Problem Solving

SWE 795, Fall 2019

Software Engineering Environments

Today

- Part 1 (Lecture)(~75 mins)
- Break!

- Part 2 (Discussion)(45 mins)
  - Discussion of readings

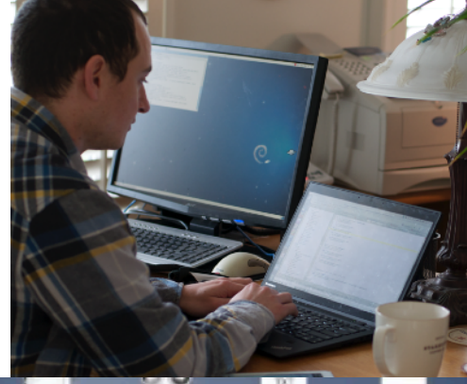
- Part 3 (Project group work)(30 mins)
  - Time to work in groups, ask questions

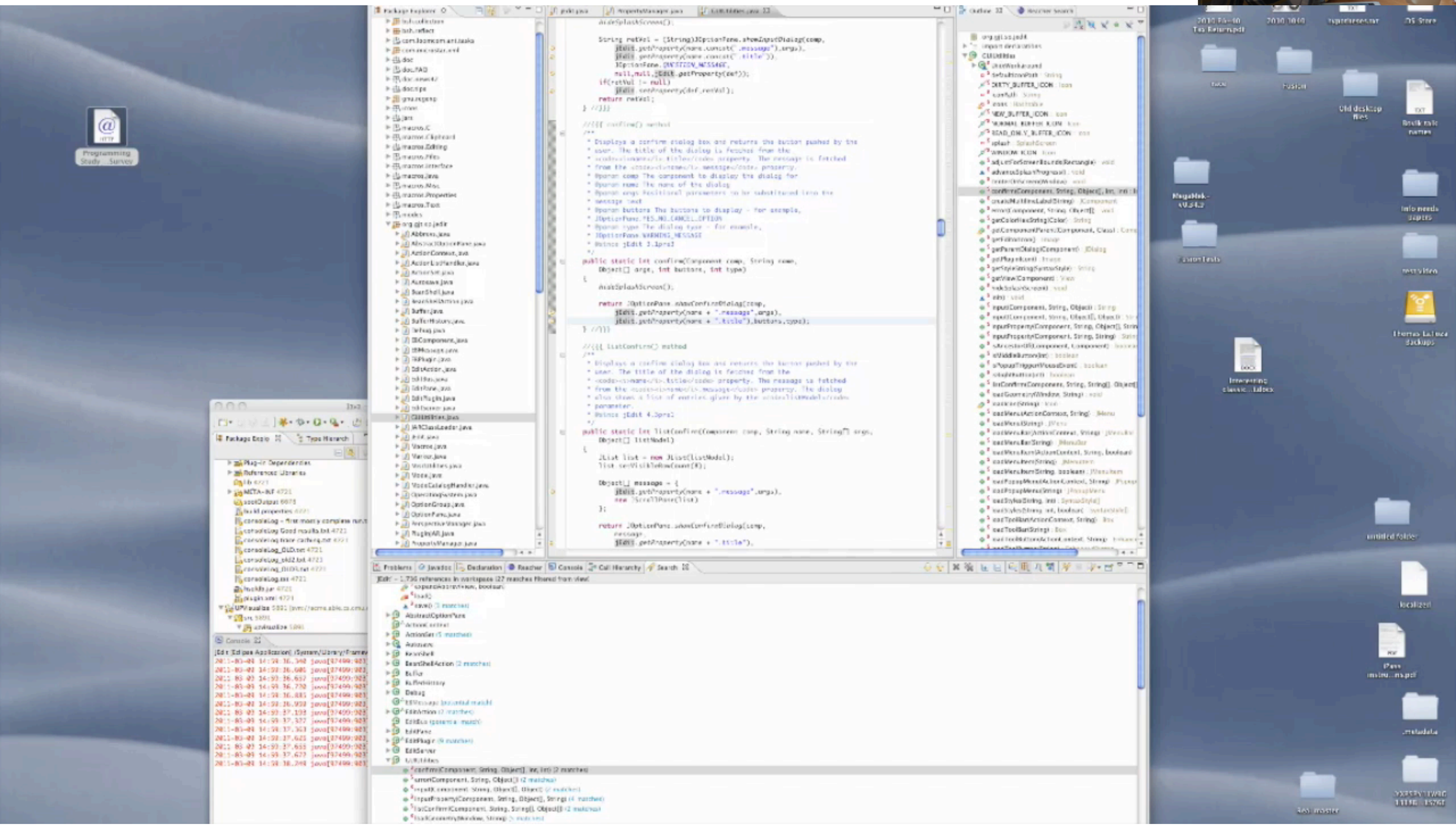
Logistics

- HW1 due next week

- No readings next week, will have project presentations instead

A few minutes in the life of a developer





A few hours in the life of a professional software developer

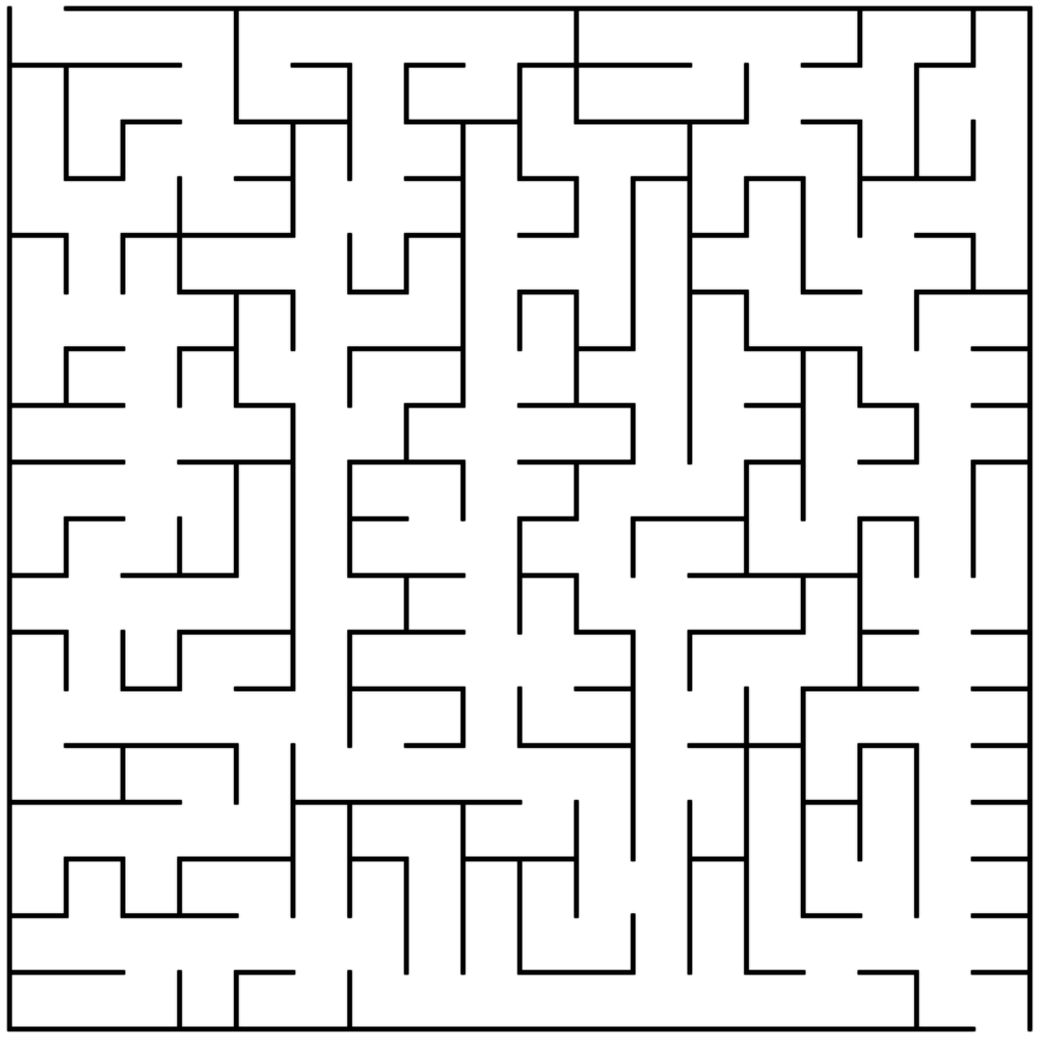
collaboration	Developer assigned bug by team
programming	Reproduces error
	Browser hits error message (500 internal error)
	Attaches debugger
	Browse to page again, hit null reference exception
	Hypothesize from call stack which function might be responsible
	Browse through code
	Uses debugger to change 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.

Problem solving

Goal: where am I trying to go?
Operators: what actions can I take to get closer to the goal?

Apply operator, look at new state, apply another operator



Newell, A.; Shaw, J.C.; Simon, H.A. (1959). Report on a general problem-solving program. *Proceedings of the International Conference on Information Processing.* pp. 256–264.

Problem solving is 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

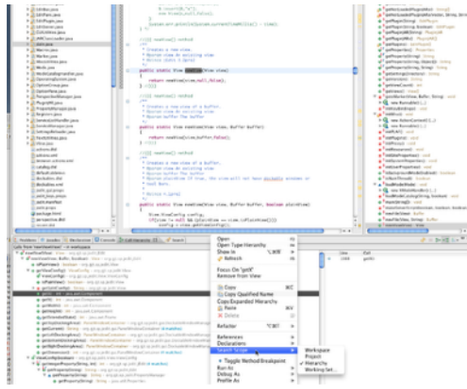
Why wouldn't isFoldStart() call getFoldLevel()

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

Implement fix

Assure correctness

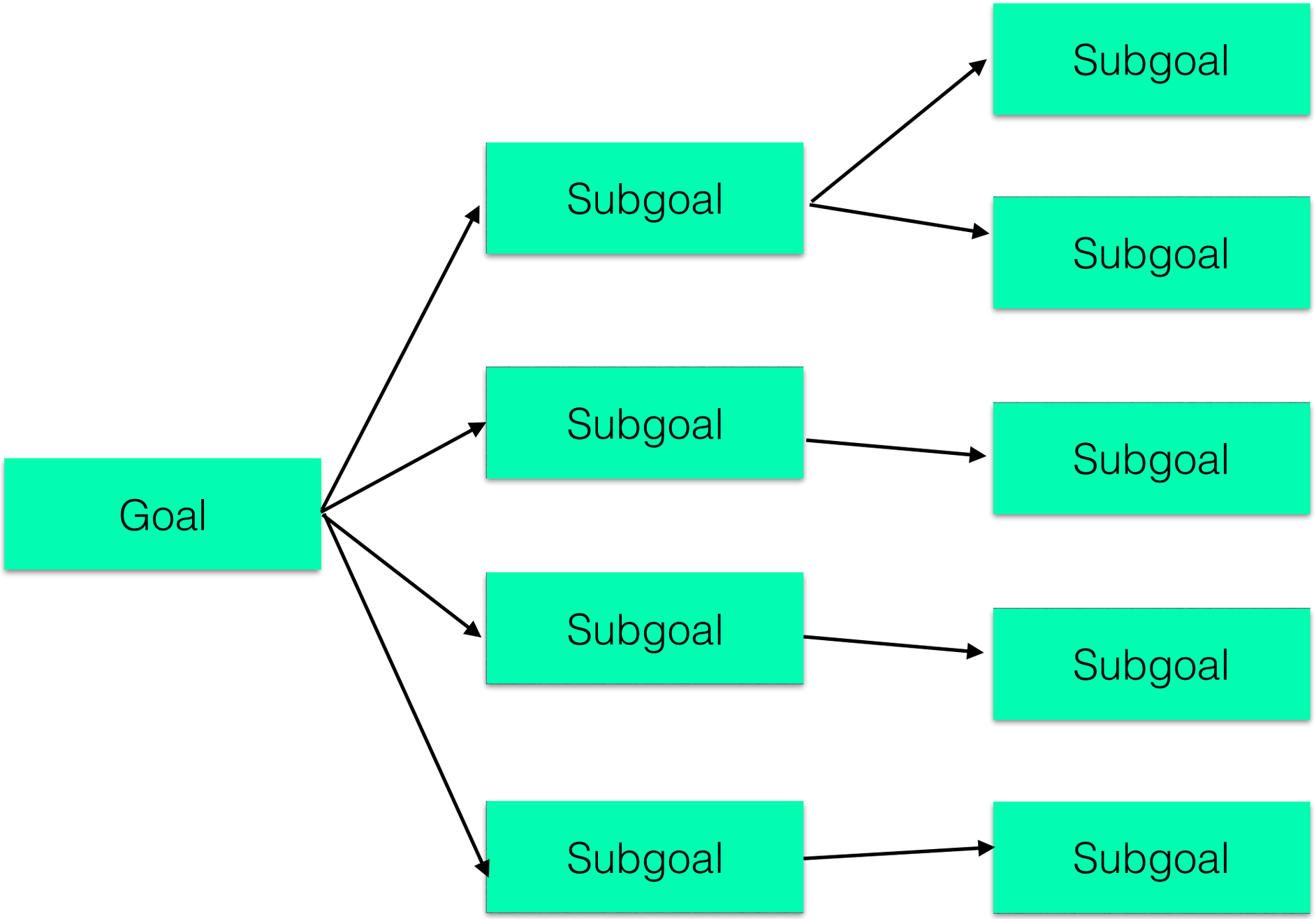
Set conditional break point
Check that jEdit still appears to work correctly
Repro original bug by reinserting



