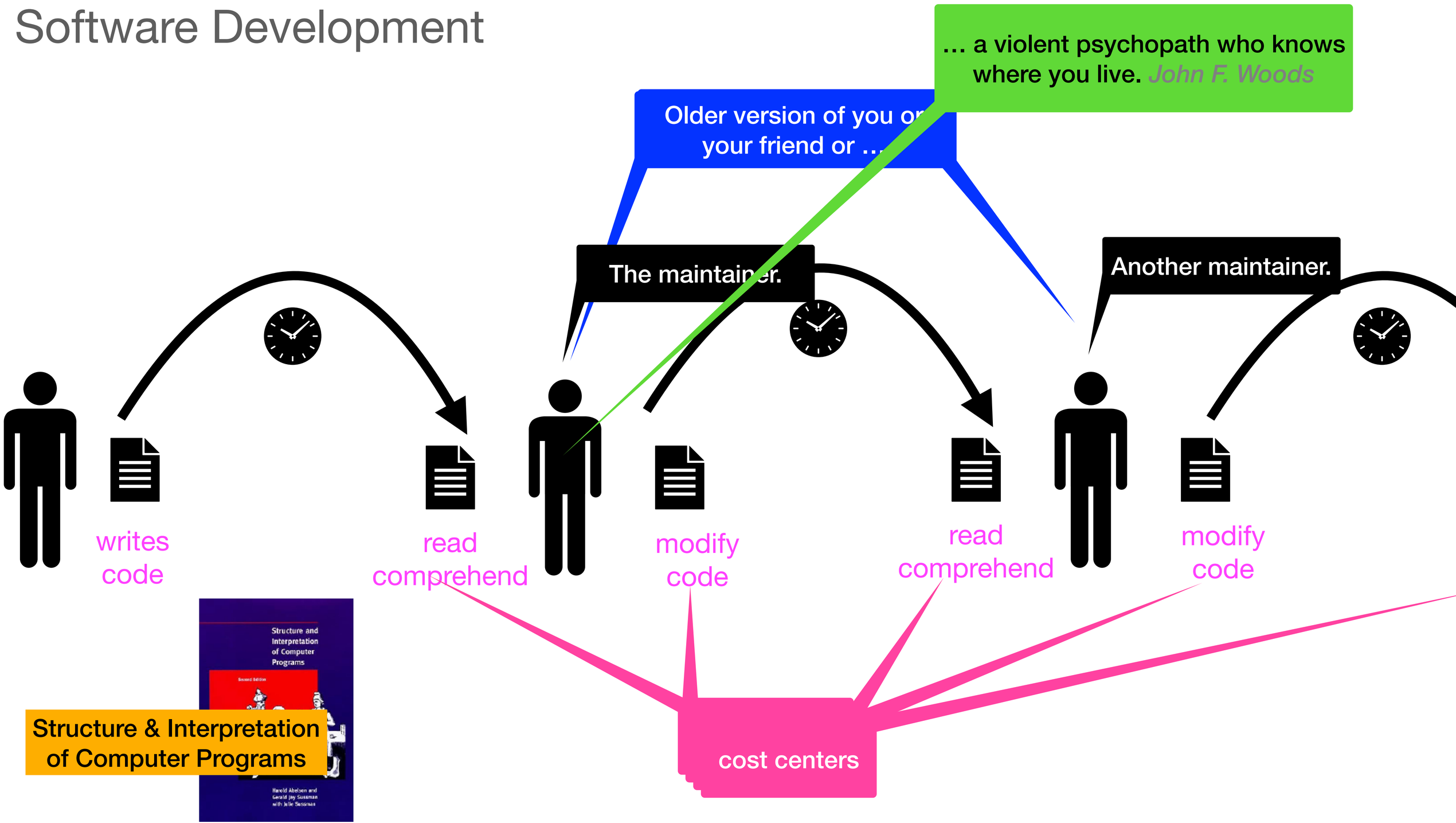
Socially Responsible Software Development

Matthias Felleisen, PLT

I, Me, Myself

- programming language researcher
- … who cares about *programming*
- founded PLT, which is behind the Racket language
- maintained student-facing sw (appr. 50-100 Kloc) for ~30 years
- developed a software development curriculum for ~25 years
- .. starting with an alternative programming curriculum for K12 and freshmen

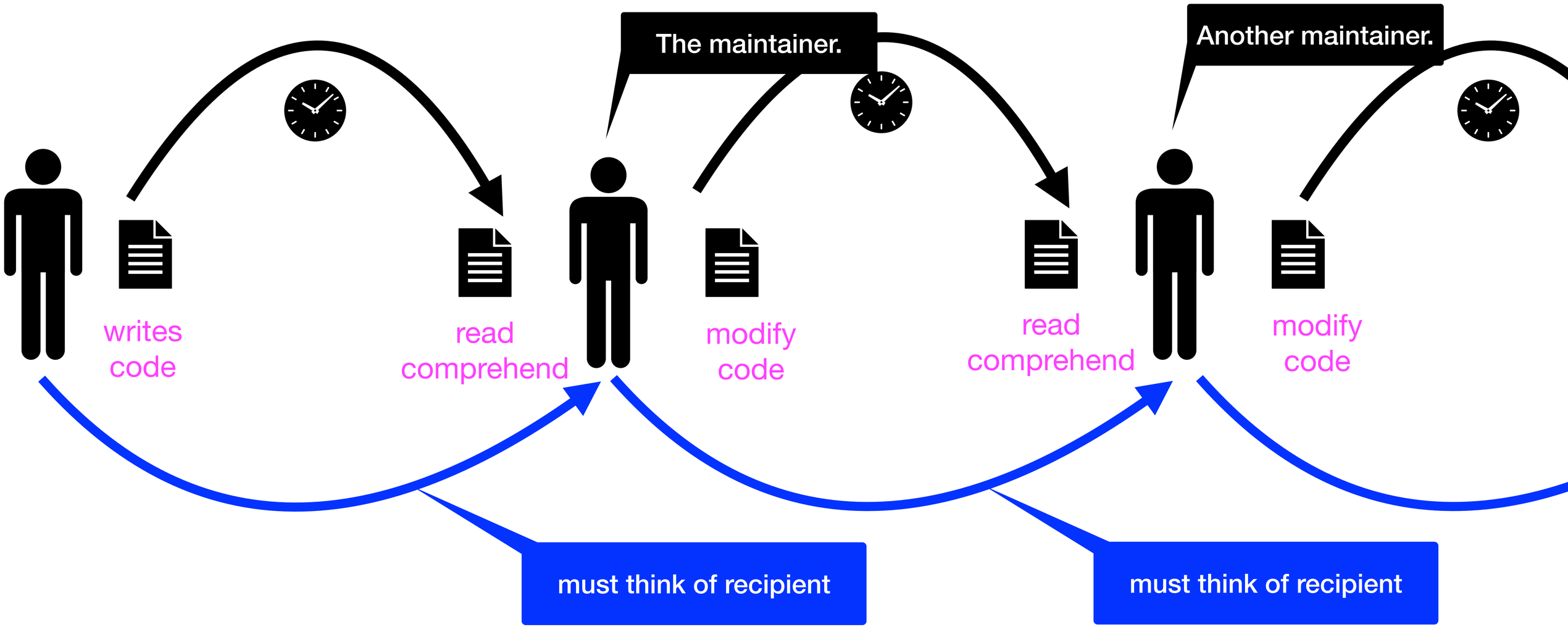
Software Development



Structure & Interpretation of Computer Programs

Socially Responsible Software Development

- reduced cost
- happier developers



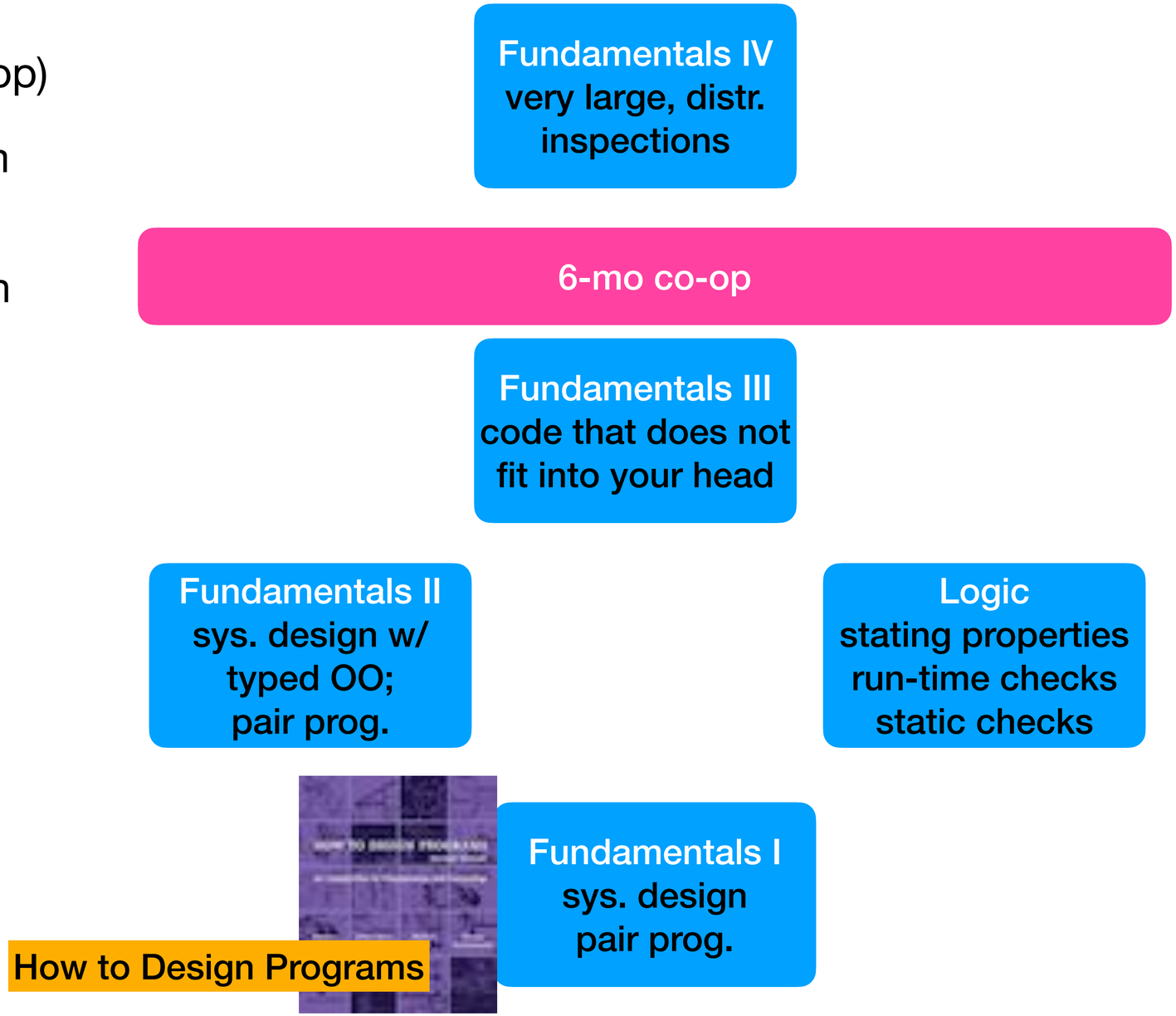
Socially Responsible Software Development

How do we get there?

- Social (“Soft”) Skills
  - Us
  - I
- Technical (“Hard”) Skills

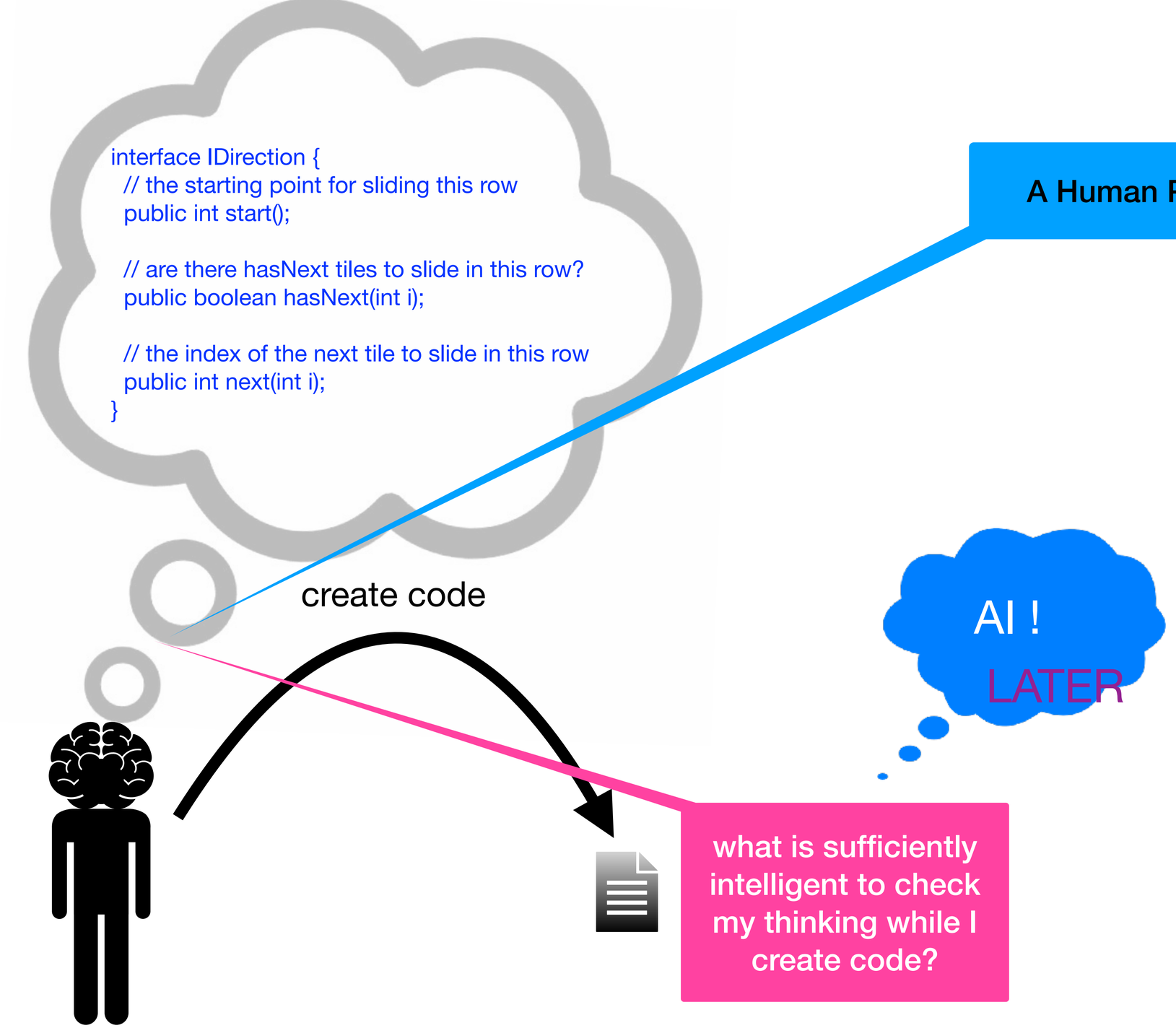
The Big Picture: How to turn novices into basic sw devs

