WHAT'S THE MOST AMERICAN MOVIE EVER MADE?
THAT'S EASY. GHOSTBUSTERS.
PEOPLE DISCOVER THAT THERE'S LIFE BEYOND THE PALE OF DEATH. SO, THEY START A SMALL BUSINESS AND CASH IN.
MOST OF THAT FILM IS ABOUT BUYING AN OFFICE, GETTING BETTER CLIENTELE, FIGHTING GOVERNMENT REGULATORS, AND INCREASING YOUR WORKFORCE.
THIS IS A FILM WHERE THE AFTERLIFE IS PROVEN TO BE REAL, YET THERE'S AN ENTIRE SCENE DEVOTED TO SALARY NEGOTIATION.
I GUESS YOU NEVER REALIZE HOW WEIRD CULTURE IS WHILE YOU'RE INSIDE IT.
WHAT'S THE MOST AMERICAN BOOK?
MOBY-DICK.
VIOLENT MAN HAS A CONFUSING REVENGE FANTASY AGAINST A CHEAP SOURCE OF OIL.
HOLY CRAP.
smbc-comics.com

Dynamic Analysis

One-Slide Summary

- A **dynamic analysis** runs an instrumented program in a controlled manner to collect information which can be analyzed to learn about a property of interest.
- Computing test coverage is a dynamic analysis.
- Instrumentation can take the form of source code or binary rewriting.
- Dynamic analysis limitations include efficiency, false positives and false negatives.
- Many companies use dynamic analyses, especially for hard-to-test bugs (concurrency).

Components of a Dynamic Analysis

- **Property of interest**
  - *What are you trying to learn about? Why?*
- **Information related to property of interest**
  - *How are you learning about that property?*
- **Mechanism for collecting that information from a program execution**
  - *How are you instrumenting it?*
- **Test input data**
  - *What are you running the program on?*
- **Mechanism for learning about the property of interest from the information you collected**
  - *How do you get from the logs to the answer?*

How to Transform Source Code?

- Regular Expressions

```
s/(\w+\(.*\);)/int t=time(); $1 print(time()-
t);/g
```

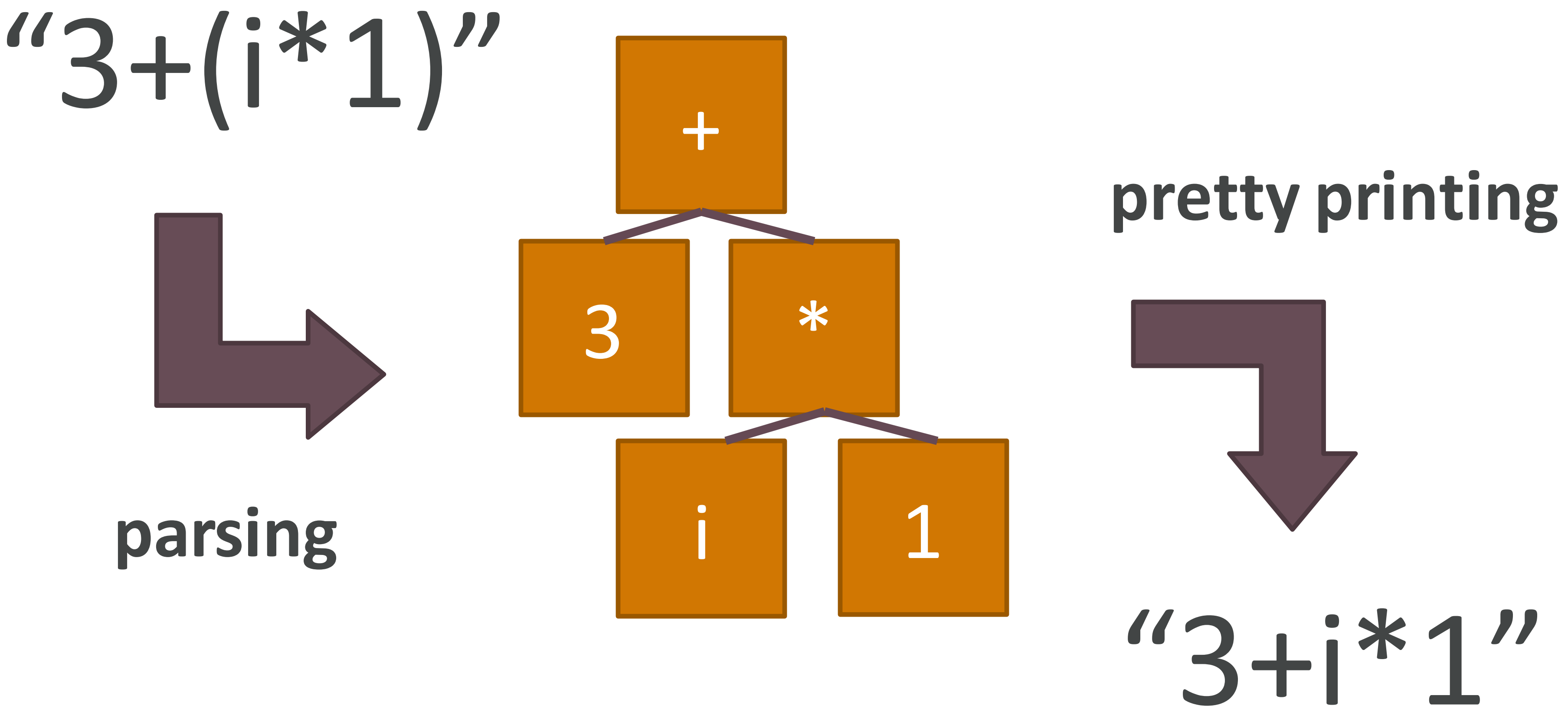
- Manually
- Other?

- Benefits?
- Drawbacks?

