Information Needs

SWE 795, Spring 2017

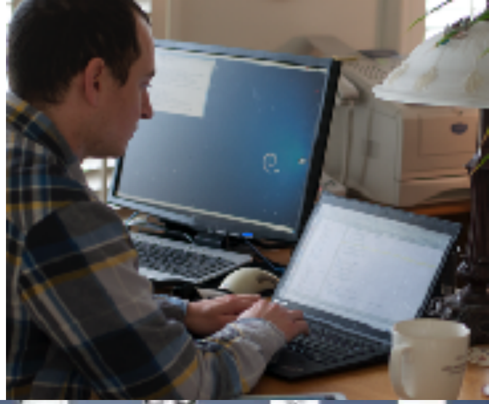
Software Engineering Environments

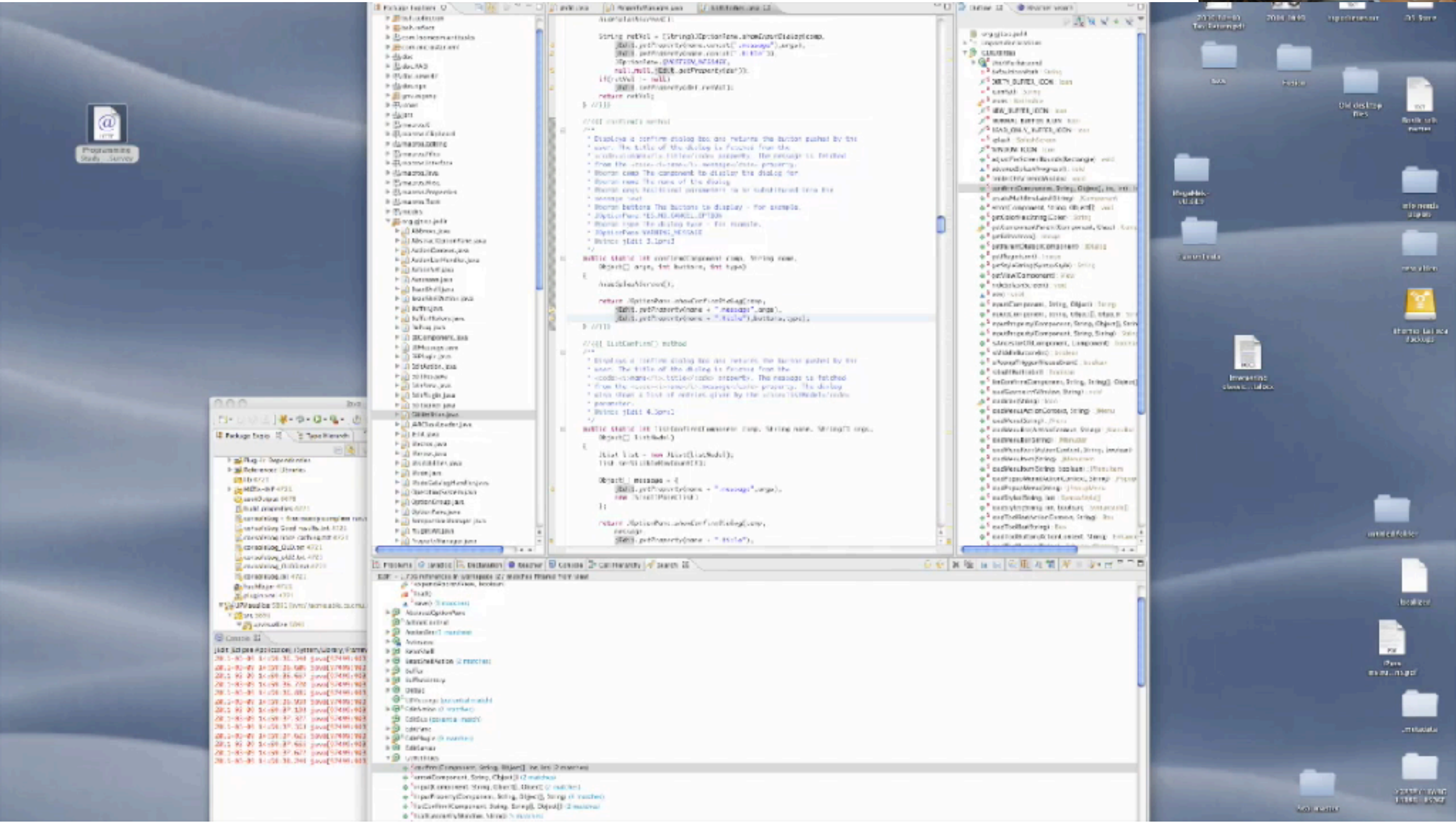
Today

- Part 1 (Lecture)(~30 mins)
  - Sketching
- Part 2 (Lecture)(~45 mins)
  - What models help describe programming tasks?
- Break!

