Understanding Code Part 1

CS 691 / SWE 699
Fall 2025

GEORGE MASON UNIVERSITY

Logistics

- Project Proposal, Reflection 3, & Lecture 7 Reading questions due today at 4:30pm
- Lecture 8 reading questions due next week at 4:30pm
- Lecture 8 activity (in class today), due by 10/17 at 4:30pm
- Updates to course schedule

Course Project

- Will give feedback on project proposal
- Next step: Project Checkpoint, due 11/6
  - Talk about the progress you've made on the project

Today

- Discussion: Experiences from Lecture 6
- Discussion: Reading questions for Lecture 7
- Lecture
  - Understanding Code
- In-Class Activity

Discussion: Experiences from Lecture 6 Activity

- Do you feel that you were faster with or without Cursor?
- Where was Cursor helpful?
- Where was it not helpful?
- To what extent did it increase your confidence in debugging?
- What did you learn about how to use Cursor effectively in debugging tasks?

Discussion: Reading questions for Lecture 7

- What questions did you have from readings for Lecture 7
  - Discuss questions & possible answers in group of 3 or 4
  - Come back with 1 question you want to discuss w/ whole class

Reading Code

Demo: Remember this code (10 seconds)

```
var express = require('express');
var app = express();
const fetch = require('node-fetch');

const body = { 'a': 1 };

fetch('http://localhost:3000/book/23', {
    method: 'post',
    body:    JSON.stringify(body),
    headers: { 'Content-Type': 'application/json' },
})
    .then(res => res.json())
    .then(json => console.log(json));
```

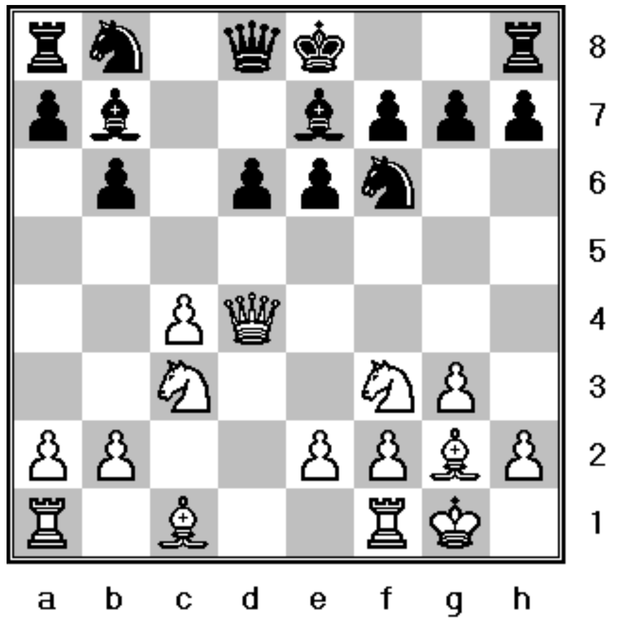
Demo: Remember this code (10 seconds)

```
Set<Integer> numbers = new HashSet<>();

numbers.add(100);
numbers.add(35);
numbers.add(89);
numbers.add(71);

Iterator<Integer> iterator = numbers.iterator();

while (iterator.hasNext()) {
    Integer aNumber = iterator.next();
    System.out.println(aNumber);
}
```

Memory and comprehension

- When stimuli are received by human, encoded into memory as they are processed.
- *How* they are encoded depends on what knowledge structures already exist
- Depending on knowledge structures, how this information is represented may be very different

What makes a grand master a chess expert?



- Memory for **random** chess boards: **same** for experts and novices
- Memory for position from **actual** game: much better for **experts** than novices
- [deGroot 1946; Chase & Simon 1973]

What makes an expert?

- Experts are more intelligent?
  - IQ doesn't distinguish best chess players or most successful artists or scientists (Doll & Mayr 1987) (Taylor 1975)

- Experts think faster or have larger memory?
  - World class chess experts don't differ from experts

- Experts have schemas!

Experts create schemas by chunking world

- Schema: a template (struct) describing a set of slots

```
while (x > 0)
{
  invokeAction(actions[x]);
  x—;
}
```

- Experts perceive the world through schemas
  - “Chunk” and interpret visual stimuli to determine which schemas are present
  - Form concepts that help developers think in abstractions

Program comprehension as text comprehension

- Developers recognize specific “beacons” (a.k.a. features) in code that activate schemas
  - e.g., for (elem in elements)

- Developers mentally represent programs in terms of schemas
  - Reason about behavior of program using schemas
  - Recall what code is or is not present using schemas

Implications of text comprehension

- Distortions of form in recall
  - Developers more likely to recall prototypical schema values rather than actual.

- Distortions of content
  - Developers more likely to recall values inferred from schemas that were not present in code.

Developers perceive programming plan, control flow, data flow representations

- Build and possess different abstractions of code
- Programming plan
  - Hierarchic decomposition of goals in program
- Control flow
  - Control flow in a method
- Data flow
  - Data flow in a method

Another code example

```
public int getFoldLevel(int line) {
    if (line < 0 || line >= lineMgr.getLineCount())
        throw new ArrayIndexOutOfBoundsException(line);

    if (foldHandler instanceof DummyFoldHandler)
        return 0;

    int firstInvalidFoldLevel = lineMgr.getFirstInvalidFoldLevel();
    if (firstInvalidFoldLevel == -1 || line < firstInvalidFoldLevel) {
        return lineMgr.getFoldLevel(line);
    } else {
        if (Debug.FOLD_DEBUG)
            Log.log(Log.DEBUG, this, "Invalid fold levels from "
                    + firstInvalidFoldLevel + " to " + line);

        int newFoldLevel = 0;
        boolean changed = false;

        for (int i = firstInvalidFoldLevel; i <= line; i++) {
            newFoldLevel = foldHandler.getFoldLevel(this, i, seg);
            if (newFoldLevel != lineMgr.getFoldLevel(i)) {
                if (Debug.FOLD_DEBUG)
                    Log.log(Log.DEBUG, this, i + " fold level changed");
                changed = true;
            }
            lineMgr.setFoldLevel(i, newFoldLevel);
        }

        if (line == lineMgr.getLineCount() - 1)
            lineMgr.setFirstInvalidFoldLevel(-1);
        else
            lineMgr.setFirstInvalidFoldLevel(line + 1);

        if (changed) {
            if (Debug.FOLD_DEBUG)
                Log.log(Log.DEBUG, this, "fold level changed: "
                        + firstInvalidFoldLevel + "," + line);
            fireFoldLevelChanged(firstInvalidFoldLevel, line);
        }

        return newFoldLevel;
    }
```

Experienced developers learn facts at a higher level of abstraction

EXPERTS

"Well, this is just updating a cache" *(1 min)*

NOVICES

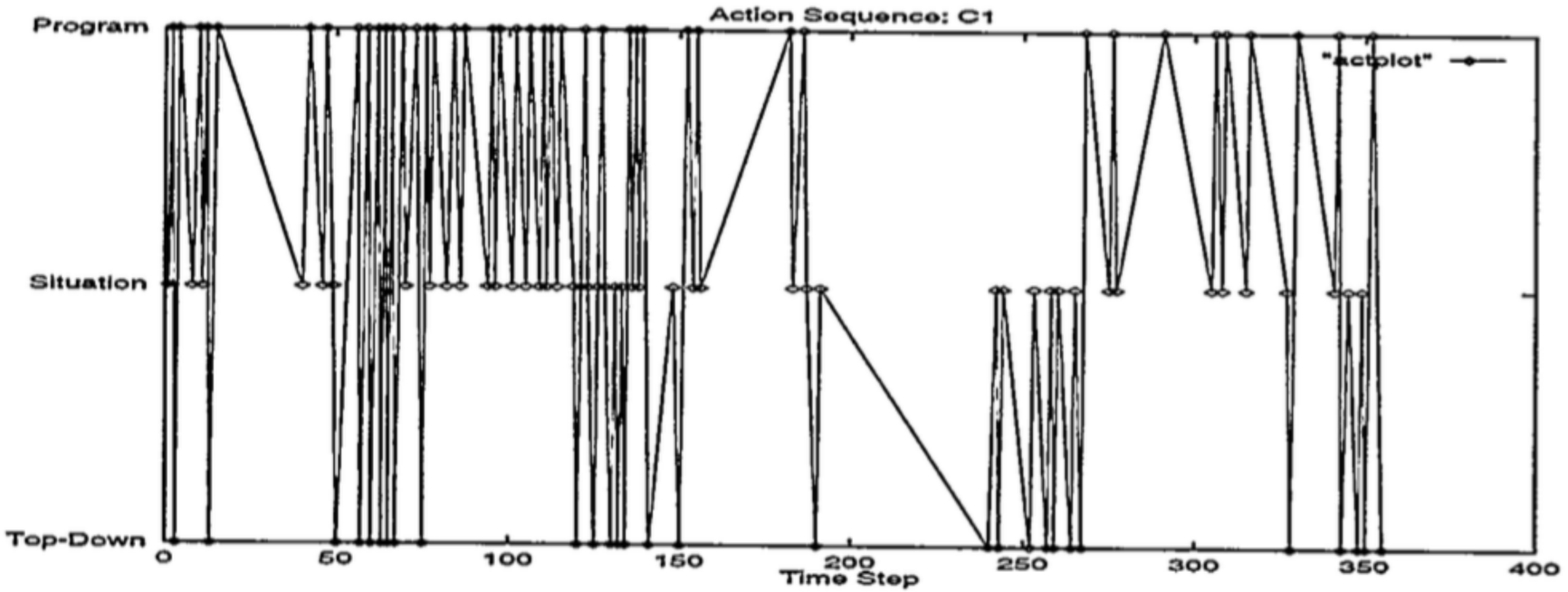
"What it did was it…computes the new line number and fires an event. But I didn't see it change any state." *(38 mins, 10 mins reading getFoldLevel)*

