Asking and Answering Questions about the Causes of Software Behavior

Andrew Ko

Human-Computer Interaction Institute
Carnegie Mellon
software is everywhere
program understanding

an essential and fundamental part of

fixing bugs...
adding features...
maintaining legacy code...
adapting code for new purposes...
reusing components...
... 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 program understanding difficult?

what could make understanding easier?
studies of program understanding in multiple contexts

technologies for different populations of users

evaluations of these technologies
studies of program understanding in multiple contexts

technologies for different populations of users

evaluations of these technologies
outline

problem

studies

the whyline

implementation

evaluation

conclusions
related work

computer science ed

programming languages

psychology of programming

human-computer interaction

software engineering
novices using Alice

- 6 participants
- varying programming experience
- created a simple Pac Man game
- asked to think aloud
- 2 hour session
- videotaped from behind
novices using Alice

33% why did

67% why didn’t questions about program output

100% of first hypotheses were wrong

46% time spent debugging

50% of errors added as a result of false hypotheses
students learning Visual Basic

- 30 students learning VB.NET.
- 4 programming assignments
- 2 TAs available in computer lab
- when asked for help, TAs recorded
  - what student was “stuck” on
  - how they became stuck
  - what student did to become “unstuck”
students learning Visual Basic

- struggled to form hypotheses
- consulted peers for hypotheses

In 11% of cases, couldn't think way to test hypothesis

students misperceived program output, investigating non-issues
debugging in Eclipse

- 31 Java programmers
- 3 debugging tasks
- 2 enhancement tasks
- worked on a painting program
- used Eclipse 2.0 and the web
- screen captured
debugging in Eclipse

- 88% of hypotheses were false
- 49% of time spent checking irrelevant code
- Many hypotheses went untested, leading to misunderstandings in later tasks
information needs at Microsoft

observed **25 hours** of coding and bug fixing, in the role of “new hires”

357 pages of handwritten notes

4,231 events in an spreadsheet
information needs at Microsoft
information needs at Microsoft

what code caused this program state?
why was this code implemented this way?
what code could have caused this behavior?
in what situations does this failure occur?
have resources I depend on changed?
what is the program supposed to do?
what have my coworkers been doing?
how do I use this data structure or function?
did I make any mistakes?
is this problem worth fixing?
what’s statically related to this code?
what are the implications of this change?
most common unsatisfied needs

<table>
<thead>
<tr>
<th>Question</th>
<th>% Unsatisfied</th>
<th>Max Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>what code caused this program state?</td>
<td>61%</td>
<td>21 min</td>
</tr>
<tr>
<td>why was this code implemented this way?</td>
<td>44%</td>
<td>21 min</td>
</tr>
<tr>
<td>what code could have caused this behavior?</td>
<td>36%</td>
<td>17 min</td>
</tr>
</tbody>
</table>

- relied heavily on coworkers to answer questions
- long periods of hypothesis refinement
- experts explored many hypotheses in parallel
program understanding is hypothesis-driven...

people ask ‘why’ questions about program output

most initial hypotheses are incorrect

incorrect hypotheses can lead to new bugs, misunderstandings about program execution

true for novices, end-users, Java programmers, industry developers
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
the idea
what if people could **point**
to **output** and see the **code** responsible?
the idea

what if people could **point**

to **output** and see the

**code** responsible?
outline

problem

studies

the whyline

implementation

evaluation

conclusions
outline

problem

studies

the whyline

implementation

evaluation

conclusions
a whyline for Alice
a whyline for Alice
a whyline for Alice
a whyline for Alice
a whyline for Alice

- Increased task completion by 40% \( (p < .05) \)
- Reduced debugging time by a factor of 8 \( (p < .05) \)
The Crystal demo text editor allows multi-font typing, styles on paragraphs and other features copied from Microsoft Word.

I can type just like with any other text editor. Why is this **bold**?

Teh
The Crystal demo text editor allows multi-font typing, styles on paragraphs and other features copied from Microsoft Word.

I can type just like with any other text editor. Why is this bold?

Teh
a whyline for documents

The Crystal Demo Text Editor

Times New Roman

Ask about a location...

Didn't auto-correct "Teh" to "The"

Auto-correct was toggled

Copy wasn't executed

The text was made bold

Auto-correct "I" to "I"

The font size was set to 20

The font family was set to Arial

The text was made italic
a whyline for documents
a whyline for documents

users completed tasks 20% faster ($p < .05$)

users completed 30% more tasks ($p < .05$)
a whyline for Java
a bug
a bug

why didn’t this color panel change?

why is this stroke black?
what normally happens

why is the stroke black?
what normally happens

why is the stroke black?

- maybe the 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, 27× speed)

stumbled onto bug accidentally
whyline demo

at least two ways to ask this question...

why was the **line** color black?

why didn’t the **color panel** repaint?
record the problem
Reading events (1,289,528 remaining)

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?
why was the line color black?

(1) why did color = rgb(0,0,0)? (source)
(2) why did this = PencilPaint #25,299? (producer)
why was the line color black?
public void paint(Graphics2D g) {
    Stroke oldStroke = g.getStroke();
g.setStroke(new BasicStroke(thickness));
g.setColor(color);

    for (int pointIndex = points.length - 1; pointIndex >= 1; pointIndex--){
        Point one = points[pointIndex];
        Point two = points[pointIndex - 1];
        g.drawLine((int)one.getX(), (int)one.getY(), (int)two.getX(), (int)two.getY());
    }

    g.setStroke(oldStroke);
}
why was the line color black?
why was the line color black?
whyline demo

at least two ways to ask this question ...

why was the line color black?

why didn’t the color panel paint?
why didn’t the panel paint?
why didn’t the panel paint?
why didn’t the panel paint?
it did paint...
it did paint...
Q why didn’t paintComponent() execute?

