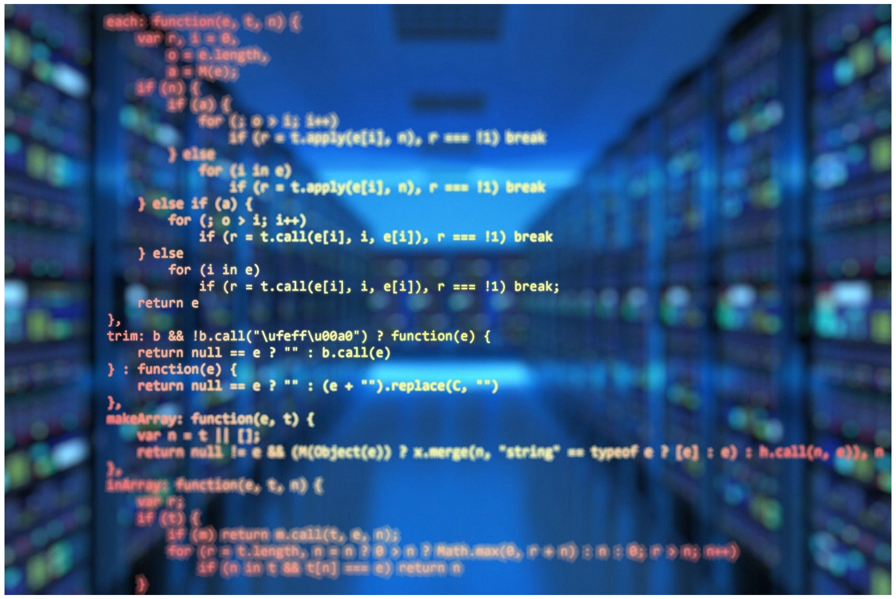


CSE 331

Software Design & Implementation

James Wilcox & Kevin Zatloukal

About Your Instructors

James Wilcox

Kevin Zatloukal

Two **professional** programmers

About Your Instructors

- Both are professional programmers
  - combined 30+ years of experience

- Built a wide range of systems and applications

Systems

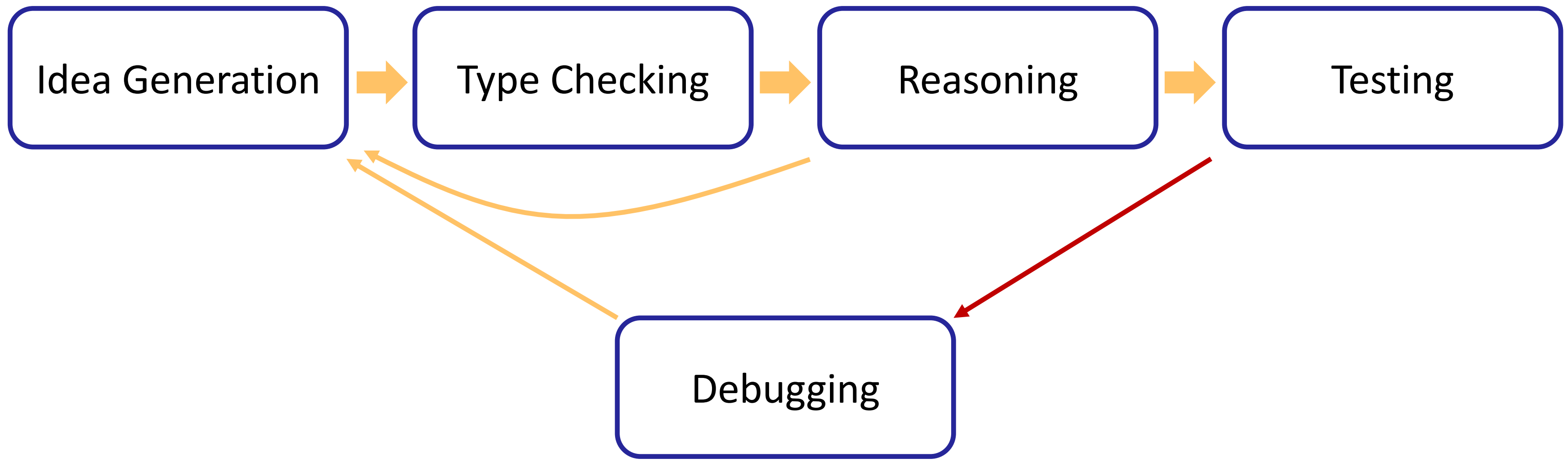
- compilers
- operating systems
- distributed systems
- networking systems
- database systems
- graphics
- …

Applications

- desktop apps
- web apps
- phone apps
- IDE
- games
- …

Software Development Process

Given: a problem description (in English)