"So what it does, it starts off from this line, it has this firstInvalidFoldLevel, it goes through all these lines, it checks whether this fold information is correct or not, which is this newFoldLevel, this is supposed to be the correct fold level. If that is not the case in the data structure, it needs to change the state of the buffer. It creates this, it does this change, it sets the fold level of that line to the new fold level." *(51 mins, 12 mins reading getFoldLevel)*

```
public int getFoldLevel(int line) {
    if (line < 0 || line >= lineMgr.getLineCount())
        throw new ArrayIndexOutOfBoundsException(line);

    if (foldHandler instanceof DummyFoldHandler)
        return 0;

    int firstInvalidFoldLevel = lineMgr.getFirstInvalidFoldLevel();
    if (firstInvalidFoldLevel == -1 || line < firstInvalidFoldLevel) {
        return lineMgr.getFoldLevel(line);
    } else {
        if (Debug.FOLD_DEBUG)
            Log.log(Log.DEBUG, this, "Invalid fold levels from "
                    + firstInvalidFoldLevel + " to " + line);

        int newFoldLevel = 0;
        boolean changed = false;

        for (int i = firstInvalidFoldLevel; i <= line; i++) {
            newFoldLevel = foldHandler.getFoldLevel(this, i, seg);
            if (newFoldLevel != lineMgr.getFoldLevel(i)) {
                if (Debug.FOLD_DEBUG)
                    Log.log(Log.DEBUG, this, i + " fold level changed");
                changed = true;
            }
            lineMgr.setFoldLevel(i, newFoldLevel);
        }

        if (line == lineMgr.getLineCount() - 1)
            lineMgr.setFirstInvalidFoldLevel(-1);
        else
            lineMgr.setFirstInvalidFoldLevel(line + 1);

        if (changed) {
            if (Debug.FOLD_DEBUG)
                Log.log(Log.DEBUG, this, "fold level changed: "
                        + firstInvalidFoldLevel + "," + line);
            fireFoldLevelChanged(firstInvalidFoldLevel, line);
        }

        return newFoldLevel;
    }
```

Thomas D. LaToza, David Garlan, James D. Herbsleb, and Brad A. Myers. 2007. Program comprehension as fact finding. In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC-FSE '07). ACM, New York, NY, USA, 361-370. DOI: https://doi.org/10.1145/1287624.1287675

Developers constantly switch the level of abstraction with which they consider code

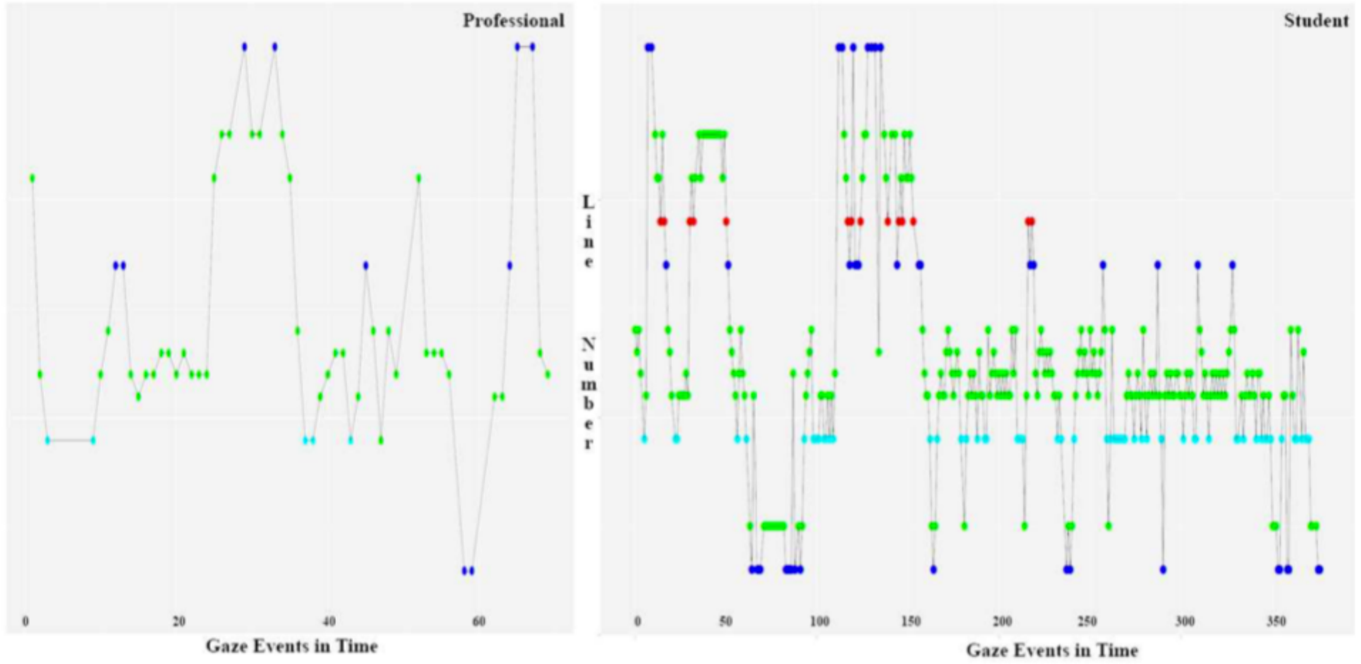


Anneliese von Mayrhauser and A. Marie Vans. 1997. Program understanding behavior during debugging of large scale software. In Papers presented at the seventh workshop on Empirical studies of programmers (ESP '97), Susan Wiedenbeck and Jean Scholtz (Eds.). ACM, New York, NY, USA, 157-179. DOI=http://dx.doi.org/10.1145/266399.266414

Reading code

- Can use eye gaze data to track moment to moment the line of code a developer is reading.

```
531 private String parseTextToken() throws IOException
532   {
533     StringBuffer token = new StringBuffer(20);
534
535     while (true)
536     {
537       int c = read();
538             //Util.pr(".. "+c);
539       if (c == -1)
540       {
541         _eof = true;
542
543         return token.toString();
544       }
545
546       if (Character.isLetterOrDigit((char) c) ||
547                         (c == ':') || (c == '-')
548                         || (c == '_') || (c == '*') || (c == '+') || (c == '.')
549                         || (c == '/') || (c == '\''))
550       {
551         token.append((char) c);
552       }
553       else
554       {
555         unread(c);
556                         //Util.pr("Pasted text token:+token.toString());
557         return token.toString();
558       }
559     }
560 }
```



Katja Kevic, Braden M. Walters, Timothy R. Shaffer, Bonita Sharif, David C. Shepherd, and Thomas Fritz. 2015. Tracing software developers' eyes and interactions for change tasks. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). ACM, New York, NY, USA, 202-213. DOI: https://doi.org/10.1145/2786805.2786864

Reading code: Some findings

- Developers only look at a few lines within methods, on average 32%.
- Developers constantly switch lines.
- Developers spend most of their time looking at method invocations and variable declaration statements.
- Developers follow data flows within a method (58% of navigations are within slice).

