# W4156

## Continuous Integration

# By the end of today

Any code change you make will magically* flow through to the production environment in seconds.

You and your team will be a pipeline of ideas to code to working software to your users

* magic = a lot of engineering work, discipline/process and culture commitment from your team. We have not discovered magic yet!

# Agenda

Continuous Integration, Continuous Delivery and Continuous Deployment

- ❏ The motivation
- ❏ CI as a *practice*
- ❏ CI *technologies*
- ❏ CD
- ❏ CD within the project

# Motivation

Our goal is *flow of value* which is idea to working software in production

- Quickly
- Low Variance
- High Quality

# Critique this process

Critique this manufacturing process for a *new concept car**

1. Initial design meeting to roughly sketch
2. Divide up tasks amongst team/people
    a. Chassis
    b. Electrics
    c. Upholstery
    d. Drive Train
3. Everyone disappear off to different rooms/countries to build
4. Once everyone is ‘done’ with their component all parts collected
5. Attempt to assemble the car

* concept car is a better analogy for the novelty of things we often build in software

# Critique

1. Is this process efficient?

2. If
   a. yes: why?
   b. no: what are the problems you can foresee?

3. How do you think this process scales in complexity?
   a. Say we were designing a concept spaceship?

* concept car is a better analogy for the novelty of things we often build in software

# Why?

Intuitively, doesn't seem smart

1. Very unlikely to be *efficient* / work
2. Components unlikely to *integrate*

Let's dig deeper. *What else?*

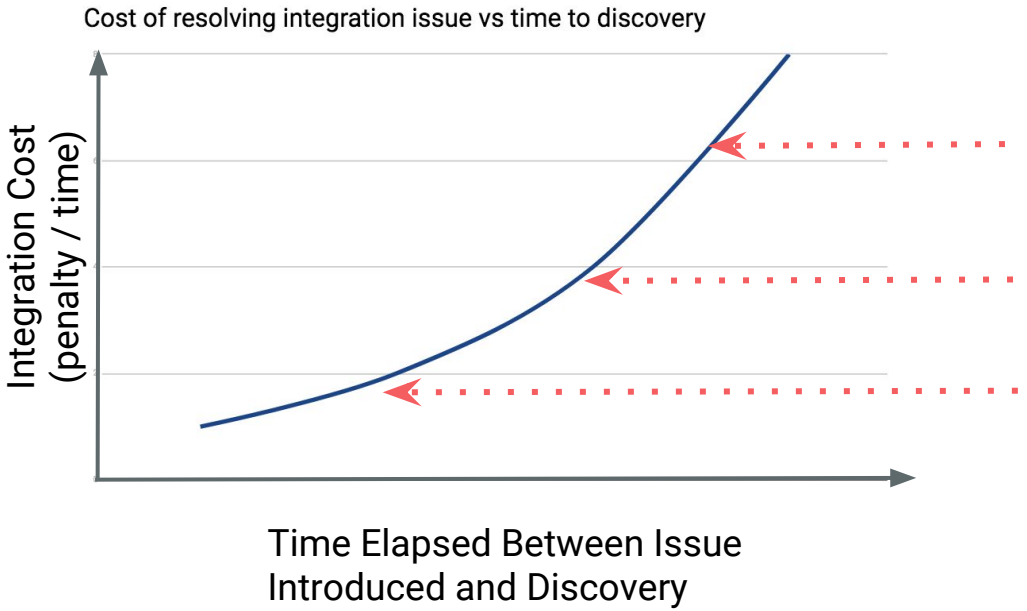
A) Significant time in integrating (may not even work / drawing board)
B) P(High) Significant cost to rework (redesign and rebuild)

Deeper still. *Why?*

C) Lack of communication
D) Longer an issue is unsurfaced the higher the cost
E) What is time taken as a function of integration 'size'?

* concept car is a better analogy for the novelty of things we often build in software

# Why is delayed integration expensive?




3. The longer it take us to uncover we went the wrong direction the more throw away work and harder it is to bring back together
(woops, wheel doesn't fit chassis)

2. We build upon that decision. We build molds, make other design decisions, build technology
(design and build wheel / chassis)

1. We make a requirement assumption, make a design decision, size a part, etc, etc
(wheel is this size and weight)

# Bingo

You wouldn’t build a concept car that way.

