How should static analysis tools explain anomalies to developers?

Titus Barik <tbarik@ncsu.edu>

Committee Members:
Dr. Emerson Murphy-Hill (Chair), Dr. Christopher Parnin, Dr. James Lester, Dr. Jing Feng (Psychology, GSR), Dr. Shriram Krishnamurthi (Computer Science, Brown University)
Agenda

Problem (10 min)

My Thesis (5 min)

Theoretical Framework (5 min)

Five evaluations of framework (20 min)
Agenda

Problem (10 min)

My Thesis (5 min)

Theoretical Framework (5 min)

Five evaluations of framework (20 min)
Dominant Visualization Paradigm
Error messages are costly to comprehend and resolve

28.5% of Java builds **fail**. Median resolution time is **12 minutes**, with high variance.

**Power-law distribution.** Top 5 errors cover 80% of errors in the dataset.

**No simple, automatic fixes.** More time consuming errors “more difficult to puzzle out” and require “thinking about design issues.”


Hypothesis: Self-explanation

Error messages fail to support developers through *self-explanation*, a process by which developers *explain* to themselves and to others in order to understand a situation.

Hypothesis: Self-explanation

The process of self-explanation occurs in a variety of problem-solving tasks, for example: geometry (Aleven 2002), chemistry (Crippen 2007), physics (Chi 1989), spreadsheets (Reimann 2008), reading historical passages (Wolfe 2005), and programming (Bielaczyc 1995).


Hypothesis: Self-explanation

Self-explanations can be *elicited* to improve comprehension (Chi 1994, Ainsworth 2003), and self-explanation significantly outperforms *elaborative interrogation* and repetition (O'Reilly 1998).


Comparison to Whyline

Comparison to HelpMeOut

Agenda

Problem (10 min)

My Thesis (5 min)

Theoretical Framework (5 min)

Five evaluations of framework (20 min)
My Thesis

The comprehensibility and utility of error messages for static analysis anomalies can be significantly improved by reframing error messages as material explanations that approximate how developers explain anomalies to other developers and to themselves.
My Thesis

The *comprehensibility and utility* of error messages for static analysis anomalies can be significantly *improved* by reframing error messages as *material explanations* that *approximate* how developers explain anomalies to other developers and to themselves.
My Thesis

The comprehensibility and utility of error messages for static analysis anomalies can be significantly improved by reframing error messages as material explanations that approximate how developers explain anomalies to other developers and to themselves.
My Thesis

The comprehensibility and utility of error messages for static analysis anomalies can be significantly improved by reframing error messages as material explanations that approximate how developers explain anomalies to other developers and to themselves.
My Thesis

The comprehensibility and utility of error messages for static analysis anomalies can be significantly improved by reframing error messages as *material explanations* that approximate how developers explain anomalies to other developers and to themselves.

Types of Explanations

Alex Kass and David Leake
Technical Report #523
Yale Artificial Intelligence Project
My Thesis

The comprehensibility and utility of error messages for static analysis anomalies can be significantly improved by reframing error messages as material explanations that *approximate* how developers explain anomalies to other developers and to themselves.

The error I get is

The method add(Class<T>, T) in the type EventListenerList is not applicable for the argument

The argument in addListener is of a type that extends EventListener, so listener.getClass() returns Class<? extendsEventListener>, which is exactly what the EventListenerList.add method expects

Can someone explain this? I have a feeling that it has something to do with getClass() not being resolved at compile time but it still doesn’t make sense to me
My Thesis

The comprehensibility and utility of error messages for static analysis anomalies can be significantly improved by reframing error messages as material explanations that approximate how developers explain anomalies to other developers and to themselves.
Expected Contributions

1. **New Idea:** A theoretical framework that formalizes explanation theory *in the context of static analysis anomalies.*

2. **Knowledge:** A set of experiments that obtain evidence and provide *scientific underpinnings* and *guidelines* for future, explanation-based tools.

3. **Feasibility:** A proof-of-concept tool that applies these learnings into a usable artifact within the modern programming environment, the IDE, to better support developers.

