SSW-555: Agile Methods for Software Development

Continuous Integration
Pair Programming

Dr. Richard Ens
Software Engineering
School of Systems and Enterprises

Today's topics

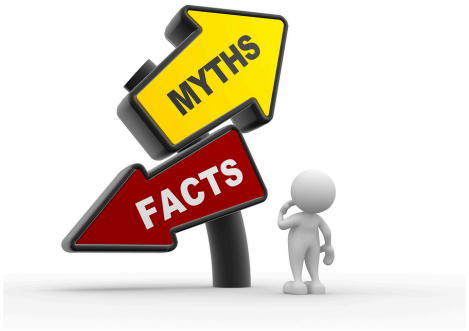
Overview of Continuous Integration

Compare deferred and continuous integration

Overview of pair programming

7 Myths of pair programming

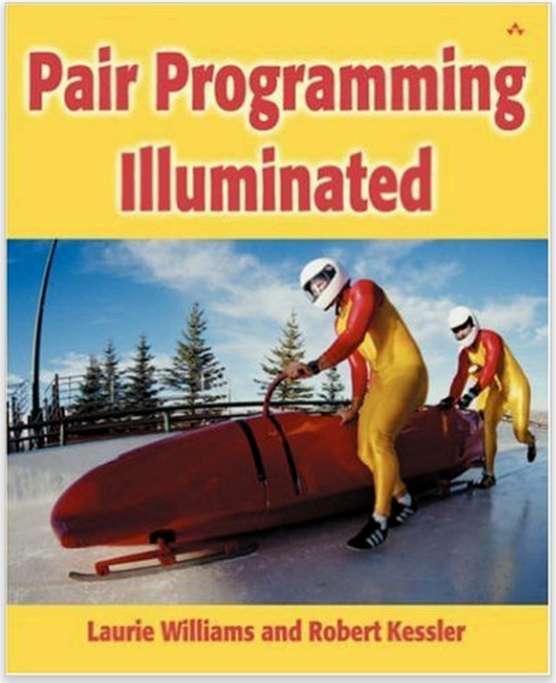
7 Synergistic practices



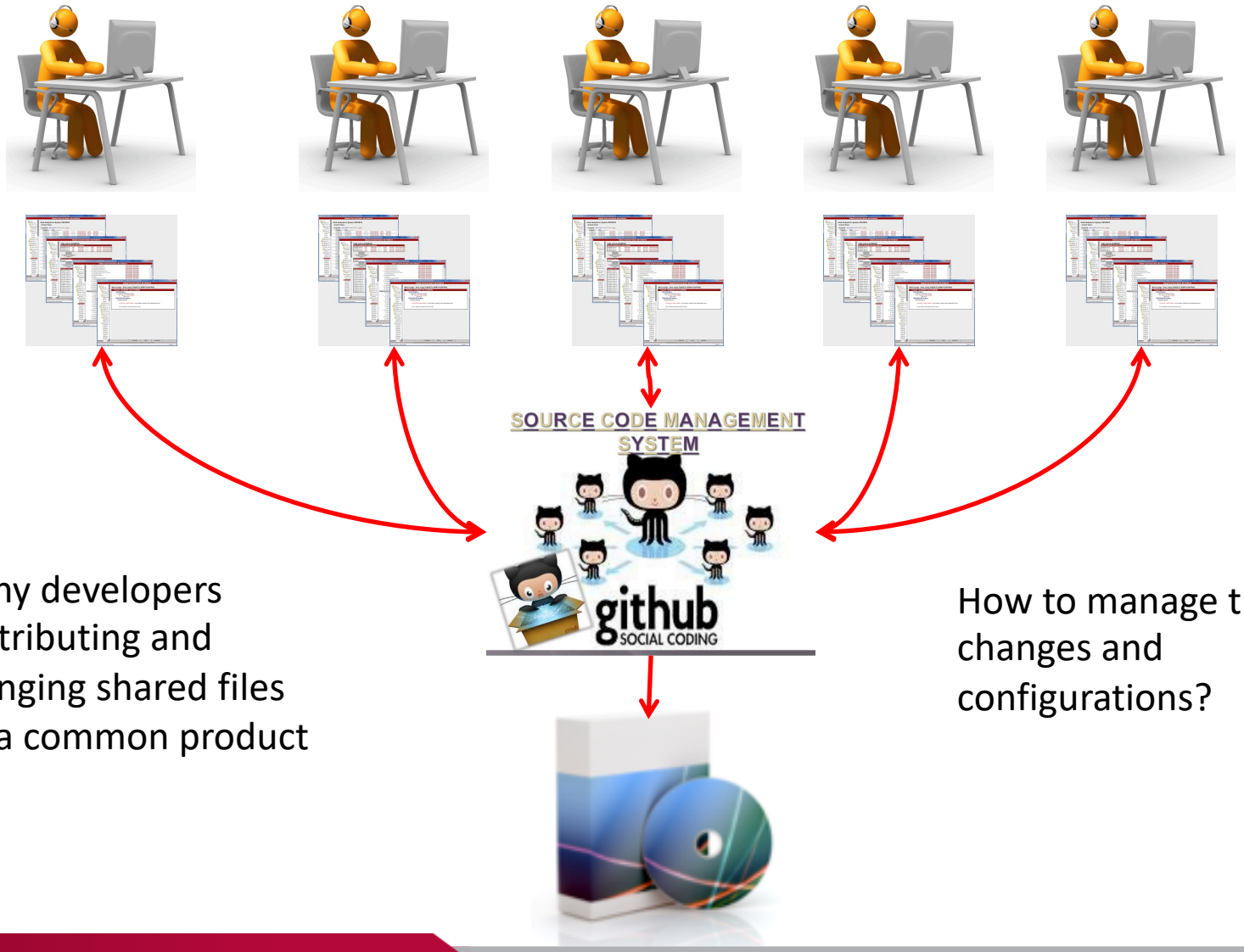
Acknowledgements

https://www.martinfowler.com/articles/continuousIntegration.html

Pair Programming Illuminated by Laurie Williams and Robert Kessler, Addison-Wesley 2003.