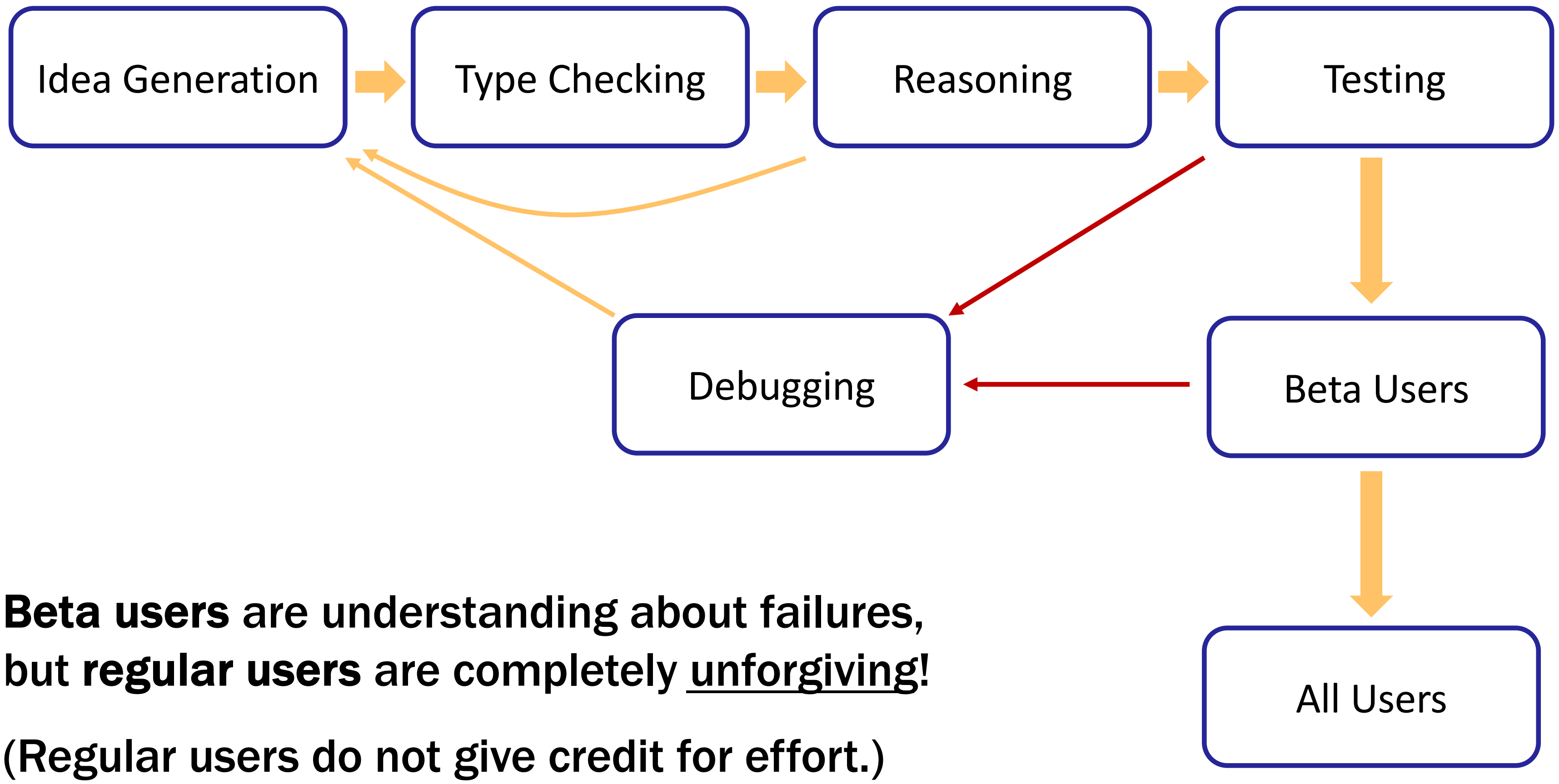


Reasoning = “thinking through” what the code does on *all* inputs

Debugging = searching back from a failure to find the bug

Software Development Process

Given: a problem description (in English)

