Programming as Communication

SWE 795, Fall 2019

Software Engineering Environments

Today

- Part 1 (Lecture)(~40 mins)
  - Programming as communication

- Part 2 (Project Presentations, Part 1)(~40 mins)

- Break

- Part 3 (Project Presentations, Part 2)(~60 mins)

HW2: Study of Current Practice

- Revise study proposal based on feedback received on HW1.
- Conduct study(s) to gather data.
- Analyze data to describe a *challenge* that developers face in their programming work.

- 15 min in-class presentation
- Due in 2 weeks on 10/7

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));
```

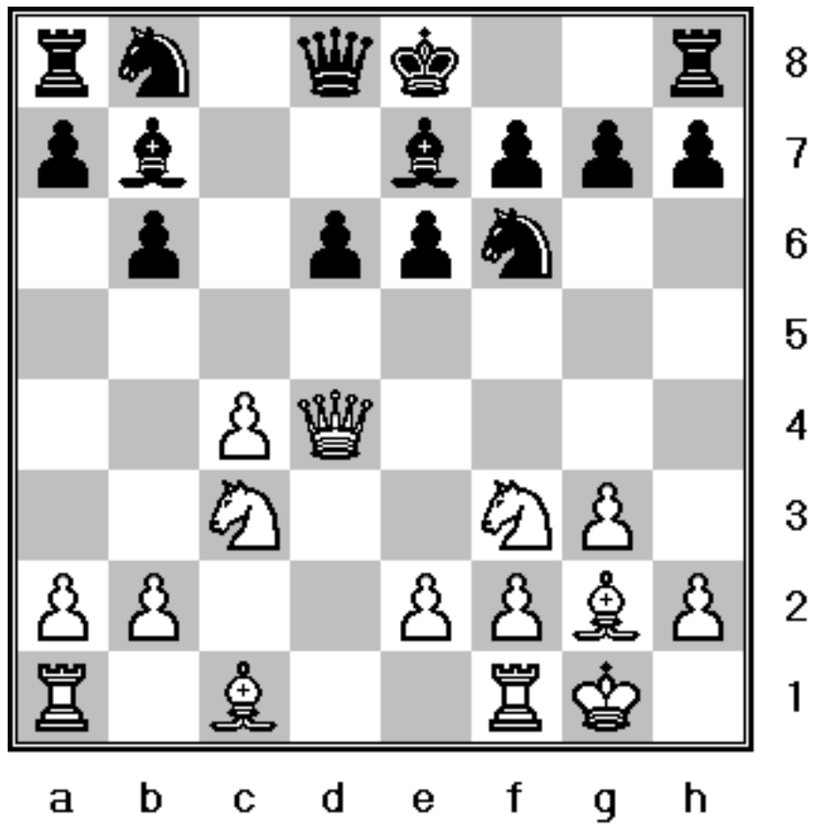
Demo: Remember this code (10 seconds)

```
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

Chunking

- Items in memory encoded as **chunks**
- A chunk may be anything that has meaning
- # of chunks in STM fixed, but remembering bigger chunks lets you remember more
- Memory retention relative to the concepts you already have

Chunking: What's easiest to remember?

- A lock combination with 8 numbers in order: 10, 20, 30, 40, 50, 60, 70, 80
- A lock combination with 8 numbers in order: 50, 30, 60, 20, 80, 10, 40, 70
- A string of 10 letter: R, P, L, B, V, Q, M, S, D, G
- A string of 52 letters: I pledge allegiance to the flag of the United State of America.