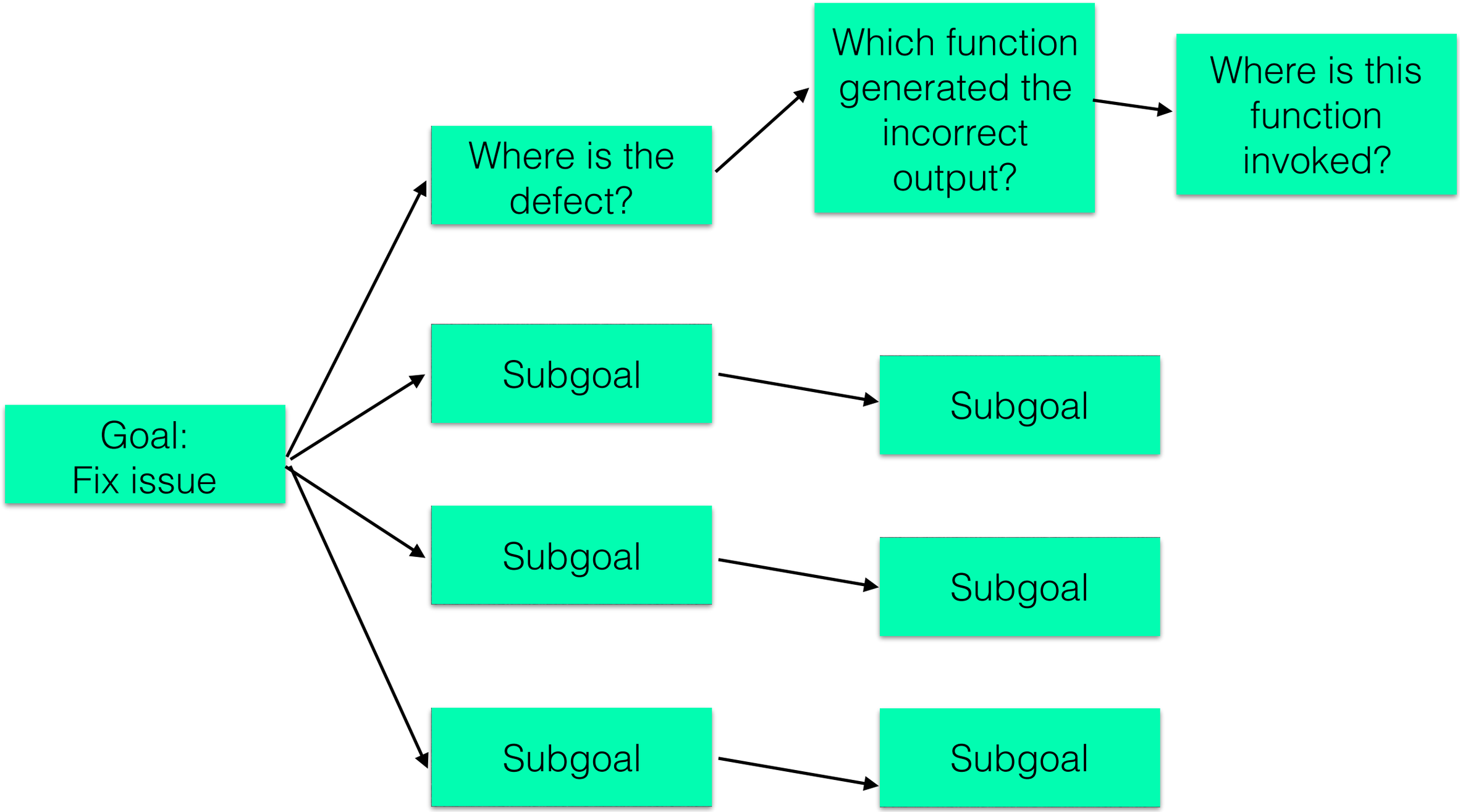
IDE

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

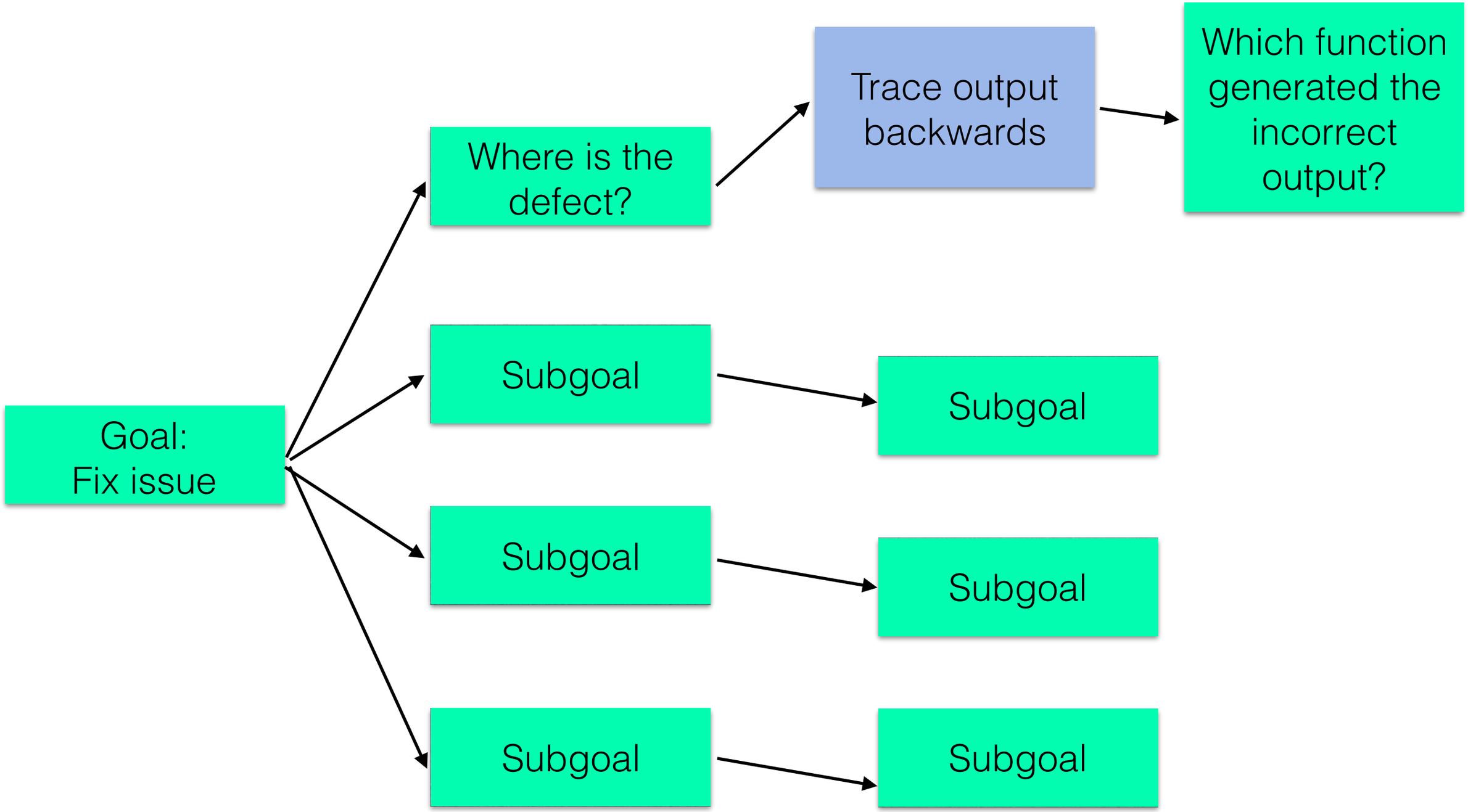
Problem solving is recursive



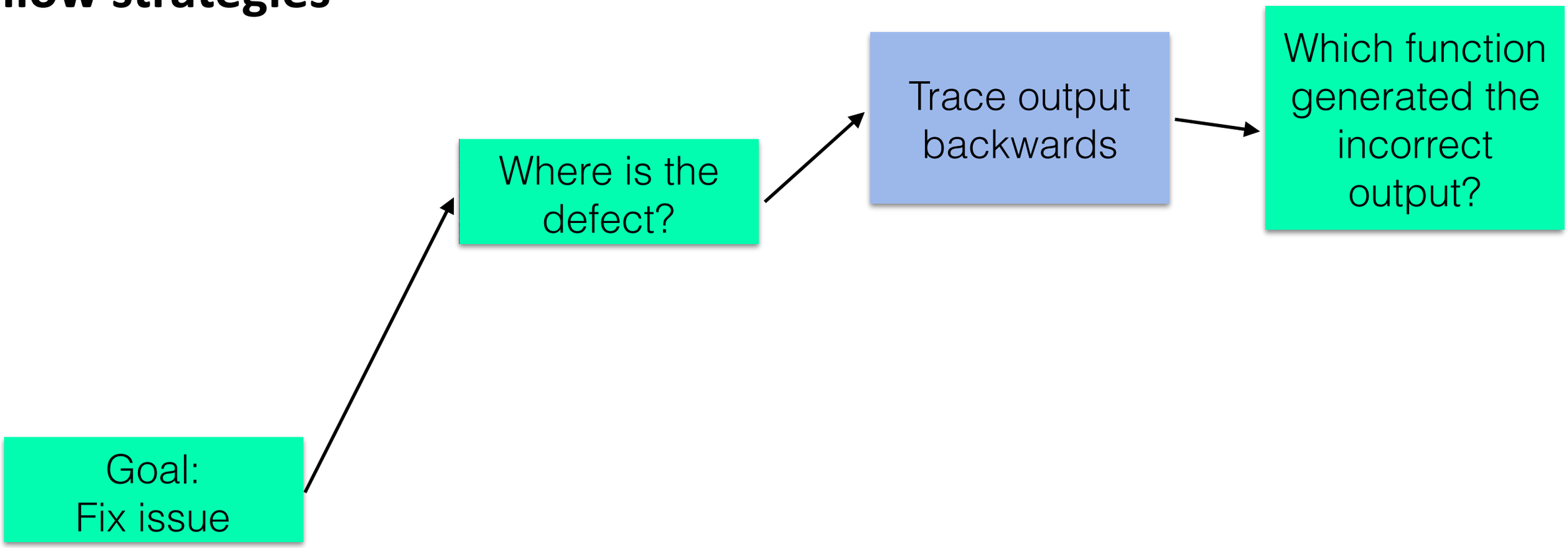
Problem solving involves answering questions