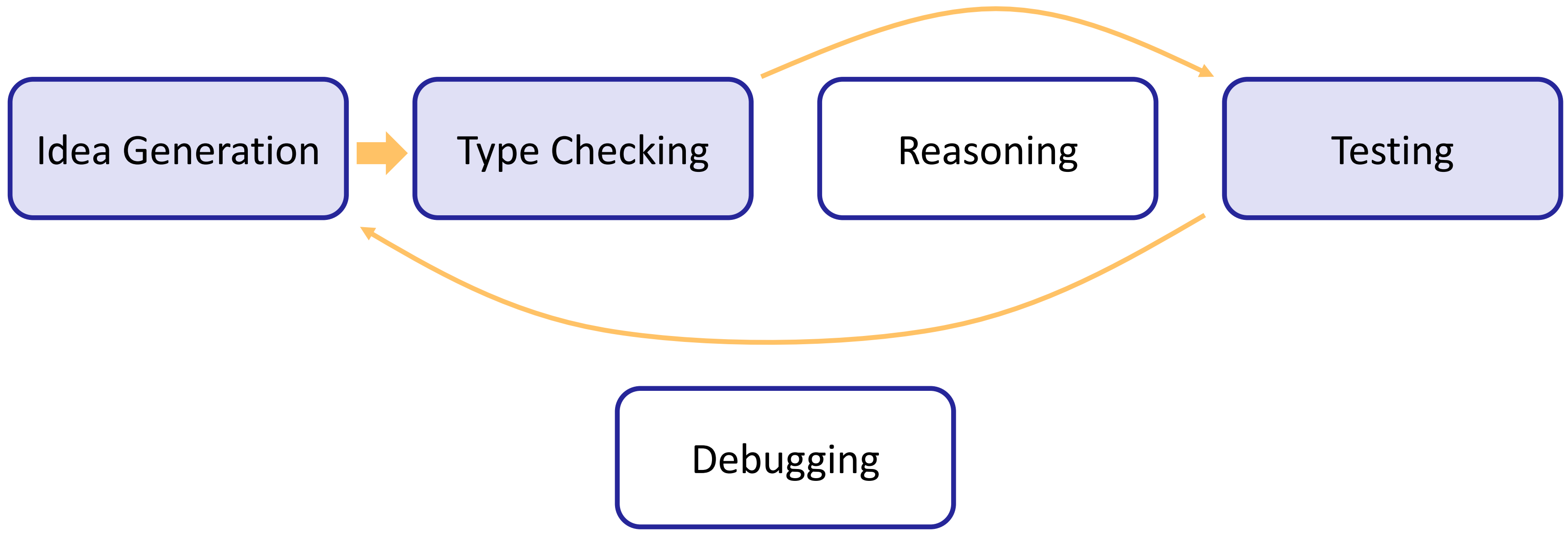


Beta users are understanding about failures,
but **regular users** are completely <u>unforgiving</u>!

(Regular users do not give credit for effort.)

Software Development Process

Given: a problem description (in English)

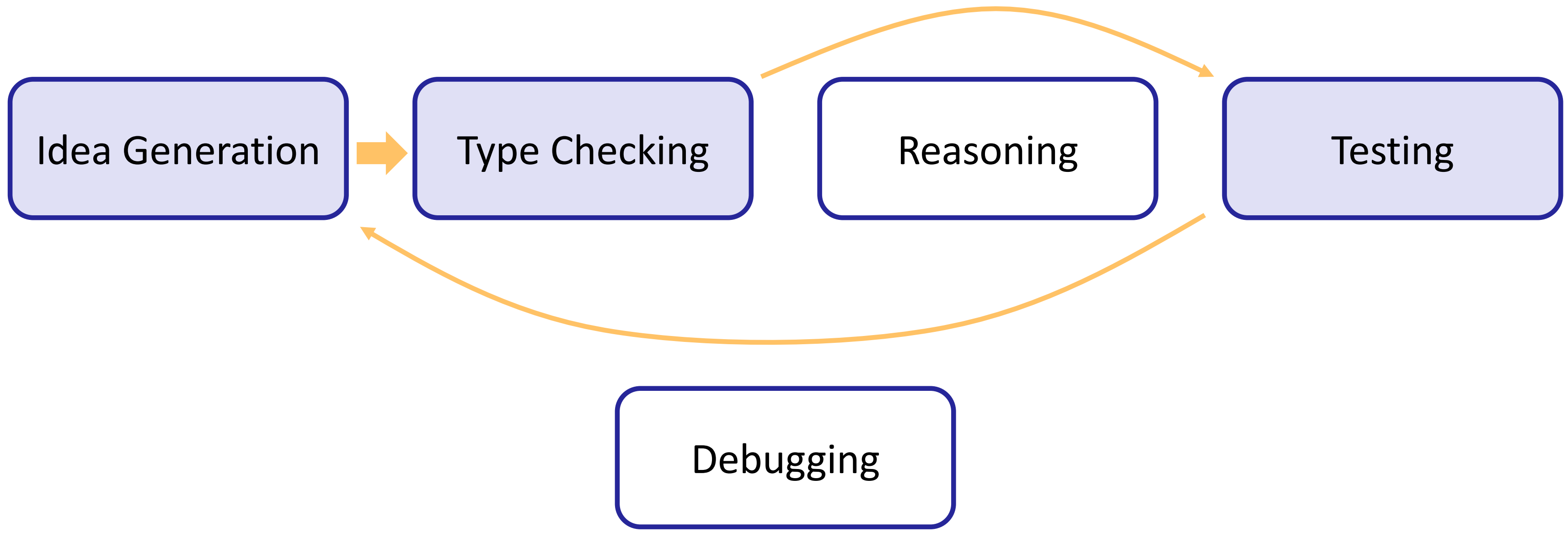


In principle, purple parts can all be automated.

