# Hard-to-Answer Questions about Code

**Thomas D. LaToza** & Brad A. Myers



Carnegie Mellon
School of Computer Science

# Designing from data about developers

What makes engineering software **challenging**,
and how can tools, languages, or processes make developers more **effective**?

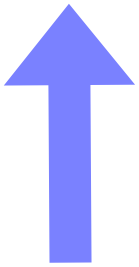
# Traditional SE / PL research

**design**

tools, languages,
processes, ...

# Empirical studies to validate claims

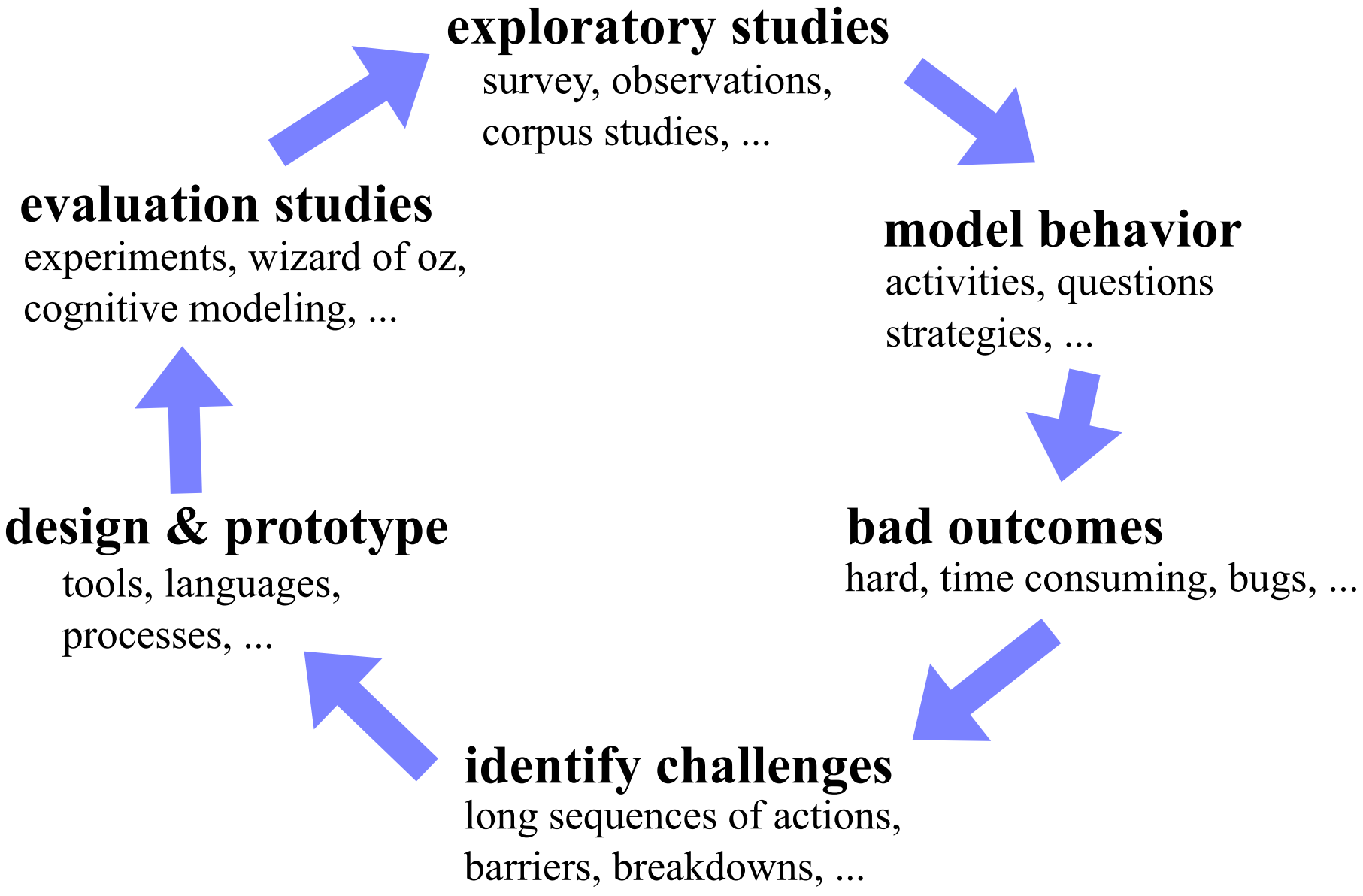
**evaluation studies**
experiments



**design**
tools, languages,
processes, ...

What are developers'
**problems**?