- five core courses (plus one 6-month co-op)
- key ideas across all courses, scaled from 5-liners to 15Kloc per semester:
  - fundamentals are more important than currently fashionable industry ideas
  - design code systematically (techn. or “hard” skills)
  - programming is a people discipline (social or “soft” skills)
- final course is about “grace under pressure”

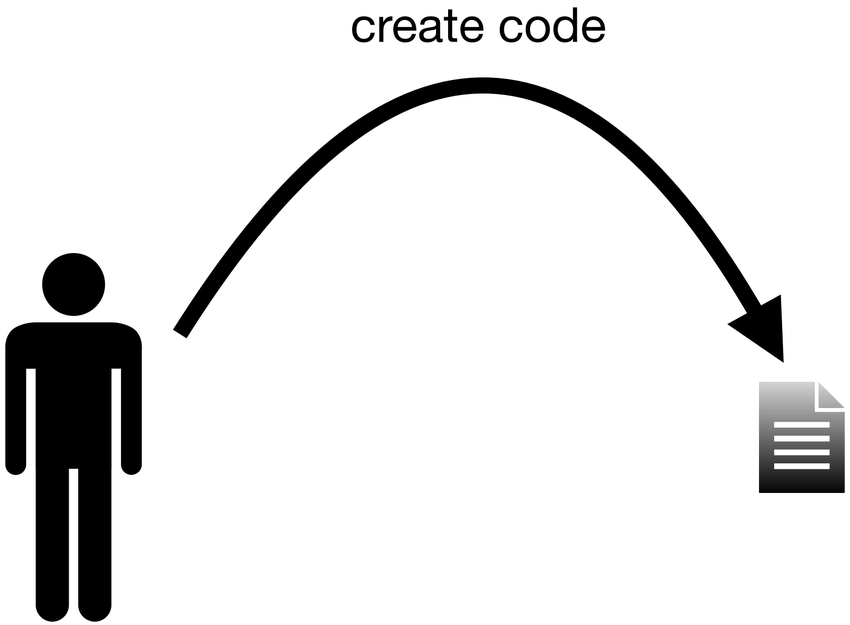


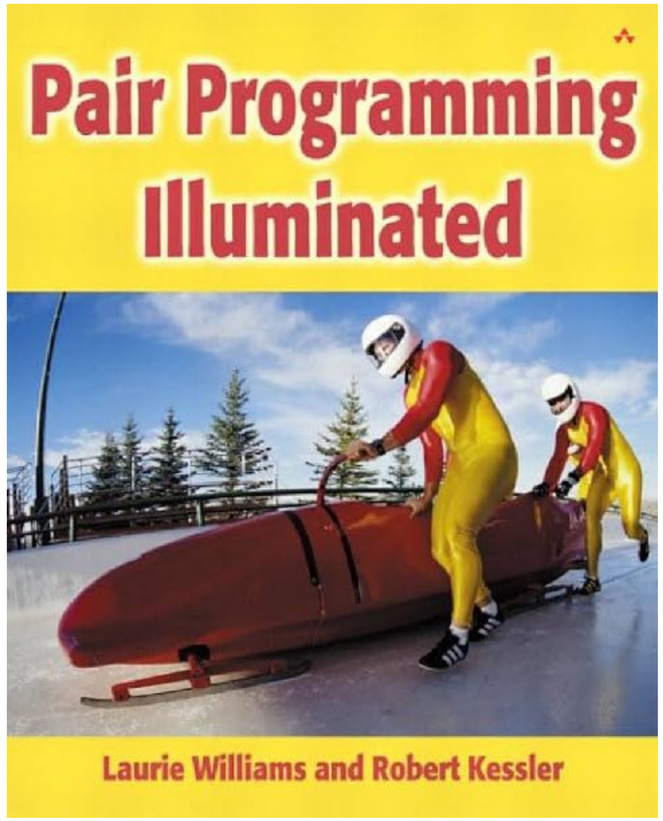
Let’s talk about “Us”

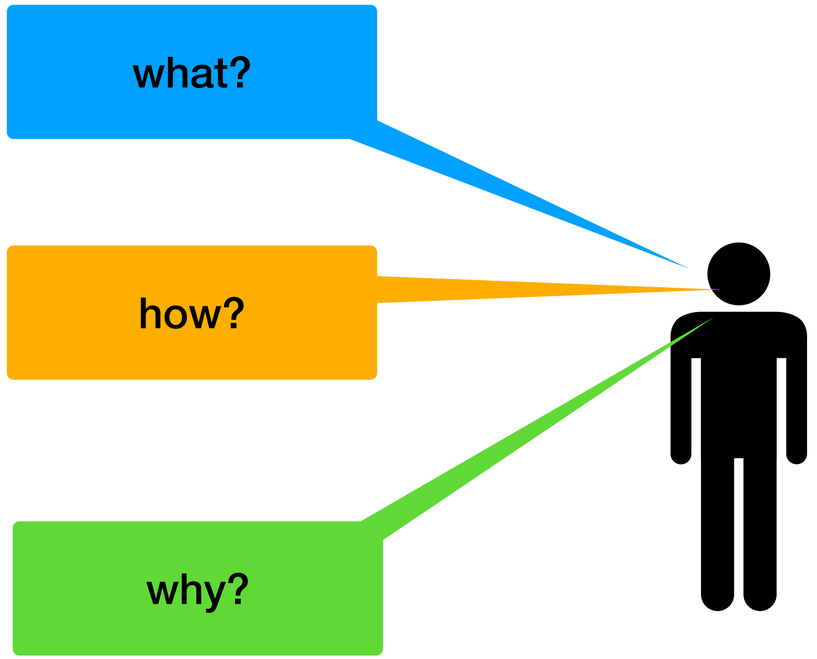
Us


interface IDirection {
// the starting point for sliding this row
public int start();
// are there hasNext tiles to slide in this row?
public boolean hasNext(int i);
// the index of the next tile to slide in this row
public int next(int i);
}
A Human Partner !!
create code
AI !
LATER
what is sufficiently intelligent to check my thinking while I create code?

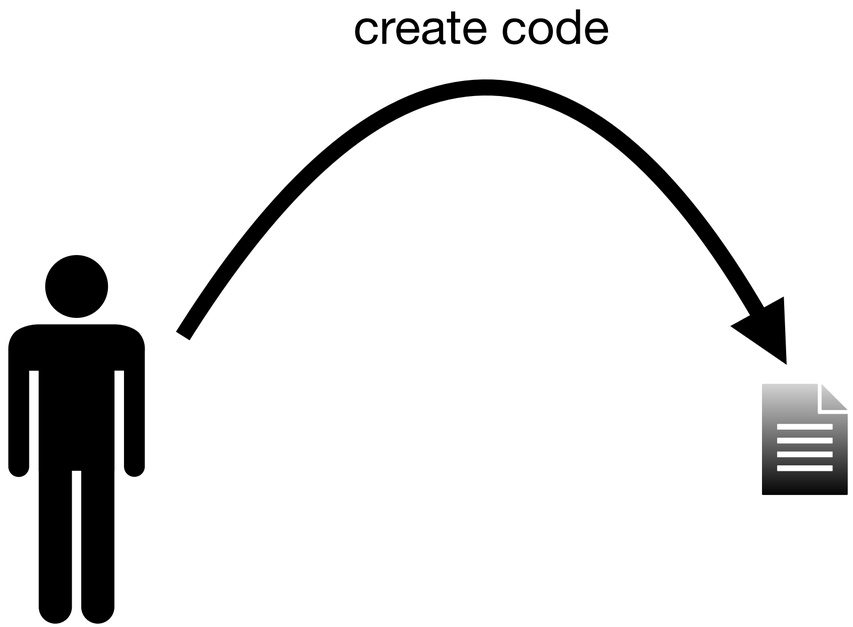
Us


create code

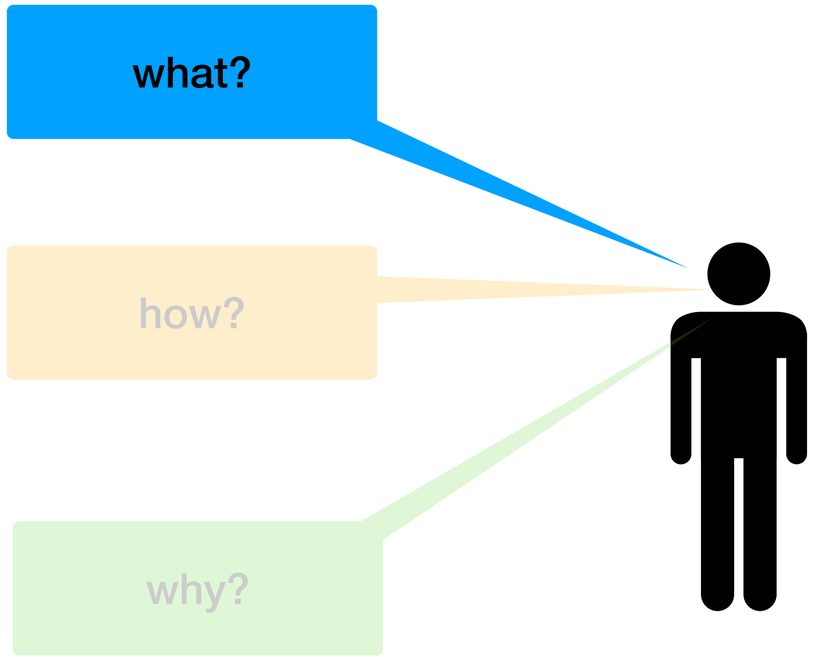

Pair Programming Illuminated
Laurie Williams and Robert Kessler


what?
how?
why?

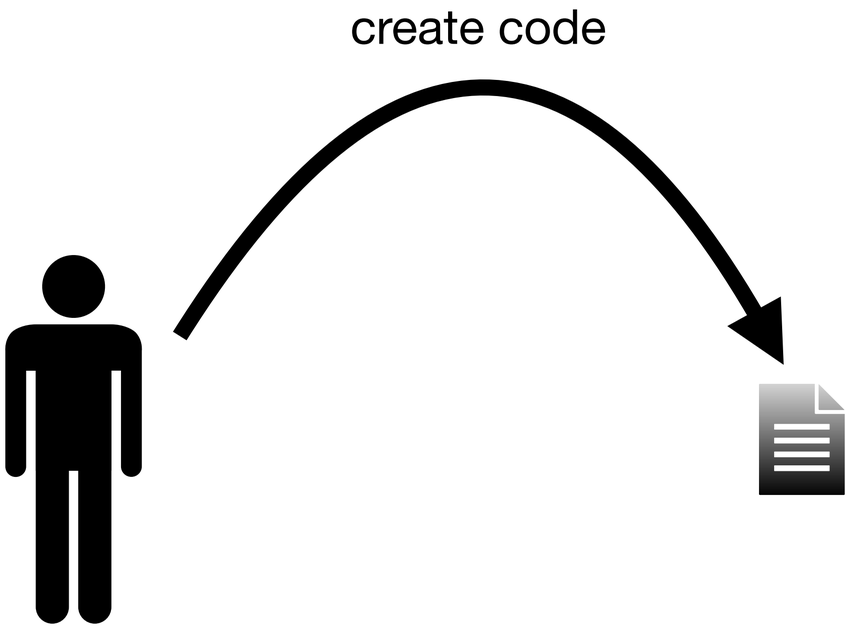
Us, Pair Programming



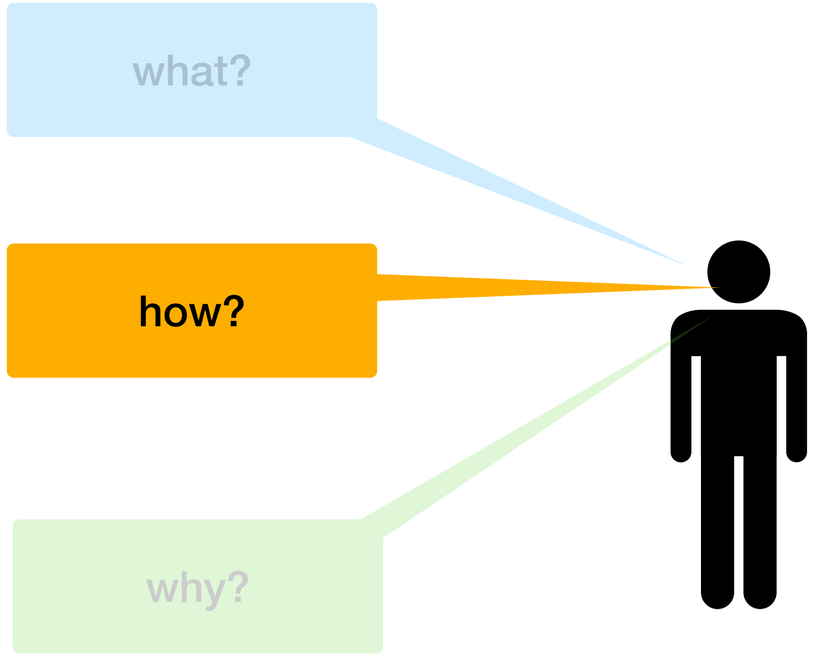
What: All code is created by pairs of software developers.



