Big ideas

behind the Whyline

Andy J. Ko, Ph.D. Associate Professor

W UNIVERSITY of WASHINGTON DUB DESIGN USE BUILD PLSE

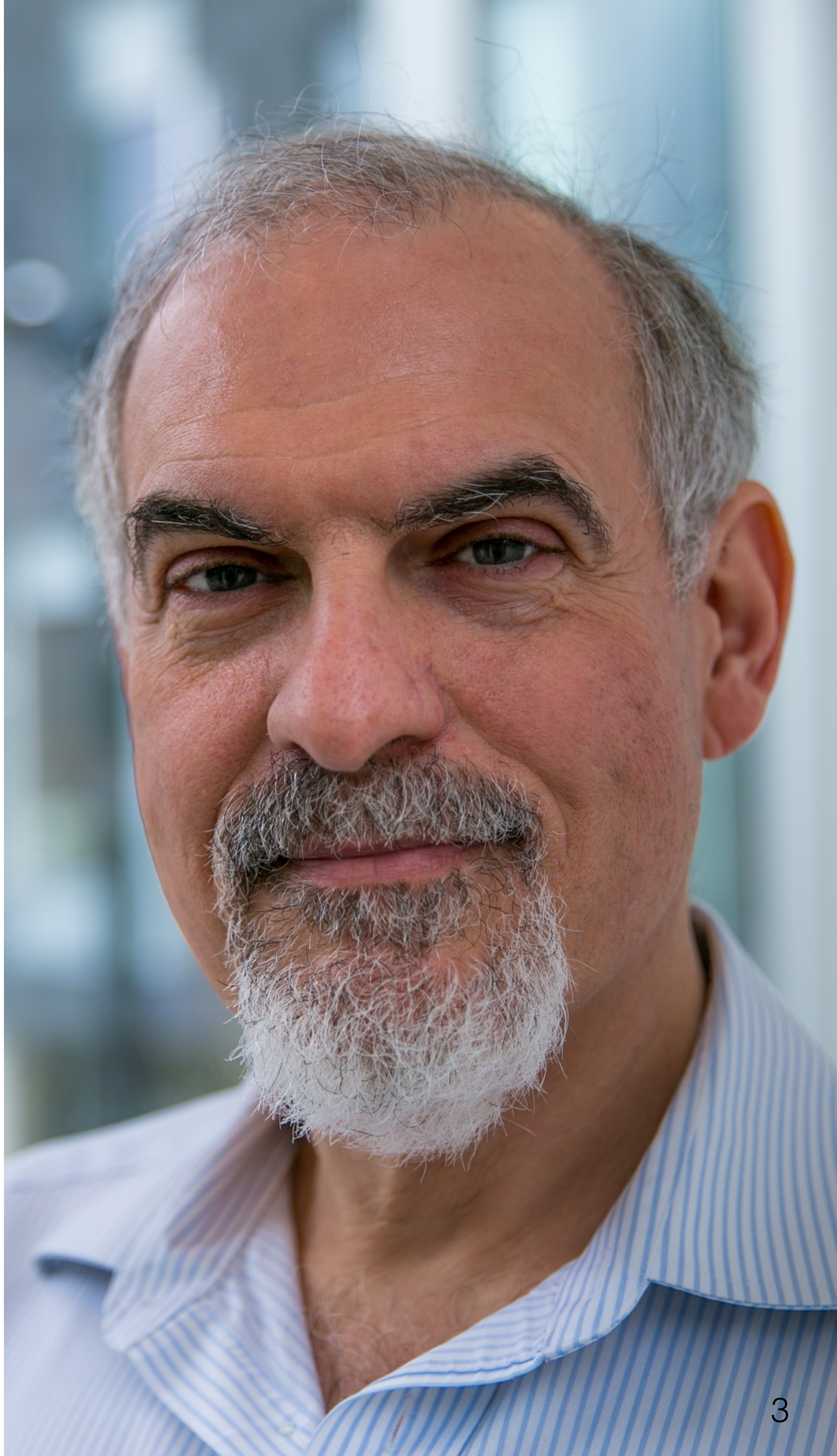
ICSE 2018
Gothenburg, Sweden

Thanks Peggy

- My undergrad research mentor **Margaret Burnett** introduced me to HCI and software engineering
- She taught me how think, how to read, and to develop scientific arguments
- She helped me navigate to graduate school, to connect with other mentors
- I wouldn't be here if she hadn't mentored me for the past 20 years

