CS 188

Scalable Internet Services

Andrew Mutz

October 3, 2019

About PTEs

If you are unsure you want to take this class and may drop, please drop early.

- For the good of your teammates
- For the good of other students
- Please consider taking the class in the future.

Introduction

There Is In Fact A Tech-Talent Shortage And There Always Will Be

*Posted May 5, 2013 by **Gregory Ferenstein** (@ferenstein)*

Next Story

For America to maintain its fragile role as the most innovative nation on earth, it must perpetually attract the world's best and brightest. There will always be trailblazing engineers who stay in their home country, leaving the United States one notch below its potential. Yet, on the heels of comprehensive immigration reform, a new viral economic study claiming that there is no tech talent shortage has skewed the national discussion over why we need to aggressively attract high-skilled immigrants in the first place.

http://techcrunch.com/2013/05/05/there-is-in-fact-a-tech-talent-shortage-and-there-always-will-be/

Introduction

IT jobs market booms, but talent in short supply

by:
Jake O'Donnell
News Writer

Published: 06 Mar 2015

Many companies need to fill IT jobs this year, but there's a shortage of available talent, and many haven't figured out the mobile skills they need.

Software developer shortage transcends international boundaries

The dearth of software development talent isn't an issue restricted to U.S. businesses. Finding programmers, especially to fill positions in the growing field of health IT, is a global challenge, said speakers Tuesday during a panel discussion on developing a health IT workforce.

By Fred O'Connor | Oct 21, 2014 | IDG News Service | Comments

sources: http://searchconsumerization.techtarget.com/news/2240241877/IT-jobs-market-booms-but-talent-in-short-supply
http://www.techworld.com/news/apps/software-developer-shortage-transcends-international-boundaries-3581959/

Introduction

Software Engineering requires judicious use of scarce resources.

You are one of those scarce resources.

Modern engineering techniques are designed to optimize for your time.

Introduction

- Industrial Software Development Techniques
  - Agile & Scrum
  - TDD
  - Continuous Integration
  - Github workflow
  - Pairing

- For next time

For Today

Sprint 2: Starts October 14, 2016.

- Conduct a retrospective on how the last sprint went and how you can improve.
- Decide on a sprint commitment.
- Implement stories from the current sprint.

After today, guidelines like this should make complete sense to you

Agile & Scrum

What is Agile software development?

- Agile a collection of different approaches for developing software that has emerged as dominant in the last 15 years.
- Scrum is a popular form of Agile software development

We will discuss both, but first let's talk about the world before Agile.

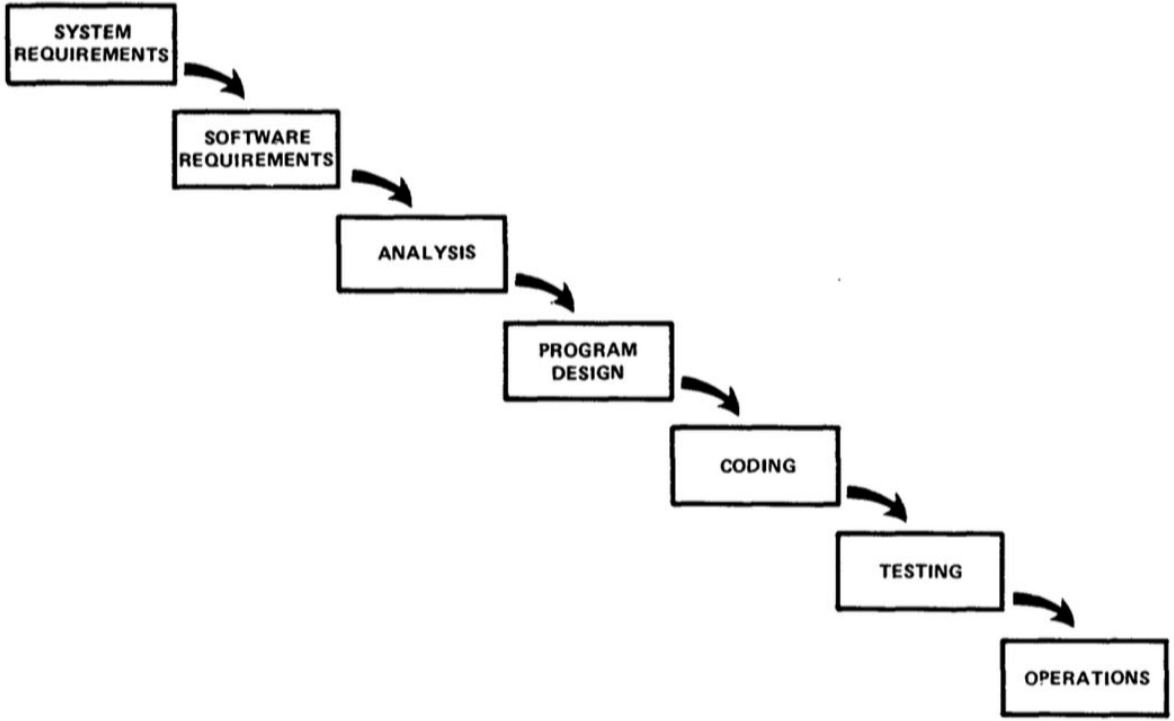
Waterfall

The origin of the Waterfall model is generally mis-attributed to Winston Royce, from his paper

- "Managing the Development of Large Software Systems"*
- In reality, he wasn't the first, and didn't advocate for what became Waterfall software engineering.
  - Herbert Benington would be a more a more accurate attribution.

*1970, Proceedings of IEEE WESCON 26 (August): 1–9.

Waterfall



Nevertheless this diagram from Royce's work is frequently cited

Waterfall

Each stage performs its role, and passes its deliverable to the next stage

- Req. Analysis -> Design -> Coding -> Testing -> Ops
- If software engineering is like manufacturing, this makes sense
- Strength: this approach allows deep specialization
- **What problems can you see with this approach?**

Waterfall

In practice, this technique works best when you have complete knowledge

- Software testers and devs know everything about how it will be deployed
- Software developers don't have too many surprises during testing
- Designers have complete understanding of the difficulty of each design option
- Requirements team understands the impact on design and development
    - Also requires this team to have a very good idea what the market wants.

Most of these things aren't true in practice.

Agile

In 2001, the “Agile Software Manifesto” was written.

- Kent Beck, Ken Schwaber, Jeff Sutherland, Dave Thomas

Values:

- **Responding to change** over following a plan
- **Working software** over comprehensive docs
- **Customer collaboration** over contract negotiation
- **Individuals and interactions** over processes and tools

Agile

Responding to change over following a plan

vs.

Agile

Working software over comprehensive docs

- Demonstration to customer
- Automated acceptance tests
- Descriptive unit tests
- Centralized styling & view code

vs.

- Description to customer
- Extensive documented requirements
- Code comments
- Style guide documentation

Agile

Customer collaboration over contract negotiation

"Here's what we've got so far."

"Uh... Now that I see it in action, I've changed my mind."

