Asking and Answering
Why and Why Not
Questions about
Program Behavior

Andrew Ko
Brad Myers
Asking and Answering Why and Why Not Questions about Program Behavior

Andrew Ko
Brad Myers

now at the University of Washington Information School!
... identifying and correcting defects during the software development process represents over **half** of development **costs** ... and accounts for **30 to 90 percent of labor** expended to produce a working program.”

National Institute of Standards and Technology, 2002
... identifying and correcting defects during the software development process represents over half of development costs ... and accounts for 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 developer teams at Microsoft
the problem

today’s tools require people to guess what code is responsible
the problem

today’s tools require people to guess what code is responsible
a bug in a painting program
a bug in a painting program

why is this stroke black?

why didn't this color panel change?

why is this stroke black?
debugging with current tools

why is the 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...

(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 will not hold
comparing executions find successful execution
the whyline

what if people could ask about output and see the code responsible?
the whyline

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

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

why was the line color black?
why was the line color black?
why was the line color black?
because gSlider was used twice, ignoring bSlider

why was the line color black?
why didn’t the panel repaint?
why didn’t the panel repaint?
find the appropriate time
find the appropriate time
objects related to rectangle

fields and methods of selected object

click on relevant output
it did paint...
it did paint...

this method did execute!

this method did execute!
where did black come from?

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

thread main-0

thread thread

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

` WHY DIDN'T PAINTCOMPONENT() EXECUTE? `
where did black come from?

public void stateChanged(ChangeEvent changeEvent) {
    gSlider.setValue(0);
    gSlider.getValue();
    repaint();
}

private JComponent currentColorComponent = new JComponent() {
    Color oldColor = g.getColor();
    g.setColor(objectConstructor.getColor());
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(oldColor);
};

public PaintWindow(int initialWidth, int initialHeight) {
    super("Paint");
}

Q why didn't paintComponent() execute?

A Check the answer below.

thread main-0  ···  thread AWTEventQueue-5

PaintWindow:43 didn't execute because This line did execute.

step forward to getColor() call

where did black come from?
why did getColor() return black?
found the bug

same buggy code (gSlider used twice)
a typical cycle

developer...

edit compile debug fix ...
the whyline cycle

developer...
edit compile record load ask fix ...

system...
instrument bytecode
record thread history
convert serial history to random access history
extract questions from code
find primitive output statements

\textbf{drawString}(x, y, string)

\textbf{drawLine}(x, y, width, height)

\textbf{setColor}(color)
find primitive output statements

**drawString**(x, y, string)

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

**setColor**(color)
extract primitive questions

\textbf{drawString}(x, y, string)

\textbf{drawLine}(x, y, width, height)

\textbf{setColor}(color)

why did \textit{argument} = \textit{value}?
**extract primitive questions**

- **drawString** \((x, y, \text{string})\)
- **drawLine** \((x, y, \text{width}, \text{height})\)
- **setColor** \((\text{color})\)

why did argument = value?
find output-invoking classes

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

upstream control dependencies
**find output-invoking classes**

class **PencilPaint**

draw() {
    ...
    drawLine(x1, y1, x2, y2)

upstream control dependencies
extract output-invoking questions

class PencilPaint

\begin{verbatim}
    draw() {
        ... 
        drawLine(x1, y1, x2, y2)
    }
\end{verbatim}

why did \textit{subject} get created?
why did \textit{variable} have this value?
why didn’t \textit{variable} change?

\begin{itemize}
    \item \textbf{properties of this line}
    \item \textbf{objects} rendering this
    \item \textbf{windows}
    \item \textbf{PencilPaint}
    \item PaintCanvas \textit{“canvas”}
    \item JScrollPane \textit{“canvasPane”}
    \item JPanel \textit{“c”}
    \item PaintWindow
    \item \textbf{why did PencilPaint get created?}
    \item \textbf{thickness}
    \item \textbf{why did \textit{thickness} = 5?}
    \item \textbf{why didn’t \textit{thickness} change?}
    \item \textbf{color}
    \item \textbf{points}
    \item \textbf{why didn’t paint() execute?}
\end{itemize}
extract output-invoking questions

class PencilPaint

draw() {
    ...
drawLine(x1, y1, x2, y2)

why did subject get created?
why did variable have this value?
why didn’t variable change?
find output-affecting fields

ComboBox combo = new ComboBox(model)
...
paint() {
    drawString(model.list.get(index))
}

upstream data dependencies
find output-affecting fields

ComboBox combo = new ComboBox(model)
...
paint() {
    drawString(model.list.get(index))

upstream data dependencies
**extract output-affecting field questions**

ComboBox combo = new ComboBox(model)
...
paint() {
  drawString(model.list.get(index))
}
extract output-affecting field questions