Agenda

Problem (10 min)

My Thesis (5 min)

Theoretical Framework (5 min)

Five evaluations of framework (20 min)
Static Analysis Explanation Framework

1a IDE
1b Static analysis tool
1c Anomaly
1d Exposed reasoning

2a Message
(Explanation)
2b Human Oracle
Ideal explanation

3a Developer
Self-explanation

3b Successful self-explanation
A B C ... X

3c Failed self-explanation
A B' C' ... X

Agenda

Problem (10 min)

My Thesis (5 min)

Theoretical Framework (5 min)

Five evaluations of framework (20 min)
Diagrams (1/5)
How do developers visualize compiler error messages?

(Completed, Spring 2014, Published VISSOFT)

T. Barik, K. Lubick, S. Christie, and E. Murphy-Hill, “How Developers Visualize
Compiler Messages: A Foundational Approach to Notification Construction,” in
Diagrams: Rationale

Understand why existing visualizations do not align with the way in which developers self-explanation.

```java
// class Brick
void m(int i, double d) {} 
void m(double d, int m) {} 
{
    m(1, 2);
}
```
Diagrams: Methodology

A between-subjects, pencil-and-paper mockup study comparing baseline visualizations against explanatory visualizations \((n = 28)\), generated from a bootstrap pilot study. Each participant received six tasks. We asked participants to explain each compiler anomaly while making annotations on the source code during their explanation.
Diagrams: Results

RQ1: Do explanatory visualizations result in more correct self-explanations by developers? Yes, participants gave significantly better explanations in the treatment group \((n_1 = n_2 = 84, Z = 2.23, p = 0.026)\). Explanatory visualizations improve comprehension.

RQ2: Do developers adopt conventions from our visual annotations in their own self-explanations? Yes, the treatment group used significantly more visual annotation types in their explanations than in the control group \((n_1 = n_2 = 84, Z = 2.15, p = 0.032)\). Moreover, we were unable to identify any significant differences in the distribution \((n = 389, \chi^2 = 1.53, p = 0.220)\), suggesting that the annotations are useful. Explanatory visualizations are used by developers in their own explanations.
Diagrams: Results

RQ3: What aspects differentiate explanatory visualizations from baseline visualizations? Explanatory visualizations reveal more of the hidden dependencies, that is, the reasoning process of the compiler than the baseline visualizations ($n_1 = n_2 = 14, Z = -2.64, p = 0.008$). *Explanatory visualizations explicate relationships.*

**TABLE V: Cognitive Dimensions Questionnaire Responses**

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Control</th>
<th></th>
<th>Treatment</th>
<th></th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Median</td>
<td>Dist</td>
<td>Median</td>
<td>Dist</td>
<td></td>
</tr>
<tr>
<td>Hidden Dependencies*</td>
<td>3</td>
<td><img src="image" alt="dist" /></td>
<td>4</td>
<td><img src="image" alt="dist" /></td>
<td>.008</td>
</tr>
<tr>
<td>Consistency</td>
<td>4</td>
<td><img src="image" alt="dist" /></td>
<td>4</td>
<td><img src="image" alt="dist" /></td>
<td>.979</td>
</tr>
<tr>
<td>Hard Mental Operations</td>
<td>3</td>
<td><img src="image" alt="dist" /></td>
<td>2.5</td>
<td><img src="image" alt="dist" /></td>
<td>.821</td>
</tr>
<tr>
<td>Role Expressiveness</td>
<td>4</td>
<td><img src="image" alt="dist" /></td>
<td>4</td>
<td><img src="image" alt="dist" /></td>
<td>.130</td>
</tr>
</tbody>
</table>
Gazerbeams (2/5)
What can eye gaze tell us about failures in self-explanation during compile error comprehension?

(In-Progress, Summer 2016)
Gazerbeams: Rationale

Provides explanation for why the compiler errors in Seo 2014 are perplexing for developers.

Offers stronger ecological validity than paper-and-pencil Diagrams by using tasks within the Eclipse IDE.

Mitigates think-aloud threat, but introduces analysis complexity.

Gazerbeams: Methodology

We collected data from 60 participants from undergraduate and graduate Software Engineering courses at our University.

Through ten randomly assigned tasks, we gave participants up to 5 minutes to identify and make program modifications to remove an injected compiler anomaly within the Eclipse IDE.

We recorded their eye gaze movements during the tasks.
Stack Overflow (3/5)

How do developers explain static analysis anomalies to other developers through computed-mediated communications?