# Designing from data about developers



# Answering reachability questions

13, 179, 17 developers

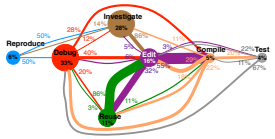
## exploratory studies

observations of developers in the **lab, survey**,
observations of developers in the **field**

| | ECLIPSE | REACHER |
|---|---|---|
| succeeded | 0% | 50% |
| gave up | 17% | 0% |
| avg time ±std dev (mins) | 26.5 ± 7.7 | 12.8 ±9.5 |

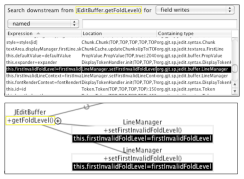
## evaluation studies

wizard of oz, lab experiment




## model behavior

developers answer **reachability** questions during debugging or implication activities by **traversing** control flow paths

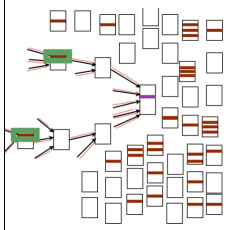


## design & prototype

Reacher lets developers **search** along control flow paths

## bad outcomes

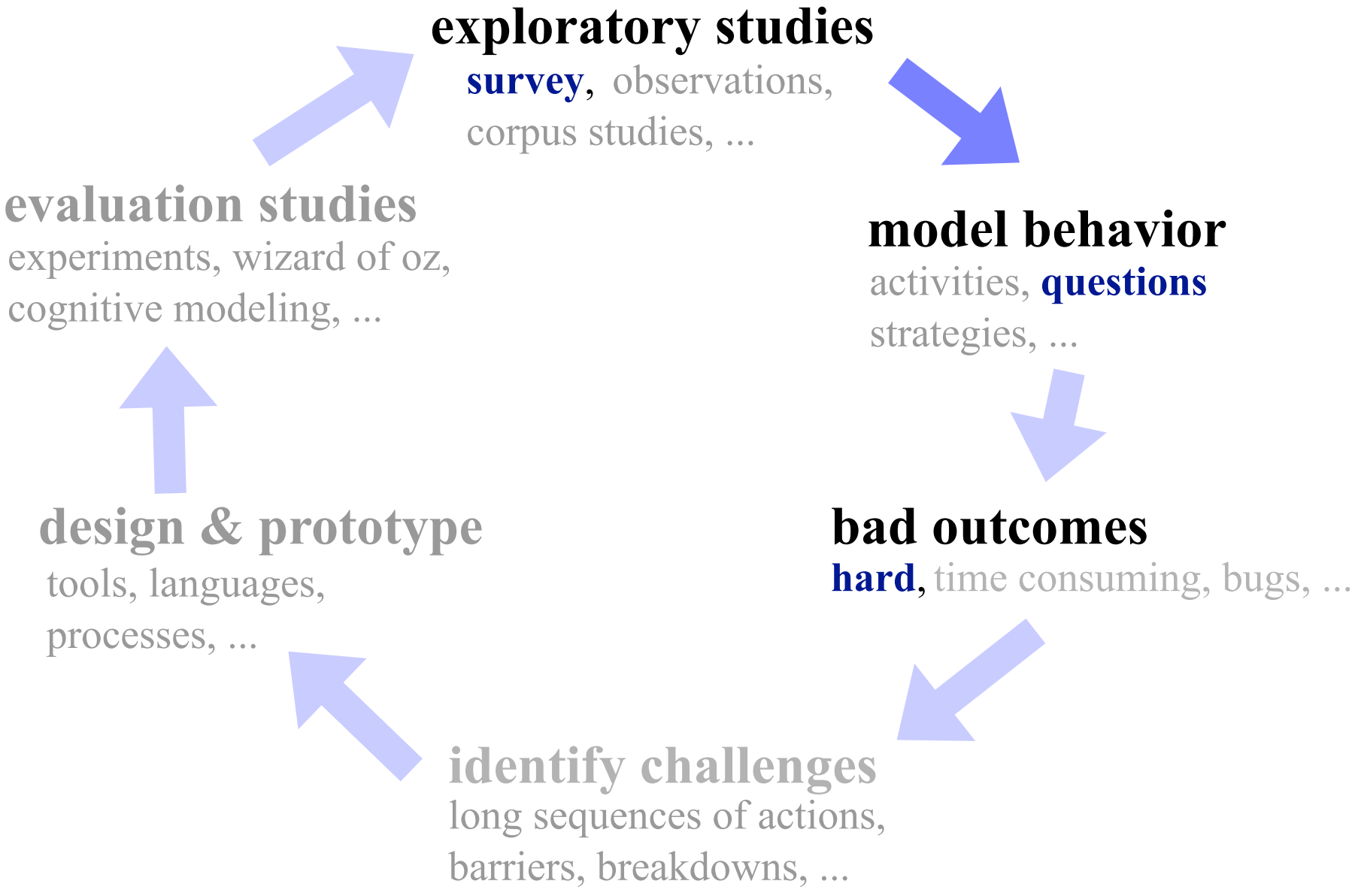
causes **bugs**,
82% agree **hard** to answer,
time consuming (**10s** of mins)