Us, Pair Programming

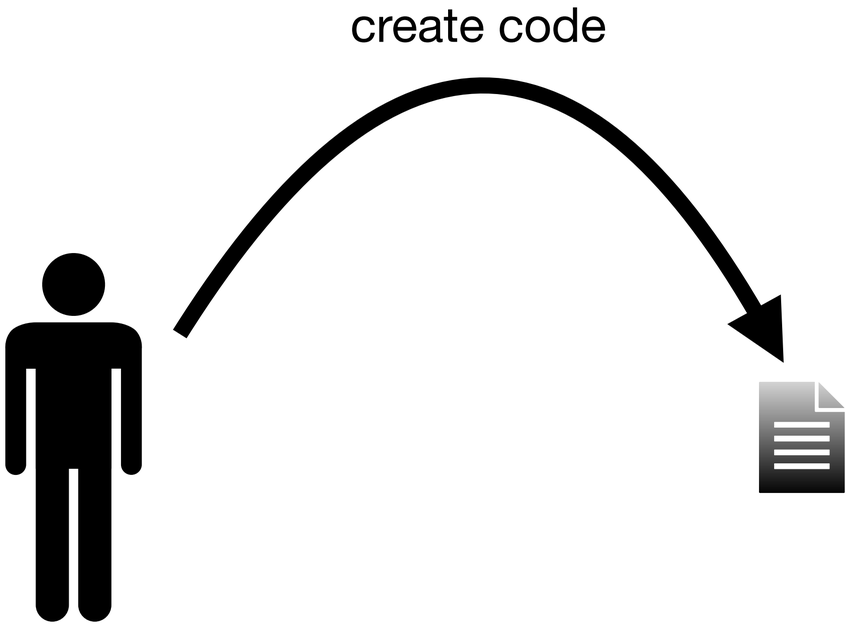


How: One developer drives the development, the other questions the code.

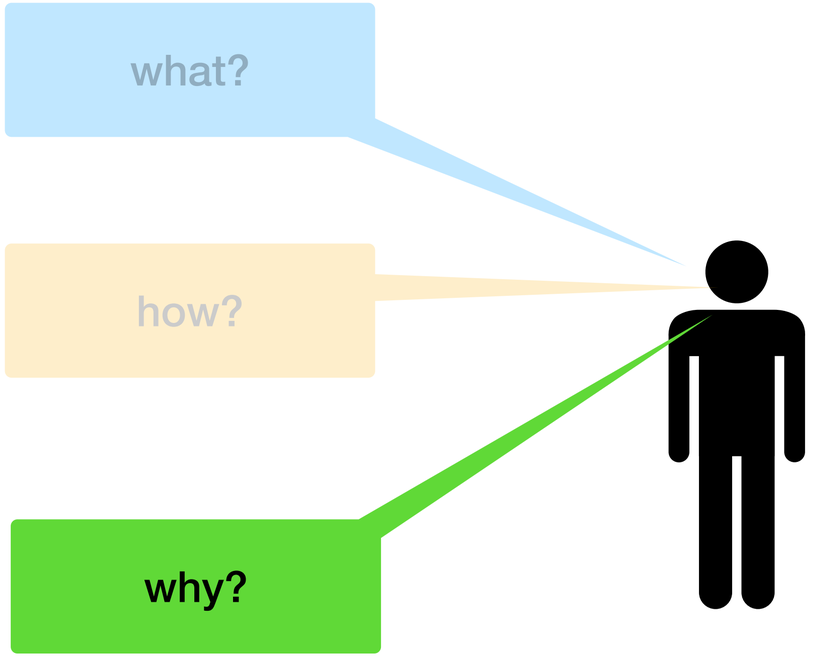


How: Every so often the developers switch roles.

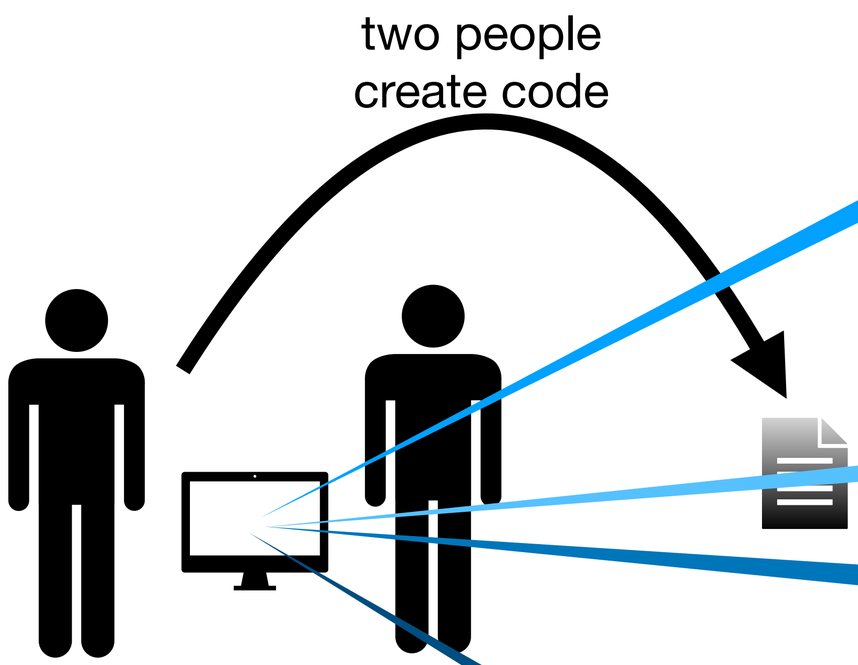
Us, Pair Programming



Why? Oh why are two people being paid to develop one piece of code?



Us, Pair Programming



represents the future receipient of this code:
— is is readable?
— does it work properly?
— are we designing it systematically?

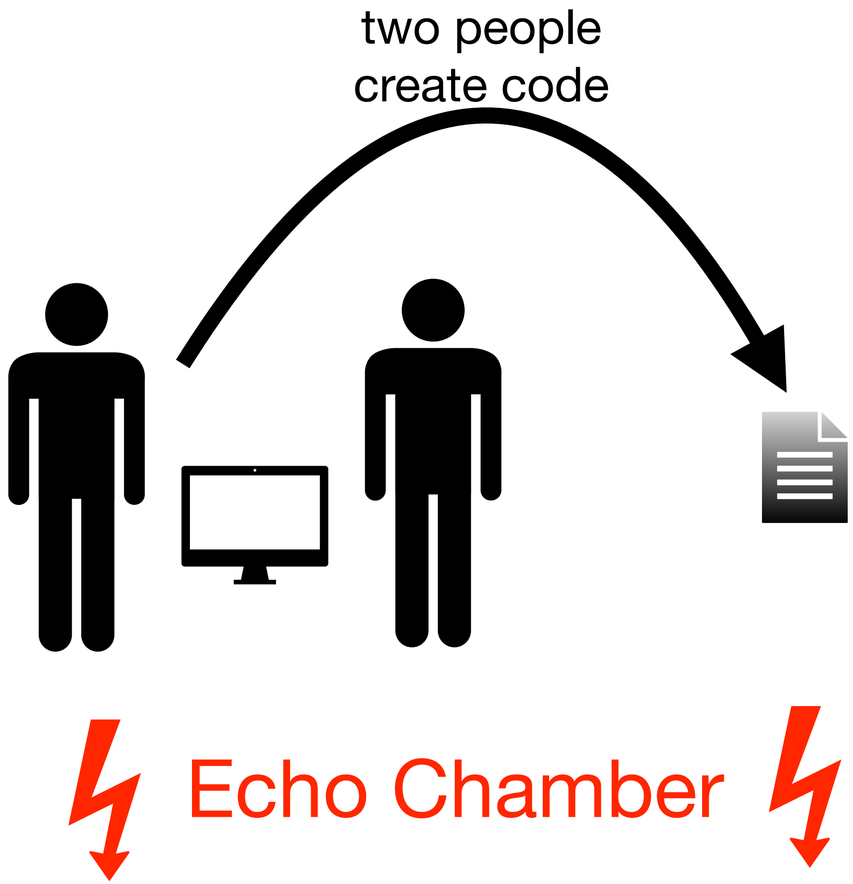
shared knowledge: one is here
— when the other quits, moves to another company
— when one person is out sick for a week
— when someone travels

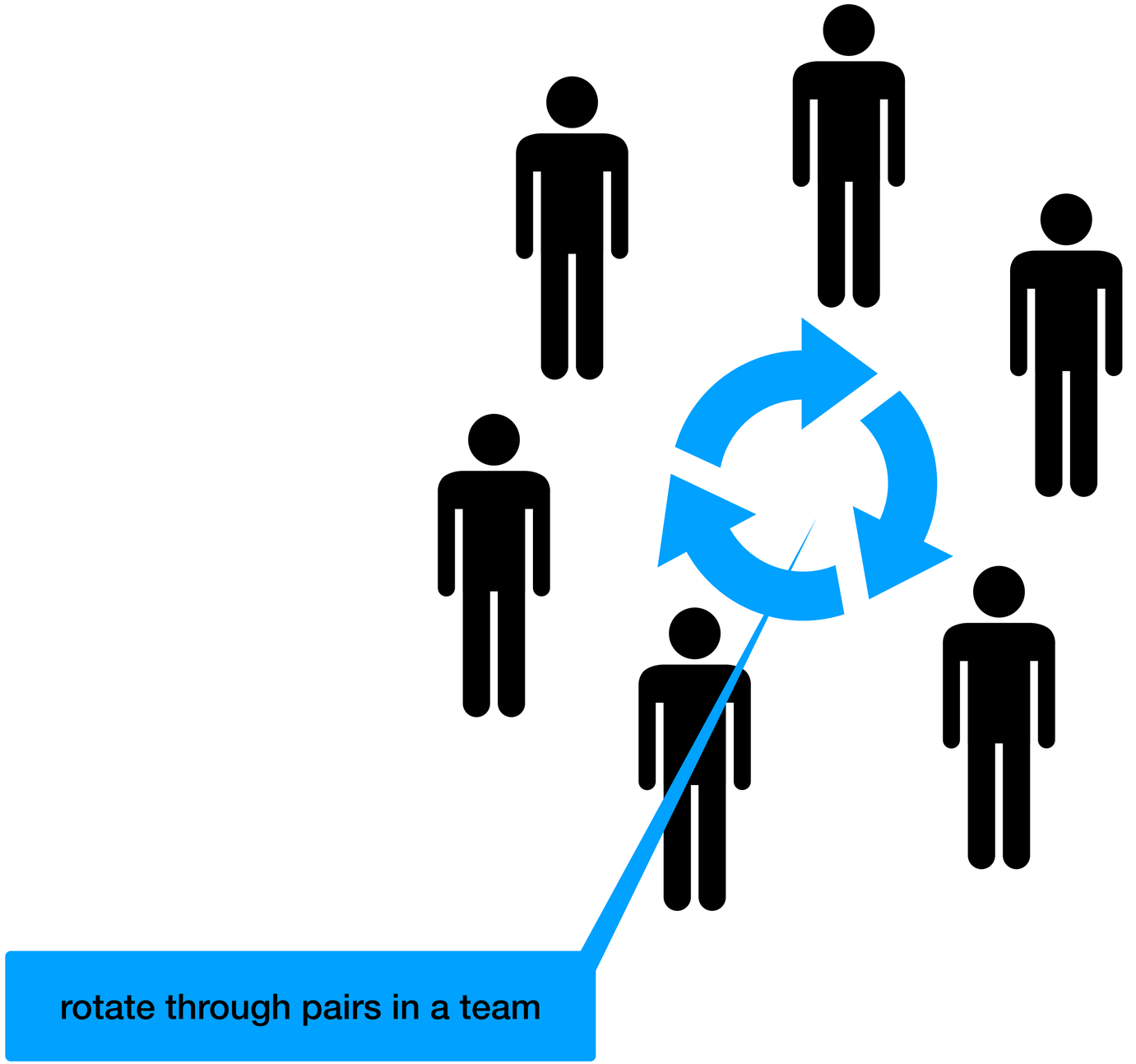
pair programming spreads coding skills and habits
— I like to learn from others
— when I teach, I learn because I have to justify ideas

some programming tasks are dull; par programming can reduce the dull part and avoid bugs due to carelessness