Thanks Brad

- My Ph.D. advisor, **Brad Myers**, taught me how to choose great projects, how to convey the essence of their insights
- He seeded me with the intriguing idea of asking systems to *explain* themselves
- His relentless constructive critique but unbounded availability helped me learn fast



Thank you

- This community taught me technical rigor, tested the limits of my humanism
- You provided a (then) 30-year history of powerful ideas about dependencies, analysis, architecture, program comprehension, and encapsulation

Thank you academia

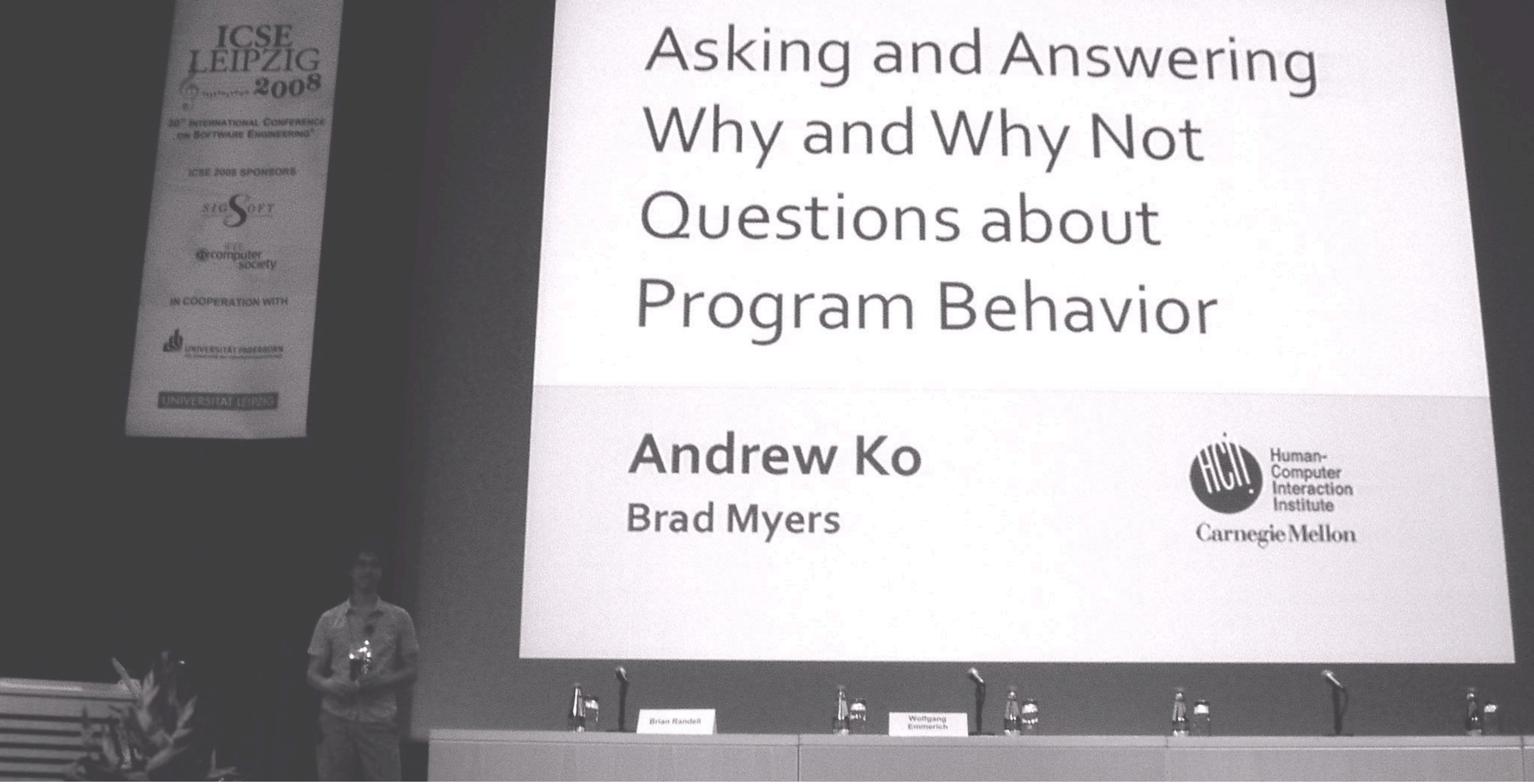
- I've been fortunate to have dozens of outstanding teachers across my life, spanning *math, physics, sociology, psychology, neuroscience, business, English, philosophy, design, art, learning, and chemical engineering*