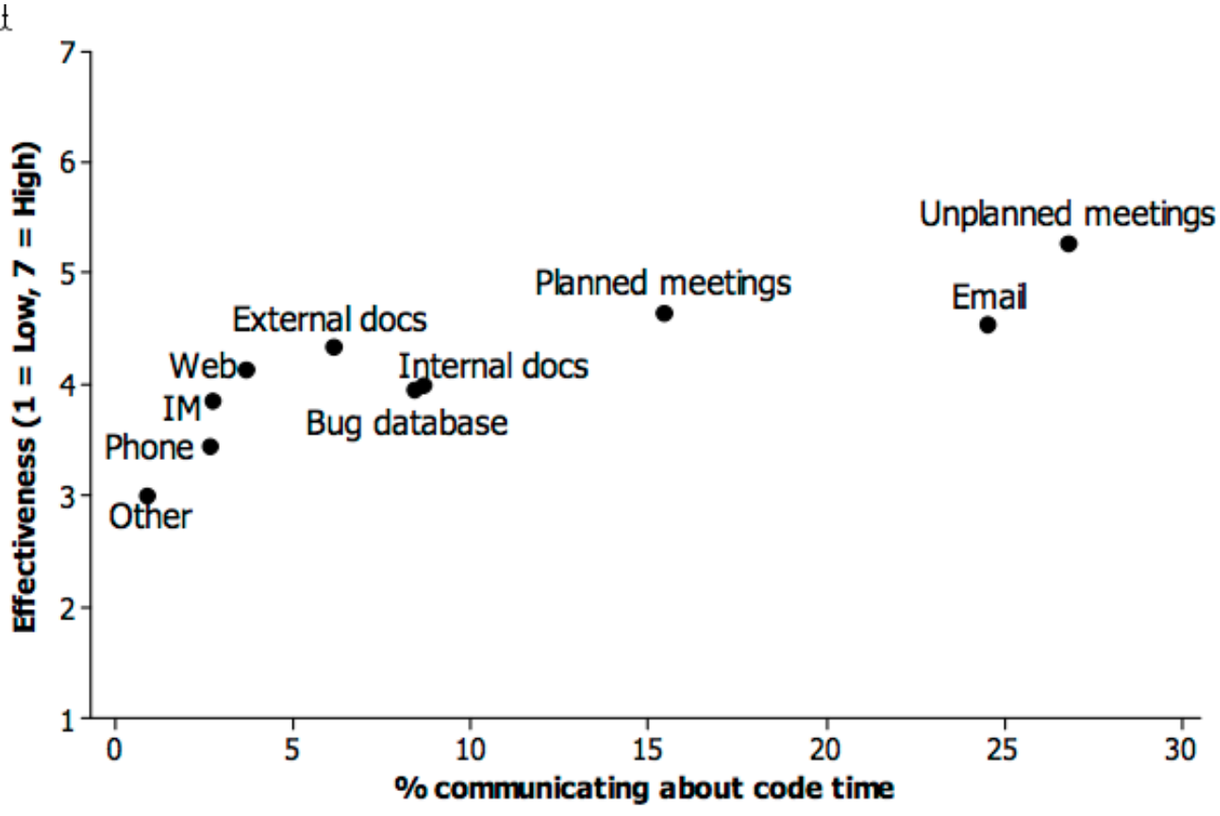
Problem solving involves strategies

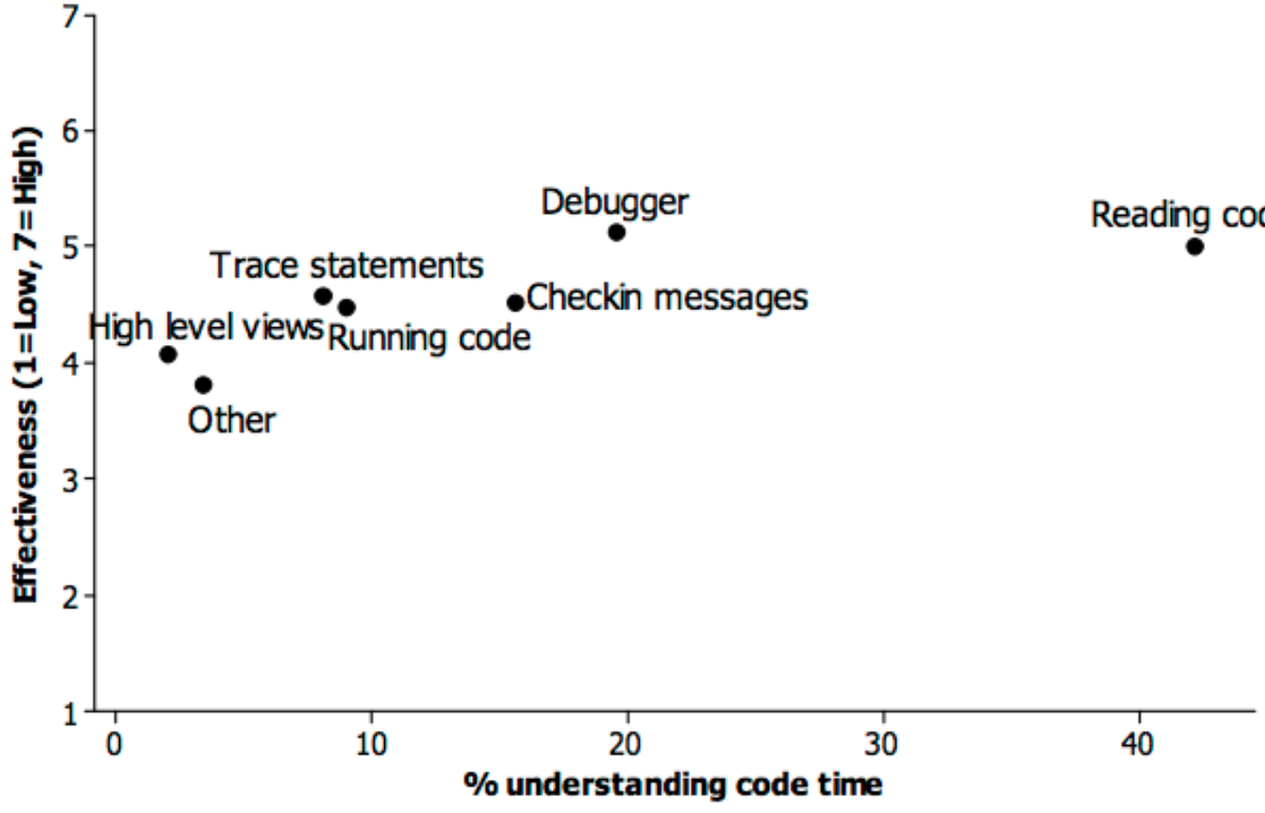


Problem solving involves taking actions to answer questions and follow strategies



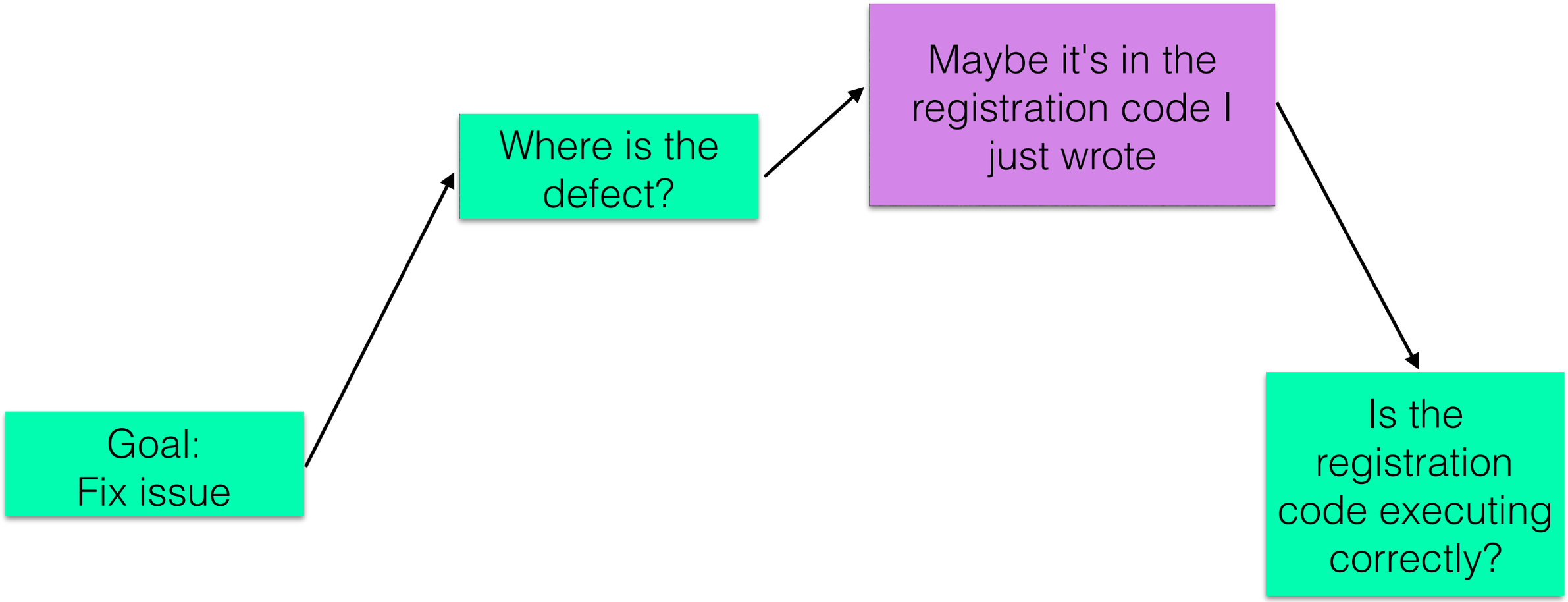
Developers use a variety of techniques for obtaining information and answering questions



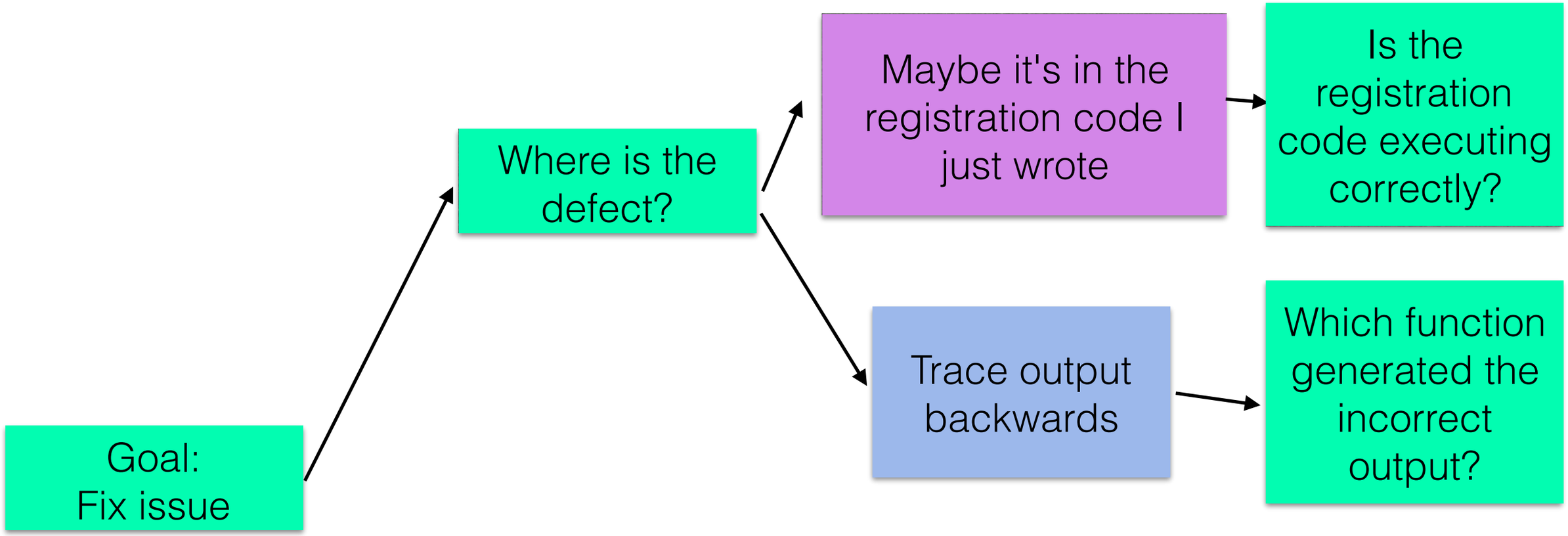


LaToza, Venolia, and DeLine. Maintaining Mental Models: A Study of Developer Work Habits. ICSE 2006.

Problem solving involves formulating hypotheses



Problem solving involves choices between strategies



Problem solving in programming

- Developers have **tasks** (e.g,. fix this defect, implement this feature) which they work to complete

