Asking and Answering Why and Why Not Questions about Program Behavior

Amy J. Ko

Brad Myers

Information School
UNIVERSITY of WASHINGTON

Carnegie Mellon
... identifying and correcting defects during the software development process represents over \textit{half} of development costs ... and accounts for \textit{30 to 90 percent} of labor expended to produce a working program.”

National Institute of Standards and Technology, 2002

Testing, debugging, deployment, maintenance...

Initial development
why is debugging so difficult?

four studies to find out...

10 Alice developers in the lab and field

30 Java developers using Eclipse

30 students learning Visual Studio

18 software teams at Microsoft
the problem

today’s tools require people to guess what code is responsible
one bug, **two symptoms**

**why** didn’t this color panel change?

**why** is this stroke black?
debugging with current tools

why is the stroke black?

maybe a slider initialization problem...

maybe the slider isn’t connected to anything...

is the JSlider argument incorrect?

maybe the color isn’t computed properly...

breakpoint

println()

10 minutes 30× speed
debugging with research tools

- reverse execution: guess where to pause execution
- visualizing execution: guess what to look for
- program slicing: guess what code to slice on
- asserting behavior: guess what properties won’t hold
- comparing executions: find successful execution
the **whyline**

what if people could **ask about output** and see the code responsible?
whyline for Java
why was the line black?
record the problem
load the recording

Resolving classes (856 remaining)
why was the line color black?
why was the line color black?
why was the line color **black**?

selected dependency highlighted in source

followup questions about selected event
why was the line color black?

because gSlider was used twice, ignoring bSlider
why didn’t the panel repaint?
find the appropriate time
click on relevant output

objects related to rectangle

fields and methods of selected object
it did paint...

this method did execute!
where did **black** come from?

```java
private void paintComponent(Graphics g) {
    Color oldColor = g.getColor();
    g.setColor(objectConstructor.getColor());
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(oldColor);
}
```

Q why didn't paintComponent() execute?
A Check the answer below.

```
start of program
```

PaintWindow.java:43 didn't execute because This line *did* execute.
found the **bug**

why did getColor() return black?

same buggy code (gSlider used twice)
how does the **Whyline** work?
the whyline cycle

developer...
edit  compile  debug  fix

system...
|  |  |
instruments bytecode  records thread history  converts serial history to random access
|  |  |
ask

Andrew J. Ko
find primitive output statements

$\text{drawString}(x, y, \text{string})$

$\text{drawLine}(x, y, \text{width, height})$

$\text{setColor}(\text{color})$
extract primitive questions

`drawString(x, y, string)`

`drawLine(x, y, width, height)`

`setColor(color)`

why did argument = value?
find output-invoking classes

class PencilPaint
    draw() {
        ...
        drawLine(x1, y1, x2, y2)
    }

upstream control dependencies
Why did subject get created?
Why did variable have this value?
Why didn’t variable change?

```java
class PencilPaint {
   draw() {
      ... drawLine(x1, y1, x2, y2)
   }
}
```
find output-affecting fields

ComboBox combo = new ComboBox(model)
...

upstream data dependencies
ComboBox combo = new ComboBox(model)
...
paint() {

extract output-affecting field questions

sorting field questions by type

“clearButton” has many fields

questions organized by primitives and superclass

i.e., three fields of type Dimension2D
filtering questions by **familiarity**

intermediaries, delegates, proxies, helpers, etc.

- may be unfamiliar

- **familiarity** = classes...
  - declared in editable code
  - referenced in editable code

- only include questions about **familiar classes**
‘why did’ answers

answer derived with **precise dynamic slicing**

a timeline visualization of dependencies

- **control** dependencies as **nested blocks**
- **data** dependencies **inside** of blocks
‘why didn’t’ answers

answer with **call graph reachability** analysis

a visualization of a **subgraph of the call graph**, with

unexecuted **methods** and **branches**

misdirected **calls** and **branches**
how effective is the Whyline?
effectiveness

in a study of two ArgoUML bugs, developers with the Whyline were ...

2x as fast

<table>
<thead>
<tr>
<th>time (min)</th>
<th>whyline</th>
<th>control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

successful 3x as often

<table>
<thead>
<tr>
<th>successful</th>
<th>whyline</th>
<th>control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# successful
performance

memory and performance (see paper)

slow to load traces

fast to answer questions

infeasible for long executions

instrumenting real time software changes behavior
limitations

quality of question phrasing $\propto$

quality of identifiers

question and answer precision $\propto$

type information
limitations

good for *causal* explanations
not change suggestions

good for ‘*where* is the buggy code’
not ‘why is the code *buggy*’
today’s tools require guessing, costing time, money and accuracy of knowledge.

the whyline limits guesswork by supporting queries on program output.

the whyline saves time, improves success rates.
This work was supported by the National Science Foundation under NSF grant IIS-0329090 and the EUSES consortium under NSF grant ITR CCR-0324770. The author is also supported by an NDSEG fellowship and by a NSF Graduate Research Fellowship.

questions?

download the Java whyline at

http://faculty.washington.edu/ajko

or Google “whyline”
## Slowdown

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Events</th>
<th>YourKit profiler slowdown</th>
<th>Whyline slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binclock</td>
<td>177K</td>
<td>140K</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>jTidy</td>
<td>12K</td>
<td>16 million</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>javac</td>
<td>54K</td>
<td>35 million</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>jEdit</td>
<td>66K</td>
<td>9 million</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>ArgoUML</td>
<td>113K</td>
<td>18 million</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

User interfaces are largely idle
## Trace Size

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Events</th>
<th>Size (mb)</th>
<th>Zipped (mb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binclock</td>
<td>177</td>
<td>140K</td>
<td>5 mb</td>
<td>2 mb</td>
</tr>
<tr>
<td>jTidy</td>
<td>12K</td>
<td>16 million</td>
<td>118 mb</td>
<td>14 mb</td>
</tr>
<tr>
<td>javac</td>
<td>54K</td>
<td>35 million</td>
<td>284 mb</td>
<td>51 mb</td>
</tr>
<tr>
<td>jEdit</td>
<td>66K</td>
<td>9 million</td>
<td>84 mb</td>
<td>12 mb</td>
</tr>
<tr>
<td>ArgoUML</td>
<td>113K</td>
<td>18 million</td>
<td>137 mb</td>
<td>18 mb</td>
</tr>
</tbody>
</table>

# of events $\propto$ complexity of computation