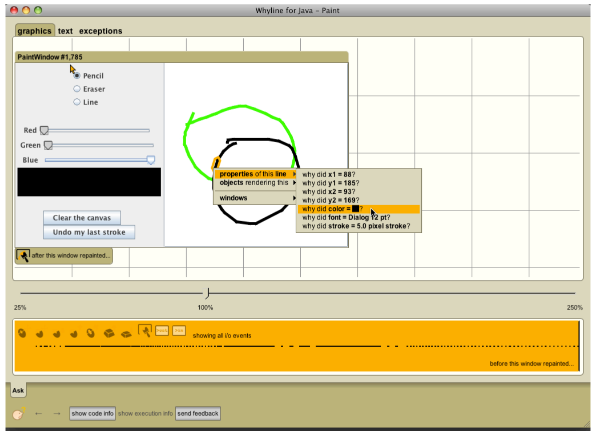
- My ideas are mere compositions of those I've learned from my teachers

Leveque, Nikki Hudson, Bart Vale
lack. Not Pictured Nick Kintz. Belo
unlap, Kathy Smith, Susan Harrel,
ng: Steve Penne, Judy Belk.

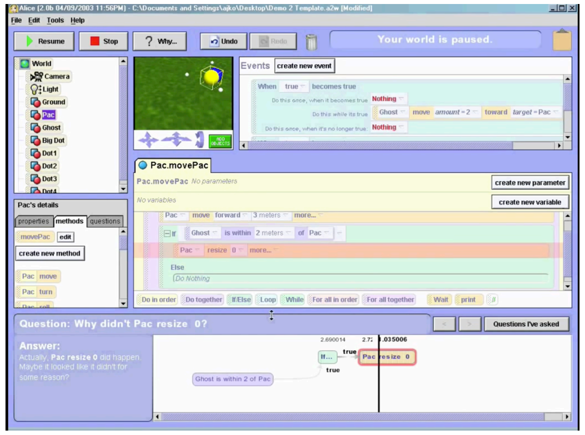
Big ideas

in the Whyline





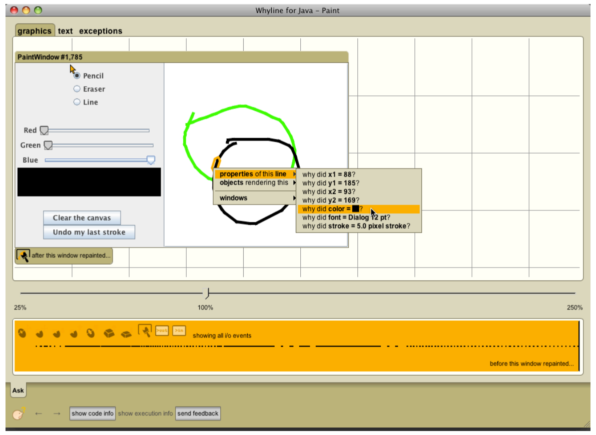
Ko, A. J., & Myers, B. A. (2008)
Debugging reinvented: asking and answering why and why not questions about program behavior.
International Conference on Software Engineering



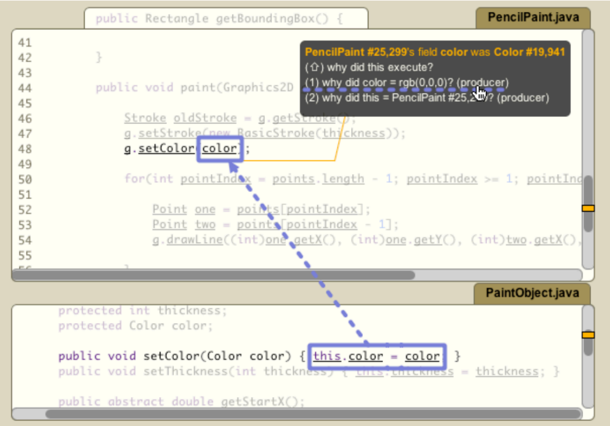
Question: Why didn't Pac resize 0?