## identify challenges

must guess which paths lead to **targets** and which are **feasible**

# Hard-to-answer questions about code



# Developers' information needs

What research tools / languages address these questions?

What **gaps** exist between questions and existing research?

Follows **similar** studies, but this study is focused on

-questions about code
-developers perceive to be **hard-to-answer**
-**larger** sample of developers from survey
-most **comprehensive** list


Ko, A. J., DeLine, R., and Venolia, G. (2007). Information needs in collocated software development teams. In *ICSE*, 344-353.

Fritz, T., and Murphy, G.C. (2010). Using information fragments to answer the questions developers ask. In *Proc. of the Int'l Conf. on Soft. Eng. (ICSE), 175-184.*

LaToza, T.D., Myers, B.A. (2010). Developers ask reachability questions. In *Proc. of the Int'l Conf. on Soft. Eng. (ICSE).*

Sillito, J., Murphy, G.C., and De Volder, K. (2008). Asking and answering questions during a programming change task. In *Transactions on Software Engineering (TSE)*, 34(4).

# Method

| | |
|---|---|
| **Prompt** | What **hard to answer questions** about code have you recently asked? |
| **Respondents** | 179 **professional** developers at Microsoft |
| **Data** | **paragraphs** describing questions, stories, scenarios |
| **Analysis** | broke up into **questions** |
| | clustered into distinct questions based on intent - **94** questions |
| | clustered into **21** categories |
| | identified **existing** approaches that address questions |

# Interpretation of results

For each question

A **professional** software developer thinks it's hard-to-answer

Hard-to-answer questions could be caused by 4 reasons

-not effectively using the best **existing** tools / languages / ...

-not using research tools / ... / ...

-research addresses question but doesn't yet help

-research hasn't looked at this problem

**#** reports - e.g., 6 developers reported it

-NOT frequency, but maybe correlated

-**saliency** / difficulty?