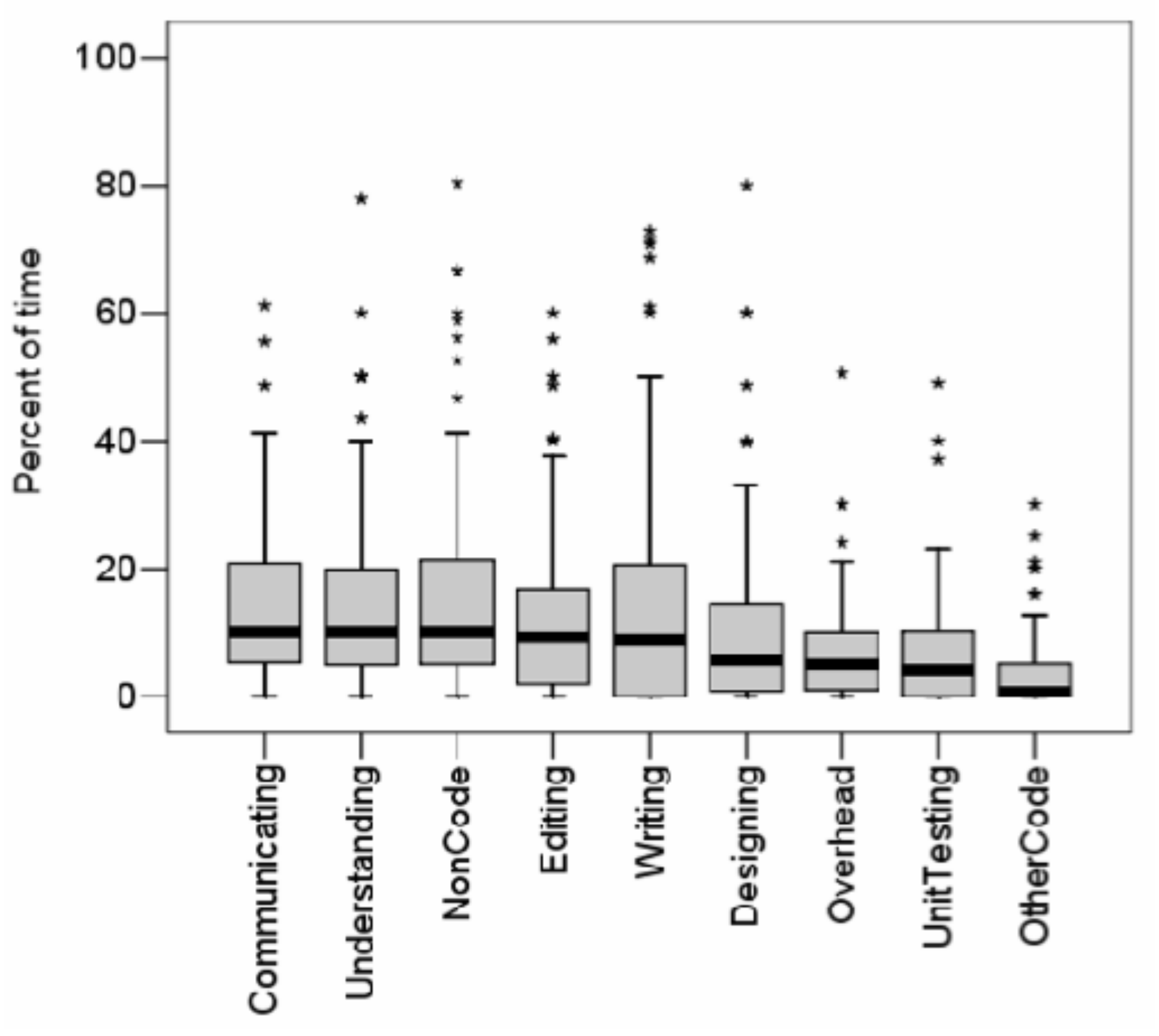
- Part 3 (Project Work)(~30 mins)
- Part 3 (Discussion)(45 mins)
  - Discussion of readings

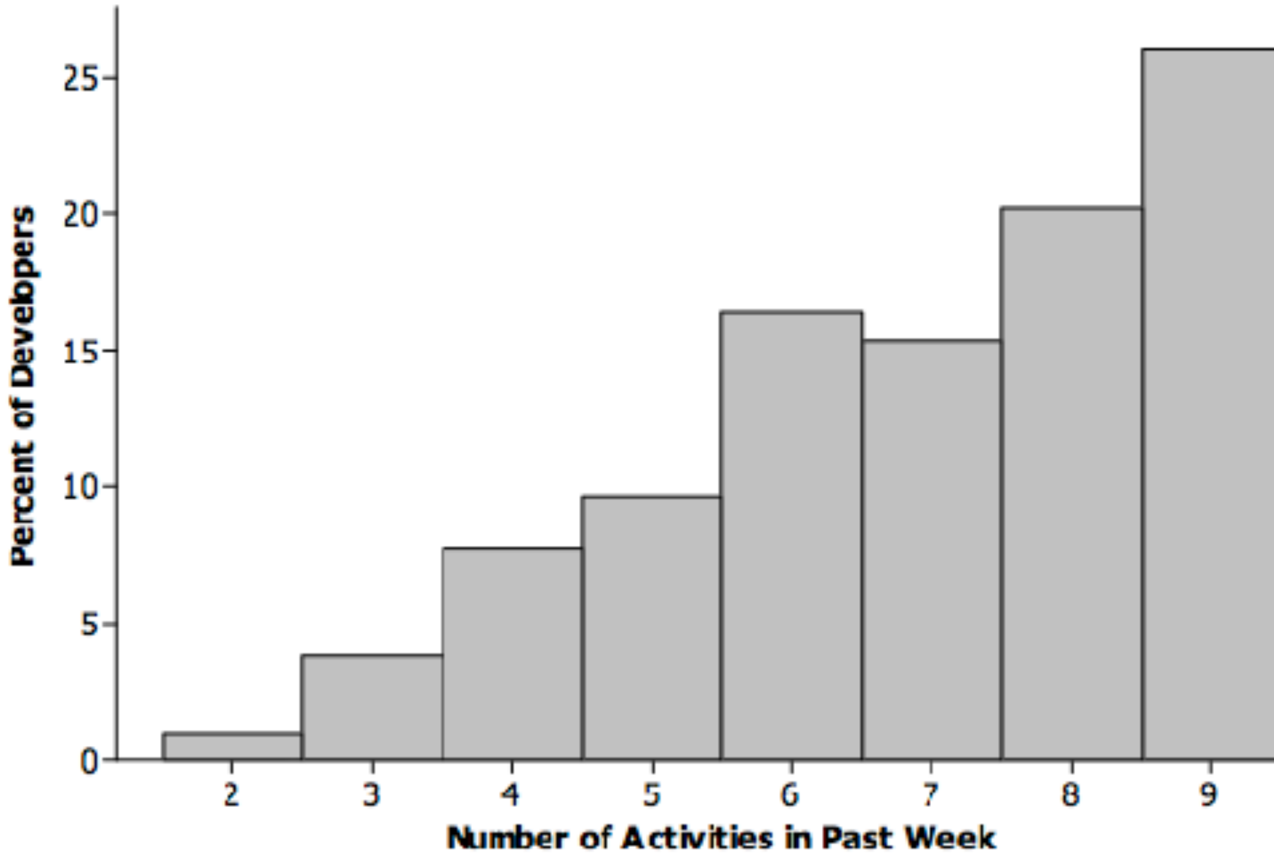
A few minutes in the life of a developer