Answer:
Actually, Pac resize 0 did happen.
Maybe it looked like it didn't for some reason?

Ko, A.J. and Myers, B.A. (2004) . Designing the Whyline: A Debugging Interface for Asking Questions About Program Failures. ACM CHI.



Ko, A. J., & Myers, B. A. (2008). Debugging reinvented: asking and answering why and why not questions about program behavior. *ICSE.*



```
public Rectangle getBoundingBox() {

        }

        public void paint(Graphics2D

            Stroke oldStroke = g.getStroke();
            g.setStroke(new BasicStroke(thickness));
            g.setColor(color);

            for(int pointIndex = points.length - 1; pointIndex >= 1; pointInd

                Point one = points[pointIndex];
                Point two = points[pointIndex - 1];
                g.drawLine((int)one.getX(), (int)one.getY(), (int)two.getX(),
```

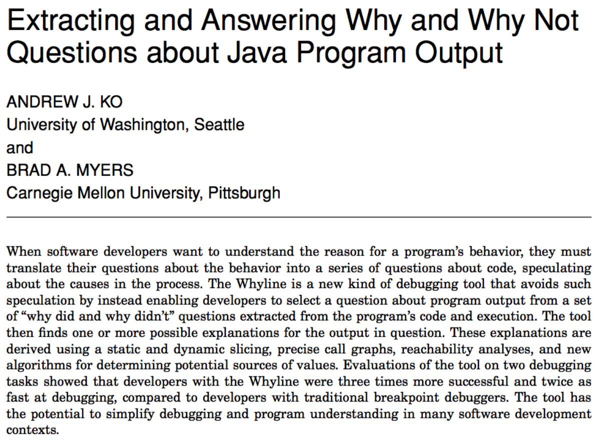
PencilPaint #25,299's field color was Color #19,941
(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (producer)
(2) why did this = PencilPaint #25,299? (producer)

```
    protected int thickness;
    protected Color color;

    public void setColor(Color color) { this.color = color; }
    public void setThickness(int thickness) { this.thickness = thickness; }

    public abstract double getStartX();
```

Ko, A.J. and Myers, B.A. (2009)
Finding Causes of Program Output with the Java Whyline. *ACM CHI.*


Extracting and Answering Why and Why Not Questions about Java Program Output

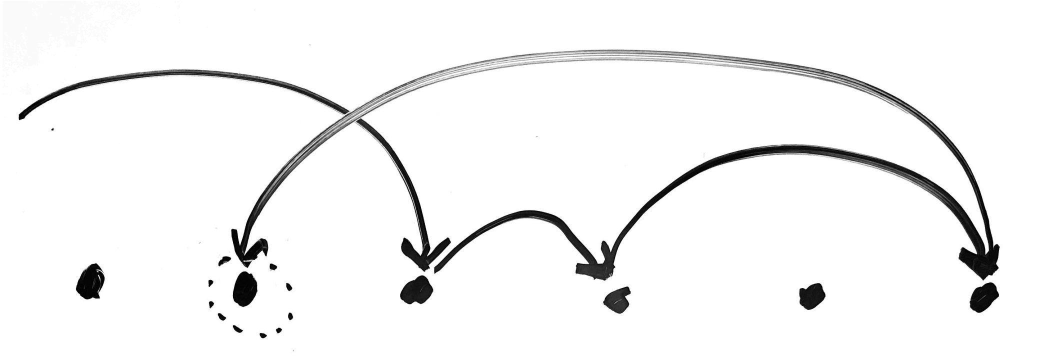
ANDREW J. KO
University of Washington, Seattle
and
BRAD A. MYERS
Carnegie Mellon University, Pittsburgh