-subject to memory / awareness biases

-from a single organization

-about questions, likely doesn't capture all problems

# Results

No **dominant** hard question

**Rationale** questions both most frequently reported and unaddressed by research

Many hard-to-answer questions already **addressed** by research

Many hard-to-answer questions **are not** addressed by research

Gaps between **situations** addressed by research and problems developers reported

# Hard-to-answer questions about code

## Rationale (42)

*Why was it done this way? (14) [15][7]*
*Why wasn't it done this other way? (15)*
*Was this intentional, accidental, or a hack? (9)[15]*
*How did this ever work? (4)*

## Debugging (26)

*How did this runtime state occur? (12) [15]*
*What runtime state changed when this executed? (2)*
*Where was this variable last changed? (1)*
*How is this object different from that object? (1)*
*Why didn't this happen? (3)*
*How do I debug this bug in this environment? (3)*
*In what circumstances does this bug occur? (3) [15]*
*Which team's component caused this bug? (1)*

## Intent and Implementation (32)

*What is the intent of this code? (12) [15]*
*What does this do (6) in this case (10)? (16) [24]*
*How does it implement this behavior? (4) [24]*

## Refactoring (25)

*Is there functionality or code that could be refactored? (4)*
*Is the existing design a good design? (2)*
*Is it possible to refactor this? (9)*
*How can I refactor this (2) without breaking existing users(7)? (9)*
*Should I refactor this? (1)*
*Are the benefits of this refactoring worth the time investment? (3)*

## History (23)

*When, how, by whom, and why was this code changed or inserted? (13)[7]*
*What else changed when this code was changed or inserted? (2)*
*How has it changed over time? (4)[7]*
*Has this code always been this way? (2)*
*What recent changes have been made? (1)[15][7]*
*Have changes in another branch been integrated into this branch? (1)*

## Implications (21)

*What are the implications of this change for (5) API clients (5), security (3), concurrency (3), performance (2), platforms (1), tests (1), or obfuscation (1)? (21) [15][24]*

## Testing (20)

*Is this code correct? (6) [15]*
*How can I test this code or functionality? (9)*
*Is this tested? (3)*
*Is the test or code responsible for this test failure? (1)*
*Is the documentation wrong, or is the code wrong? (1)*

## Implementing (19)

*How do I implement this (8), given this constraint (2)? (10)*
*Which function or object should I pick? (2)*
*What's the best design for implementing this? (7)*

## Control flow (19)

*In what situations or user scenarios is this called? (3) [15][24]*
*What parameter values does each situation pass to this method? (1)*
*What parameter values could lead to this case? (1)*
*What are the possible actual methods called by dynamic dispatch here? (6)*
*How do calls flow across process boundaries? (1)*
*How many recursive calls happen during this operation? (1)*
*Is this method or code path called frequently, or is it dead? (4)*
*What throws this exception? (1)*
*What is catching this exception? (1)*

## Contracts (17)

*What assumptions about preconditions does this code make? (5)*
*What assumptions about pre(3)/post(2)conditions can be made?*
*What exceptions or errors can this method generate? (2)*
*What are the constraints on or normal values of this variable? (2)*
*What is the correct order for calling these methods or initializing these objects? (2)*
*What is responsible for updating this field? (1)*

## Performance (16)

*What is the performance of this code (5) on a large, real dataset (3)? (8)*
*Which part of this code takes the most time? (4)*
*Can this method have high stack consumption from recursion? (1)*
*How big is this in memory? (2)*
*How many of these objects get created? (1)*

## Teammates (16)

*Who is the owner or expert for this code? (3)[7]*
*How do I convince my teammates to do this the "right way"? (12)*
*Did my teammates do this? (1)*

## Policies (15)

*What is the policy for doing this? (10) [24]*
*Is this the correct policy for doing this? (2) [15]*
*How is the allocation lifetime of this object maintained? (3)*

## Type relationships (15)

*What are the composition, ownership, or usage relationships of this type? (5) [24]*
*What is this type's type hierarchy? (4) [24]*
*What implements this interface? (4) [24]*
*Where is this method overridden? (2)*

## Data flow (14)

*What is the original source of this data? (2) [15]*
*What code directly or indirectly uses this data? (5)*
*Where is the data referenced by this variable modified? (2)*
*Where can this global variable be changed? (1)*
*Where is this data structure used (1) for this purpose (1)? (2) [24]*
*What parts of this data structure are modified by this code? (1) [24]*
*What resources is this code using? (1)*

## Location (13)

*Where is this functionality implemented? (5) [24]*
*Is this functionality already implemented? (5) [15]*
*Where is this defined? (3)*