My Prediction: AI will be able to write all programs that can be built using only these parts within 5 years.

Software Development Process

Given: a problem description (in English)



My Prediction: AI will be able to write all programs that can be built using only these parts within 5 years.

Corollary: expect only to be paid for reasoning and debugging (reasoning) (debugging)

"Engineers are paid to think and understand."

— Class slogan #1

Reasoning is Required

- **In principle**: you have a professional responsibility to think through what your code does on all allowed inputs
  - unethical to send code to users and just hope it works

- **In practice**: skipping reasoning up front means debugging… and then reasoning afterward
  - second approach is more work

    debugging is hard!
  - problems you get paid to solve are never solved "by accident"

    too many ideas for you to bump into the solution by accident

“Reasoning is not optional: either reason up front or debug and then reason.”

— Class slogan #2

Reasoning is Expected

- **In industry**: you will be expected to think through your code
  - standard practice is to do this *twice* (“code review”)

    you think through your code then ask someone else to also

- Professionals spend most of their coding time reasoning
  - reasoning is the core skill of programming

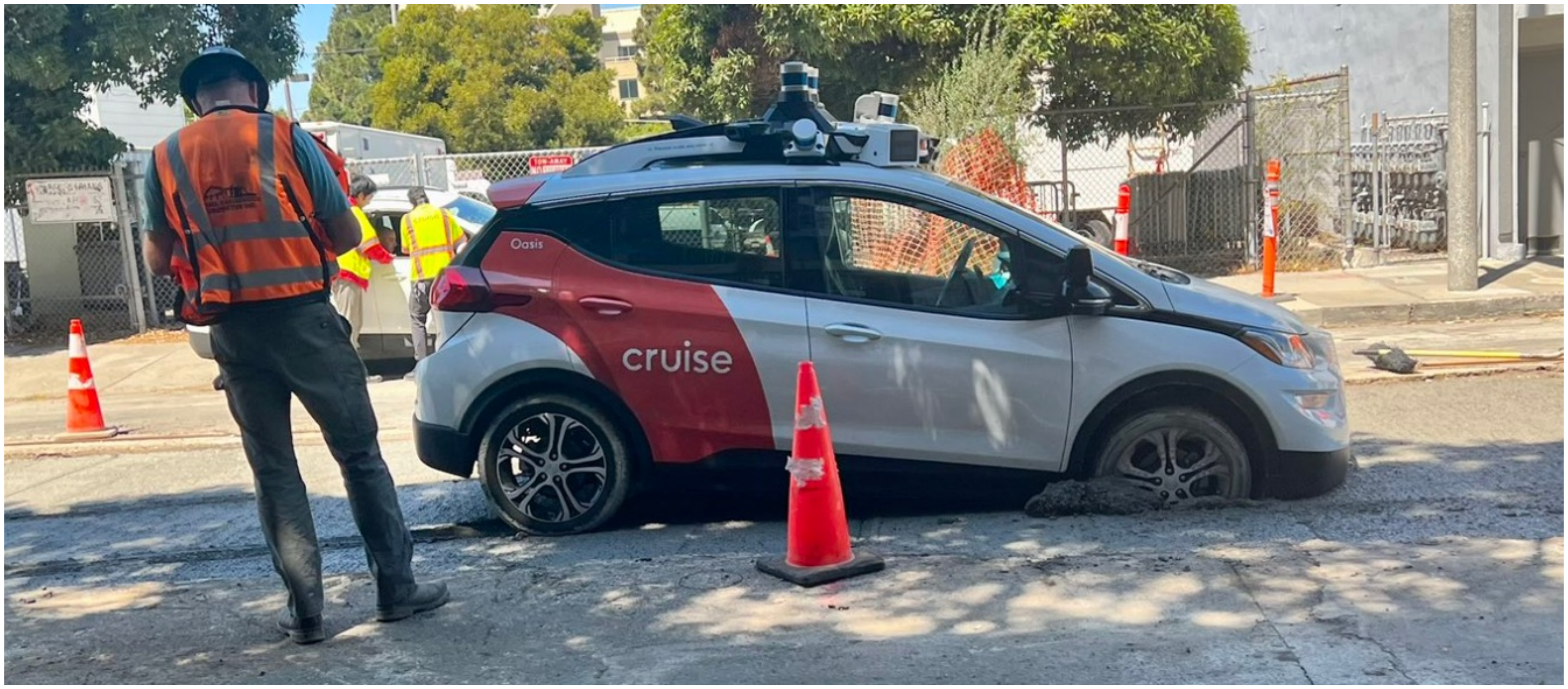
- Interviews have always been tests of reasoning
  - take the computer away so you only have reasoning
  - typical coding problem has lots of cases that are easy to miss if you don’t think through carefully
  - (not about knowing “the answer” to the question

    interviewers will throw out interviews that went too well!)