Short term memory (STM)

- Primary, active memory used for holding current context for System 2
- Unless actively maintained (or encoded to long-term memory), decays after seconds
- Capacity ~ 4 items
- (classic estimate of 7 +/- 2 is wrong)

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

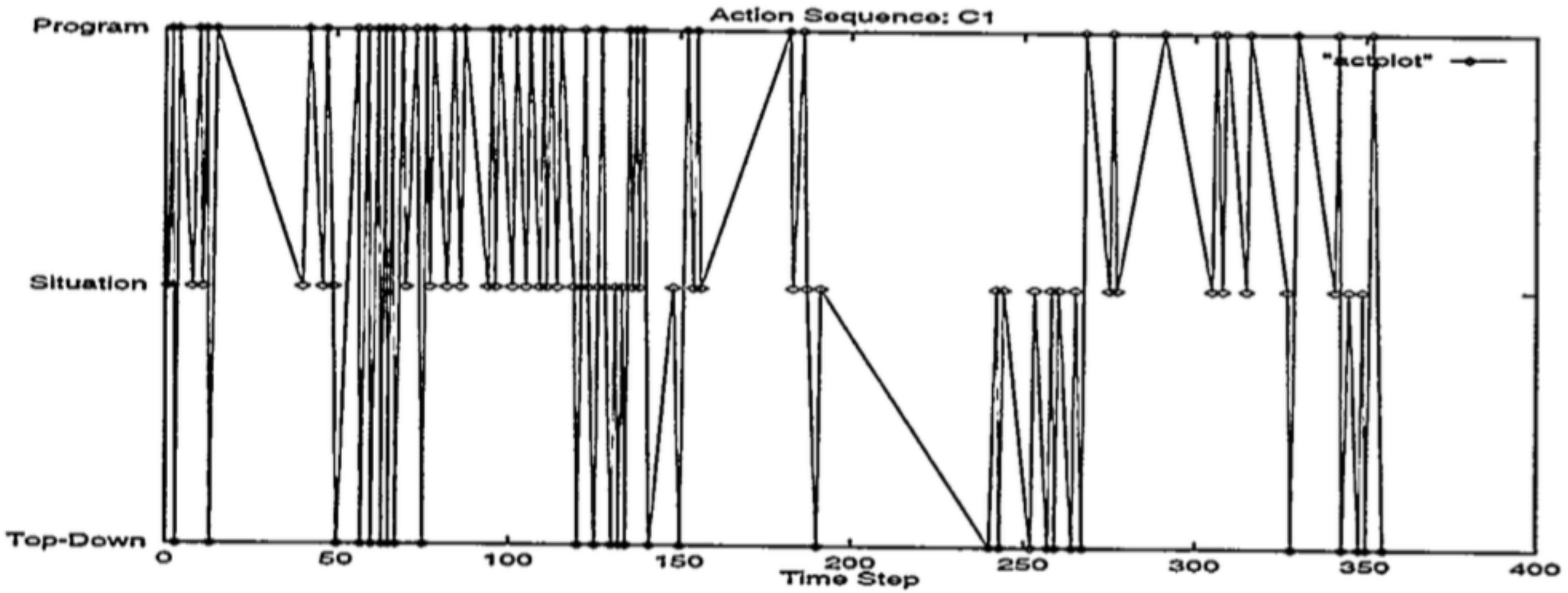
"What it did was it…computes the new line number and fires an event. But I didn't see it change any state." *(38 mins, 10 mins reading getFoldLevel)*

