STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

SSW-555: Agile Methods for Software Development

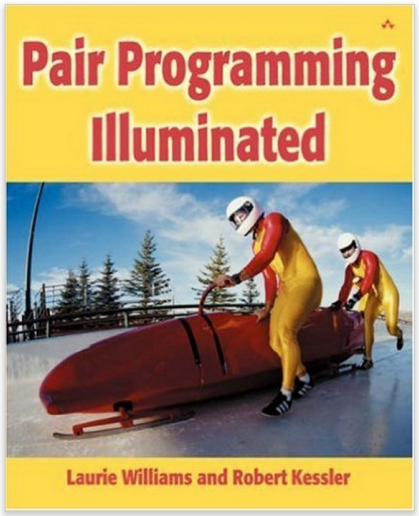
Continuous Integration & Pair Programming
Week 5

Today's Topics

- Overview of Continuous Integration
  - Motivation
  - Practices
  - Benefits
- Overview of Pair Programming
  - 7 Myths
  - 7 Synergistic Practices
- In class practice of PP

Acknowledgements

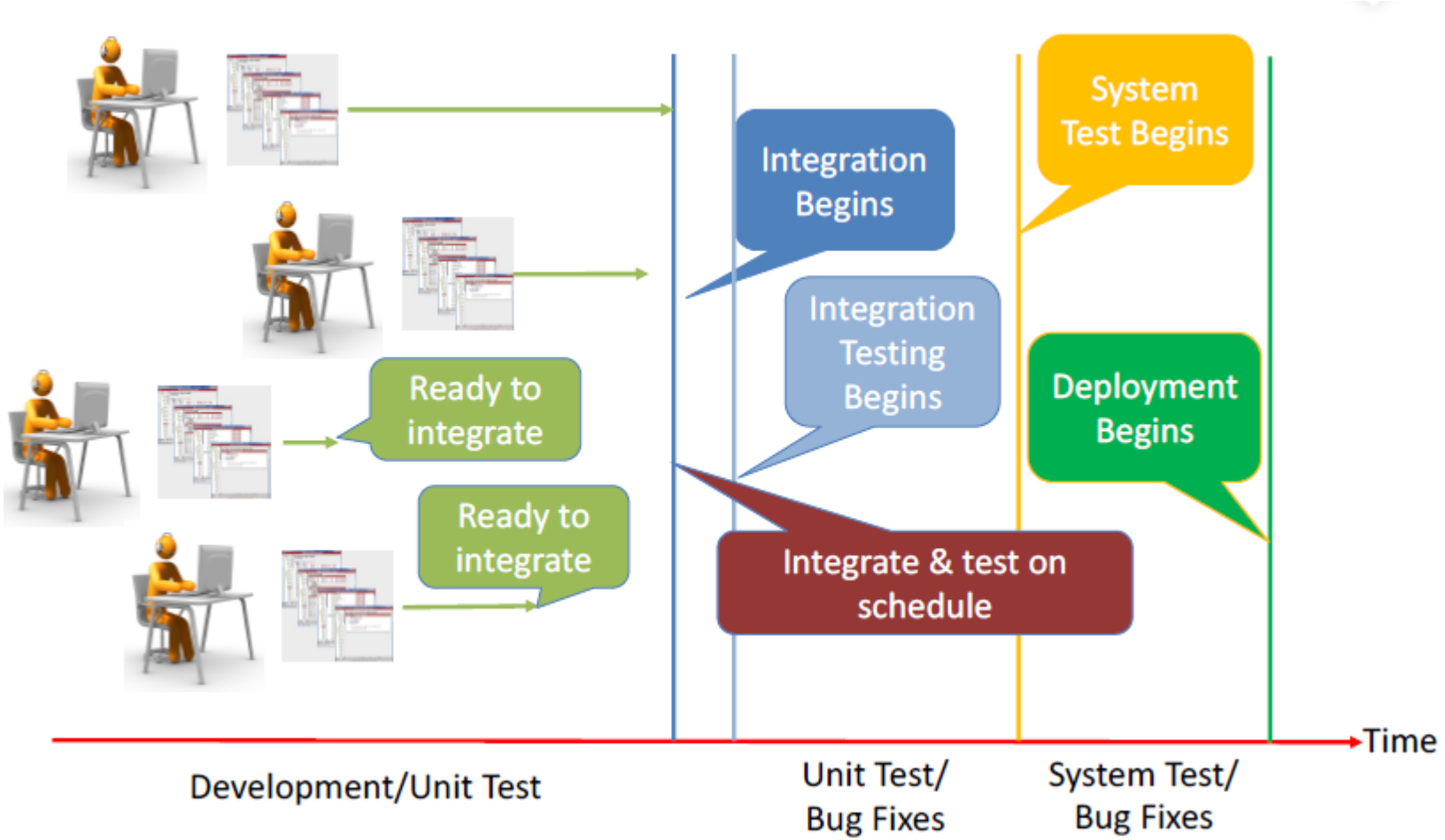
- https://www.martinfowler.com/articles/continuousIntegration.html
- *Pair Programming Illuminated* by Laurie Williams and Robert, Kessler, Addison-Wesley 2003.



