

# Working in Teams

Martin Kellogg

# Working in Teams

Today's agenda:

- **Reading Quiz**
- Team structures and roles
- Deciding who to work with: interviews
  - how to be interviewed
  - how to interview

# Reading quiz: working in teams

Q1: “Two-pizza” teams have which of the following benefits or characteristics, according to the article (select all that apply):

- A. Pizzas are cheap, so management can easily bribe them to work late
- B. Software built by two-pizza teams must have clear APIs
- C. They take on all responsibilities needed to support their customers
- D. They’re organized around specific skills (e.g., databases or testing)

Q2: **TRUE** or **FALSE**: the “favorite coding question” mentioned in one of the readings is ambiguous

# Reading quiz: working in teams

Q1: “Two-pizza” teams have which of the following benefits or characteristics, according to the article (select all that apply):

- A. Pizzas are cheap, so management can easily bribe them to work late
- B. Software built by two-pizza teams must have clear APIs
- C. They take on all responsibilities needed to support their customers
- D. They’re organized around specific skills (e.g., databases or testing)

Q2: **TRUE** or **FALSE**: the “favorite coding question” mentioned in one of the readings is ambiguous

# Reading quiz: working in teams

Q1: “Two-pizza” teams have which of the following benefits or characteristics, according to the article (select all that apply):

- A. Pizzas are cheap, so management can easily bribe them to work late
- B. Software built by two-pizza teams must have clear APIs
- C. They take on all responsibilities needed to support their customers
- D. They’re organized around specific skills (e.g., databases or testing)

Q2: **TRUE** or **FALSE**: the “favorite coding question” mentioned in one of the readings is ambiguous

# Working in Teams

Today's agenda:

- Reading Quiz
- **Team structures and roles**
- Deciding who to work with: interviews
  - how to be interviewed
  - how to interview

# Why have a team at all?

Benefits:

# Why have a team at all?

Benefits:

- Attack **bigger problems** in a short period of time
- Utilize the **collective experience** of everyone



# Why have a team at all?

## Benefits:

- Attack **bigger problems** in a short period of time
- Utilize the **collective experience** of everyone

Humans are **social** -  
we naturally work in  
teams

# Why have a team at all?

## Benefits:

- Attack **bigger problems** in a short period of time
- Utilize the **collective experience** of everyone

## Risks:

# Why have a team at all?

## Benefits:

- Attack **bigger problems** in a short period of time
- Utilize the **collective experience** of everyone

## Risks:

- **Communication** and coordination issues
- **Groupthink**: diffusion of responsibility; going along
- Working by **inertia**; not planning ahead
- **Conflict** or mistrust between team members

# What impacts team success?

# What impacts team success?

- Presence of a **shared mission** and goals
- **Motivation** and commitment of team members
- **Experience** level (and presence of experienced members)
- Team size
  - and the need for bounded yet sufficient **communication**
- Team **organization**
- **Reward structure** within the team
  - incentives, enjoyment, empowerment (ownership, autonomy)

# Communication: powerful but costly

- Communication requirements increase with **team size**
  - Implication: having the right size team is important!

# Communication: powerful but costly

- Communication requirements increase with **team size**
  - Implication: having the right size team is important!
- Everybody to everybody: **quadratic** (aka  $O(n^2)$ ) cost
  - Implication: need more efficient structures in large teams

# Communication: powerful but costly

- Communication requirements increase with **team size**
  - Implication: having the right size team is important!
- Everybody to everybody: **quadratic** (aka  $O(n^2)$ ) cost
  - Implication: need more efficient structures in large teams
- Every attempt to communicate is a chance to **mis-communicate**
  - But not communicating will guarantee miscommunicating



# Communication: powerful but costly

- Communication requirements increase with **team size**
  - Implication: having the right size team is important!
- Everybody to everybody: **quadratic** (aka  $O(n^2)$ ) cost
  - Implication: need more efficient communication methods as **team size** increases
- Every attempt to communicate is costly
  - But not communicating is also costly

Group project advice: choose a communication platform for your group at the start and stick to it. Slack, Discord, Teams, whatever: just choose one. **Protip:** not email.

