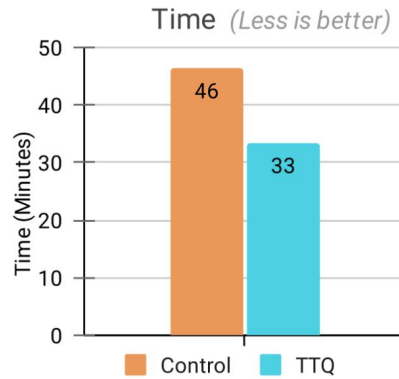
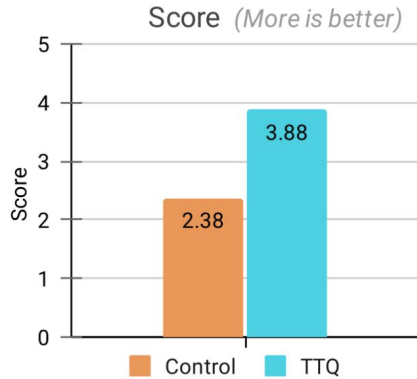
Time to complete all 'Control' and 'TTQs' tasks.

Participants Debugging Actions



Count of debugging actions of participants

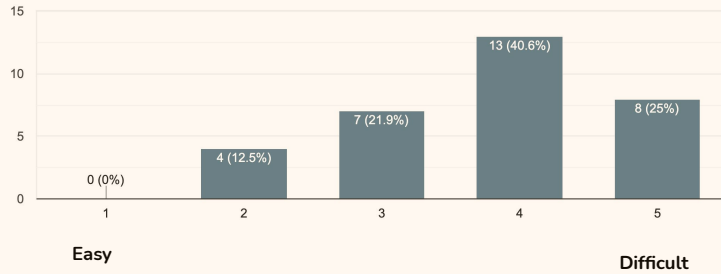
Results Summary



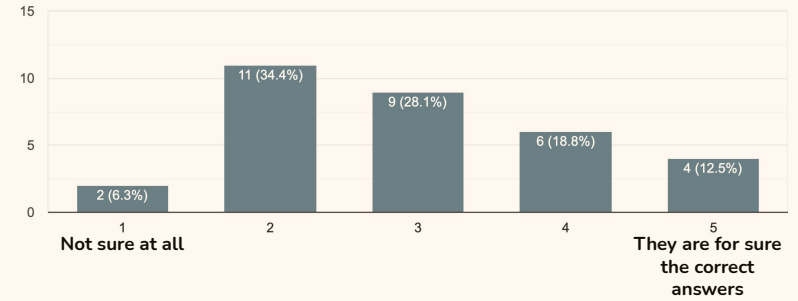
Qualitative Survey

Control

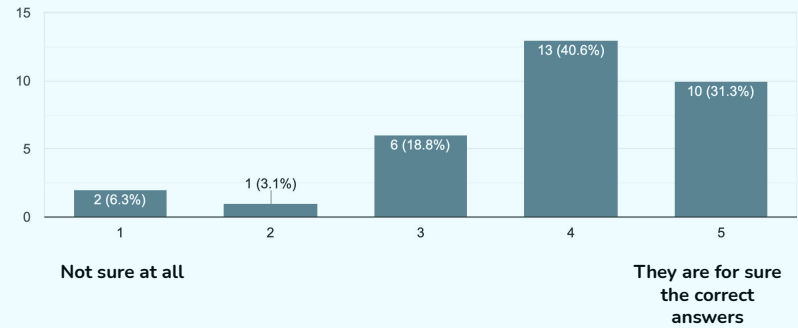
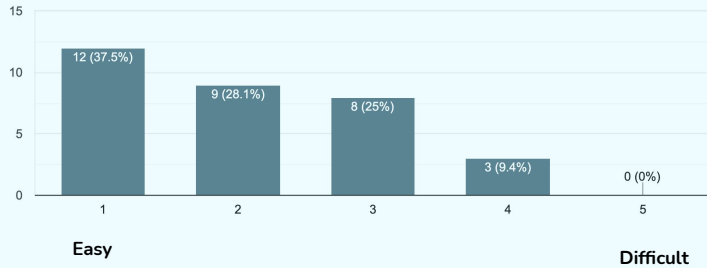
Perceived tasks difficulty



Confidence in answers



TTQs



Qualitative Survey

SeekerDebugger with TTQs evaluation





Experiment Conclusion

- We can express general program comprehension questions as queries over programs executions.
- Results show that TTQs improve program exploration regarding developers' efforts, time spent and precision, compared to standard debugging tools.
- Even with little instruction time for participants, the results were positive.
- Current TTQs don't cover the complete set of problems developers face during their debugging sessions.



Summary

- Different tools and methodologies for program understanding.
- Program exploration using interactive debuggers remains difficult and tedious.
- Proposed TTQs to improve exploration and comprehension.
- Controlled experiment to evaluate our solution.
- With TTQs, developers perform program comprehension tasks more accurately, faster, and with less effort than with standard debugging tools.
- We will continue Time-Traveling Queries research:
 - New relevant queries
 - Improving Time-Traveling Debugger limitations.