Deferred Integration

- A project had been in development for a couple of years
  - Parts have been built and tested in separate
- Developers start to integrate separate parts into a deliverable product to their customers.
  - It has been several months…

Deferred Integration Timeline (e.g. Waterfall)

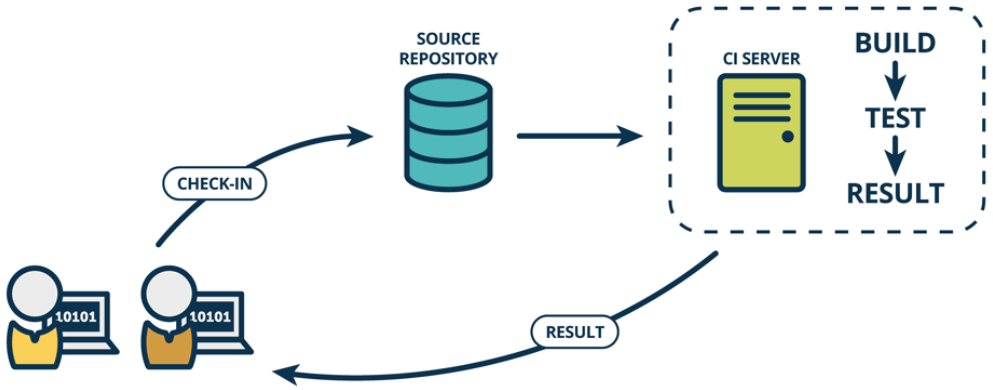


Problems with Deferred Integration

- Nobody knows how long it would take to finish integrating
  - You are at a complete blind spot
  - Bugs destroy confidence and mess up schedules and reputations
  - Bugs are cumulative. The more you find and the longer you wait, the harder to remove (broken window syndrome)

Continuous Integration

- Everyone on the team integrating frequently, usually daily, against a controlled source code repository.
  - A non-event, no blind spot: you know where you are, what works, what doesn't
  - Any integration errors are found rapidly and can be fixed rapidly
  - Users get features more rapidly---more rapid feedback on features