What percentage of the last week have you spent…





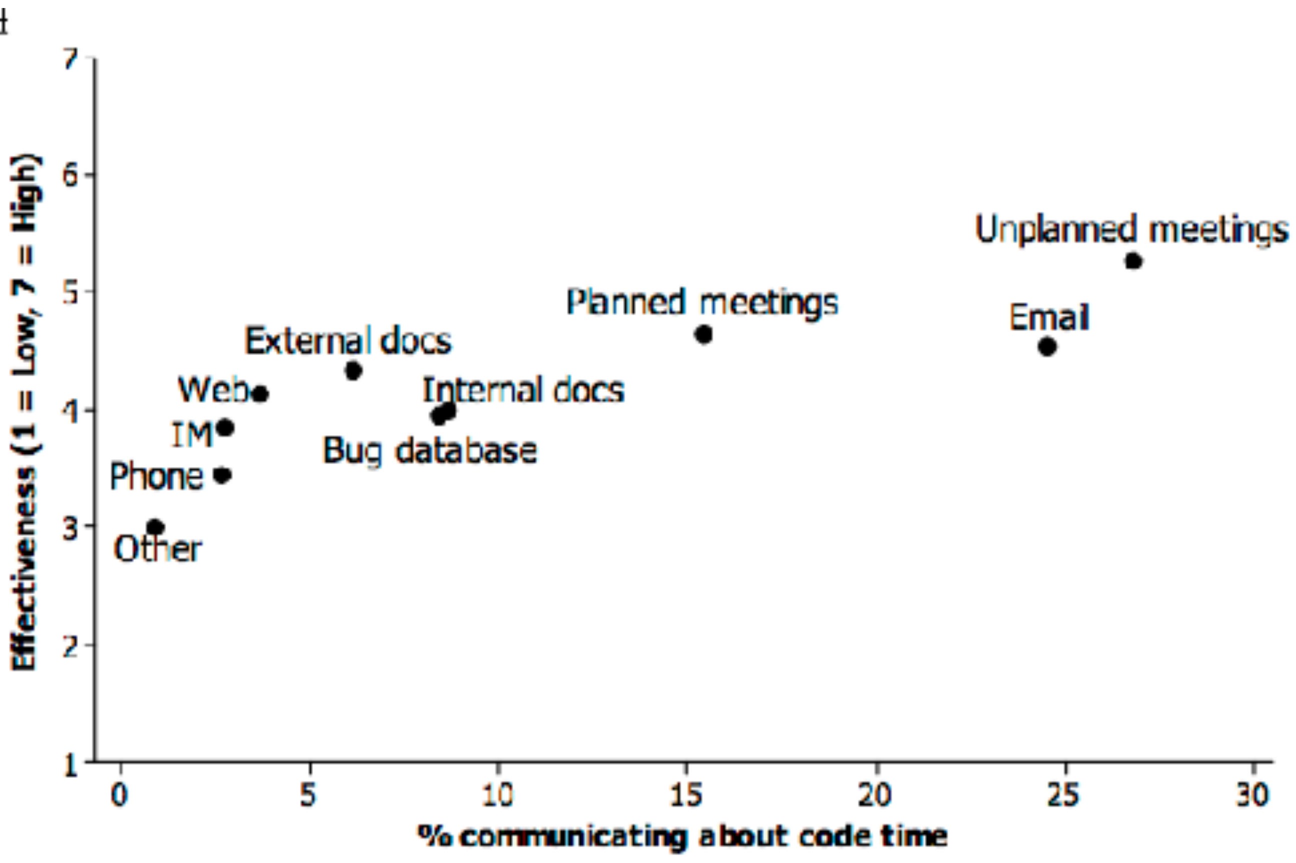
LaToza, Venolia, and DeLine. Maintaining Mental Models: A Study of Developer Work Habits. ICSE 2006.

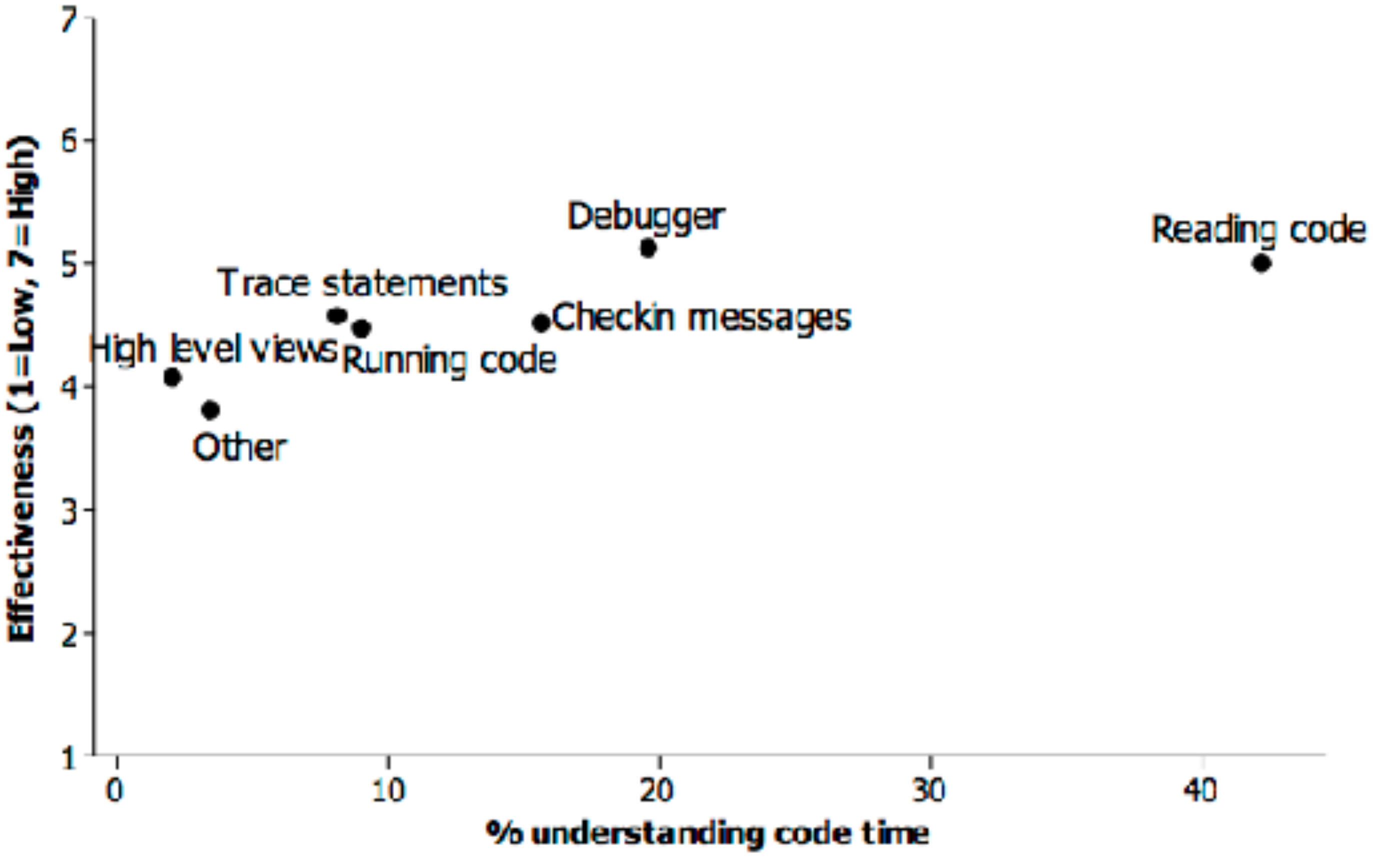
A few hours in the life of a professional software developer