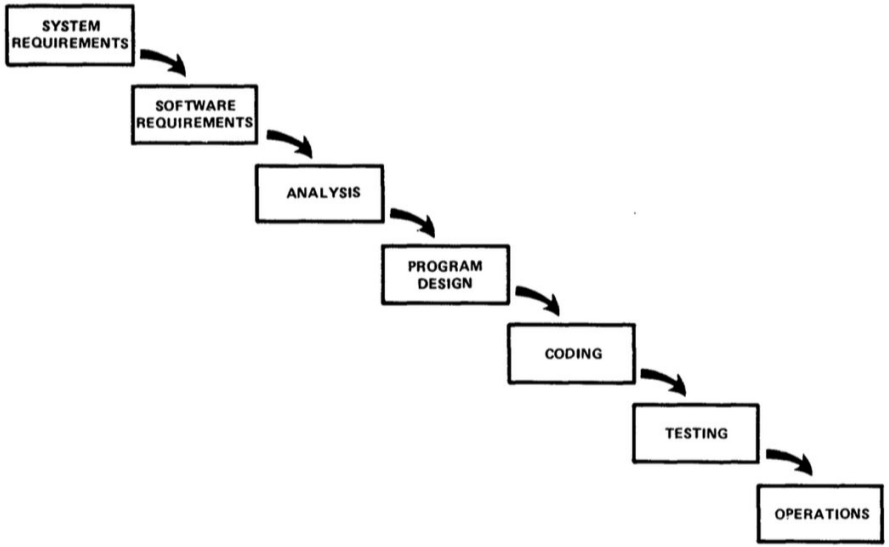
vs.

"I'm sure if I just build what they've asked for, they'll love it."

Agile

Individuals and interactions over processes and tools

vs.



Scrum

Scrum is a specific type of Agile software development

- Developed by Ken Schwaber and Jeff Sutherland in the early 1990s
- We will use an abridged version of Scrum for this class

- Other popular alternatives:
  - Kanban: Focus on controlling W.I.P., no sprints
  - XP: Emphasis on feedback systems, through automated testing and pair programming

Scrum

Scrum defines roles

- Product Owner: Understands the needs of the end user and prioritizes needs.
- The Team: the people who are building the software. Roles are intentionally vague.
- Scrum Master: Responsible for making sure the process is followed and helping to resolve blocking issues

In this class we won't have a defined product owner or scrum master

Scrum

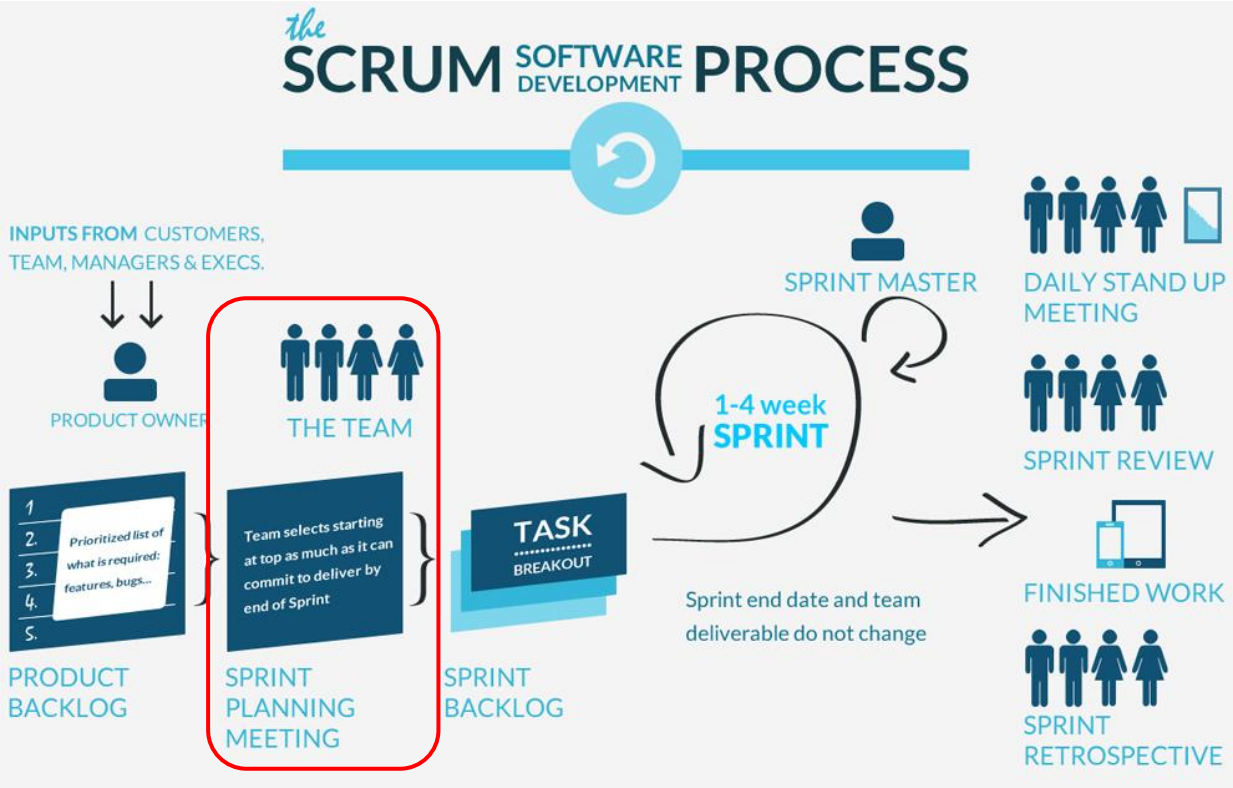
Story

- A unit of functionality. Ideally this is user visible.
  - “As a _____, I can ___, in order to ____”
  - “As a user, I can add items to my shopping cart, in order to eventually purchase them.”

Sprint

- A specific length of time within which to work
- Sprints may have the same schedule as releases, but not necessarily

