

# Pair Programming and Skill-Based Interviews



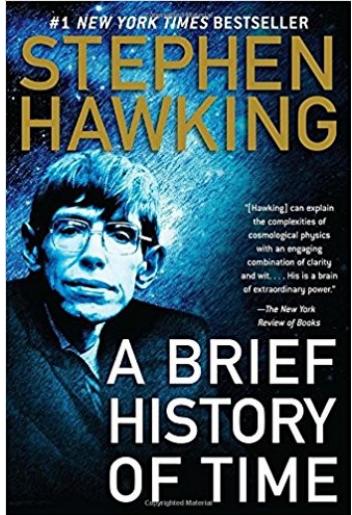
# The Story So Far ...

- We want to deliver and support a quality software product
  - We understand process and design
  - We understand quality assurance
  - We don't really understand humans
- How should we make process and design designs the first time ...
- ... in light of how humans work?

# One-Slide Summary

- There are many **programming** and **development** approaches for improving aspects of software development
  - Tackling **abstraction**, **modularity**, changing **requirements**, and software **quality**
- **Agile** development focuses on reducing the cost to respond to requirements change
- **Pair programming** is a well-studied technique within Agile involving a driver and a navigator; it increases development **time** but decreases **defects**.
- **Skill-based interviews** help companies rule out poor-fit employees. They include both **programming** and **behavioral** questions. Interviewees should show and communicate **all aspects** of the software engineering process.

# A Brief History of Time



- Structured Programming (1950-1960+)
  - Structured Programming Theorem (1966)
- Object-oriented Programming (1970-1980+)
  - Dominant in 1990+
- Aspect-oriented Programming (1997+)
- Iterative & Incremental Development (1960+)
- Agile Development (2001+)
- Scrum (1986+, 2001+)

# And Many More

- Adaptive software development (1970)
- Rapid application development (1991)
- Unified Process (1994)
- Dynamic Systems Development Method (1994)
- Crystal Clear (1996)
- Extreme Programming (1996)
- Feature-Driven Development (1997)
- TDM TLA! “So what?”



The untold history of web development:

1990: HTML invented

1994: CSS invented to fix HTML

1995: JS invented to fix HTML/CSS

2006: jQuery invented to fix JS

2010: AngularJS invented to fix jQuery

2013: React invented to fix AngularJS

2014: Vue invented to fix React & Angular

2016: Angular 2 invented to fix AngularJS & React

2019: Svelte 3 invented to fix React, Angular, Vue

2019: React hooks invented to fix React

2020: Vue 3 invented to fix React hooks

2020: Solid invented to fix React, Angular, Svelte,  
Vue

2020: HTMX 1.0 invented to fix React, Angular,  
Svelte, Vue, Solid

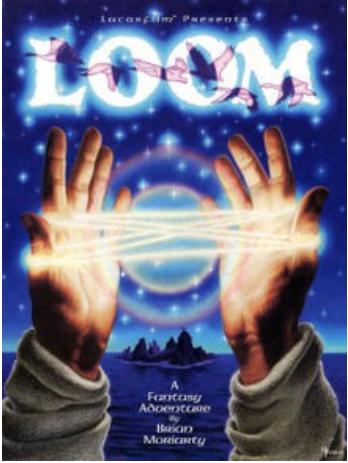
2021: React suspense invented to fix React,  
again

2023: Svelte Runes invented to fix Svelte

2024: jQuery still used on 75% of websites

# The How And Why

- *Structured*: structure source code control flow to improve clarity, quality, and development time
- *OO*: structure source code by encapsulating data and methods to improve reusability and modularity
- *AOP*: structure source code by separating cross-cutting concerns to increase modularity
- *IID*: develop software through repeated cycles in small portions to improve user involvement, reduce variability and development effort
- *Agile*: develop software through collaborating cross-functional teams, small work increments and tight feedback loops to ...
- *Scrum*: small teams complete work units in short sprints and hold daily stand-up meetings to rapidly react to change



# Common Threads (1/2)

- With respect to software source code
- **Abstraction** (e.g., inheritance, polymorphism) allows the same code to be applied to different data
  - This saves development and QA effort
- **Modularity** (e.g., interfaces) permits a **separation of concerns**, allowing code both sides of the interface to be changed independently
  - This reduces maintenance (change) effort

# Common Threads (2/2)



- With respect to software development
- **Smaller work increments** reduce the effort lost to, and minimize risk from, changing **requirements**
- **Smaller teams** and **customer involvement** reduce risks from changing **requirements** and align software with stakeholders
- **Quality techniques** (continuous integration, unit testing, pair programming, design patterns, refactoring, etc.) assure **quality**

# Agile Development



DODGE

Should have put more points in it, eh?