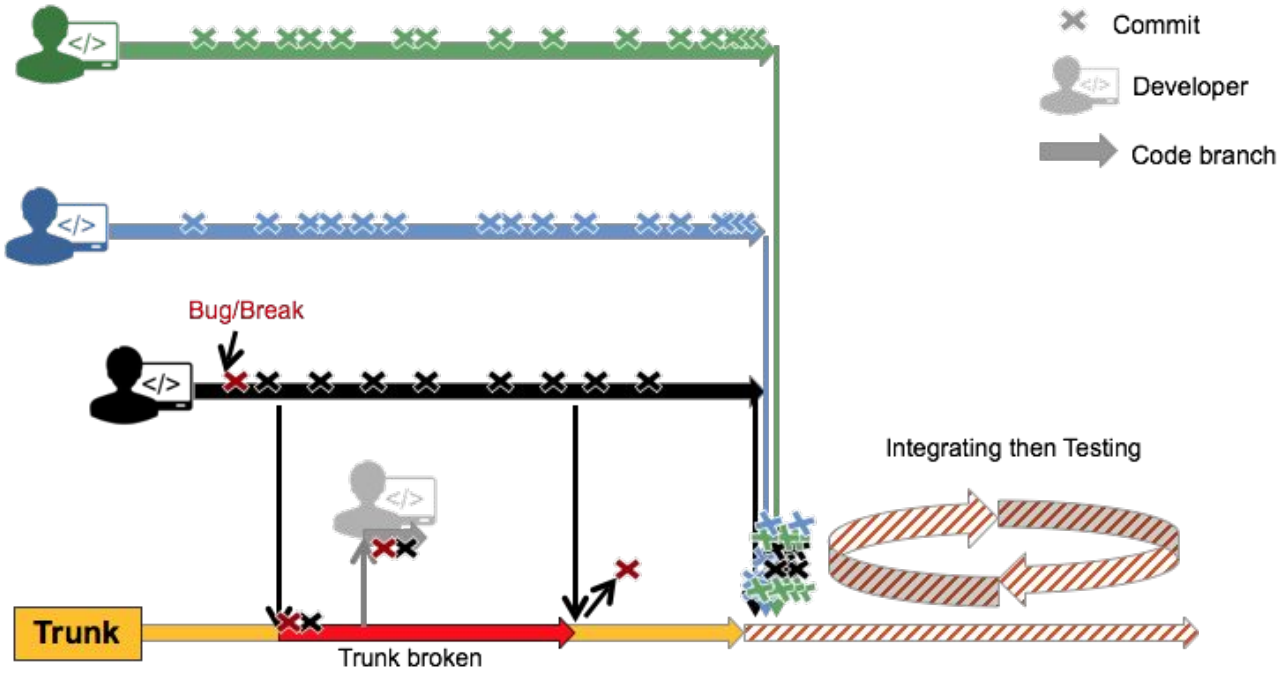
So why build software that way?

# And yet ..... In Software Engineering

Actually, we did (and many teams still do) build software with the process we outlined for the concept car.

When we follow this process and it goes wrong we call it **integration hell**

# Integration Hell



# Integration Hell

**Integration Hell:** painful stage in the software development cycle when teams attempt to integrate different units of code that have been designed and built independently. Make take hours, days, weeks (or even imperil the project) to make code compile, tests pass and system to work. Integration hell is likely to uncover latent design assumptions of independent units. Integration hell grows exponentially more difficult as a function of the complexity of the code/architecture and time since last integration.

# Searching for a solution .....

| Integration Hell Process | What if we aimed for? |
| --- | --- |
| Elongated feedback | Quick feedback |
| Broken trunk | Trunk must work |
| Infrequent communication | Frequent communication |
| Delayed Integration | *Continually* ingrate |

**Continuous Integration:** CI is a ***practice**** composed of

1. **Process:** Whereby developers
   a. Commit & integrate code into the common repository at least daily
   b. Each commit is independently built and tested (different machine)
   c. Build failures are visible to the team

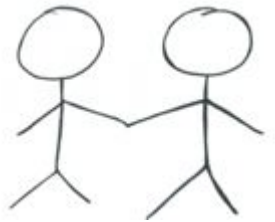
2. **Culture:** Where developers *value/follow* the process, write 'good' tests and fix problems before writing new code

3. **Tools:** Supporting and enabling technologies including SCM and 'continuous integration servers'

* some definitions focus on process or tool. While partially correct, this is only part of the problem. Therefore, I want our definition to explicitly cover culture, process and tools
We will also find Culture > Process > Tools. The dominance of culture over process and tools (although shocking) will repeat. When a culture chooses to fix a problem they find a tool.