# Team structures and roles

**Definition:** A *team structure* is a way of organizing a team of people.

- you're probably familiar with some from other domains, such as professional sports or musical theater

# Team structures and roles

**Definition:** A *team structure* is a way of organizing a team of people.

- you're probably familiar with some from other domains, such as professional sports or musical theater

Just like in those domains, software teams typically have multiple possible *roles*, each of which has *responsibilities* and *competencies*.

# Team structures and roles

**Definition:** A *team structure* is a way of organizing a team of people.

- you're probably familiar with some from other domains, such as professional sports or musical theater

Just like in those domains, software teams typically have multiple possible *roles*, each of which has *responsibilities* and *competencies*.

- By analogy, roles on a sports team might be defined by competency in a skill (e.g., pitching, goalie) or by responsibility (e.g., offense, defense)

# Roles in a software team

# Roles in a software team

- Software engineer (aka “individual contributor”)
  - most of you will probably end up with this role
- Engineering manager (aka “people manager”)
- Project manager (“PM”)
- Tester/quality assurance
- Operations engineer/site reliability engineer
- Business expert (in Agile teams, usually playing the role of “customer”)
- etc.

# Roles in a software team

- Software engineer (aka “individual contributor”)
  - most of you will probably end up
- Engineering manager (aka “people manager”)
- Project manager (“PM”)
- Tester/quality assurance
- Operations engineer/site reliability engineer
- Business expert (in Agile teams, usually the “customer”)
- etc.

These could be all different team members, or some members could span multiple roles.

# Roles in a software team

- **Software engineer** (aka “individual contributor”)
  - most of you will probably end up
- Engineering manager (aka “people manager”)
- Project manager (“PM”)
- Tester/quality assurance
- Operations engineer/site reliability engineer
- Business expert (in Agile teams, usually the “customer”)
- etc.

These could be all different team members, or some members could span multiple roles.



# What does a software engineer do?

- Mostly, what we're talking about in this class!
  - Write code (or direct an AI model that's writing code)
  - Write specifications
  - Fix bugs
  - Triage bug reports
  - Write tests
  - Debug
  - etc.

# Varieties of software engineer

- Not all engineers have the same background, experience, expertise, etc.
  - typical division is between “junior”, “senior”, and “staff” engineers (these roles have different names)
    - e.g., “principal” engineer at Amazon

# Varieties of software engineer

- Not all engineers have the same background, experience, expertise, etc.
  - typical division is between “junior”, “senior”, and “staff” engineers (these roles have different names)
    - e.g., “principal” engineer at Amazon
- Some places ask engineers to further specialize: e.g., “backend engineer”, “front end engineer”

# Varieties of software engineer

- Not all engineers have the same background, experience, expertise, etc.
  - typical division is between “junior”, “senior”, and “staff” engineers (these roles have different names)
    - e.g., “principal” engineer at Amazon
- Some places ask engineers to further specialize: e.g., “backend engineer”, “front end engineer”
- High-level engineers are sometimes “tech leads” or “architects”

# Leveling

- In big tech, often associated with a number
  - numbers vary by company!
  - e.g., Google L3 = Amazon SDE I (4) = Microsoft SDE (59/60)
- Usually associated with your salary
- Promotion: usually requires you to already be operating “at the next level up”
- More senior engineers are expected to mentor more junior engineers

✕ Apple

✕ Facebook

✕ Amazon

✕ Microsoft

✕ Google

ICT2 Junior Software Engineer	E3	SDE I L4	SDE 59	L3 SWE II
ICT3 Software Engineer	E4	SDE II L5	SDE II 61	L4 SWE III
ICT4 Senior Software Engineer	E5	SDE III L6 Senior SDE	Senior SDE 63	L5 Senior SWE
ICT5	E6	Principal SDE L7	Principal SDE 65	L6 Staff SWE
ICT6	E7	Senior Principal SDE L8	Partner 68	L7 Senior Staff SWE
Distinguished Engineer	E8	Distinguished Engineer L10	67	L8 Principal Engineer
Senior Distinguished Engineer	E9		69	L9 Distinguished Engineer
Engineering Fellow			70 Distinguished Engineer	L10 Distinguished Engineer
			80 Technical Fellow	L10 Google Fellow

from [levels.fyi](https://levels.fyi)