- Software development is considered **agile** when the team requires relatively little time, cost, personnel, and resources to respond to a requirement change
- Team **autonomy**: the extent to which the software team has authority and control in making decisions to carry out the project
- Team **diversity**: the extent to which team members have different functional backgrounds, skills, expertise and experience



# Does Agile Work? (1/2)

- “A systematic review of empirical studies of agile software development up to and including 2005 was conducted. The search strategy identified 1996 studies, of which 36 were identified as empirical studies. ... We identified a number of reported benefits and limitations of agile development within each of these themes. **However, the strength of evidence is very low**, which makes it difficult to offer specific advice to industry.”

[ Dyba and Dingsoyr. *Empirical studies of agile software development: A systematic review.* ]

# Does Agile Work? (2/2)

- “Using an integrated research approach that combines quantitative and qualitative data analyses ... of survey responses of 399 software project managers suggest ... team autonomy has a **positive effect on response efficiency** [on-time completion] and a **negative effect on response extensiveness** [software functionality], and that team diversity has a **positive effect on response extensiveness.**”

[ Lee and Xia. *Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility.* ]

THAT WHICH  
DOES NOT  
KILL ME...  
  
GIVES ME  
**XP**

# Extreme Programming

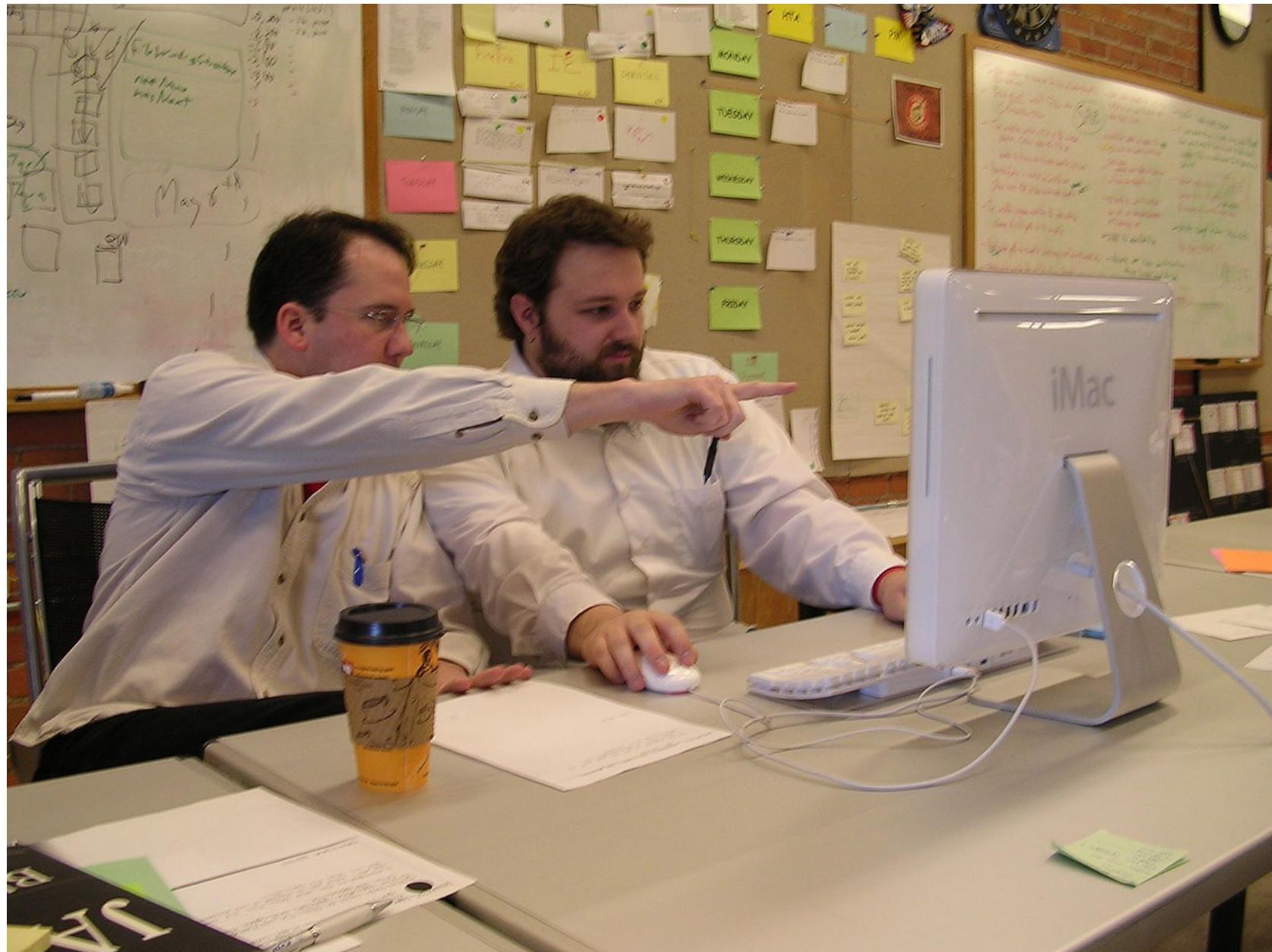
- **Extreme programming** (XP) is a software development methodology for improving software quality and responsiveness to changing customer requirements
  - It is one type of agile software development
  - It advocates frequent "releases" in short development cycles
    - This improves productivity and introduces checkpoints at which new customer requirements can be adopted
  - It advocates pair programming, extensive code review, unit testing, code readability, etc.

