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)

NC STATE UNIVERSITY
Department of Computer Science

ABB

NSF

Developer
Liberation
Front

Agenda

Problem (*10 min*)

My Thesis (*5 min*)

Theoretical Framework (*5 min*)

Five evaluations of framework (*20 min*)

Agenda

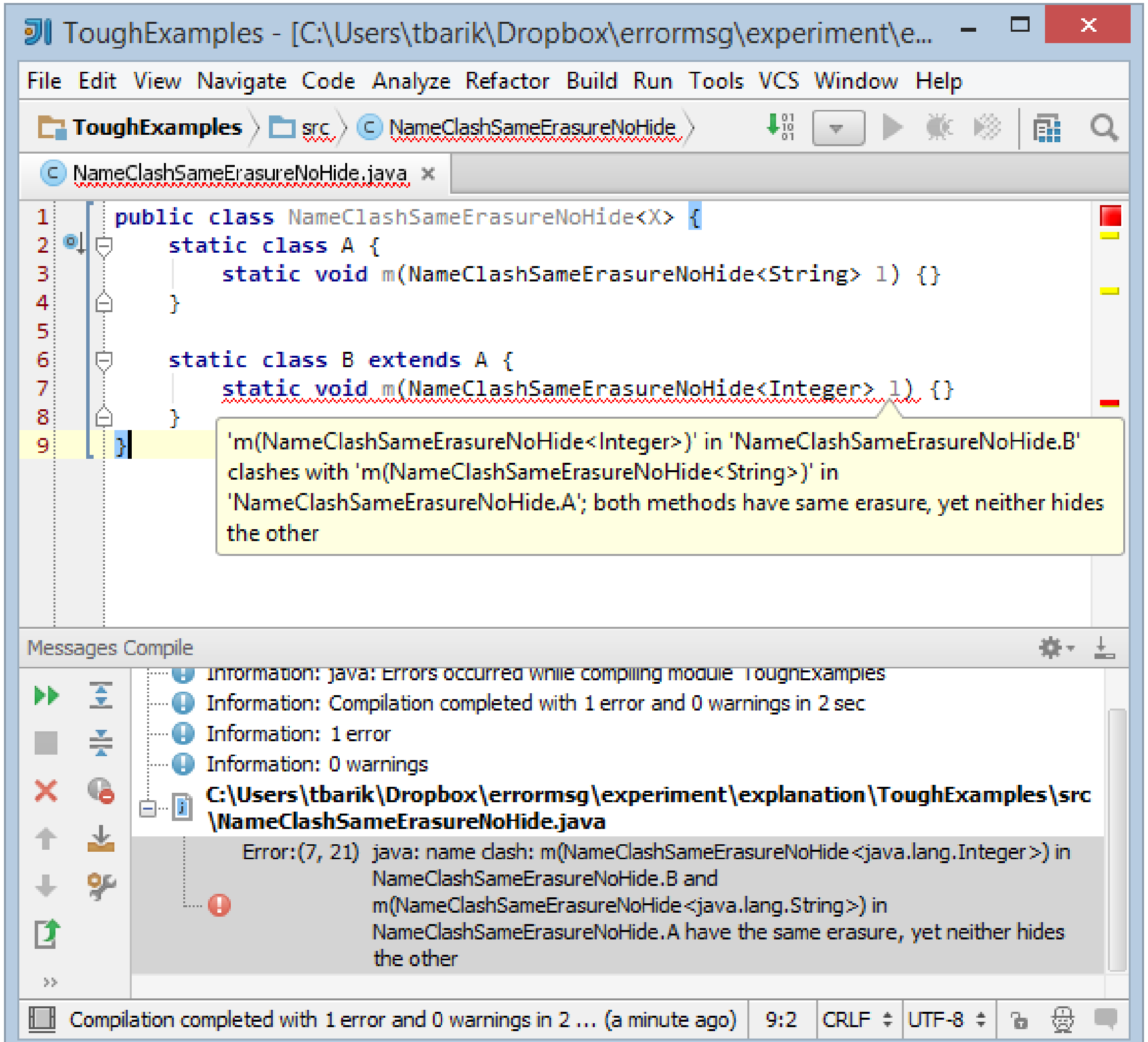
Problem (*10 min*)

My Thesis (*5 min*)

Theoretical Framework (*5 min*)

Five evaluations of framework (*20 min*)

Dominant Visualization Paradigm

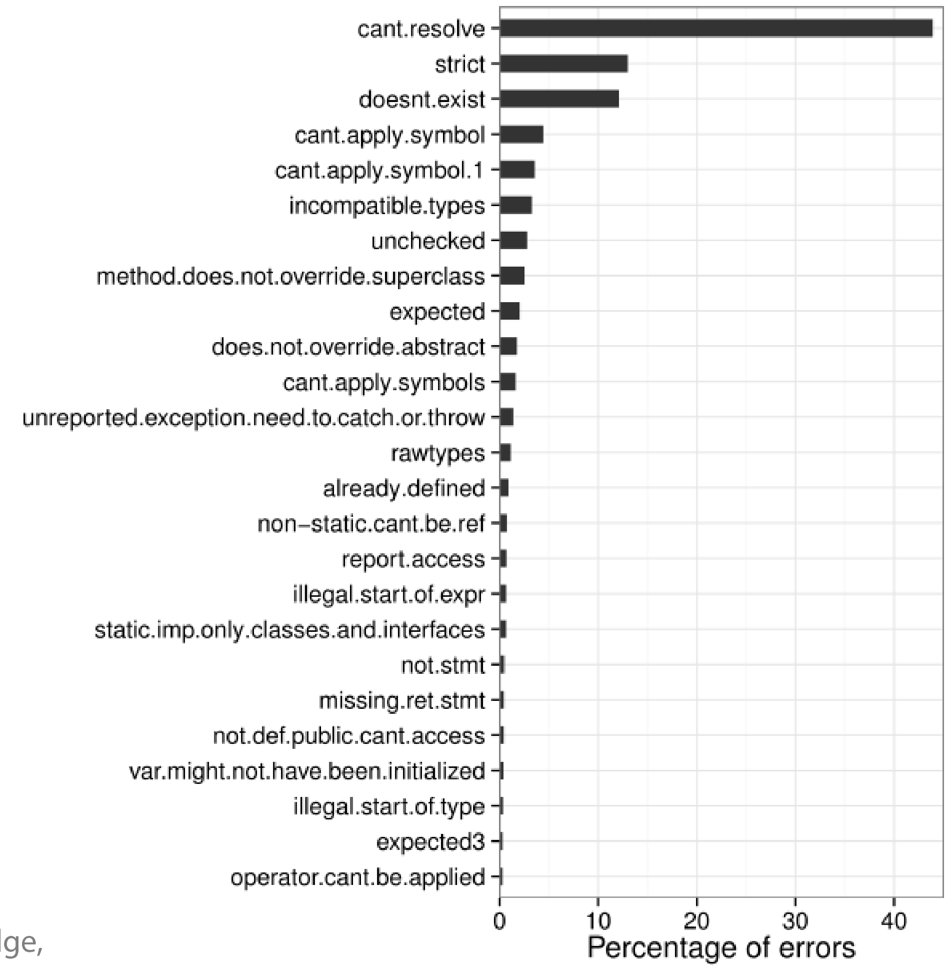


Error messages are costly to comprehend and resolve

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."



H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge, "Programmers' build errors: a case study (at Google)," in ICSE 2014.

D. Pritchard, "Frequency distribution of error messages," in *PLATEAU 2015*.

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.

M. T. H. Chi, M. Bassok, M. W. Lewis, P. Reimann, and R. Glaser, “Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems,” *Cogn. Sci.*, vol. 13, no. 2, pp. 145–182, Apr. 1989.

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).

V. Aleven, "An effective metacognitive strategy: learning by doing and explaining with a computer-based Cognitive Tutor," *Cogn. Sci.*, vol. 26, no. 2, pp. 147–179, Apr. 2002.

K. J. Crippen and B. L. Earl, "The impact of web-based worked examples and self-explanation on performance, problem solving, and self-efficacy," *Comput. Educ.*, vol. 49, no. 3, pp. 809–821, Nov. 2007.

M. T. H. Chi, M. Bassok, M. W. Lewis, P. Reimann, and R. Glaser, "Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems," *Cogn. Sci.*, vol. 13, no. 2, pp. 145–182, Apr. 1989.

P. Reimann and C. Neubert, "The role of self-explanation in learning to use a spreadsheet through examples," *J. Comput. Assist. Learn.*, vol. 16, no. 4, pp. 316–325, Oct. 2008.

M. B. W. Wolfe and S. R. Goldman, "Relations Between Adolescents' Text Processing and Reasoning," *Cogn. Instr.*, vol. 23, no. 4, pp. 467–502, Dec. 2005.

K. Bielaczyc, P. L. Pirolli, and A. L. Brown, "Training in Self-Explanation and Self-Regulation Strategies: Investigating the Effects of Knowledge Acquisition Activities on Problem Solving," *Cogn. Instr.*, vol. 13, no. 2, pp. 221–252, Jan. 1995.