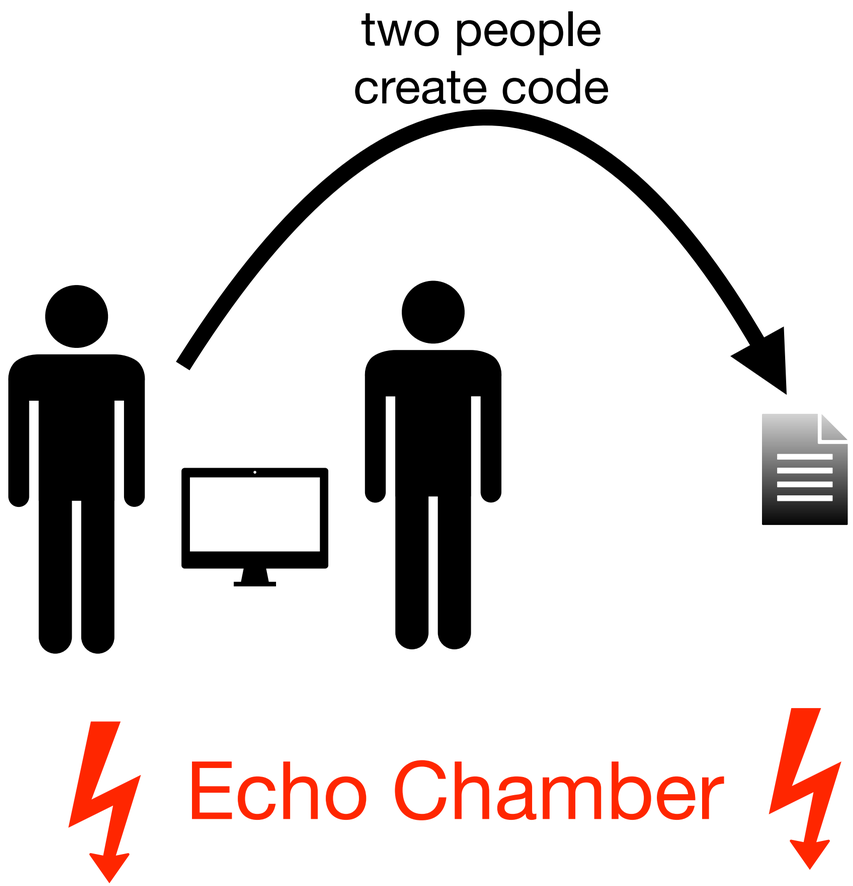
Let’s talk about “Us” some more

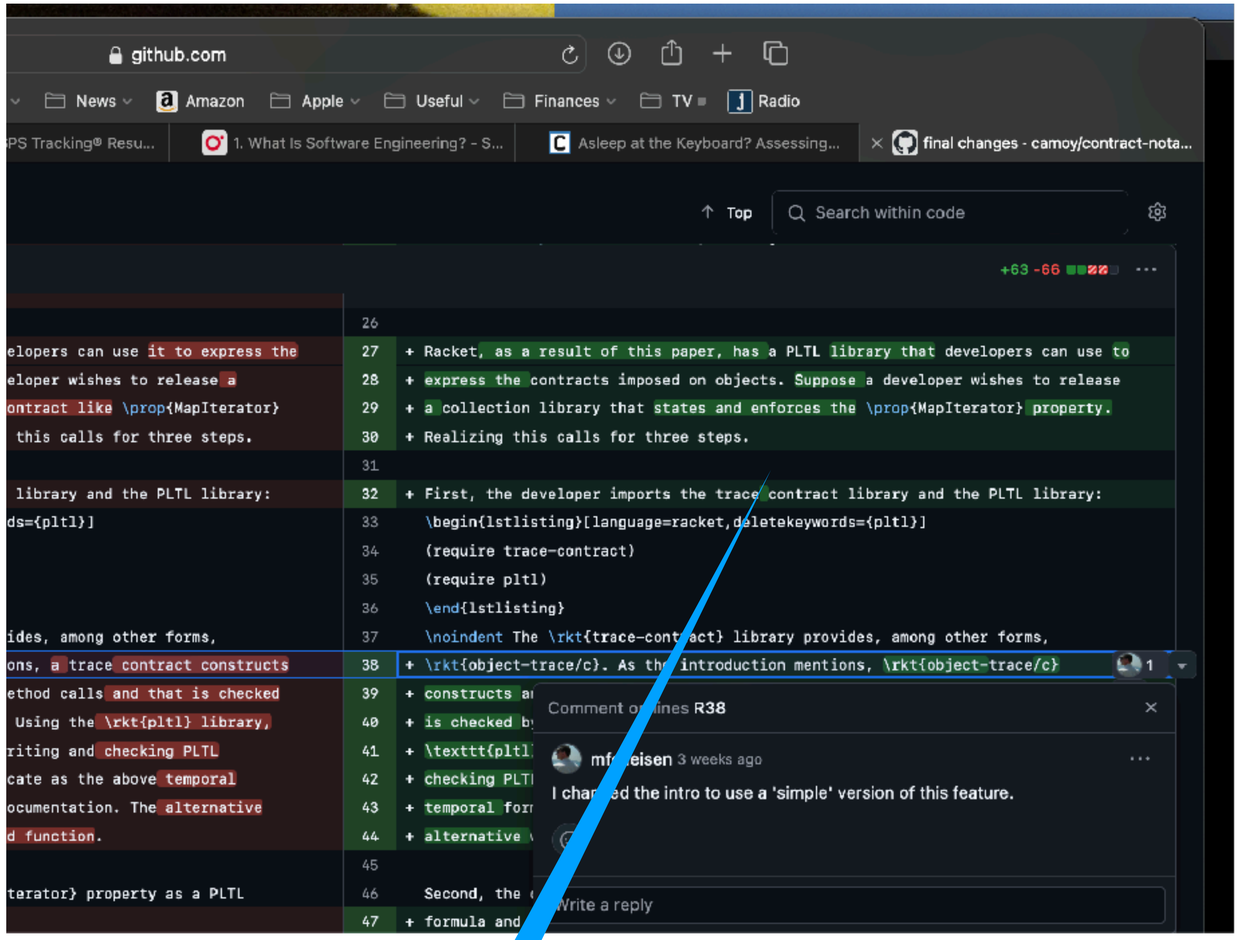
Us, Pair Programming





Us, Pair Programming

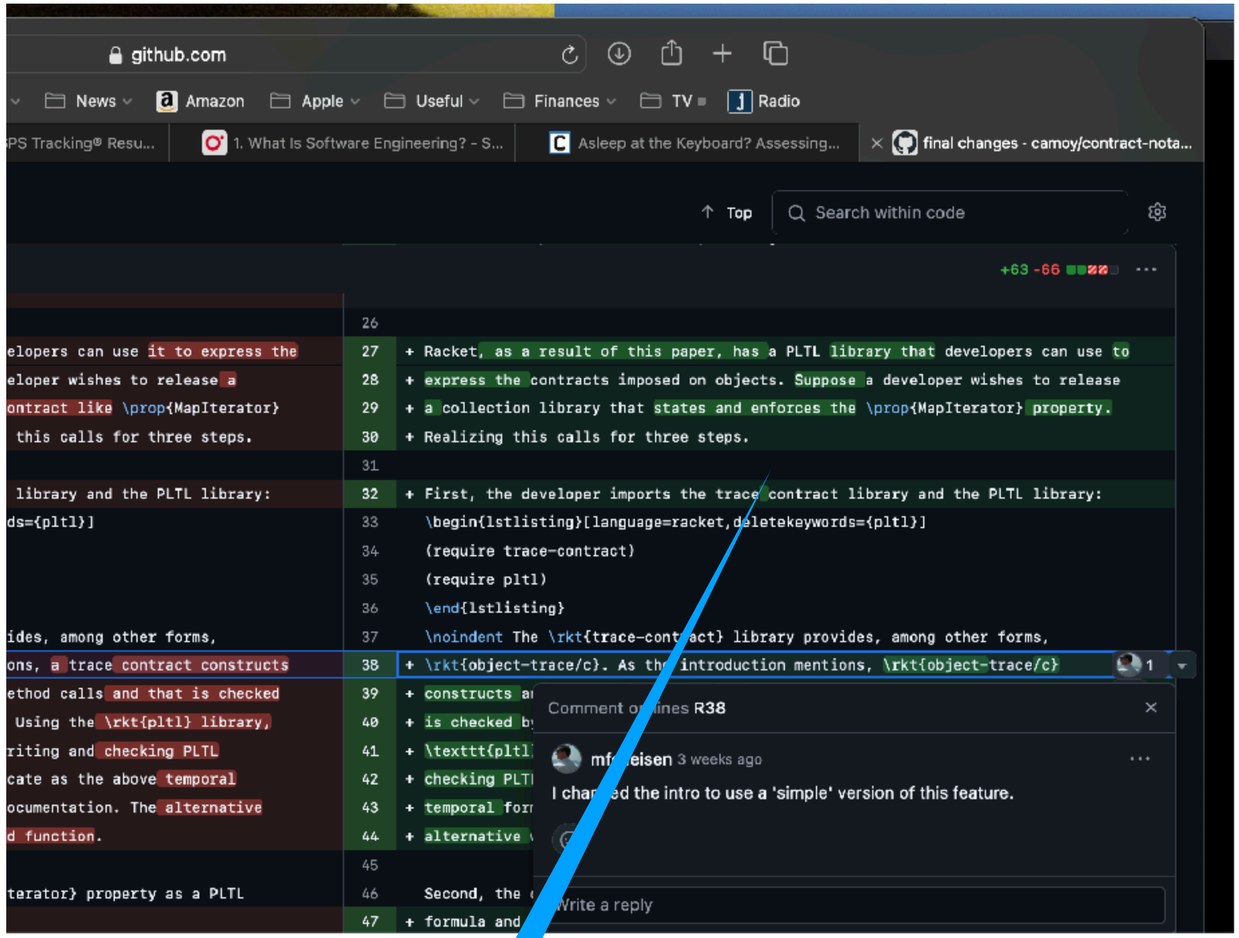




use github to review PRs

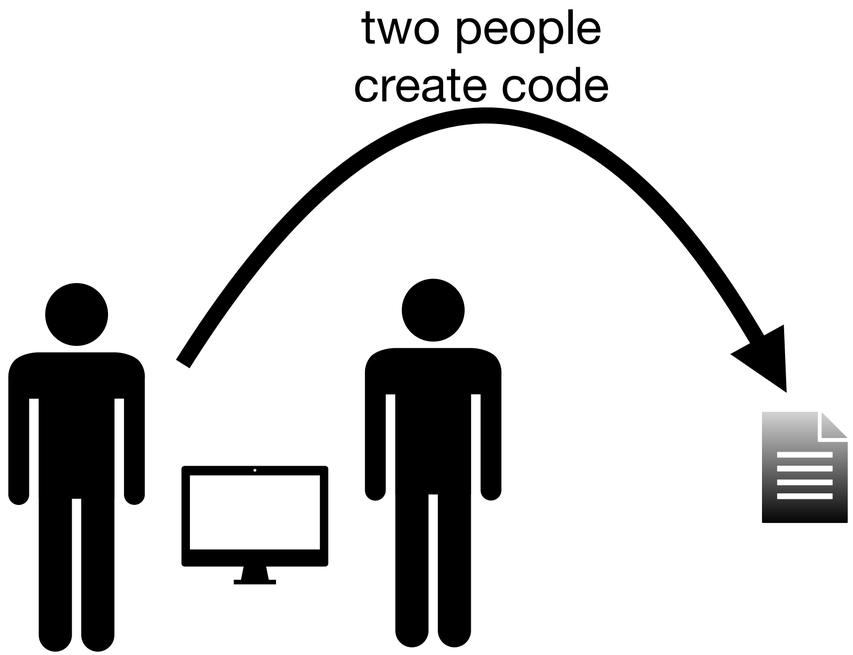
Us, Pair Programming

- useful? yes!
- stress? it is a social network.
- bad patterns? very much so.
- big picture idea? usually not!
- teaching moments? no.



use github to review PRs

Us, Pair Programming



Echo Chamber

NASA:
Formal Code Inspection

OFFICE OF SAFETY AND MISSION ASSURANCE NASA-GB-A302

SOFTWARE FORMAL INSPECTIONS GUIDEBOOK

Approved: August 1993

Substantial net improvements in programming quality and productivity have been obtained through the use of formal inspections of design and code. Improvements are made possible by of a systematic and efficient design and code verification process, with well-defined roles for inspection participants. The manner in which inspection data is categorized and made suitable for process analysis is an important factor in attaining the improvements. It is shown that by using inspection results, a mechanism for initial error reduction followed by ever-improving error rates can be achieved.

Design and code inspections to reduce errors in program development

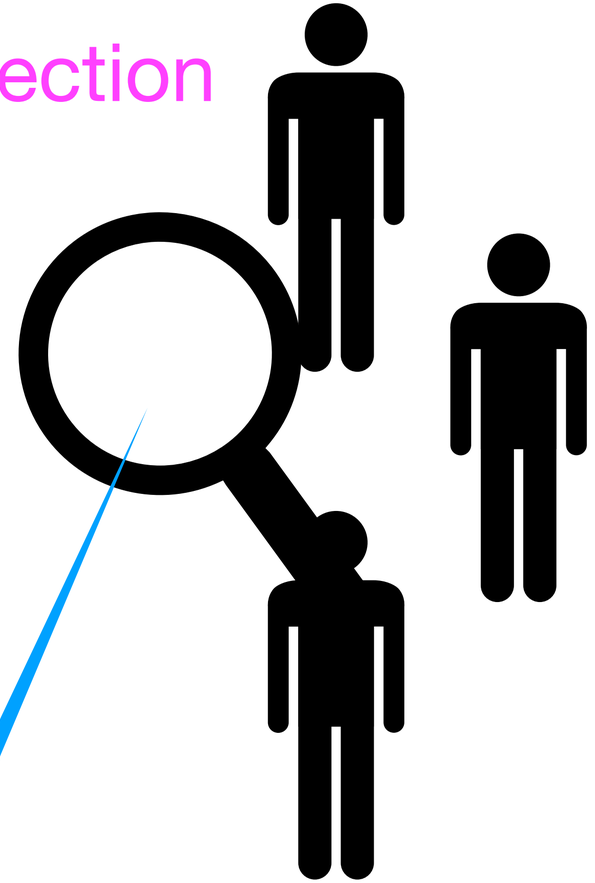
by M. E. Fagan

National Aeronautics and Space Administration
Washington, DC 20546

Fagan:
Code Inspections

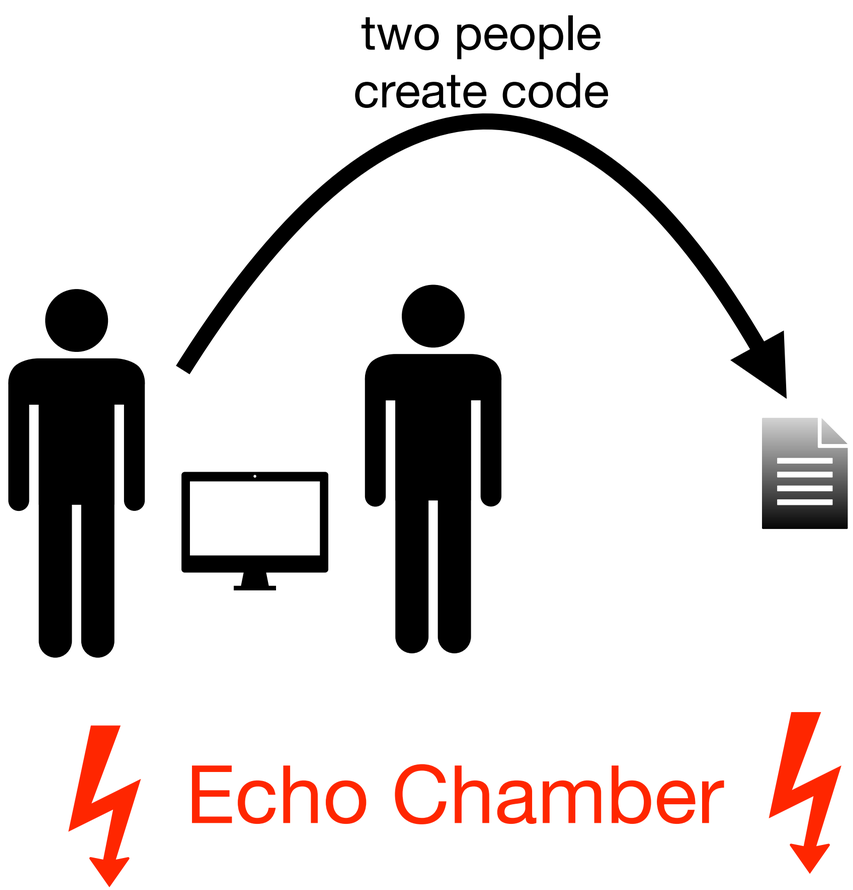
Successful management of any process requires planning, measurement, and control. In programming development, these requirements translate into defining the programming process in terms of a series of operations, each operation having its own exit criteria. Next there must be some means of measuring completeness of the product at any point of its development, by inspection or testing. And finally, the measurement data must be used for controlling the process. This approach is not only conceptually interesting, but has been applied successfully in several programming projects embracing systems and applications programming, both large and small. It has not been found to "get in the way" of programming, but has instead enabled higher predictability than other means, and the use of inspections has improved productivity and product quality. The purpose of this paper is to explain the planning, measurement, and control functions as they are affected by inspections in programming terms.