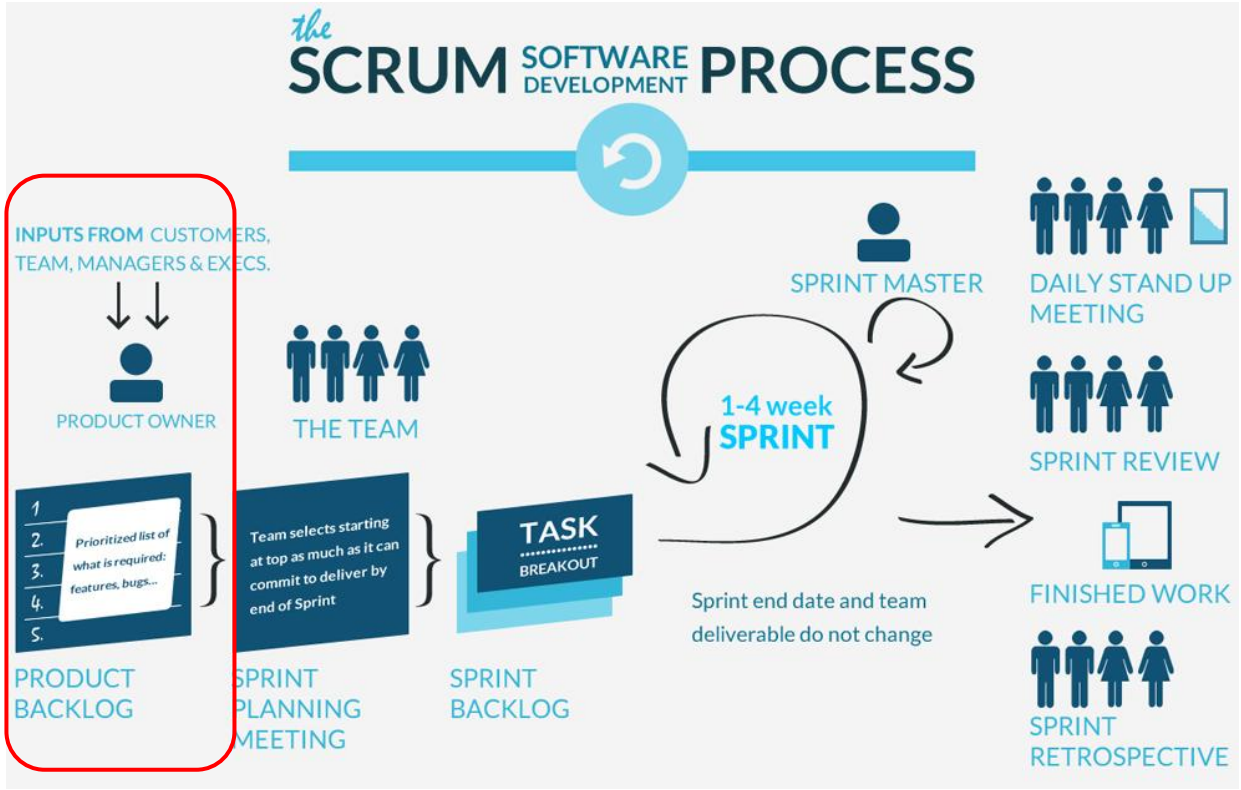
Scrum



Source: https://www.maxxor.com/images/Scrum.jpg

Each sprint begins with a sprint planning meeting to decide the "sprint commitment"

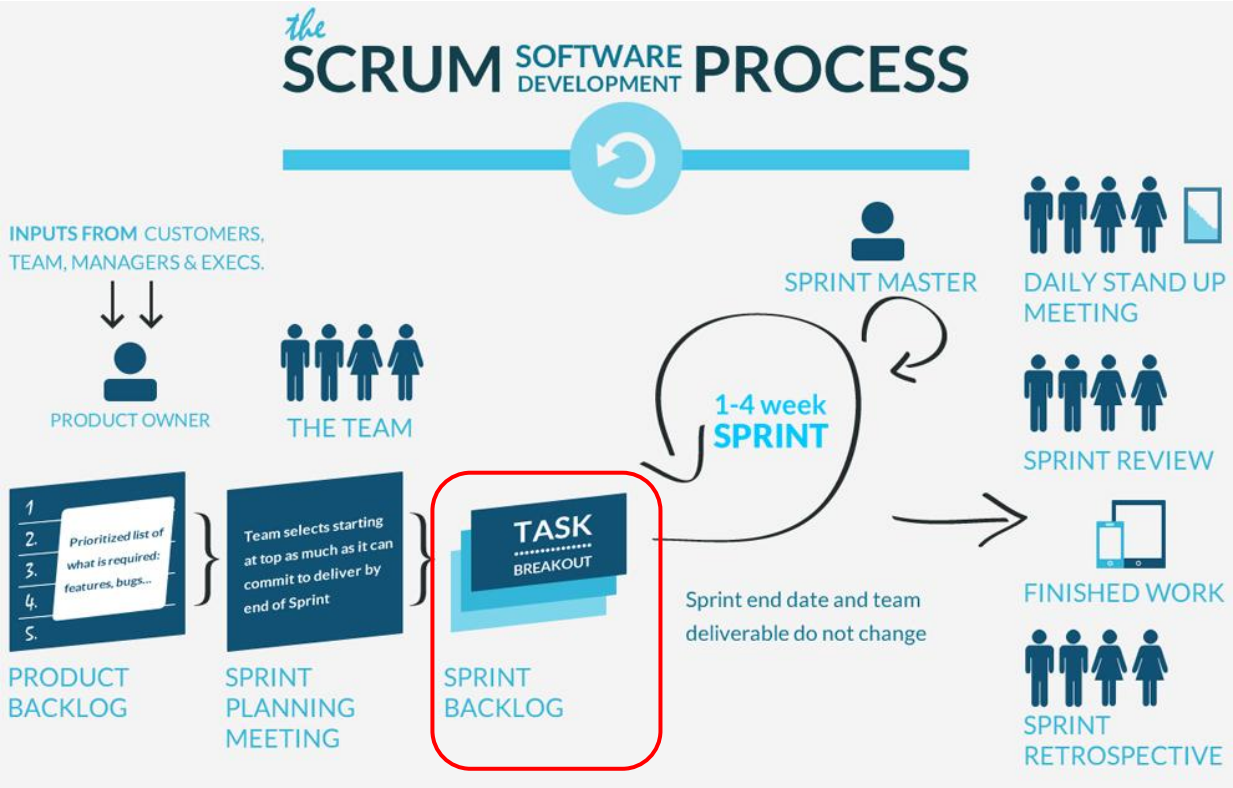
Scrum



Source: https://www.maxxor.com/images/Scrum.jpg

The stories considered are the ones at the top of the product backlog (prioritized by product owner).