Katja Kevic, Braden M. Walters, Timothy R. Shaffer, Bonita Sharif, David C. Shepherd, and Thomas Fritz. 2015. Tracing software developers' eyes and interactions for change tasks.
In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). ACM, New York, NY, USA, 202-213. DOI: https://doi.org/10.1145/2786805.2786864

Information Needs

Theories of Information Needs in Programming

- Developers ask **questions**

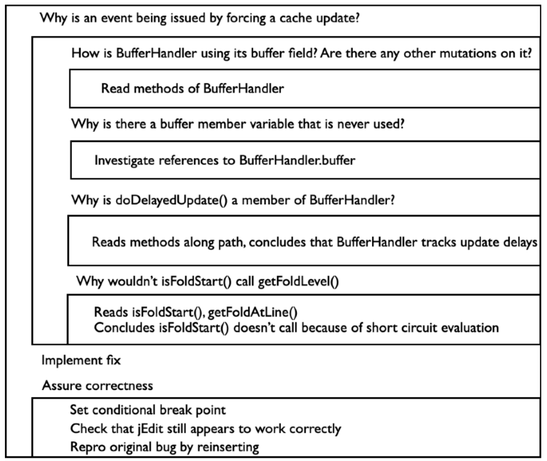
What does this do when input is null?

What part of this is being done client side and what part server side?

- Questions are **task-specific**

debugging refactoring testing testing

- Answering questions raises **more questions.**



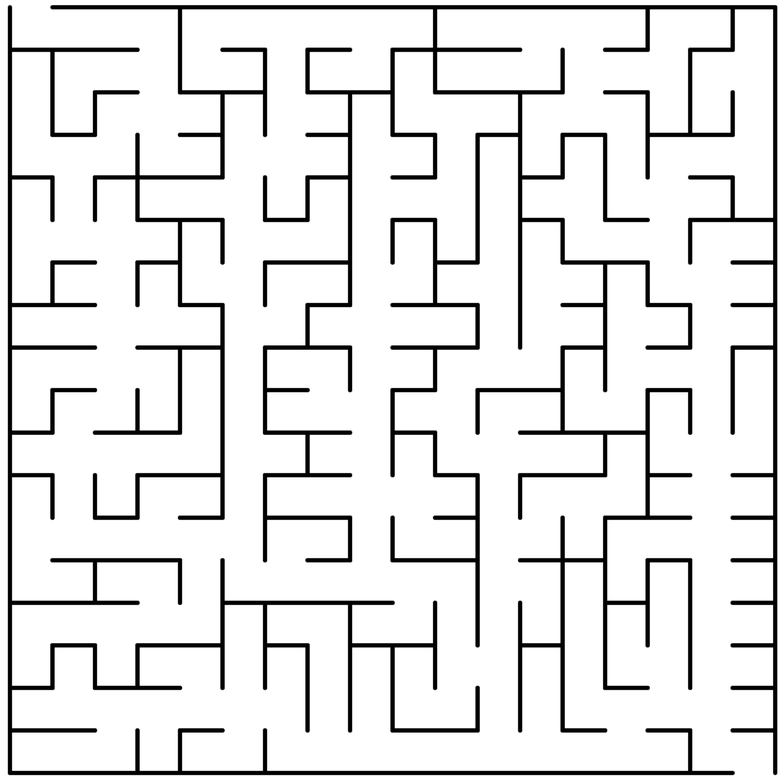
- Tool which successfully supports the questions a developer asks increases their productivity

T. D. LaToza, "Information Needs: Lessons for Programming Tools," in *IEEE Software*, vol. 37, no. 6, pp. 52-57, Nov.-Dec. 2020, doi: 10.1109/MS.2020.3014343.

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?

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

- 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

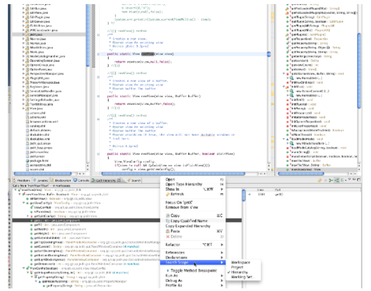
Why wouldn't isFoldStart() call getFoldLevel()

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

Implement fix

Assure correctness

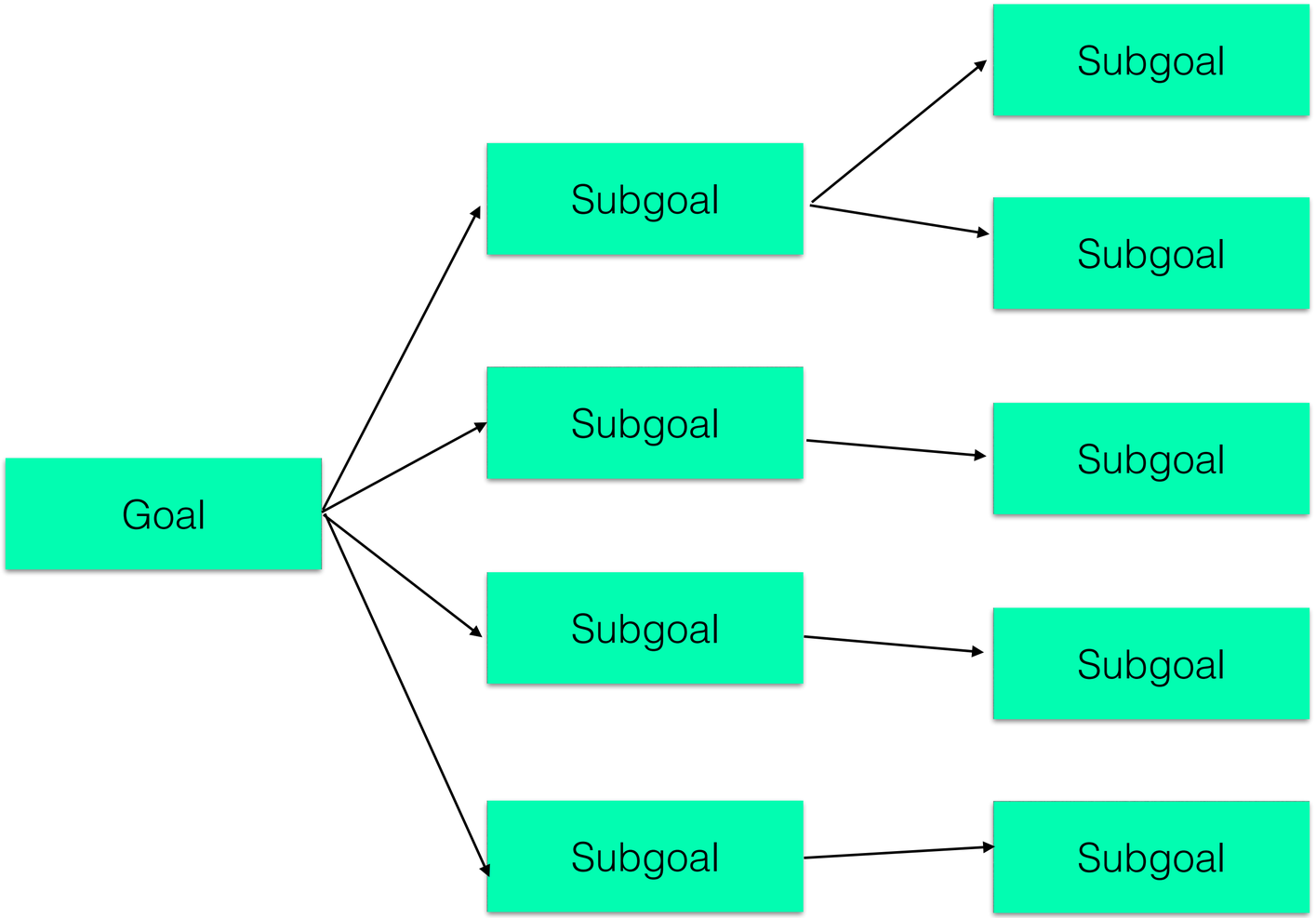
- Set conditional break point
  Check that jEdit still appears to work correctly
  Repro original bug by reinserting