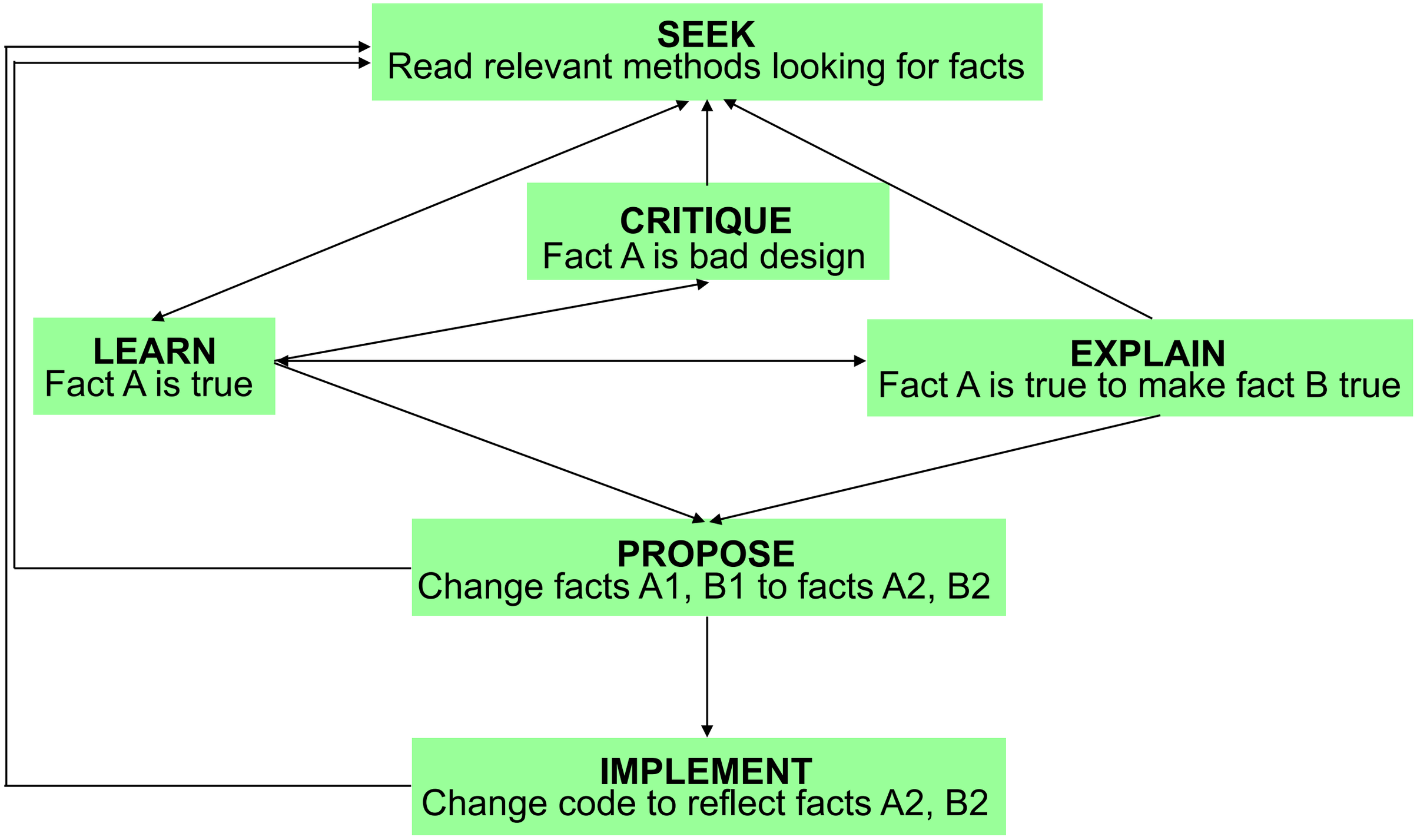
- Developers ask **questions** reflecting information they need in order to complete tasks.
  - e.g., What's the best design for implementing this?

- Developers may formulate **hypotheses** representing answers to questions.

- Developers select **strategies** to gather evidence answer questions and to support or reject hypotheses.
  - From code, from external resources, from teammates

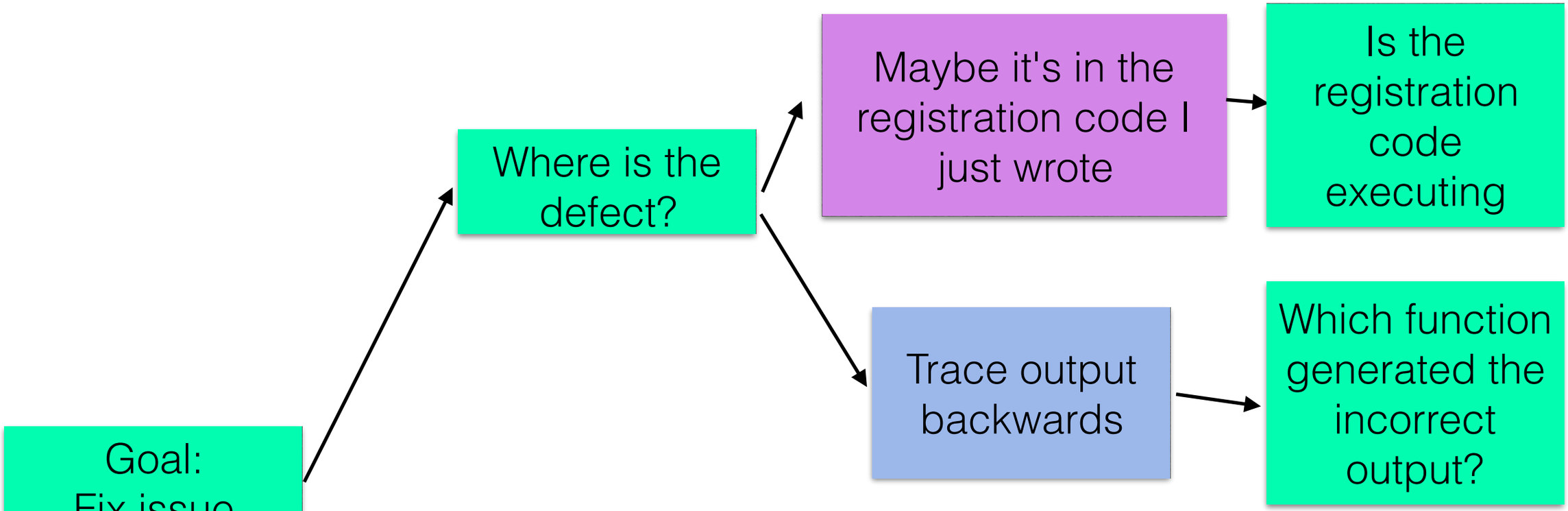
- Developers often have multiple strategies to answer questions

Program comprehension as fact finding



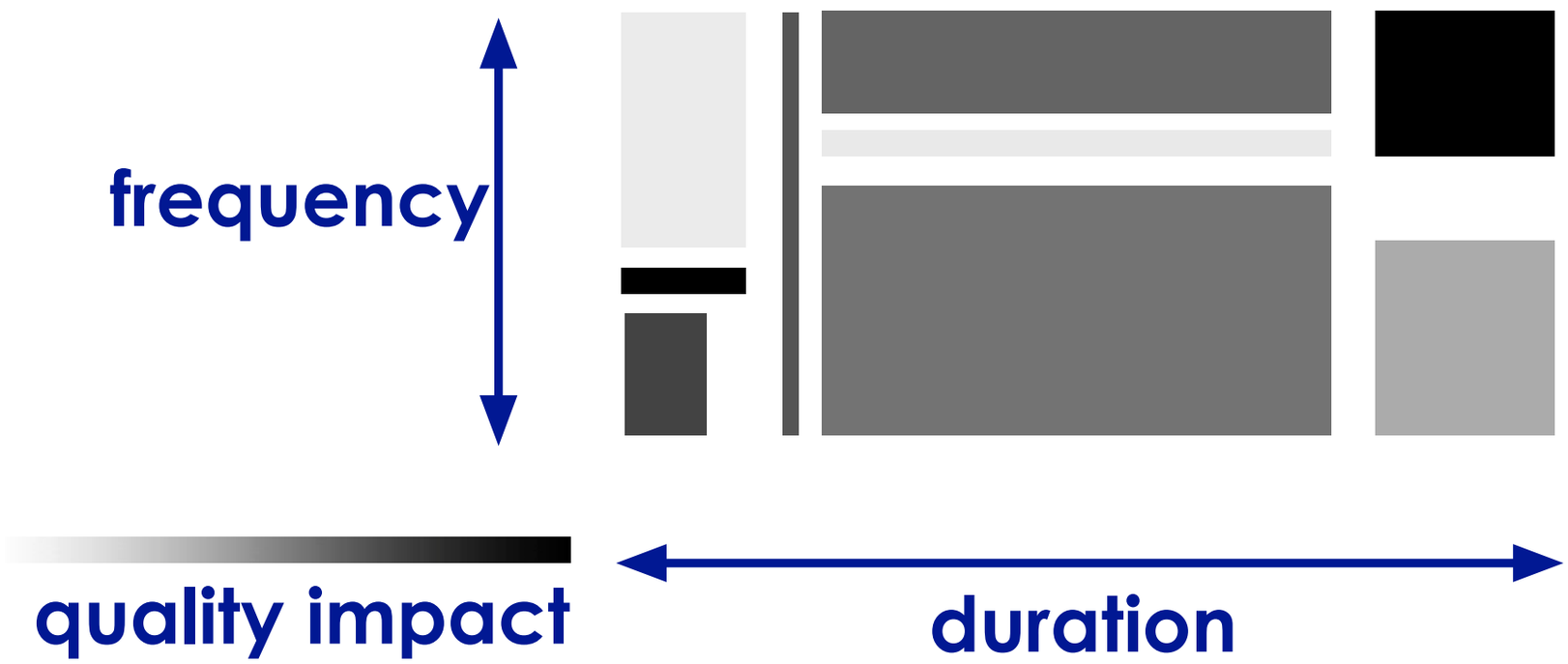
LaToza, Garlan, Herbsleb, Myers. Program comprehension as fact finding. FSE 07.

Supporting programming activities



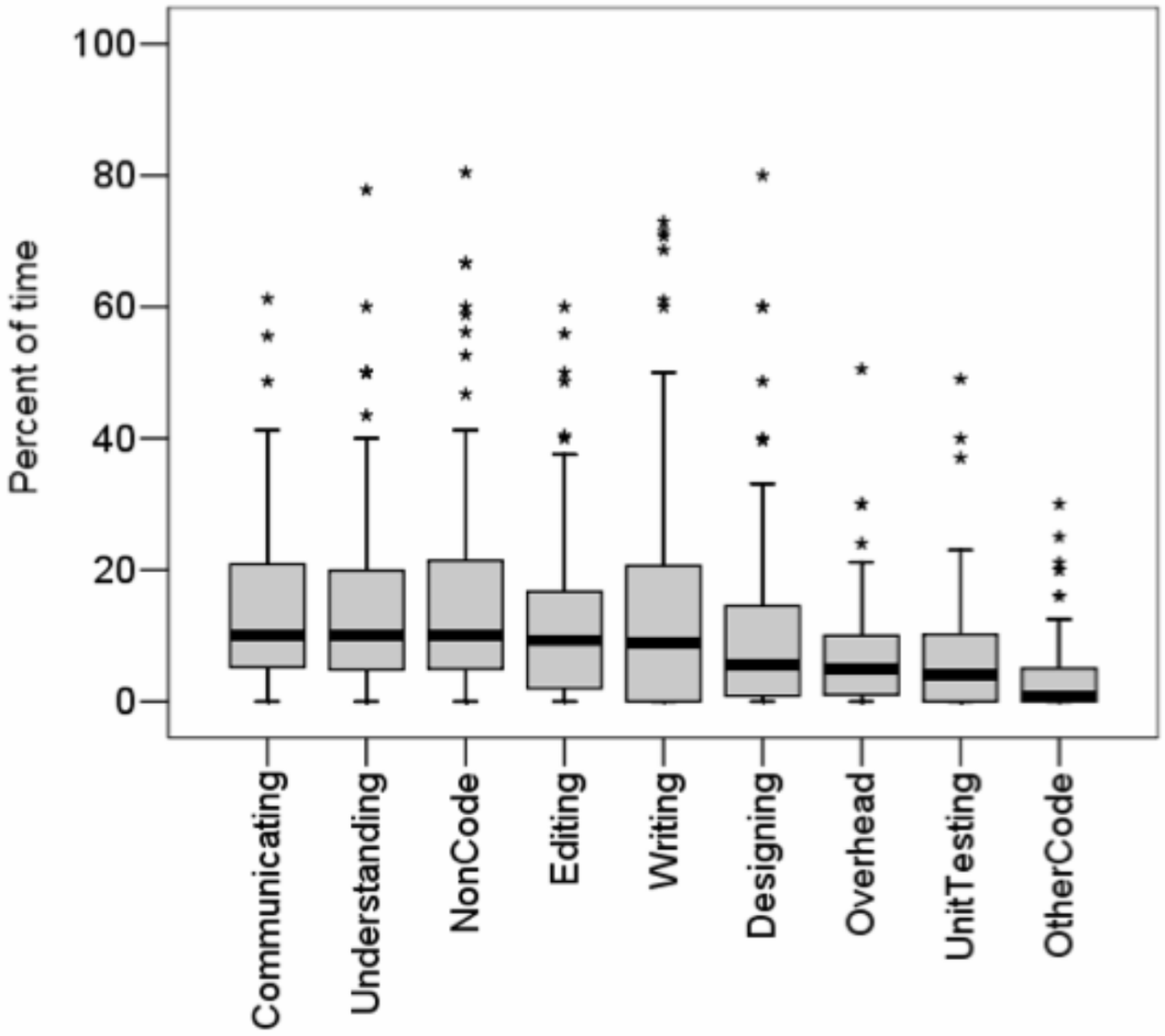
- Many potential points of intervention, supporting subgoals / strategies / question answering / testing hypotheses