ComboBox combo = new ComboBox(model)
...
paint() {
    drawString(model.list.get(index))
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

```java
class Button
    paint() {
        lookAndFeel.paint()
    }
```

all classes

familiar classes

```
PencilPaint
PaintCanvas "canvas"
ScrollPane "canvasPane"
JPanel "c"
PaintWindow
ComponentUI
ScrollPaneUI
ScrollPane
JScrollPane
ComponentUI
JPanel
WindowUI
PaintWindow
```
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

```java
class Button
  paint() {
    lookAndFeel.paint()
  }
```

all classes

familiar classes

- PencilPaint
- PaintCanvas
- JScrollPane
- JPanel
- WindowUI
- PaintWindow
‘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 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 (in paper)

a visualization of a subgraph of the call graph, with

unexecuted methods and branches

misdirected calls and branches
‘why didn’t’ answers

answer with **call graph reachability analysis** (in paper)

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

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

misdirected **calls** and **branches**

unexecuted method

misdirected branch

unexecuted method

unexecuted method

misdirected execution events
effectiveness (in dissertation)

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

2x as fast

successful 3x as often

<table>
<thead>
<tr>
<th>time (min)</th>
<th>successful #</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
</tr>
</tbody>
</table>

whyline control

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

memory and performance (see paper)

slow to load traces

fast to extract and 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’
summary

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
future work

question coverage

 currently gathering questions that developers want to ask but cannot find or ask

Whyline for other domains

 web programming, applications, document history

teach debugging with the Whyline

Whyline for software development teams
questions

download the Java whyline at
www.cs.cmu.edu/~natprog/whyline-java.html
or Google whyline

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.

Andrew Ko + Brad Myers
questions?

download the Java whyline at

www.cs.cmu.edu/~natprog/whyline-java.html
or Google whyline

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.

Andrew Ko + Brad Myers

Human-Computer Interaction Institute
Carnegie Mellon
# 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>

- influence on UI is negligible
- data heavy programs fare worse
## trace size

<table>
<thead>
<tr>
<th>program</th>
<th>LOC</th>
<th>events</th>
<th>size</th>
<th>zipped</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>(mb)</td>
<td>(mb)</td>
</tr>
<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>

- 7 mb / million events, uncompressed
- 1 mb / million events, compressed
- # of events $\propto$ complexity of computation
# loading time

<table>
<thead>
<tr>
<th>program</th>
<th>LOC</th>
<th>events</th>
<th>loading (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binclock</td>
<td>177</td>
<td>140K</td>
<td>3</td>
</tr>
<tr>
<td>jTidy</td>
<td>12K</td>
<td>16 million</td>
<td>13</td>
</tr>
<tr>
<td>javac</td>
<td>54K</td>
<td>35 million</td>
<td>17.5</td>
</tr>
<tr>
<td>jEdit</td>
<td>66K</td>
<td>9 million</td>
<td>45</td>
</tr>
<tr>
<td>ArgoUML</td>
<td>113K</td>
<td>18 million</td>
<td>14</td>
</tr>
</tbody>
</table>

- **1 million** events / sec on first load
- **10 million** events / sec on subsequent loads
value sources

- chains of **direct data** dependencies are quite long
- whyline shows data **sources**
  
  continue following dependencies until reaching constant, allocation, or expression with multiple dependencies

- user can show **direct dependency** or **source**

```java
paint(new Color(0,0,0))
paintChildren(children, color)
paint(color)
paintChildren(color)
setColor(color)
```

source
value sources

- chains of **direct data** dependencies are quite long
- whyline shows data **sources**
  - continue following dependencies until reaching constant, allocation, or expression with multiple dependencies
- user can show **direct dependency** or **source**

```java
paint(new Color(0,0,0))
paintChildren(children, color)
paint(color)
paintChildren(color)
setColor(color)
```

source
subject program

- ArgoUML, an open source software design tool
- ~150,000 lines of code
- 22 external libraries
- chose two bug reports from version 18.1
  one easy
  one difficult
task 1 results

more successful in half the time
task 2 results

whyline          control

# successful

<table>
<thead>
<tr>
<th></th>
<th>whyline</th>
<th>control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 of 10 gave up

p < .05

more successful in the same time

time (min)

<table>
<thead>
<tr>
<th></th>
<th>whyline</th>
<th>control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>