An ingredient that gives maximum play to the planning, measurement, and control elements is consistent and vigorous *discipline*. Variable rules and conventions are the usual indicators of a lack of discipline. An iron-clad discipline on all rules, which can stifle programming work, is not required but instead there should be a clear understanding of the flexibility (or non flexibility) of each of the rules applied to various aspects of the project. An example of flexibility may be waiving the rule that all main paths will be tested for the case where repeated testing of a given path will logically do no more than add expense. An example of necessary inflexibility would be that *all* code must be inspected. A clear statement of the project rules and changes to these rules along with faithful adherence to the rules go a long way toward practicing the required project discipline.



in person code inspections

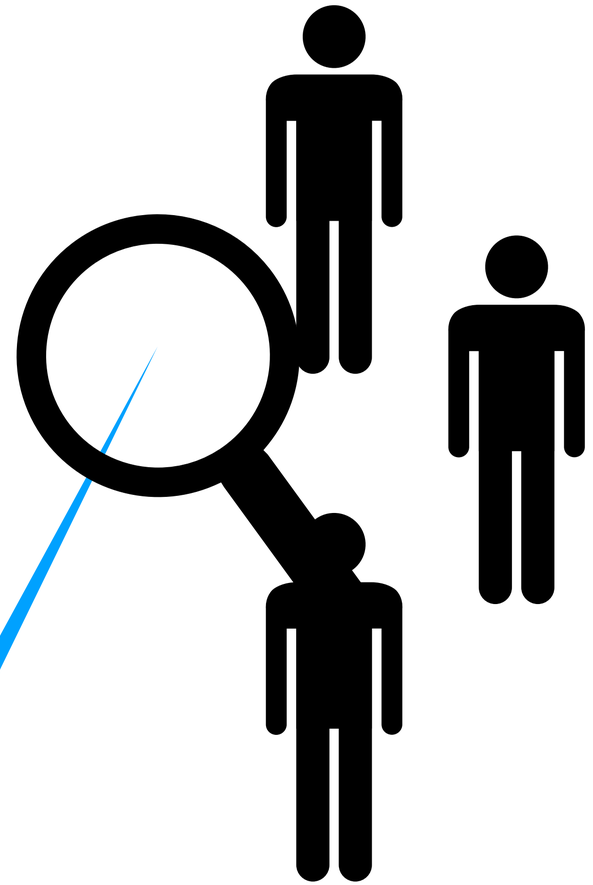
Us, Pair Programming



Echo Chamber

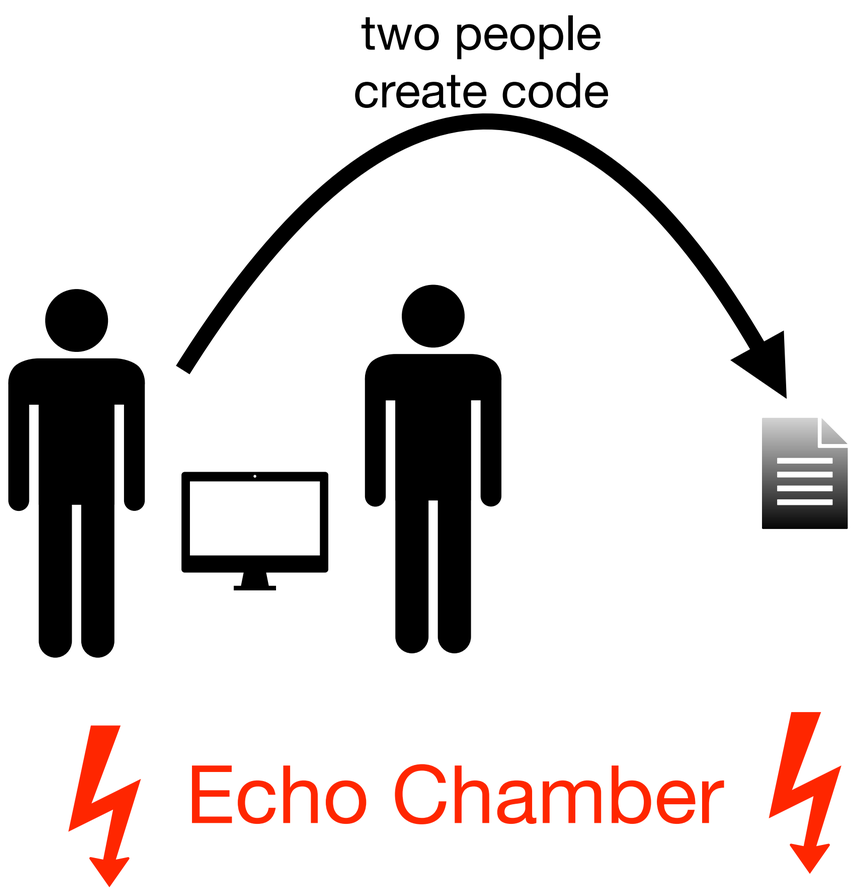
How: small panels

- head (moderator)
- reader (1 or 2)
- secretary?



in person code inspections

Us, Pair Programming



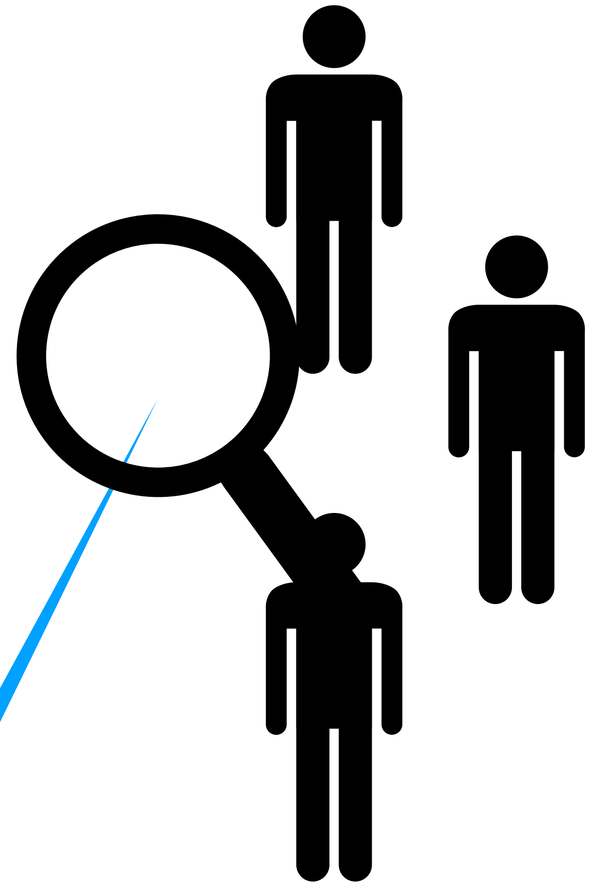
Echo Chamber

What:

creators present

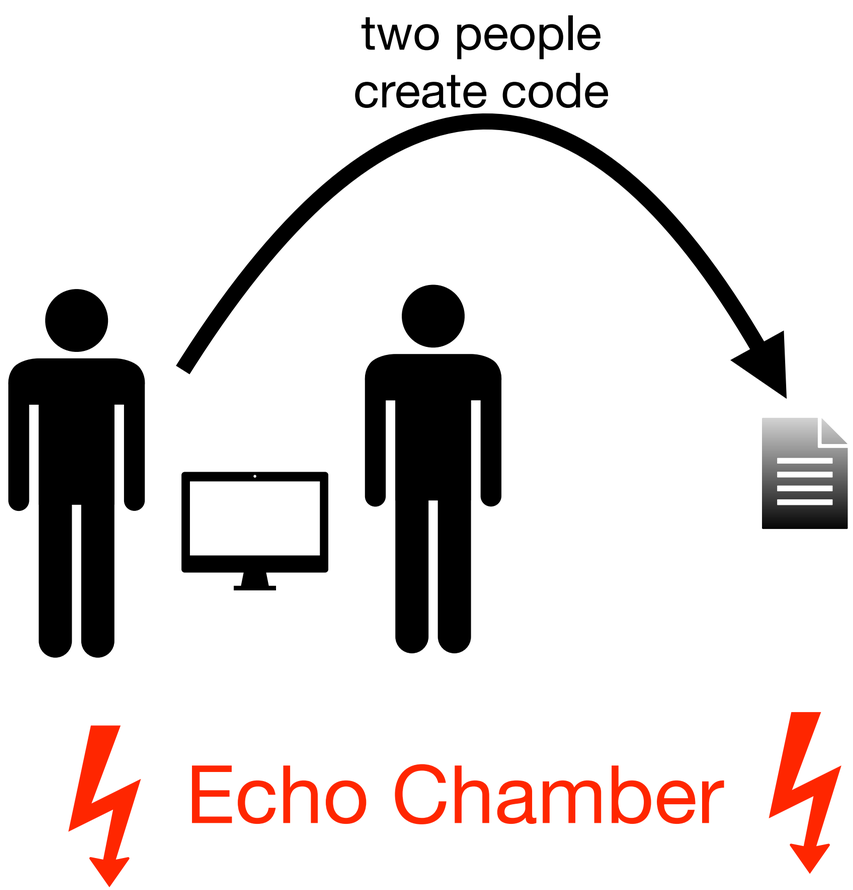
readers inspect

— readability
— bugs
— design flaws



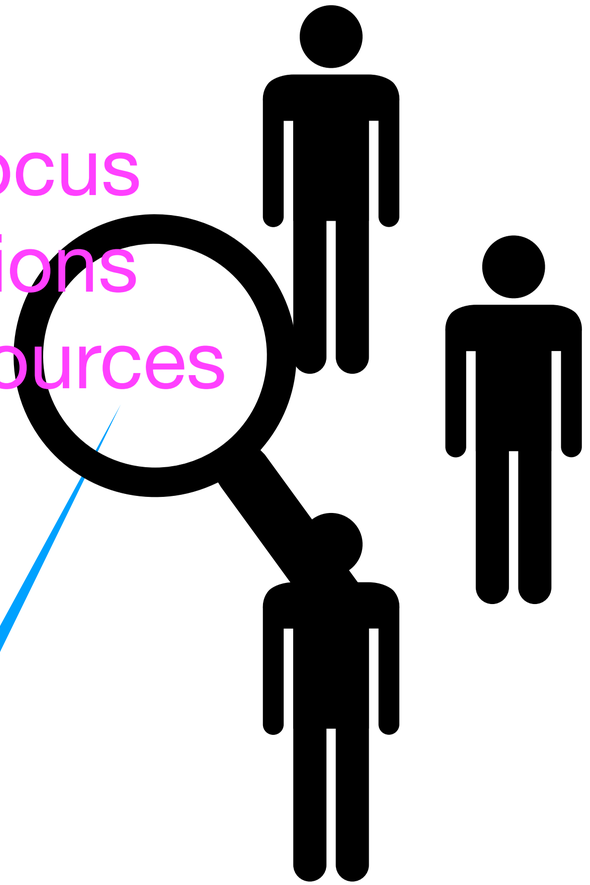
in person code inspections

Us, Pair Programming



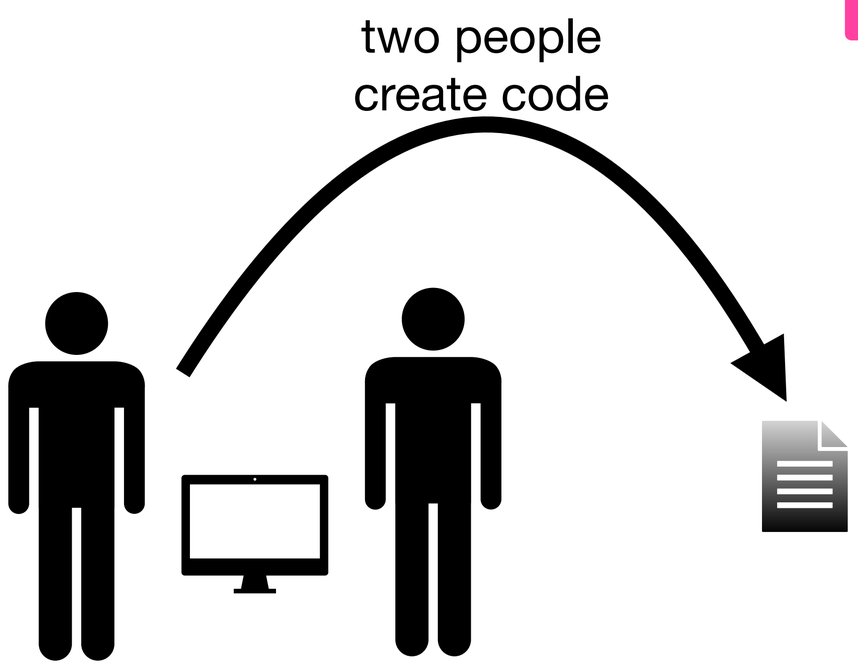
Why:

- presenting forces focus
- real-time conversations
- develop human resources



in person code inspections

Us, Pair Programming

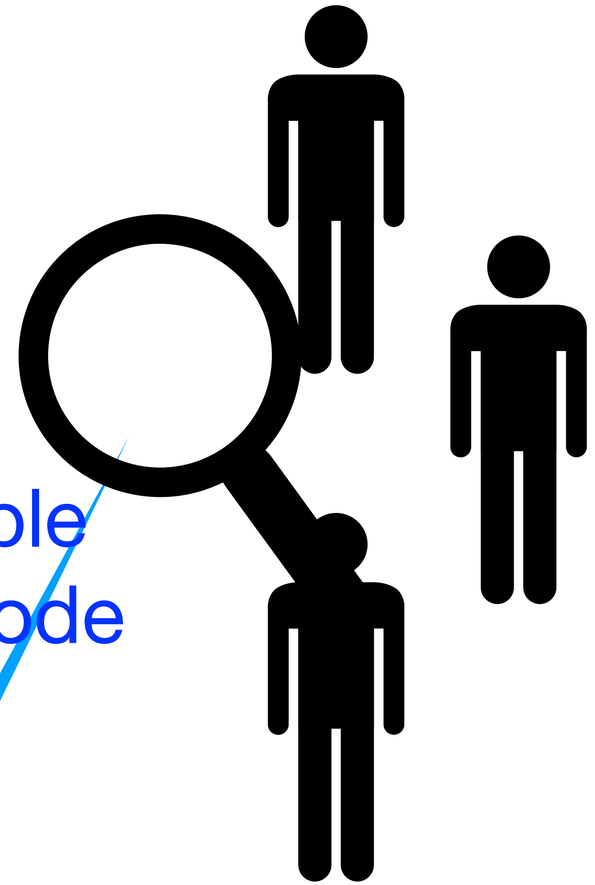


Echo Chamber

is this really a cost?