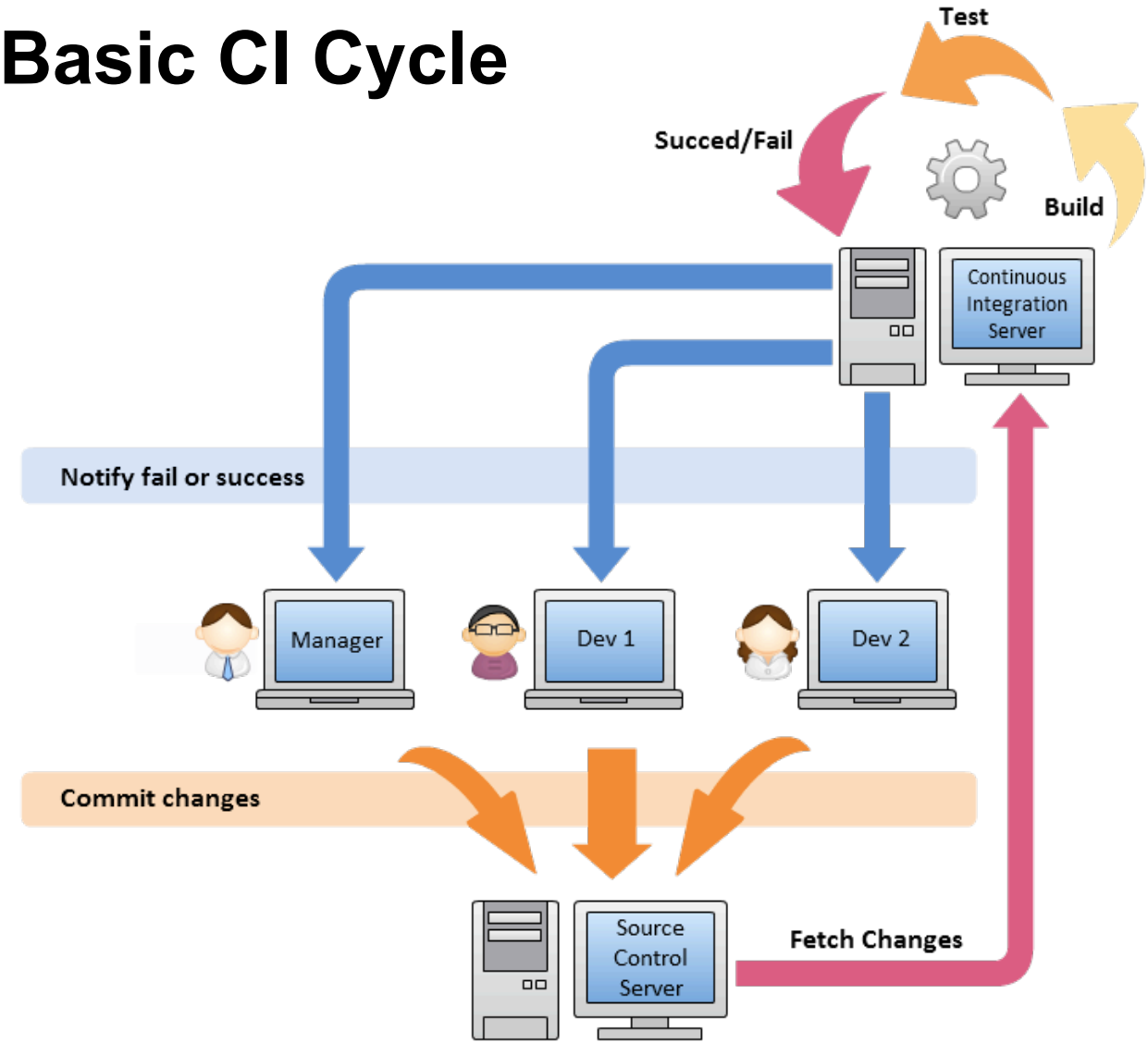


https://github.com/snap-ci/snap-ci-blog-content/blob/master/posts/2016-03-22-fail-fast.md

Amy Builds a Feature with CI

- Amy checks out a working copy of the mainline source code to her local machine
- Amy complete her task on the working copy: altering the production code and adding/changing automated tests, compiles and build executable, run and passes tests.
- Amy commit her changes to the mainline source code
  - Amy find Bob commit changes after her check out and before she commit: Bob's changes clashes with Amy's change
  - Amy is responsible for fixing this and repeat until she can build a working copy