Hypothesis: Self-explanation

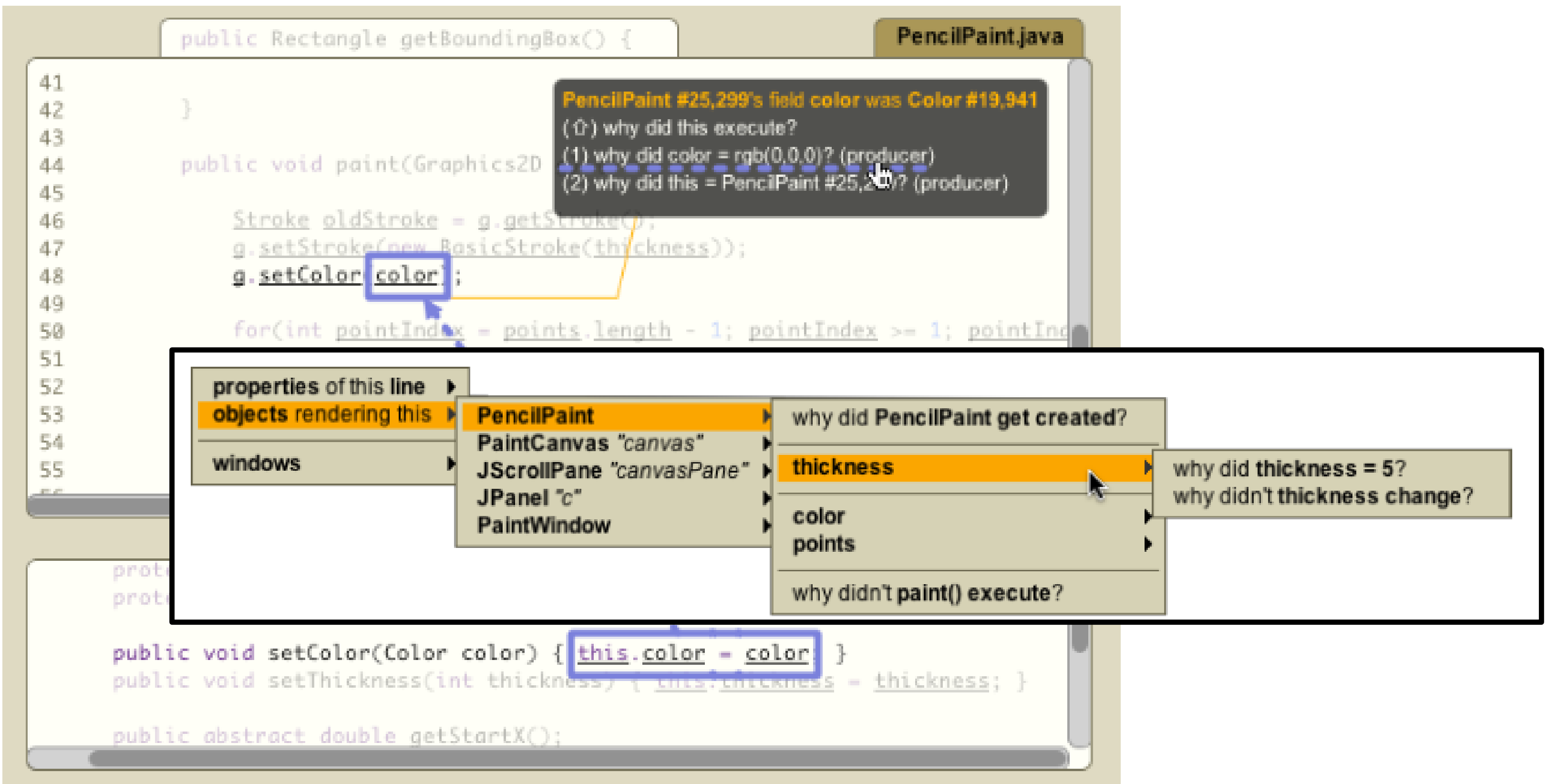
Self-explanations can be *elicited* to improve comprehension (Chi 1994, Ainsworth 2003), and self-explanation significantly outperforms *elaborative interrogation* and repetition (O'Reilly 1998).

M. T. H. Chi, N. De Leeuw, M.-H. Chiu, and C. Lavancher, "Eliciting Self-Explanations Improves Understanding," *Cogn. Sci.*, vol. 18, no. 3, pp. 439–477, Jul. 1994.

S. Ainsworth and A. Th Loizou, "The effects of self-explaining when learning with text or diagrams," Cogn. Sci., vol. 27, no. 4, pp. 669–681, Aug. 2003.

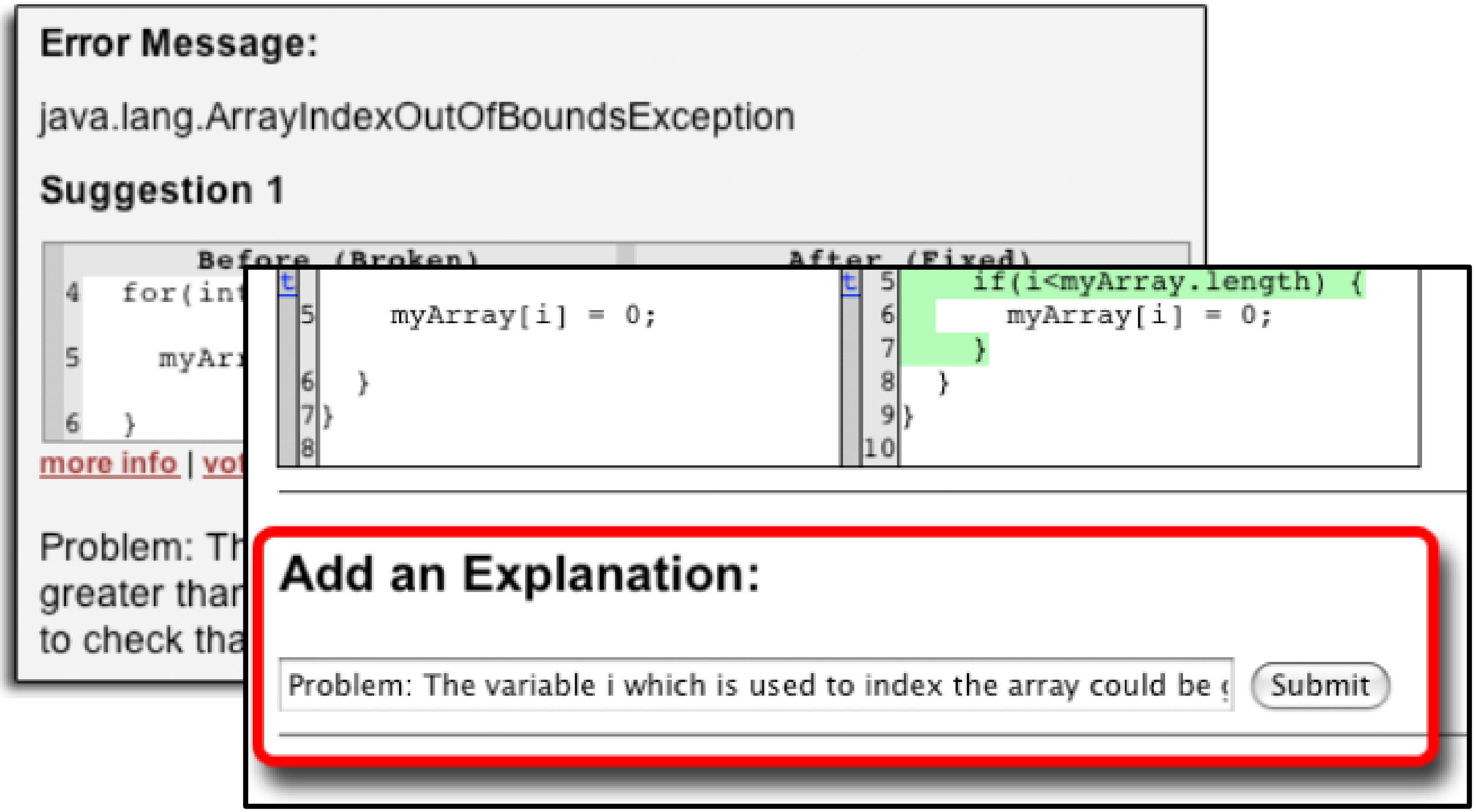
T. O'Reilly, S. Symons, and H. MacLatchy-Gaudet, "A Comparison of Self-Explanation and Elaborative Interrogation," Contemp. Educ. Psychol., vol. 23, no. 4, pp. 434–445, Oct. 1998.

Comparison to *Whyline*



A. J. Ko and B. A. Myers, "Finding causes of program output with the Java Whyline," in *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*, 2009, pp. 1569–1578.

Comparison to *HelpMeOut*



B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, "What would other programmers do," in *Proceedings of the 28th International Conference on Human Factors in Computing Systems - CHI '10*, 2010, pp. 1019–1028.

Agenda

Problem (*10 min*)

My Thesis (*5 min*)

Theoretical Framework
(*5 min*)

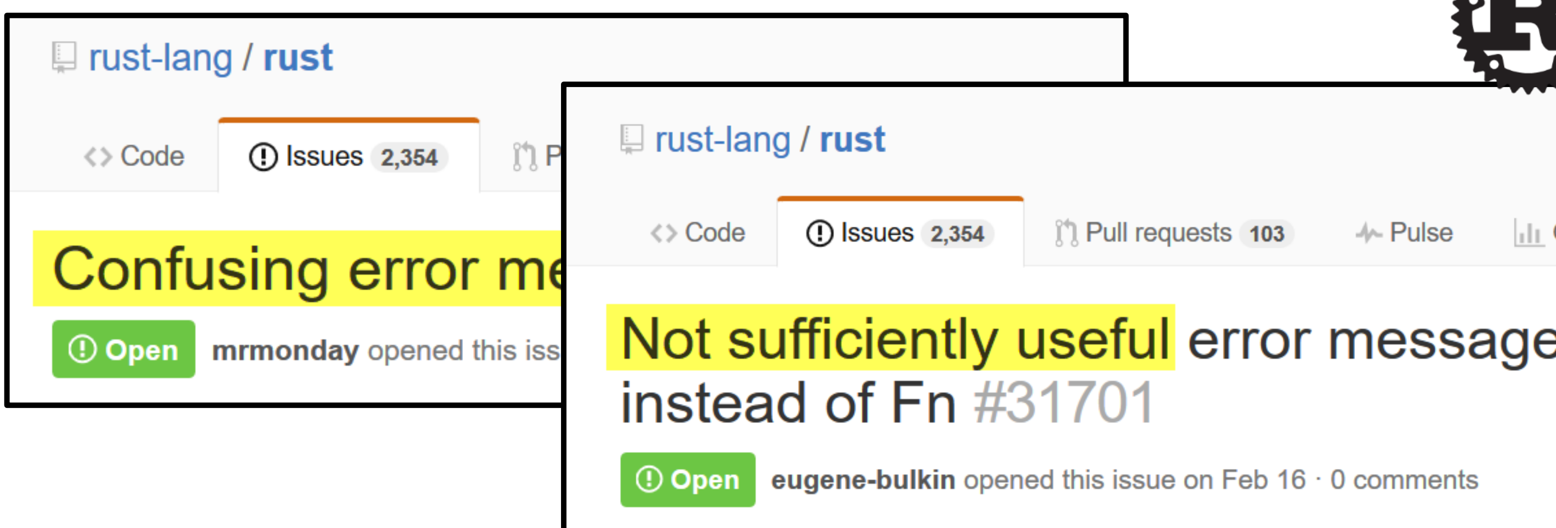
Five evaluations of
framework (*20 min*)

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

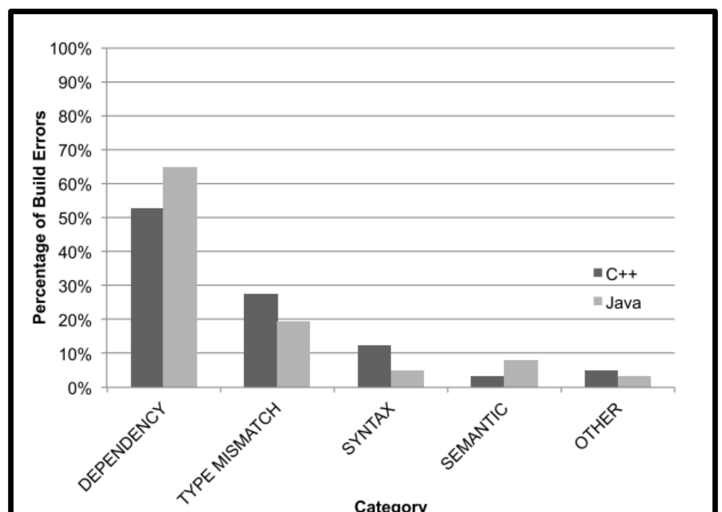
The *comprehensibility* and *utility* of error messages for static analysis anomalies can be significantly *improved* by reframing error messages as *material explanations* that *approximate* how developers explain anomalies to other developers and to themselves.