# What's an engineering manager?

- Mostly manages people - not that different from managers in other industries, sometimes
  - performance evaluations, talk to upper management, etc
- Some places also ask the engineering manager to be the PM - see next slide
- At some places (e.g., Google is famous for this), managers are expected to be technically-competent, too

# What's a program/product manager?

- Manage the product!
- Often this includes, but isn't limited to:
  - talking to customers
  - writing design docs
  - interfacing with other teams
  - thinking about how different parts of the project fit together



# What's a program/product manager?

- Manage the product!
- Often this includes, but isn't limited to:
  - talking to customers
  - writing design docs
  - interfacing with other teams
  - thinking about how different parts of the project fit together

“my job is sending emails that **people actually read**” - a friend of mine who is a PM at Microsoft

# What does a tester do?

- Mostly write tests
- This role isn't as common anymore: most places I'm familiar with have asked software engineers to also be responsible for testing
  - E.g., Microsoft laid off most of its Windows QA teams in 2014 and replaced them with crowdsourced testing (“Windows Insiders”)

# What does a site reliability engineer do?

- “SRE” is a Google-specific term for an engineer whose job is to keep systems running
  - on-call more often than “line” engineers
  - knows how several services work, can debug issues in any of them
  - spends ~50% of their time on ops, ~50% on developing new automation to improve ops
  - we’ll talk more about this role (and how companies that don’t have an SRE role, like Amazon, manage) in the DevOps lectures

# Business expert/other

- Depending on the industry and company, a software team might have other kinds of experts embedded into it
  - e.g., a UX expert, an expert in some business process that the team is automating
- Even on very software-focused teams, once the org size becomes large enough, there are other roles:
  - high-level managers or ICs might have executive assistants
  - HR, payroll, all the other things a small company needs to function

# Interns!

- Most software teams have interns, at least some of the time
- No offense, but most interns are pretty useless
  - Not enough time to ramp up on a very large codebase
- Typically requires planning from the team: what will the interns work on this year?
  - Ideally a well-defined, well-scoped project
- Training interns is a recruitment tool and a service to their future teams (ideally at your company, since they had a great time!)

# How is a software team structured?

- Tricky balance among
  - **progress** on the project/product
  - **expertise** and knowledge (and need to train team members)
  - **communication** needs

# How is a software team structured?

- Tricky balance among
  - **progress** on the project/product
  - **expertise** and knowledge (and need to train team members)
  - **communication** needs

“A team is a set of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable.”

– Katzenbach and Smith

# Organizing principle

Two common principles used to organize teams:



# Organizing principle

Two common principles used to organize teams:

- **Functional** teams organized around a particular goal
  - everyone is working towards a **common goal**
  - typically have people with **different skills**
  - e.g., the “Google Slides” team or the “DynamoDB” team

# Organizing principle

Functional teams are the most typical organization for teams in big tech

Two common principles used to organize teams

- **Functional** teams organized around a particular goal
  - everyone is working towards a **common goal**
  - typically have people with **different skills**
  - e.g., the “Google Slides” team or the “DynamoDB” team

# Organizing principle

Two common principles used to organize teams:

- **Functional** teams organized around a particular goal
  - everyone is working towards a **common goal**
  - typically have people with **different skills**
  - e.g., the “Google Slides” team or the “DynamoDB” team
- **Skill** teams organized around a particular set of skills
  - typically are **specialists** in some hard-to-acquire skills
  - often “**consult**” for functional teams
  - e.g., AWS ARG, most IT teams

# Organizing principle

Two common principles used to organize teams

- **Functional** teams organized around a particular functional area
  - everyone is working towards a common goal
  - typically have people with different skills
  - e.g., the “Google Slides” team or the “DynamoDB” team
- **Skill** teams organized around a particular set of skills
  - typically are **specialists** in some hard-to-acquire skills
  - often “consult” for functional teams
  - e.g., AWS ARG, most IT teams

Skill teams are more common for SDEs at non-tech companies

# Team size

- most teams are relatively small
  - Amazon's famous "two-pizza" rule: teams should be small enough that two pizzas can feed everyone
  - 6-10 people is typical
- larger-scale organization of teams varies a lot by company
  - "divisional" approaches are common

# Key questions for any software team

- Who makes **decisions**? Is there a process for making decisions that impact most of the team?