Basic CI Cycle



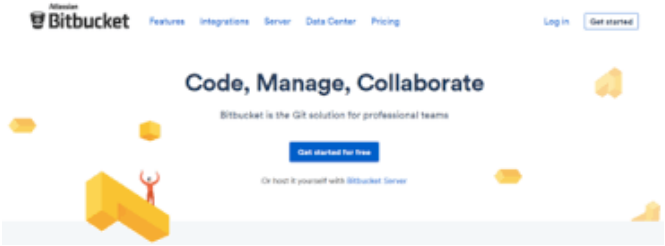
https://www.code-maze.com/what-is-continuous-integration/

Practices of Continuous Integration

1. Maintain a single source repository
2. Automate the build
3. Make your build self-testing
4. Everyone commits to the mainline every day
5. Fix broken builds immediately
6. Keep the build fast
7. Test in a clone of the production environment
8. Make it easy for anyone to get the latest executable

1. Maintain a single code repository

- Use source code management system to keep track of all the files and changes
- Have a mainline: a single branch (master branch) of the project currently under development
  - Put everything required for a build in the source control system: test scripts, property files, database schema, install scripts, and third party libraries
  - Pretty much everyone should work off this mainline most of the time: allow multiple branches to handle different streams of development, but don't overuse branches.



2. Automate build

- Turn sources into a running system
  - Compilation, moving files around, loading schemas into databases, etc.
- This process can be automated:
  - Unix: make
  - Java: Ant, Maven
  - .NET: Nant, MSBuild
- A big build takes time, a good build tool analyzes what needs to be changed as part of the build process.

3. Self-testing

- A good way to catch bugs more quickly and efficiently is to include automated tests in the build process
  - XP and TDD have popularized self-testing code
- Self-testing code: a suite of automated tests that can check a large part of the code base for bugs.
  - Kick off from a simple command and self-checking
- You can't count on tests to find everything: tests don't prove the absence of bugs
  - Imperfect tests, run frequently are better than perfect tests that are never written at all

4. Developers frequent commit to mainline

- Developers quickly find out if there's a conflict between two developers.
  - The key to fixing problems quickly is finding them quickly
  - Conflicts stay undetected for weeks can be hard to resolve.
- Self-testing helps developers to find bugs quickly
    - Developers use diff-debugging to find the bug
- General rule of thumb: every developer should commit to the repository every day
  - Break down their work into small chunks

5. Fix broken builds immediately

- The whole point of CI is that developers always developing on a known stable base.
  - When the mainline break, it is important that it gets fixed fast.
- Often the fastest way to fix the built is to revert the latest commit from the mainline
  - Taking the system back to the last-known good build
  - Debug a problem on a working copy

6. Keep build fast

- The whole point of CI is to provide rapid feedback
  - 1 hour built is unreasonable
  - 10 min build is achieved in most modern projects
- Every minute reduced off the build time is a minute saved for each developer every time they commit
  - The trick is to balance the needs of bug finding and speed so that good commit build is stable enough for other people to work on.

7. Test in production environment

- If you test in a different environment, what happens under test won't happen in production.
- Set up your test environment to be as exact a mimic of your production environment as possible
  - Same database software, same versions, same version of operation system, all libraries, IP addresses and ports, same hardware
  - Sometimes not practical, expensive
- Use virtualization to make it easy to put together test environments
  - E.g. simulate multiple machines in a network on a single machine

8. Make the latest executable available

- It's very hard to specify what you want in advance and be correct
  - It is much easier to see something that's not quite right and say how it needs to be changed.
- Anyone involved with a software project should be able to get the latest executable and be able to run it: for demonstrations, exploratory testing, or just see what new this week…

CI Benefits

- Reduced risk
- Detect and fix bugs more quickly and easily
  - Relatively little new, untested code at any given time
  - Developers fix bugs when code fresh in their minds
- Converge on a solution more quickly
- Fewer bugs associated with automated testing
- Enables continuous delivery to customers
  - Increases communication between developers and customers