My Thesis

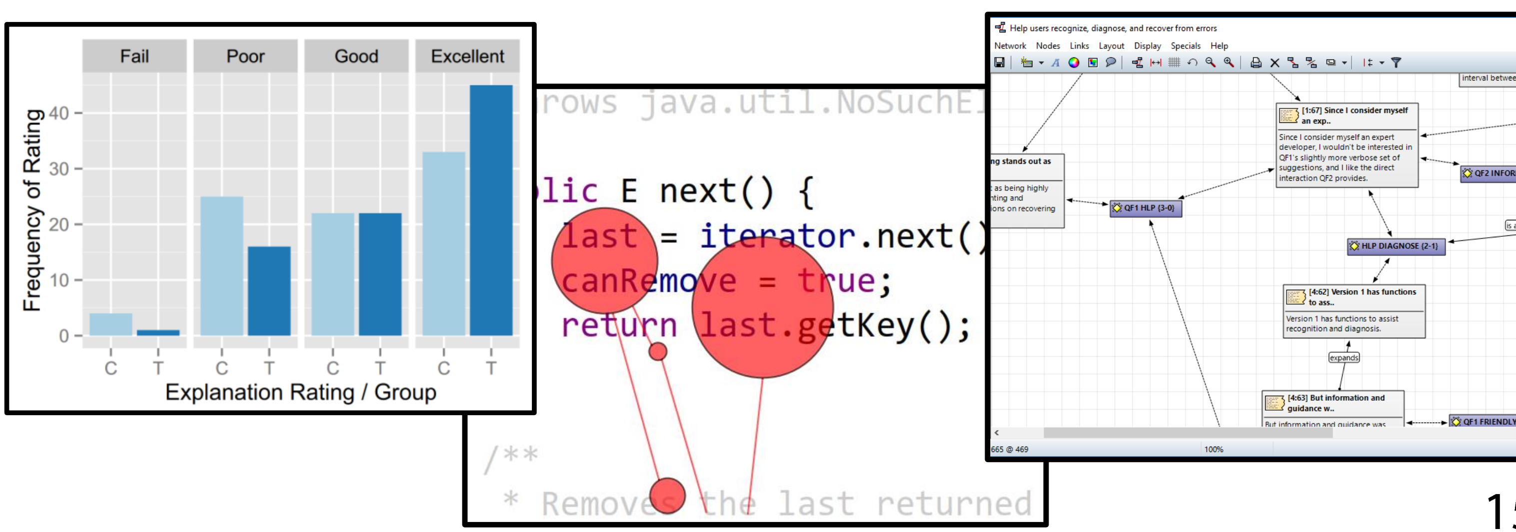
The *comprehensibility* and *utility* of error messages for static analysis anomalies can be significantly *improved* by reframing error messages as *material explanations* that *approximate* how developers explain anomalies to other developers and to themselves.

eclipse



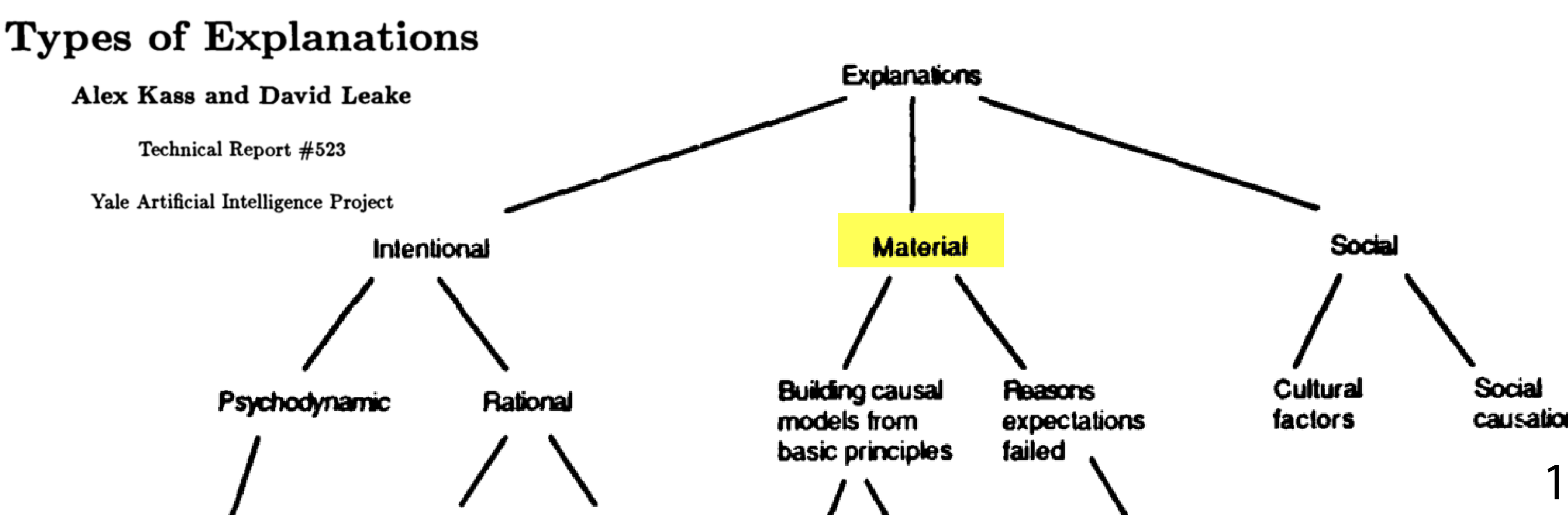
My Thesis

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.

stackoverflow

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<? extends EventListener>` , 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

java generics compiler-errors

share edit flag

asked May 16 '12 at 16:59

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.

T. Barik, J. Witschey, B. Johnson, and E. Murphy-Hill, "Compiler error notifications revisited: An interaction-first approach for helping developers more effectively comprehend and resolve error notifications," in *ICSE Companion 2014*, 2014, pp. 536–539.

Agenda

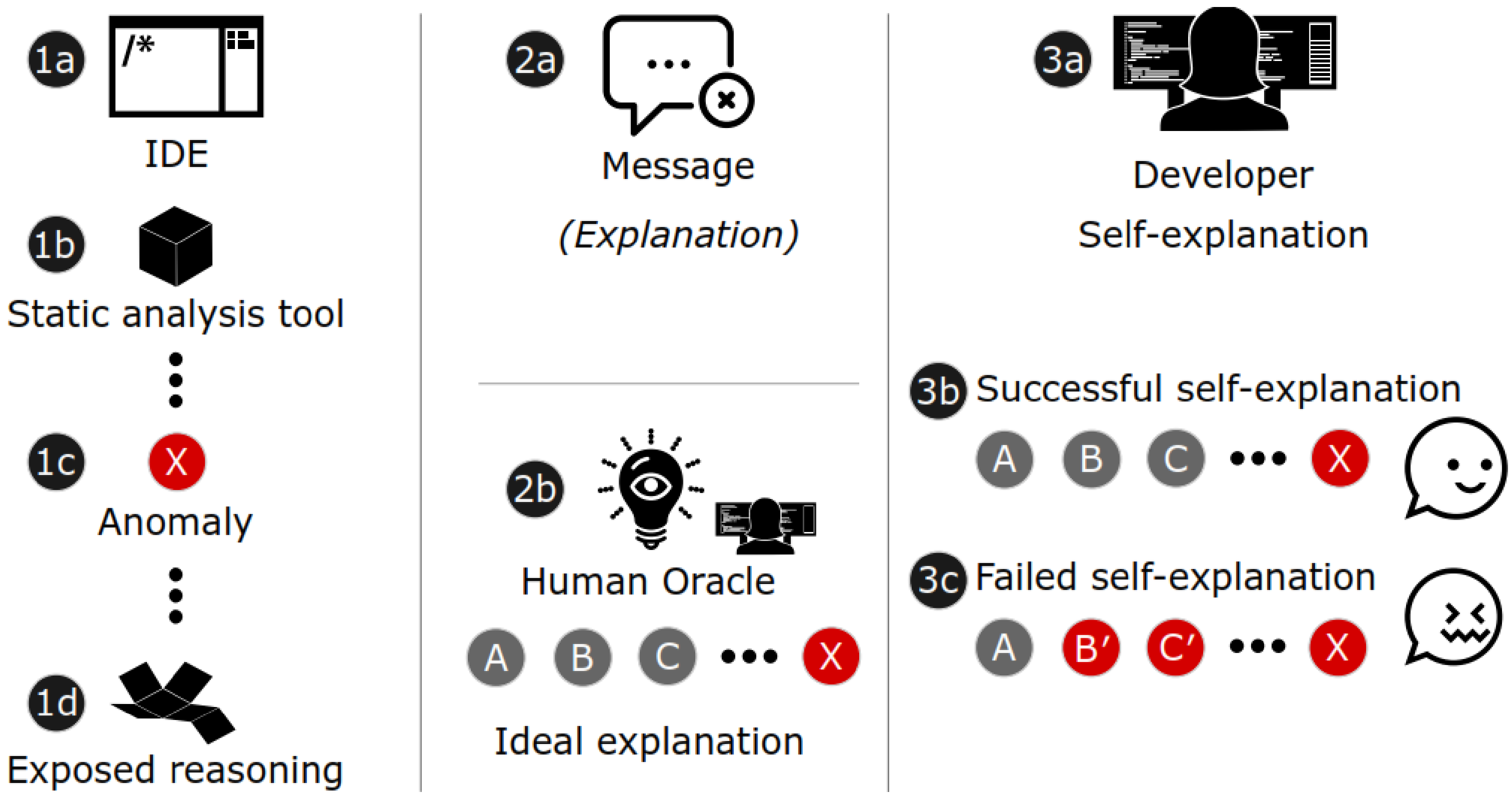
Problem (*10 min*)

My Thesis (*5 min*)

Theoretical Framework (*5 min*)

Five evaluations of framework (*20 min*)

Static Analysis Explanation Framework



C. R. Berger, "The covering law perspective as a theoretical basis for the study of human communication," *Commun. Q.*, May 2009.