collaboration	Developer assigned bug by team
programming	Reproduces error
	Browser hits error message (500 internal error)
	Attaches debugger
	Browse to page again, hit null reference exception
	Hypothesize from call stack which function might be responsible
	Browse through code
	Uses debugger to change values & experiment
	Make change, recompile, check, doesn't work
	Navigates slice, wrong values came from objects
	In complicated code doesn't understand
collaboration	Walks to B's office and asks where data comes from
	B working on high profile feature in area
programming	Tries to make change, still doesn't work
collaboration	Walks back to B, realize related to C's feature, C at lunch
	After lunch, A and B walk to C's office,
design	A, B, C change design to work with new feature
collaboration	Bug passed from A to C to change feature

LaToza, Venolia, and DeLine. Maintaining Mental Models: A Study of Developer Work Habits. ICSE 2006.

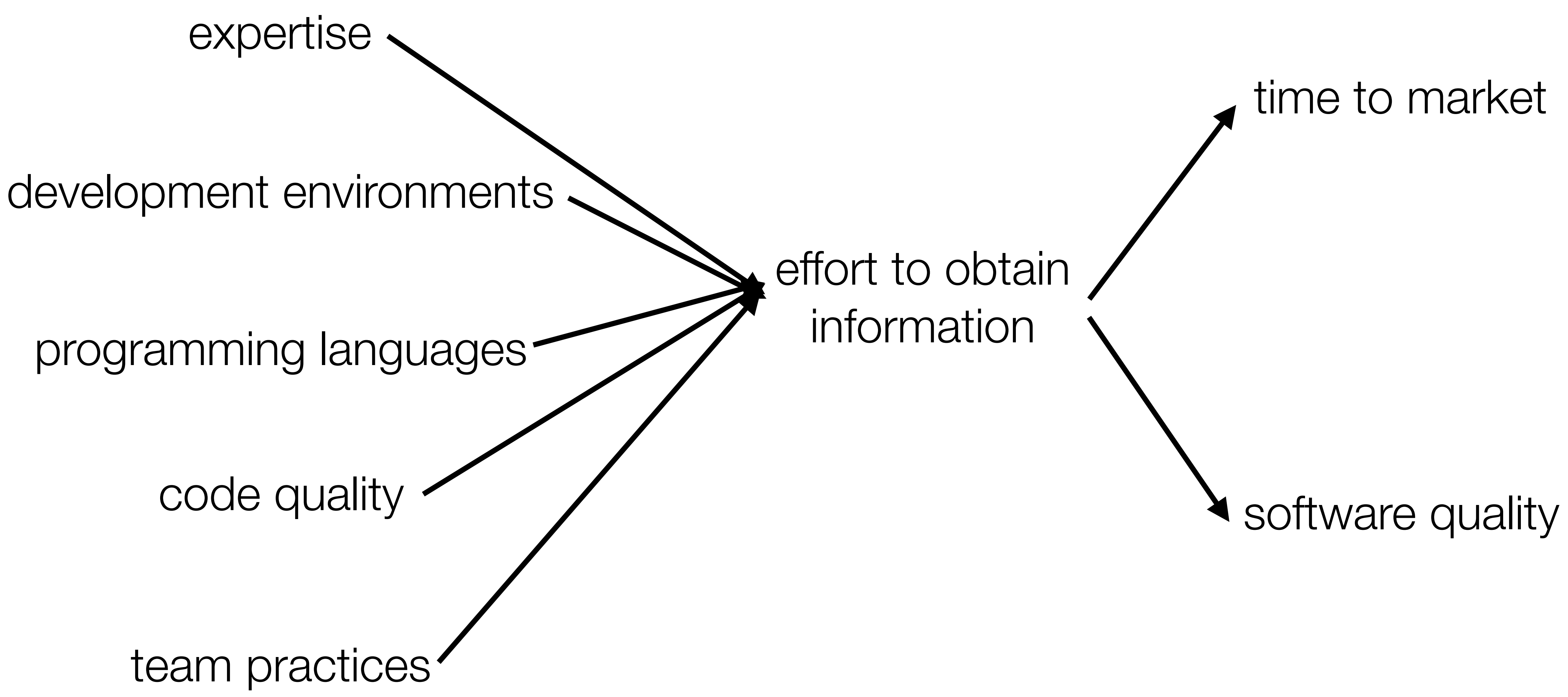
Developers use a variety of techniques for obtaining information about code





LaToza, Venolia, and DeLine. Maintaining Mental Models: A Study of Developer Work Habits. ICSE 2006.