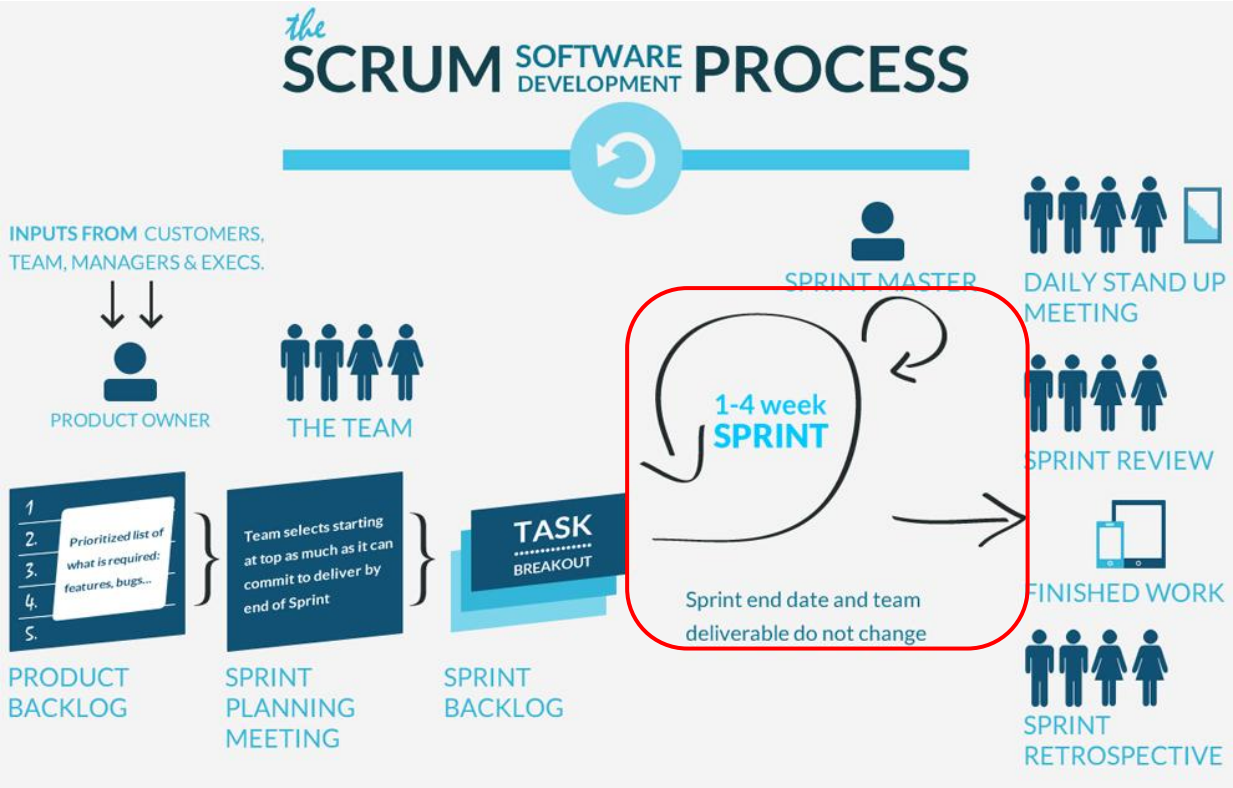
Scrum



Source: https://www.maxxor.com/images/Scrum.jpg

The team decides on its commitment with consideration of its current velocity.

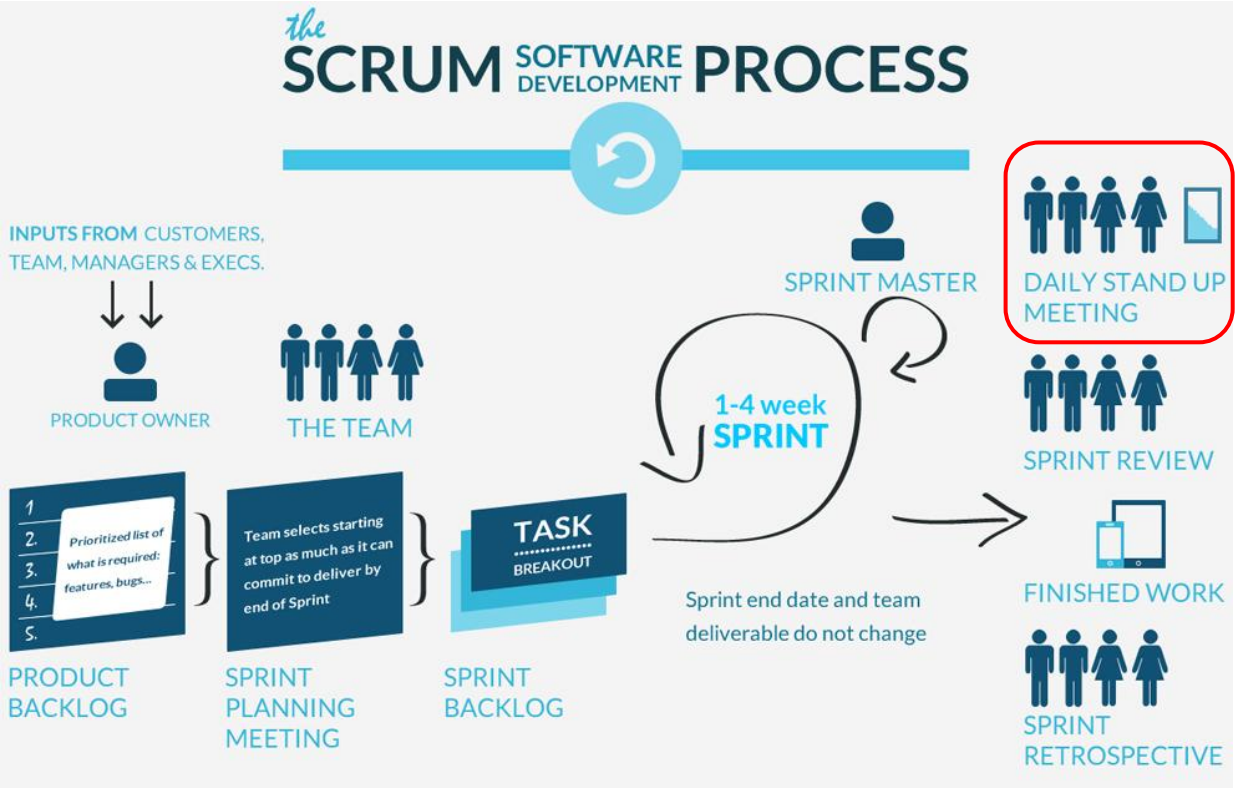
Scrum



Source: https://www.maxxor.com/images/Scrum.jpg

The sprint commitment is frozen and the team works on that material for the duration of the sprint