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

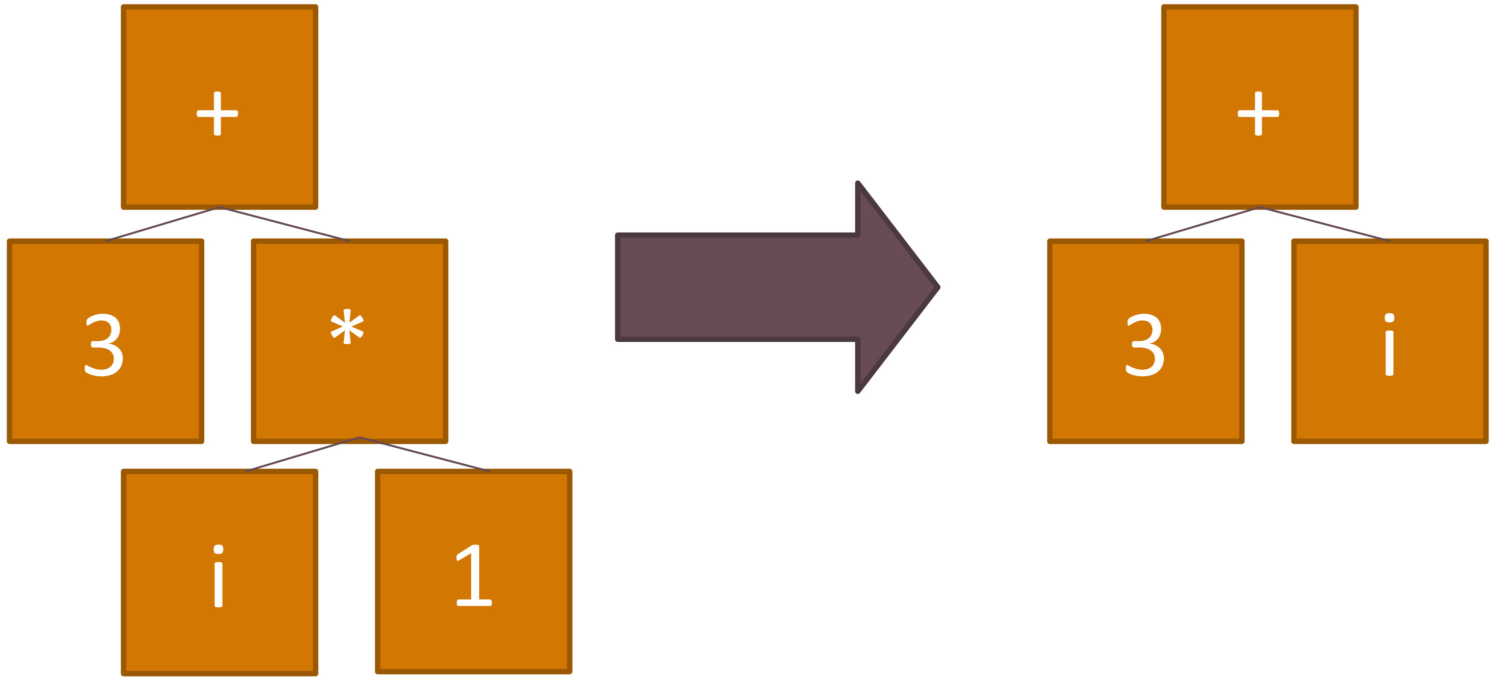
Parsing and Pretty Printing

- **Parsing** turns program text into an intermediate representation (abstract syntax tree or control flow graph). **Pretty printing** does the reverse.



AST Rewriting

- Parsing is a standard technology (CSxxxx Compilers)
  - Pretty printers are often written separately
  - Visitors, pattern matchers, etc., exist
  - You will get a chance to try rewriting ASTs in HW3



Binary or Byte Code Rewriting

- It is also possible to rewrite a compiled binary, object file or class file
- Java Byte Code is the Java VM input (.class)
  - Stack machine
  - Load, push, pop values from variables to stack
  - Similar to x86 assembly (but much nicer!)
- Java AST vs. Java Byte Code
  - You can transform back and forth (lose comments)

Byte Code Example

- Method with a single int parameter

```
my.Demo.foo( 1 ) becomes:

ALOAD 0
ILOAD 1
ICONST 1
IADD
INVOKEVIRTUAL “my/Demo” “foo” “(I)Ljava/lang/Integer;”
ARETURN
```

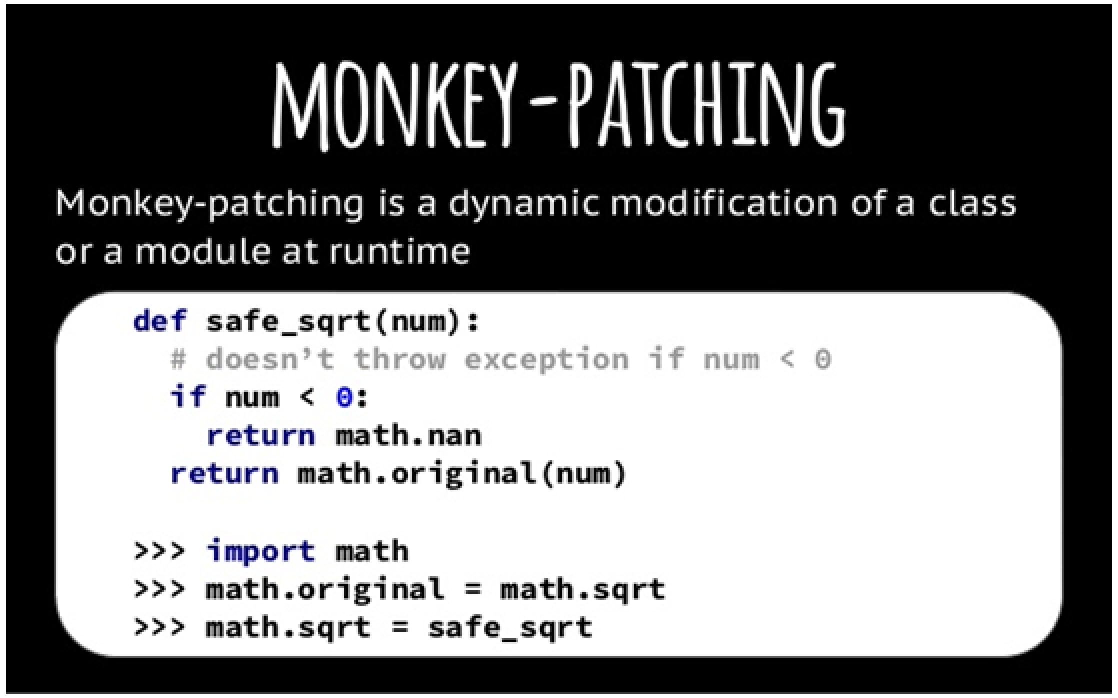
STOP TRYING TO MAKE THE ASK TOOLBAR HAPPEN
IT'S NOT GOING TO HAPPEN
memegenerator.net
U will LOL when u visit us! - LOLzHumor.com!

JVM Specification

- https://docs.oracle.com/javase/specs/
- You can see the byte code of Java classes with **javap** or the **ASM** Eclipse plugin
- Many analysis and rewrite frameworks. Ex:
  - Apache Commons **Byte Code Engineering Library**
  - https://commons.apache.org/proper/commons-bcel/
  - “is intended to give users a convenient way to analyze, create, and manipulate (binary) Java class files (those ending with .class). Classes are represented by objects which contain all the symbolic information of the given class: methods, fields and byte code instructions …”

Other Approaches

- Virtual machines and emulators
  - Valgrind, IDA Pro, GDB, etc.
    - Selectively rewrite running code or add special instrumentation (e.g., software breakpoints in a debugger)
- Metaprogramming
  - “Monkey Patching” in Python
- Generic Instrumentation Tools
  - Aspect-Oriented Programming



MONKEY-PATCHING

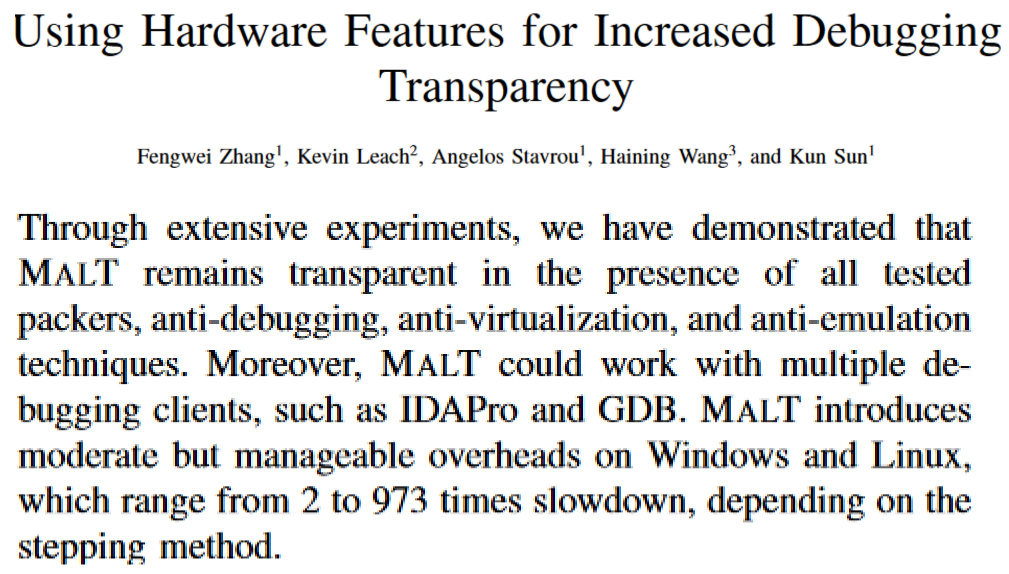
Monkey-patching is a dynamic modification of a class or a module at runtime

```
def safe_sqrt(num):
  # doesn’t throw exception if num < 0
  if num < 0:
    return math.nan
  return math.original(num)

>>> import math
>>> math.original = math.sqrt
>>> math.sqrt = safe_sqrt
```

Costs and Limitations of Analysis

- Performance overhead for recording
  - Acceptable for use in testing?
  - Acceptable for use in production?
- Computational effort for analysis
- Transparency limitations of instrumentation
  - “Heisenbugs” - a jargon (later in 5 slides)
- Accuracy
  - False positives?
  - False negatives?