(In-Progress, Spring 2016)
Stack Overflow: Rationale

**Rationale:** Developers go to Stack Overflow after failing to comprehend an error message, but do comprehend the message after reading a response to a post. Identifying the gap between human-authored explanations and computer-authored explanations allows us to address this gap. Allows for generalization.


Stack Overflow: Methodology

Use the StackExchange Data Explorer API to retrieve posts related to compiler error messages. The tag compiler-errors returns ~10,000 posts.

For analysis, perform a stratified sample to obtain compilers errors from different programming languages.

<table>
<thead>
<tr>
<th>Results</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>cc</td>
</tr>
<tr>
<td>379</td>
<td>119...</td>
</tr>
<tr>
<td>4087</td>
<td>9837</td>
</tr>
<tr>
<td>9993</td>
<td>4699</td>
</tr>
<tr>
<td>3981</td>
<td>4364</td>
</tr>
<tr>
<td>1716</td>
<td>3355</td>
</tr>
</tbody>
</table>
Stack Overflow: Methodology

Use the votes table for the answer to cluster good responses versus bad responses.

The method is ambiguous

The problem is that your compiler don't know which method to use if you call `o.sum(5, 5);`

1. He could use `void sum(int i, long j) { }` where he takes the first 5 as int and the second 5 as long

2. He could use `void sum(long i, int j) { }` where he takes the first 5 as long and the second 5 as int

since both methods would be valid to use in this example and your compiler always needs exactly one valid option, you get the error msg that your method is ambiguous.

if you call `o.sum(5, (long)5);` it matches only the method `void sum(int i, long j) { }`

Rust (4/5)

How do compiler authors design static analysis message in their tools?

(Proposed, Summer 2016)
Traver 2010 conducted a literature review of error message difficulties and found a lack of formal study on how compiler toolsmiths design compiler error messages.

Understanding the “life cycle” of static analysis anomalies from the perspective of toolsmiths will enable researchers to identify and addresses barriers for toolsmiths.

Rust: Methodology

A **case study**: semi-structured interviews with static analysis authors as identified by version history commits and communications, and qualitative analysis of mailing lists, version control issues, and IRC logs to form a grounded theory. Investigator will interact with the community through these channels.

<table>
<thead>
<tr>
<th>Contributors</th>
<th>Commits</th>
<th>Code frequency</th>
<th>Punch card</th>
<th>Network</th>
<th>Members</th>
</tr>
</thead>
</table>

Woah, this network is huge! We’re showing only some of this network’s repositories.

- rust-lang / rust
- Oc0w3 / rust
Radiance (5/5)

Can instrumenting our findings as a practical tool improve developer static analysis error comprehension?

(In-Progress, Spring 2016)
Radiance: Rationale (and Principles)
Radiance: Design Principles

**Explicate relationships.** Make explicit the relations of *what* and *why* (Legare 2014).

**Prefer diagrammatic over sentential representations.** Problem solving with proceeds more smoothly with *diagrammatic* representations (Larkin 1987).

**Support gap-filling.** Explainer incorrectly believes that he or she has the complete domain knowledge needed to understand a problem (VanLehn 1993).


Radiance: Methodology

A theoretical replication of the Gazerbeams study, using the explanatory visualizations within the Eclipse IDE.

As an addition to the original experimental design, I will conduct an follow-up participatory design exercise to collect opinions and judgments about the Radiance tool against experiences with their own tools.
Summary: Framework Coverage

1a. IDE

1b. Static analysis tool

1c. Anomaly

1d. Exposed reasoning

2a. Message (Explanation)

2b. Human Oracle

Ideal explanation:

A B C ... X

3a. Developer

Self-explanation

3b. Successful self-explanation

A B C ... X

3c. Failed self-explanation

A B' C' ... X
Project Plan: HCI + Software Engineering Researcher

Summer 2016
  Gazerbeams (70%): ICSE 2017 - Aug 26, 2016
  Stack Overflow (10%): CSCW 2017 - May 22, 2016

Fall 2016
  Rust (0%): CHI 2017 - Sep 25, 2016

Spring 2017
  Radiance (0%): FSE 2017 - Mar 11, 2017

Summer 2017
  Dissertation Writing and Resubmit Cycle

Fall 2017
  Dissertation Defense and Resubmit Cycle
Conclusion

The comprehensibility and utility of error messages for static analysis anomalies can be significantly improved by reframing error messages as material explanations that approximate how developers explain anomalies to other developers and to themselves.
Gazerbeams: Research Questions

**RQ1** How does the sequence of information that developers use when understanding a static analysis anomaly differ between *successful* and *unsuccessful* developers?