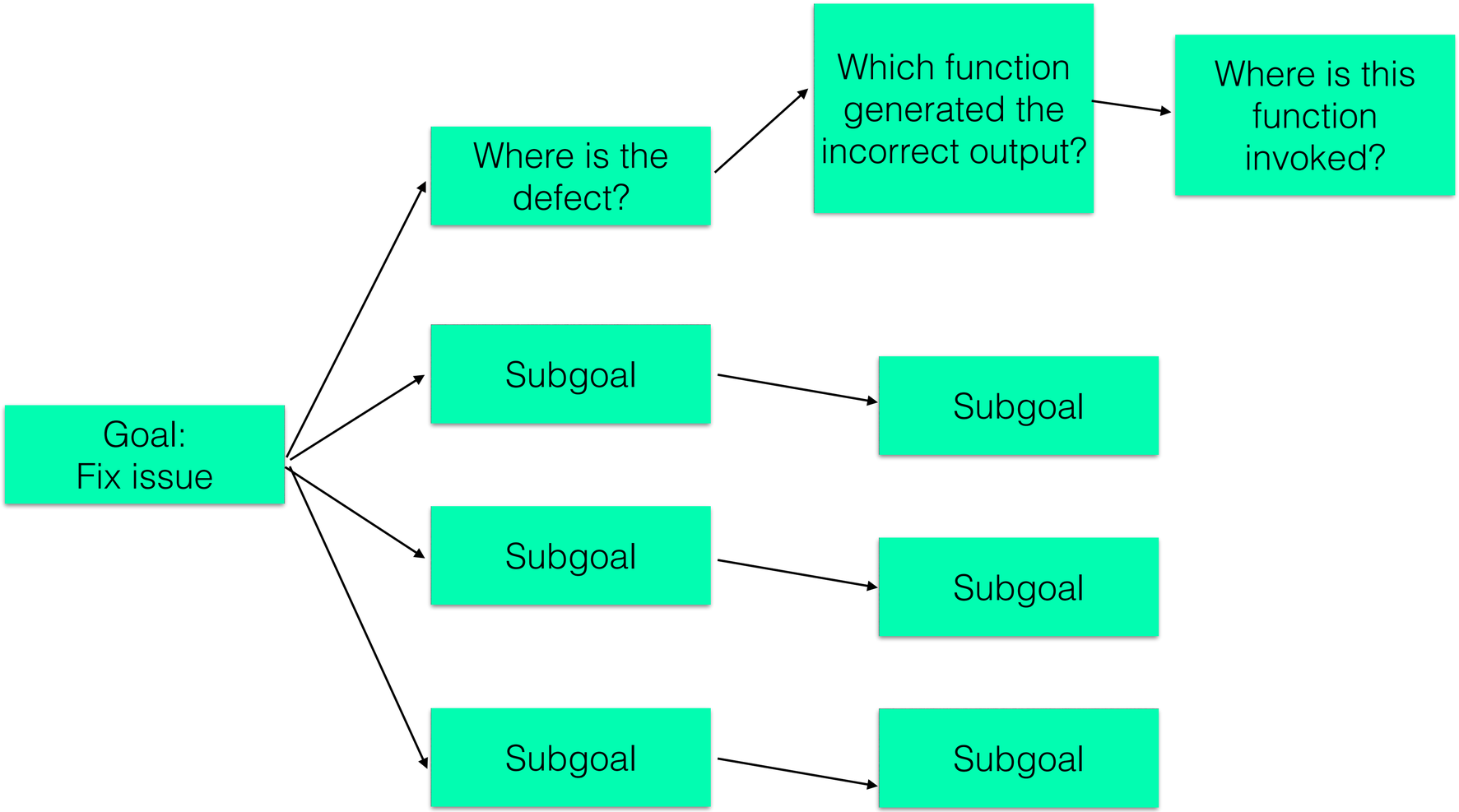
IDE

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

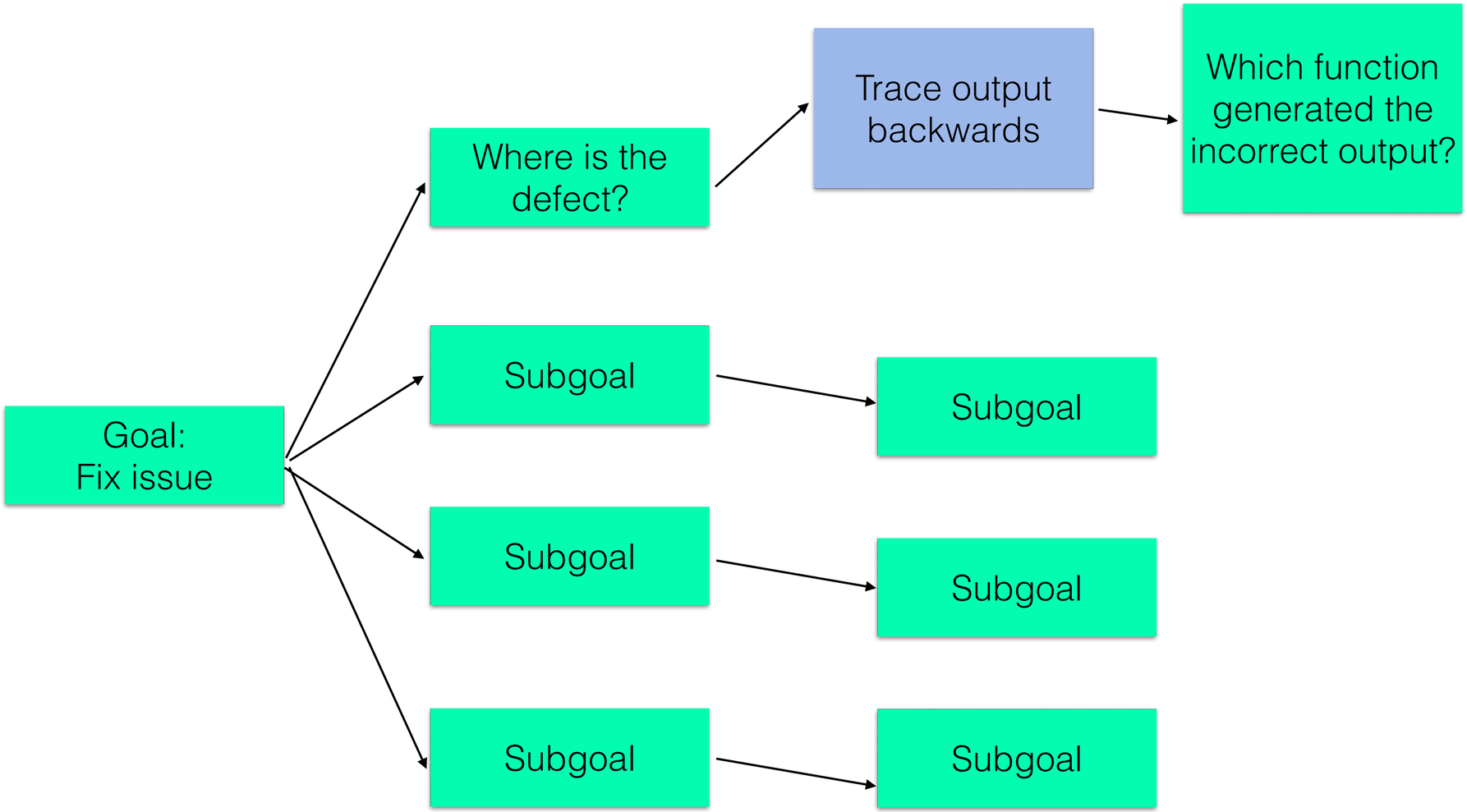
Problem solving is recursive



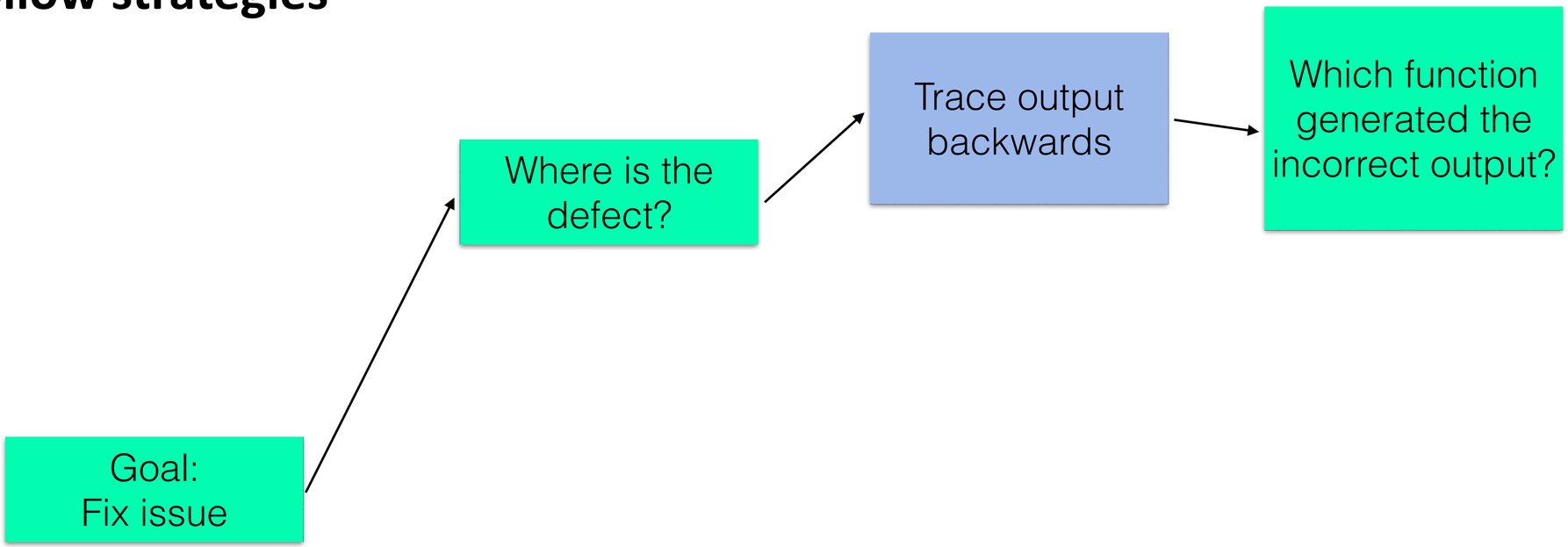
Problem solving involves answering questions



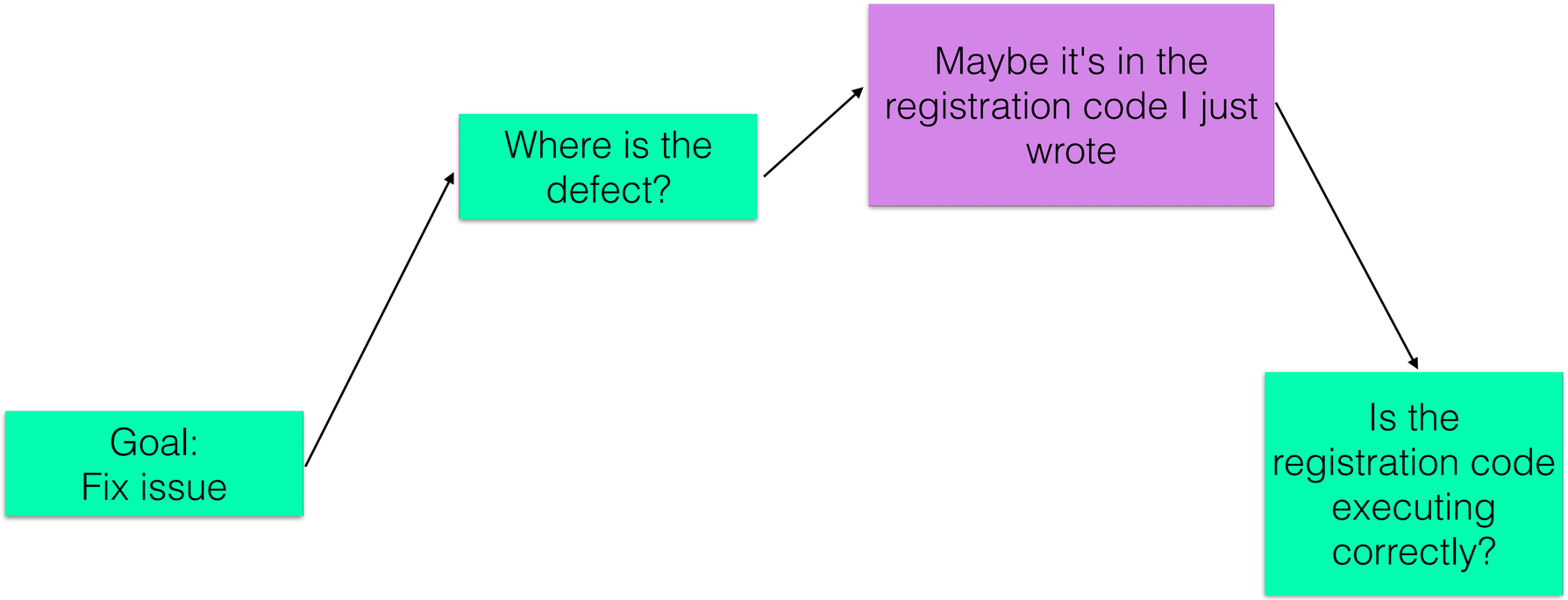
Problem solving involves strategies



Problem solving involves taking actions to answer questions and follow strategies



Problem solving involves formulating hypotheses



Studies of questions developers ask

Information Needs in Collocated Software Development Teams

Amy J. Ko
Human-Computer Interaction Institute
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh PA 15213
ajko@cs.cmu.edu

Robert DeLine and Gina Venolia
Microsoft Research
One Microsoft Way
Redmond, WA 98052
{rdeline, ginav}@microsoft.com

Abstract

Previous research has documented the fragmented nature of software development work. To explain this in more detail, we analyzed software developers' day-to-day information needs. We observed seventeen developers at a large software company and transcribed their activities in 90-minute sessions. We analyzed these logs for the information that developers sought, the sources that they used, and the situations that prevented information from being acquired. We identified twenty-one information types and cataloged the outcome and source when each type of information was sought. The most frequently sought information included awareness about artifacts and coworkers. The most often deferred searches included knowledge about design and program behavior, such as why code was written a particular way, what a program was supposed to do, and the cause of a program state. Developers often had to defer tasks because the only source of knowledge was unavailable coworkers.

1. Introduction

Software development is an expensive and time-intensive endeavor. Projects ship late and buggy, despite developers' best efforts, and what seem like simple projects become difficult and intractable [2]. Given the complex work involved, this should not be surprising. Designing software with a consistent vision requires the consensus of many people, developers exert great efforts at understanding a system's dependencies and behaviors [11], and bugs can arise from large chasms between the cause and the symptom, often making tools inapplicable [6].