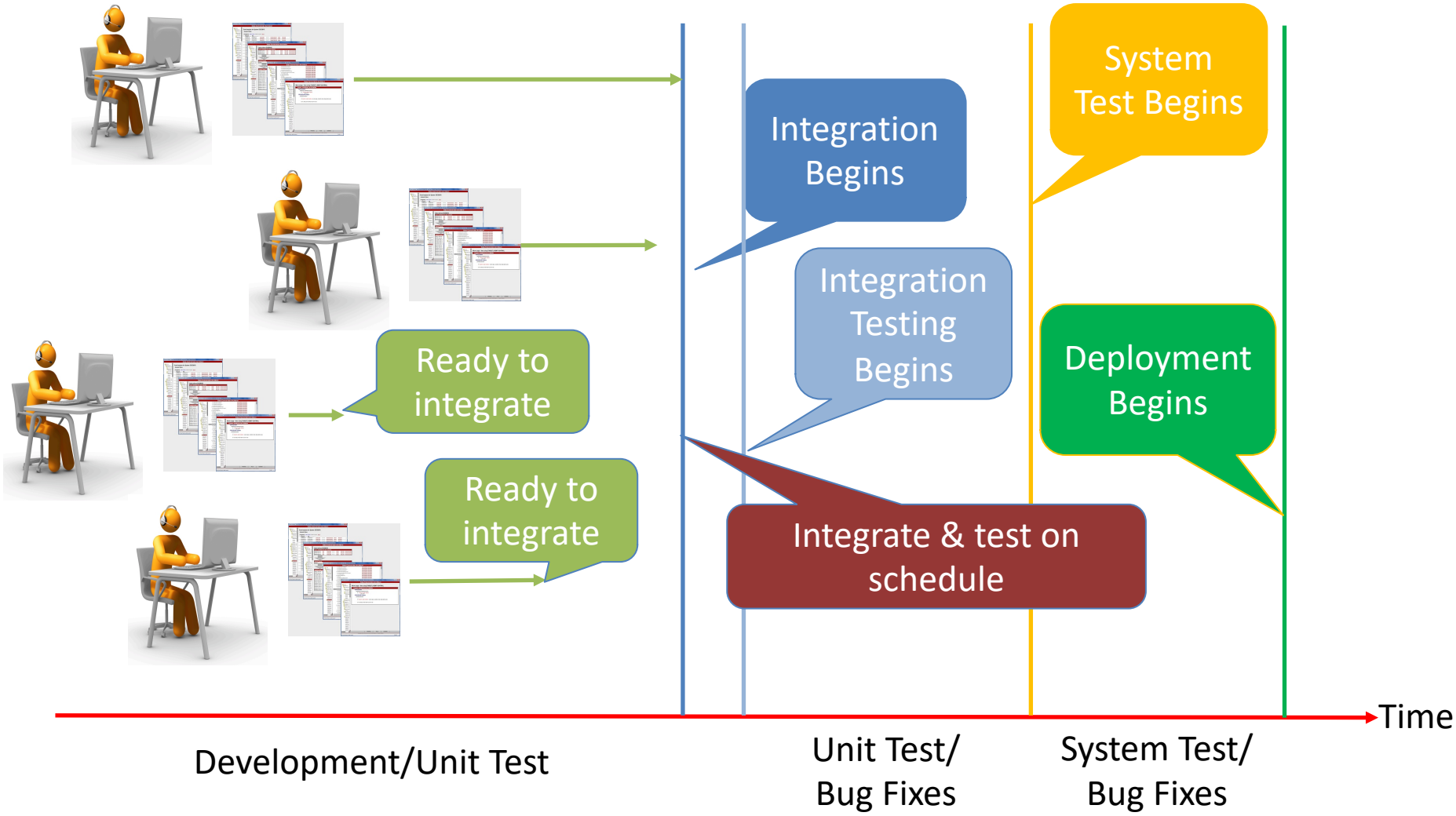
CI: What problem are we trying to solve?