"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

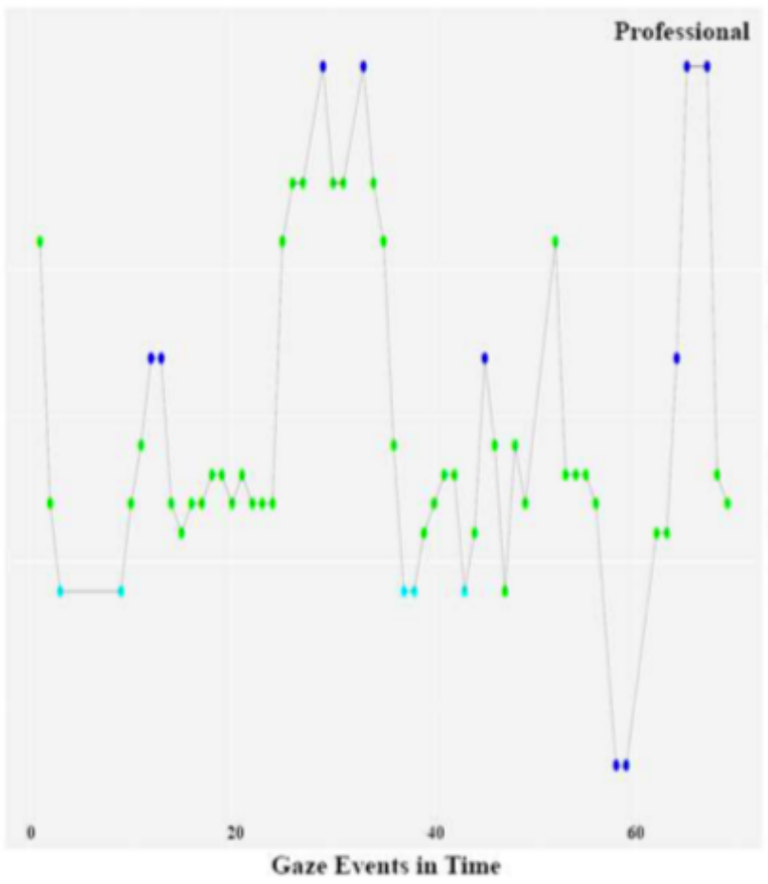


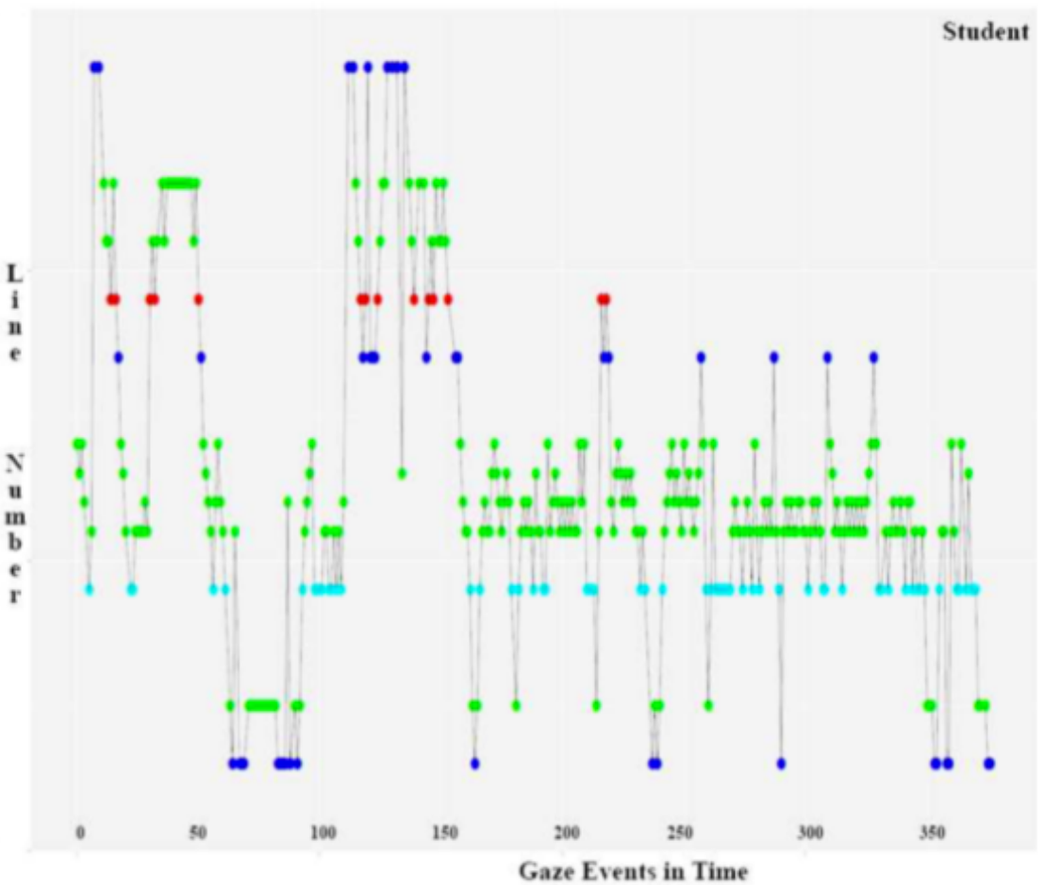
Anneliese von Mayrhauser and A. Marie Vans. 1997. Program understanding behavior during debugging of large scale software. In Papers presented at the seventh workshop on Empirical studies of programmers (ESP '97), Susan Wiedenbeck and Jean Scholtz (Eds.). ACM, New York, NY, USA, 157-179. DOI=http://dx.doi.org/10.1145/266399.266414