CI Tools

Travis CI

- Open Source
- Integrated with GitHub
- Hosted on GitHub

Jenkins

- Open Source
- Extensible through plugins
- Supports many languages

TeamCity

- JetBrains: PyCharm, IntelliJ Idea, …
- Extensible through plugins
- Supports many languages

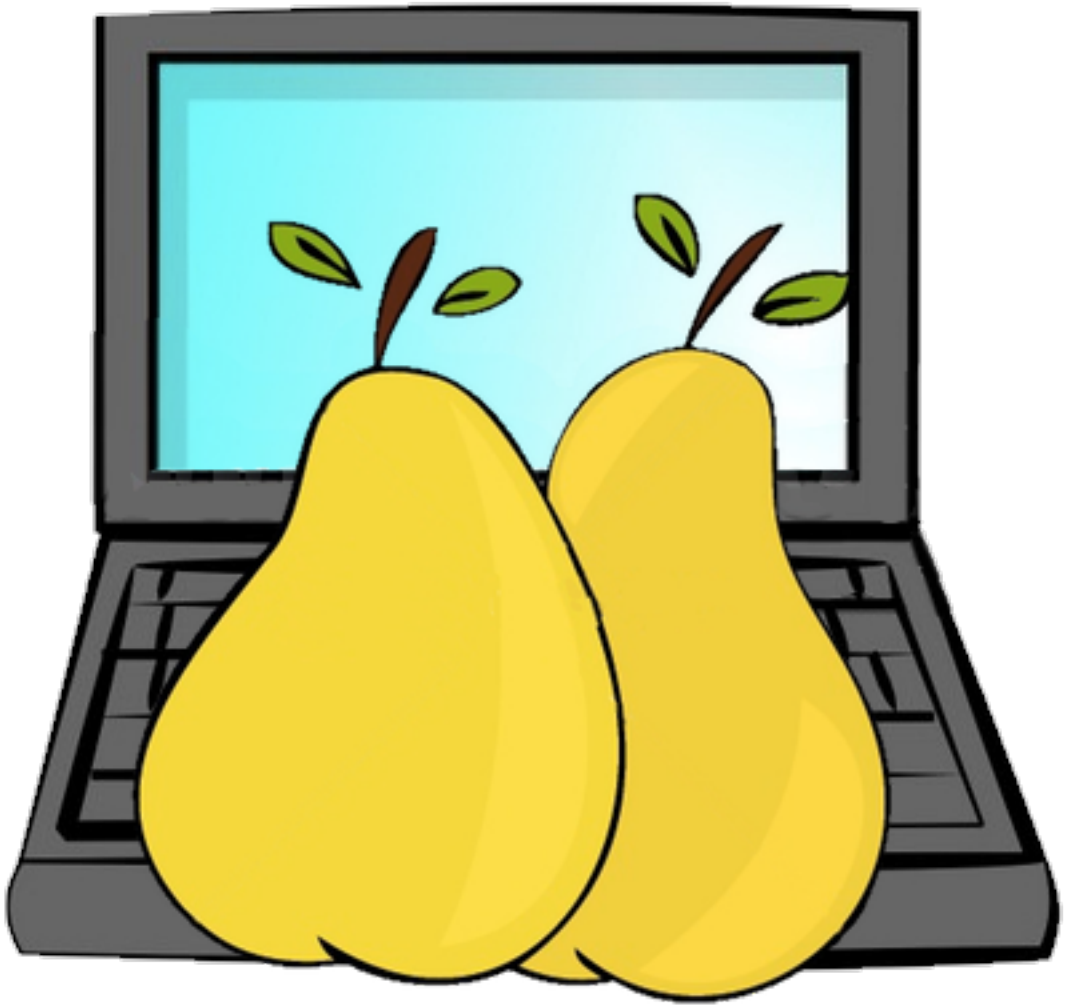
Bamboo

- Atlassian: Jira, Trello, BitBucket, …
- Proprietary solution
- Cloud based or hosted solution

https://www.code-maze.com/top-8-continuous-integration-tools/

Agile in Practice: Pair Programming

https://www.youtube.com/watch?v=ET3Q6zNK3Io



Pair Programming Overview

- 2 programmers sit in front of the same computer:
  - 1 programmer (the driver) types
  - 1 programmer (the navigator) watches, catches mistakes, suggests alternatives, designs tests
- The 2 programmers switch roles frequently
  - Every 15-20 minutes
- Works best if both are co-located but it can also work if not