# Key questions for any software team

- Who makes **decisions**? Is there a process for making decisions that impact most of the team?
- Who is **responsible** for various important tasks:
  - scheduling/process
  - testing/quality assurance
  - documentation (spec, design, write-ups, presentations)
  - build/release preparation
  - external communication (with other teams, customers, upper management)

# Key questions for any software team

- Who makes **decisions**? Is there a process that impact most of the team?
- Who is **responsible** for various important tasks?
  - scheduling/process
  - testing/quality assurance
  - documentation (spec, design, write-ups, presentations)
  - build/release preparation
  - external communication (with other teams, customers, upper management)

**Group project advice:**  
write down the answers  
to these questions for  
your team (we will ask!)



# More group project advice: motivation

How can you get the most out of your team members?

# More group project advice: motivation

How can you get the most out of your team members?

- give them **specific, small, attainable** goals that they can visualize

# More group project advice: motivation

How can you get the most out of your team members?

- give them **specific, small, attainable** goals that they can visualize
- have frequent communication and updates

# More group project advice: motivation

How can you get the most out of your team members?

- give them **specific, small, attainable** goals that they can visualize
- have frequent communication and updates
- meet in person to work as much as possible (video call counts as “in-person”): **pair programming** is especially effective

Aside: pair programming

# Aside: pair programming

**Definition:** *Pair programming* refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test.

# Aside: pair programming

**Definition:** *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*, who watches the work of the driver and attentively identifies problems, asks clarifying questions, and makes suggestions. Both are also continuous brainstorming partners.

# Aside: pair programming: worth it?

Surveys of professional programmers:



# Aside: pair programming: worth it?

Surveys of professional programmers:

- 90+% “**enjoyed** collaborative programming more than solo programming”
- 95% were “more **confident** in their solutions” when they pair programmed
- **Reduces defects** by 15% and **reduces code size** by 15%

# Aside: pair programming: worth it?

Surveys of professional programmers:


- 90+% “**enjoyed** collaborative programming more than solo programming”
- 95% were “more **confident** in their solutions” when they pair programmed
- **Reduces defects** by 15% and **reduces code size** by 15%
- Increases development cost by **15% to 100%**

# Aside to the aside: “pair programming” with AI

- Marketing for modern AI tools (**large language models**) suggests that you can use a chatbot as a “pair programmer”
  - Do you think you’ll get the benefits we discussed on the last slide that you would from a human partner from an AI?

# Aside to the aside: “pair programming” with AI


- Marketing for AI tools that you can use
  - Do you think this slide that

 GitHub  
<https://github.com> > features > copilot


10/1/25

**GitHub Copilot · Your AI pair programmer**  
GitHub Copilot works alongside you directly in your editor, suggesting whole lines or entire functions for you.

[Take flight with GitHub Copilot](#) [Tutorials](#) [Copilot Business](#) [What's new](#)

 Aider  
<https://aider.chat>

**Aider - AI Pair Programming in Your Terminal**  
AI pair programming in your terminal. Aider lets you pair program with LLMs to start a new project or build on your existing codebase.

 Microsoft Learn  
<https://learn.microsoft.com> > mobility > architecture > ai-...

**AI pair programmer**  
Apr 1, 2025 — GitHub Copilot, an AI pair programmer that helps developers write code faster and with less work. GitHub Copilot is powered by OpenAI.

# Aside to the aside: “pair programming” with AI

- Marketing for modern AI tools (**large language models**) suggests that you can use a chatbot as a “pair programmer”
  - Do you think you’ll get the benefits we discussed on the last slide that you would from a human partner from an AI?