When software developers want to understand the reason for a program's behavior, they must translate their questions about the behavior into a series of questions about code, speculating about the causes in the process. The Whyline is a new kind of debugging tool that avoids such speculation by instead enabling developers to select a question about program output from a set of "why did and why didn't" questions extracted from the program's code and execution. The tool then finds one or more possible explanations for the output in question. These explanations are derived using a static and dynamic slicing, precise call graphs, reachability analyses, and new algorithms for determining potential sources of values. Evaluations of the tool on two debugging tasks showed that developers with the Whyline were three times more successful and twice as fast at debugging, compared to developers with traditional breakpoint debuggers. The tool has the potential to simplify debugging and program understanding in many software development contexts.

Ko, A.J. and Myers, B.A. (2010). Extracting and Answering Why and Why Not Questions about Java Program Output. *ACM TOSEM*.

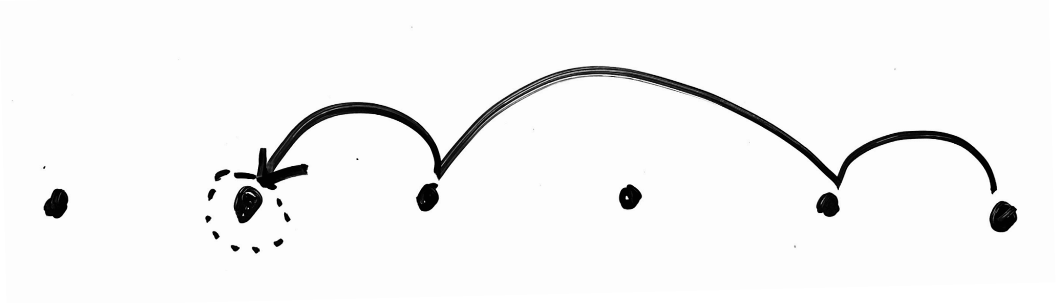
Key theoretical insight

- Debugging is **slow** because *developers iteratively test brittle hypotheses* about what caused a failure by manually collecting runtime data