Pair Programming Guidelines

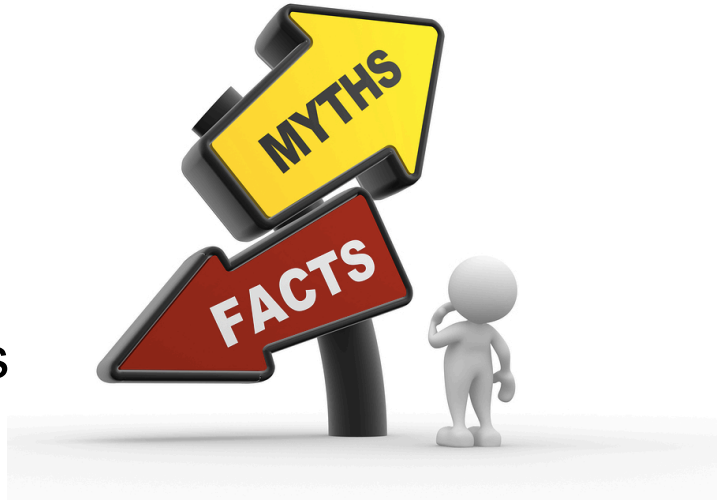
- Change roles often, every 15-20 minutes
- Work with someone at the same level of experience
- Take breaks
- Communicate:
  - 15 seconds without talking is a very long time
  - 30 seconds without talking is an eternity
  - Constant communication – explain what you're doing
- Listen to your partner and be a good listener

Research Results

- Pairs work almost twice as fast as individuals
- Pairs produce higher quality work than individuals
  - Higher quality leads to less time and effort in testing and fixing bugs
- Some pairings may not work effectively
  - Pairs with mismatched experience may not be effective

7 Pair Programming Myths

1. It will take twice as long
2. I'll never get to work alone
3. It only works with the right partner
4. It’s only good for training
5. I'll have to share credit for everything
6. The navigator finds only syntax mistakes
7. I won't be able to concentrate with my partner interrupting me all the time



Myth 1: It will take twice as long

- You've allocated twice as many people to do the same task…won't it take twice as long?
- There is evidence that pairs are twice as fast as individuals
  - we'll explore this in the benefits
- Quality produced by pairs seems to be higher than for individuals

Myth 2: I'll never get to work alone

- Pair programming only takes up part of the day
- Individual developers spend only about 30% of their time working alone
  - Meetings
  - Discussions
  - …

Myth 3: It only works with the right partner

- It seems to work with almost anyone
  - Works best with two people with similar skills
- There seems to be one type of person who causes trouble for everyone -- "my way or the highway"
  - These folks are a problem for any organization

Myth 4: It's only good for training

- Different people bring different experiences and skills to bear
- Different people have different knowledge of the project

Sit with a colleague and explain how you perform some common task, e.g. writing a function or using a tool. Compare notes. You'll be surprised what you can learn from others, even if you are an expert…

Myth 5: I'll have to share credit for everything

- Rewards can take many different forms
- Peer evaluation helps reward those who help the project
- Individuals can still own tasks