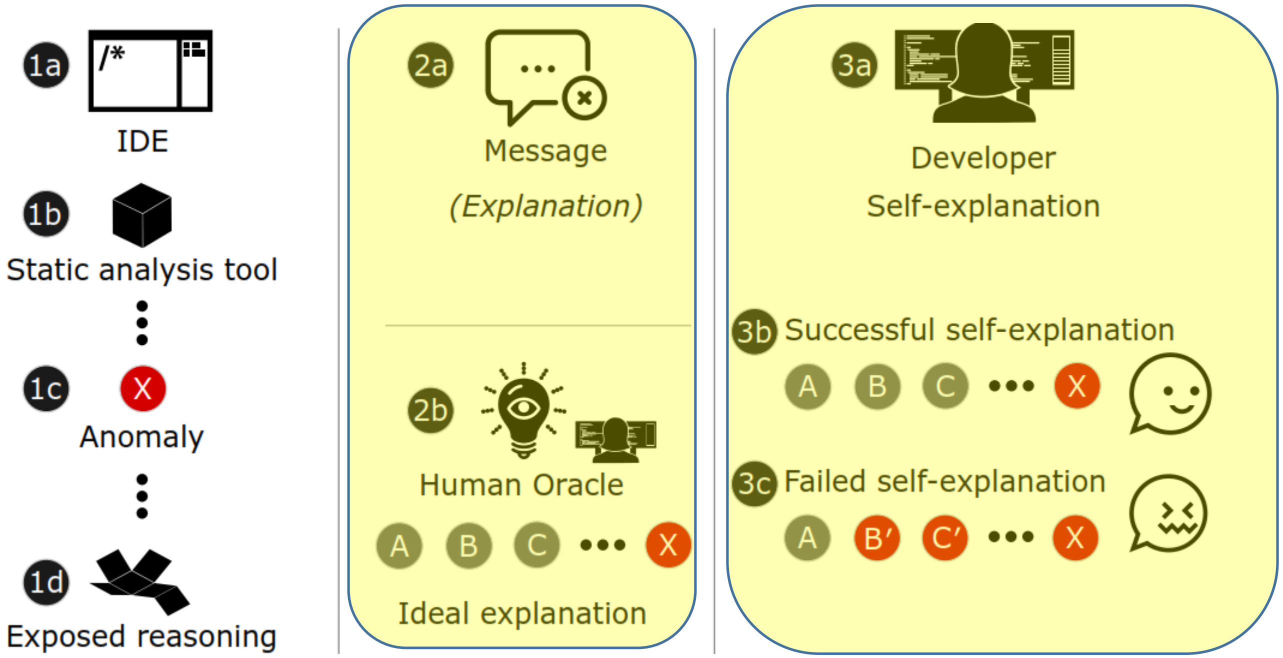
Agenda

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 *2014 Second IEEE Working Conference on Software Visualization*, 2014, pp. 87–96.