scientific method

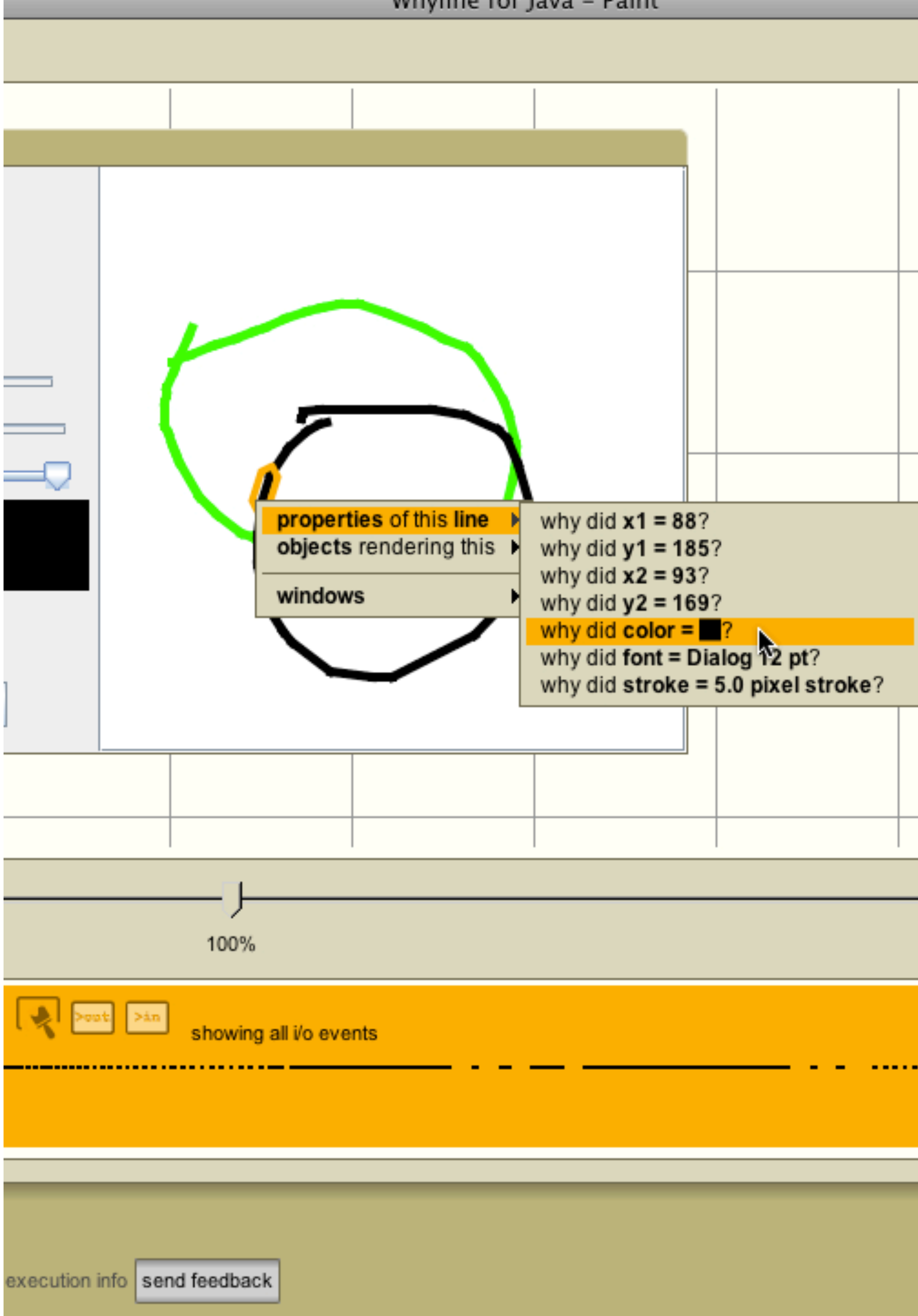
- Debugging would be **faster** if developers *worked backwards from well-understood failure to cause*, relying on dynamic dependencies precisely gathered by a tool