# Aside to the aside: “pair programming” with AI

- Marketing for modern AI tools (**large language models**) suggests that you can use a chatbot as a “pair programmer”
  - Do you think you’ll get the benefits we discussed on the last slide that you would from a human partner from an AI?
- The jury on this is still out:
  - some research suggests that engineers using AI assistants are *faster* [1]
  - but other research suggests the opposite - that engineers only *think* they are faster (but are actually slower) [2]

[1] *How much does AI impact development speed? An enterprise-based randomized controlled trial.* Elise Paradis, Kate Grey, Quinn Madison, Daye Nam, Andrew Macvean, Vahid Meimand, Nan Zhang, Ben Ferrari-Church, Satish Chandra. ICSE SEIP, 2025.

[2] *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity.* Joel Becker, Nate Rush, Beth Barnes, David Rein. <https://arxiv.org/pdf/2507.09089>. 2025

# More group project advice: motivation

How can you get the most out of your team members?

- give them **specific, small, attainable** goals that they can visualize
- have frequent communication and updates
- meet in person to work as much as possible (video call counts as “in-person”): **pair programming** is especially effective
- put people in small teams; minimize work done "solo"

# More group project advice: motivation

How can you get the most out of your team members?

- give them **specific, small, attainable** goals that they can visualize
- have frequent communication and updates
- meet in person to work as much as possible (video call counts as “in-person”): **pair programming** is especially effective
- put people in small teams; minimize work done "solo"
- build good **team camaraderie**



# More group project advice: motivation

How can you get the most out of your team members?

- give them **specific, small, attainable** goals that they can visualize
- have frequent communication and updates
- meet in person to work as much as possible (video call counts as “in-person”): **pair programming** is especially effective
- put people in small teams; minimize work done "solo"
- build good **team camaraderie**
- be **professional**, and your teammates often will too

# Examples of team organization

# Team organization: Microsoft (pre 2014)

- Most teams were functional, and composed of these roles:
  - **Program Manager**. Leads the technical side of a product development team, managing and defining the functional specifications and defining how the product will work.
  - **Software Design Engineer**. Codes and designs new software, often collaborating as a member of a software development team to create and build products.
  - **Software Test Engineer**. Tests and critiques software to assure quality and identify potential improvement opportunities and projects.

# Team organization: Airbnb

- fewer than ten people
- teams are functional
  - mix of engineers, product managers, designers, and data scientists
  - sometimes include domain experts: e.g., “Payments includes people from finance”
- Engineers have (relative) freedom to change teams
  - Don't need to re-apply for a job, just need manager approval

# Working in Teams

Today's agenda:

- Reading Quiz
- Team structures and roles
- **Deciding who to work with: interviews**
  - how to be interviewed
  - how to interview

# Typical SE 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 1-2 phone 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 a job, you'll negotiate the offer to end up with the best deal possible
  - If this offer is the best out of all the offers you've received, you accept!
- Can take as long as 1-2 months or as short as 10-14 days

# Goals of a technical interview

- “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.)”

# Goals of a technical interview



# Goals of a technical interview

- Determine if you have the right **technical skills**:
  - Can you write code? Test it? Document it? Etc.
  - Can you think on your feet?

# Goals of a technical interview

- Determine if you have the right **technical skills**:
  - Can you write code? Test it? Document it? Etc.
  - Can you think on your feet?
- Can you **communicate** computer science concepts?
  - Can you explain your ideas to coworkers?
  - Would you “raise the bar” for the team?

# Goals of a technical interview

- Determine if you have the right **technical skills**:
  - Can you write code? Test it? Document it? Etc.
  - Can you think on your feet?
- Can you **communicate** computer science concepts?
  - Can you explain your ideas to coworkers?
  - Would you “raise the bar” for the team?
- Are you a **nice person**?
  - Do they want to work with you?

# Goals of a technical interview

- Determine if you have the right **technical** skill!
  - Can you write code? Test it? Document it?
  - Can you think on your feet?
- Can you **communicate** computer science concepts?
  - Can you explain your ideas to coworkers?
  - Would you “raise the bar” for the team?
- Are you a **nice person**?
  - Do they want to work with you?