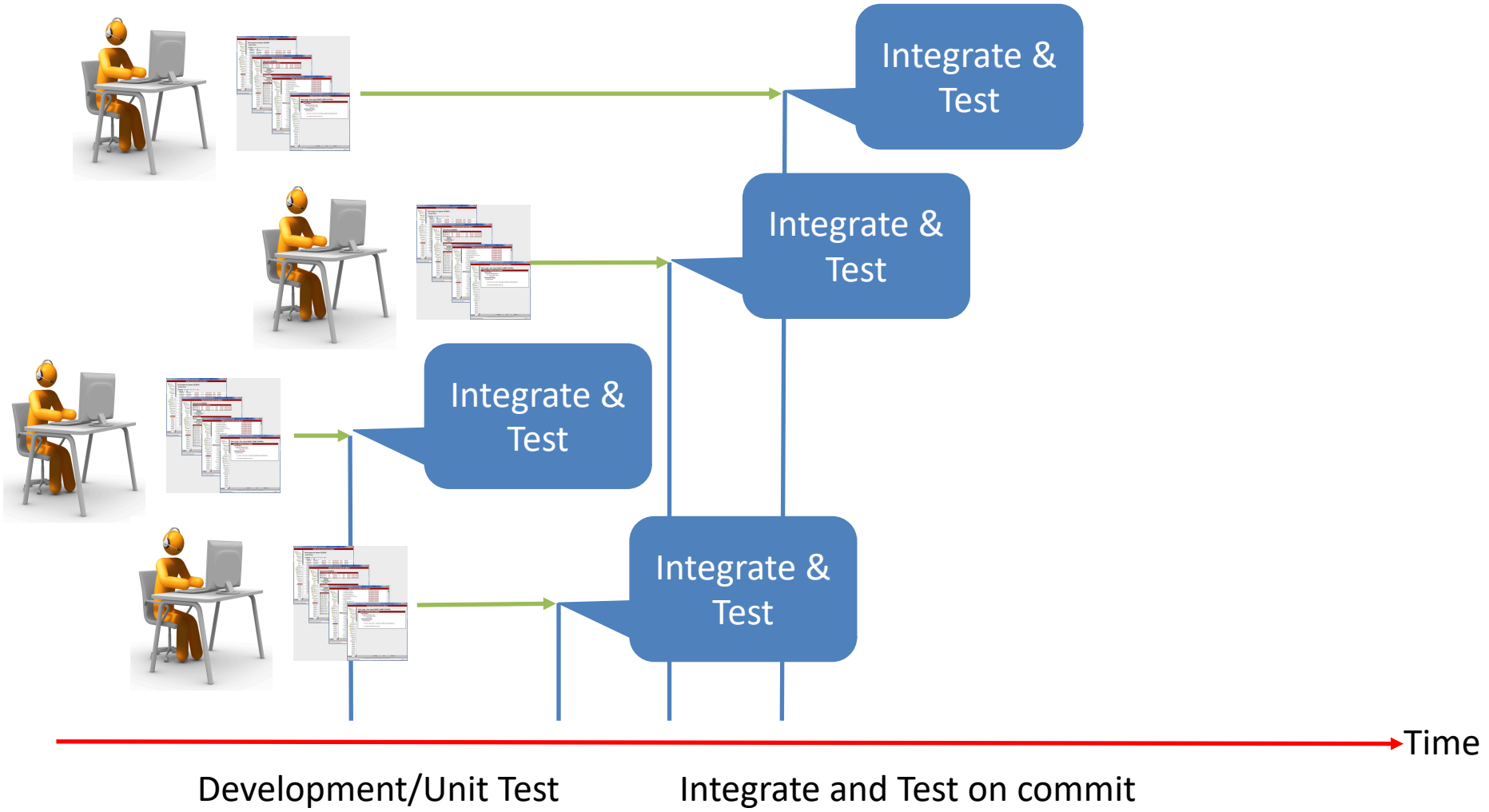
Many developers contributing and changing shared files for a common product

How to manage the changes and configurations?