root cause analysis

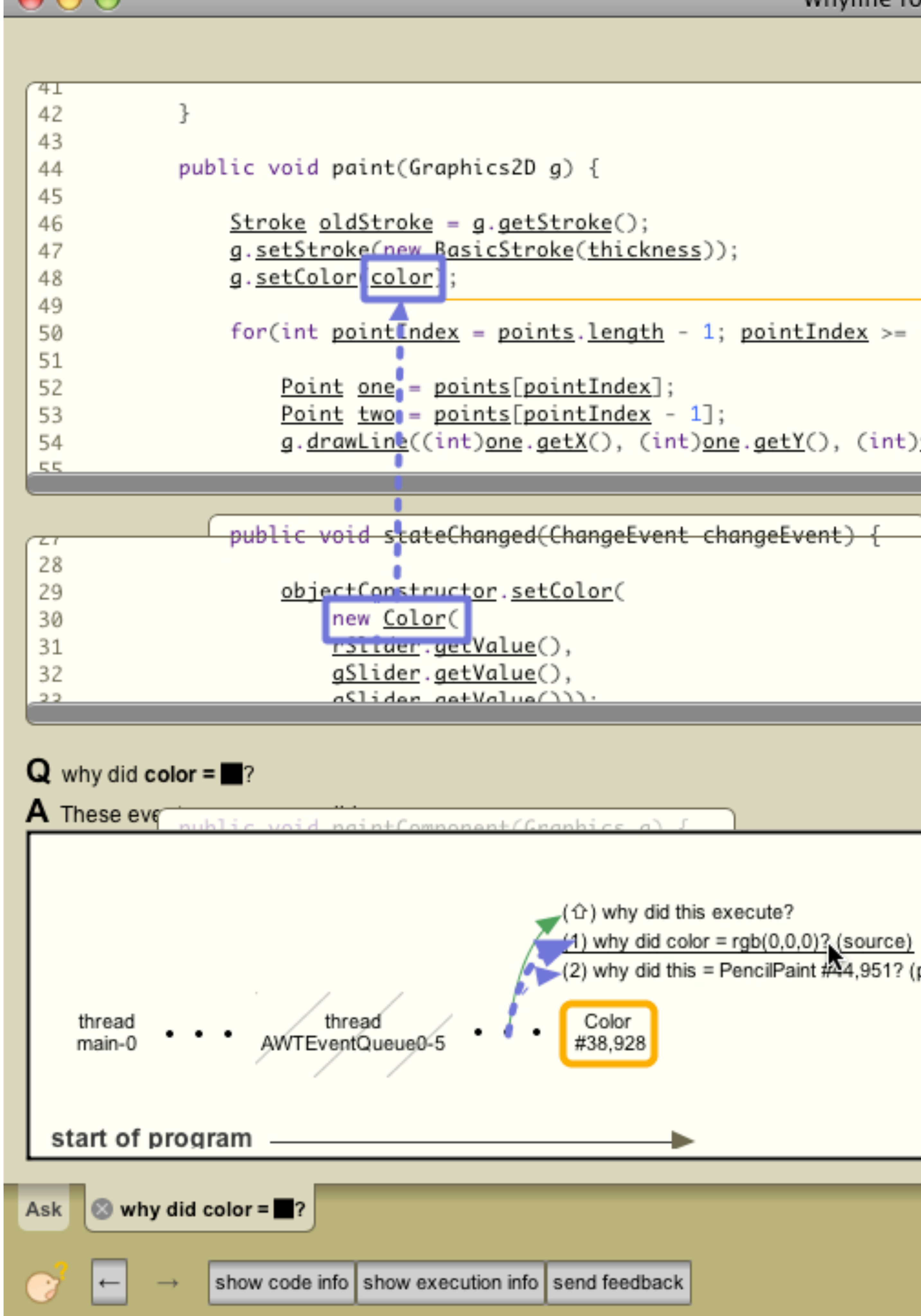
The tool

- Record an execution trace, reproducing an interactive timeline of **program output**
- Allow developers to select questions about **properties of output** they know to be wrong



The tool

- Answer questions with **precise backwards dynamic slicing** on output properties
- Present slice **interactively**, allowing developers to navigate causes to isolate the defect, using their knowledge of architecture and requirements to identify defects



```
        }

        public void paint(Graphics2D g) {

            Stroke oldStroke = g.getStroke();
            g.setStroke(new BasicStroke(thickness));
            g.setColor(color);

            for(int pointIndex = points.length - 1; pointIndex >=

                Point one = points[pointIndex];
                Point two = points[pointIndex - 1];
                g.drawLine((int)one.getX(), (int)one.getY(), (int)
```

```
public void stateChanged(ChangeEvent changeEvent) {

            objectConstructor.setColor(
                new Color(
                rSlider.getValue(),
                gSlider.getValue(),
                gSlider.getValue()));
```

Q why did **color = ■**?

A These eve

```
public void paintComponent(Graphics g) {
```

Key results

- Novices with the Whyline debug **8x** faster than novices without it Ko & Myers 2004
- *Novices* with the Whyline **2x** faster than *experts* without it Ko & Myers 2008
- Experts with the Whyline were **3x** more successful and **2x** faster than experts without it Ko & Myers 2009

Academic impact

- Across 4 papers and many citations:
  - Influenced the design of dozens of other interactive developer tools in SE and HCI
  - Inspired dozens of empirical studies about other hard questions to answer about software behavior in SE
  - Replicated and extended on dozens of other platforms and languages in SE, HCI, CSEd, Databases
  - Helped trigger a resurgence of research on trace-based debugging tools in SE, HCI, PL

Industry impact

- Caused **Adobe** to investigate debugging tools for Flash and other design tools
- Influenced **Microsoft**’s efforts at building .NET execution tracing infrastructure, Debugger Canvas, ChakraCore
- Influencing **Apple**’s Safari developer tools
- Influencing **code.org**'s K-12 tools for learning to code

Big ideas

about scientific practice

❶ Reading accelerates innovation

- *"The way to get good ideas is to get lots of ideas and throw the bad ones away" – Linus Pauling, Nobel laureate, Chemistry*
- One way I took this was to never forget that there are **hundreds of thousands** of papers full of powerful ideas, and we should use them
- I spent **3 months** reading 900+ papers about debugging, diagnostics, human error, root cause analysis, well beyond the boundaries of CS