Reasoning is Unlikely to be Automated

- Reasoning & Debugging are provably impossible for a computer to solve in all cases

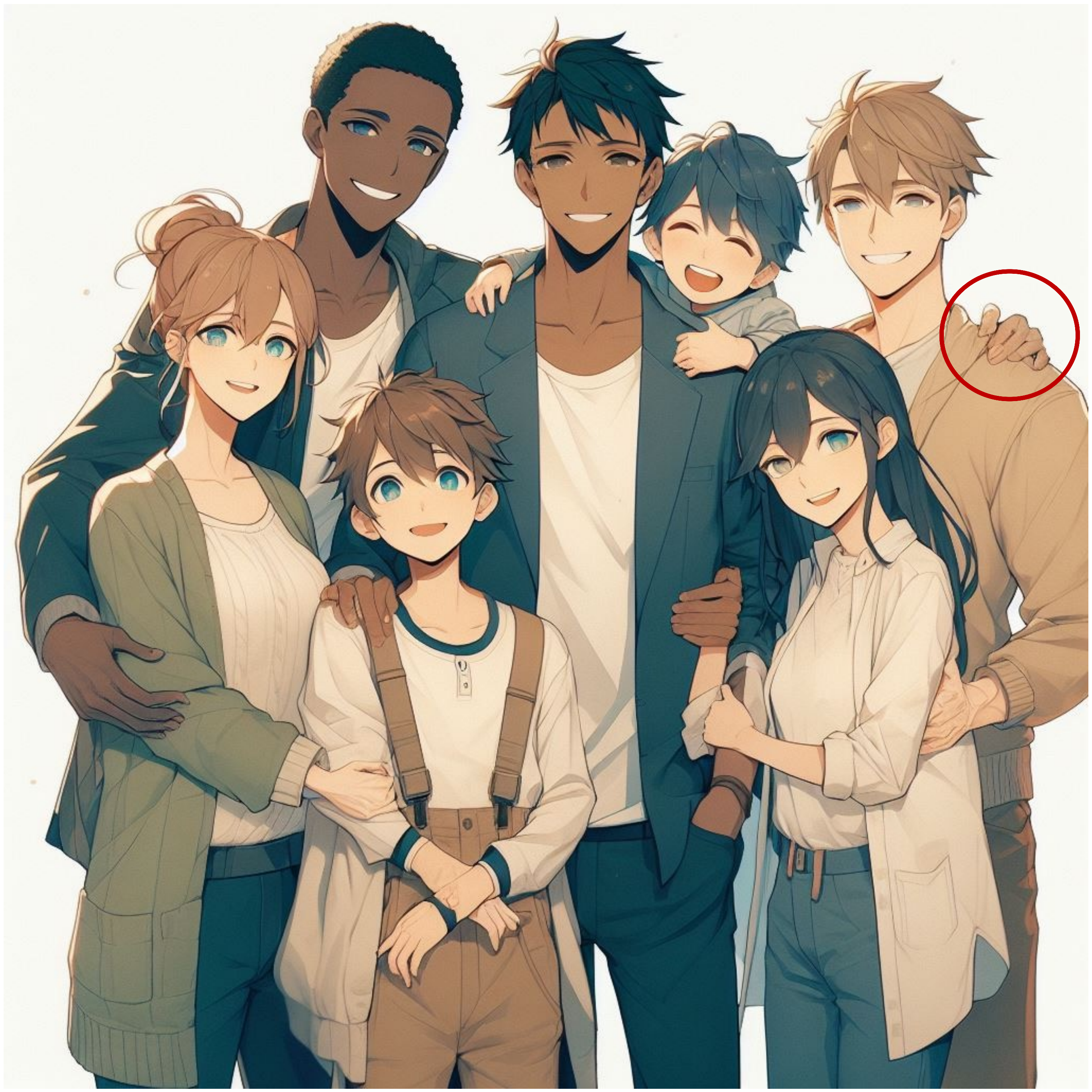
- Also seems to be hard for AI in practice
  - AI is bad at rare cases…