# Pair Programming



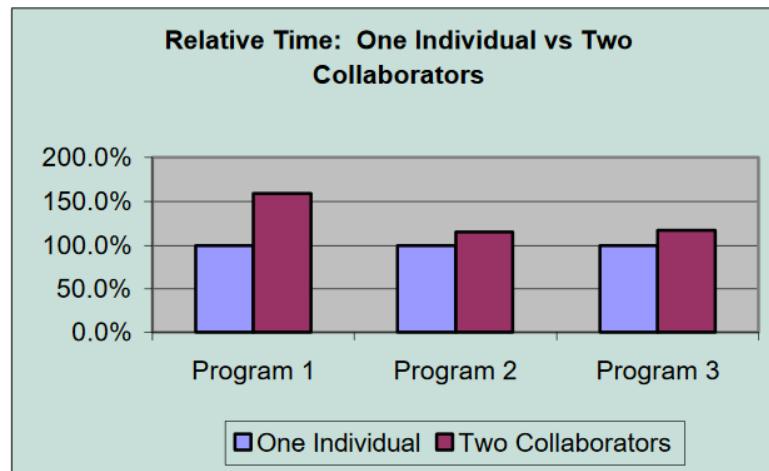
- **Pair programming** refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test.
- The pair is made up of a **driver**, who actively **types** at the computer or records a design; and a **navigator** (or **observer**), who **watches** the work of the driver and attentively **identifies** problems, **asks** clarifying questions, and **makes** suggestions. Both are also continuous brainstorming partners.

# One Thousand Words



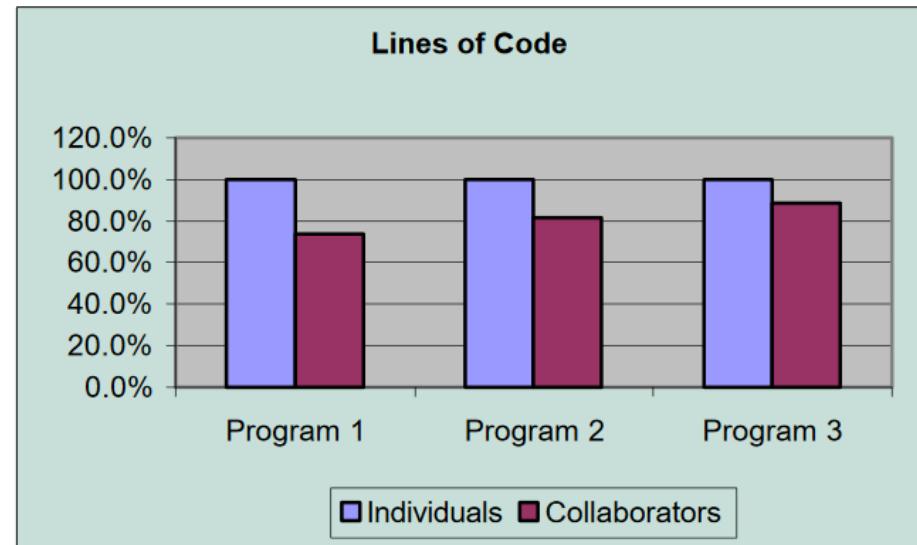
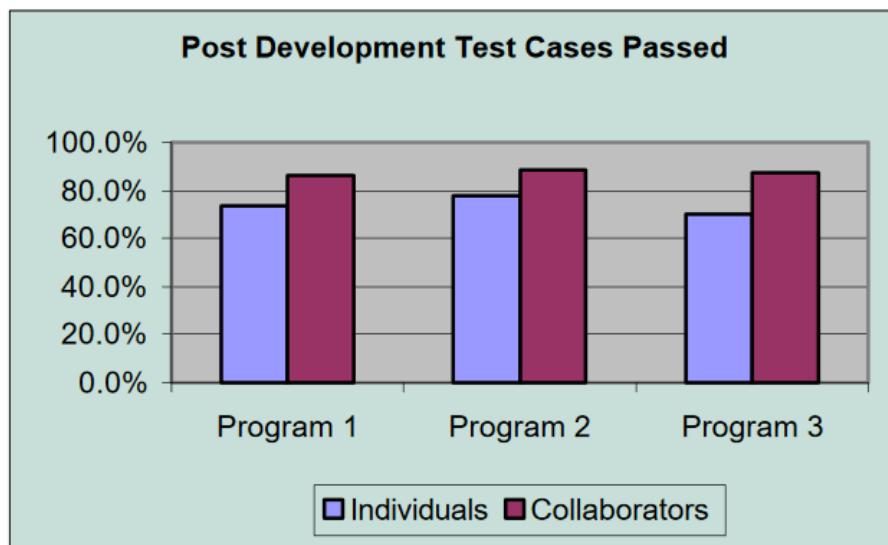
# Pair Programming and Programmers