Using Hardware Features for Increased Debugging Transparency

Fengwei Zhang[1], Kevin Leach[2], Angelos Stavrou[1], Haining Wang[3], and Kun Sun[1]

Through extensive experiments, we have demonstrated that MALT remains transparent in the presence of all tested packers, anti-debugging, anti-virtualization, and anti-emulation techniques. Moreover, MALT could work with multiple debugging clients, such as IDAPro and GDB. MALT introduces moderate but manageable overheads on Windows and Linux, which range from 2 to 973 times slowdown, depending on the stepping method.

Soundness vs. Completeness

Are there defects in a program? Positive: "there is a bug, in fact"; negative: "there is no bug, in fact"

- **Sound** Analyses
  - Report all defects → no false negatives
  - Typically overapproximate possible bad behavior
  - Are "conservative" with respect to safety: when in doubt, say it is unsafe

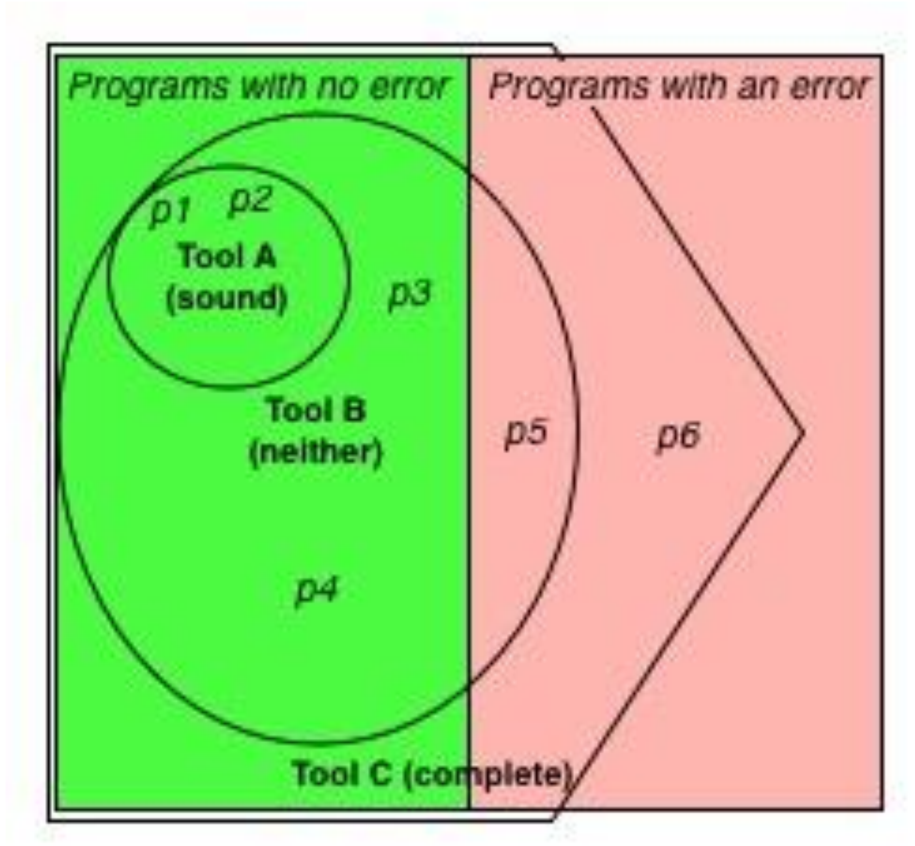
False negative: "there is a bug, but I say there is no bug"

- **Complete** Analyses
  - Every reported defect is an actual defect → no false positives
  - Typically underapproximate possible bad behavior

False positive: "there is no bug, but I say there is a bug"

False Positives, False Negatives

- "You can trust me when I say your radiation dosing software is safe."
  - Sound Analysis A says P1 is safe → P1 is actually safe
    - But P3 may be safe and A may think it unsafe!
  - If P1 is actually safe → Complete Analysis *C* says P1 is safe
    - But *C* may say P5 is safe but P5 is actually unsafe!



Bad News

- Every interesting analysis is either unsound or incomplete or both.

Bonus: check "Rice Theorem"

Input Dependent

- Dynamic analyses are very input dependent
- This is good if you have many tests
  - Whole-system tests are often the best
  - Per-class unit tests are not as indicative
- Are those tests indicative of normal use?
  - Is that what you want?
- Are those tests specific inputs that replicate known defect scenarios?
  - (e.g., memory leaks or race conditions)

Heisenbuggy Behavior

- Instrumentation and monitoring can change the behavior of a program
  - Through slowdown, memory overhead, etc.
- Consideration 1: Can/should you deploy it live?
- Consideration 2: Will the monitoring meaningfully change the program behavior with respect to the property you care about?

Dynamic Analysis Examples

- Digital Equipment Corporation's **Eraser**
- Netflix's **Chaos Monkey**
- Microsoft's **CHESS**
- Microsoft's **Driver Verifier**

Eraser: Is There A Race Condition?

```
// Thread #1
while (true) {
  lock(mutex);
  v := v + 1;
  unlock(mutex);
  y := y + 1;
}
```

```
// Thread #2
while (true) {
  lock(mutex);
  v := v + 1;

unlock(mutex);
  y := y + 1;
}
```

Eraser: Is There A Race Condition?

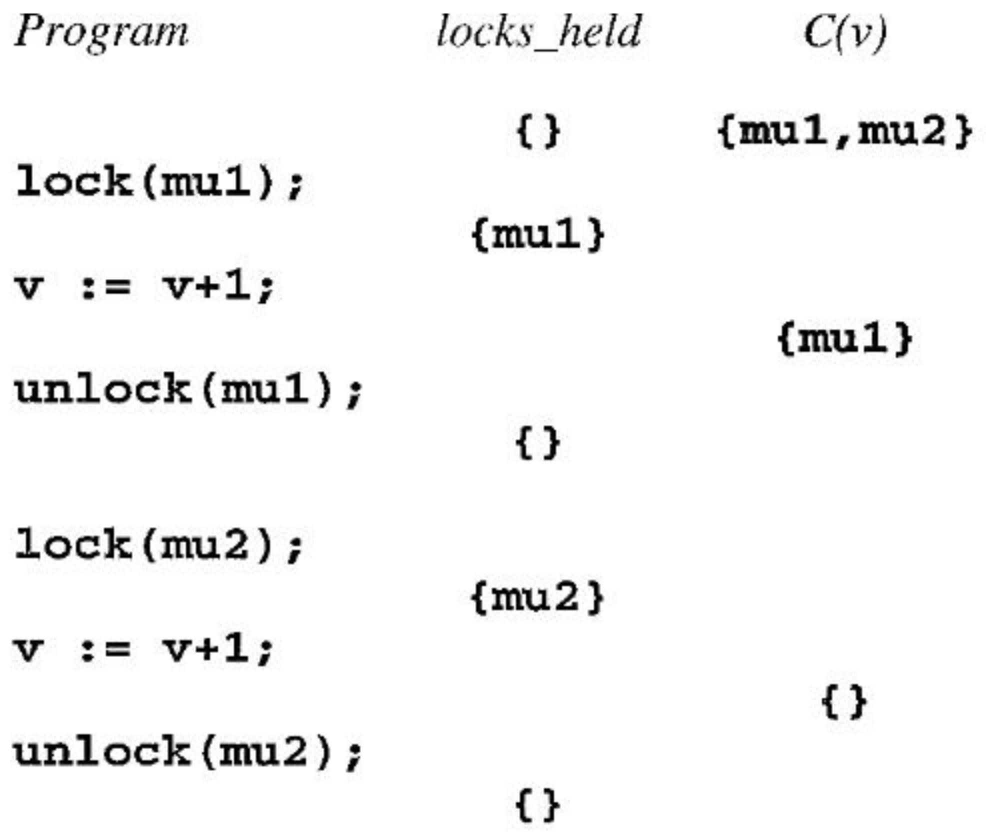
```
// Thread #1
while (true) {
  lock(mu1);
  v := v + 1;
  unlock(mu1);
  y := y + 1;
  lock(mu2);
  v := v + 1;
  unlock(mu2); }
```

```
// Thread #2
while (true) {
  lock(mu1);
  v := v + 1;
  unlock(mu1);
  y := y + 1;
  lock(mu2);
  v := v + 1;
  unlock(mu2); }
```