Diagrams: Rationale

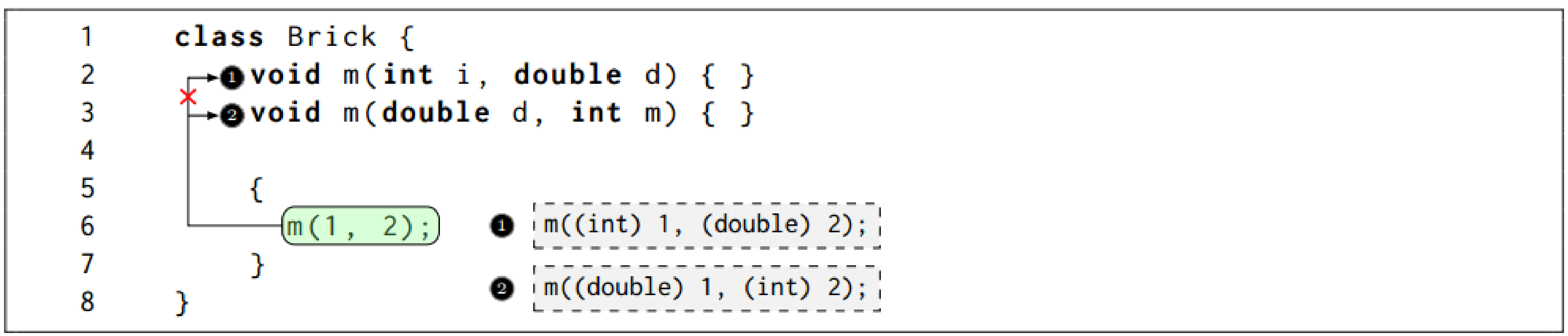
Understand why existing visualizations do not align with the way in which developers self-explanation.

```
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*.

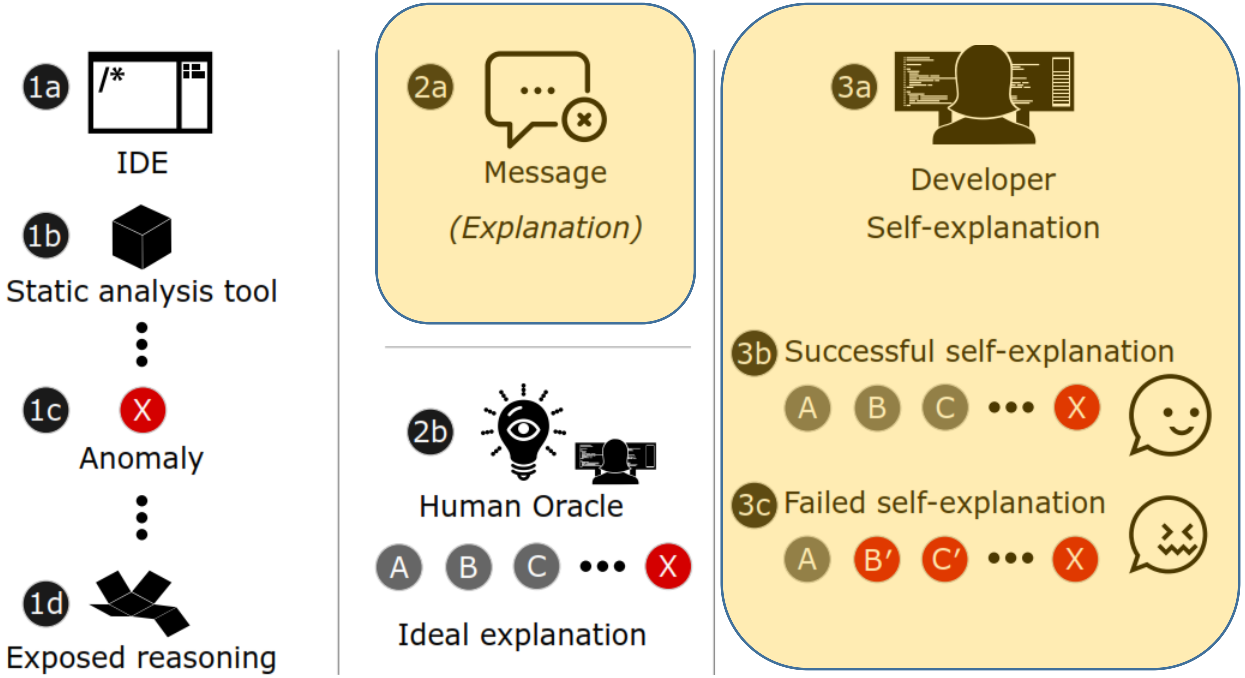
RQ2: Do developers adopt conventions from our visual annotations in their own self-explanations? Yes, the treatment group used significantly more visual *annotation types* in their explanations than in the control group ($n_1 = n_2 = 84$, $Z = 2.15$, $p = 0.032$). Moreover, we were unable to identify 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 depencies, 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

	Control		Treatment		
Dimension	Median	Dist	Median	Dist	p
Hidden Dependencies*	3		4		.008
Consistency	4		4		.979
Hard Mental Operations	3		2.5		.821
Role Expressiveness	4		4		.130



Gazerbeams (2/5)

What can eye gaze tell us about failures in self-explanation during compile error comprehension?

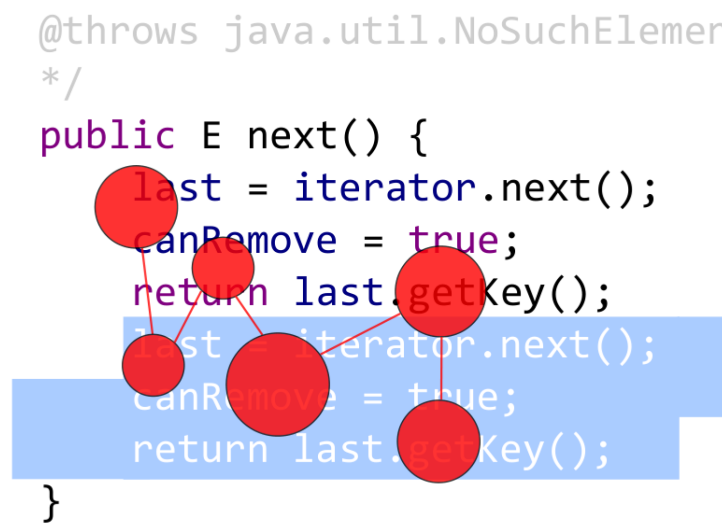
(In-Progress, Summer 2016)

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.



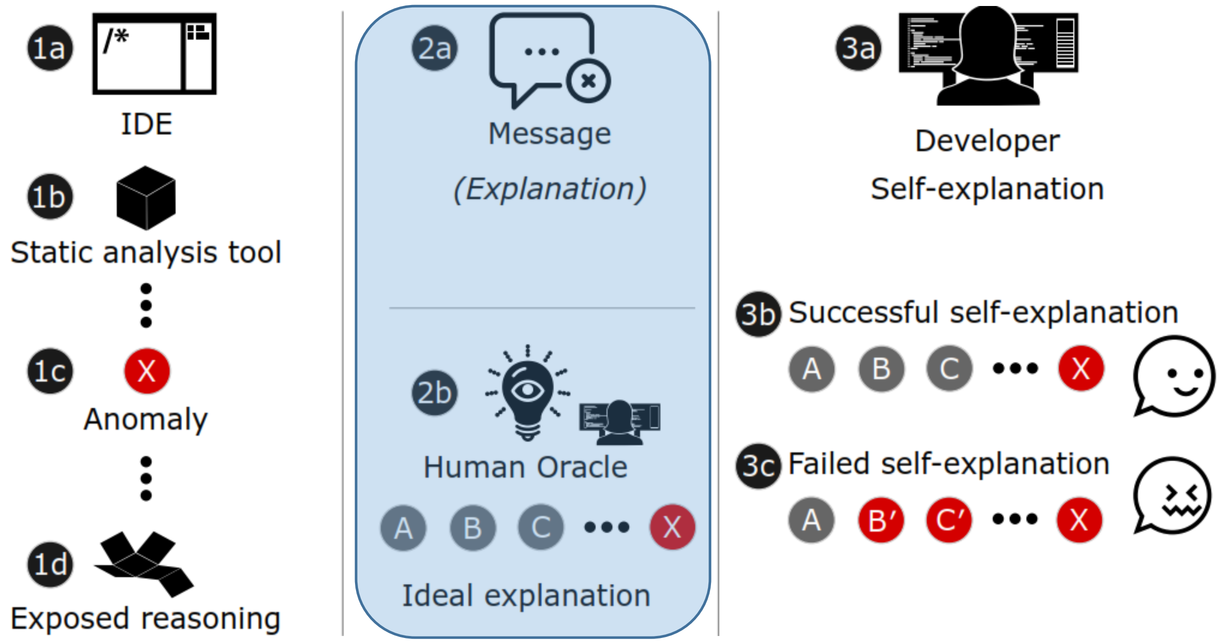
L. Cooke and E. Cuddihy, "Using eye tracking to address limitations in think-aloud protocol," in *IPCC 2005. Proceedings. International Professional Communication Conference, 2005.*, 2005, pp. 653–658.

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 makee 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.

S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 25–34.

A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in Stack Overflow," *Empir. Softw. Eng.*, vol. 19, no. 3, pp. 619–654, Nov. 2012.