- Surveys of professional programmers
  - 90+% “enjoyed collaborative programming more than solo programming”
  - 95% were “more confident in their solutions” when they pair programmed
- **Increases development cost** by 15% to 100%



# Pair Programming and Program Quality

- Reduces defects by 15%
- Reduces code size by 15%



[ Cockburn and Williams. *The Costs and Benefits of Pair Programming.* ]

# Example Process Decision

(suppose 15% slower coding total, 15% fewer bugs total)

- 50,000 LOC program
- Coding at 50 LOC/hour (wait, what?)
- Defect rate of 10 defects / KLOC
- Defect fix time of 10 hours /defect
- As Individuals:
  - 1,000 hr coding + 5,000 hr fixing defects = 6,000
- As Pairs:
  - 1,150 hr coding + 4,250 hr fixing defects = 5,400
- Do these numbers match your project?

# Important Math Note

- The total “costs” and “benefits” of pair programming are **already included** in the numbers quoted to you
  - For example, when we say pair programming increases costs by 15% to 100%, if it's 15%, you **do not** first multiply by 2 (for the pair) and then calculate the 15%
  - The cost of having two people work **is already factored in** to the 15% to 100% overhead. So the 100% worst-case is the “multiply by 2”, but the 15% case is “we are magically much faster working together”. **That's the pair benefit!**
  - Similarly, we **do not** both say “the code is 15% smaller and then the 15% smaller code has 15% fewer defects on top of that” - the 15% fewer defects is already the total benefit. No double counting (pro or con)!

# Pair Programming vs. Education

- North Carolina State University and the University of California at Santa Cruz, did extensive pair programming studies with ~1200 beginning computer science students (CS1) and with ~300 third/fourth year software engineering students over three year periods
  - Students who paired in CS1 were more likely to attempt CS2 (77% vs. 62%)
  - Students who paired in CS1 were more likely to major in CS (57% vs. 34% at NCSU, 25% vs. 11% at UCSC,  $p < 0.01$ )

# Pair Programming vs. Outcomes

(Laurie Williams et al., p < 0.018)

Table 1: Exam Scores

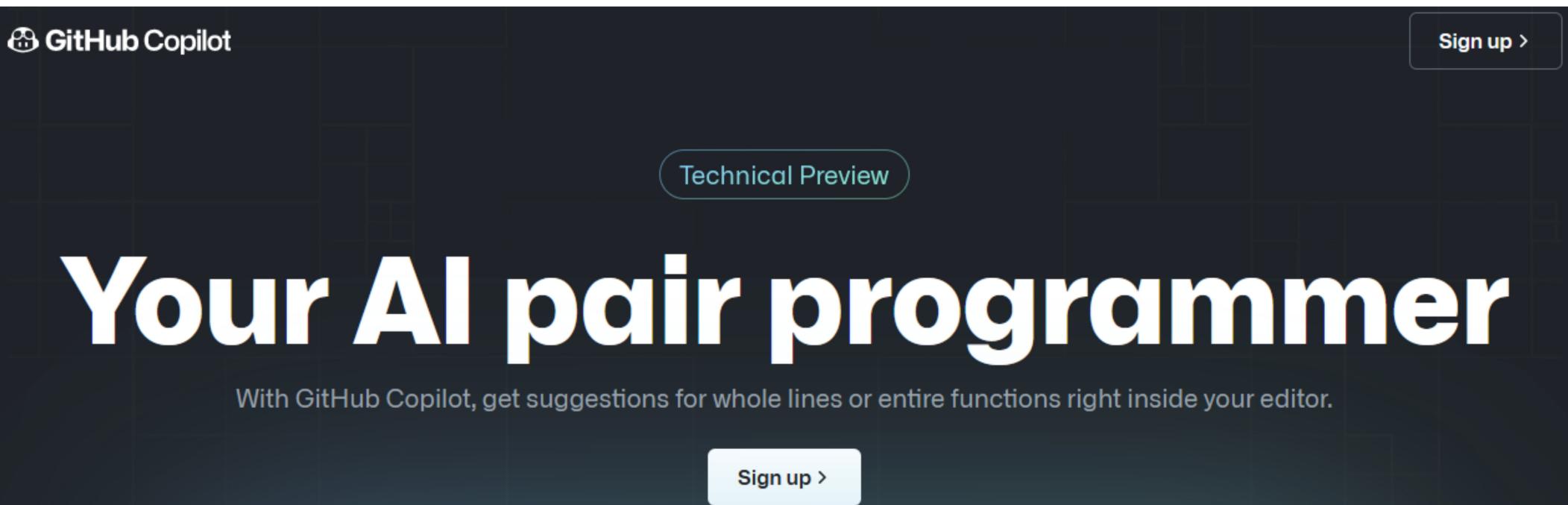
<b>Exam</b>	<b>Paired Mean</b>	<b>Paired Standard Deviation</b>	<b>Solo Mean</b>	<b>Solo Standard Deviation n</b>
Midterm 1	78.7	11.8	73.4	13.8
Midterm 2	65.8	24.2	49.5	27.2
Final	74.1	16.5	67.2	18.4