Eraser Insight: Lockset Algorithm

- Each shared variable must be guarded by a lock for the whole computation. If not, you have the possibility of a race condition.
  - Start with “all locks could possibly protect *v*”
  - If you observe that lock *i* is not held when you access *v*, remove lock *i* from the set of locks that could possibly guard v
  - If the set of locks that could possibly guard *v* is ever empty, then no lock can guard *v*, so you can have a race condition (even if you didn't actually see the race this time!)

Eraser Lockset Example



```
Program              locks_held        C(v)

                        {}          {mu1,mu2}
lock(mu1);
                      {mu1}
v := v+1;
                                      {mu1}
unlock(mu1);
                        {}

lock(mu2);
                      {mu2}
v := v+1;
                                       {}
unlock(mu2);
                        {}
```

Fig. 3. If a shared variable is sometimes protected by **mu1** and sometimes by lock **mu2**, then no lock protects it for the whole computation. The figure shows the progressive refinement of the set of candidate locks $C(v)$ for v. When $C(v)$ becomes empty, the Lockset algorithm has detected that no lock protects v.

[Savage, Burrows, Nelson, Sobalvarro, Anderson. *Eraser: A Dynamic Data Race Detector for Multithreaded Programs.* ACM Trans. Comp. Sys. 15(4) 1997.]

Eraser: Does It Work?

- “Applications typically slow down by a factor of 10 to 30 while using Eraser.”
- “It can produce false alarms.”
- Applied to web server (mhttpd), web search indexing engine (AltaVista), cache server, and distributed filesystem
- One example: cache server is 30KLOC C++, 10 threads, 26 locks
  - “serious data race” in fingerprint computation

Chaos Monkey

- **Chaos Monkey** was invented in 2011 by Netflix to test the resilience of its IT infrastructure
- “Imagine a monkey entering a "data center", these "farms" of servers that host all the critical functions of our online activities. The monkey randomly rips cables, destroys devices and returns everything that passes by the hand. The challenge for IT managers is to design the information system they are responsible for so that it can work despite these monkeys, which no one ever knows when they arrive and what they will destroy.” – Antonio Martinez, *Chaos Monkey*

Chaos Monkey

- "We have created Chaos Monkey, a program that randomly chooses a server and disables it during its usual hours of activity. Some will find that crazy, but we could not depend on the random occurrence of an event to test our behavior in the face of the very consequences of this event. Knowing that this would happen frequently has created a strong alignment among engineers to build redundancy and process automation to survive such incidents, without impacting the millions of Netflix users. Chaos Monkey is one of our most effective tools to improve the quality of our services."
    - Greg Orzell, *Netflix Chaos Monkey Upgraded*

Simian Army Examples

- Latency Monkey induces artificial delays in our RESTful client-server communication layer to simulate service degradation
- Conformity Monkey finds instances that don't adhere to best-practices and shuts them down (e.g., instances that don't belong to an auto-scaling group
- Doctor Monkey taps into health checks that run on each instance as well as monitors other external signs of health (e.g. CPU load) to detect unhealthy instances and remove them
- 10–18 Monkey (short for Localization-Internationalization) detects configuration and run time problems in instances serving customers in multiple geographic regions, using different languages and character sets

Chaos Monkey
Randomly disables production instances
Janitor Monkey
Identifies and disposes unused resources
Chaos Kong
Drops a full AWS Region
Conformity Monkey
Shuts down instances not adhering to best-practices
NETFLIX
SIMIAN ARMY
Chaos Gorilla
Outage of entire Amazon Availability Zone
Security Monkey
Finds security violations and vulnerability
@geosley
Doctor Monkey
Taps into health checks and fixes unhealthy resources
Latency Monkey
Simulate degradation or outages in a network

CHESS

- "**CHESS** is a tool for finding and reproducing Heisenbugs in concurrent programs. CHESS repeatedly runs a concurrent test ensuring that every run takes a different interleaving. If an interleaving results in an error, CHESS can reproduce the interleaving for improved debugging. CHESS is available for both managed and native programs."

CHESS Intuition

- Recall the coupling effect hypothesis:
  - A test suite that detect simple faults will likely also detect complex faults
- Suppose you have some AVL tree balancing or insertion code with a bug
  - There is a size-100 tree that shows off the bug
  - Is there also a small tree that shows it off?

CHESS Intuition

- Suppose you have a concurrency bug that you can show off with a complicated sequence of sixteen thread interleavings and preemptions
  - Is there also a sequence of one or two preemptions to show off the same bug? Likely!

CHESS: Does It Work?

- “a lightweight and effective technique for dynamically detecting data races in kernel modules … oblivious to the synchronization protocols (such as locking disciplines) … This is particularly important for low-level kernel code … To *reduce* the runtime overhead … **randomly samples** a *small percentage* of memory accesses as *candidates* for data-race detection … uses breakpoint facilities already supported by many hardware architectures to achieve *negligible runtime overheads* … the Windows 7 kernel and have found 25 confirmed erroneous data races of which 12 have already been fixed.”

[Erickson, Musuvathi, Burckhardt, Olynyk. Effective Data-Race Detection for the Kernel. OSDI 2010.]