Cost:

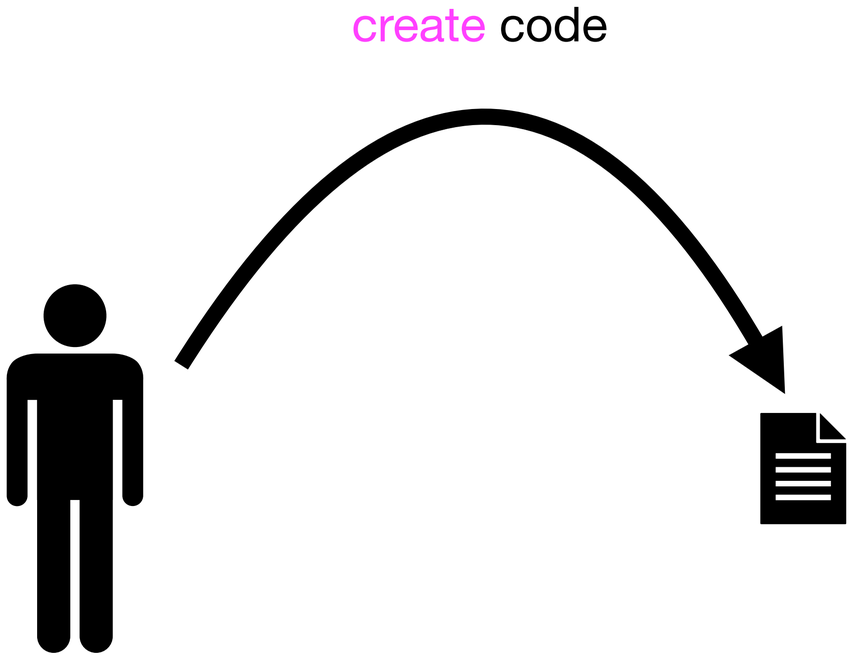
- training
- time for reviews
- stress of facing people with not-so-great code



code inspections

Let’s talk about “I” or really “Ego”

I, Ego



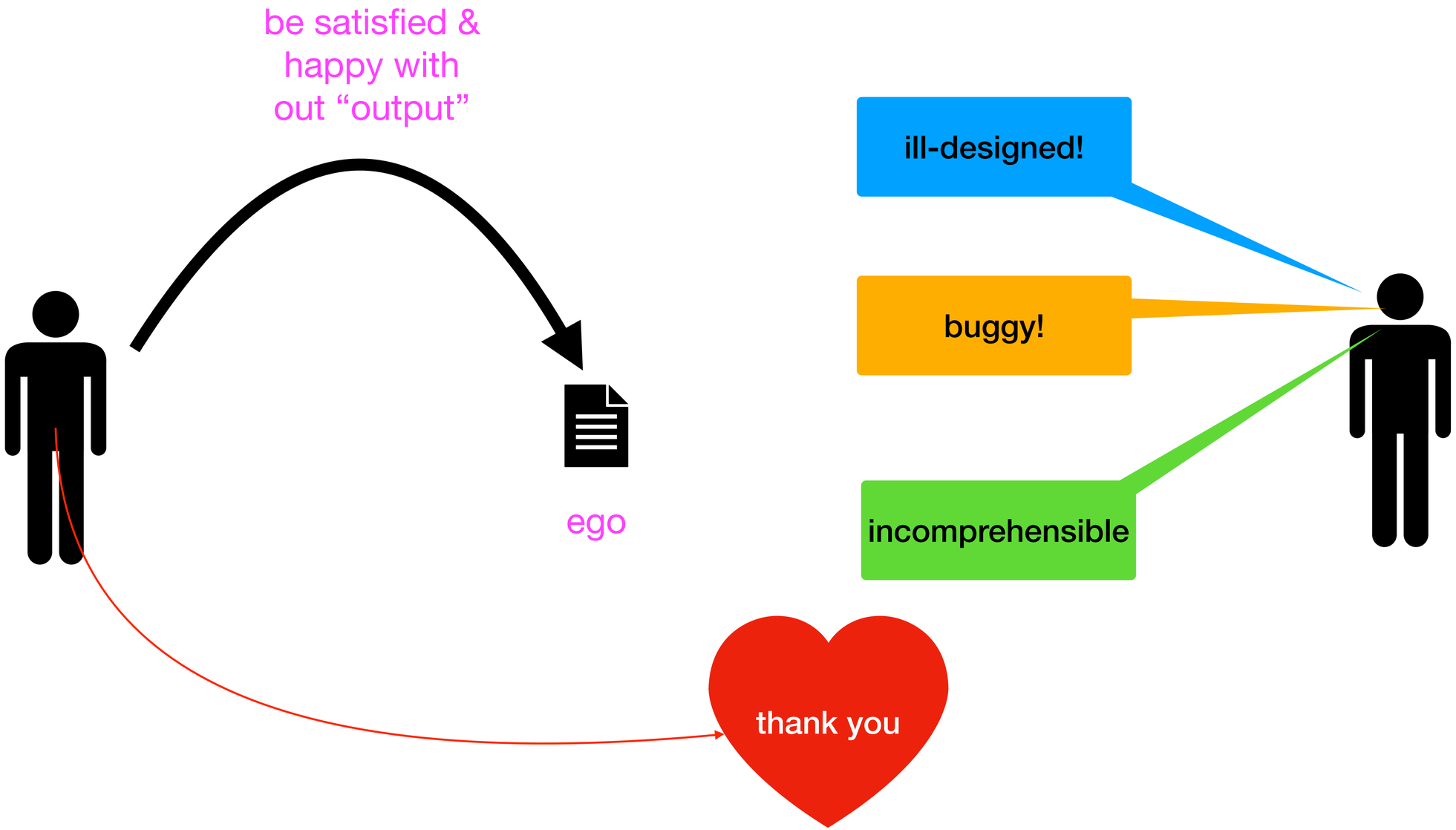
Creators

- authors
- composers
- painters
- …
- developers

who fail

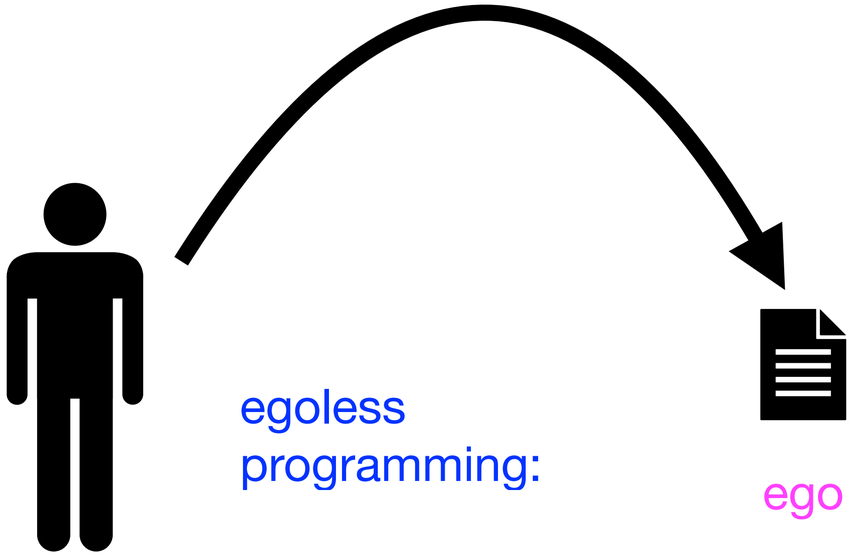
- bad reviews, no sales
- booed at premier
- “ouch” at vernissage
- gets paid

I, Ego


be satisfied &
happy with
out “output”
ego
ill-designed!
buggy!
incomprehensible
thank you

I, Ego

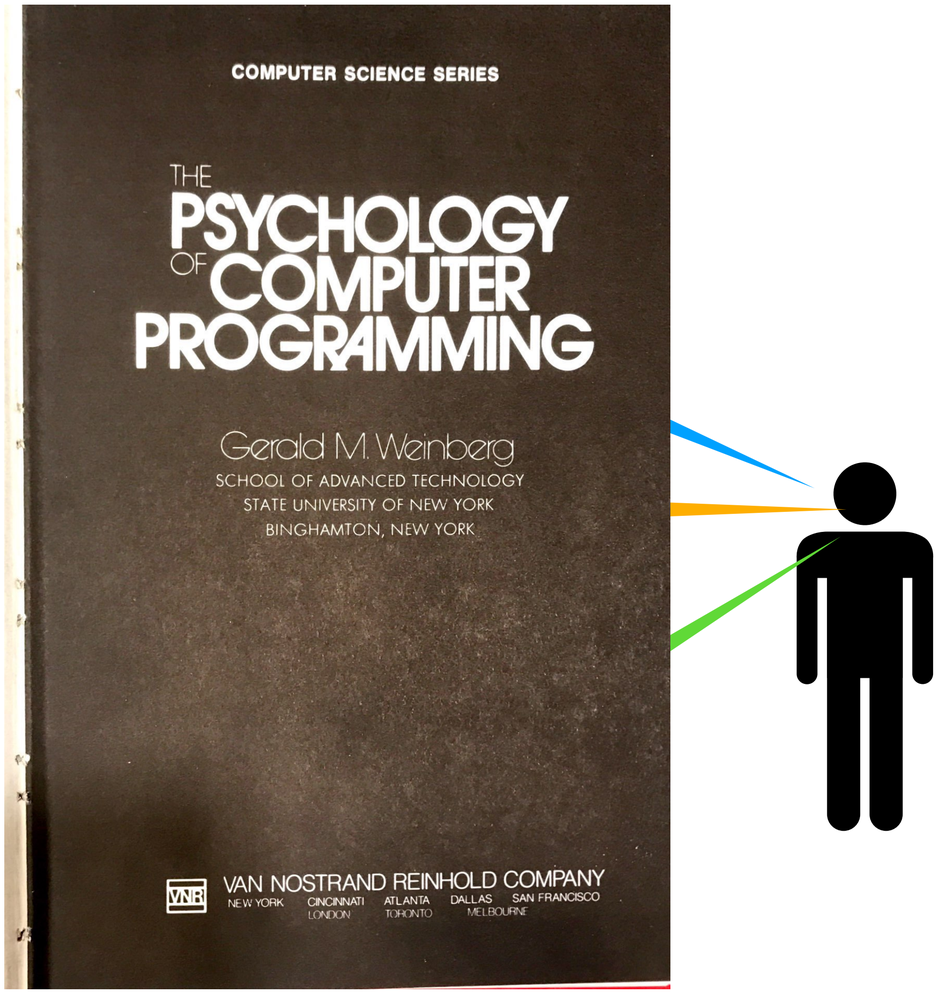
be satisfied &
happy with
out “output”



egoless
programming:

ego

critique doesn’t hurt our ego,
it improves our creation



I, Ego

egoless programming taken seriously:

- create code that you are happy about, put some “ego” into it
- solicit feedback, often
- take negative feedback for help to improve this product of yours
- improve code, rinse and repeat

Social Skills

to develop software in a socially responsible manner, we need

- a mental state of ``egoless'' programming
  - welcome critique
- continuous feedback to our thinking while we code
  - read code aloud to a partner
- active milestone reviews
  - github at a minimum
  - presentations to the team
  - formal panel reviews at a maximum

Let’s talk about Technical Skills

The Big Picture: How to turn novices into basic sw devs

- five core courses (plus one 6-month co-op)
- key ideas across all courses, scaled from 5-liners to 15Kloc per semester:
  - fundamentals are more important than currently fashionable industry ideas
  - design code systematically (techn. or “hard” skills)
  - programming is a people discipline (social or “soft” skills)
- final course is about “grace under pressure”

Fundamentals IV
very large, distr.
imspections

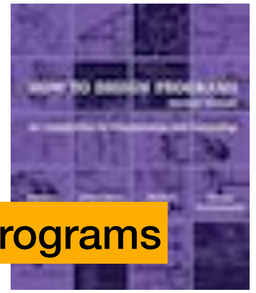
6-mo co-op

Fundamentals III
code that does not
fit into your head

Fundamentals II
sys. design w/
typed OO;
pair prog.

Logic
stating properties
run-time checks
static checks

Fundamentals I
sys. design
pair prog.