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 }
```





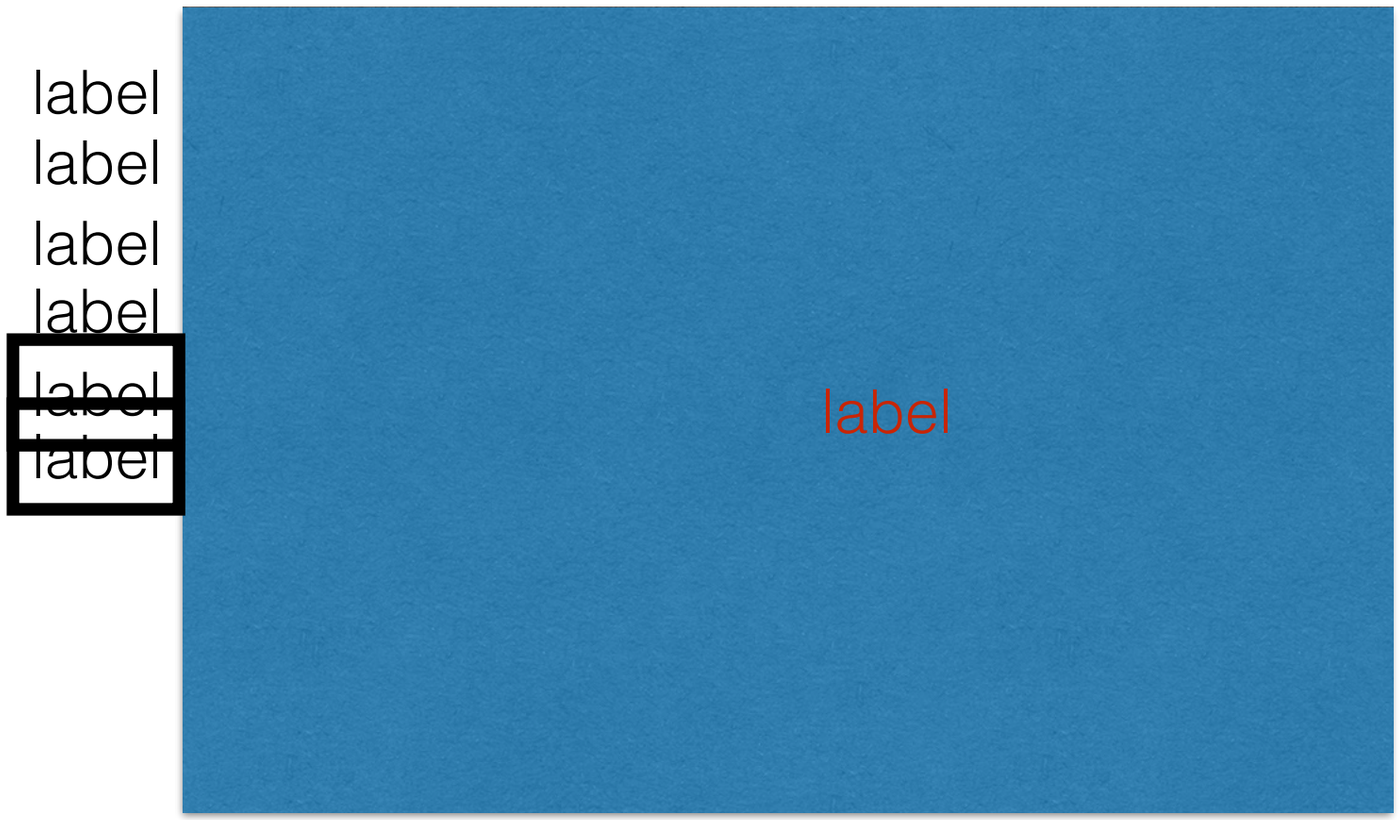
Katja Kevic, Braden M. Walters, Timothy R. Shaffer, Bonita Sharif, David C. Shepherd, and Thomas Fritz. 2015. Tracing software developers' eyes and interactions for change tasks. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). ACM, New York, NY, USA,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

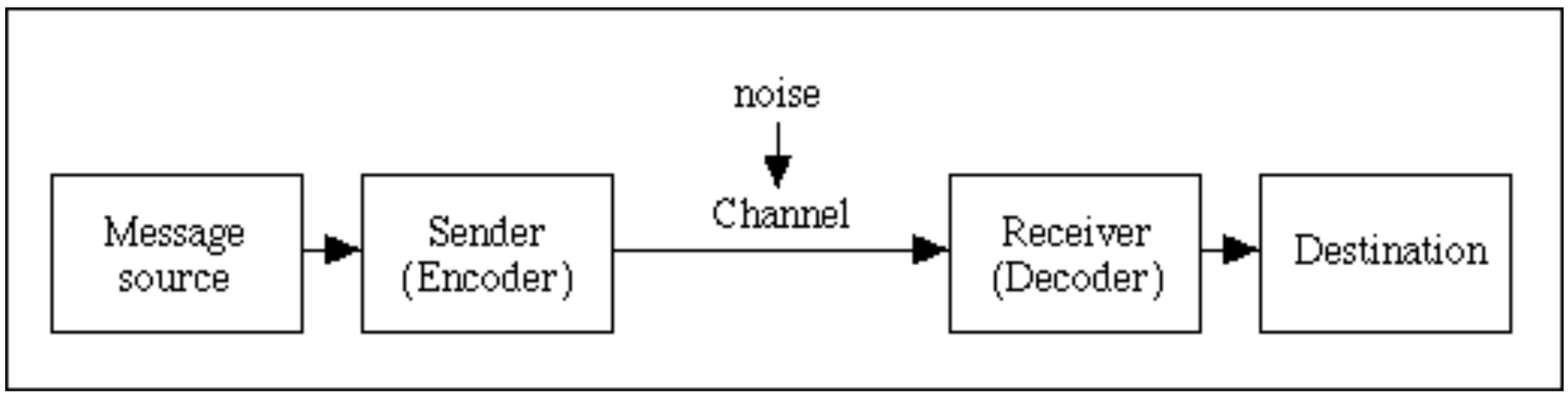
Programming as communication



label label

Programming as communication

- Goal: **efficiently** & accurately transmit information from environment to developer
- Developers may choose between which evidence they would like to consider.



Example: Reuse

- Goal: display data to the user.
- Can m_k do this?
- Some potential strategies
  - Read its description
  - Read its implementation
  - Understand how it is typically used by calling code
  - Read documentation
  - Conduct a web search for tutorials and discussion
  - Call it and see what happens
- When is there enough evidence to decide that this is the function to use?