Driver Verifier Overview

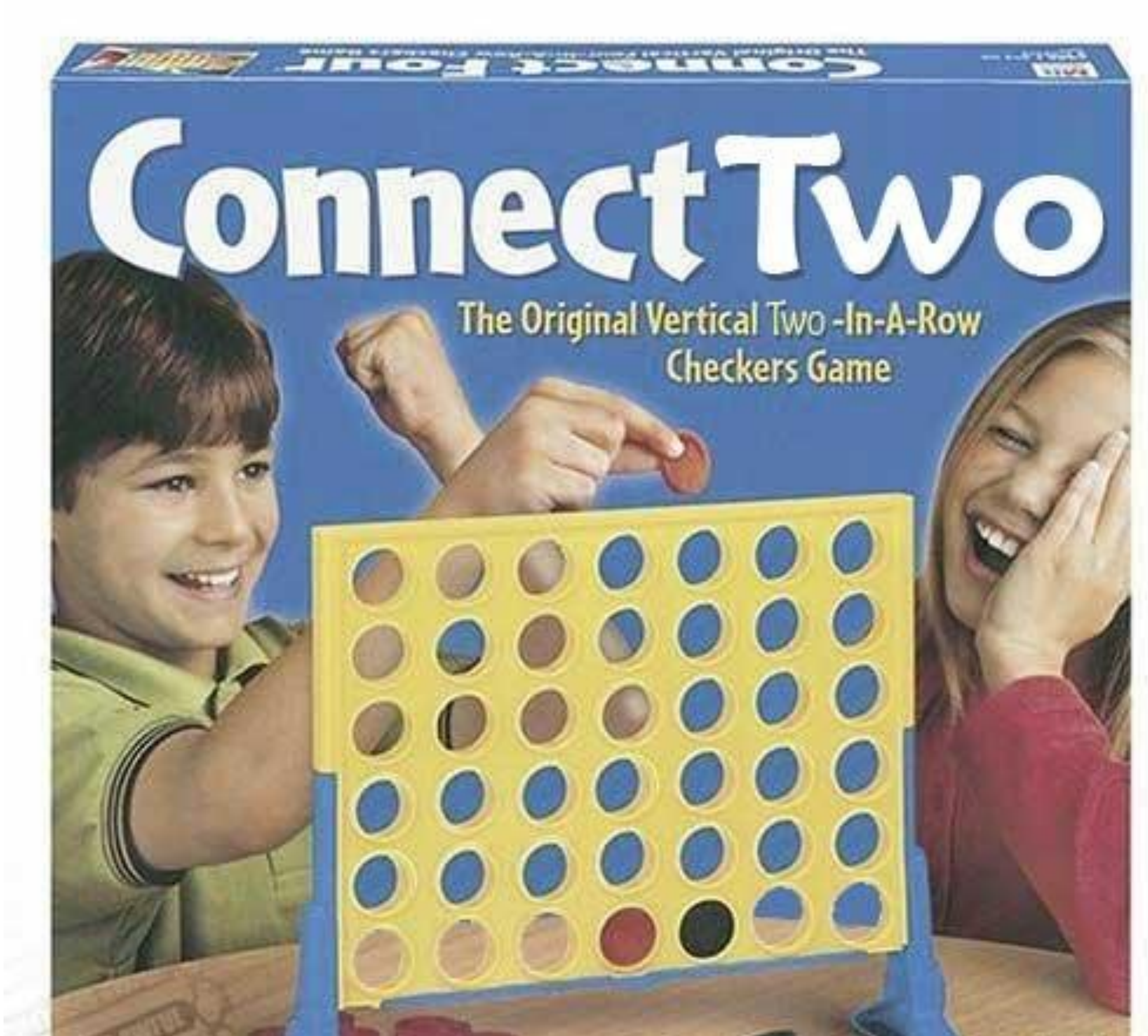
- “**Driver Verifier** is a tool included in Microsoft Windows that replaces the default operating system subroutines with ones that are specifically developed to catch device driver bugs. Once enabled, it monitors and stresses drivers to detect illegal function calls or actions that may be causing system corruption.”
  - Simulates low memory, I/O problems, IRQL problems, DMA checks, I/O Request Packet problems, power management, etc.

Driver Verifier: Did It Work?

- "The Driver Verifier tool that is included in every version of Windows since Windows 2000"
  - https://support.microsoft.com/en-us/help/244617/using-driver-verifier-to-identify-issues-with-windows-drivers-for-adva

Bonus question: Driver Verifier is a dynamic analysis tool from Microsoft. What is a static analysis tool from Microsoft that detects malfunctioning drivers?

Pair Programming and Skill-Based Interviews



The Story So Far …

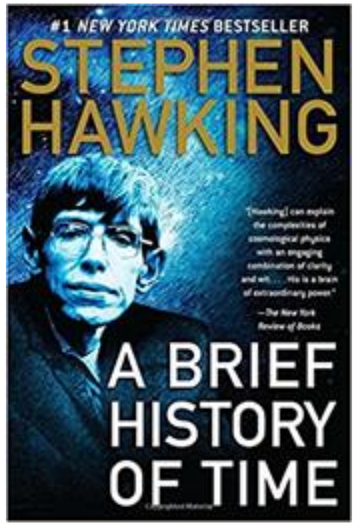
- We want to deliver and support a quality software product
  - We understand the stakeholder requirements
  - We understand process and design
  - We understand quality assurance
  - We somewhat understand humans
- How should we make process and design designs the first time …
- … in light of how humans 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

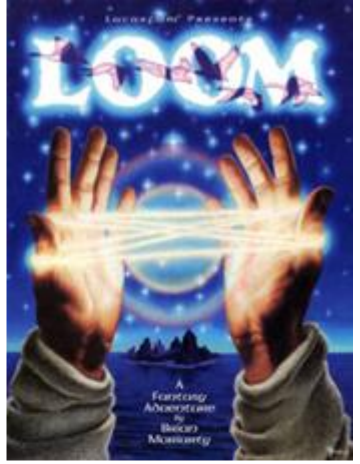
- Structured Programming (1950-1960+)
  - Structured Programming Theorem (1966)
- Object-oriented Programming (1970-1980+)
  - Dominant in 1990+
- Aspect-oriented Programming (1997+)
- Iterative & Incremental Development (1960+)
- Agile Development (2001+)
- Scrum (1986+, 2001+)



The What, 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 (Aspect-oriented)*: structure source code by separating cross-cutting concerns to increase modularity
- *IID (Iterative and incremental)*: 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)

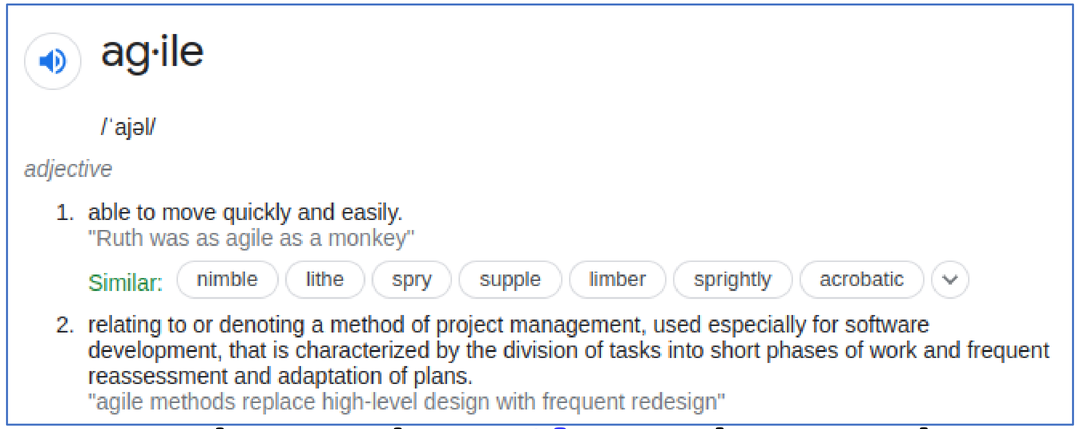


- With respect to software source code
- **Abstraction** (e.g., inheritance, polymorphism) allows the same code to be applied to different data
  - This saves development and QA effort
- **Modularity** (e.g., interfaces) permits a separation of concerns, allowing code both sides of the interface to be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


ag·ile

/ˈajəl/

adjective

1. able to move quickly and easily.
   "Ruth was as agile as a monkey"

   Similar: nimble lithe spry supple limber sprightly acrobatic

2. relating to or denoting a method of project management, used especially for software development, that is characterized by the division of tasks into short phases of work and frequent reassessment and adaptation of plans.
   "agile methods replace high-level design with frequent redesign"

- 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.*]

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

Pair Programming

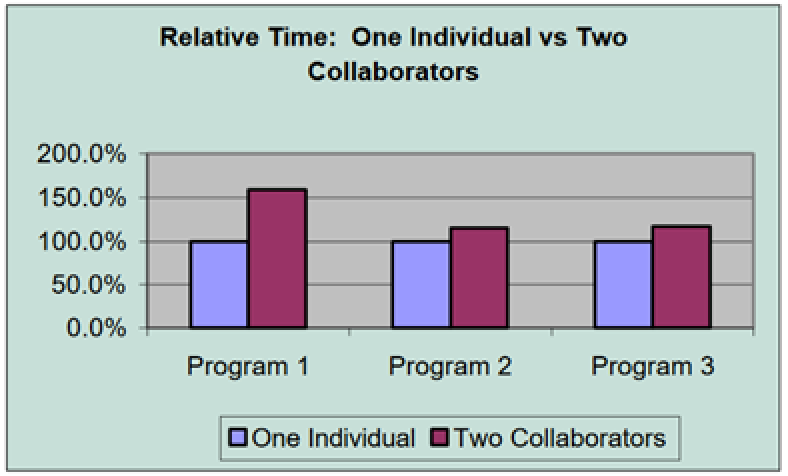
- **Pair programming** refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test.
- The pair is made up of a **driver**, who actively types at the computer or records a design; and a **navigator** (or **observer**), who watches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