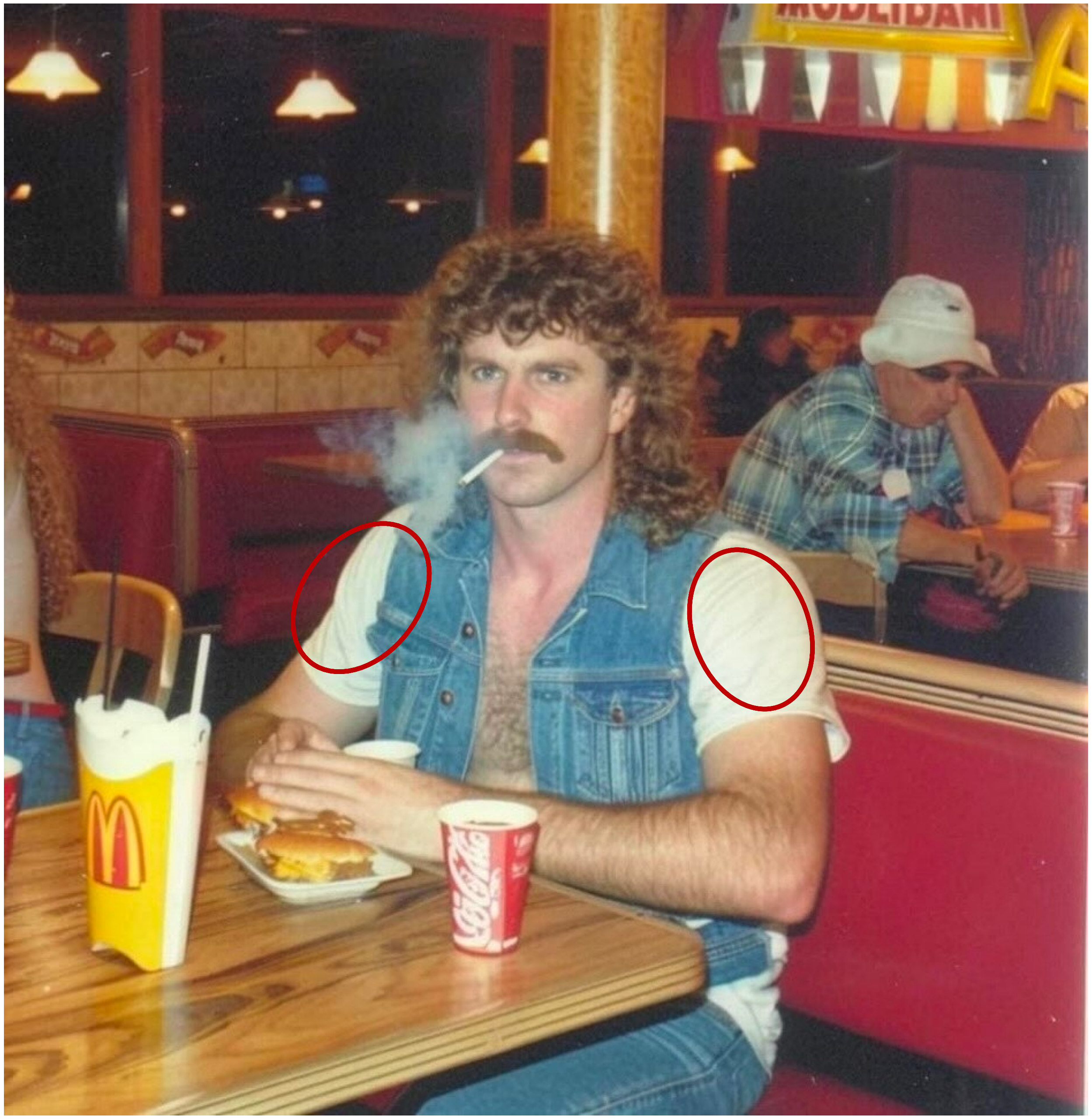


Cruise Autonomous Car Gets Itself Stuck In Wet Cement

Reasoning is Unlikely to be Automated

- Reasoning & Debugging are provably impossible for a computer to solve in all cases
- Also seems to be hard for AI in practice
  - AI is bad at rare cases (you get paid for thinking those through)
  - AI is bad at holistic understanding (necessary for debugging)




Coca-Cola

Reasoning is Unlikely to be Automated

- Reasoning & Debugging are provably impossible for a computer to solve in all cases

- Also seems to be hard for AI in practice
  - AI is bad at rare cases (you get paid for thinking those through)
  - AI is bad at holistic understanding (necessary for debugging)
  - AI fails to learn general properties despite seeing 1b+ examples

    e.g., A parent of B means that B is a child of A
  - such properties are fundamental to (formal) reasoning

    e.g., we know that a < b means b > a

    would be hard to do algebra without that knowledge

“These models have read every piece of code on Github, every StackOverflow question answer, every programming book, every tweet about coding, transcripts of every YouTube walkthrough and they still can’t code as well as I can in every situation.”

— Nat Friedman (former GitHub CEO)

Formal and Informal Reasoning

- Reasoning can be formal or informal
  - most professionals reason informally (in their head)

- We will teach formal reasoning because
  1. It’s teachable

     informal reasoning = formal reasoning concepts + intuition

     intuition is built up by *years* of practice
  2. It’s necessary for the hardest problems

     everyone needs the formal toolkit for the hardest problems

- Doesn’t matter to users which you use
  - does matter that you thought through every input