M. Squire and C. Funkhouser, "'A Bit of Code': How the Stack Overflow Community Creates Quality Postings," in *2014 47th Hawaii International Conference on System Sciences*, 2014, pp. 1425–1434.

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.

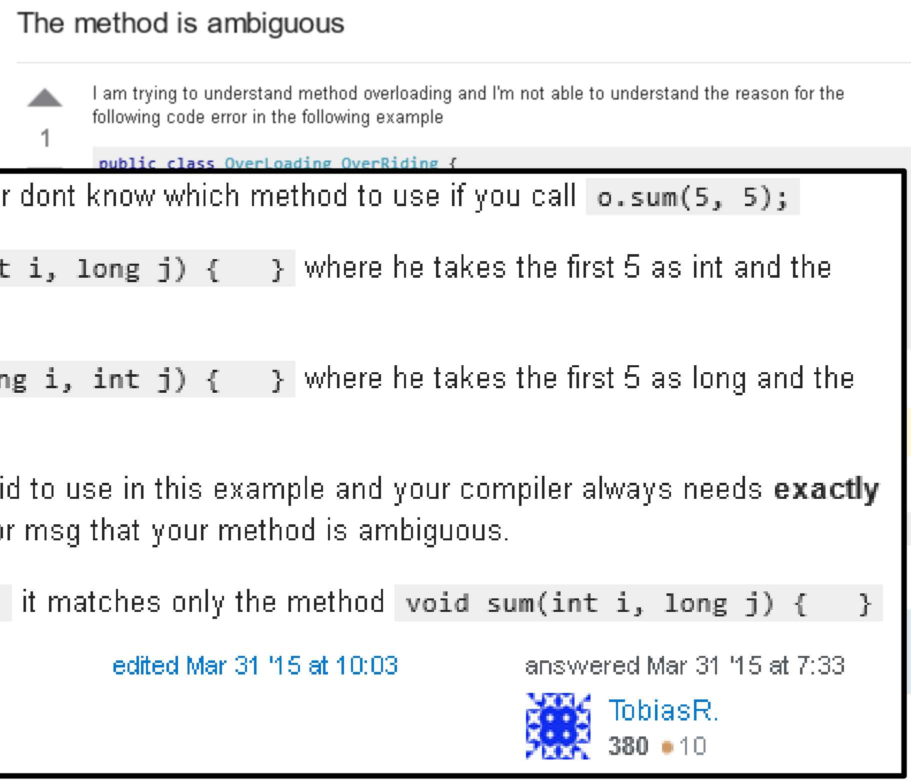
Results | Messages

Id	cc	TagName	ExcerptPostId
379	119…	error-handling	5506485
4087	9837	compiler-errors	5628763
9993	4699	syntax-error	7690280
3981	4364	runtime-error	8365146
1716	3355	warnings	9650452

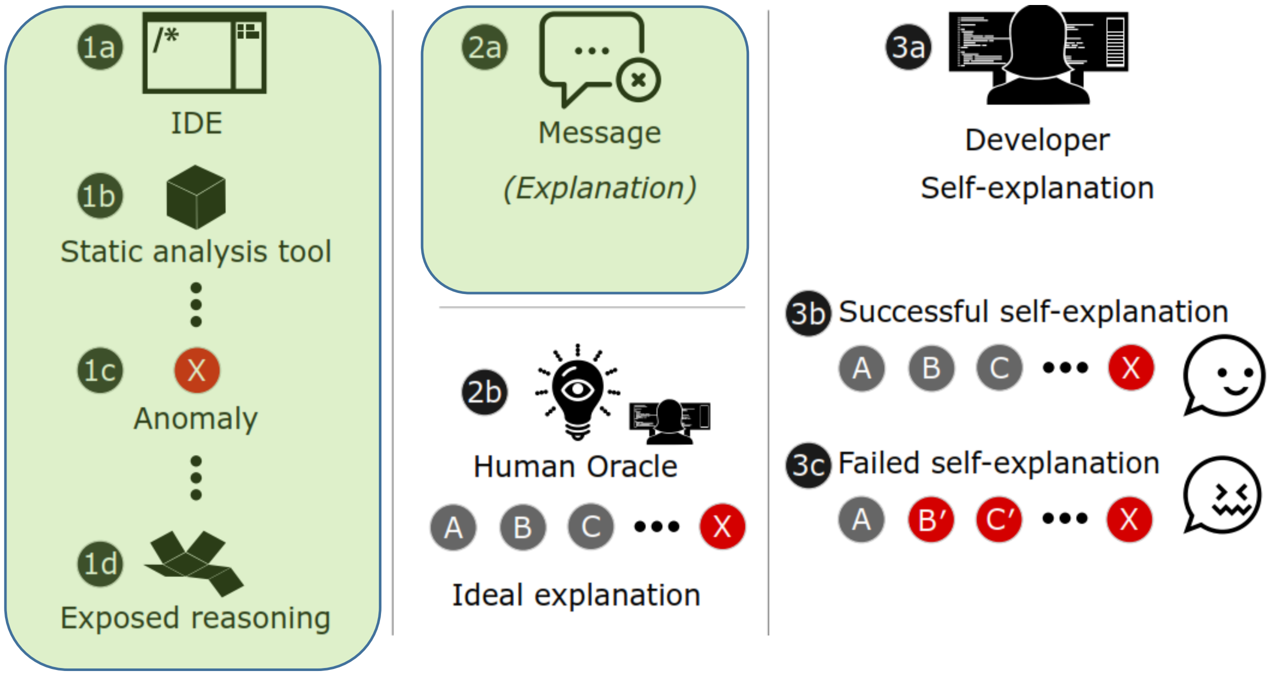
Stack Overflow: Methodology

Use the `votes` table for the answer to cluster good responses versus bad
respon
subset
and *qu*
messa

Quant
the fea
votes.



T. Barik, B. Johnson, and E. Murphy-Hill, "I ♥ Hacker News: Expanding qualitative research findings by analyzing social news websites," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, 2015, pp. 882–885.



Rust (4/5)

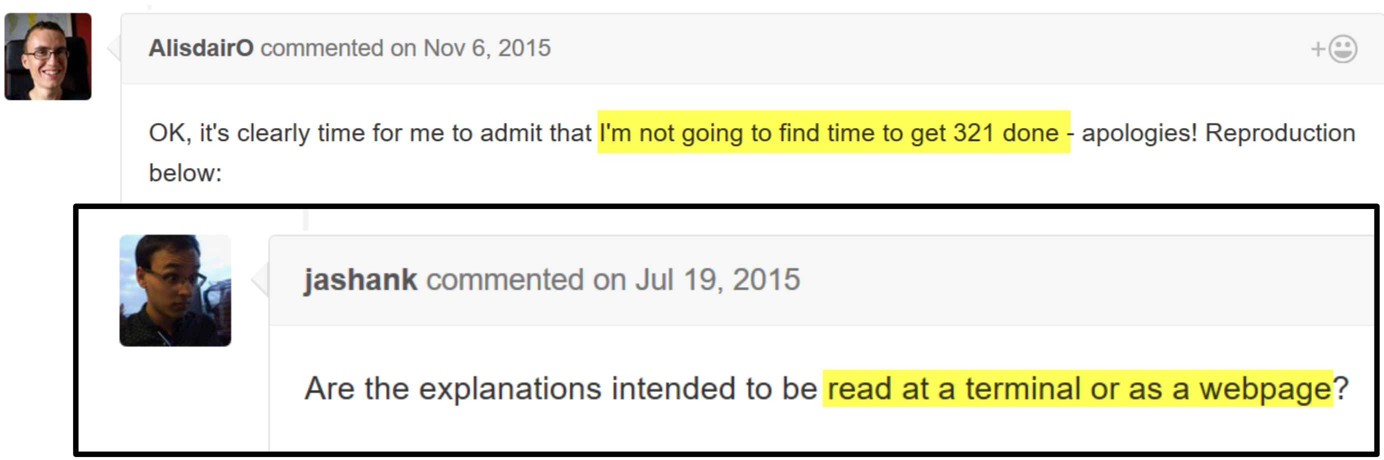
How do compiler authors design static analysis message in their tools?

(Proposed, Summer 2016)

Rust: Rationale

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.

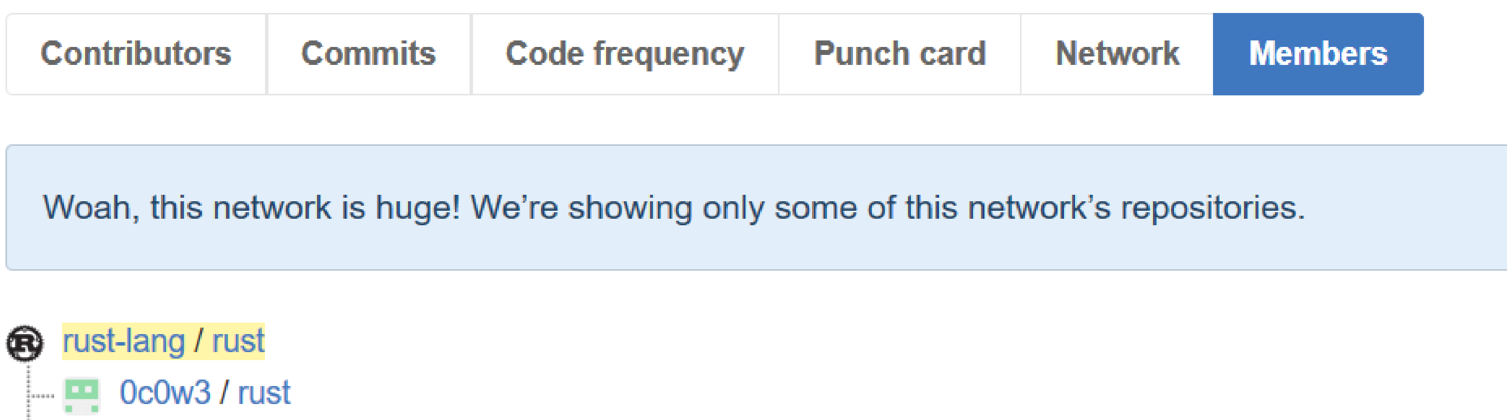
Understanding the "life cycle" of static analysis anomalies from the perspective of toolsmiths will enable researchers to identify and addresses barriers for toolsmiths.

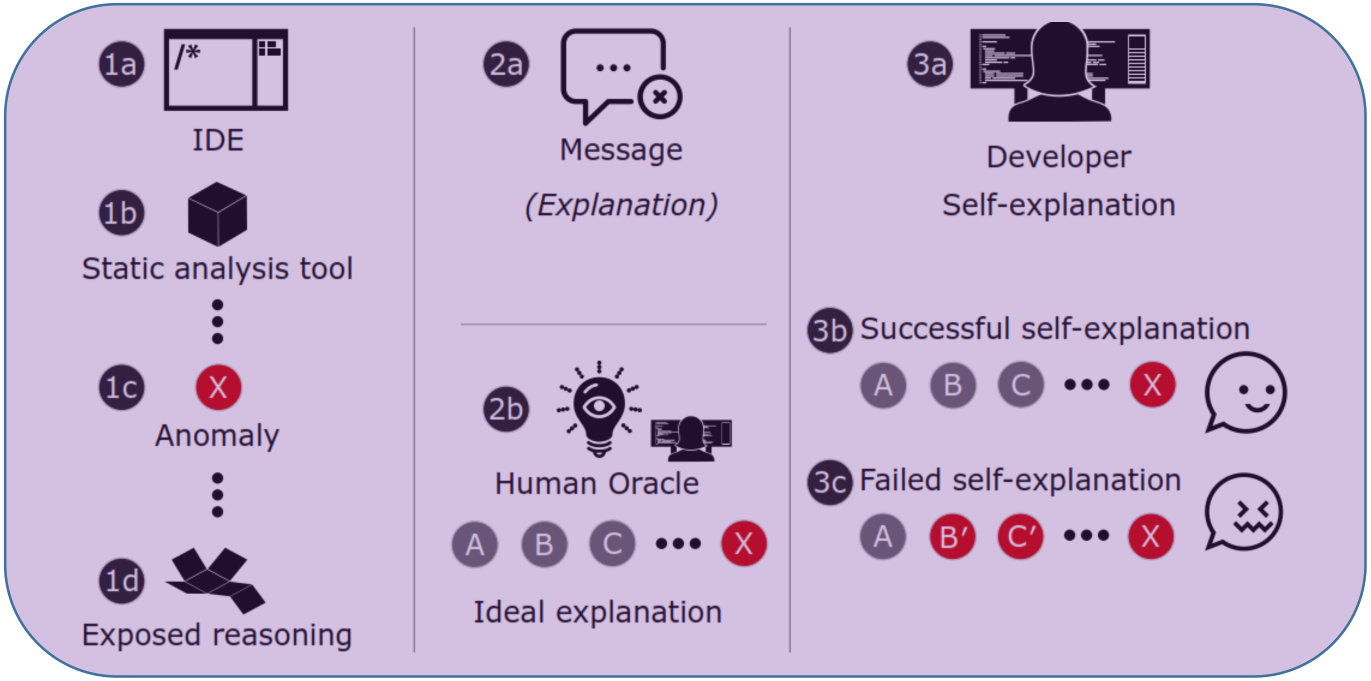


V. J. Traver, "On compiler error messages: What they say and what they mean," *Adv. Human-Computer Interact.*, vol. 2010, pp. 1–26, 2010.

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.



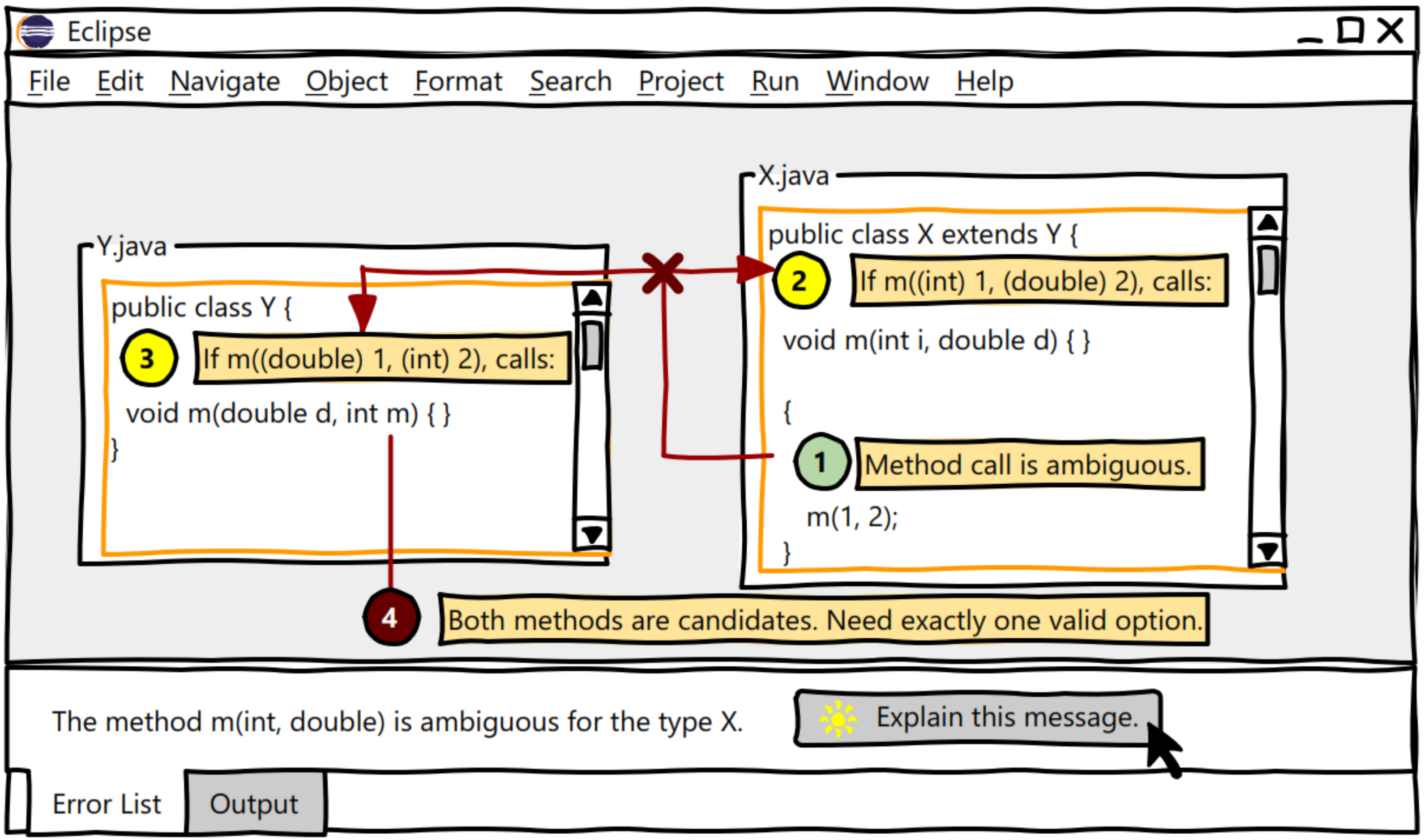


Radiance (5/5)

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

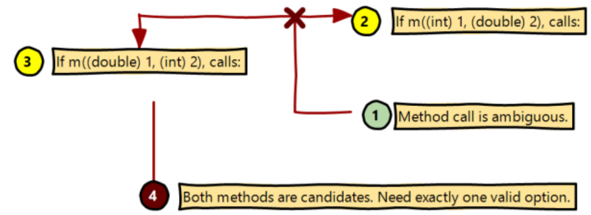
(In-Progress, Spring 2016)