One Thousand Words

One Thousand Words

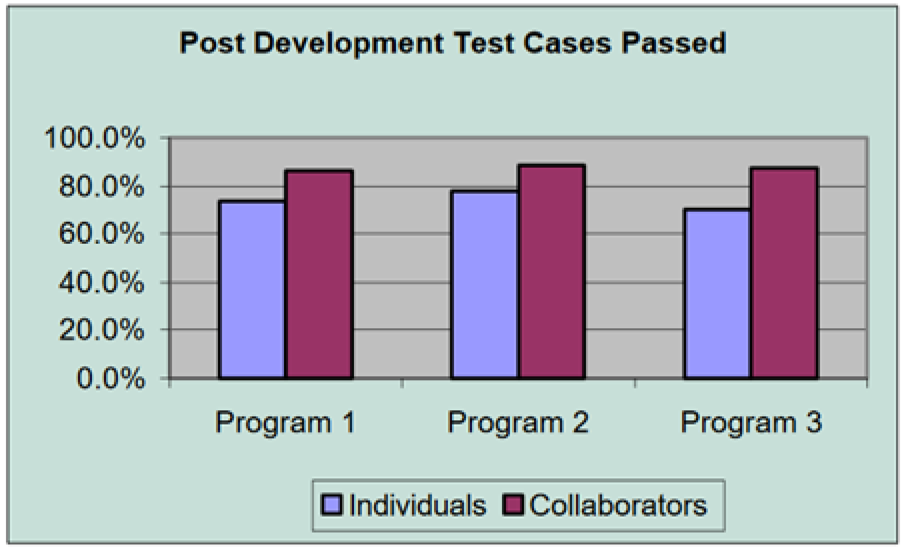
Pair Programming and Programmers

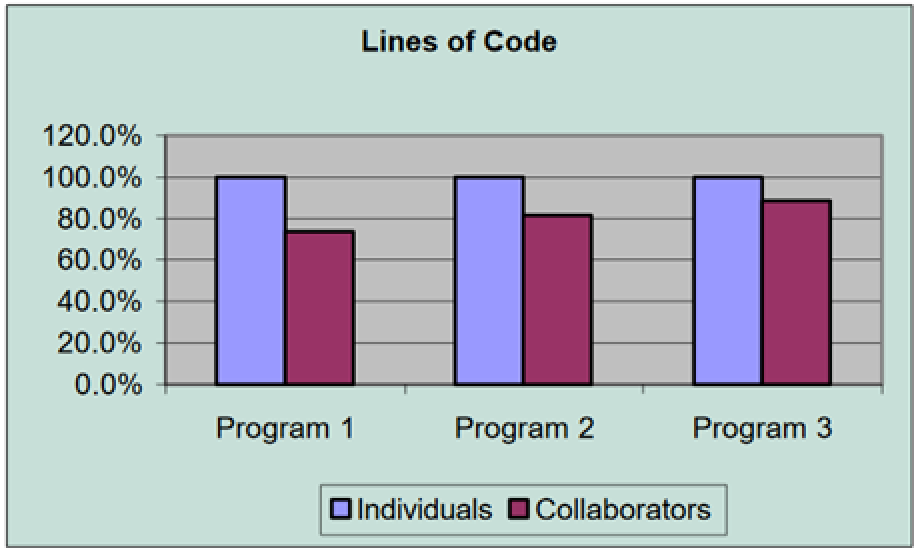
- Surveys of professional programmers
  - 90+% “enjoyed collaborative programming more than solo programming”
  - 95% were “more confident in their solutions” when they pair programmed
- Increases development cost by 15% to 100% - "absolute time taken"



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

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, in the previous slide ***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!

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, $p < 0.01$)

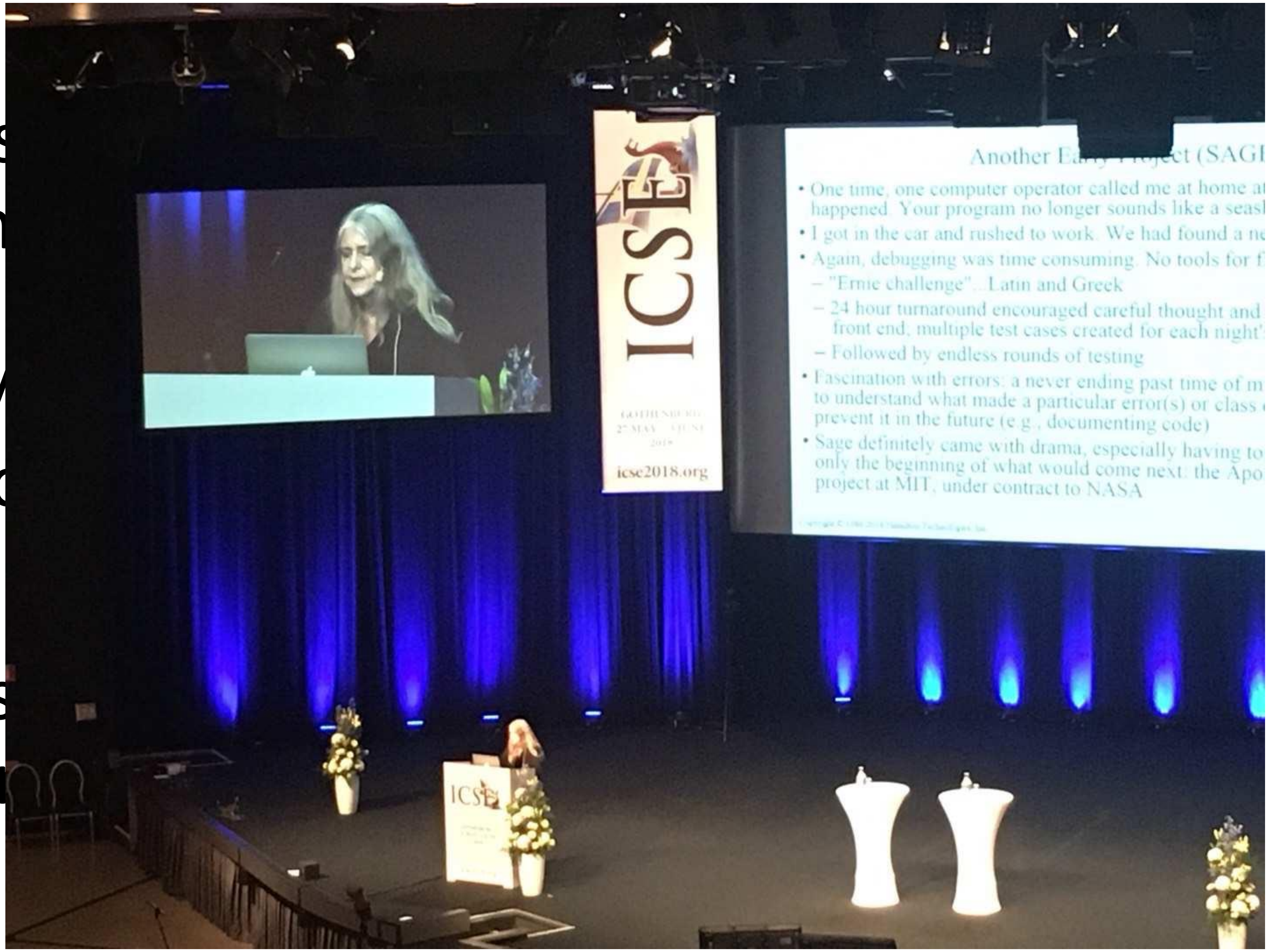
Trivia:

- This computer scientist, system engineer, and business owner, was director of the Software Engineering Division of the MIT Instrumentation Lab, which developed on-board flight software for NASA's Apollo program. This computer scientist is one of the people credited with coning the term “Software Engineering”.
- In Apollo 11 Mission, this computer scientist’s on-board flight software averted an abort of the landing on the moon.