One approach to understanding why these activities are so difficult is to understand them from an information perspective. Some studies have investigated information sources, such as people [13], code repositories [5], and bug reports [16]. Others have studied means of acquiring information, such as email, instant messages (IM), and informal conversations [16]. Studies have even characterized developers' strategies [9], for example, how they decide whom to ask for help.

While these studies provide several concrete insights about aspects of software development work, we still know little about what information developers look for and why they look for it. For example, what information do developers use to triage bugs? What knowledge do developers seek from their coworkers? What are developers looking for when they search source code or use a debugger? By identifying the types of information that developers seek, we might better understand what tools, processes and practices could help them more easily find such information.

To understand these information needs in more detail, we performed a two-month field study of software developers at Microsoft. We took a broad look, observing 17 groups across the corporation, focusing on three specific questions:

- What information do software developers' seek?
- Where do developers seek this information?
- What prevents them from finding information?

In our observations, we found several information needs. The most difficult to satisfy were design questions: for example, developers needed to know the intent behind existing code and code yet to be written. Other information seeking was deferred because the coworkers who had the knowledge were unavailable. Some information was nearly impossible to find, like bug reproduction steps and the root causes of failures.

In this paper, we discuss prior field studies of software development, and then describe our study's methodology. We then discuss the information needs that we identified in both qualitative and quantitative terms. We then discuss our findings' implications on software design and engineering.