1 Reading accelerates innovation

- The work that most influenced me was a paper that Mark Weiser cited in his *Program Slicing* paper:
  - Gould, J. D., & Drongowski, P. (1974). *An exploratory study of computer program debugging*. Human Factors, 16(3).
- It showed that
  - Debugging required analyzing data flow
  - Developers satisficed their data flow analysis
  - Developers analyzed many more irrelevant than relevant statements

2 Observation develops insight

- *"A few observations and much reasoning lead to error; many observations and a little reasoning lead to truth" – Alexis Carrel, Nobel laureate, Physiology*
- As an HCI researcher, I took this to mean that if I didn't deeply understand the **experience** of debugging, I could not simplify it, no matter how much I reasoned about it.

2 Observation develops insight

- I spent another **3 months** after reading *observing* people debug: hundreds of novices, experts, and myself.

- Led to a rich *intuition* about debugging that helped me predict the utility of design choices I made in the Whyline

- I still use this intuition today to judge the utility of my research ideas and the ideas published in this community

3 Explain *why*, not just how

- *“He who loves practice without theory is like the sailor who boards ship without a rudder and compass and never knows where he may cast” – Leonardo da Vinci*
- I took this to mean that the true value of inventions is not in explaining *how* they work, but *why* they work.
- These explanations are the generalizable knowledge that stands the test of time, that transfer from tool to tool

❸ Explain *why*, not just how

- The key thing that made the Whyline work was that I synthesized my intuition about debugging into an theory of how people debug and how tools mediate their strategies.
- It was this theory, and not the tool itself, that was the core of the Whyline's innovation.
- The tool was merely an **embodiment** of that theory, helping me test and refine the theory.

Big ideas

about automation

❹ Automation is insufficient

- *"…in practice slicing is fairly fast, and can often eliminate large numbers of unnecessary statements from slices of programs"* – Mark Weiser, "Program slicing." *ICSE*.
- He did not claim that it was *useful*.
- And yet, of 4,500 papers that have investigated slicing, only 3 evaluated developers' *use* of slicing tools, all finding that slices are *too large*, *hard to navigate*, and *incomprehensible at scale*.

4 Automation is insufficient

- Our field's key mistake was assuming that
  1. *Useful slices are trivial for developers to express*
  2. *The size of a slice determines its comprehensibility*
- The Whyline showed neither are true. Making slicing useful required:
  - A new paradigm for **expressing** a slice (output interrogation)
  - A new paradigm for **navigating** a slice (one dependency at a time)
  - **Re-architecting** of slicing algorithms themselves to align with these new paradigms

4 Automation is insufficient

- Since the Whyline, others have shown that automation is also insufficient for other technologies to be useful:
  - Refactoring (e.g., Murphy-Hill)
  - Static analysis (e.g., Pugh; Ernst)
  - Machine learning (e.g., Burnett; Fogarty)
- Probably also true for formal verification, program synthesis, testing tools, bug patching, etc.

5 Augmentation > automation

- We like to believe that with enough data and the right algorithms, our tools can outperform humans
- The Whyline showed that this overlooks the power of developers' knowledge and intuition
  - In the evaluations of the Whyline, participants interacted with slices with 50,000+ LOC
  - By leveraging their knowledge, expertise, and intuition, developers only ever looked at a few dozen LOC, and still found the defects

❺ Augmentation > automation

- Two consequences of ignoring developer knowledge:
    1. Our innovations often aren't useful at all, because they don't account for what developers know
    2. We miss opportunities to *combine* human and machine insights to achieve even greater power
- We must invent for the **entire system** of tools+developers+teams+organizations

Wisdom old and new

1. Accelerate progress by reading
2. Develop a personal intuition for SE practice
3. Explain *why* your tools work

---

4. Automation is insufficient
5. Augmentation > automation

Thank you.

1. Accelerate progress by reading
2. Develop a personal intuition for SE practice
3. Explain *why* your tools work
4. Automation is insufficient
5. Augmentation > automation