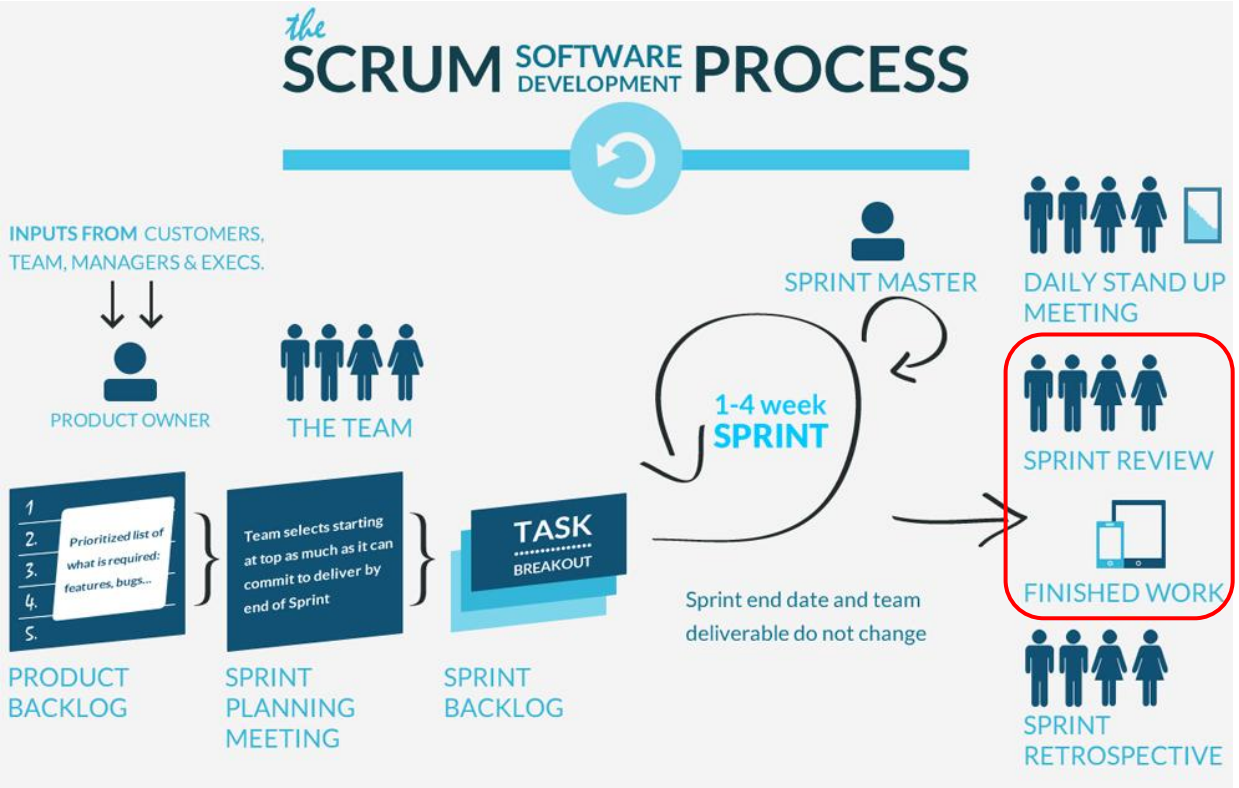
Scrum



Source: https://www.maxxor.com/images/Scrum.jpg

Each day the team meets quickly to discuss what remains to be done in the sprint, and who will do it.

Scrum



Source: https://www.maxxor.com/images/Scrum.jpg

At the end of each sprint, the work is presented. Each sprint should produce a **potentially** shippable product

Scrum

Source: https://www.maxxor.com/images/Scrum.jpg

After each sprint, the team meets to discuss how they can improve the manner in which they work.

Scrum

For our projects we will use a lighter version of Scrum:

- No scrum master or product owner
- Each week the team will choose a sprint commitment
- Each week the team will work together to accomplish the sprint, and have working code deployed on AWS at the end.
- Each week the team will demo to me
- Each week the team will conduct a retrospective

Test-Driven Development

Let’s say we have a large group of software engineers working on the same project.

- Every day, each engineer makes many changes to the project
- Human error is common and information is not global
- Errors will happen, how do we find out about them?
  - Have the QA department check for mistakes?
  - Let the compiler warn us of mistakes?

Test-Driven Development

Humans can be used to check for defects, but this is expensive.

Type systems and compilers work well to statically check for defects

- Formal verification tools exist, but are not widely used in industry

Automated testing is another level of static defect detection

Test-Driven Development

Automated testing…

- … is writing testing code to execute your production code and make assertions about how it should behave.
- … can be measured by code coverage tools that determine which code paths are executed by your tests.
- … allows you to build large and complex systems with very permissive languages.
    - For a language like Ruby, it replaces many of the static type checking features of other languages

Test-Driven Development

So if automated testing is important and we want high code coverage (not 100%), how do we get there?

Test-Driven Development

So if automated testing is important and we want high code coverage (not 100%), how do we get there?

- Don't write any production code, until there is test code that tests it
- Write the minimal amount of production code to make it pass

1. Write the test and watch it fail (AKA "Red")
2. Write the production code to pass (AKA "Green")
3. Clean up your design (AKA "Refactor")

Test-Driven Development Example

Fizz Buzz

- If the argument is divisible by three, return “Fizz”
- If the argument is divisible by five, return “Buzz”
- If the argument is divisible by both, return “Fizz Buzz”
- If none of these, return the argument.

Test-Driven Development Example

```
def test_divisible_by_3
  assert_equal ‘Fizz’, fb(3)
end
```

```
def fb(n)
end
```

Test-Driven Development Example

```
def test_divisible_by_3
  assert_equal ‘Fizz’, fb(3)
end
```

```
def fb(n)
  ‘Fizz’
end
```

Test-Driven Development Example

```
def test_divisible_by_3
  assert_equal ‘Fizz’, fb(3)
end
def test_divisible_by_both
  assert_equal ‘Fizz Buzz’, fb(15)
end
```

```
def fb(n)
  ‘Fizz’
end
```

Test-Driven Development Example

```
def test_divisible_by_3
  assert_equal ‘Fizz’, fb(3)
end
def test_divisible_by_both
  assert_equal ‘Fizz Buzz’, fb(15)
end
```

```
def fb(n)
  if n % 5 == 0
    ‘Fizz Buzz’
  else
    ‘Fizz’
  end
end
```

Test-Driven Development Example

```
def test_divisible_by_3
  assert_equal ‘Fizz’, fb(3)
end
def test_divisible_by_both
  assert_equal ‘Fizz Buzz’, fb(15)
end
def test_divisible_by_5
  assert_equal ‘Buzz’, fb(5)
end
```

```
def fb(n)
  if n % 5 == 0
    ‘Fizz Buzz’
  else
    ‘Fizz’
  end
end
```

Test-Driven Development Example

```
def test_divisible_by_3
  assert_equal ‘Fizz’, fb(3)
end
def test_divisible_by_both
  assert_equal ‘Fizz Buzz’, fb(15)
end
def test_divisible_by_5
  assert_equal ‘Buzz’, fb(5)
end
```

```
def fb(n)
  if n % 5 == 0
    if n % 3 == 0
      ‘Fizz Buzz’
    else
      ‘Buzz’
    end
  else
    ‘Fizz’
  end
end
```

Test-Driven Development Example

```
def test_divisible_by_3
  assert_equal ‘Fizz’, fb(3)
end
def test_divisible_by_both
  assert_equal ‘Fizz Buzz’, fb(15)
end
def test_divisible_by_5
  assert_equal ‘Buzz’, fb(5)
end
def test_divisible_by_neither
  assert_equal 17, fb(17)
end
```

```
def fb(n)
  if n % 5 == 0
    if n % 3 == 0
      ‘Fizz Buzz’
    else
      ‘Buzz’
    end
  else
    ‘Fizz’
  end
end
```

Test-Driven Development Example

```
def test_divisible_by_3
  assert_equal ‘Fizz’, fb(3)
end
def test_divisible_by_both
  assert_equal ‘Fizz Buzz’, fb(15)
end
def test_divisible_by_5
  assert_equal ‘Buzz’, fb(5)
end
def test_divisible_by_neither
  assert_equal 17, fb(17)
end
```

```
def fb(n)
  if n % 5 == 0
    if n % 3 == 0
      ‘Fizz Buzz’
    else
      ‘Buzz’
    end
  elsif n % 3 == 0
    ‘Fizz’
  else
    n
  end
end
```

Test-Driven Development Example

```
def test_divisible_by_3
  assert_equal ‘Fizz’, fb(3)
end
def test_divisible_by_both
  assert_equal ‘Fizz Buzz’, fb(15)
end
def test_divisible_by_5
  assert_equal ‘Buzz’, fb(5)
end
def test_divisible_by_neither
  assert_equal 17, fb(17)
end
```

```
def fb(n)
  if n % 3 == 0
    ‘Fizz’
  elsif n % 5 == 0
    ‘Buzz’
  elsif n % 3 ==0 && n % 5 == 0
    ‘Fizz Buzz’
  else
    n
  end
end
```

Test-Driven Development

This example used Unit Tests. We also can use:

- Controller tests
- Integration tests
- Selenium tests

Whenever you find and fix a bug, there should be a corresponding test to prevent regression.