# I need two volunteers with good cardiovascular health

# C.I. LARP

| Culture | Process | Tools |
| --- | --- | --- |
|  |  |  |
|  |  |  |

http://www.jamesshore.com/Blog/Continuous-Integration-on-a-Dollar-a-Day.html

# CI Process

1. Run the build/test script locally and make sure it passes 100%.
2. Get the rubber chicken from its resting place. If it's not there, find the person who has it and annoy them for it
3. Get the latest code from the repository
    a. Run the build script again just in case.
        i. If it doesn't pass, you know that there's some integration problem with the code you just got.
            1. *^%*@, put the chicken back, get the person who last checked-in to help you
            2. Start over when you're ready.
4. Check in your code
5. Walk over to the crappy build computer
    a. get the latest code from the repository
    b. run the build script again
    c. If it doesn't pass, revert (undo) your check-in.

       You messed up: didn’t add a file, forgot a local dependency, environment variable, anything

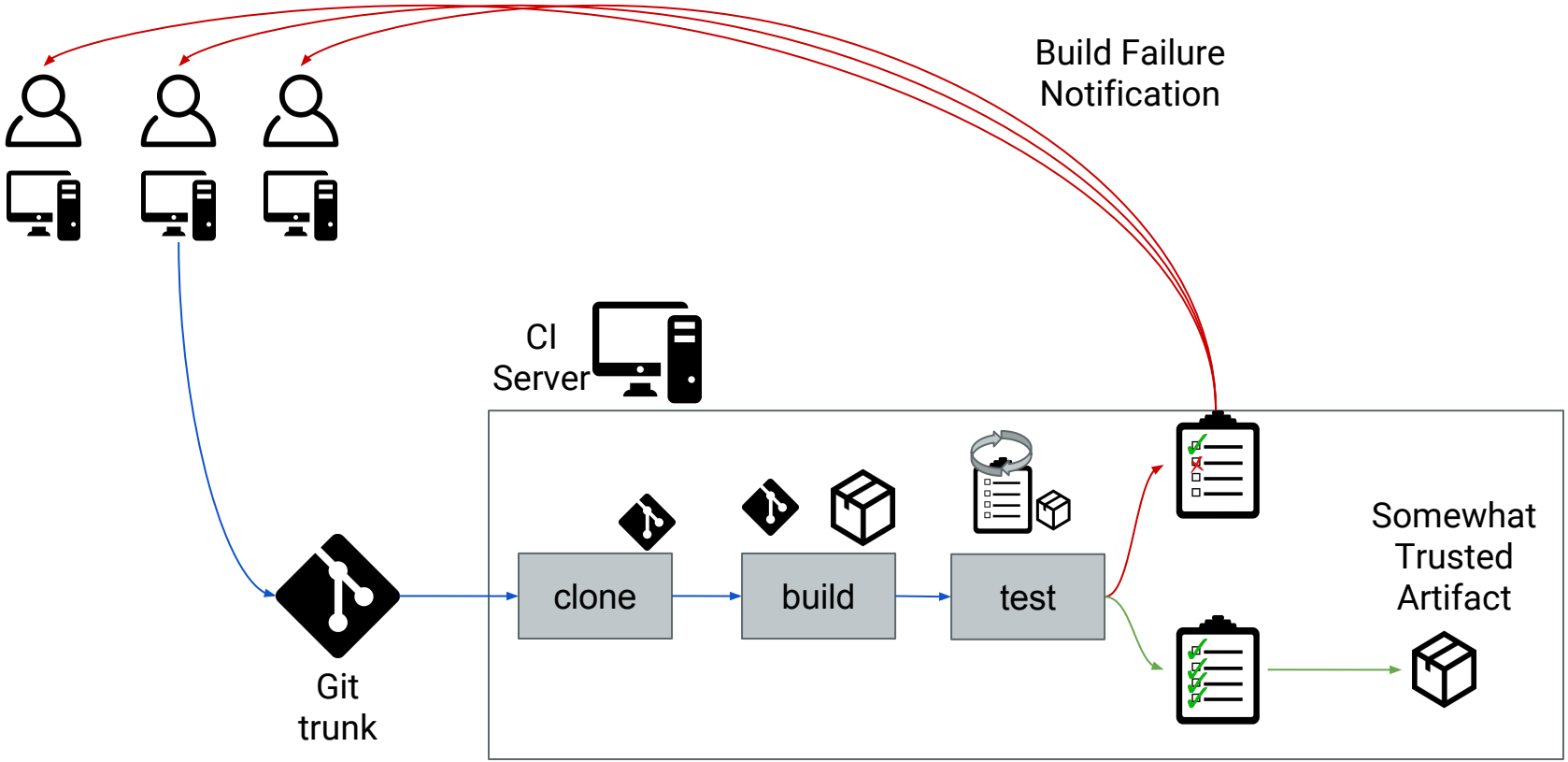
       Fix the problem on your computer and try again (hang onto chicken unless required)
6. Ring the bell. (Everyone else, that's your cue to applaud, or otherwise rejoice. "Yaaay.")
7. Put the chicken back. You're done.