2. Related Work

Several previous studies have documented the social nature of development work. Perry, Staudenmayer and Votta reported that over half of developers' time was spent interacting with coworkers [15]. Much of this communication is to maintain awareness. De Souza, Redmiles, Penix and Sierhuis found that developers send emails before check-ins to allow their peers to prepare for

Asking and Answering Questions during a Programming Change Task

Jonathan Sillito, *Member, IEEE*,
Gail C. Murphy, *Member, IEEE*, and Kris De Volder

Abstract—Little is known about the specific kinds of questions programmers ask when evolving a code base and how well existing tools support those questions. To better support the activity of programming, answers are needed to three broad research questions: 1) What does a programmer need to know about a code base when evolving a software system? 2) How does a programmer go about finding that information? 3) How well do existing tools support programmers in answering those questions? We undertook two qualitative studies of programmers performing change tasks to provide answers to these questions. In this paper, we report on an analysis of the data from these two user studies. This paper makes three key contributions. The first contribution is a catalog of 44 types of questions programmers ask during software evolution tasks. The second contribution is a description of the observed behavior around answering those questions. The third contribution is a description of how existing deployed and proposed tools do, and do not, support answering programmers' questions.

Index Terms—Change tasks, software evolution, empirical study, development environments, programming tools, program comprehension.

1 Introduction

Little is known about the specific kinds of questions programmers ask when evolving a code base and how well existing and proposed tools support those questions. Some previous work has focused on developing models of program comprehension, which are descriptions of the cognitive processes a programmer uses to build an understanding of a software system (e.g., [50], [34]). Other work has focused on how programmers perform change tasks, including how programmers use tools in that context (e.g., [13], [54]). These previous efforts do not consider in detail what a programmer needs to know about a code base when performing a change task, how the programmer finds that information, nor how well tools support those activities.

To address this gap, we undertook two qualitative studies. In each of these studies, we observed programmers making source changes to medium (20 KLOC) to large-sized (over 1 million LOC) code bases. To structure our data collection and the analysis of our data, we used a *grounded theory* approach [16], [63]. Based on our analysis of the data from these user studies, as well as an analysis of the support that current programming tools provide for these activities, this research makes three key contributions. The first contribution is a catalog of 44 types of questions programmers ask, organized into four categories based on the kind and scope of information needed to answer a question. The second contribution is a description of the behavior we observed around answering those questions. The third contribution is a description of how well tools support a programmer in answering questions. Based on these results, we discuss the support that is missing from existing programming tools.

Section 2 of this paper compares the work presented in this paper to previous efforts in the area of program comprehension and empirical studies of how programmers manage change tasks. Section 3 describes the two studies we performed. Section 4 presents the 44 types of questions organized around four top-level categories and a description of the behavior we observed around answering questions. Section 5 considers the support existing research and industry tools provide for those activities. In Section 6, we discuss gaps in tool support. In Section 7, we discuss the limits of our results. We conclude with a summary in Section 8.

2 Related Work

In this section, we discuss three categories of related work. The first is the area of program comprehension, in particular efforts to use theories about program comprehension to inform tool design (see Section 2.1). The second covers work involving the analysis of programmers' questions (see Section 2.2). The third category includes empirical studies that have looked at how programmers use tools and generally how they carry out change tasks and other programming activities (see Section 2.3). Our review of these studies includes a discussion of studies that

- *J. Sillito is with the Department of Computer Science, University of Calgary, 2500 University Dr. NW, Calgary, AB, T2N 1N4 Canada. E-mail: sillito@ucalgary.ca.*
- *G.C. Murphy and K. De Volder are with the Department of Computer Science, University of British Columbia, ICICS/CS Building, 201-2366 Main Mall, Vancouver, BC, V6T 1Z4 Canada. E-mail: {murphy, kdvolder}@cs.ubc.ca.*

Manuscript received 11 May 2007; revised 24 Nov. 2007; accepted 27 Mar. 2008; published online 21 Apr. 2008.
Recommended for acceptance by M. Young and P. Devanbu.

Asking and Answering Questions during a Programming Change Task in Pharo Language

Juraj Kubelka Alexandre Bergel Romain Robbes
PLEIAD Laboratory, Department of Computer Science (DCC)
University of Chile, Santiago, Chile
{jkubelka,abergel,rrobbes}@dcc.uchile.cl

Abstract

Previous studies focus on the specific questions software engineers ask when evolving a codebase. Though these studies observe developers using statically typed languages, little is known about the developer questions using dynamically typed languages. Dynamically typed languages present new challenges to understanding and navigating in a codebase and could affect results reported by previous studies.

This paper replicates a previous study and presents the analysis of six programming sessions made in Pharo, a dynamically typed language. We found a similar result when comparing sessions on an unfamiliar codebase with the previous work. Our result on the familiar code greatly deviates from the replicated study, likely caused by different tasks and development strategies. Both missing type information and test driven development affected participant behavior and prudence on codebase understanding, where some participants made changes based on assumptions.

We provide a set of questions that are useful in characterizing activity related to the use of a dynamically typed language and test-driven development — questions not explicitly considered in previous research. We also present a number of issues that we would like to discuss during the PLATEAU workshop.

1. Introduction

Programming environments have tremendously improved over the last decade. What were previously simple text editors are now fully fledged studios for code production. Navigating between source code elements is now supported in many different ways by most programming environments.

PLATEAU '2014, October 21, 2014, Portland, Oregon, United States.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

Sillito *et al.* [9] (herein designated as *Sillito*) made a number of observations on developer navigation. They identify four question categories and levels of tool support for getting answers. They conducted two studies observing software programmers of statically typed languages C++, C, C#, and Java. In their first study, the participants worked on a change task for one unique open source project, ArgoUML[1], of which they were not familiar. The second study was conducted in an industrial setting including software engineers working on a change task of familiar codebase. The context setting used by Sillito in their experiment does not cover some commonly found software engineering practices. For example, they only consider statically typed languages, one industrial codebase, and one open source codebase.

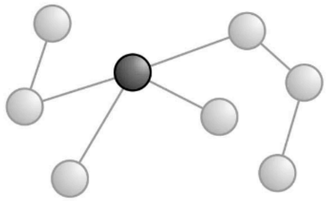
Research question. Our work replicates the experiment by Sillito *et al.* and validates it in a new scenario. The participants worked on tasks in Pharo, a dynamically typed programming language, and in distinct open source software systems. The dynamically typed languages present new challenges to understanding and navigating in a codebase. Both aspects — dynamically typed language and different codebases could affect results reported by Sillito. In summary, our research question is:

> *Are findings presented by Sillito applicable to programming change tasks using the Pharo programming language?*

Pharo. The Pharo[2] environment (Pharo IDE) illustrated in Figure 1 is largely different from the ones considered in the Sillito experiment. The Pharo programming environment offers a set of expressive and flexible programming tools. The System Browser (2) is the main tool for writing and reading source code. Navigation within the source code is essentially based on the SendersOf (4), ImplementorOf, and UsersOf tools; whenever a user asks to where a particular method is called, or asks for method definition, field reference, or class

[1] `http://argouml.tigris.org`, verified September 2014
[2] `http://pharo.org`, verified September 2014

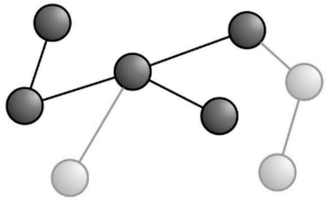
Developers ask different types of questions



Finding focus points

5 kinds of questions.

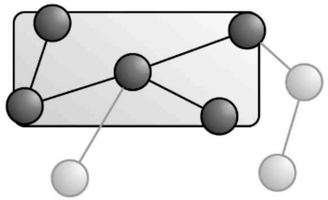
For example: Which type represents this domain concept?



Expanding focus points

15 kinds of questions.

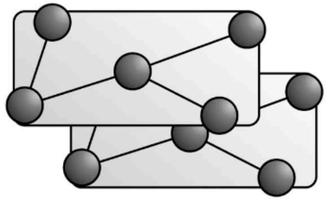
For example: Which types is this type a part of?



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?

J. Sillito, G. C. Murphy and K. De Volder, "Asking and Answering Questions during a Programming Change Task," in *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 434-451, July-Aug. 2008, doi: 10.1109/TSE.2008.26.