A Check the answer below.

step forward to the color used...
Q why didn’t paintComponent() execute?

A Check the answer below.

find the bug
outline

problem

studies

the whyline

implementation

evaluation

conclusions
outline

problem

studies

the whyline

implementation

evaluation

conclusions
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

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

**fillRect**($x$, $y$, width, height)

**setFont**(font)
find primitive output statements

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

\texttt{fillRect}(x, y, width, height)

\texttt{setFont}(font)
ask primitive questions

drawString(x, y, string)

fillRect(x, y, width, height)

setFont(font)

why did argument = value?
class PencilPaint

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

find output-invoking data
find output-invoking data

class PencilPaint

draw() {
    ...
    drawLine(
        x1, y1,
        x2, y2)
}
ask 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 data

ComboBox combo = new ComboBox(model)
...
paint() {
    drawString(model.list.get(index))
}
find output-affecting data

ComboBox combo = new ComboBox(model)
...
paint() {
    drawString(model.list.get(index))
ask output-affecting data questions

ComboBox combo = new ComboBox(model)
...
paint() {
    drawString(model.list.get(index))
}
filtering questions by ‘familiarity’

class Button
  paint() {
    lookandfeel.paint()
  }

- intermediaries
  look and feel delegates
  proxies

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

- only include questions about familiar code entities
presenting ‘why did’ answers

- answer derived with **precise dynamic slicing**
- a timeline (left to right)
- **control** dependencies as **nested blocks**
- **data** dependencies **inside** of blocks
presenting ‘why did’ answers

- answer derived with precise dynamic slicing
- a timeline (left to right)
- control dependencies as nested blocks
- data dependencies inside of blocks
presenting ‘why didn’t’ answers

- answer derived from static call graph reachability analysis
- a graph of unexecuted methods and instructions
outline

problem
studies
the whyline
implementation
evaluation
conclusions
outline

problem

studies

the whyline

implementation

evaluation

conclusions
a comparison study

Whyline group vs control group

both groups had modern IDE features
show declaration, show callers, show references, etc.
the conventional debugger

*simulated with a Whyline trace*

**Supported**

- breakpoints
- step into
- step over
- step out
- run to breakpoint/line
- pause at selected program output
- print variable value to console

**Unsupported**

- pausing live program
- editing live program
- arbitrary print statements
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 one

ArgoUML bug 3121: “Remove ‘Report Usage Statistics’ since it does not do anything”
task two

ArgoUML bug 3128: “Problems with two classes with the same name in different packages”
Task 2

Sharon needs your help again, this time on a more complicated bug. In the screen shot below, there are three classes. Two have the same name, but are in different packages. A third class has an attribute. The user is trying to select the attribute's type, but notice that the list only includes "MyClass". It should include two classes named "MyClass," one from each package.

Again, you have two responsibilities.

1. Find out why this menu only has one item labeled "MyClass".
2. Write a change recommendation, detailed enough for Sharon to understand the cause of the problem. You should also include at least one idea for a solution to the problem.

You should be confident about the correctness of your recommendation (you don’t want to waste Sharon’s time), but you shouldn’t spend too much time understanding the system, since you’re on a tight schedule and don’t have your code. You can afford to spend up to 30 minutes on this bug.

When 5 minutes remain, I will remind you about writing the recommendation. Tell the experimenter when you are done with your recommendation.
sample

- 20 masters students in software engineering
  - all non-native English speakers
  - 0-10 years in the software industry, median 1.5 years
  - average self-rated Java expertise ("beginner" to "expert" scale)
- groups did not significantly differ on any measures
task 1 results

more successful in half the time

whyline
control

# successful

p<.05

time (min)

p<.05
task 2 results

whyline | control

# successful

4 of 10 gave up

more successful in the same time

p<.05

time (min)
observations

- still need to choose question **carefully**
  
  makes choice **explicit**, unlike current tools

- right questions take **closer** to bug, get you there **faster**
  
  less relevant questions get you there, but with **more work**

- whyline gives **confidence** about **causality**?
  
  control condition got near the bugs but didn’t know it
quotes

“It's so nice and straight and simple...”

“My god, this is so cool...”

“When can I get this for C?”
some limitations

memory and performance can be bottlenecks

infeasible for long executions, real time software

quality of question phrasing $\propto$ quality of identifiers

question and answer precision $\propto$ type information
some limitations

no change suggestions, just causal explanations

good for functional correctness, less for other qualities

good for ‘where is the buggy code’, not ‘why is the code buggy’
current 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

whyline for education

whyline for teams

discovering collaboration requirements

designing annotations and communication tools

the other half of fixing a bug

understanding design rationale behind code

why is the code written this way?

is this bug important to fix?
future work

- information work
- interaction designers’ collaboration with developers
- scientists’ use of technology
- students’ use of statistics
- engineers’ use of specifications
- democratizing access to computing
- new domain-specific languages and tools
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.

thank you to

my thesis committee
Brad Myers
Bonnie John
Jonathan Aldrich
Gail Murphy

past and present
HCII students

my family

and many others...

questions

thanks to
Polo Chau
for this icon!