## Building and branching (11)

*Should I branch or code against the main branch? (1)*
*How can I move this code to this branch? (1)*
*What do I need to include to build this? (3)*
*What includes are unnecessary? (2)*
*How do I build this without doing a full build? (1)*
*Why did the build break? (2)[59]*
*Which preprocessor definitions were active when this was built? (1)*

## Architecture (11)

*How does this code interact with libraries? (4)*
*What is the architecture of the code base? (3)*
*How is this functionality organized into layers? (1)*
*Is our API understandable and flexible? (3)*

## Concurrency (9)

*What threads reach this code (4) or data structure (2)? (6)*
*Is this class or method thread-safe? (2)*
*What members of this class does this lock protect? (1)*

## Dependencies (5)

*What depends on this code or design decision? (4)[7]*
*What does this code depend on? (1)*

## Method properties (2)

*How big is this code? (1)*
*How overloaded are the parameters to this function? (1)*

# Hard-to-answer questions about code

## Rationale (42)
*Why was it done this way? (14) [15][7]*
*Why wasn't it done this other way? (15)*
*Was this intentional, accidental, or a hack? (9)[15]*
*How did this ever work? (4)*

## Debugging (26)
*How did this runtime state occur? (12) [15]*
*What runtime state changed when this executed? (2)*
*Where was this variable last changed? (1)*
*How is this object different from that object? (1)*
*Why didn't this happen? (3)*
*How do I debug this bug in this environment? (3)*
*In what circumstances does this bug occur? (3) [15]*
*Which team's component caused this bug? (1)*

## Intent and Implementation (32)
*What is the intent of this code? (12) [15]*
*What does this do (6) in this case (10)? (16) [24]*
*How does it implement this behavior? (4) [24]*

## Refactoring (25)
*Is there functionality or code that could be refactored? (4)*
*Is the existing design a good design? (2)*
*Is it possible to refactor this? (9)*
*How can I refactor this (2) without breaking existing users(7)? (9)*
*Should I refactor this? (1)*
*Are the benefits of this refactoring worth the time investment? (3)*

## History (23)
*When, how, by whom, and why was this code changed or inserted? (13)[7]*
*What else changed when this code was changed or inserted? (2)*
*How has it changed over time? (4)[7]*
*Has this code always been this way? (2)*
*What recent changes have been made? (1)[15][7]*
*Have changes in another branch been integrated into this branch? (1)*

## Implications (21)
*What are the implications of this change for (5) API clients (5), security (3), concurrency (3), performance (2), platforms (1), tests (1), or obfuscation (1)? (21) [15][24]*

## Testing (20)
*Is this code correct? (6) [15]*
*How can I test this code or functionality? (9)*
*Is this tested? (3)*
*Is the test or code responsible for this test failure? (1)*
*Is the documentation wrong, or is the code wrong? (1)*

## Implementing (19)
*How do I implement this (8), given this constraint (2)? (10)*
*Which function or object should I pick? (2)*
*What's the best design for implementing this? (7)*

## Control flow (19)
*In what situations or user scenarios is this called? (3) [15][24]*
*What parameter values does each situation pass to this method? (1)*
*What parameter values could lead to this case? (1)*
*What are the possible actual methods called by dynamic dispatch here? (6)*
*How do calls flow across process boundaries? (1)*
*How many recursive calls happen during this operation? (1)*
*Is this method or code path called frequently, or is it dead? (4)*
*What throws this exception? (1)*
*What is catching this exception? (1)*

## Contracts (17)
*What assumptions about preconditions does this code make? (5)*
*What assumptions about pre(3)/post(2)conditions can be made?*
*What exceptions or errors can this method generate? (2)*
*What are the constraints on or normal values of this variable? (2)*
*What is the correct order for calling these methods or initializing these objects? (2)*
*What is responsible for updating this field? (1)*

## Performance (16)
*What is the performance of this code (5) on a large, real dataset (3)? (8)*
*Which part of this code takes the most time? (4)*
*Can this method have high stack consumption from recursion? (1)*
*How big is this in memory? (2)*
*How many of these objects get created? (1)*