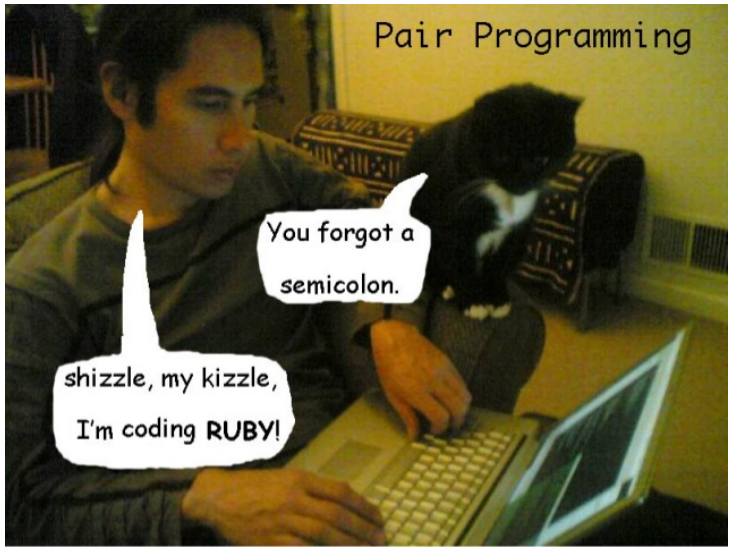
Myth 6: The navigator finds only syntax errors

- The navigator has to be seeing the big picture
  - Navigator thinking at higher level of abstraction
  - Driver thinking through the details

  Gestalt: whole is more than the sum of the parts
- The driver and the navigator should be talking
  - Collaboration identifies deeper problems

Myth 7: I won't be able to concentrate

- Pairs engage in pair mental flow
- Pairs keep each other on task and focused on the problem



Synergistic behaviors of Pair Programming

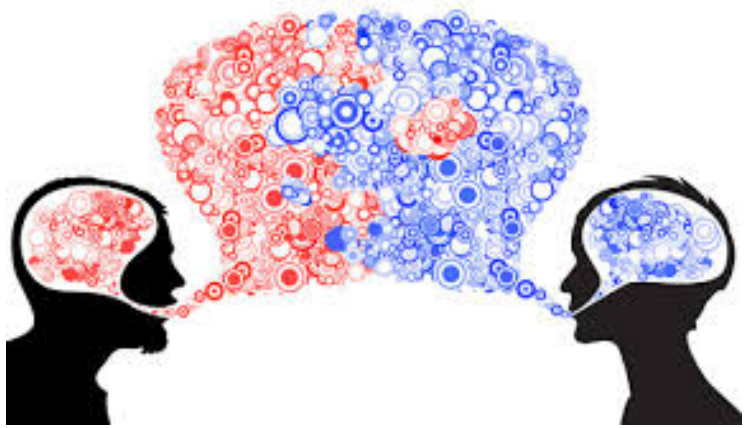
1. Pair pressure
2. Pair negotiation
3. Pair courage
4. Pair reviews
5. Pair debugging
6. Pair learning
7. Pair trust

Benefit 1: Pair pressure

- Pairs keep each other on task
  - Less likely to be distracted by other activities
- Pairs treat their shared time as more valuable
- Pairs follow standard processes more readily
  - E.g. following coding style guidelines

Benefit 2: Pair negotiation

- Pairs share the same goal
- They bring different ideas and points of view
- They discuss which strategies would work best
- They congratulate one another when they work out the best solution



Benefit 3: Pair courage

It is easier to get started if you know you have help

Feedback from your partner is encouraging

It’s okay to admit you don't know something

Benefit 4: Pair review

- Formal code reviews are uncommon with agile methods
  - e.g. Extreme Programming
- It is better to catch mistakes the moment they occur
  - Pair programming provides informal code reviews as the code is written
- It is more fun to pair than to do code inspections

Benefit 5: Pair Debugging

- Sometimes you need to describe the problem to someone else in order to solve it
  - Thinking out loud
- An intelligent partner will ask questions that you should have asked yourself