**RQ2** What features of the static error messages do developers actually use, and to what extent?

**RQ3** Does gap-filling explain the difference between successful and unsuccessful developers?

**RQ4** What eye tracking measures explain the difference between successful and unsuccessful developers?

Stack Overflow: Research Questions

RQ1 What questions do developers ask when they want to understand a static analysis error?

RQ2 What plausible self-explanations have they already generated when they frame their question to other developers?

RQ3 What features of a response make for a good explanation?

RQ4 What features of a response make for a poor explanation?
Rust: Research Questions

**RQ1** At what points in the software development lifecycle do static analysis authors work with static analysis anomalies?

**RQ2** How do static analysis authors make decisions about which anomalies to implement?

**RQ3** How do static analysis authors evaluate their error messages?

**RQ4** What types of discussions happen around error messages?

**RQ5** What are the challenges that static analysis authors identify that hinder the generation of good error messages?
Radiance: Research Questions

**RQ1** How do developers assess the tool under traditional HCI measures, such as effectiveness and efficiency, when compared with my prior Eclipse experiment (\cref{sec:gazerbeams})?

**RQ2** Do developers adopt different strategies with Radiance than with Eclipse?

**RQ3** Replicating the eye tracking research questions from the Eclipse study, in what ways do they differ?

**RQ4** What do developers say about why Radiance helps or prevents them from performing their task?
Explicit Relationality

Table 1. Opportunities for Improvement: Explicit Relationality

<table>
<thead>
<tr>
<th>Static Analysis</th>
<th>All Relational Pct (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse JDK (Luna)</td>
<td>598</td>
</tr>
<tr>
<td>Microsoft C# 5.0 (Roslyn)</td>
<td>1107</td>
</tr>
<tr>
<td>Microsoft F# 3.1</td>
<td>1136</td>
</tr>
<tr>
<td>Microsoft TypeScript 1.0</td>
<td>538</td>
</tr>
<tr>
<td>Oracle OpenJDK 7</td>
<td>487</td>
</tr>
</tbody>
</table>
Diagrams: Participant Annotations

```java
1   class Brick {
2     void m(int i, double d) {
3     void m(double d, int m) {
4     void m = int m;
5     
6       m(1, 2);  \Rightarrow \text{either 1.0 or 2.0}
7     }
8   }
```

Compiler Output

Brick.java:6: error: reference to m is ambiguous,
both method m(int,double) in Brick and method m(double,int) in Brick match
m(1, 2);

1 error

Questionnaire

1. Have you ever encountered this error message before?
   [X] Yes [ ] No [ ] Unsure

2. How confident are you about the accuracy of your explanation for this error message?
   [ ] Not at all confident
   [ ] Somewhat confident
   [ ] Moderately confident
   [ ] Highly confident
   [ ] Completely confident

50
# Diagrams: Annotation Legend

## TABLE I: Frequency of Visual Annotations in Pilot

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Frequency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>49</td>
<td>A particular token or set of tokens has been marked. Examples include underlining or circles the token(s).</td>
</tr>
<tr>
<td>Text</td>
<td>45</td>
<td>Natural language text. For example, “assign a value to the variable” or “dead code”.</td>
</tr>
<tr>
<td>Association</td>
<td>33</td>
<td>An association between two or more program elements, which is accomplished by drawing a connecting line between the elements, with or without arrow heads.</td>
</tr>
<tr>
<td>Symbol</td>
<td>20</td>
<td>Symbols include visual annotation such as ? or x, or numbered circles, to name a few.</td>
</tr>
<tr>
<td>Code</td>
<td>14</td>
<td>Explanatory code that is written in order to explain the error message, for example, if (b == false) or m(1.0, 2). This does not have to be correct Java code, but should be interpretable as pseudocode.</td>
</tr>
<tr>
<td>Strikethrough</td>
<td>5</td>
<td>The strikethrough is separated from the point annotation because this annotation is provided by IDEs today, and has pre-established semantics.</td>
</tr>
<tr>
<td>Multicolor</td>
<td>-</td>
<td>The use of more than a single color to explain a concept. For example, green may be used to indicate lines that are okay, and red to indicate lines that are problematic. This option was not available to students in the pilot study.</td>
</tr>
</tbody>
</table>