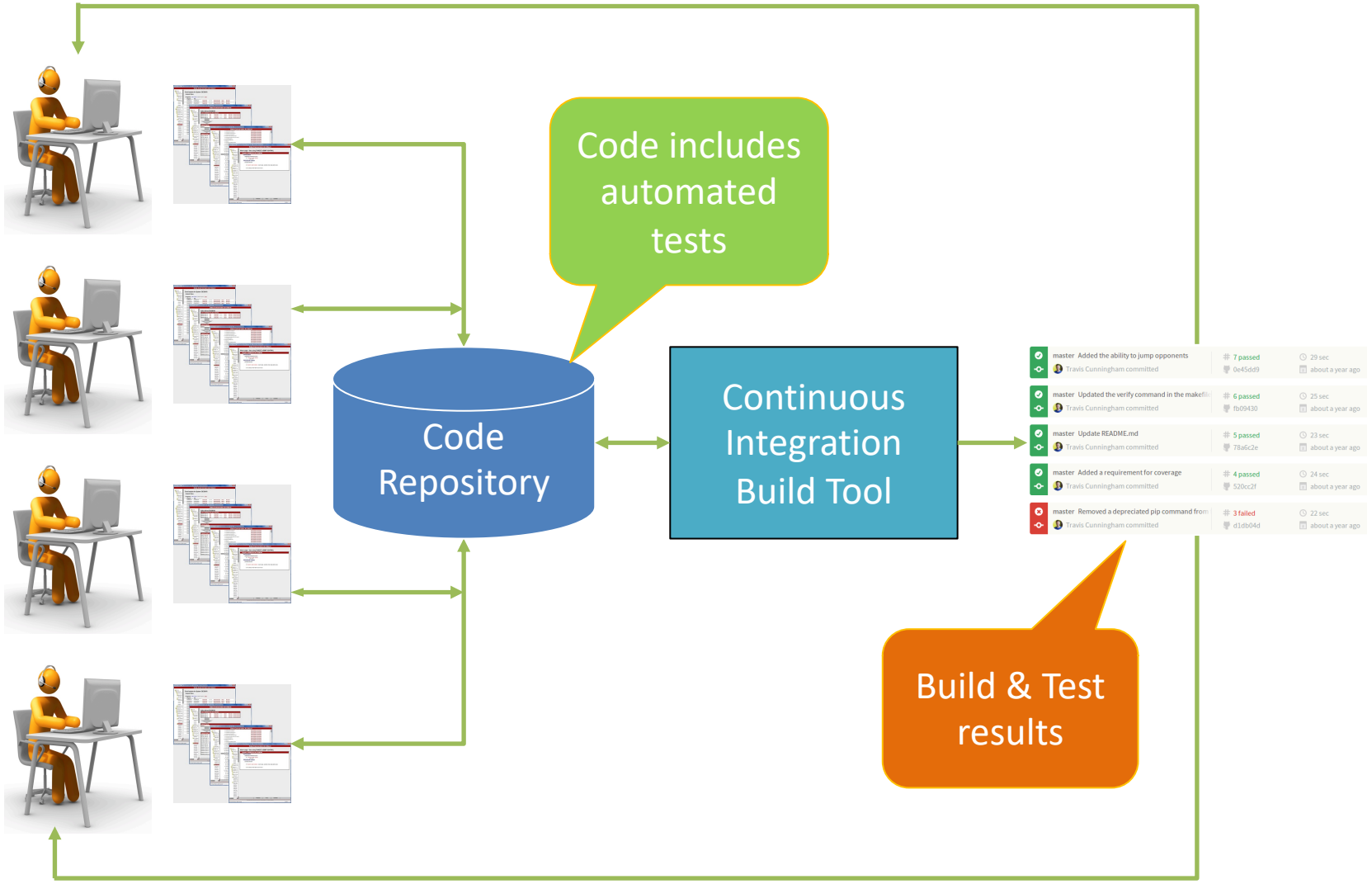
Timing: Deferred integration (e.g. Waterfall)



Timing: Continuous integration (e.g. Agile)



Continuous Integration Flow



CI Best Practices

- Maintain a single source code repository
  - Keep "everything" needed to build in the repository
- Automate the build process
- Automate testing of new builds
- Frequent code commits from all developers
  - At least daily if not more frequent
- Build and test in deployment environment
- Keep builds fast
- Fix build problems quickly
- Make it easy for everyone to get latest build

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

- Open Source
- Integrated with GitHub
- Hosted on GitHub

- Open Source
- Extensible through plugins
- Supports many languages

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

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

https://dzone.com/articles/top-8-continuous-integration-tools

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

Have you seen this?

Is this efficient?

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

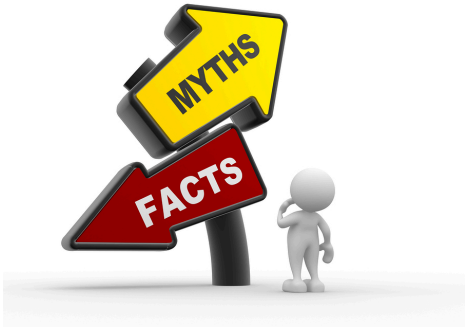
Pairs produce higher quality work than individuals