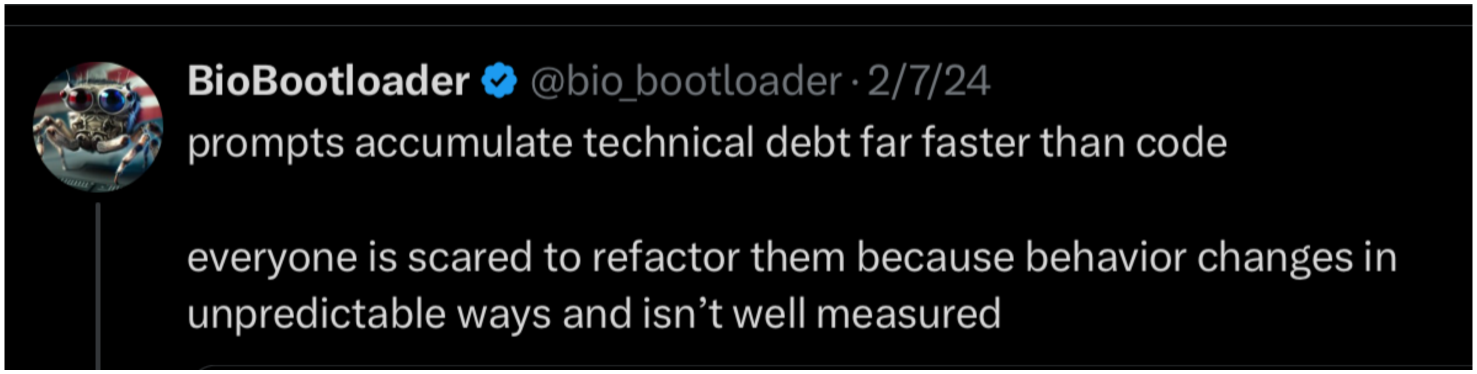
Practicing Reasoning

- Do not “just try things until the tests pass”
  - not a useful skill… AI can do that even faster
- We will *try* to make that hard
  - will reduce immediate feedback as course progresses
  - HW9 will have no immediate feedback
- But this is mainly up to you
  - points you get from passing tests don’t get you a job
  - reasoning skill you get by practicing is what matters

Other Properties of High-Quality Code

- Professionals are expected to write high-quality code
- Correctness is the most important part of quality
  - users **hate** products that do not work properly
- Also includes the following
  - easy to understand
  - easy to change
  - modular

$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\}$ will also discuss these


BioBootloader @bio_bootloader · 2/7/24
prompts accumulate technical debt far faster than code

everyone is scared to refactor them because behavior changes in unpredictable ways and isn't well measured

AI does not write high-quality code

(not easy to change, easy to understand, or modular)

Other Properties of High-Quality Code

- Note that list did not include **efficiency**

- We will focus on correctness
  - a prerequisite for efficiency
    - speed doesn't matter if the code is not correct
  - other classes give plenty of attention to efficiency

- General rules about programmers (50+ years of evidence):
  - overestimate the importance of efficiency
  - underestimate the difficulty of correctness

“Programmers overestimate the importance of efficiency
and underestimate the difficulty of correctness.”

— Class slogan #3

Will Focus on Timeless Skills

- **Focus our time on skills that will be useful 10+ years on…**
- **Not specific languages or libraries**
  - **specific knowledge is only impressive to junior programmers**
    - you will know 3-5 languages by the time you graduate!
    - you will not be impressed by someone who knows 1 more
  - **AI knows how to to write a loop in every language and how to call every well-known function in every library**
    - do not expect to be paid for this knowledge
- **Reasoning has proven to be the core skill of programming**
  - **useful for as long as humans have written code**
  - **it is language- and library-independent**

We Will Ask You to Write Code Differently

- Our goal is **not** to teach you to write code that looks exactly like what you will see in industry
  - nor is it to use the libraries most common in industry

    the most popular languages and libraries change all the time

- Our goal is to teach you to **think** through your code and to **understand** how all the parts work

- That is best served by writing slowly and carefully

- We will force that by
  1. changing programming languages to something *unfamiliar*
  2. having *unusual* coding conventions at times

Homework

- CSE 331 is a **hard** class
  - because reasoning & debugging are hard!

- Most of the work is done **outside of class**
  - university policy is 2 hours per hour of class time
  - plan for 8 hours per week, but…