It's not just technical skill! Many interview questions are **behavioral**

# 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.”
  - some variations of this, such as “tell me about a technical problem you’ve solved” and “design (but don’t implement) a solution to this problem”
- Interviewer perspective: “you know in the first ten minutes”

# Example interview problem

“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.”

# Example interview problem

“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.”

**What do you do first?**  
(Hint: it's not trying to solve the problem!)

Example interview problem: ask questions!



# Example interview problem: ask questions!

- 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 interview problem: brute force

- don't prematurely optimize your solution: write something that works

# Example interview problem: brute force

- don't prematurely optimize your solution: write something that works
- e.g., for the two-sum problem, you could write:

```
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 interview problem: brute force

- don't prematurely optimize your solution: write something that works
- e.g., for the two-sum problem, you could write:

```
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;  
}
```

Usually at this point the interviewer will ask you about **how good** this solution is.

# Example interview problem: brute force

- don't prematurely optimize your solution: write something that works
- e.g., for the two-sum problem, you could write:

```
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;  
}
```

Usually at this point the interviewer will ask you about **how good** this solution is.  $O(n^2)$  time!

# Example interview problem: be clever

- once you have something that works, try to come up with a clever solution

# Example interview problem: be clever

- once you have something that works, try to come up with a clever solution
- e.g., for the two-sum problem, you might come up with:

```
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;  
}
```

# Example interview problem: be clever

- once you have something that works, try to come up with a clever solution
- e.g., for the two-sum problem, you might come up with:

```
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;  
}
```

Why is this better?



# Example interview problem

- there are lots of possible solutions to the problem
- part of your goal while you're interviewing is showing that you understand the trade-offs between them

# Example interview problem

- there are lots of possible solutions to the problem
- part of your goal while you're interviewing is showing that you understand the trade-offs between them
- think of interviewing as a **microcosm** of software engineering:
  - if you don't show them you know it, they'll assume you don't

# Example interview problem

- there are lots of possible solutions to the problem
- part of your goal while you're interviewing is showing that you understand the trade-offs between them
- think of interviewing as a **microcosm** of software engineering:
  - if you don't show them you know it, they'll assume you don't
  - implication: even though the interview problem is **small and simple**, you show try to **show all the steps** of the software engineering process

# Do Not Forget!

Even though the problem is small, you should:

- perform requirements elicitation (ask questions!)
- ask about both functional and non-functional properties
- talk about process considerations
  - e.g., mention maintainability when relevant
- write **good quality code**, including e.g., documentation, tests, etc.
  - mention things you'd be thinking about if this was part of a real system

# Interviewing mistakes

- #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

# Interviewing mistakes

#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

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!

# Behavioral questions

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!
  - 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.



# Behavioral questions

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!
  - 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.

It's easy to sound unimpressive if you haven't thought about your answers ahead of time.

# Behavioral questions

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!

What is your greatest weakness?

How can they tell about a time you missed a deadline.

if you are nice?

What about a time you experienced a conflict with a

# Behavioral questions

Remember, they want to know if you can **communicate well** and whether you are **nice**

- implication: they will ask questions to try to find out!

What is your greatest weakness?

How can they tell about a time you missed a deadline.

if you are nice? about a time you experienced a conflict with a

**lunch!**

# Interviewing: the other side

# Interviewing: the other side

- Choose the technical problem you ask carefully
  - Common solution: use the “best” interview question you’ve ever been asked
  - Alternative: base the problem on something you personally had to deal with at work
- Think through all the possible solutions to the problem
- Remember that it’s stressful for the person being interviewed!

# Interviewing: does it work?

- The answer is that we don't really know
- Technical interviews haven't been studied in depth
  - (it's too expensive to run controlled studies with real engineers...)
- But they're the industry standard, so we have to deal with them
- Open area of research!

# Takeaways

- How you organize your team can have a big impact on your productivity
- Communication is key
- For the group project, especially, make sure you decide on how you'll make decisions (no one is the manager!)
- Interviewing is a microcosm of software engineering
  - Show the interviewer what you know, even if it seems like too much for the problem at hand