Radiance: Rationale (and Principles)

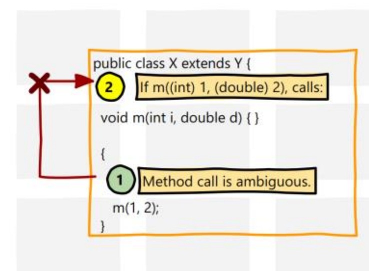


Radiance: Design Principles

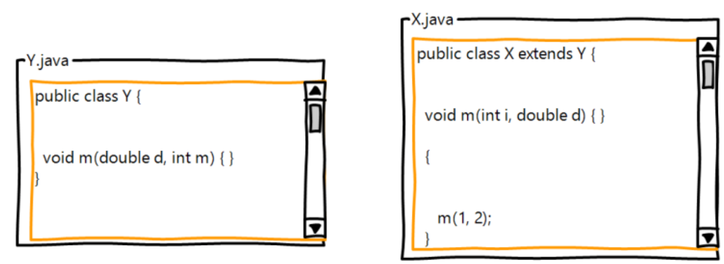
Explicate relationships. Make explicit the relations of *what* and *why* (Legare 2014).



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).



C. H. Legare, "The Contributions of Explanation and Exploration to Children's Scientific Reasoning," *Child Dev. Perspect.*, vol. 8, no. 2, pp. 101–106, Jun. 2014.

K. VanLehn and M. R. Johnes, "What mediates the self-explanation effect?" in *Conference of the Cognitive Science Society*, 1993, pp. 1034–1039.

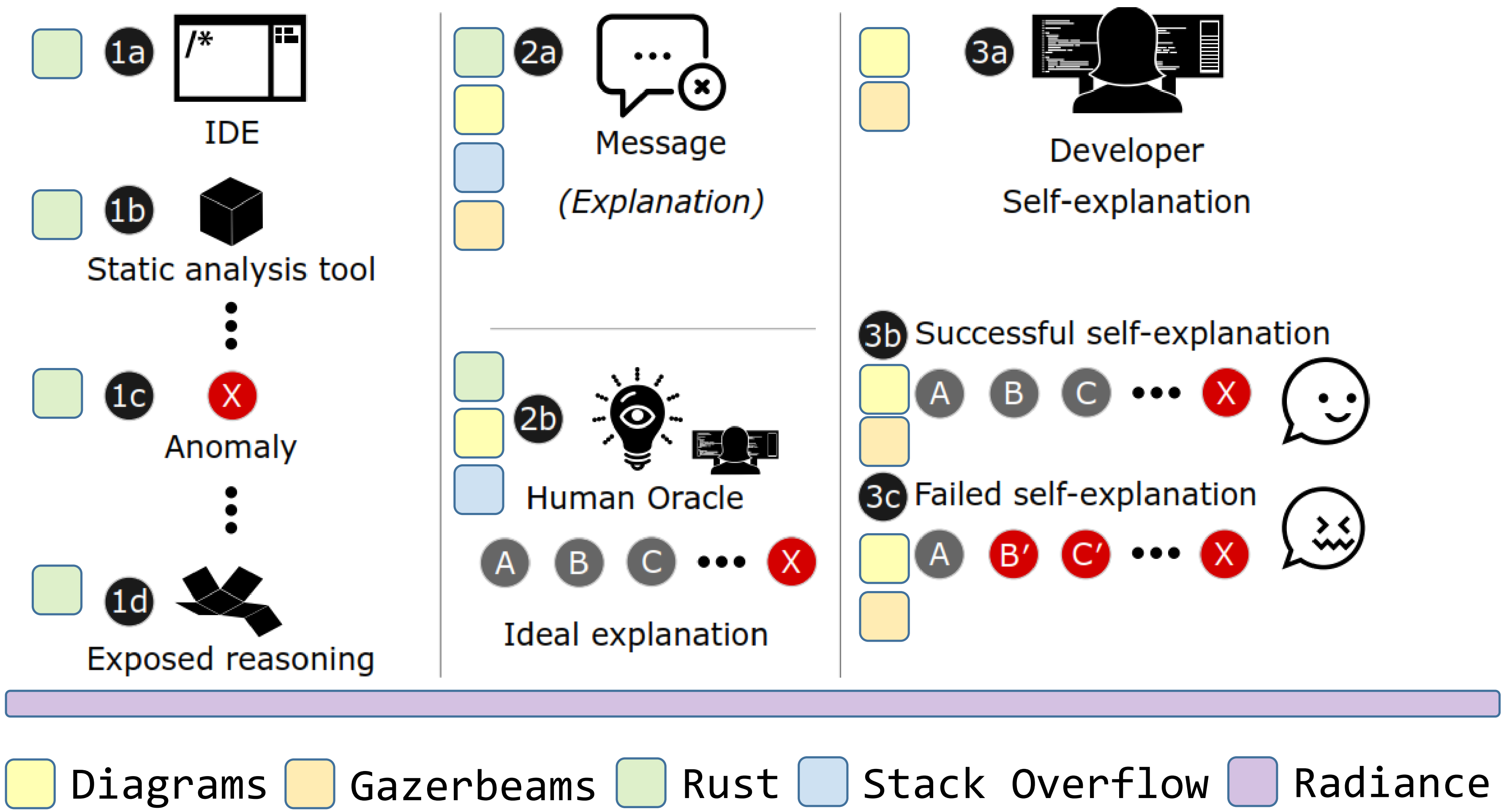
J. Larkin and H. Simon, "Why a Diagram is (Sometimes) Worth Ten Thousand Words," *Cogn. Sci.*, vol. 11, no. 1, pp. 65–100, Jan. 1987.

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



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

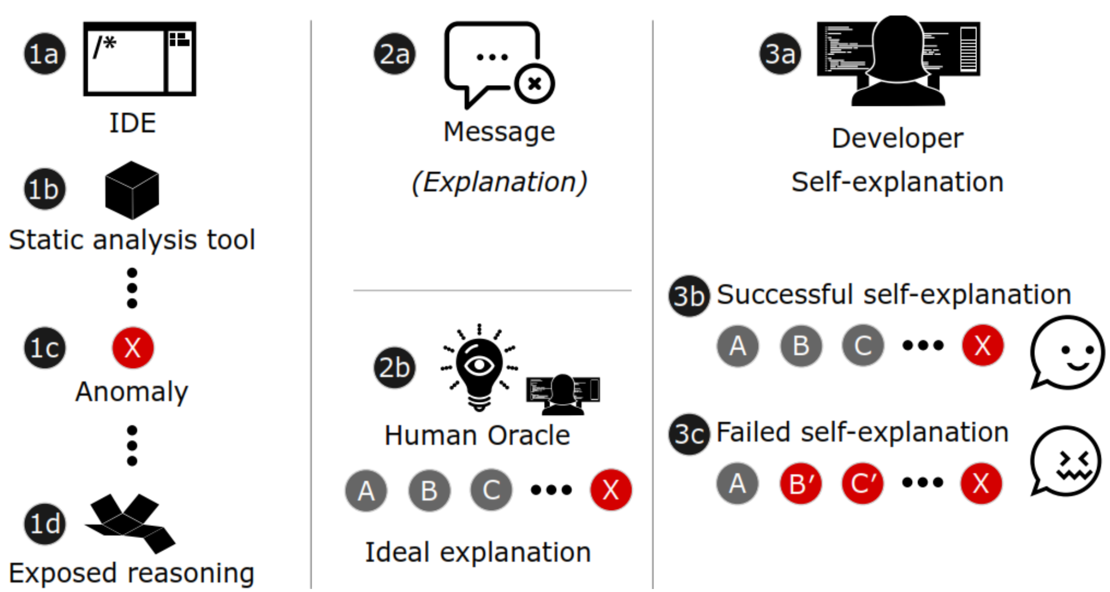
Dissertation Writing and Resubmit Cycle

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?

R. Azevedo and V. Aleven, *International Handbook of Metacognition and Learning Technologies*, vol. 28. New York, NY: Springer New York, 2013.

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?

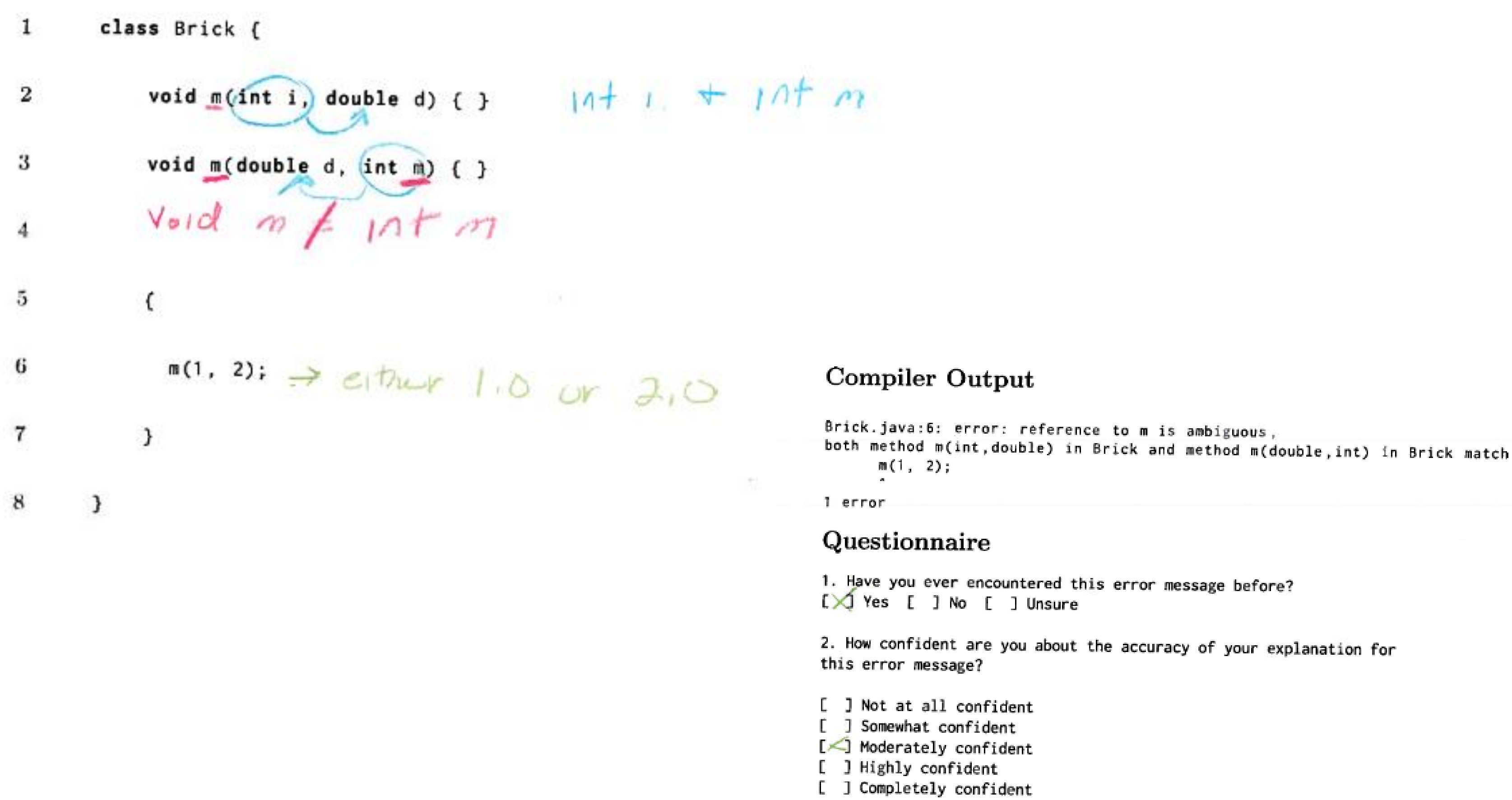
RQ4 What do developers say about why Radiance helps or prevents them from performing their task?

Explicit Relationality

Table 1. Opportunities for Improvement: Explicit Relationality

Static Analysis	All	Relational	Pct (%)
Eclipse JDK (Luna)	598	250	41.8%
Microsoft C# 5.0 (Roslyn)	1107	282	25.5%
Microsoft F# 3.1	1136	198	17.4%
Microsoft TypeScript 1.0	538	116	21.6%
Oracle OpenJDK 7	487	126	25.9%

Diagrams: Participant Annotations