## Teammates (16)
*Who is the owner or expert for this code? (3)[7]*
*How do I convince my teammates to do this the "right way"? (12)*
*Did my teammates do this? (1)*

## Policies (15)
*What is the policy for doing this? (10) [24]*
*Is this the correct policy for doing this? (2) [15]*
*How is the allocation lifetime of this object maintained? (3)*

## Type relationships (15)
*What are the composition, ownership, or usage relationships of this type? (5) [24]*
*What is this type's type hierarchy? (4) [24]*
*What implements this interface? (4) [24]*
*Where is this method overridden? (2)*

## Data flow (14)
*What is the original source of this data? (2) [15]*
*What code directly or indirectly uses this data? (5)*
*Where is the data referenced by this variable modified? (2)*
*Where can this global variable be changed? (1)*
*Where is this data structure used (1) for this purpose (1)? (2) [24]*
*What parts of this data structure are modified by this code? (1) [24]*
*What resources is this code using? (1)*

## Location (13)
*Where is this functionality implemented? (5) [24]*
*Is this functionality already implemented? (5) [15]*
*Where is this defined? (3)*

## Building and branching (11)
*Should I branch or code against the main branch? (1)*
*How can I move this code to this branch? (1)*
*What do I need to include to build this? (3)*
*What includes are unnecessary? (2)*
*How do I build this without doing a full build? (1)*
*Why did the build break? (2)[59]*
*Which preprocessor definitions were active when this was built? (1)*

## Architecture (11)
*How does this code interact with libraries? (4)*
*What is the architecture of the code base? (3)*
*How is this functionality organized into layers? (1)*
*Is our API understandable and flexible? (3)*

## Concurrency (9)
*What threads reach this code (4) or data structure (2)? (6)*
*Is this class or method thread-safe? (2)*
*What members of this class does this lock protect? (1)*

## Dependencies (5)
*What depends on this code or design decision? (4)[7]*
*What does this code depend on? (1)*

## Method properties (2)
*How big is this code? (1)*
*How overloaded are the parameters to this function? (1)*

# Results

✻ ***Question (# reports)***

**existing tools, languages, processes, address**

✖ situations developers described

**but have limitations in situations respondents described**

✖ ***Question (# reports)***

**(existing research does not address this question)**

# Rationale (42)

- *Why **wasn't** it done this other way? (15)*
- ***Why** was it done this way? (14)*

naming, code structure, inheritance relationships, where resources freed, code duplication, algorithm choice, optimization, parameter validation visibility, exception policies

- *How did this **ever** work? (4)*
- *Was this **intentional**, accidental, or a hack? (9)*

# Debugging (26)

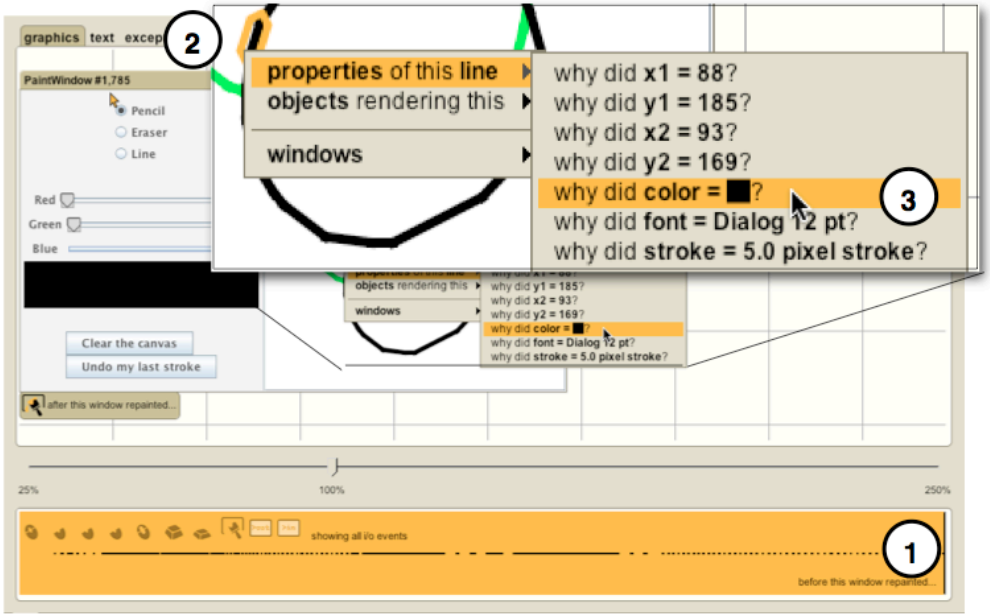
- *How did this **runtime state** occur? (12)*
  data, memory corruption, race conditions, hangs, crashes, failed API calls, test failures, null pointers
- *Where was this **variable** last changed? (1)*
- *Why **didn't** this happen? (3)*