Test-Driven Development Example

For this class, I encourage you to try test-driven development.

High test coverage can help you avoid getting stuck on bugs.

I won't be grading based on code coverage.

Continuous Integration

Integration:

- Taking independently developed changes and reconciling their conflicts

Integration can be very difficult and painful

- **So should we do it as rarely as possible or as frequently as possible?**

Continuous Integration

Martin Fowler, Chief Scientist, ThoughtWorks

"The effort of integration is exponentially proportional to the amount of time between integrations."

Continuous Integration

If we never let our changes diverge too much from the rest of the group, reconciling our changes will never be too hard.

Conclusion: commit early and often, and merge others' changes early and often.

How do we make sure we have successfully reconciled?

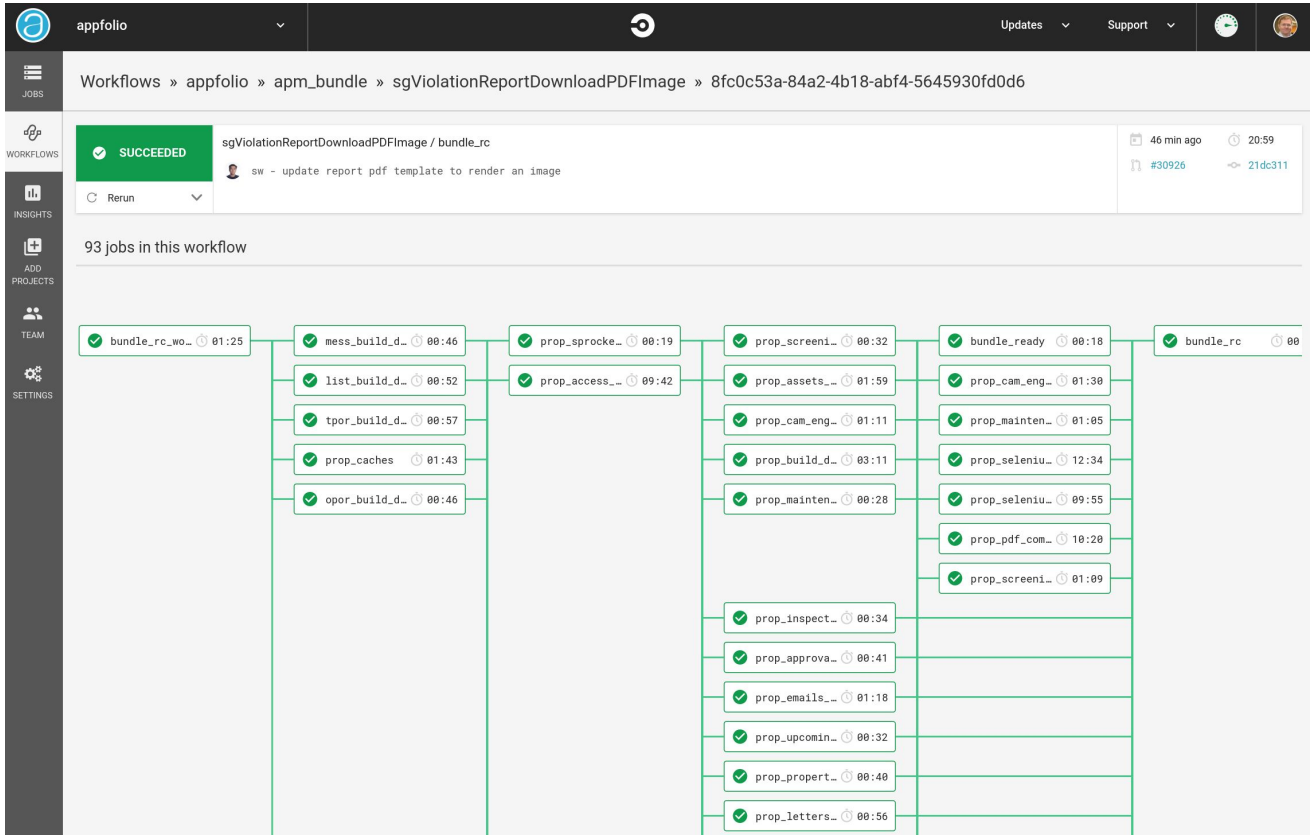
Continuous Integration

Automated tests!

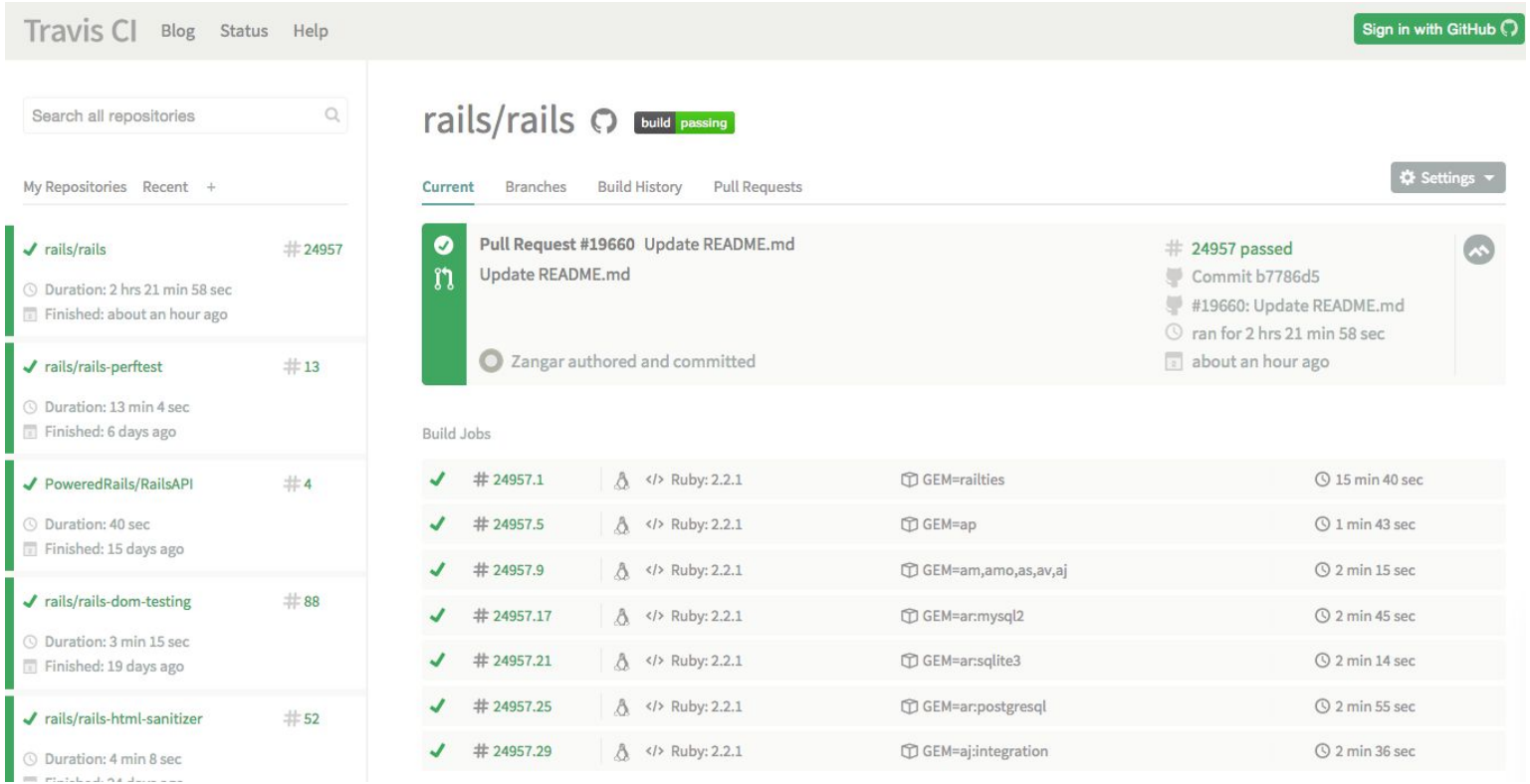
- Set up a “Continuous Integration” server that will monitor your source control system.
- Whenever anyone checks in new code, run all the tests.
  - The sooner you find and fix integration problems, the better.
  - This also prevents defects unrelated to integration.