Table 2: Programming Project Scores

<b>Exam</b>	<b>Paired Mean</b>	<b>Paired Standard Deviation</b>	<b>Solo Mean</b>	<b>Solo Standard Deviation</b>
Project 1	94.6	5.3	78.2	26.5
Project 2	86.3	19.7	68.7	33.7
Project 3	73.7	27.1	74.4	29.0

# GitHub Copilot

- Despite its advertising, GitHub's **Copilot** is a code synthesis and recommendation tool (akin to, but more complex than, autocomplete) and is *not* pair programming.
  - It is neither a driver nor a navigator. (See later lectures.)



The image shows a screenshot of the GitHub Copilot homepage. At the top left is the GitHub logo and the text "GitHub Copilot". At the top right is a "Sign up >" button. In the center is a "Technical Preview" button. Below these, the main headline reads "Your AI pair programmer" in large white text. A subtext below it says, "With GitHub Copilot, get suggestions for whole lines or entire functions right inside your editor." At the bottom center is another "Sign up >" button.

GitHub Copilot

Sign up >

Technical Preview

# Your AI pair programmer

With GitHub Copilot, get suggestions for whole lines or entire functions right inside your editor.

Sign up >

# Agile Criticism

- “The agile movement is in some ways a bit like a teenager: very self-conscious, checking constantly its appearance in a mirror, accepting few criticisms, only interested in being with its peers, rejecting en bloc all wisdom from the past, just because it is from the past, adopting fads and new jargon, at times cocky and arrogant. But I have no doubts that it will mature further, become more open to the outside world, more reflective, and therefore, more effective.”

– Philippe Kruchten, 2011

# Trivia: Books and Magic

- Each one of these is either a *Magic: The Gathering* card or a *Romance* book available on Amazon. Identify four.
  - “Unlikely Alliance”
  - “Dangerous Curves”
  - “Lay Bare the Heart”
  - “Honor’s Price”
  - “Blazing Hope”
  - “Desert Rogues”
  - “Rogue’s Passage”



# Trivia: International Business and Social Media

- Zhang “Nancy” Zetian ( 章泽天 ), born in 1993, is a Chinese businesswoman and investor credited with becoming China's youngest female billionaire (\$8.2B USD in 2015). She initially rose to fame from the popularity of a wholesome photograph posted to Baidu Tieba. Give her nickname ( 奶茶妹妹 ).



# Trivia: US Political Commentary

- This Jon Stewart protege started in improvisational theater, worked at Second City, and is a fan of Catholicism and Tolkein. Associated with a Bump, a Rally, a SuperPAC, and a White House Correspondent's Association Dinner, this comedian and commentator has received nine Emmies, two Grammies, two Peabodies, and has written a #1 best-selling book.

# Trivia: Geography

- This northeastern Italian city is situated across a group of over 100 islands. The land areas are separated by canals and linked by hundreds of bridges. Once the capital of its own Republic, it is also home to the Bridge of Sighs.



# Psychology: Intelligence?

- In psychology, **g** (**general intelligence factor**) is a variable that summarizes positive correlations among cognitive tasks. It typically accounts for 40-50% of between-individual performance on many different cognitive tests. The most widely-accepted modern theories of intelligence incorporate it.
- Problem: if you are not careful, you mistakenly measure socioeconomic status (etc.) instead of intelligence.
- Interestingly, **g** is highly **heritable**. How?

# Psychology: Natural Experiment



- We can study parents, children and cognitive ability ... but how do we help rule out socioeconomic status and parenting choices?
- Identical twins share 100% of their genes
- Fraternal twins share ~50% of their genes
- Twins reared together share certain environmental aspects (e.g., religious practices at home)
- *Twins reared apart*, however ... !
  - Separated at birth, adopted by different families

# Psychology: Minnesota Twin Registry

- Tracks over 8,000 twin pairs for use in psychological studies
- Early study by T. Bouchard found that identical twins reared apart had an equal chance of being similar to their co-twins in terms of personality, interests, and attitudes as twins reared together
  - Differences must be due to the environment
  - Similarities are likely due to genetics, especially if twins share trait X far more often than others

# Psychology: Twins Reared Apart

- 70% of variance in IQ was found to be associated with genetic variation
- On temperament, occupational and leisure-time interests, social attributes, monozygotic twins reared apart are as similar as monozygotic twins reared together
  - Study carefully controls for SES, pre- and post-reunion contact, parent education, etc.