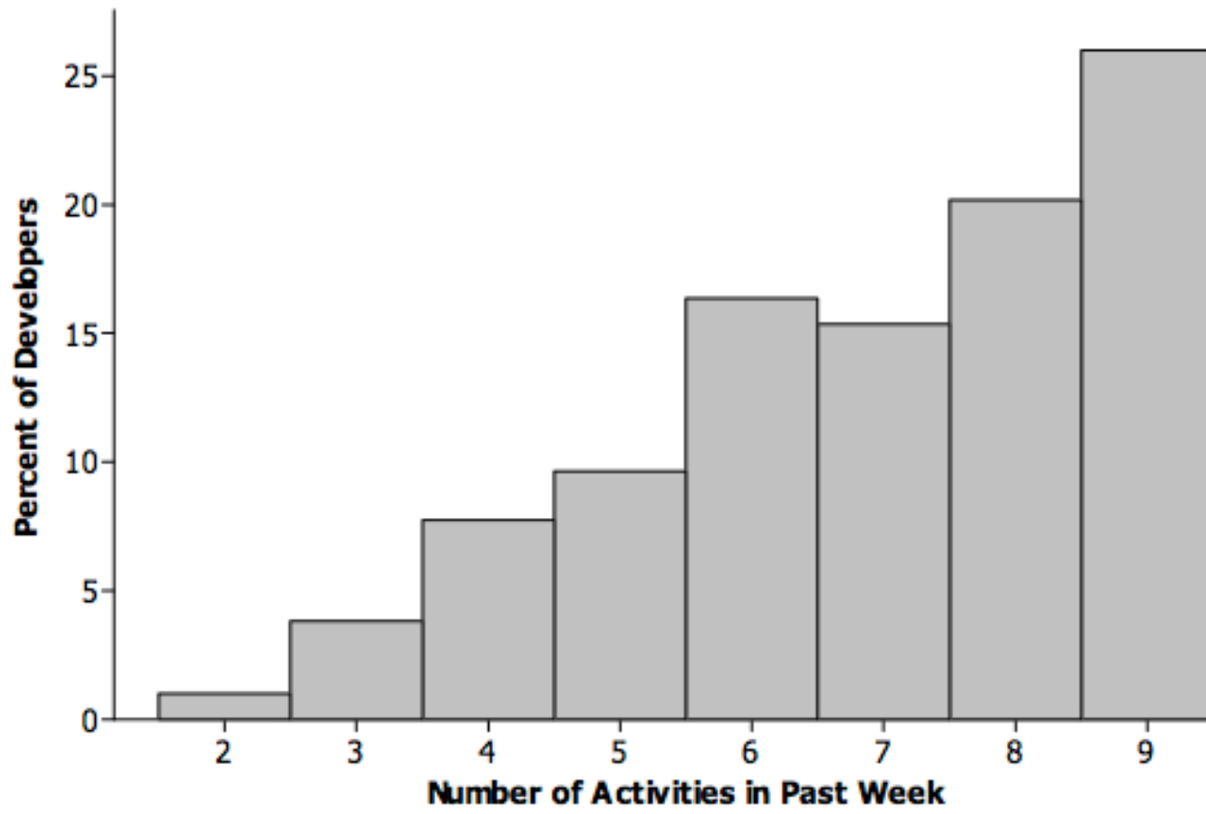
Useful interventions solve important problems



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

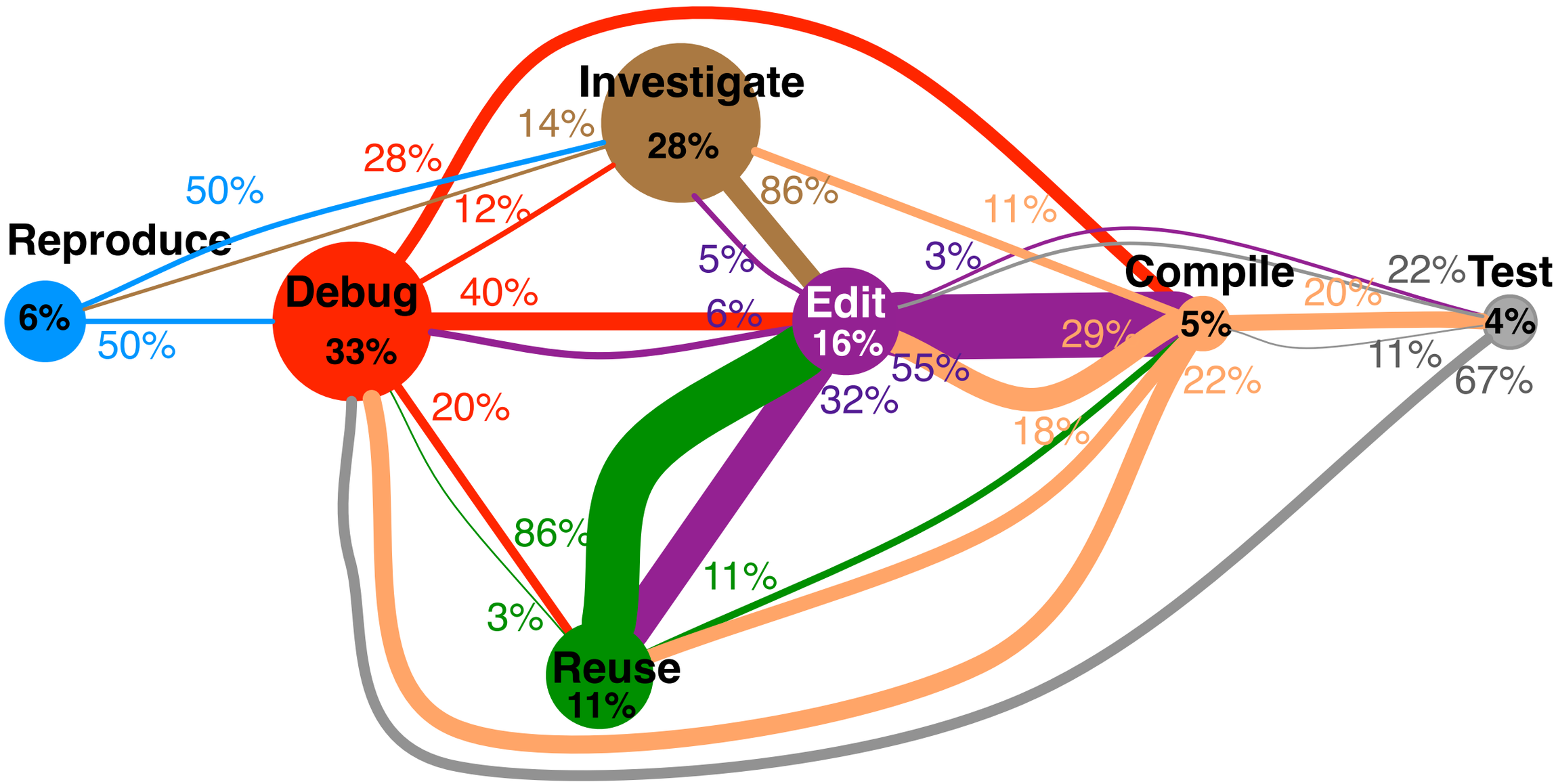
What percentage of the last week have you spent…





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

Example: Activities in fixing a defect



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

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

LaToza and Myers. Developers ask reachability questions. ICSE 2010.

Pseudo-initial *title, years on team*	Product *customers*	Work *months in phase*
A *SDE, 2*	Communication tool *end users and devs*	Investigated two bugs *12 in bug fixing*
B *SDE, 1*	Development tool *developers*	Fixed generated code *6 in development*
C *SDE, 3*	File metadata *developers*	Interpreted spec *6 in bug fixing*
D *SDE, 4*	Mobile device tool *end users*	Wrote internal tool *8 in development*
E *SDE, 2*	Service packs *developers*	Reproduced failures *2 in bug fixing*
F *SDE, <1*	Build automation *developers*	Diagnosed build failure *6 in development*
G *SDE, 1*	Mobile device tool *end users*	Prepared build config *6 in development*
H *SDE, 1*	Discussion boards *end users*	Read unfamiliar code *1 in bug fixing*
J *SDE, 4*	Communication tool *end users and devs*	Triaged 7 bug reports *6 in bug fixing*
K *Lead, 5*	Education tool *end users*	Wrote feature code 5 in development
L *SDE, 10*	Development tool *end users and devs*	Triaged w/ coworker *2 in bug fixing*
M *SDE, 3*	Input device UI *end users*	Investigated two bugs 2 in bug fixing
N *SDE, 4*	Data processing tool *end users*	Prepared check in *5 in development*
R *SDE, 3*	Communication tool *end users*	Triaged and fixed bugs *1 in bug fixing*
T *SDE, 1*	Office application *end users*	Investigated two bugs *5 in bug fixing*
U *SDE, 3*	Office application *end users*	Waited for build & test *12 in bug fixing*
V *Lead, < 1*	Content protection *end users and devs*	Debated fix with team *5 in bug fixing*

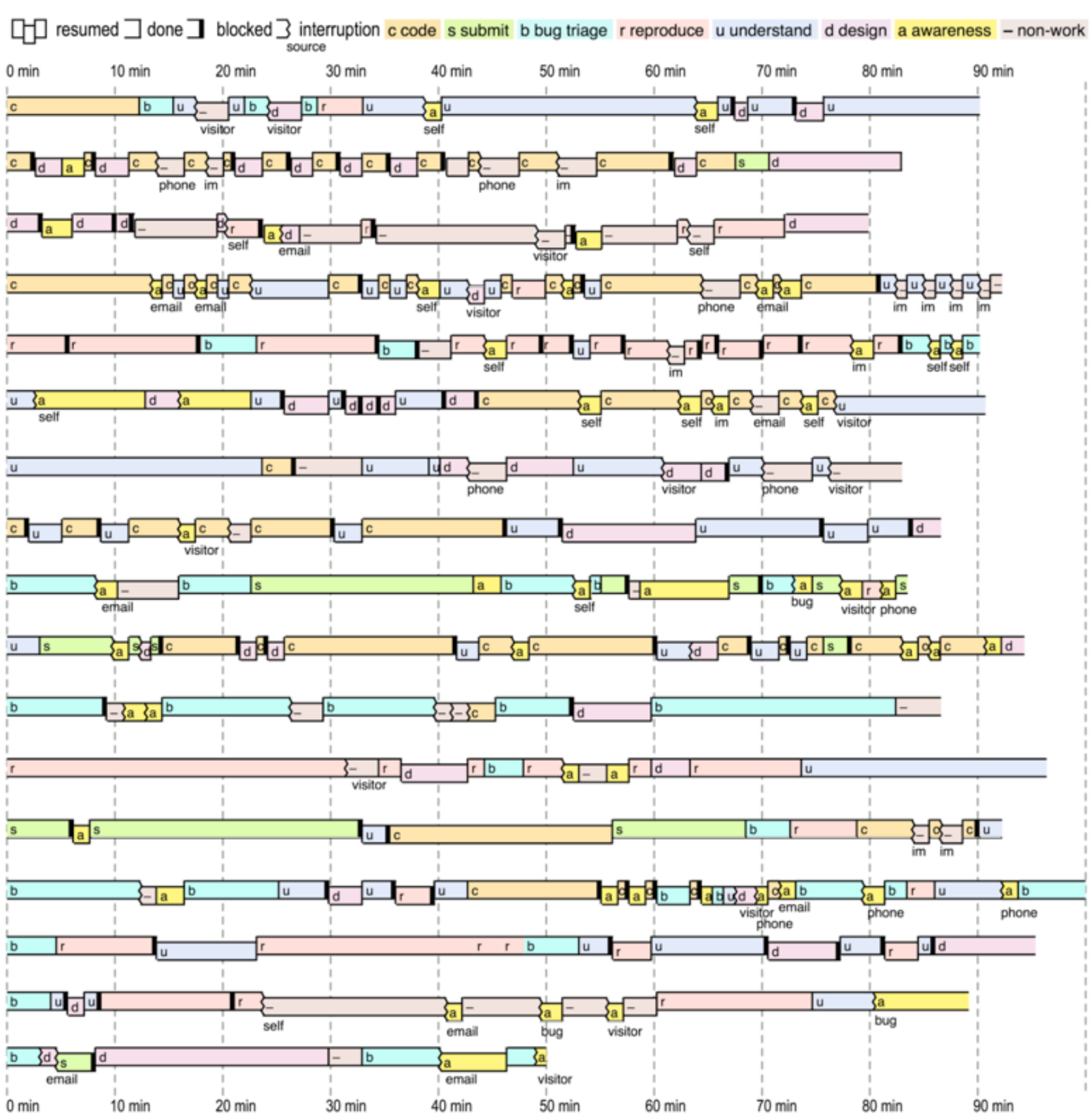


Figure 2. The backgrounds and task structures of the 17 observed developers. The right edge of each task block indicates the reason for the task switch (thin line for *done*, thick line for *blocked*, jagged line for *interrupted*). When a task gets broken up by interruptions or blocking, we draw its fragments at the same vertical level to show resumption.