**omniscient debuggers**

Record execution history
Provide interactions for browsing or searching

## WhyLine [1]

directly supports all 3 questions in some situations




[1] Ko, A.J., and Myers, B.A. (2008). Debugging reinvented: asking and answering why and why not questions about program behavior. In *Proc. of the Int'l Conf. on Soft. Eng. (ICSE).*

# Debugging (26)

* *How do I debug this bug in this **environment**?(3)*

* *In what **circumstances** does this bug occur? (3)*

**statistical debugging [1]**

*-Sample execution traces on **user** computers*
*-Find **correlations** between crashes and predicates*

No need to reproduce environment on developer computer

Examine correlations between crashes and predicates

[1] Liblit, B., Aiken, A., Zheng, A. X., and Jordan, M. I. 2003. Bug isolation via remote program sampling. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*.

# Debugging (26)

- *How is this object **different** from that object? (1)*
- *Which **team's** component caused this bug? (1)*

  Which team should I assign this bug to?
- *What runtime state **changed** when this executed? (2)*

# Refactoring (25)

* *Is the existing design a **good** design? (2)*

**smell detectors [1], metrics**

Look for bad design idioms
Suggests developer refactor

data clumps
feature envy
refused bequest
typecast

instanceof
magic number
long method
large class

* *Is there functionality or code that **could** be refactored? (4)*

**clone detectors [2]**

Detects syntactically similar code
Suggests developer refactor

clone

```
ComponentUI mui = new MultiButtonUI();
return MultiLookAndFeel.createUIs(mui,
        (MultiButtonUI) mui);
```

```
ComponentUI mui = new MutilColorChooserUI();
return MultiLookAndFeel.createUIs(mui,
        (MultiColorChooserUI) mui);
```

✖ obsolete code, duplicated functionality, redundant data between equally accessible data structures


[1] Murphy-Hill, E. and Black, A. P. (2008). Seven habits of a highly effective smell detector. In *Proc of Recommendation Systems for Software Engineering* at *FSE*.

[2] Kamiya, T., Kusumoto, S., and Inoue, K. (2002). CCFinder: a multi-linguistic token-based code clone detection system for large scale source code. In *TSE*, 28(7).

# Refactoring (25)

- ***Should*** *I refactor this? (1)*
- *Are the* ***benefits*** *of this refactoring worth the time investment? (3)*

# Refactoring (25)

* *Is it **possible** to refactor this? (9)*
* ***How** can I refactor this (2) without breaking existing users(7)?*