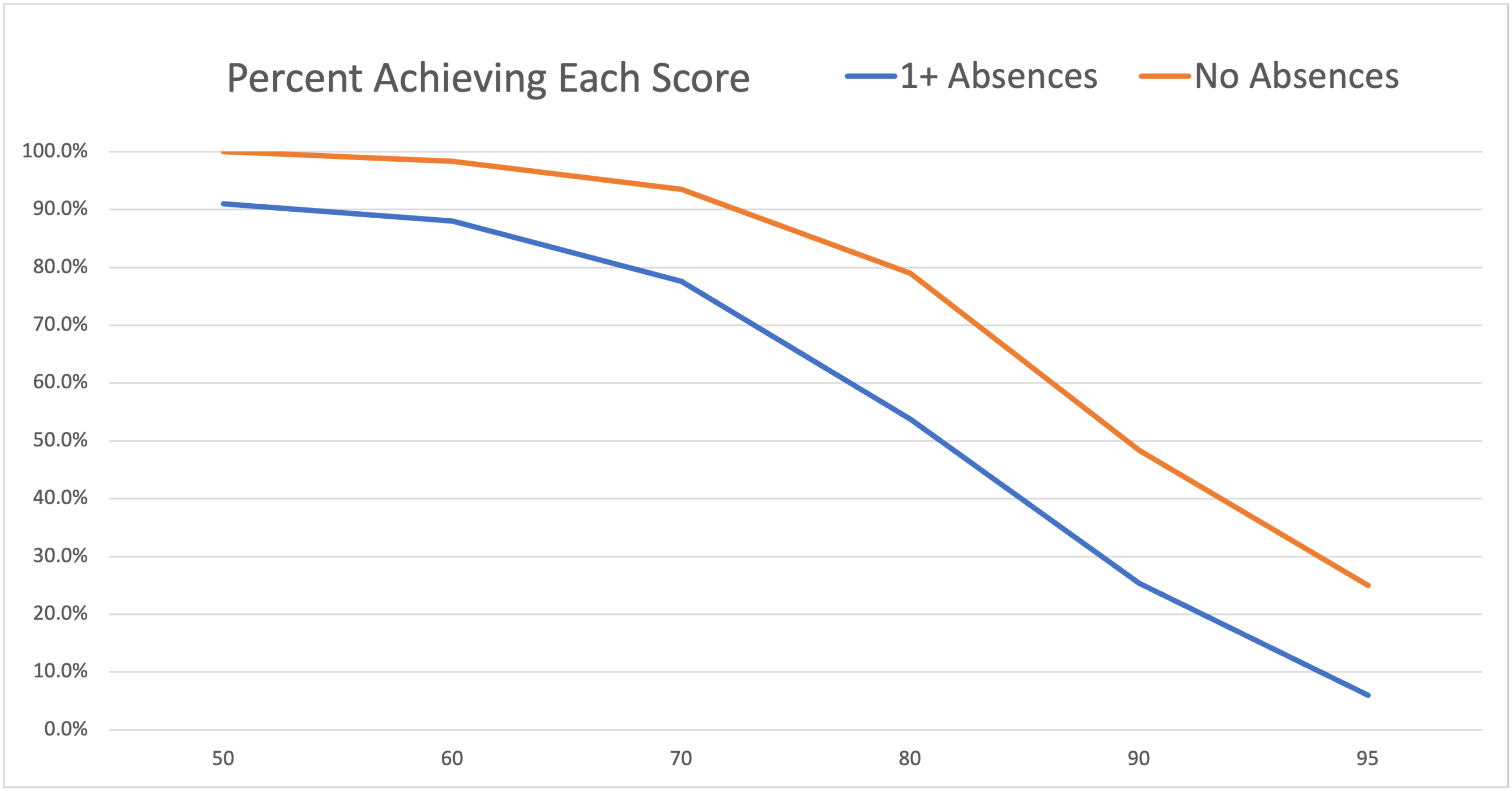
- Wide variation in time required
  - some students will average 10-15 hours

    but this is not expected!

    be sure to get help if you are averaging over 15 hours

Quiz Sections

- Get an ungraded attempt at solving HW-like problems
  - extremely beneficial to student success…



Quiz Sections

- Get an ungraded attempt at solving HW-like problems
  - extremely beneficial to student success
- Plan to attend all quarter
- If you are unable to attend, can submit online
  - submit solutions to <u>all</u> worksheet problems by 5pm
- Participation is not required, but non-participation is interpreted as confidence that you do not need extra help
  - specifically, that you do not need to attend **office hours**
  - OH time is the most limited resource in the course, so it will be **restricted** to those who attended that week's section

Late Days

- Students are allowed 4 late days total
  - anything more than 10 minutes late is 1 day
  - no more than 1 late day per assignment
- To go over these limits, you must talk to the instructors
  - no reason to ask for more until you are out of late days
- Plan to complete assignments on time
  - schedule is set up to be done on the due date
  - save late days for emergencies

Exams

- No midterm exam

- Final exam at an **unusual** time and place
  - on Tuesday, June 4th at 12:30
  - in KNE 120 (both lectures at once)

Advice

- Start homework assignments early
  - wide variation in the time required
  - never know how long **debugging** will take!
- Use the message board whenever possible
  - will get an answer promptly (during working hours)
- Do not skip class to work on homework

More Advice

- Start homework assignments early!
- Make sure you understand how lectures apply
  - seeing no connection to lecture is a giant red flag
- Focus on understanding, not points
  - understanding, not points, will not get you a job
  - losing points is the best reminder to review later

Class slogans

“Engineers are paid to think and understand.”

— Class slogan #1

“Reasoning is not optional: either reason up front or debug and then reason.”

— Class slogan #2

“Programmers overestimate the importance of efficiency and underestimate the difficulty of correctness.”

— Class slogan #3