```
class Brick {
    void m(int i, double d) { }
    void m(double d, int m) { }

    {
      m(1, 2);
    }
}
```


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
[X] Moderately confident
[ ] Highly confident
[ ] Completely confident

Diagrams: Annotation Legend

TABLE I: Frequency of Visual Annotations in Pilot

Annotation	Frequency	Description
Point	49	A particular token or set of tokens has been marked. Examples include underlining or circles the token(s).
Text	45	Natural language text. For example, “assign a value to the variable” or “dead code”.
Association	33	An association between two or more program elements, which is accomplished by drawing a connecting line between the elements, with or without arrow heads.
Symbol	20	Symbols include visual annotation such as ? or x, or numbered circles, to name a few.
Code	14	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.
Strikethrough	5	The strikethrough is separated from the point annotation because this annotation is provided by IDEs today, and has pre-established semantics.
Multicolor	-	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.

TABLE II: Visual Annotation Legend

Symbol	Description
code	The starting location of the error.
code	Indicates issues related to the error.
	Arrows can be followed. They indicate the next relevant location to check.
❶	Enumerations are used to number items of potential interest, especially when the information doesn’t fit within the source code.
?	The compiler expected an associated item, but cannot find it.
×	Conflict between items.
code	Explanatory code or code generated internally by the compiler. The code is not in the original source.
	Indicates code coverage. Green lines indicate successfully executed code. Red lines indicate failed or skipped lines.

Diagrams: Cognitive Dimensions

TABLE V: Cognitive Dimensions Questionnaire Responses

	Control		Treatment		
Dimension	Median	Dist	Median	Dist	p
Hidden Dependencies*	3		4		.008
Consistency	4		4		.979
Hard Mental Operations	3		2.5		.821
Role Expressiveness	4		4		.130

Diagrams: Tasks

TABLE III: Participant Explanation and Recall Tasks

Task Order	Task Name	OpenJDK File	Error Message
T1	Melon	VarMightNotHaveBeenInitialized.java	variable i might not have been initialized
T2	Kite	UnreportedExceptionDefaultConstructor.java	unreported exception Exception in default constructor
T3	Brick	RefAmbiguous.java	reference to m is ambiguous, both method m(int,double) in Brick and method m(double,int) in Brick match
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 FileNotFoundException,EOFException have already been caught

Radiance: Methodology

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

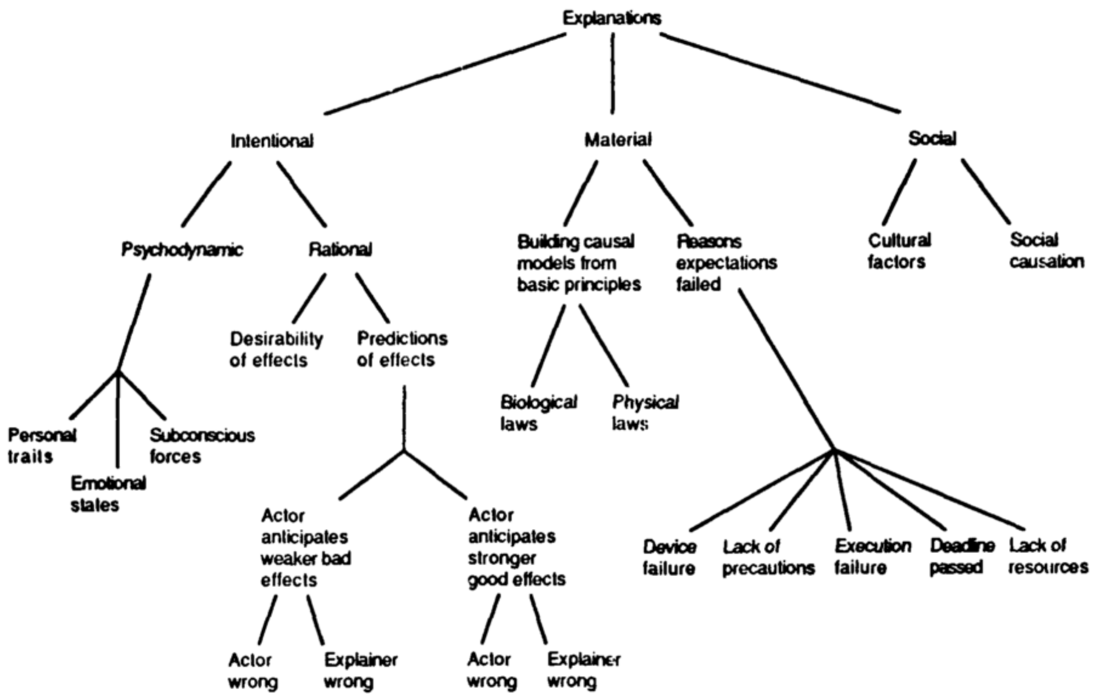
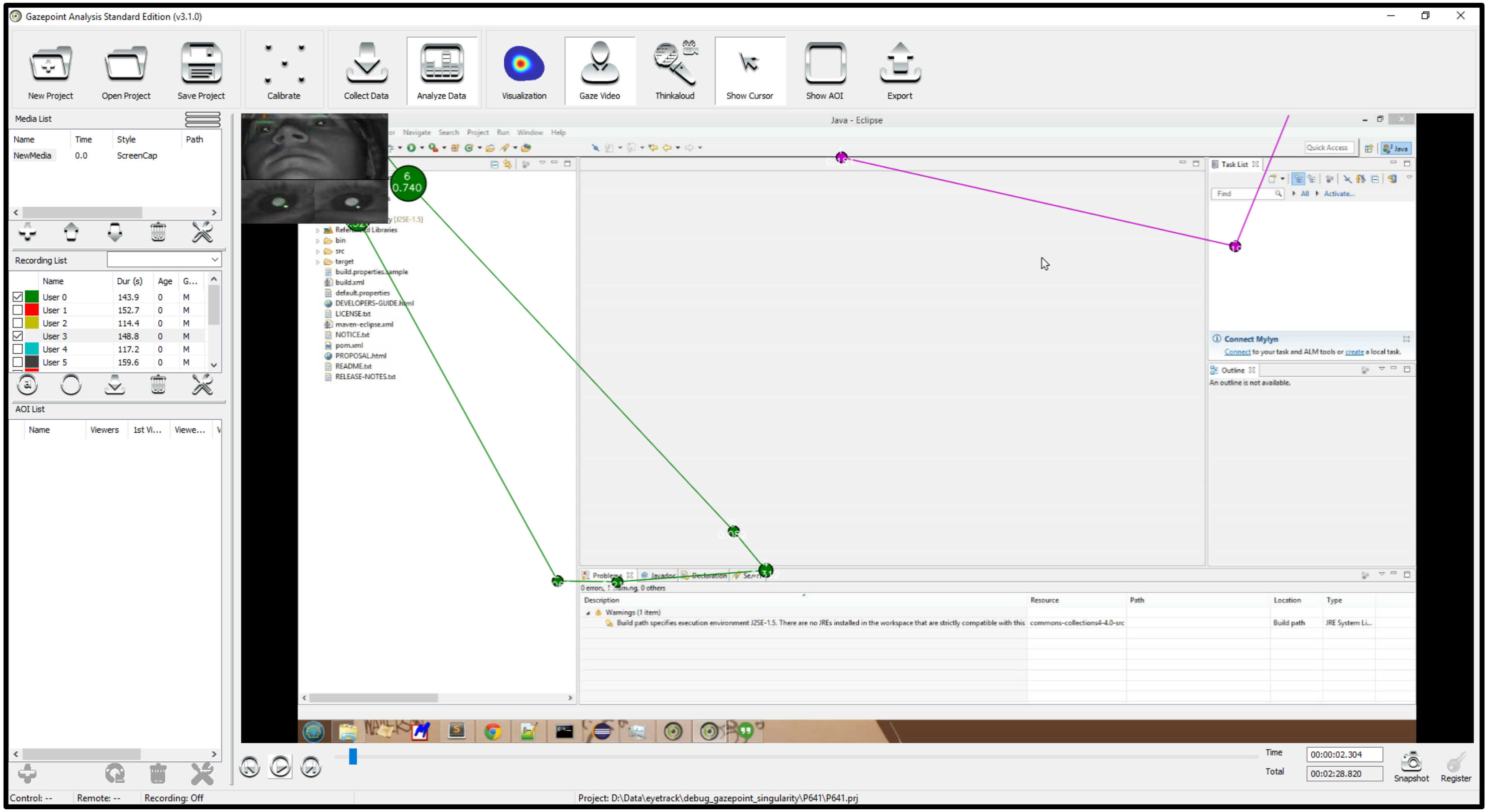


Figure 1: A hierarchy of types of explanations

D. B. Leake, *Evaluating explanations: A content theory*. Psychology Press, 2014.

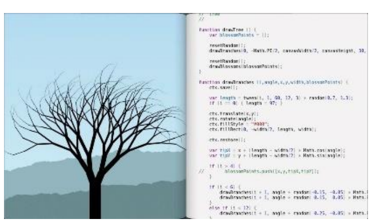
Gazerbeams: GazePoint Interface



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



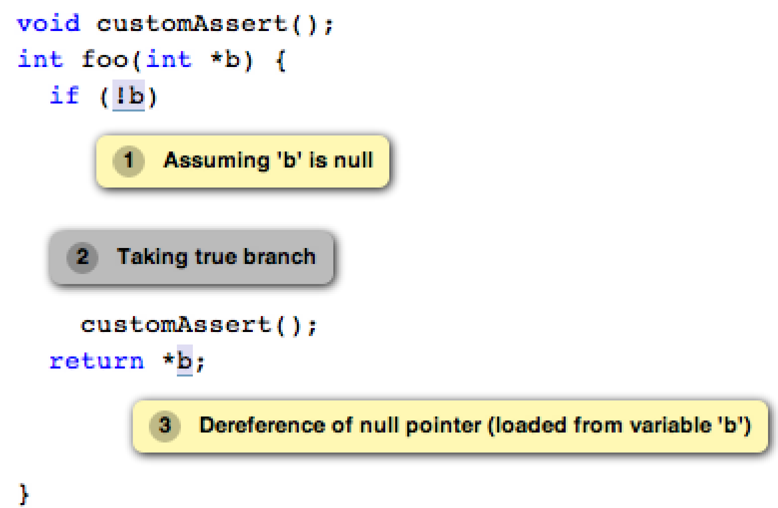
"If we're writing our code on a computer, why are we simulating what a computer would do in our head? Why doesn't the computer just do it, and show us?" – Bret Victor, *Inventing on Principle*.

Are incorrect explanations useful?

Increasing *completeness* is better than *soundness* (Kulesza 2013).

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

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



T. Kulesza, S. Stumpf, M. Burnett, S. Yang, I. Kwan, and W.-K. Wong, "Too much, too little, or just right? Ways explanations impact end users' mental models," in *VL/HCC*, 2013, pp. 3–10.

C. H. Legare, "Exploring explanation: explaining inconsistent evidence informs exploratory, hypothesis-testing behavior in young children.," *Child Dev.*, vol. 83, no. 1, pp. 173–85, Jan. 2012.

B. Y. Lim, A. K. Dey, and D. Avrahami, "Why and why not explanations improve the intelligibility of context-aware intelligent systems," in *CHI*, 2009, pp. 2119–2129.