Productivity in programming



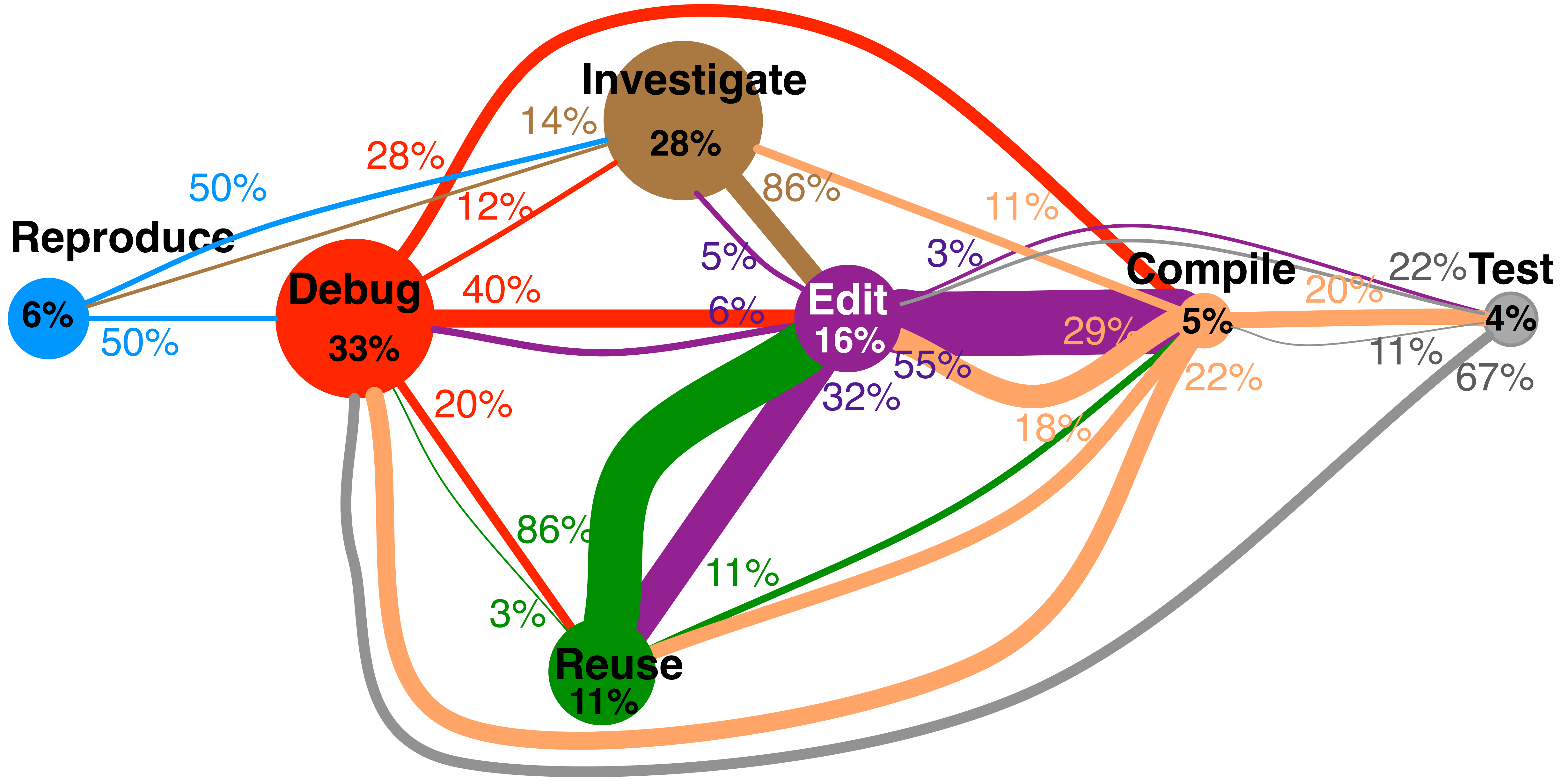
Models of Programming Work

- Programming activities
  - Steps needed to complete programming task
- Asking and answering questions
  - Formulating & testing hypotheses
  - Selecting strategies
- Structural relationship traversal
  - Information foraging

Programming Activities

- Goal of programming is to perform tasks
  - Implement features, fix defects, etc.
- Programming is a sequence of steps developers must do to accomplish these tasks
  - Design change, find points in code to change, determine implications of change, …

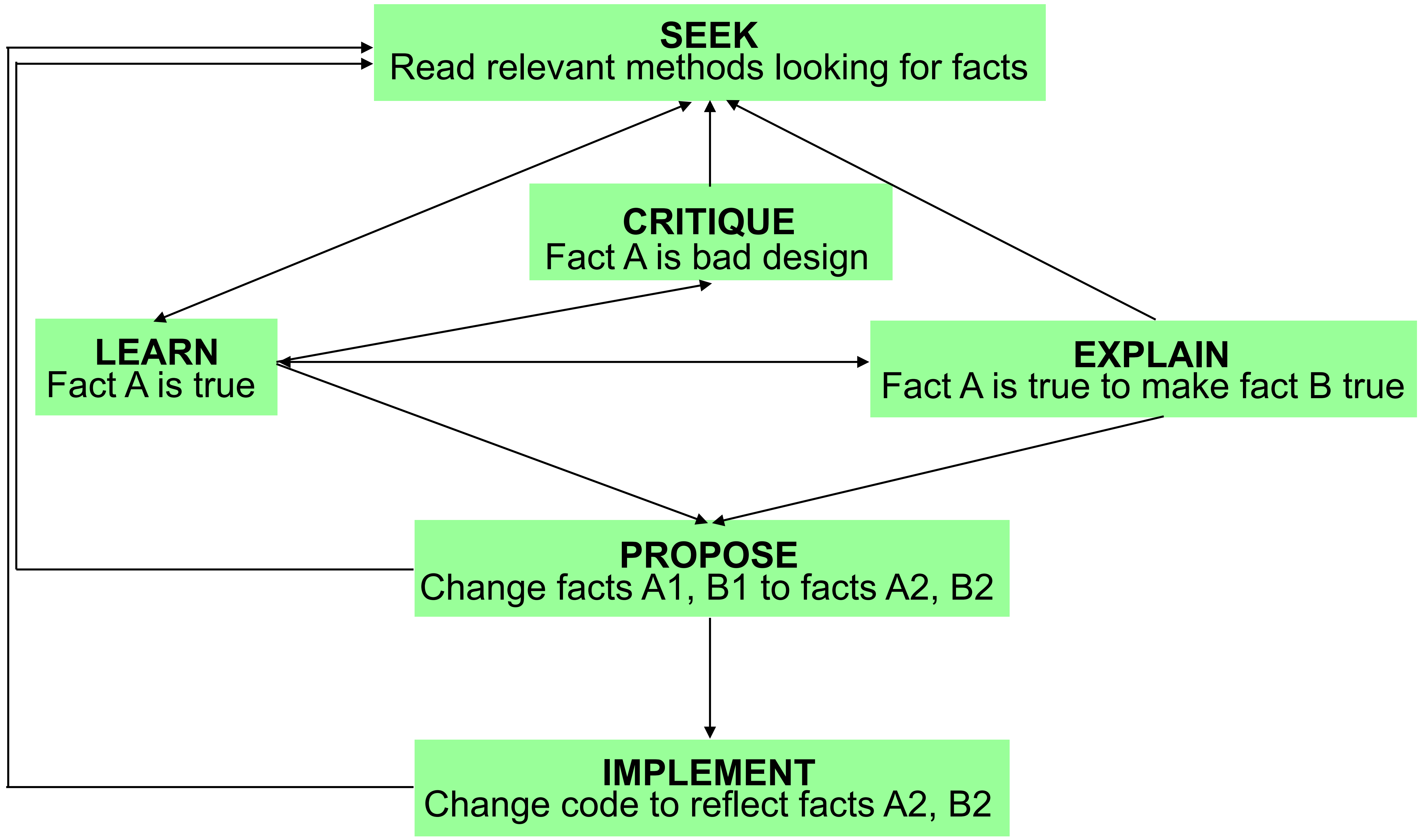
Edit / Debug Cycle



Circle size: % of time **Edge thickness:** % of transitions observed

For tasks in code in your own codebase that you haven't seen recently

Program comprehension as fact finding


SEEK
Read relevant methods looking for facts
CRITIQUE
Fact A is bad design
LEARN
Fact A is true
EXPLAIN
Fact A is true to make fact B true
PROPOSE
Change facts A1, B1 to facts A2, B2
IMPLEMENT
Change code to reflect facts A2, B2

LaToza, Garlan, Herbsleb, Myers. Program comprehension as fact finding. FSE 07.