Andrew J. Ko, Robert DeLine, and Gina Venolia. 2007. Information Needs in Collocated Software Development Teams. In Proceedings of the 29th international conference on Software Engineering (ICSE '07). IEEE Computer Society, Washington, DC, USA, 344-353. DOI: https://doi.org/10.1109/ICSE.2007.45

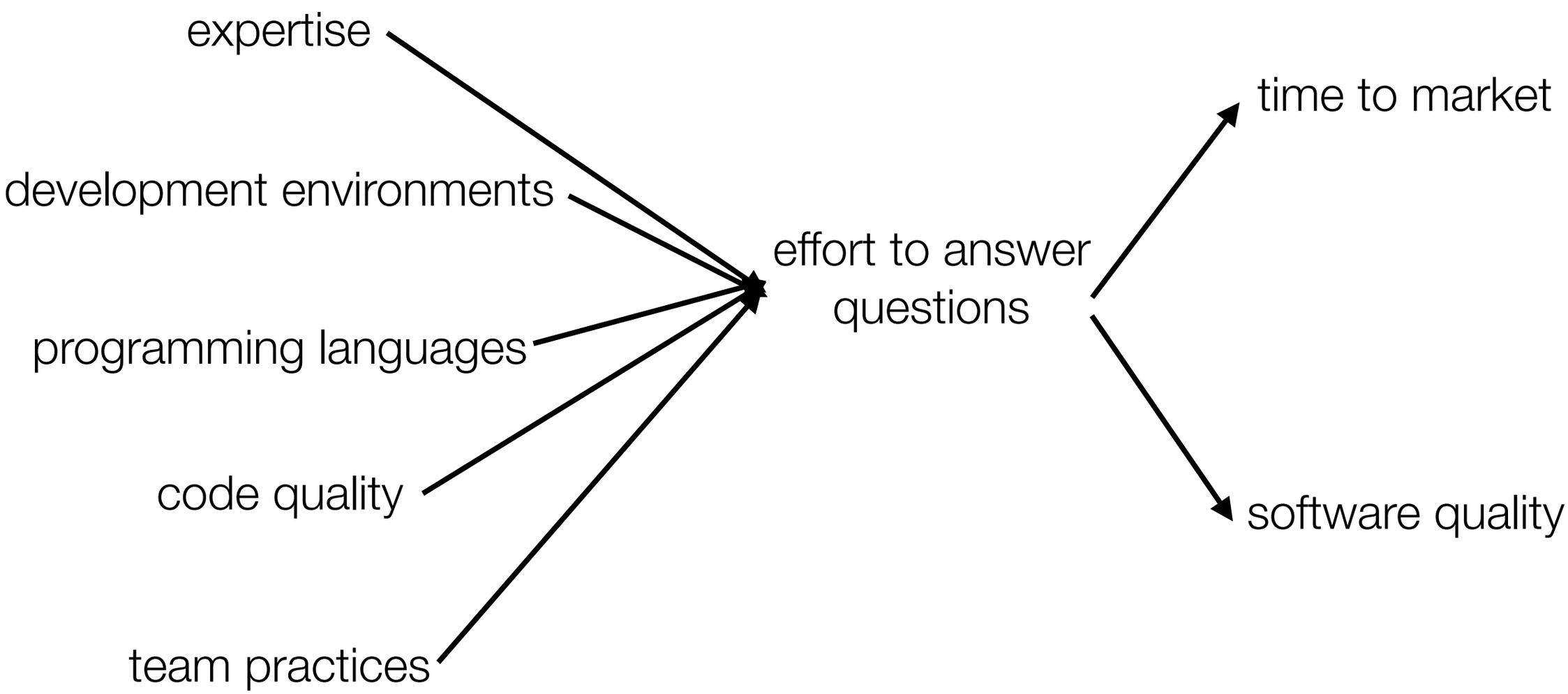
Some methods for supporting problem solving

- Find an important question, build tool that makes it easier to answer

- Find an action that helps developers answer questions, make it easier to take

- Find a new strategy that helps developers answer question more effectively

Many other factors influence difficulty answering questions



- Interventions might also target these factors

Some methods for supporting problem solving

- **Find an important question, build tool that makes it easier to answer**

- Find an action that helps developers answer questions, make it easier to take

- **Find a new strategy that helps developers answer question more effectively**

Making questions easier to answer

- 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

Example: Questions about object structure

Is a

Who implements type X? [who can be an object or a type]

Navigability

Let's say I am in the StandardDrawing class and I want the JavaDrawApp object which is a DrawingEditor [...]. What would save me a lot of time is to say now I am at the Drawing and I want to go to the DrawingEditor, show me my options.

Part of

Maybe I would start with the Drawing object and that should have a list of listeners?

How to get

How I will get hold of the DrawingEditor object? [...] Basically I need to know the instance of the current window.

I know I need to get the view from here; so how do I do that?

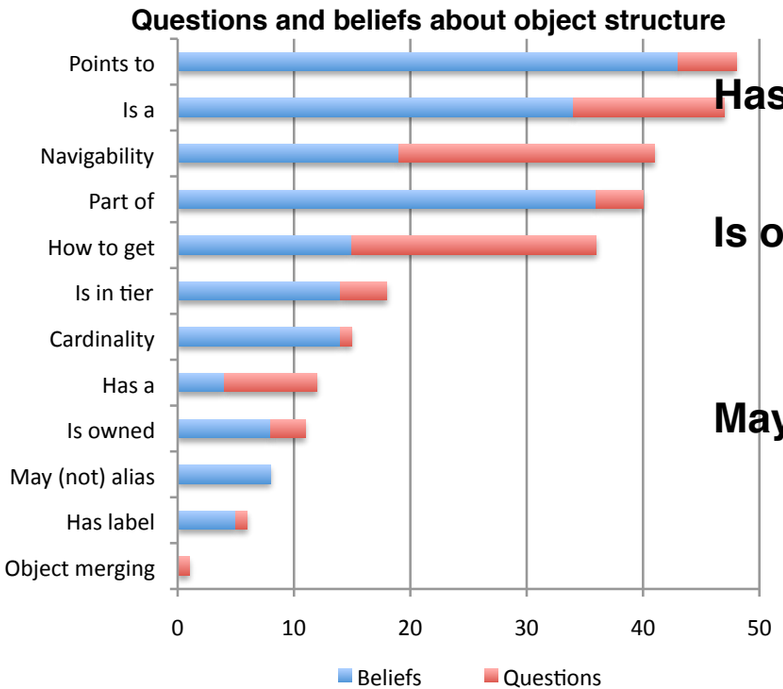
Is in tier

What I would be interested in is looking in the code to try to understand where are the view and model

Cardinality

The class diagram says that the DrawingEditor has one DrawingView and the StandardDrawingView may or may not have a Drawing.

I would like to know the cardinality: so Window has one or more StandardDrawingViews?



Has a

Maybe I would start with the Drawing object and that should have a list of listeners

Is owned

[...] the window itself has a reference to the UndoManager but you can't tell from this diagram whether each window has its own UndoManager, or whether it is just one global manager.

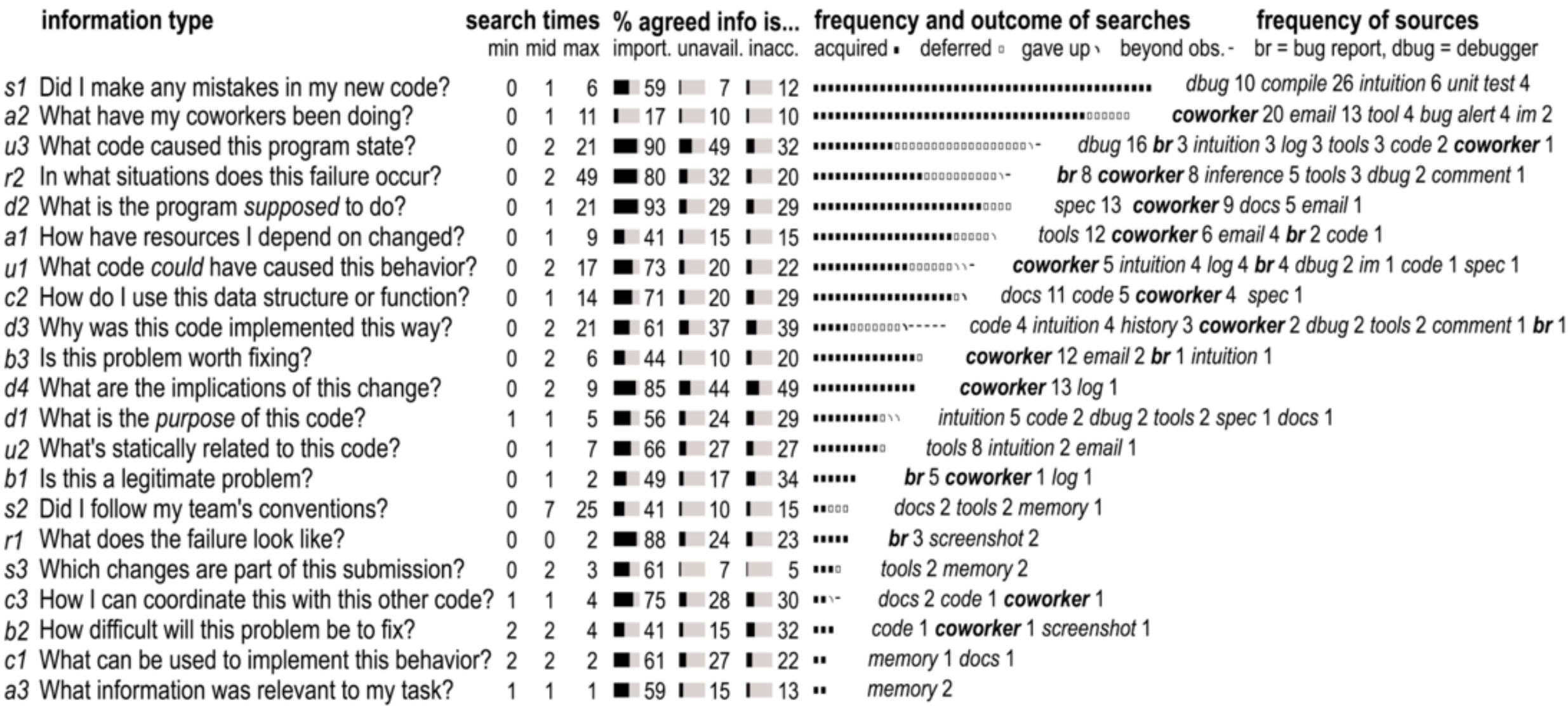
May alias

So I have different selections in the different views.

Both of them are two views on the same Drawing, but if there are two windows...

Marwan Abi-Antoun, Nariman Ammar, and Thomas LaToza. 2010. Questions about object structure during coding activities. Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '10). ACM, New York, NY, USA, 64-71. DOI=http://dx.doi.org/10.1145/1833310.1833321

Example: Programming questions



	information type	search times min	search times mid	search times max	% agreed info is... import.	% agreed info is... unavail.	% agreed info is... inacc.	frequency and outcome of searches (acquired ▪ deferred ▫ gave up \ beyond obs. -)	frequency of sources (br = bug report, dbug = debugger)
s1	Did I make any mistakes in my new code?	0	1	6	59	7	12	▪▪	*dbug* 10 *compile* 26 *intuition* 6 *unit test* 4
a2	What have my coworkers been doing?	0	1	11	17	10	10	▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▫▫▫▫▫▫	***coworker*** 20 *email* 13 *tool* 4 *bug alert* 4 *im* 2
u3	What code caused this program state?	0	2	21	90	49	32	▪▪▪▪▪▪▪▪▪▪▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫\-	*dbug* 16 ***br*** 3 *intuition* 3 *log* 3 *tools* 3 *code* 2 ***coworker*** 1
r2	In what situations does this failure occur?	0	2	49	80	32	20	▪▪▪▪▪▪▪▪▪▪▪▪▪▫▫▫▫▫▫▫▫▫▫\-	***br*** 8 ***coworker*** 8 *inference* 5 *tools* 3 *dbug* 2 *comment* 1
d2	What is the program *supposed* to do?	0	1	21	93	29	29	▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▫▫▫▫	*spec* 13 ***coworker*** 9 *docs* 5 *email* 1
a1	How have resources I depend on changed?	0	1	9	41	15	15	▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▫▫▫▫▫\	*tools* 12 ***coworker*** 6 *email* 4 ***br*** 2 *code* 1
u1	What code *could* have caused this behavior?	0	2	17	73	20	22	▪▪▪▪▪▪▪▪▪▪▪▪▫▫▫▫▫▫▫\\-	***coworker*** 5 *intuition* 4 *log* 4 ***br*** 4 *dbug* 2 *im* 1 *code* 1 *spec* 1
c2	How do I use this data structure or function?	0	1	14	71	20	29	▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▫\	*docs* 11 *code* 5 ***coworker*** 4 *spec* 1
d3	Why was this code implemented this way?	0	2	21	61	37	39	▪▪▪▪▫▫▫▫▫▫▫▫\-----	*code* 4 *intuition* 4 *history* 3 ***coworker*** 2 *dbug* 2 *tools* 2 *comment* 1 ***br*** 1
b3	Is this problem worth fixing?	0	2	6	44	10	20	▪▪▪▪▪▪▪▪▪▪▪▪▪▪▫	***coworker*** 12 *email* 2 ***br*** 1 *intuition* 1
d4	What are the implications of this change?	0	2	9	85	44	49	▪▪▪▪▪▪▪▪▪▪▪▪▪▪	***coworker*** 13 *log* 1
d1	What is the *purpose* of this code?	1	1	5	56	24	29	▪▪▪▪▪▪▪▪▪▫\\	*intuition* 5 *code* 2 *dbug* 2 *tools* 2 *spec* 1 *docs* 1
u2	What's statically related to this code?	0	1	7	66	27	27	▪▪▪▪▪▪▪▪▪▫	*tools* 8 *intuition* 2 *email* 1
b1	Is this a legitimate problem?	0	1	2	49	17	34	▪▪▪▪▪▪	***br*** 5 ***coworker*** 1 *log* 1
s2	Did I follow my team's conventions?	0	7	25	41	10	15	▪▪▫▫▫	*docs* 2 *tools* 2 *memory* 1
r1	What does the failure look like?	0	0	2	88	24	23	▪▪▪▪▪	***br*** 3 *screenshot* 2
s3	Which changes are part of this submission?	0	2	3	61	7	5	▪▪▪▫	*tools* 2 *memory* 2
c3	How I can coordinate this with this other code?	1	1	4	75	28	30	▪▪\-	*docs* 2 *code* 1 ***coworker*** 1
b2	How difficult will this problem be to fix?	2	2	4	41	15	32	▪▪▪	*code* 1 ***coworker*** 1 *screenshot* 1
c1	What can be used to implement this behavior?	2	2	2	61	27	22	▪▪	*memory* 1 *docs* 1
a3	What information was relevant to my task?	1	1	1	59	15	13	▪▪	*memory* 2