Trivia: **Margaret Hamilton**

• s
h
n
/
e

• s
ar



Another Ea ect (SAGI

- One time, one computer operator called me at home a happened. Your program no longer sounds like a seas
- I got in the car and rushed to work. We had found a n
- Again, debugging was time consuming. No tools for f
  - "Ernie challenge"...Latin and Greek
  - 24 hour turnaround encouraged careful thought and front end; multiple test cases created for each night'
  - Followed by endless rounds of testing
- Fascination with errors: a never ending past time of m to understand what made a particular error(s) or class prevent it in the future (e.g., documenting code)
- Sage definitely came with drama, especially having to only the beginning of what would come next: the Apo project at MIT, under contract to NASA

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

Minnesota Twin Registry

What's Special About Twins to Science?

By studying identical and fraternal twins and their families, we can estimate how genes and environment interact to influence character, strengths, vulnerabilities, and values. To find out more about twin research, why twins are so special to science, and the kind of work that your contribution makes possible - click here.

Identical or Fraternal?

Have you always wanted to know? Find out with our online zygosity calculator, or a DNA test.

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: 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 several months, 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.)"

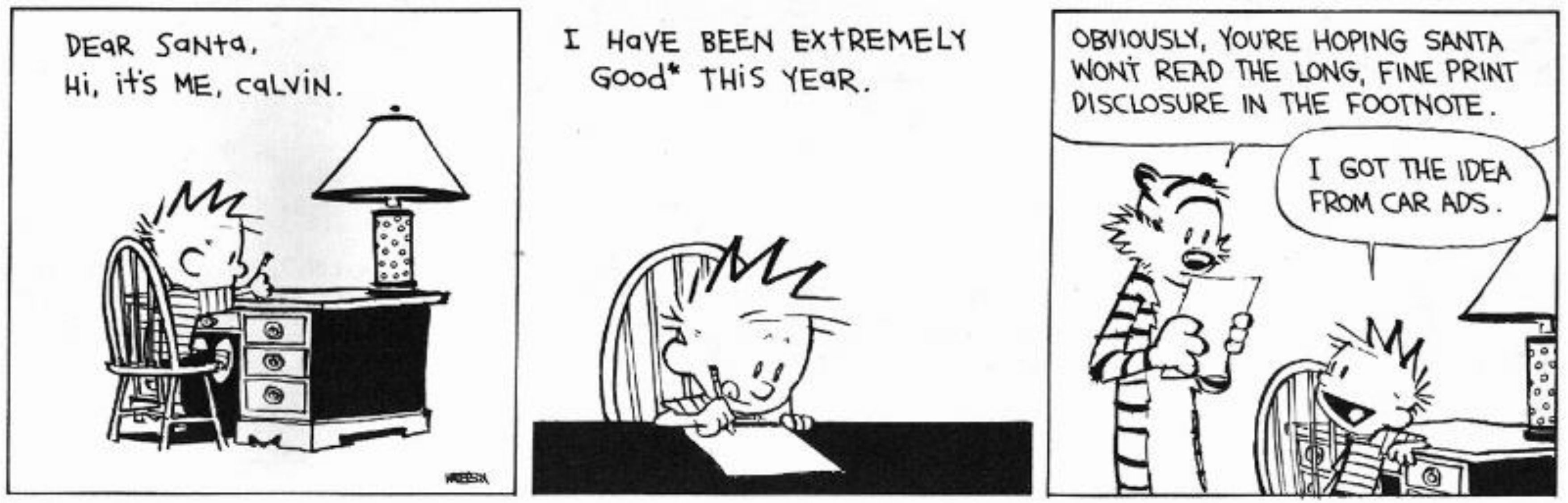
I THINK WE SHOULD BUILD A BLOCKCHAIN
UH-OH
DOES HE UNDERSTAND WHAT HE SAID OR IS IT SOMETHING HE SAW IN A TRADE MAGAZINE AD?
WHAT COLOR DO YOU WANT THAT BLOCKCHAIN?
I THINK MAUVE HAS THE MOST RAM.

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"


DEAR SANTA,
HI, IT'S ME, CALVIN.
I HAVE BEEN EXTREMELY GOOD* THIS YEAR.
OBVIOUSLY, YOU'RE HOPING SANTA WON'T READ THE LONG, FINE PRINT DISCLOSURE IN THE FOOTNOTE.
I GOT THE IDEA FROM CAR ADS.

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?

DOGBERT THE GENERIC MANAGER
WE NEED MORE PEOPLE ON THE PROJECT.
FIGURE IT OUT. WORK SMARTER NOT HARDER. MAKE A PLAN. MOVE SOME THINGS AROUND. ADJUST PRIORITIES. JUST GET IT DONE. GIVE ME A STATUS REPORT.
THAT DID NOTHING BUT MAKE ME HATE YOU.
I CAN REPLACE YOU WITH SOMEONE WHO WILL PRETEND TO BE INSPIRED.
Dilbert.com DilbertCartoonist@gmail.com
3-8-10 © 2010 Scott Adams, Inc./Dist. by UFS, Inc.

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

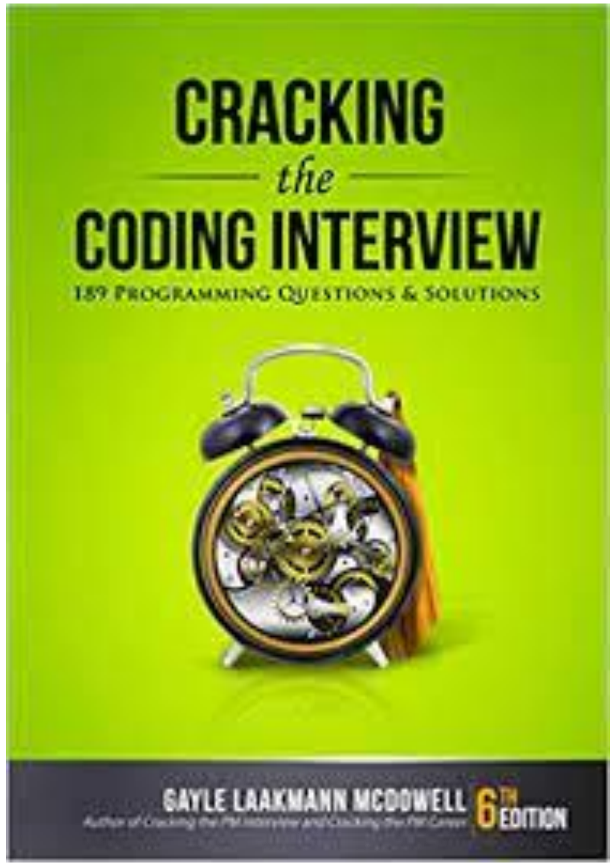
Do Not Forget

- 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


CRACKING the CODING INTERVIEW
189 PROGRAMMING QUESTIONS & SOLUTIONS
GAYLE LAAKMANN MCDOWELL
6TH EDITION

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!