Finding Focus Points

1. Which type represents this domain concept or this UI element or action? (1.1, 1.2, 1.3, 1.5, 1.6, 1.7, 1.8, 1.9)
2. Where in the code is the text in this error message or UI element? (1.1, 1.5, 1.9)
3. Where is there any code involved in the implementation of this behavior? (1.1, 1.2, 1.3, 1.5, 1.6, 1.10, 1.11, 2.11, 2.13)
4. Is there a precedent or exemplar for this? (1.1, 1.10, 1.12, 2.4, 2.6, 2.14, 2.15)
5. Is there an entity named something like this in that unit (project, package, or class, say)? (1.1, 1.2, 1.4, 1.5, 1.6, 1.10, 2.9)

J. Sillito, G. C. Murphy and K. De Volder, "Asking and Answering Questions during a Programming Change Task," in *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 434-451, July-Aug. 2008, doi: 10.1109/TSE.2008.26.

Expanding focus points

12. Where is this method called or type referenced? (1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.10, 1.11, 1.12, 2.1, 2.4)
13. When during the execution is this method called? (1.2, 1.4, 1.5, 2.15)
14. Where are instances of this class created? (1.2, 1.3, 1.5, 1.7, 1.8, 1.10)
15. Where is this variable or data structure being accessed? (1.4, 1.5, 1.6, 1.7, 1.12, 2.1, 2.8, 2.14)
16. What data can we access from this object? (1.8, 2.15)
17. What does the declaration or definition of this look like? (1.2, 1.5, 1.8, 1.10, 1.11, 2.1, 2.2, 2.11, 2.13, 2.15)
18. What are the arguments to this function? (1.3, 1.4, 1.5, 1.7, 1.8, 1.10, 1.11, 1.12)
19. What are the values of these arguments at runtime? (1.4, 1.9, 1.12, 2.15)
20. What data is being modified in this code? (1.6, 1.11)

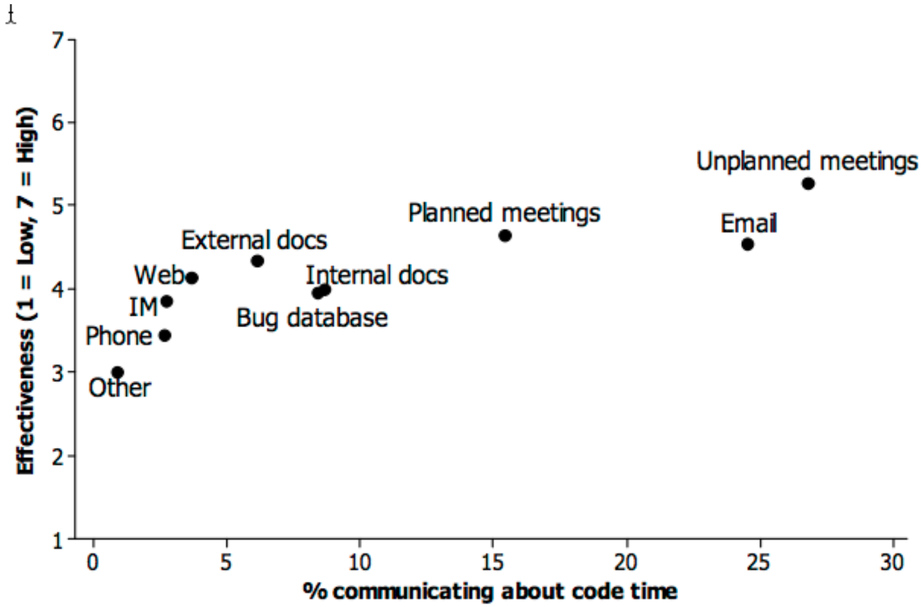
Understadning a Subgraph

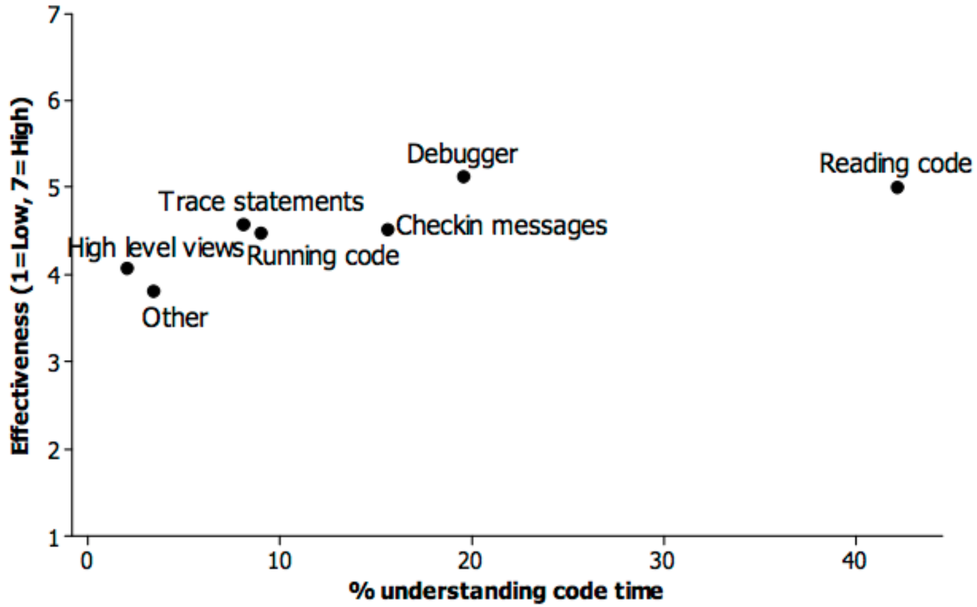
21. How are instances of these types created and assembled? (1.1, 1.2, 1.4, 1.7, 1.9, 1.10, 1.11, 1.12)
22. How are these types or objects related? (whole-part) (1.2, 1.10)
23. How is this feature or concern (object ownership, UI control, etc.) implemented? (1.1, 1.2, 1.4, 1.7, 1.11, 1.12, 2.1, 2.13)
24. What in this structure distinguishes these cases? (1.2, 1.12, 2.8)
25. What is the behavior that these types provide together and how is it distributed over the types? (1.1, 1.2, 1.3, 1.4, 1.6, 1.11, 2.11)
26. What is the "correct" way to use or access this data structure? (1.8, 2.3, 2.15)
27. How does this data structure look at runtime? (1.8, 1.9, 1.10, 2.15)
28. How can data be passed to (or accessed at) this point in the code? (1.5, 1.6, 1.8, 1.12, 2.14)
29. How is control getting (from here to) here? (1.3, 1.4)
30. Why is not control reaching this point in the code? (1.4, 1.9, 1.10, 1.12, 2.1, 2.10)
31. Which execution path is being taken in this case? (1.2, 1.3, 1.7, 1.9, 1.12, 2.2, 2.9)
32. Under what circumstances is this method called or exception thrown? (1.3, 1.4, 1.5, 1.9)
33. What parts of this data structure are accessed in this code? (1.6, 1.8, 1.12)

Questions over groups of subgraphs

34. How does the system behavior vary over these types or cases? (1.3, 1.4, 2.14)
35. What are the differences between these files or types? (1.2, 2.1, 2.2, 2.13, 2.15)
36. What is the difference between these similar parts of the code (e.g., between sets of methods)? (1.7, 1.8, 1.11, 2.6, 2.11, 2.14, 2.15)
37. What is the mapping between these UI types and these model types? (1.1, 1.2, 1.5, 1.7)
38. Where should this branch be inserted or how should this case be handled? (1.4, 1.5, 1.6, 1.8, 1.9, 2.11, 2.15)
39. Where in the UI should this functionality be added? (1.1, 1.5, 1.7, 2.1, 2.6)
40. To move this feature into this code, what else needs to be moved? (2.7, 2.13)
41. How can we know that this object has been created and initialized correctly? (1.10, 1.12)
42. What will be (or has been) the direct impact of this change? (1.5, 1.7, 1.8, 1.10, 1.11, 1.12, 2.1, 2.2, 2.4, 2.6, 2.7, 2.8, 2.12, 2.15)
43. What will the total impact of this change be? (2.1, 2.2, 2.3, 2.4, 2.5, 2.9, 2.10, 2.11)
44. Will this completely solve the problem or provide the enhancement? (1.1, 1.9, 1.11, 2.12, 2.14)

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





Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining mental models: a study of developer work habits. In Proceedings of the 28th international conference on Software engineering (ICSE '06), Experience Report, 492–501. https://doi.org/10.1145/1134285.1134355

What makes understanding code hard?