Asking and Answering Questions

- Developers ask questions in order to complete programming tasks
  - e.g., What's the best design for implementing this?
- Developers may formulate **hypotheses** representing answers to questions.
- Developers select **strategies** to gather evidence to support or reject hypotheses.
  - From code, from external resources, from teammates
- Developers often have multiple strategies to answer questions
- Developers may have different levels of certainty about how strongly they believe the answers they have found.

Questions are recursive

task Investigate and fix a design problem

question Why is an event being issued by forcing a cache update?

How is BufferHandler using its buffer field? Are there any other mutations on it?

action Read methods of BufferHandler

Why is there a buffer member variable that is never used?

Investigate references to BufferHandler.buffer

Why is doDelayedUpdate() a member of BufferHandler?

Reads methods along path, concludes that BufferHandler tracks update delays

Why wouldn't isFoldStart() call getFoldLevel()

Reads isFoldStart(), getFoldAtLine()
Concludes isFoldStart() doesn't call because of short circuit evaluation

Implement fix

Assure correctness

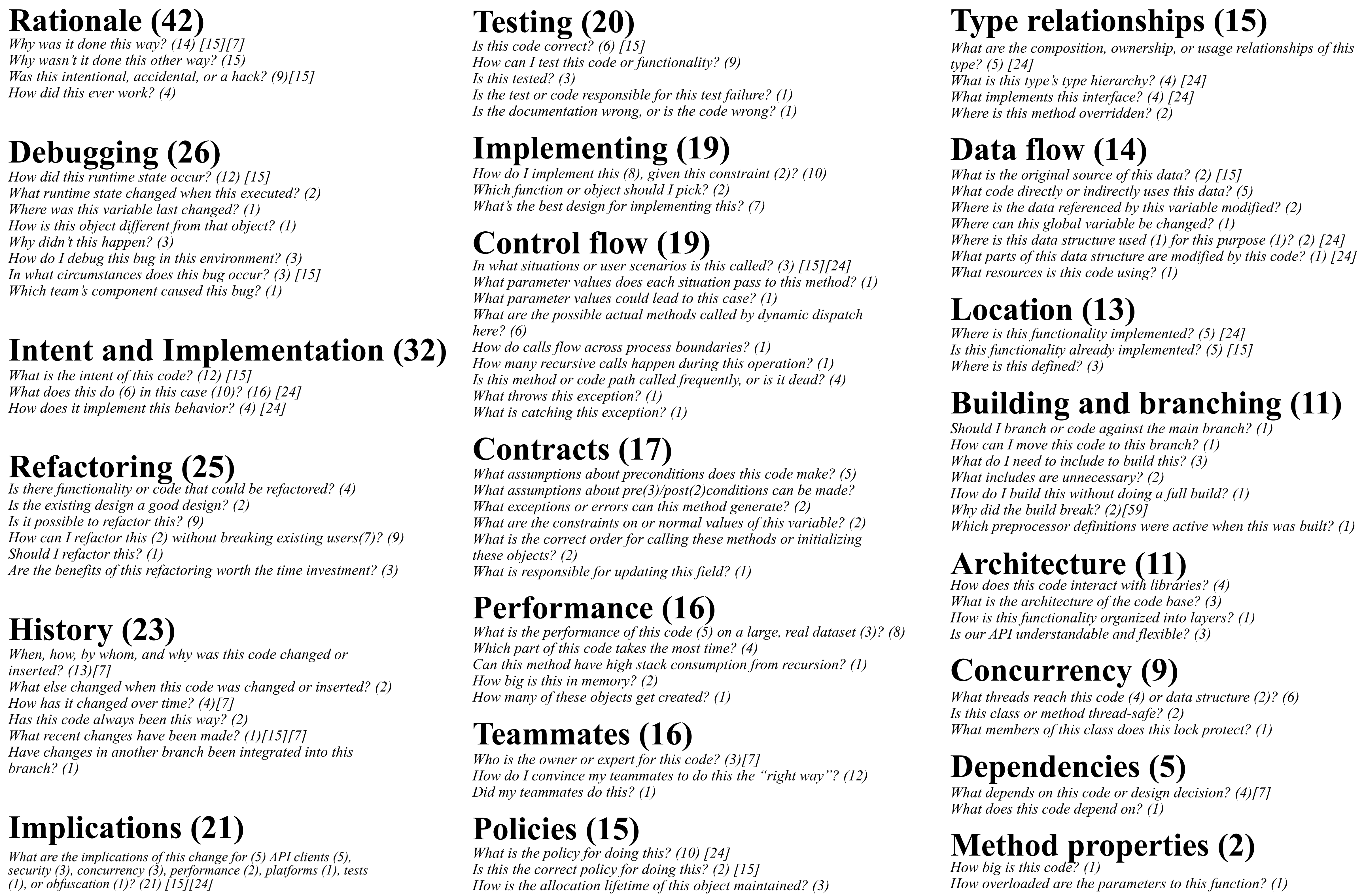
Set conditional break point
Check that jEdit still appears to work correctly
Repro original bug by reinserting

IDE

LaToza and Myers. Designing useful tools for developers. PLATEAU 2011.

Questions span many topics

What hard to answer questions about code have you recently asked?



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)

LaToza and Myers. Hard-to-answer questions about code. PLATEAU 2010.