- 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

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

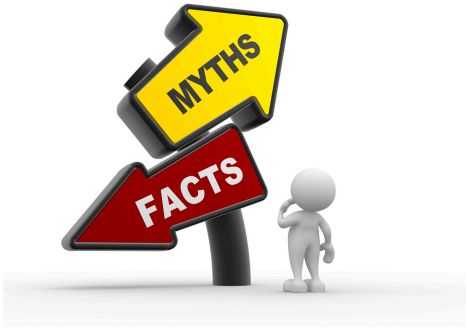
Myth 1: It will take twice as long

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

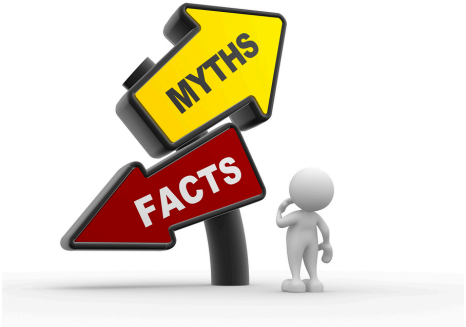


Myth 2: I'll never get to work alone

Pair programming only takes up part of the day

Individual developers spend only about 30% of their time working alone

- Meetings
- Discussions
- …



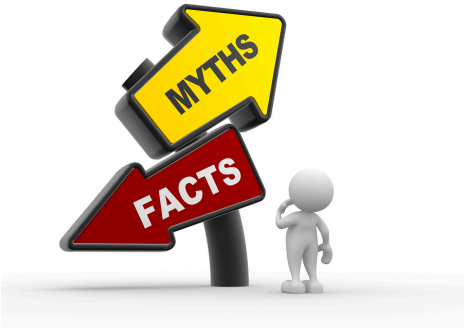
Myth 3: It only works with the right partner

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

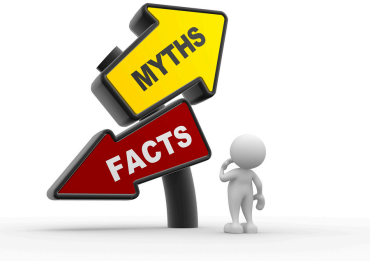


Myth 4: It's only good for training

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…

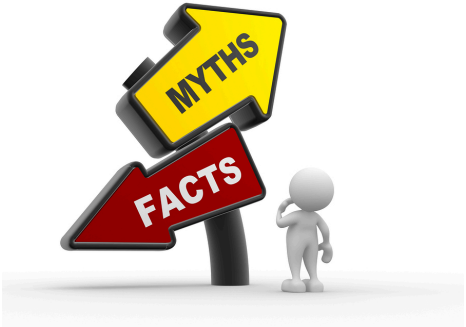


Myth 5: I'll have to share credit for everything

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

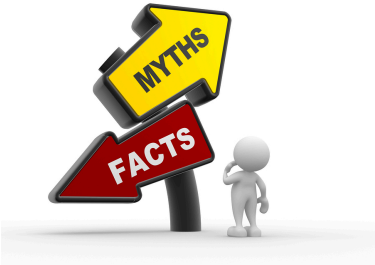
The navigator has to be seeing the big picture

- 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

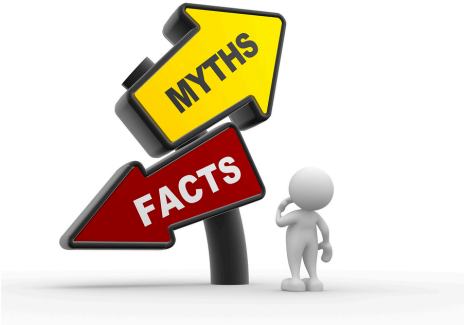
- Collaboration identifies deeper problems



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



Break Out Groups

1) Break into groups of around 6 people
2) Discuss the following and one person email me the answers including group members names
3) Total time 15 minutes

1. You are a Scrum Master and one of Scrum's foundations is to allow the developers to form their own Scrum teams.
    - Based on your experience working with teams, how would you resolve conflicts? Example – Employee complaining that his teammate is not doing his share of work.
    - Based on your experience working with teams, how would you resolve dysfunctional teams? – Example – One team member does not like working with the other team member.