[ Bouchard, Lykken, McGue, Segal, Tellegen. *Sources of Human Psychological Differences: The Minnesota Study of Twins Reared Apart.* ]

# Psychology: Twins Reared Apart

- “... does not detract from the value or importance of parenting, education, and other propaedeutic interventions.”
- “MZA twins are so similar in psychological traits because their identical genomes make it probable that their effective environments are similar. Specific mechanisms by which genetic differences in human behavior are expressed in phenotypic differences are largely unknown. It is a plausible conjecture that a key mechanism by which the genes affect the mind is indirect, and that genetic differences have an important role in determining the **effective psychological environment** of the developing child. Infants with different temperaments elicit different parenting responses. ...”

# Psychology: Twins Reared Apart

- This sort of research continues to this day
  - “While adoption studies have provided key insights into the influence of the familial environment on IQ scores of adolescents and children, few have followed adopted offspring long past the time spent living in the family home [...] These families, tested previously on measures of IQ when offspring averaged age 15, were assessed a second time nearly two decades later (M offspring age = 32 years) [...] The heritability was estimated to be 0.42 [95% CI 0.21, 0.64].”
  - [ Emily Willoughby et al. *Genetic and environmental contributions to IQ in adoptive and biological families with 30-year-old offspring.* Intelligence, Vol 88, Sep-Oct 2021. ]

# Psychology: Heritable Traits

- One interpretation is “biology is destiny”
  - Be careful!
- Alternatively (abusing math for clarity), if the correlation of intelligence between twins is 0.7, the dual is that the environment and your choices control 30% of it!
- Also: if effective learning environments exist and vary between individuals, pay attention as a manager when directing training

# Typical CS Hiring Process

- Someone at the company, typically a recruiter or an engineer, gets your resume and puts it into their pipeline
  - If they're interested, you'll probably get one or two phone screen interviews
  - If you pass the phone screen, you'll probably be invited to interview with the company on-site
  - Depending on the company, you may then have some follow-up phone calls to find a team to be placed on
  - If they offer you a job, you'll negotiate the offer to end up with the best deal possible
  - If this particular offer is the best out of all the offers you've received, you accept!
- This can be spread out as much as a month and a half, or as compact as two weeks

# Skill-Based Technical Interview Goals

- “The interview process at Google has been designed (and redesigned!) from the ground up to avoid false positives. **We want to avoid making offers to candidates who would not be successful at Google.** (The cost of this unfortunately includes more false negatives, which are times when we turn down somebody who would have done well.)”

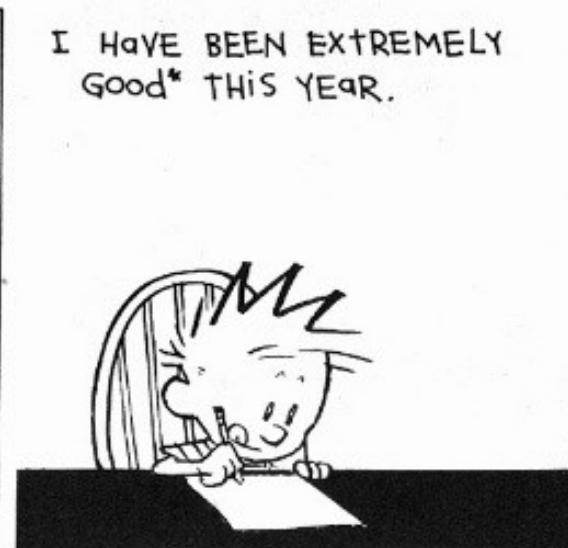


# Google's Information Needs: “A Good Fit”

- Are you good at CS? [Skill]
  - Can you write and test code?
  - Are you someone they want writing code they will use and depend on?
  - Can you think on your feet?
- Can you communicate CS concepts? [Behavioral]
  - Can you explain your ideas to coworkers?
  - Are you someone who would make their team better?
- Are you a nice person? [Behavioral]
  - Are you someone they want to work with?
  - And are you friendly enough to chat with every day?

# Interview Format

- “For about 45 minutes you meet with a single technical interviewer, who will present a programming problem and ask you to work out one or more solutions to it.”
- Interviewer perspective: “you know in the first ten minutes”



# A Medium-Difficulty Example

## (“The Two-Sum Problem”)

- You are given an array of  $n$  integers and a number  $k$ . Determine if there is a pair of elements in the array that sums to exactly  $k$ .
- For example, given the array [1, 3, 7] and  $k = 8$ , the answer is “yes,” but given  $k = 6$  the answer is “no.”

# Questions You Ask

- Can you modify the array? Yes.
- Do we know something about the range of the numbers in the array? No, they can be arbitrary integers.
- Are the array elements necessarily positive? No, they can be positive, negative, or zero.
- Do we know anything about the value of  $k$  relative to  $n$  or the numbers in the array? No, it can be arbitrary.
- Can we consider pairs of an element and itself? No, the pair should consist of two different array elements.
- Can the array contain duplicates? Sure, that's a possibility.
- What about integer overflow? Don't worry about it.