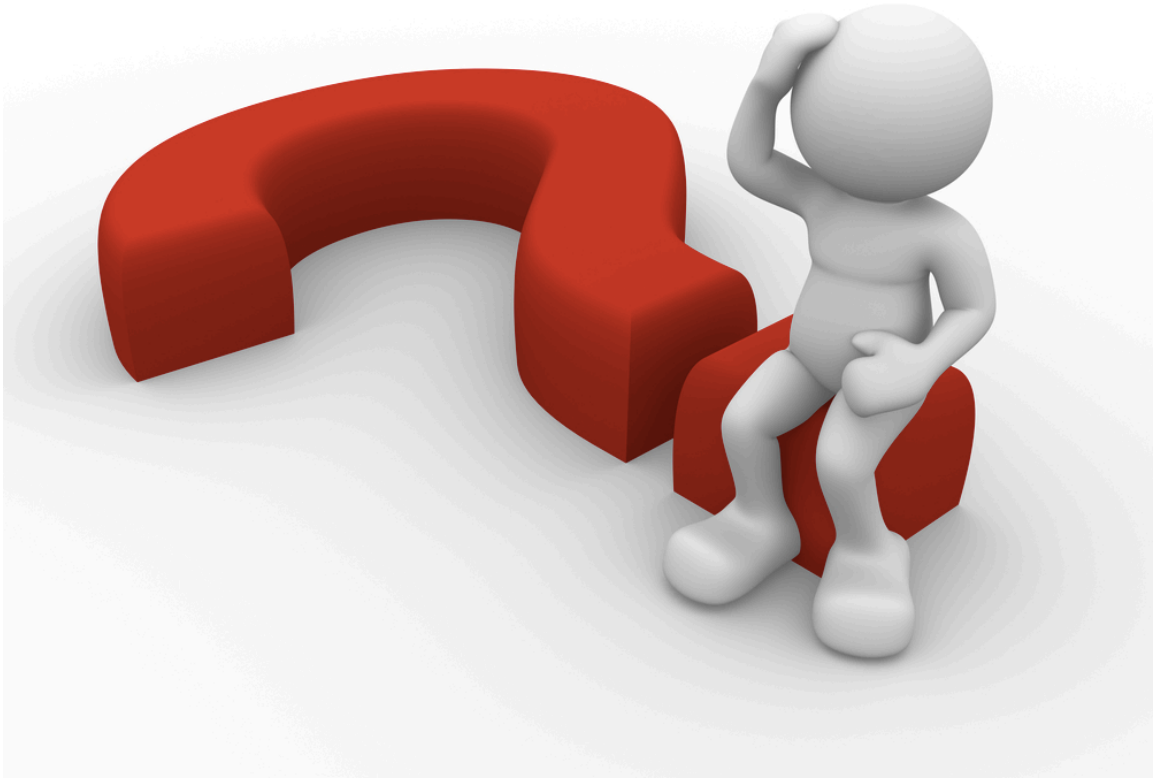
Benefit 6: Pair learning

- "You can observe a lot just by watching." – Yogi Berra
- Observe and learn from how someone else solves the problem
  - What techniques, tips, and tricks do they know and use?
- Pairs learn about the domain by working with others

Benefit 7: Pair trust

- The good of the many outweighs the good of the one
- May lead to better quality
  - You know that everyone is depending on you
  - Everyone is trusting you to do the right thing
- Trust encourages confidence which may improve speed

Questions?



In-class PP Practice

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target. You may assume that each input would have ***exactly*** one solution, and you may not use the *same* element twice.
Example:

Given nums = [2, 7, 11, 15], target = 9, Because nums[0] + nums[1] = 2 + 7 = 9, return [0, 1].

https://leetcode.com/problems/two-sum/submissions/1

In-class PP Practice

Given a string, find the length of the **longest substring** without repeating characters.
Examples:
Given "abcabcbb", the answer is "abc", which the length is 3.
Given "bbbbb", the answer is "b", with the length of 1.
Given "pwwkew", the answer is "wke", with the length of 3.
Note that the answer must be a **substring**, "pwke" is a *subsequence* and not a substring.

https://leetcode.com/problems/longest-substring-without-repeating-characters/submissions/1

STEVENS

INSTITUTE *of* TECHNOLOGY

THE INNOVATION UNIVERSITY®

stevens.edu

Thank You