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

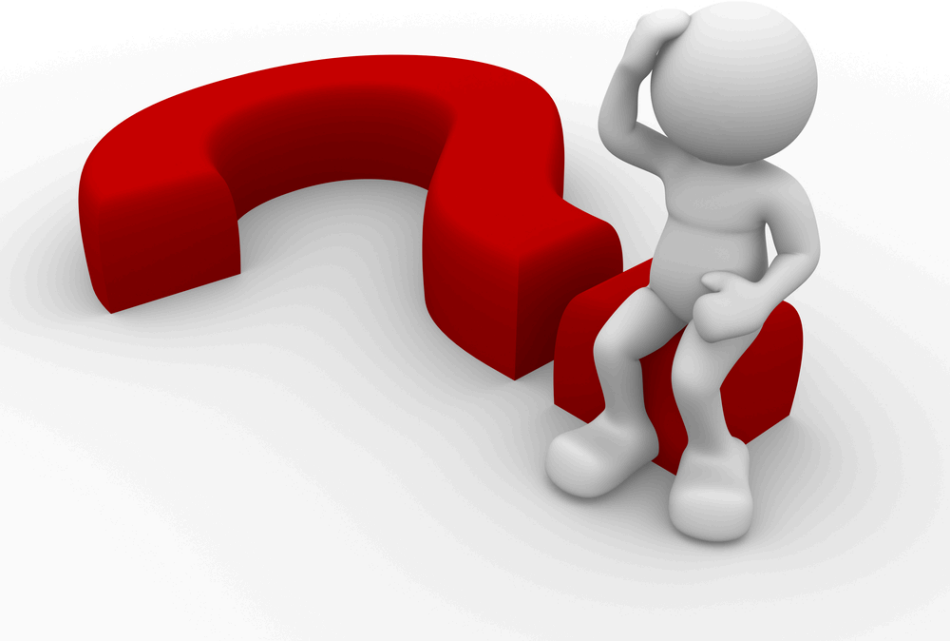
The good of the many outweighs the good of the one

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?



Classroom exercise

Write a function that takes two arguments:

- A string, s
- An integer, most

Return a copy of the string that has no more than 'most' consecutive occurrences of any character, e.g.

aaabbc, 2 → aabbc
aabcc, 1 → abc
aaaabbbc, 2 → aabbc

One solution

```
36  def remove_dups2(s, most):
37      """ remove repeated characters in s where the number of consecutive occurrences > most
38          Add the char the first time you see it and each subsequent time
39      """
40      last = None  # track the last character checked
41      cnt = 0 # number of consecutive occurrences of last
42      out = ""  # the new output string
43
44      for c in s:
45          if c != last:
46              last = c
47              cnt = 0
48
49          cnt += 1
50
51          if cnt <= most:
52              out += c
53
54      return out
```

Reverse digits

Given a positive integer, return an integer with the digits reversed, e.g.

reverse_digits(123) == 321
reverse_digits(12345678) == 87654321
reverse_digits(0) == 0
reverse_digits(1) == 1

One solution

```
import unittest

def reverse_digits(n):
    """ given an integer, reverse the digits in the string using only numbers, not strings """
    rev = 0

    while n != 0:
        print("n={} rev={}".format(n, rev))
        rev = (rev * 10) + (n % 10)
        n //= 10

    return rev

class ReverseDigitsText(unittest.TestCase):
    def test_reverse_digits(self):
        self.assertEqual(reverse_digits(123), 321)
        self.assertEqual(reverse_digits(12345678), 87654321)
        self.assertEqual(reverse_digits(0), 0)
        self.assertEqual(reverse_digits(1), 1)

if __name__ == "__main__":
    unittest.main(exit=False)
```

Questions?