# Example Solution 1: Brute Force

- $O(N^2)$  time,  $O(1)$  space

```
boolean sumsToTarget (int[]arr, int k) {  
    for (int i = 0; i < arr.length; i++) {  
        for (int j = i + 1; j < arr.length; j++) {  
            if (arr[i] + arr[j] == k) {  
                return true;  
            } } }  
    return false;  
}
```

# Example Solution 2: Hashing

- Expected  $O(N)$  time, expected  $O(N)$  space

```
boolean sumsToTarget (int[]arr, int k) {  
    HashSet < Integer > values = new HashSet < Integer > ();  
    for (int i = 0; i < arr.length; i++) {  
        if (values.contains (k - A[i])) return true;  
        values.add (A[i]);  
    }  
    return false;  
}
```

# Other Solutions

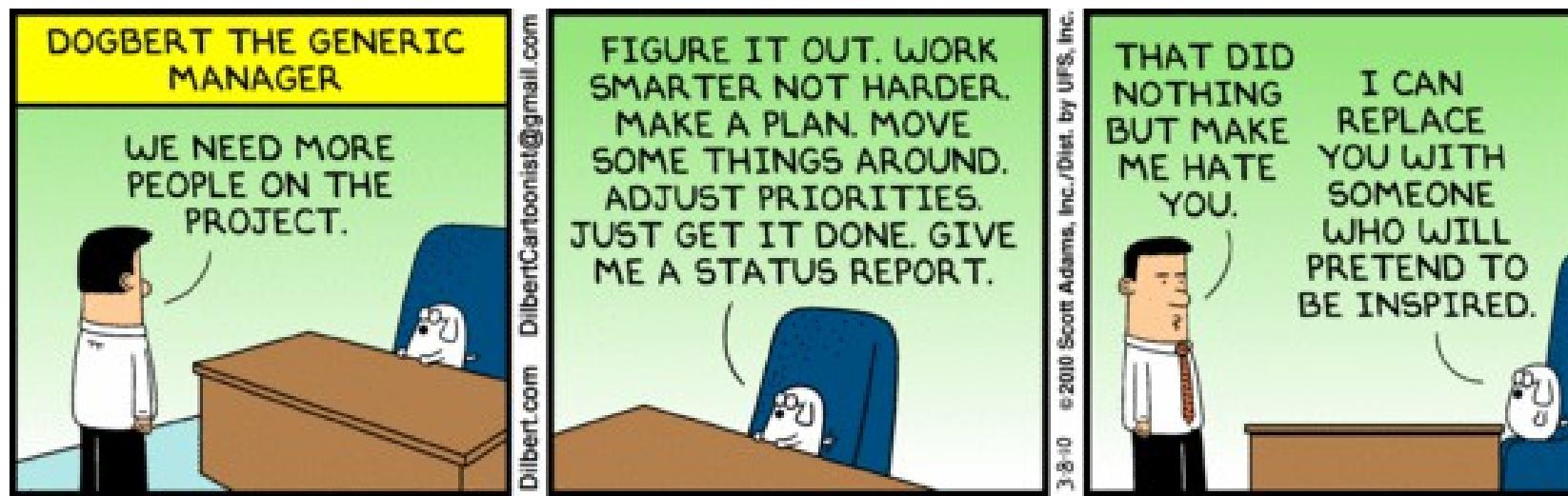
- Sort and Binary Search
  - $O(n \log n)$  time,  $O(\log n)$  to  $O(1)$  space
- Radix Sort and Walk Inward
  - $O(n \log X)$  time,  $O(\log n)$  space

```
boolean sumsToTarget (int[] arr, int k) {  
    Arrays.radixSort(arr);  
    int lhs = 0, rhs = arr.length - 1;  
    while (lhs < rhs) {  
        int sum = arr[lhs] + arr[rhs];  
        if (sum == k) return true;  
        else if (sum < k) lhs++;  
        else rhs--;  
    }  
    return false;  
}
```

# Were those solutions good?

- What are your thoughts?

```
boolean sumsToTarget (int[] arr, int k) {  
    for (int i = 0; i < arr.length; i++) {  
        for (int j = i + 1; j < arr.length; j++) {  
            if (arr[i] + arr[j] == k) {  
                return true;  
            } } }  
    return false;  
}
```



# Software Microcosm

- If you do not convey that you have mastered skill X, they will assume you have not
- They will assume how you write this program is how you will write every program
- They are looking for reasons to reject you
- “Saying true things” vs. “Not saying false things”
- Thus, even though the problem is small and simple, you should show all of the steps of the software engineering process

# Do Not Forget

- Even though the problem is small, you should
  - Perform requirements elicitation
  - Ask about functional and quality properties
  - Talk about process considerations
    - Talk about how you design for maintainability
  - Write commented code, including method-level and statement-level documentation (what/why)
  - Write tests that show off corner cases
    - Talk about other approaches to QA (within reason)

# Top 10 Mistakes in Interview Prep

[Gayle McDowell, *Cracking the Coding Interview*]

#1 Practicing on a computer

**#2 Not rehearsing behavioral questions**

#3 Not doing a mock interview

#4 Trying to memorize solutions

#5 Not solving problems out loud

#6 Rushing

#7 Sloppy coding (bad style),

#8 Not testing

#9 Fixing mistakes carelessly

#10 Giving up

# Behavioral Questions

- What is your greatest weakness?
- Tell me about a time you missed a deadline.
- Tell me about a time you experienced a conflict with a teammate.
- Very easy to sound unimpressive if you have not practiced!