Types of questions

- Questions about changes
- Questions about elements
- Questions about relationships between elements

Questions about Changes - Rationale

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

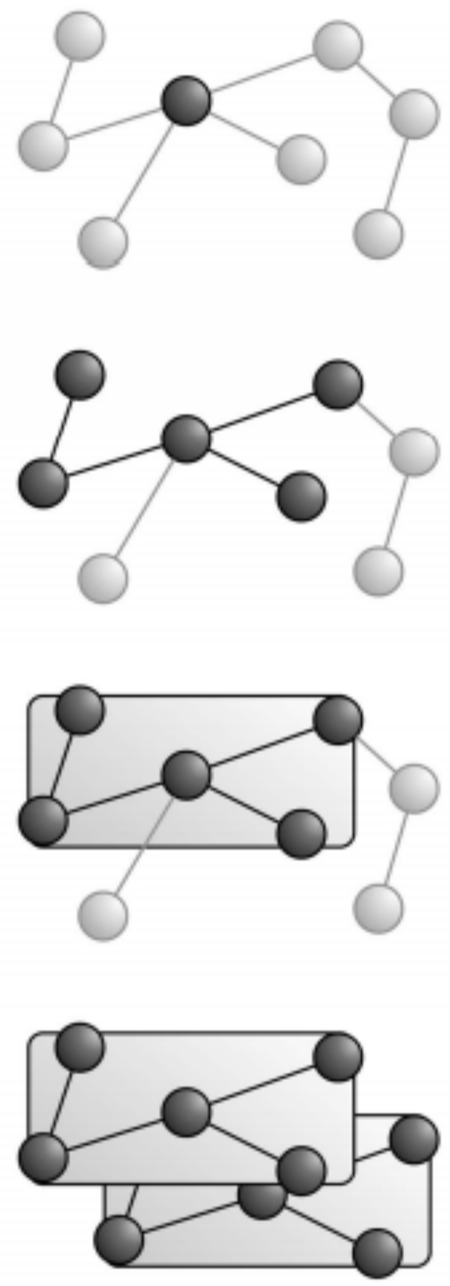
Questions about elements

- Where is this functionality implemented? (5)
  Is this functionality already implemented? (5)
  Where is this defined? (3)

Questions about element relationships - Data flow

- What is the original source of this data? (2)
  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)
  What parts of this data structure are modified by this code? (1)
  What resources is this code using? (1)

Graphs of Elements



Finding focus points

5 kinds of questions.

For example: Which type represents this domain concept?

Expanding focus points

15 kinds of questions.

For example: Which types is this type a part of?

Understanding a subgraph

13 kinds of questions.

For example: What is the behavior these types provide together?

Questions over groups of subgraphs

11 kinds of questions.

For example: What is the mapping between these UI types and these model types?

Sillito, Murphy, De Volder. Asking and answering questions during a programming change task. TSE 08.

Structural Relationship Traversal

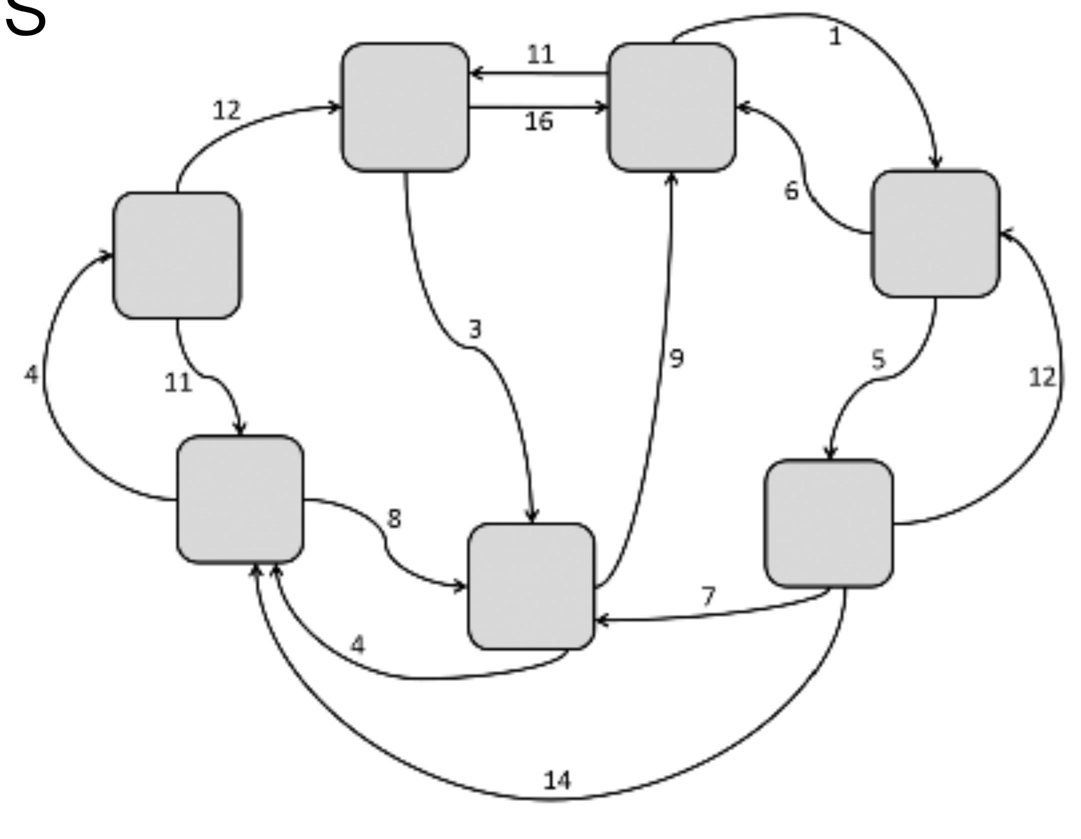
- Developer is currently viewing an element in code
    - e.g,. class, method, statement, field reference
- Developers wishes to navigate to a **related** method
    - By reference, call, data dependency, …
- How do developers make navigation decisions?

Information foraging

- Mathematical model describing navigation
- Analogy: animals foraging for food
  - Can forage in different patches (locations)
  - Goal is to maximize chances of finding **prey** while minimizing time spent in hunt
- Information foraging: navigating through an information space (patches) in order to maximize chances of finding prey (information) in minimal time

Information environment

- Information environment represented as **topology**
  - Information patches connected by traversable **links**
  - For SE, usually modeled as call graphs
    - methods are nodes and function invocations are edges



Traversing links

- Links - connection between patch offered by the information environment
- Cues - information features associated with outgoing links from patch
  - E.g., text label on a hyperlink
- User must choose which, of all possible links to traverse, has best chance of reaching prey

Scent

- User interprets cues on links by likelihood they will reach prey
  - e.g., do I think that the “invoke” method is likely to implement the functionality I’m looking for?