- Questions developers ask about code that are hard to answer.
- May require substantial time and effort to answer.
- May lead to many other questions to answer

Longest investigation activities

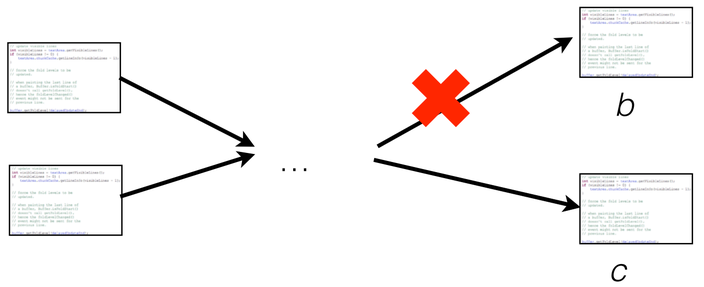
	Time (mins)
How is this data structure being mutated in this code?	83
“Where [is] the code assuming that the tables are already there?”	53
How [does] application state change when *m* is called denoting startup completion?	50
What decisions might be incompatible with reuse in new context?	24
“Is [there] another reason why *status* could be non-zero?”	11

Thomas D. LaToza and Brad A. Myers. 2010. Developers ask reachability questions. In International Conference on Software Engineering, 185–194. https://doi.org/10.1145/1806799.1806829

Failure in information needs

- Developers guess and make assumptions about answers to questions, and sometimes are wrong, leading to defects.

False belief held by developer	**Correct fact about control flow**
Method *a* need not call method *b*, as all calls to be are redundant.	*m* is called in several additional situations in which *n* has not been called.



Thomas D. LaToza and Brad A. Myers. 2010. Developers ask reachability questions. In International Conference on Software Engineering, 185–194. https://doi.org/10.1145/1806799.1806829

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)

What does this do?
What does these functions do?

What does this do in this case?
What happens if an exception is thrown?
What happens if this operation times out?
What happens if the remote service is slow?

What is the intent of the code?
What is it trying to accomplish?

How does it implement this behavior?
How is this data aggregated and how is it translated from one place to another.
How does this class (or collection of classes) fulfill the functional feature of the application?

Thomas D. LaToza and Brad A. Myers. 2010. Hard-to-answer questions about code. In Evaluation and Usability of Programming Languages and Tools (PLATEAU '10), 1–6. https://doi.org/10.1145/1937117.1937125

What depends on this code or design decision?

What else depends on this code?
Who else uses this code / function. (i.e. If we change this, what will break simply because someone else has found a way to use this and we don't even know they are doing so...)

What are the implications of this change for API clients, security, concurrency, performance, platforms, tests, or obfuscation?

What is the implication of these changes in terms of the backward compatibility?
Across components with a code base the size of complete applications, what are the implications of a functional change in base storage to all accessors in the system (including clients of applications built on top of the place where the change is occurring)

How can I refactor this without breaking existing users?

How can I refactor this piece w/o causing an avalanche of new places to refactor?

Thomas D. LaToza and Brad A. Myers. 2010. Hard-to-answer questions about code. In Evaluation and Usability of Programming Languages and Tools (PLATEAU '10), 1–6. https://doi.org/10.1145/1937117.1937125

Why was it done this way?

Why was this code structured in this way?
Why was this done this way? Is there some reason for this ancient code doing what it does that I'm missing?

Why wasn't it done this other way?

Why didn't they use this method/object/interface as it appears to have been designed?
Why did the original developer not use library function X? (was there a good reason or just ignorance)

Was this intentional, accidental, or a hack?

Is the lack of parameter validation (most often lack of null checks) intentional or incidental?
Is the lack of ''sealed'' on the class intentional or incidental? If intentional, why? (assuming no virtual methods are present).

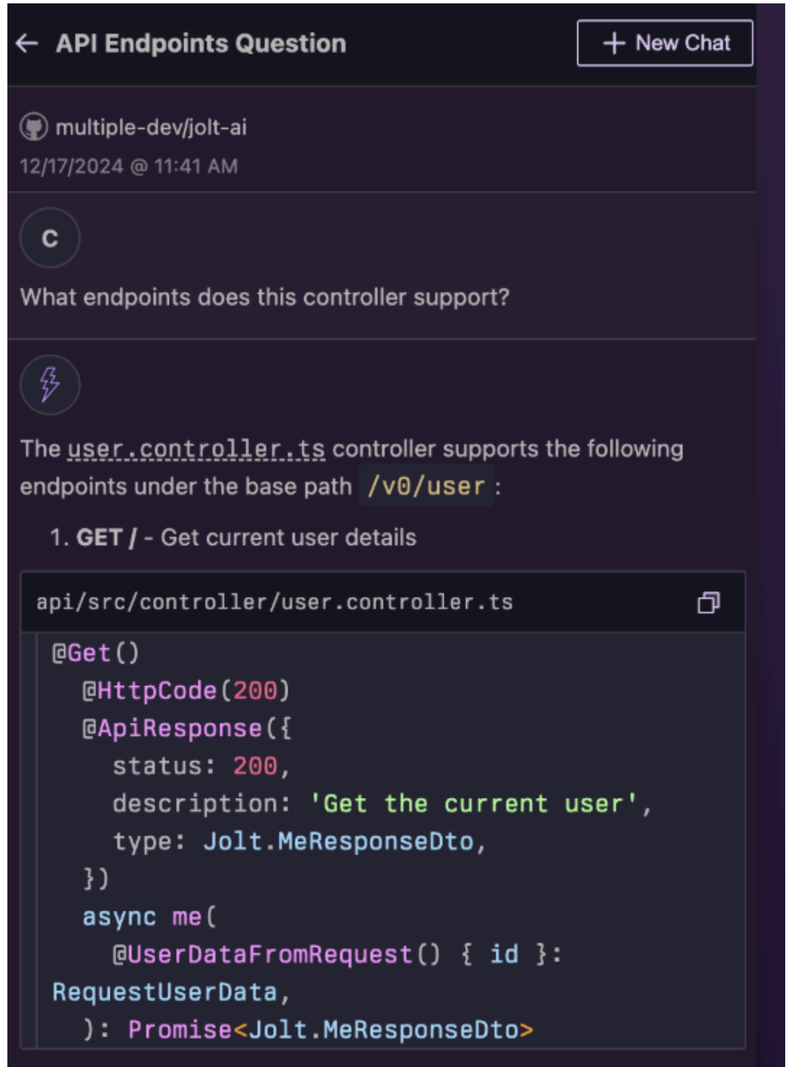
Thomas D. LaToza and Brad A. Myers. 2010. Hard-to-answer questions about code. In Evaluation and Usability of Programming Languages and Tools (PLATEAU '10), 1–6. https://doi.org/10.1145/1937117.1937125

Understanding Code & LLMs

Examples of answering questions with an LLM

Chatting with agent

- Agentic LLMs can answer questions about code
  - Uses tools (grep) to find relevant context
  - Summarizes and synthesizes context
  - Decides what else it should read



Challenges answering questions about code

- If identifiers are poor, code is poorly organized, grep may not be effective in finding focus points
- Missing docs or out of date design docs that are misleading
- Need access to tools to access code history, slack, design docs, other non-code resources
- Connecting quality attributes to code: may or may not effectively understand what quality attributes motivated code to be written as it is
- Out of distribution: design that is inconsistent with standard practices

10 min break

In-Class Activity

- In groups of 1 or 2, try to find questions your tool (Cursor or Claude Code) **can't** answer
  - Start with the codebase you used for last time (Lecture 6)
  - Goal: find questions the tool doesn't seem to create good answers for
  - Start with simpler questions, works towards more complex questions
    - Free to use example of questions from slides today or make up your own
- Want to hear about each of the types of questions you tried, what you expected, what worked, and what didn't
- Deliverables
  - Screen recording through Kaltura
    - Upload to OneDrive, turn on link sharing, share link in Lecture 7 activity submission on Canvas
    - Submit answers to questions on your experiences on Canvas (next slide)
- Aim to finish by 7:10pm today; Due tomorrow at 4:30pm

Questions to answer

- List all of the different questions you tried
- Which did the LLM seem to answer correctly? Which did the answer seem to be wrong? Were there answers where you didn't know what was right or wrong?
- What were the types of questions where the LLM was most helpful?
- What were the types of questions where the LLM was least helpful?
- To what extent was the LLM able to tell you when it didn't know an answer?

- **Deliverable**: Submit through Canvas, at least a page