# Situation, Action, Result

- Recommendation: structure your responses (especially to “negative” questions):
  - Situation: describe objectively
  - Action: what did you do?
  - Result: how were things better after?
- Be specific, not arrogant



THE SELF ESTEEM BOOSTER

# Resume and Interview “Stats”

- Your resume says you worked on *XYZ Project*. What was the most challenging aspect of that?
  - What did you learn the most from? What was the most interesting? What was the hardest bug? What did you enjoy the most? What was the biggest conflict? Most significant requirements change?
- What is the largest program (LOC) you have written? Modified? What is the largest number of tests you have written? Worked with? What is the largest team you have worked with? What is the largest process you automated? How many customers have you spoken to?

# What do we know? Little so far!

## Dazed: Measuring the Cognitive Load of Solving Technical Interview Problems at the Whiteboard

Mahnaz Behroozi<sup>1</sup>, Alison Lui<sup>2</sup>, Ian Moore<sup>1</sup>, Denae Ford<sup>1</sup>, Chris Parnin<sup>1</sup>

<sup>1</sup>North Carolina State University, Raleigh, NC, USA

<sup>2</sup>University of Notre Dame, Notre Dame, IN, USA

{mbehroo, ipmoore, dford3, cjparin}@ncsu.edu, alison.m.lui.2@nd.edu

### ABSTRACT

Problem-solving on a whiteboard is a popular technical interview technique used in industry. However, several critics have raised concerns that whiteboard interviews can cause excessive stress and cognitive load on candidates, ultimately reinforcing bias in hiring practices. Unfortunately, many sensors used for measuring cognitive state are not robust to movement. In this paper, we describe an approach where we use a head-mounted eye-tracker and computer vision algorithms to collect robust metrics of cognitive state. To demonstrate the feasibility of the approach, we study two proposed interview settings: on the whiteboard and on paper with 11 participants. Our preliminary results suggest that the whiteboard setting pressures candidates into keeping shorter attention lengths and experiencing higher levels of cognitive load compared to solving the same problems on paper. For instance, we observed 60ms shorter fixation durations and 3x more regressions when solving problems on the whiteboard. Finally, we describe a vision for creating a more inclusive technical interview process through future studies of interventions that lower cognitive load and stress.

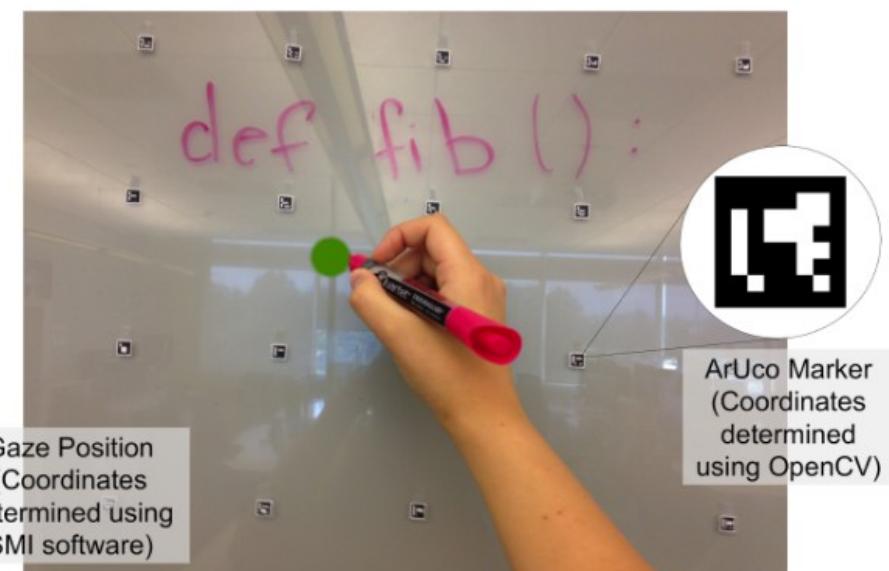


Figure 1: Feasibility study of using ArUco markers to calculate regressions.

• Medium and affordances. Whiteboards are often selected for

# Suggestion

- Remember this “from the other side”
- Ask what people look for during interviews!
  - Guest speakers in classes are great for this!



# Questions?

- HW2
- Exam 1 coming up!