Simplified mathematical model

- Users make choices to maximize **possibility** of reaching prey per cost of interaction
- Predators (idealized) choice = max [V / C]
    - V - value of information gain, C - cost of interaction
- Don’t usually know ground truth, have to estimate
- Predator’s desired choice = max [E[V] / E[C]]

Models of Programming Work

- Programming activities
  - Steps needed to complete programming task
- Asking and answering questions
  - Formulating & testing hypotheses
  - Selecting strategies
- Structural relationship traversal
  - Information foraging

Productivity Mechanisms

- Lots of models of programming
- How do these help in the design of programming tools?

Challenge Productivity Model

- Developers experience challenges (e.g., answering questions) in their programming tasks
- Tools help developers be more productive by reducing the time to answer questions, increasing likelihood of success

- This requires
  - understanding **precisely** the information required and context available to developers
  - insight into a **mechanism** to make a question easier to answer

Supporting information needs

- Debugging is hard.
  - Tool **x** claims to make debugging easier!
- Does tool **x** help?
- Depends…
  - Does tool **x** apply in the situations that make debugging challenging?
  - Do developers have the context they need to invoke tool **x**
  - Does tool **x** reliably produce the information required
  - Are the interactions for using tool x usable

Debugging (26)

* *How did this **runtime state** occur? (12)*
  data, memory corruption, race conditions, hangs, crashes, failed API calls, test failures, null pointers

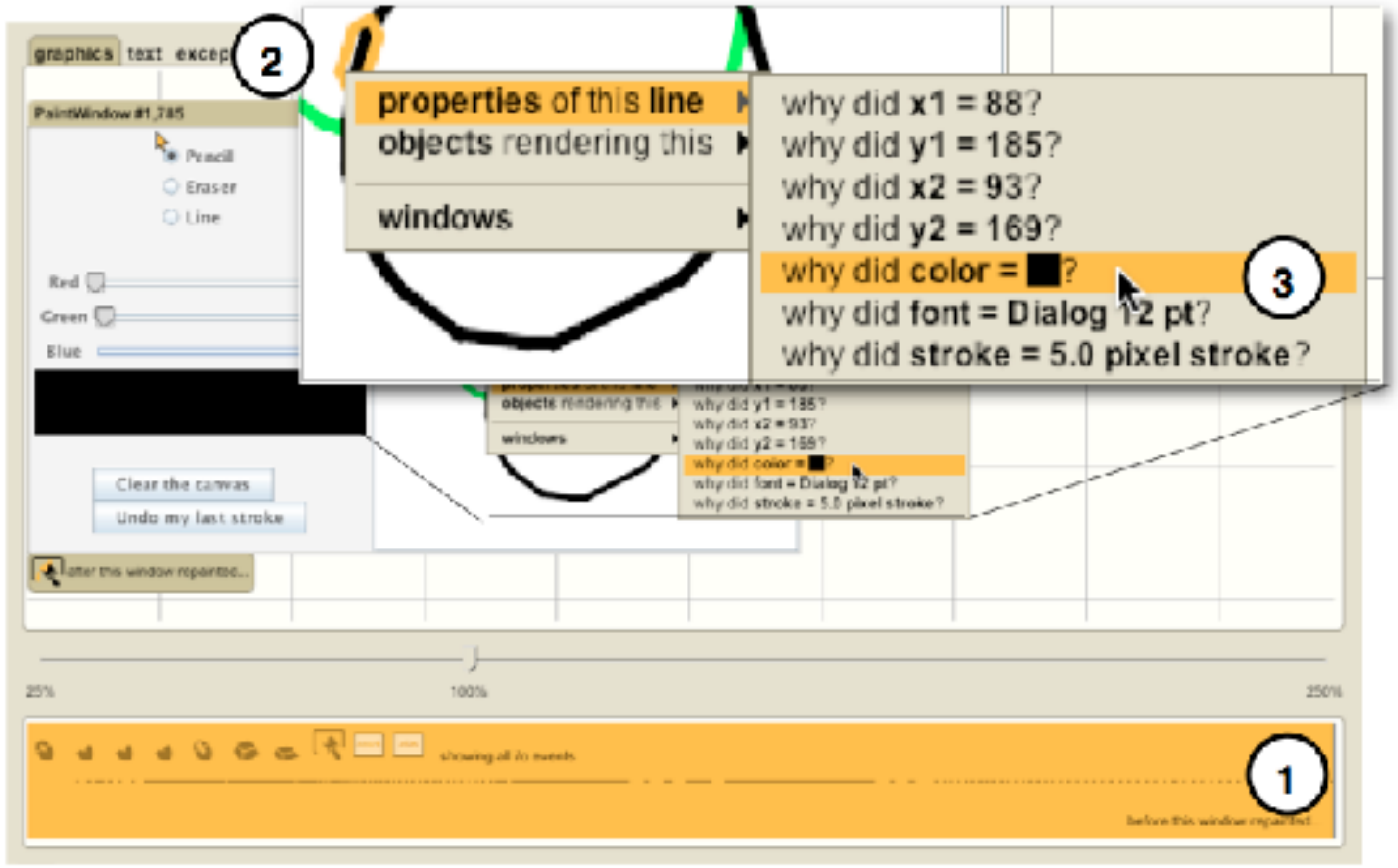
* *Where was this **variable** last changed? (1)*
* *Why **didn't** this happen? (3)*

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)*

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)*

Refactoring (25)

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

smell detectors [1], metrics

Look for bad design idioms
Suggests developer refactor

data clumps	instanceof
feature envy	magic number
refused bequest	long method
typecast	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

Productivity mechanisms

- What do tools have in common?
  - Are there common **mechanisms** that tools share in how they are intended to make developers more productive?

Productivity Mechanisms in Programming

- **Information minimization**
  - Processing information about code takes time.
  - Reducing information to process reduces time, increases productivity
- **Feedback**
  - Edits to code have implications.
  - More information about implications increases quality.
  - Faster information reduces rework time, increasing productivity.

Information minimization: Identifying task-relevant information

- What's the minimal amount of **relevant** information for a programming task?
- "Proto" theories exist in software engineering.
- **Modularity** posits that relevant information is related code that can be placed nearby
    - Information hiding (OO), coupling / cohesion, module systems, aspects, refactoring
- **Traversal tools** posit that relevant code is structurally connected through dependencies
    - Impact analysis, slicers, debuggers
- **Recommenders** posit that past co-changes predict future co-changes across modules.
    - Concern management, method recommenders
- How relevant is information provided —> productivity benefits of approach