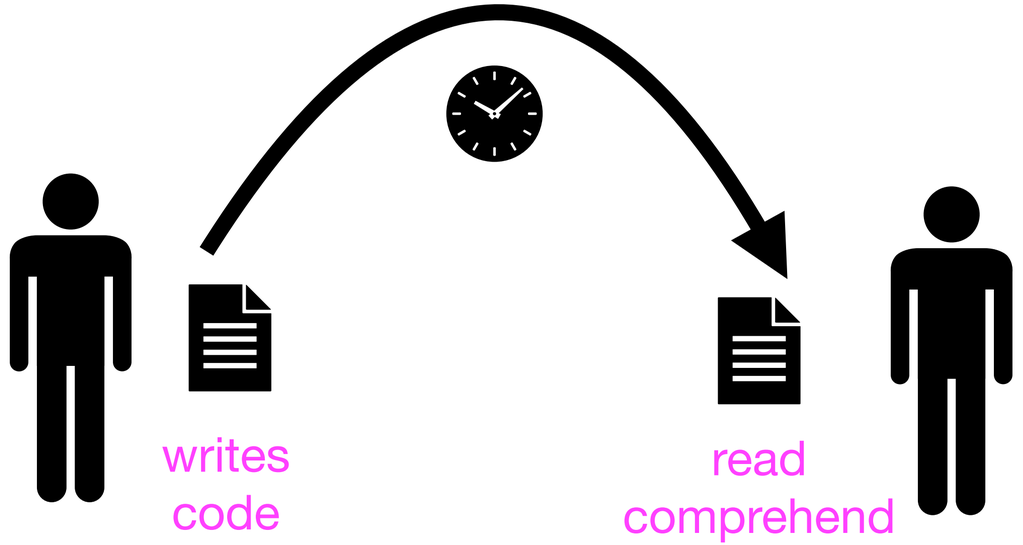


How to Design Programs

Technical Skills: The Purpose

Every unit of code needs a
focused purpose statement:

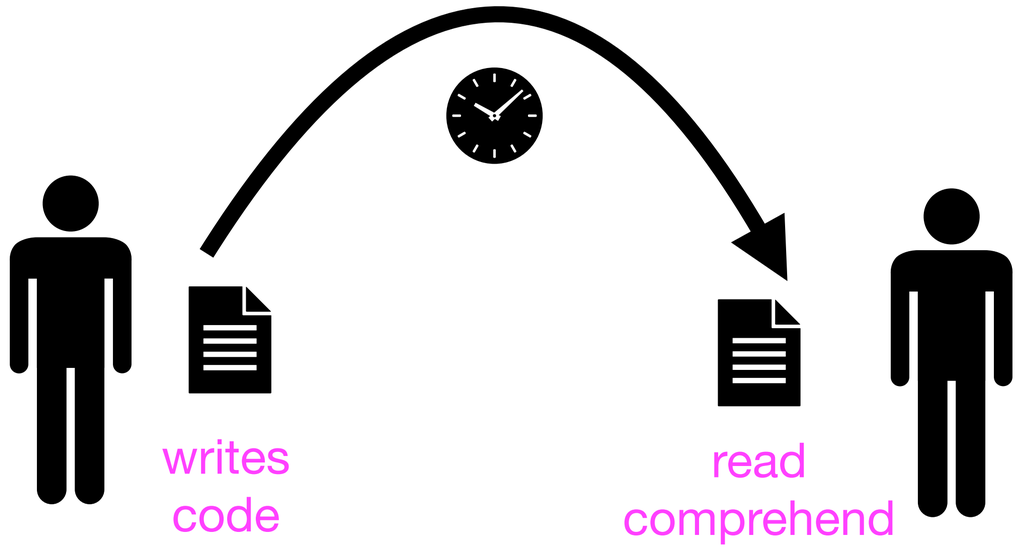
- method
- class
- module
- package



Technical Skills: The Purpose

Every method needs a
focused purpose statement:

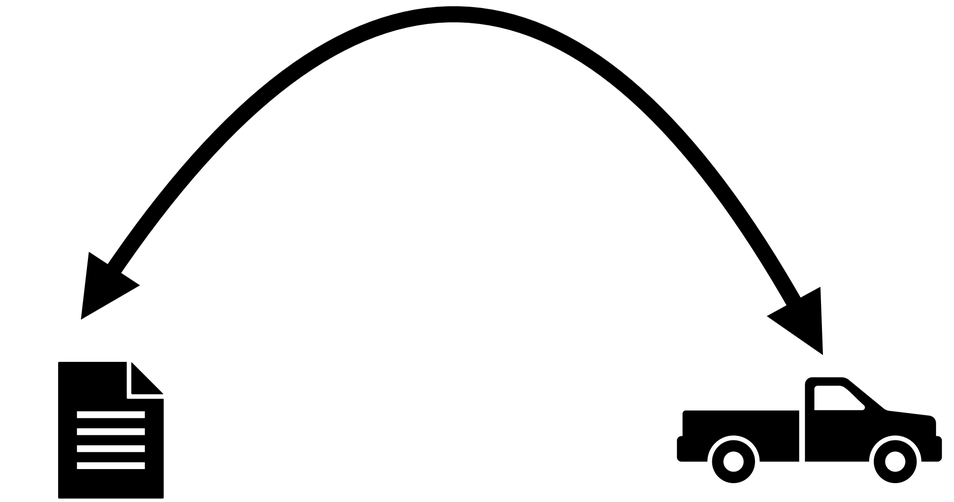
- *what* it computes relative to the class
  - `this`
- clarifies whether
  - it is atomic.
  - it is composite.



Technical Skills: The Purpose

Every class (collection) needs a focused purpose statement:

- *data* represents *information*
  - abstraction!
- clarifies how to turn
  - an actual car into an instance of `Car`
  - interpret an instance of `Car` in the real world



Technical Skills: The Purpose

```
class Car {

   int shortest_Distance;

   int move_Car() {

      ... }

}
```

To what?

To where?

Technical Skills: The Purpose

We know we need GREAT NAMES for methods and fields and so on.

Technical Skills: The Purpose

```
class Car {
  int shortest_Distance_To_Car_On_The_Left_From_Front_Left_In_CM;
  int move_x_CM_to_the_Right_Relative_to_Front_Right_of_Car() {
    ... }
}
```

Technical Skills: The Purpose

```
class Car {

   int shortest_Distance_To_Car_On_The_Left_From_Front_Left_In_CM;

   int move_x_CM_To_The_Right_Relative_To_Front_Right_of_Car() {

      ... }

}
```

F# to the Rescue

Technical Skills: The Purpose

Every reader must parse names such as these.

```
class Car {

   int shortest_Distance_To_Car_On_The_Left_From_Front_Left_In_CM;

   int move_x_CM_To_The_Right_Relative_To_Front_Right_of_Car() {

      ... }

}
```

Other than `cm` little about this name can be enforced.

Where do we stop with this name game?

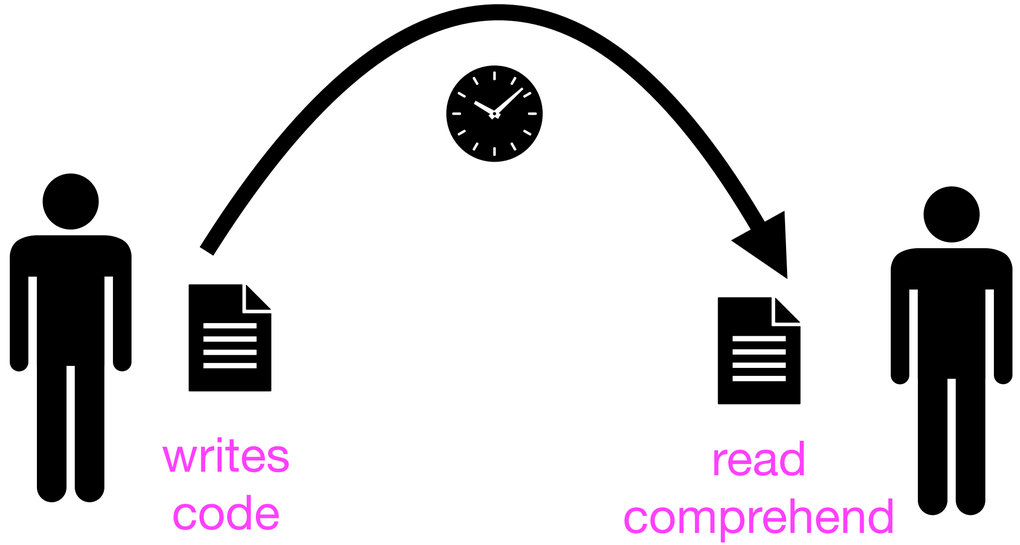
Technical Skills: The Purpose

```
class Car {

   // shortest_distance to car on the left from front left
   int distance_To_Left; // in cm

   // determine how many CM THIS car's front must move to the right
   int moveTo_The_Right() {

      ... }

}
```

Technical Skills: The Method

Every method must convey the "how" in a concise manner. It is either *atomic* or *composite.*

An *atomic* method comes with a purpose statement that explains *what* it computes (and occasionally *how* it computes).



A *composite* method comes with a purpose statement that *enumerates* the tasks it composes.

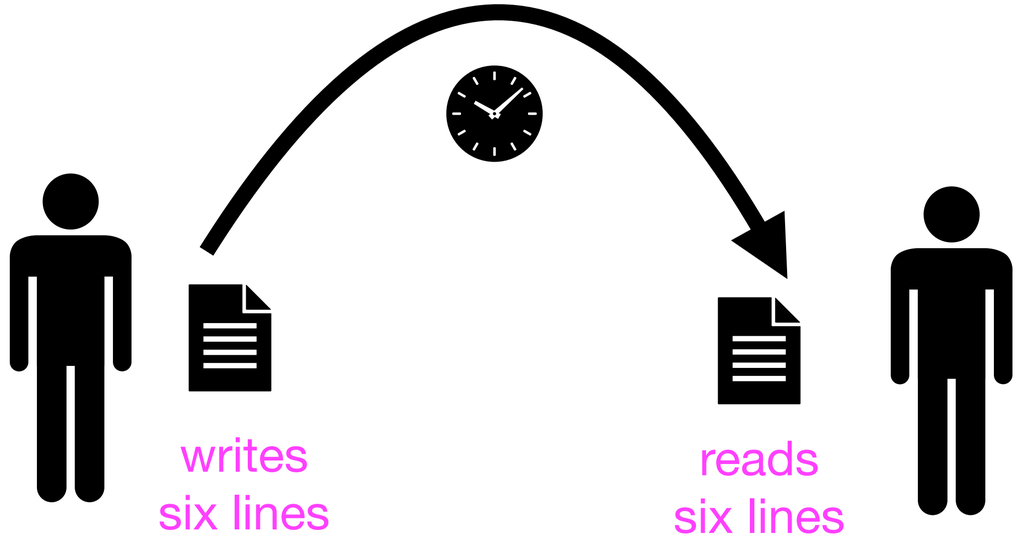
(And if the names of the "subroutines" are well-chosen, we can erase the purpose statement.)

Technical Skills: The Method

Squeak [2020] consists of 600,000 lines of code.

How long is the average method?

Six lines.



Technical Skills: The Method

Ending a Game

The game ends if

- all players have been kicked at the end of a player's turn;
- a player has 20 or more points at the end of its turn;
- no more cards are available for purchase; or
- the bank is empty and no player can buy a card.

```
class GameState {
  // is the game over according to the rules?
  public boolean gameOver() {
   return
      this.allPlayersEliminated()
      || this.aPlayerHasGoodScore()
      || this.allCardsBought()
      || this.noPebblesOrBuyers();
  }
}
```

Technical Skills: The Method

passes the exact same tests

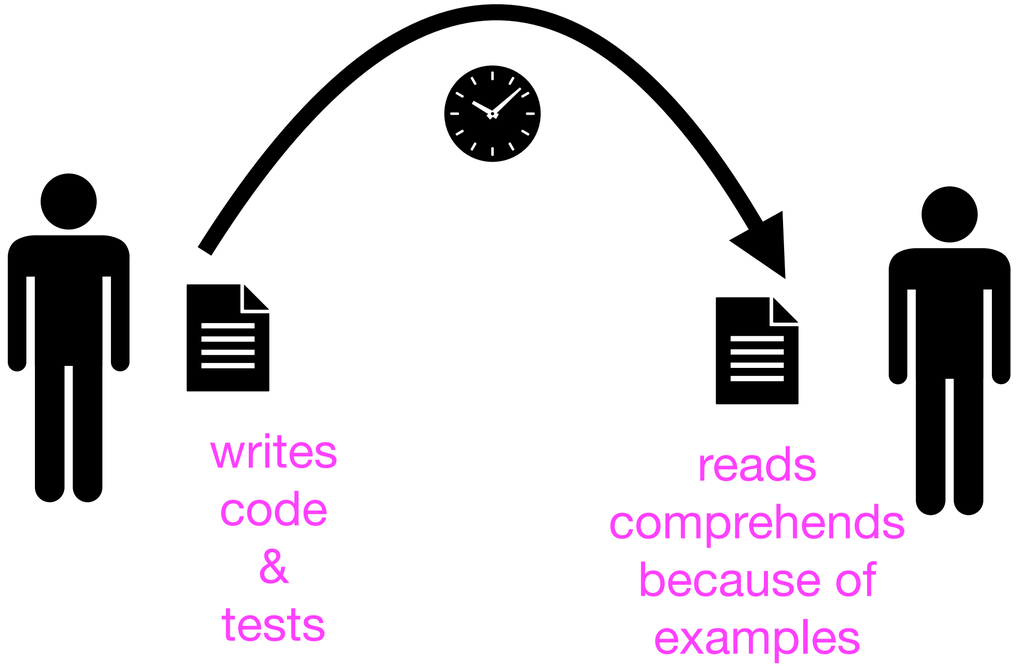
```
class GameState {
  // is the game over according to the rules
  public boolean gameOver() {
    return
      this.allPlayersEliminated()
      || this.aPlayerHasGoodScore()
      || this.allCardsBought()
      || this.noPebblesOrBuyers();
  }
}
```

```
class GameState {
  // is the game over according to
  public boolean gameOver() {
    if (this.players.isEmpty())
      return true;
    if (this.cards.isEmpty())
      return true;
    for(Player p : this.players) {
      if (p.score >= PlayerWins)
        return true;
    for(Player p : this.players) {
      if (! p.canBuy(this.cards))
        return false;
    return true;
  }
}
```

Technical Skills: The Tests

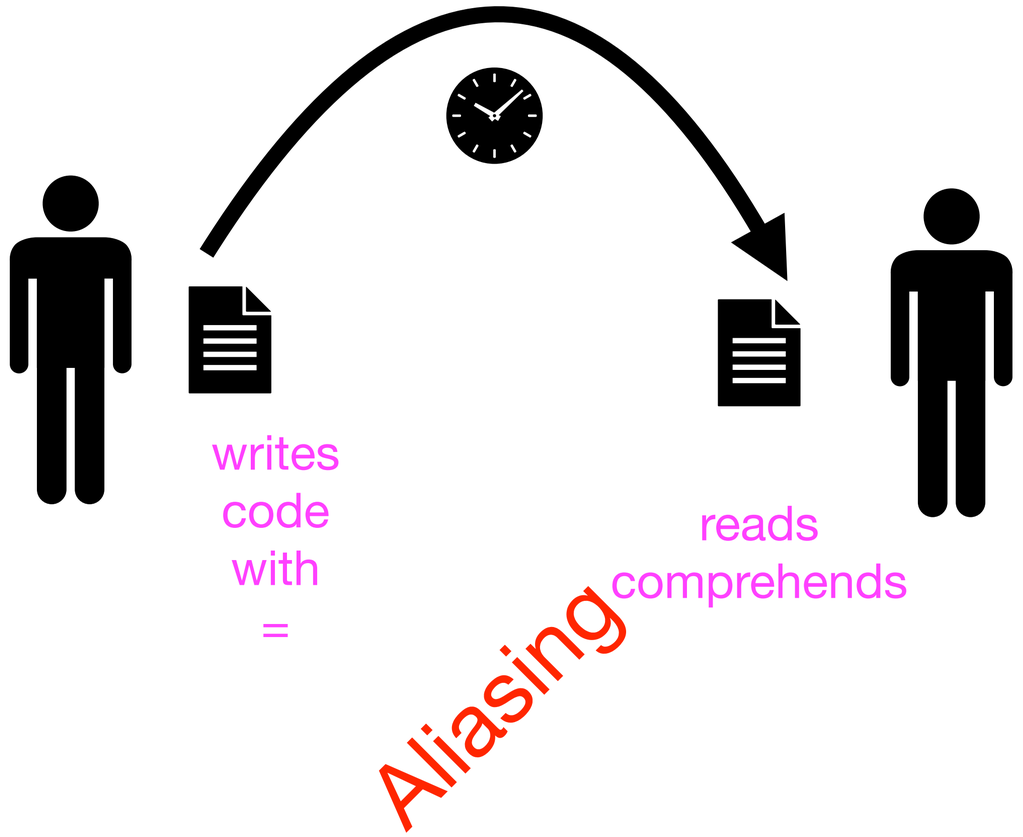
Every method needs tests.