```
1  function f(p)
2  {
3    ...
4    p.m_k(x);
5    ...
6  }
7
8  function m_k(y)
9  {
10   ...
11   if (this.f)
12       doDataDisplay(data);
13   ...
14 }
```

Figure 1. Whenever a developer considers invoking a function `m_k` to achieve a desired behavior (e.g., displaying data), they must reason about what constraints they must follow to achieve this behavior.

Example: Debugging

- Calls it, does not work.
- Goal: debug why expected output is not generated
- Some potential strategies
  - Set a breakpoint and step through in debugger
  - Add log statements
  - Compare against other callers of function
  - Check for output being overwritten
  - Try a different function

```
1  function f(p)
2  {
3    ...
4    p.m_k(x);
5    ...
6  }
7
8  function m_k(y)
9  {
10   ...
11   if (this.f)
12     doDataDisplay(data);
13   ...
14 }
```

Figure 1. Whenever a developer considers invoking a function `m_k` to achieve a desired behavior (e.g., displaying data), they must reason about what constraints they must follow to achieve this behavior.

Example: Testing

- Appears to work on example, will it always work?
- Goal: check if it will always work
- Some potential strategies
  - Manually execute program with different inputs
  - Build test suite
  - Read documentation to double check usage
  - Compare against other uses of function elsewhere

```
1  function f(p)
2  {
3    ...
4    p.m_k(x);
5    ...
6  }
7
8  function m_k(y)
9  {
10   ...
11   if (this.f)
12       doDataDisplay(data);
13   ...
14 }
```

Figure 1. Whenever a developer considers invoking a function `m_k` to achieve a desired behavior (e.g., displaying data), they must reason about what constraints they must follow to achieve this behavior.

Constraint Communication Theory

- Violating a constraint results in either observably **incorrect** behavior or code **decay** in which design decisions are violated.
- Developers accrue **information** about constraints by gathering and interpreting evidence through asking and answering questions.
- Developers choose from a variety of forms of evidence offered by alternative programming **methodologies**.
- Developers **choose** between methodologies based on their perception of the current programming context, expectations about what constraints exist, their beliefs about the efficiency of alternative methodologies, and habit.
- When developers experience a **barrier** which prevents use or increases the perceived cost of using a methodology, developers **shift** to using a different methodology.

Programming methodologies

- Give guidance about what evidence a developer should consider when reasoning about constraints
  - Should they read the description of the function or implementation?
  - Should they try to reason about the code before changing it or just modify it and see what happens?
- Offer normative guidance on what a developer ***should*** do to be effective
- May generate testable predictions about what actions are most effective

Some programming methodologies

- Design by contract
  - Specifying the behavior of a function through a contract enables developers to reason about a function through its **interface** rather than its implementation.
- Domain-driven design
  - Creating functions which correspond to operations on domain elements enables developers to reason about the behavior of a function by performing mental simulations of the **domain**.
- Information hiding
  - Hiding a design decision behind an **interface** enables developers to reason about the function oblivious to the decision.
- Example-centric programming
  - Developers may reason about a function through copy and paste reuse, identifying, selecting, and adapting code **examples**.
- Program slicing
  - Developers may reason about a function by navigating **slices** to understand its impact and why it has generated erroneous output.
- Live programming
  - Live programming enables developers to reason about a function by rapidly **varying** its inputs and observing its **output**.

Using programming methodologies

- Reuse, debugging, testing all supported by methodology
  - If you learn some evidence which sheds light on constraint in one step, will influence how behave in other steps

Example

- obj.a();
- ...
- obj.b();

- Suppose that there is a protocol constraint that *a* should always be invoked on *obj* before *b*. Consider a task in which a developer wishes to invoke *b*. How might a developer learn that they should first invoke *a*?

Example

- Imagine a look and feel constraint, where developer expects an animation to "look good" by proceeding at an appropriate rate. To satisfy this constraint, the developer must pass 42:
  - obj.animate(42);
- How might developers use a methodology to learn this constraint?