Andrew J. Ko, Robert DeLine, and Gina Venolia. 2007. Information Needs in Collocated Software Development Teams. In Proceedings of the 29th international conference on Software Engineering (ICSE '07). IEEE Computer Society, Washington, DC, USA, 344-353. DOI: https://doi.org/10.1109/ICSE.2007.45

This is a serious problem for me	% agree
Code Understanding	
Understanding the rationale behind a piece of code	66%
Understanding code that someone else wrote	56%
Understanding the history of a piece of code	51%
Understanding code that I wrote a while ago	17%
Task Switching	
Having to switch tasks often because of requests from my teammates or manager	62%
Having to switch tasks because my current task gets blocked	50%
Modularity	
Being aware of changes to code elsewhere that impact my code	61%
Understanding the impact of changes I make on code elsewhere	55%
Links between Artifacts	
Finding all the places code has been duplicated	59%
Understanding who "owns" a piece of code	50%

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

Questions developers report as hard to answer span many topics

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.

Many of these already have tools that support them

- Debugging
- Refactoring
- Design Rationale

- So if there's **already** a tool designed to **support** this, why is it still so hard??

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

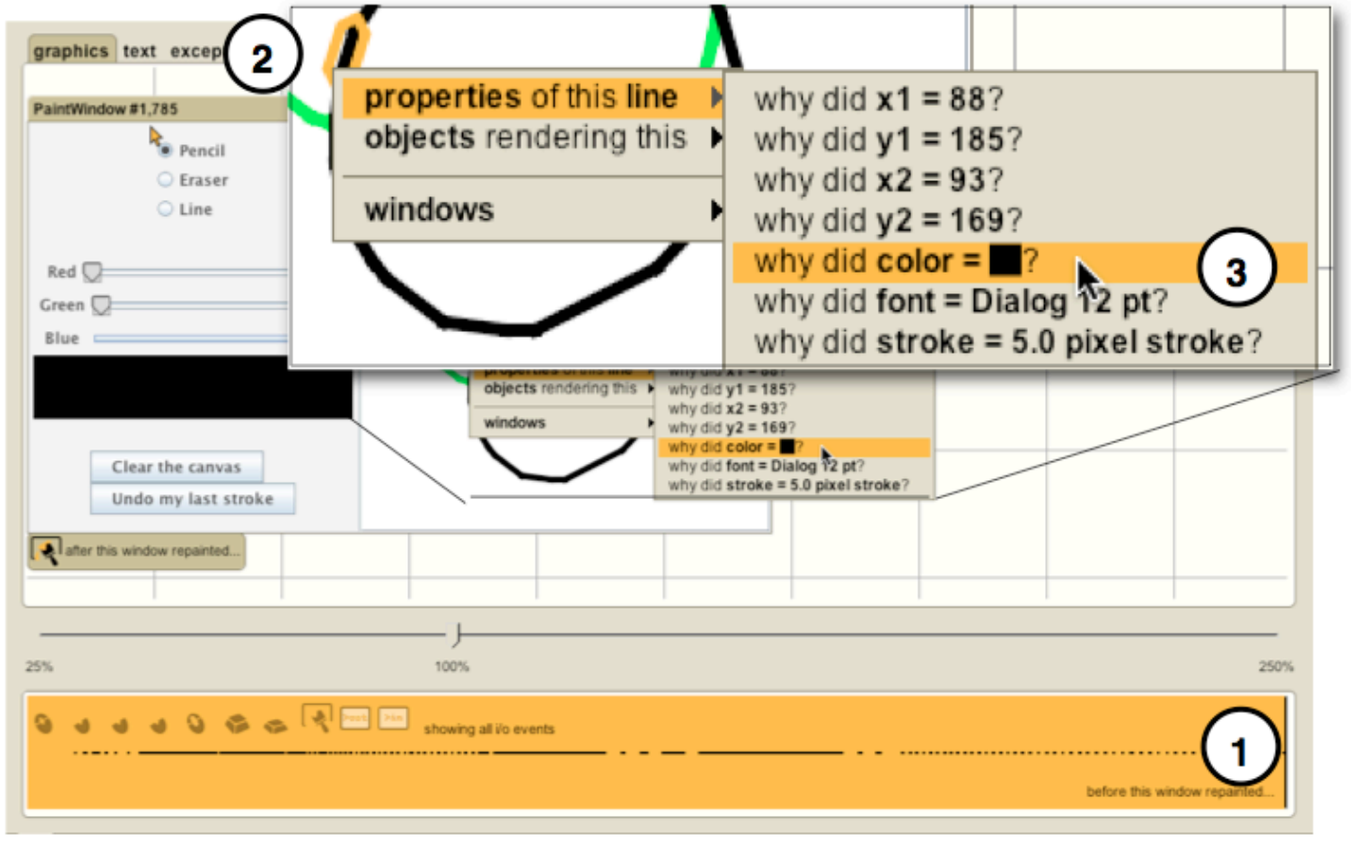
- *Where was this **variable** last changed? (1)*
- *Why **didn't** this happen? (3)*

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)*
- *What runtime state **changed** when this executed? (2)*
- *Which **team's** component caused this bug? (1)*

Which team should I assign this bug to?

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
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

Find a new strategy that makes question easier to answer

Example of a programming strategy

```
# This Strategy helps you merge 2 branches in githup and resolve conflicts
#Required Tools and Environments
Installing git
Github account
Ongoing project which is progressing in at least 2 branches
Git repository that is not associated with Github
#Required Knowledge
Basic git command knowledge
Knowledge of how to work with terminal and run commands
STRATEGY GitMerge()
    # Open the teminal, and use cd(change directory) command to move to the
    local git project directory
    Open the terminal and navigate to your git project directory
    IF you are not in the master branch
        # Run the command git checkout master
        checkout to the master branch
IF you are in the master branch
    # To merge the second branch with the master branch run the command
    "git merge secondBranch", which secondBranch is the name of your git
    second branch
    Merge the two branches
    IF the merge has a conflict
        SET   'conflictedFiles'   TO the project files that have a conflict
        FOR EACH 'file' IN 'conflictedFiles'
            DO   fixConflict('file')
    # Run GIT STATUS to see the latest changes
    # Run GIT ADD
    # Run GIT COMMIT -m ""
    # Run GIT PUSH
    Commit and push the changes
RETURN nothing
```

Developers often have choices between strategies

Question Can I remove this call?

Strategy: ***Implement & test***	vs.	**Strategy:** ***Understand***
Remove the call & test behavior change		Understand implications before editing by investigating callees

Guess and check debugging

1. Describe in words how the program is failing
2. Brainstorm a list of possible causes of this failure
3. For each possible cause:
    1. Read the potentially defective code.
    2. Gather data about program execution to verify that it is the defect.
    3. If it is the defect, repair it.
4. If you didn't find the defect, return to 2

```
STRATEGY debug()
  # Is the faulty output you're investigating printed to a command line?
  IF the faulty output is logged to a command line
    # To find print statements, try searching for keywords related to 'log' or 'print'
    SET outputLines TO the line numbers of calls to console logging functions
  # Graphical output includes things like colored lines and rectangles
  IF the faulty output is graphical output
    # To find these lines, try searching for keywords related to graphical output, like '
    # draw' or 'fill'. Focus on lines that directly render something, not on higher-level
     # functions that indirectly call rendering functions.
    SET outputLines TO the line numbers of function calls that directly render graphics to the screen
  # Now that you have some lines that could have directly produced the faulty output, you're
  # going to check each line, see if it executed, and then find the cause of it executing. If
  # you're lucky, you only have one output line to check.
  FOR EACH 'line' IN 'outputLines'
    IF the program executed 'line'
      Analyze the line to determine its role in the overall behavior of the program
      # Check for errors such as the wrong function being called, the wrong argument being
      # passed to a function, the wrong variable being referenced, or a wrong operator being
      # used.
      IF any part of 'line' is inconsistent with its purpose
        # You found the bug
        RETURN 'line'
      # If the output statement is not wrong, perhaps the line was not supposed to execute at all?
      IF 'line' was not supposed to execute at all
        # The conditional might be in the same function as the output statement, or it might
        # have been a conditional in a function that called this function. Check the call
        # stack if necessary by setting a breakpoint. Find the conditional that led this line
        # to being executed
        # Some value in the conditional's boolean expression must have been wrong. Which
        # value was it?
        SET 'wrongValue' to the value in the conditional's boolean expression that ultimately
            allowed the faulty output to execute
        # We'll use another strategy to find the source of the incorrect value.
        RETURN  localizeWrongValue('wrongValue')
      # If the line was supposed to execute, but it executed with an incorrect value, find
      # that value.
      IF 'line' executed with an incorrect value
        SET 'wrongValue' TO the incorrect value
        # We'll use another strategy to find the cause of the incorrect value.
        RETURN localizeWrongValue('wrongValue')
# If you made it to this line, then you must have missed something. Is it possible you
# made a mistake above? If so, go back and verify your work, because something caused the
# faulty output.
RETURN nothing
```

Many factors influence the effectiveness of a strategy in a situation

Influencing factor	**Strategy: *Implement & test*** vs.	**Strategy: *Understand***
Work style [Clarke+04]	Opportunistic	Systematic
Development process	Test-driven development	Few unit tests
Cost of bugs	Low	High
Time to implement	Easy to implement	Hard to implement
Difficulty of testing	An easily tested property (e.g., performance)	Non-functional property (e.g., testing usability)
Test execution time	Short-running test suites	Long-running test suites

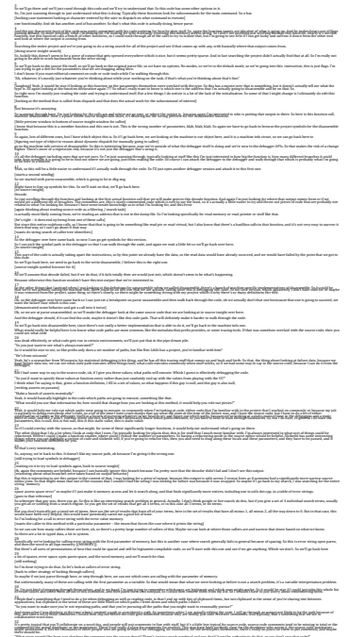
Developers often rapidly switch between alternative actions or strategies

Where is method *m* generating an error?

Rapidly found method *m* implementing command
Unsure **where** it generated error

static call traversal — Statically traversed calls looking for something that would generate error

debugger — Tried debugger



grep — Did string **search** for error, found it, but many callers

debugger — **Stepped** in debugger to find something relevant

static call traversal — Statically **traversed** calls to explore

debugger — Went back to **stepping** debugger to inspect values
Found the answer

(**66 minutes)**

Developers often rapidly switch between alternative actions or strategies

Lacks **knowledge** to determine how these lines influence program behavior

Tries to recover **rationale**, but no explanation in check-in message

Tests might have identified a bug, but don't prove **absence**.

Teammates **remembered** another scenario.