- unit tests encode examples
  - … that help comprehension
- unit tests discover simple bugs
  - … and sometimes complex ones
- have you considered property tests?
- how good are your test suites?
  - mutation testing helps answer this question



Technical Skills: Mutation

Object-oriented programming is not C with class and extends sprinkled over the code.



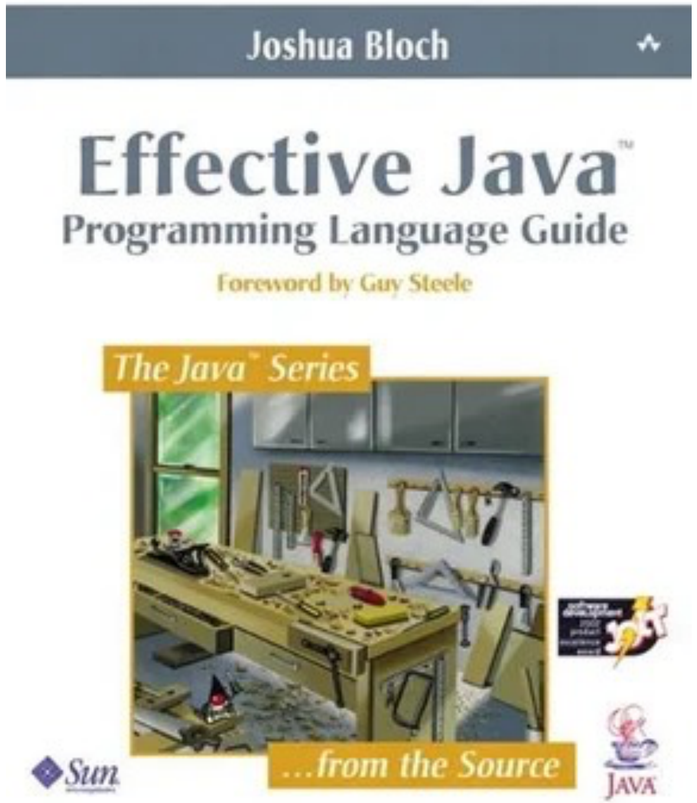
Technical Skills: Mutation

"OOP came from many motivations, two were central. … the small scale one was to find a more flexible version of assignment, and then to try to *eliminate it altogether*."

Alan Kay.
The Early History of SmallTalk.

Technical Skills: Mutation

“Favor Immutability.”



(chapter 4)

Technical Skills

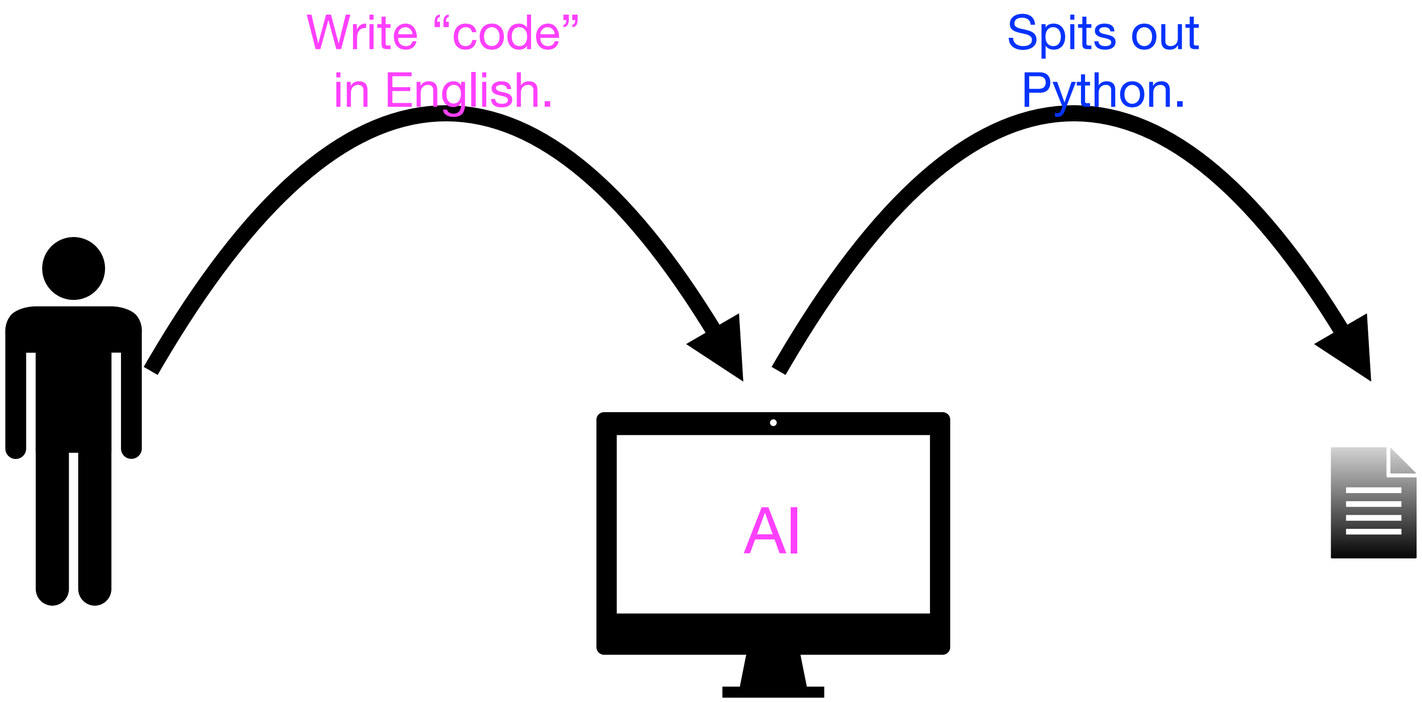
to develop software in a socially responsible manner, we need

- learn to write focused purpose statements
  - learn to check focused purpose statements
- distinguish between atomic and composite units of code
  - challenge any method that is longer than 10 lines
- use mutation when needed
  - avoid it whenever possible

I wish you could …

- .. read
- .. write
- .. stop, drop, …

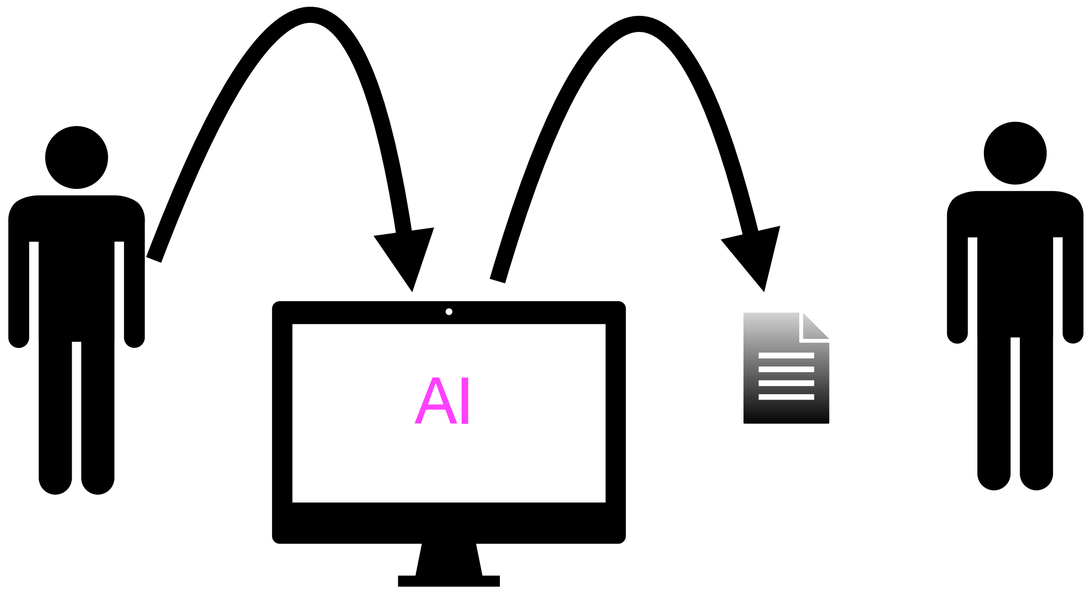
I wish you could read & write.



What if the co-worker is an AI?

Reads Python “code”
and
checks whether it has
something to do with
the English.

I wish you could write.

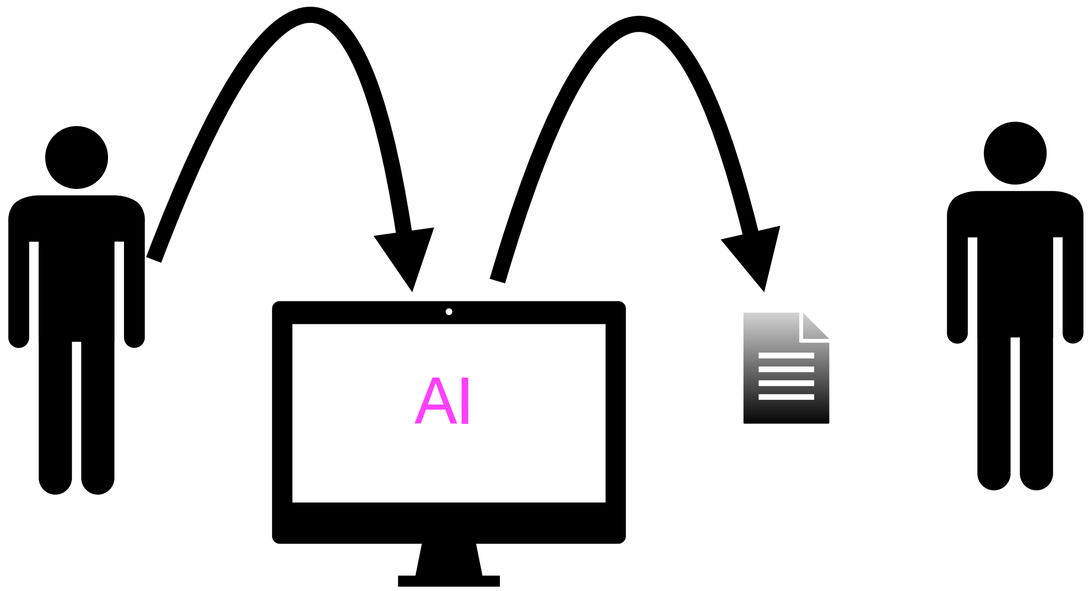


What if the co-worker is an AI?

What does it take to get an AI to create the code that we want?

- precise, focused *purpose statements*
- with a clear distinction between
  - atomic units of code
  - composite units of code
- a properly organized plan

I wish you could read.



What if the co-worker is an AI?

What does it take to get an AI to create good code?

Why should we doubt AI-generated code?

- bugs and git-reverts
- safety holes
- security holes

I wish you could read.

Study finds AI assistants help developers produce code that's more likely to be buggy

At the same time, tools like Github Copilot and Facebook InCoder make developers believe their code is sound

https://www.theregister.com/2022/12/21/ai_assistants_bad_code/

"We find disconcerting trends for maintainability. Code churn -- the percentage of lines that are reverted or updated less than two weeks after being authored -- is projected to double in 2024 compared to its 2021, pre-AI baseline. We further find that the percentage of 'added code' and 'copy/pasted code' is increasing in proportion to 'updated,' 'deleted,' and 'moved 'code. In this regard, AI-generated code resembles an itinerant contributor, prone to violate the DRY-ness [don't repeat yourself] of the repos visited."

New GitHub Copilot Research Finds 'Downward Pressure on Code Quality'
Visual Studio Magazine, 25 Jan 2024

I wish you could read.

by **Matt Asay**
Contributing Writer

“Google’s Chrome team, writes, “AI tools help experienced developers more than beginners.”

Why AI coding assistants are best for experienced developers

6. Discussion

Asleep at the Keyboard?
CACM 21 Jan 2025

Copilot’s response to our scenarios is mixed from a security standpoint, given the large number of generated vulnerabilities (across all axes and languages, 39.33 % of the top and 40.73 % of the total options were vulnerable). The security of the top options are particularly important—novice users may be more likely to accept the ‘best’ suggestion.

How do I learn to read and write?

Books.

I wish you would stop, drop, …

… and reflect.

The End

Thanks for listening.

Technical Skills: Classes

Every class must convey the
“how” in a concise manner. It
is either *atomic* or *composite.*

An *atomic* class comes with
a purpose statement that explains
what unique information it represents.

A *composite* class comes with
a purpose statement that indicates
to *which* classes it delegates.

Favor Composition
Over Inheritance.

(chapter 4)