## TABLE II: Visual Annotation Legend

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>📑</td>
<td>The starting location of the error.</td>
</tr>
<tr>
<td>🔴</td>
<td>Indicates issues related to the error.</td>
</tr>
<tr>
<td>➔</td>
<td>Arrows can be followed. They indicate the next relevant location to check.</td>
</tr>
<tr>
<td>🟢</td>
<td>Enumerations are used to number items of potential interest, especially when the information doesn’t fit within the source code.</td>
</tr>
<tr>
<td>✗</td>
<td>The compiler expected an associated item, but cannot find it.</td>
</tr>
<tr>
<td>🖐</td>
<td>Conflict between items.</td>
</tr>
<tr>
<td>📍</td>
<td>Explanatory code or code generated internally by the compiler. The code is not in the original source.</td>
</tr>
<tr>
<td>🔴</td>
<td>Indicates code coverage. Green lines indicate successfully executed code. Red lines indicate failed or skipped lines.</td>
</tr>
</tbody>
</table>
## TABLE V: Cognitive Dimensions Questionnaire Responses

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Control</th>
<th>Treatment</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Median</td>
<td>Dist</td>
<td>Median</td>
</tr>
<tr>
<td>Hidden Dependencies*</td>
<td>3</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Consistency</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Hard Mental Operations</td>
<td>3</td>
<td></td>
<td>2.5</td>
</tr>
<tr>
<td>Role Expressiveness</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

* Indicates statistical significance.
### TABLE III: Participant Explanation and Recall Tasks

<table>
<thead>
<tr>
<th>Task Order</th>
<th>Task Name</th>
<th>OpenJDK File</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Melon</td>
<td>VarMightNotHaveBeenInitialized.java</td>
<td>variable i might not have been initialized</td>
</tr>
<tr>
<td>T2</td>
<td>Kite</td>
<td>UnreportedExceptionDefaultConstructor.java</td>
<td>unreported exception Exception in default constructor</td>
</tr>
<tr>
<td>T3</td>
<td>Brick</td>
<td>RefAmbiguous.java</td>
<td>reference to m is ambiguous, both method m(int,double) in Brick and method m(double,int) in Brick match</td>
</tr>
</tbody>
</table>
| T4         | Zebra     | InferredDoNotConformToBounds.java | cannot infer type arguments for BlackStripe<>;
|            |           |                                  | reason: inferred type does not conform to declared bound(s)                 |
|            |           |                                  | inferred: String                                                            |
|            |           |                                  | bound(s): Number                                                            |
| T5         | Apple     | RepeatedModifier.java             | repeated modifier                                                            |
| T6         | Trumpet   | UnreachableCatch1.java            | unreachable catch clause thrown types IllegalArgumentException,EOFException have already been caught |
Radiance: Methodology

**Material Explanations**: One challenge is mapping a cognitive process of self-explanation to computationally supporting that process.

![Diagram of types of explanations]

*Figure 1: A hierarchy of types of explanations*

Gazerbeams: GazePoint Interface
Perspectives on Modern IDE Errors

“Compilers should be assistants, not adversaries. A compiler should not just detect bugs, it should then help you understand why there is a bug.” – Evan Czaplicki

“As compilers perform their magic, they build up deep understanding of the code they are processing, but that knowledge is unavailable to anyone but the compiler implementation wizards. The information is promptly forgotten after the translated output is produced. For decades, this world view has served us well, but it is no longer sufficient.” – Microsoft Roslyn Team

“If we're writing our code on a computer, why are we simulating what a computer would do in our head? Why doesn't the computer just do it, and show us?” – Bret Victor, Inventing on Principle.
Are incorrect explanations useful?

Increasing completeness is better than soundness (Kulesza 2013).

Having to explaining inconsistent evidence yields ore exploratory behavior (Legare 2012).

Explaining why a system behaves a certain way increases trust (Lim 2009).