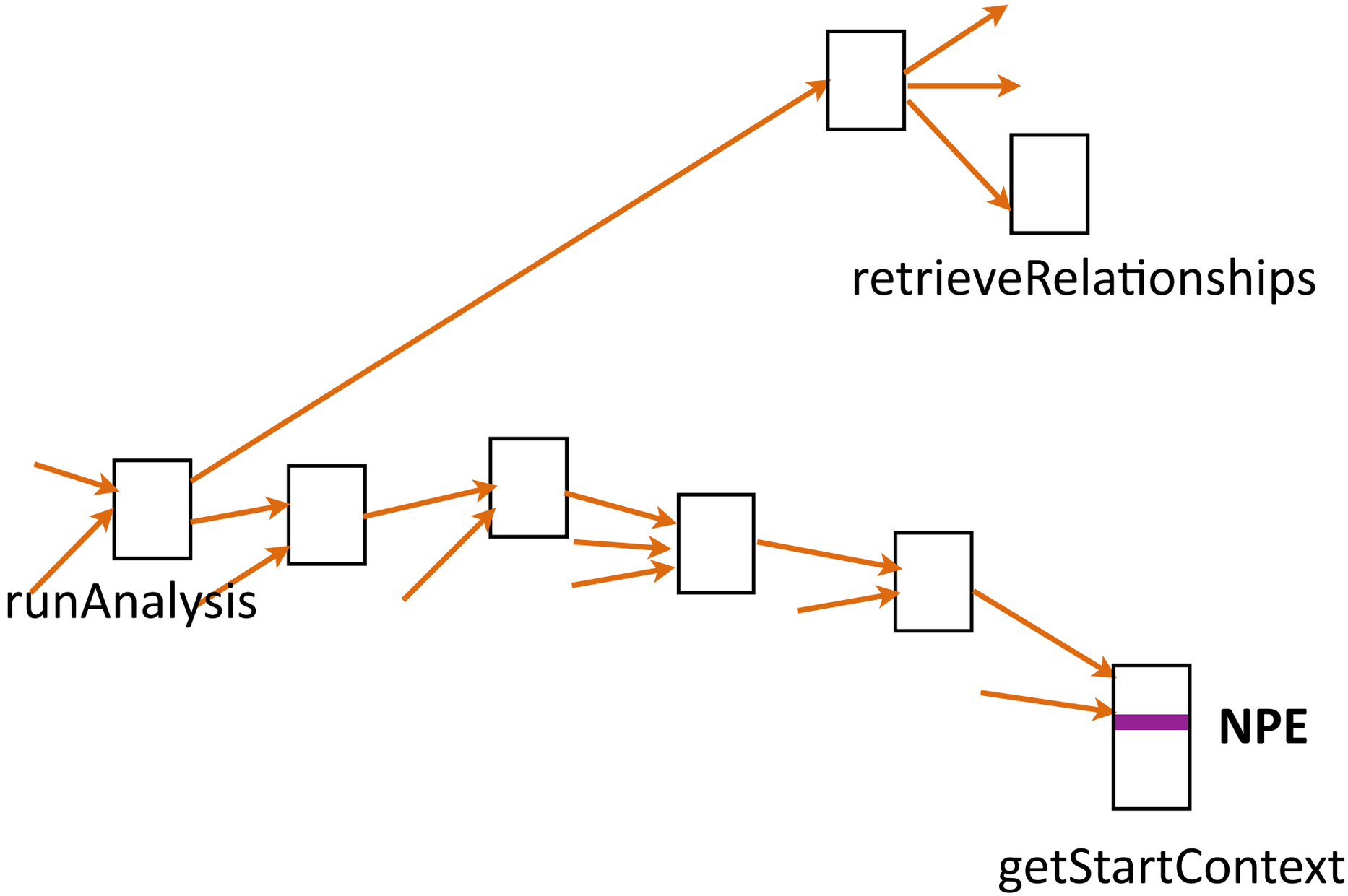
Strategy 1. ***Guess the answer.***
— This was a quick hack, not a reasoned changed because otherwise they would have been removed. But what would break if they were here?

Strategy 2. ***Check code history.***
— I commented these out 2 years ago along with many other changes. But why?

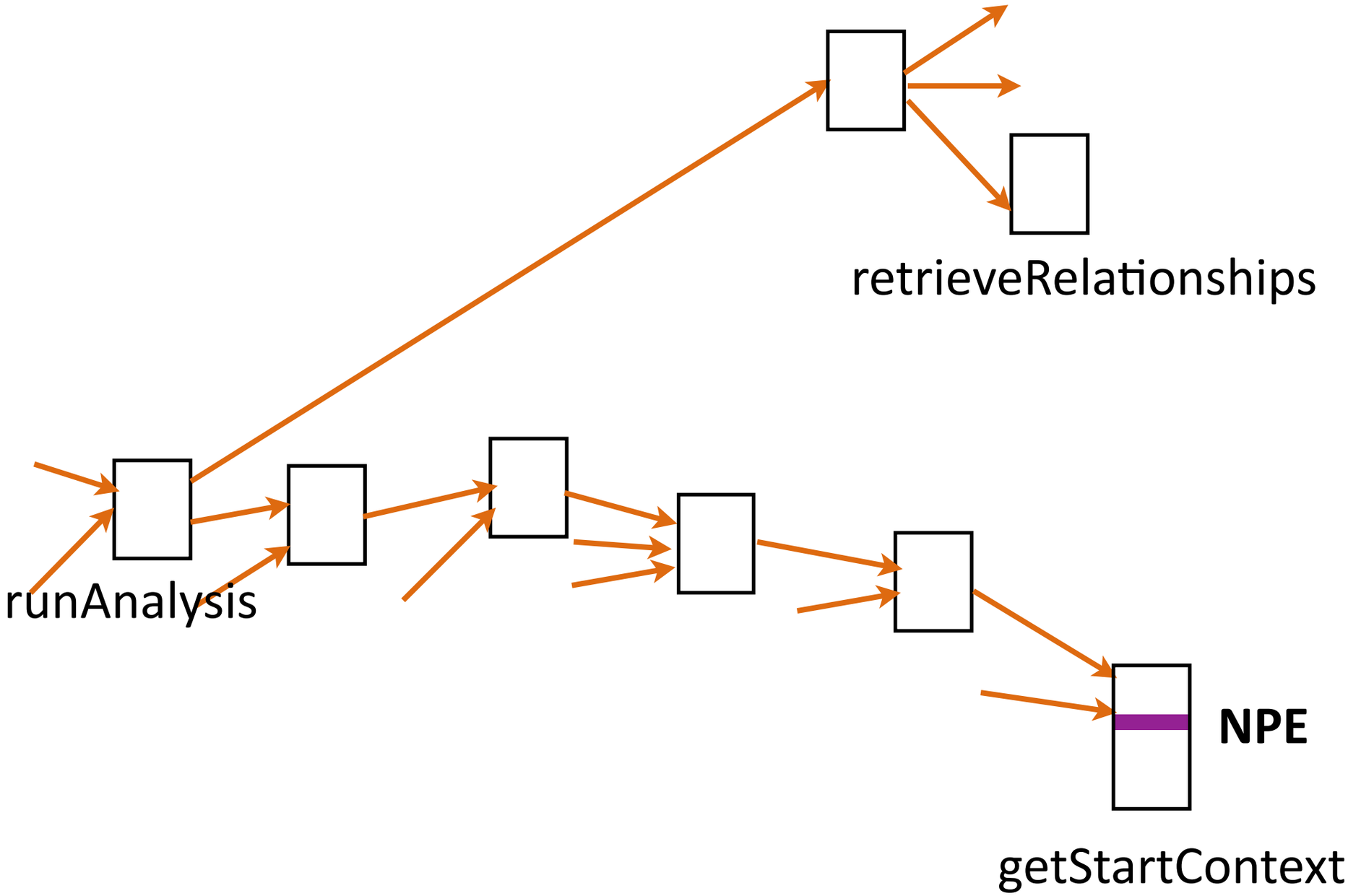
Strategy 3. ***Implement & test.***
— Removed comments, all tests still pass.
But did I break anything?

Strategy 4. ***Ask my teammates.***
— Sent an email. Teammates replied with a description of a rare input which causes it to break. Success!

Some strategies are more effective than others in a specific situation



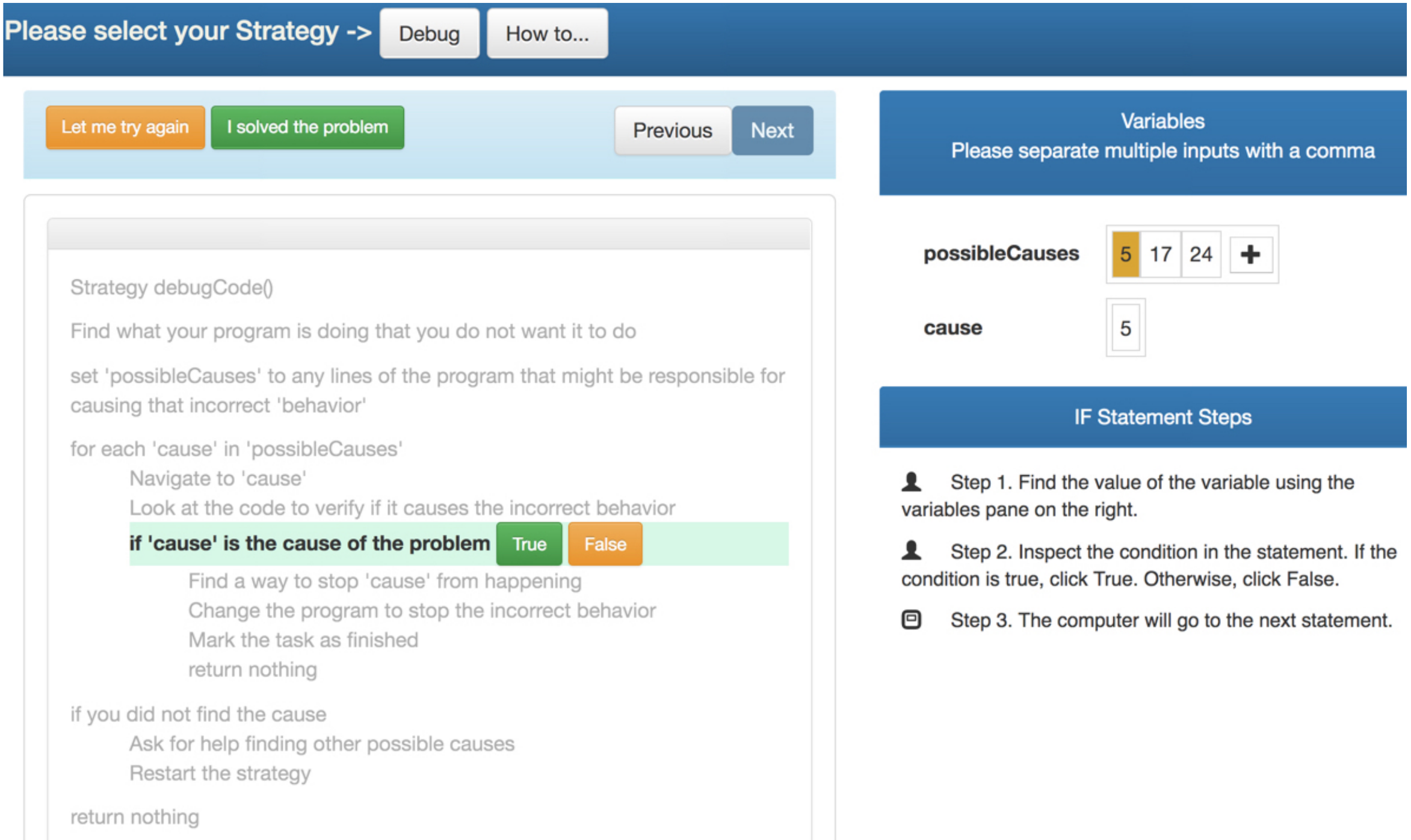
Strategies can make a large difference in task performance



Teaching strategies

- If some strategies are more effective, maybe we could just teach them?
  - Write down lots of expert strategies.
  - Give developers the written version of the strategy?
- Developers are habitual, solve problems as they always have
  - Read it once, try to follow it, return to prior strategy

Scaffolding novel strategy use



Andrew J. Ko, Thomas D. LaToza, Stephen Hull, Ellen Ko, William Kwok, Jane Quichocho, Harshitha Akkaraju and Rishin Pandit. Teaching Explicit Programming Strategies to Adolescents. SIGCSE 2019.

Teaching Strategies to Developers

- Recruited 28 participants with JS experience and varying levels of industry experience (mean 2.5 years)

- Asked participants to debug a defect and complete a code design task, compared natural strategies against strategy shared through Strategy Tracker

- Most participants with Strategy Tracker switched strategy
  - Design: Decompose or Template --> TDD
  - Debugging Forwards search --> backwards search

- Enabled debugging participants to make more progress

What's next: Supporting information needs

- Many of the next lectures will focus on information needs in specific programming activities
  - Editing code
  - Debugging
  - Navigating code
  - Refactoring
  - Code reuse
- Some will focus on specific tool approaches
  - Visualization
  - Crowdsourcing
  - Program synthesis
- Next week: Programming as communication