CI server can also test code quality and security.

Continuous Integration



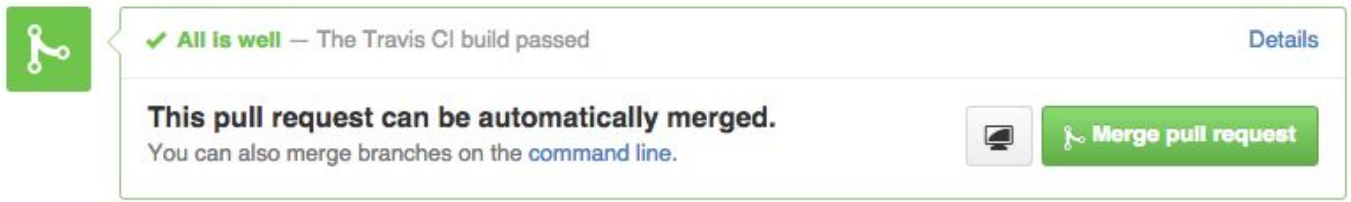
Continuous Integration



Continuous Integration

For this class, we will be using Travis-CI: http://travis-ci.org/

If you are doing TDD and creating automated tests, you can get immediate feedback on your changes in the GitHub interface:



Github Workflow

So we know that we don't want our changes to diverge too far from the rest of the group. What's the right way to use our source control system in this light?

There are two popular git-based workflow systems:

- Git-flow
- Github-flow

Github flow is simpler and recommended for this class.

Github Workflow

Github Flow:

- Get your local copy of master up to date
- Create a new branch for your feature
- Commit regularly
- Push to origin regularly
- Open a pull request when complete
- Peer can review your changes and merge to master
- Only merge to master if changes are green

Github Workflow

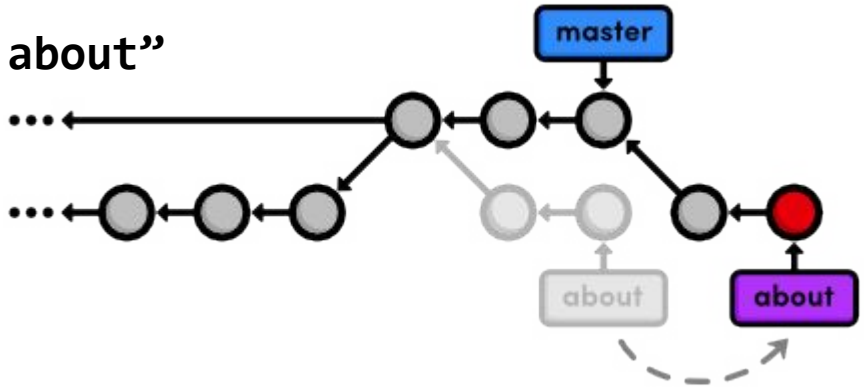
Github Flow:

- Get your local copy of master up to date
  - `git pull`
- Create a new branch for your feature
  - `git checkout -b feature_name`
- Commit regularly
  - `git add …; git commit -m “comments”`
- Push to origin regularly
  - `git push origin feature_name`

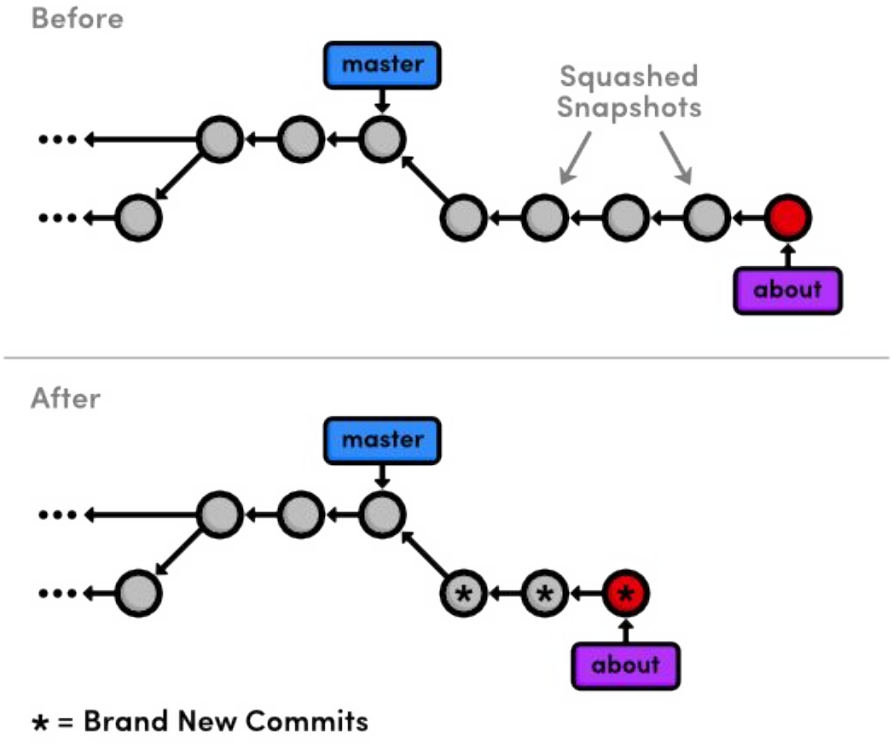
Github Workflow

- Remember, we want to be integrating regularly.
- Feature branches should live no longer than a day or two.
- If you want to reconcile your changes without merging to master, a git rebase is a great way to do this:

`“git rebase master about”`



Github Workflow



If you have been committing frequently and don't want to squash some commits, consider an interactive rebase:

"`git rebase -i about`"

Pair Programming

Github Flow:

- Get your local copy of master up to date
  - `git pull`
- Create a new branch for your feature
  - `git checkout -b feature_name`
- Commit regularly
  - `git add …; git commit -m “comments”`
- Push to origin regularly
  - `git push origin master`

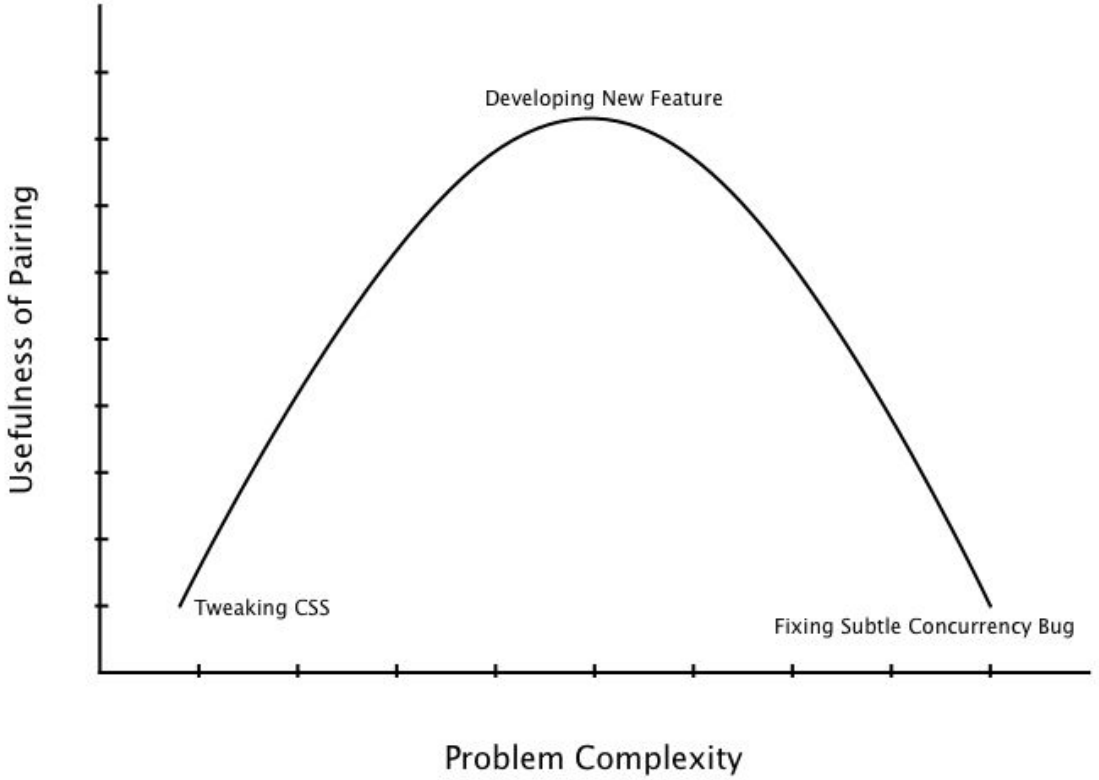
Pair Programming

Pair Programming: two developers share one computer and discuss all code that is being written.

Different flavors:

- Driver-Navigator: One person does most of the implementation while the other watches, discusses, thinks of consequences.
- Ping-pong pairing: One person writes the test, the other makes it pass
    - Frequently used while learning TDD and pair programming

Pair Programming

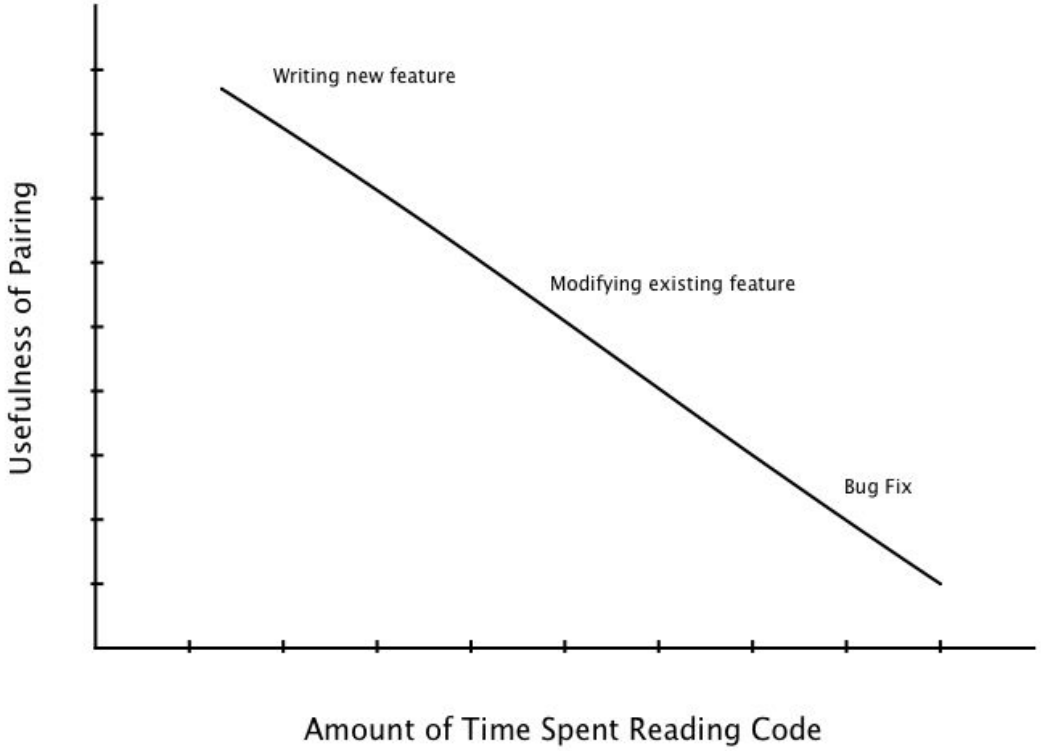


Like any tool, it has strengths and weaknesses

Pair Programming

Like any tool, it has strengths and weaknesses

Pair Programming



Like any tool, it has strengths and weaknesses

Pair Programming

For this class, pair programming is not required but is recommended.

- Can make some monotonous tasks more fun

It is possible to be bad a pairing. If you're really not enjoying it, don't do it.

Conclusion

Sprint 2: Starts October 25, 2019.

- Conduct a retrospective on how the last sprint went and how you can improve.
- Decide on a sprint commitment.
- Implement stories from the current sprint.

After today, guidelines like this should make complete sense to you

For Next Time...

For Friday:

- Bring your laptop (charged) to Dodd Hall 78 or Rolfe Hall 3126 (depending on your section)

For Next Tuesday:

- Read and do ch. 3 through 7 from Hartl's Rails book
- Finish Ruby Code Academy
    - https://www.codecademy.com/learn/learn-ruby