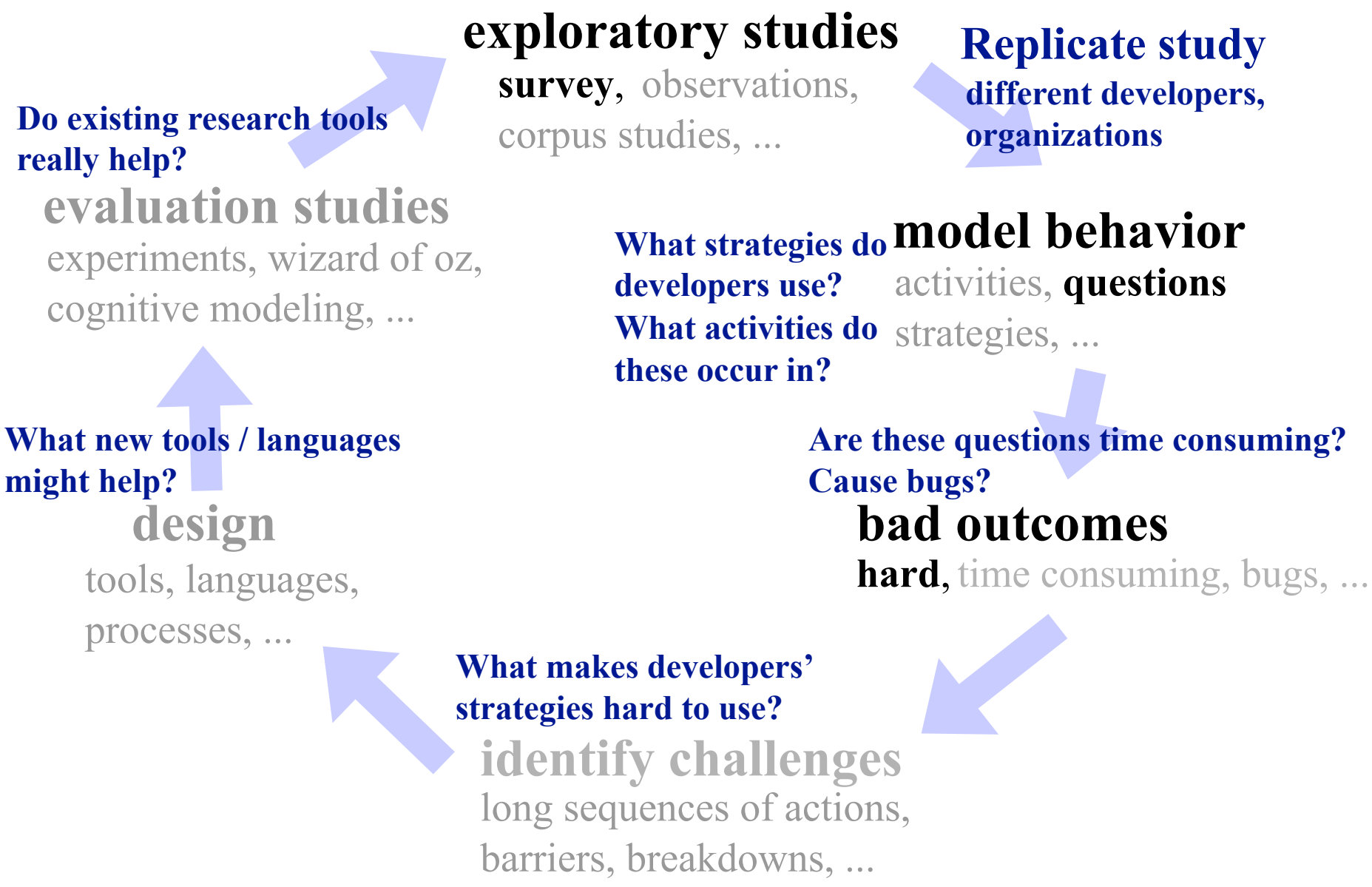
**IDE refactoring automation**

rename
move
pull up
push down
encapsulate field
convert local variable to field
....

✖ changing a method's scope, moving functionality between layers, changing semantics of config values, making operations more data driven, generalizing code to be more reusable

**higher-level refactorings**

# Many opportunities for future work



# Conclusions

Biggest value of exploratory studies in **specifics**, not generalities

- -Not surprising that debugging, rationale, refactoring are hard
- -But developers sometimes debug to triage
- -Ask why wasn't it done this other way
- -Refactor to remove redundant data

**Some** SE / PL research addresses developers' hard-to-answer questions

- Does it help? How could it better answer these questions?

Many opportunities for whole new **research areas** focused on unaddressed questions

# Questions?

Thomas LaToza
tlatoza@cs.cmu.edu

**Are you hiring?**

## Acknowledgements

Microsoft Research HiP NSF grant CCF-0811610