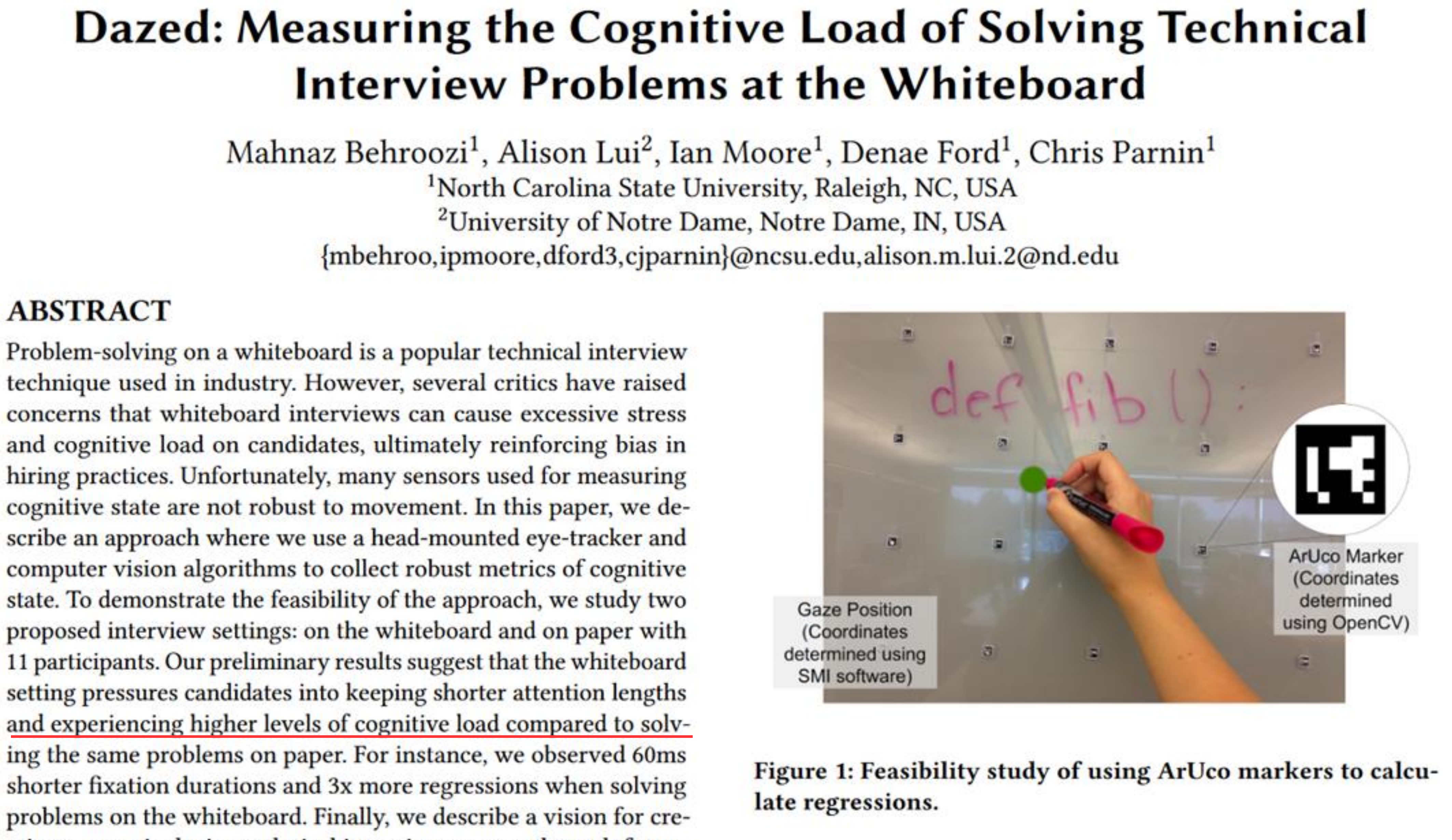
Situation, Action, Result

- 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

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[1], Alison Lui[2], Ian Moore[1], Denae Ford[1], Chris Parnin[1]
[1]North Carolina State University, Raleigh, NC, USA
[2]University of Notre Dame, Notre Dame, IN, USA
{mbehroo,ipmoore,dford3,cjparnin}@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 cre-

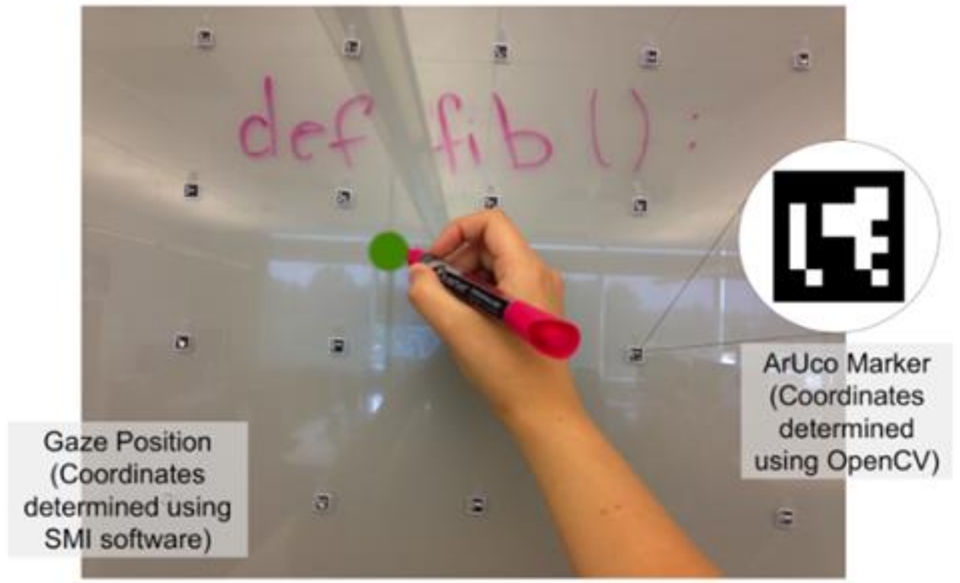
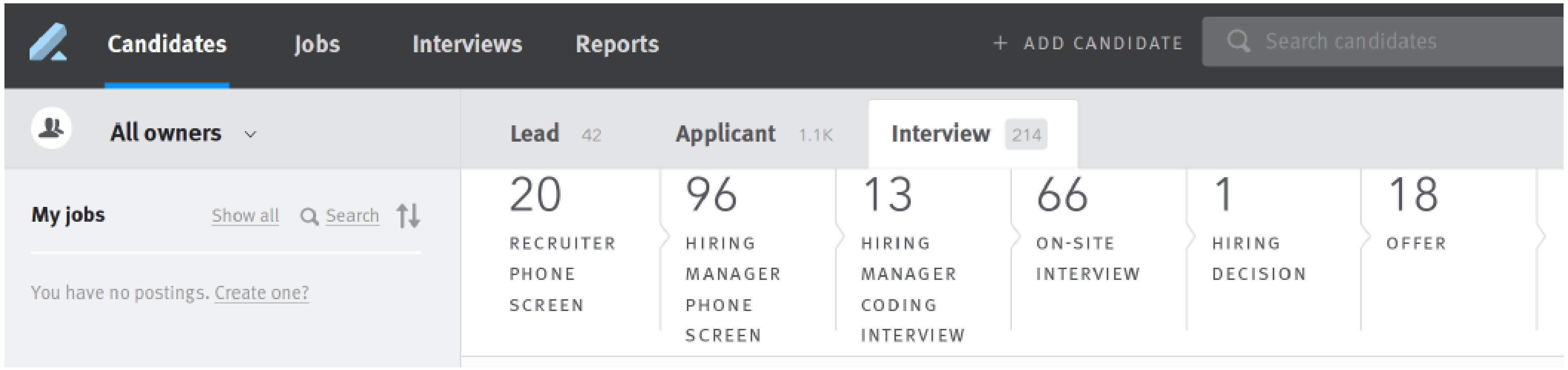

def fib():
Gaze Position (Coordinates determined using SMI software)
ArUco Marker (Coordinates determined using OpenCV)

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

Suggestion

- Remember this “from the other side”



You have no postings. Create one?

Lead 42 | Applicant 1.1K | Interview 214

20	96	13	66	1	18
RECRUITER PHONE SCREEN	HIRING MANAGER PHONE SCREEN	HIRING MANAGER CODING INTERVIEW	ON-SITE INTERVIEW	HIRING DECISION	OFFER