Adapted from http://www.jamesshore.com/Blog/Continuous-Integration-on-a-Dollar-a-Day.html

1. Ok - got the process
2. I *assume* the rubber chicken isn't used in industry?
3. How is this actually implemented?

Not shown. CI Server has status page listing all alerts

# CI In Industry




(Jenkins, Hudson, CodeBuild/CodePipeline, TeamCity)

# In Practice: As a Software Engineer

Development

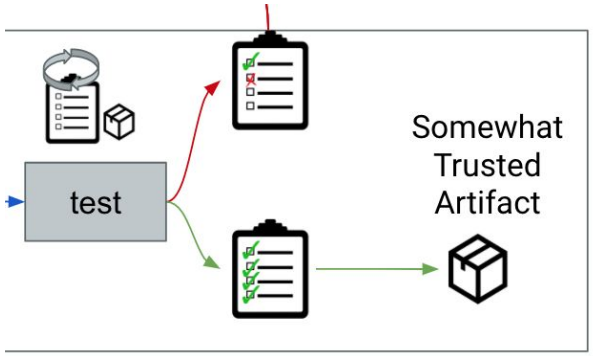
1. Clone/update code from repository
2. Run tests (were you given a working copy?)
3. Develop locally
4. Run tests
5. Take update (get other people changes committed between #1 and now)
6. Run tests

```
IF pass(tests)
    Commit();
ELSE
    Fix Code();
    Goto 4;
```

Broken build interrupt:

1. Work out what went wrong and fix (following above)
2. Return to development

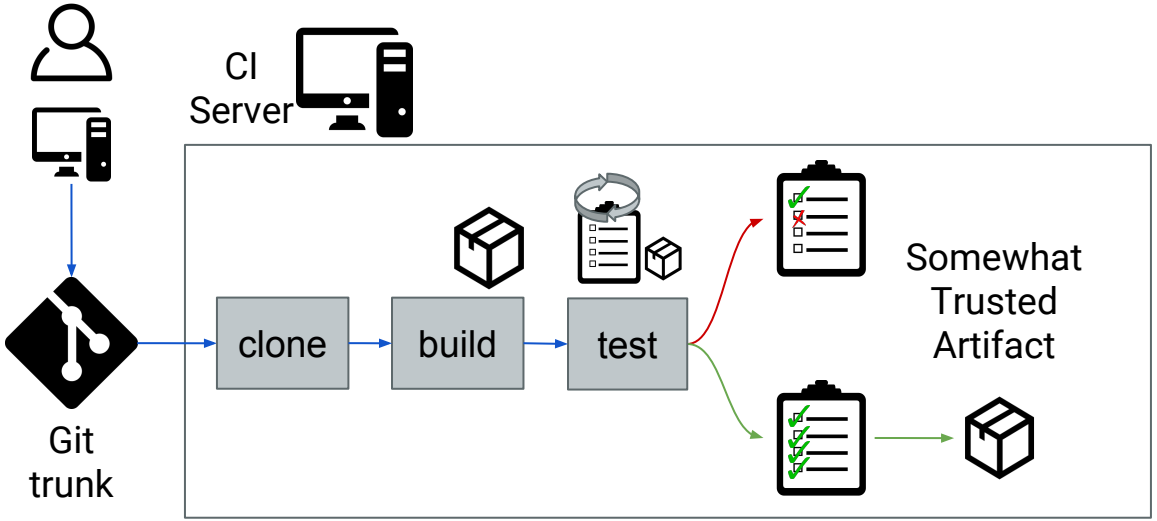
# Relationship: CI and Test Suite Quality




1. Why “somewhat trusted”?
2. What is relationship between CI and test suite quality?
3. Are you willing to deploy that artifact to production?

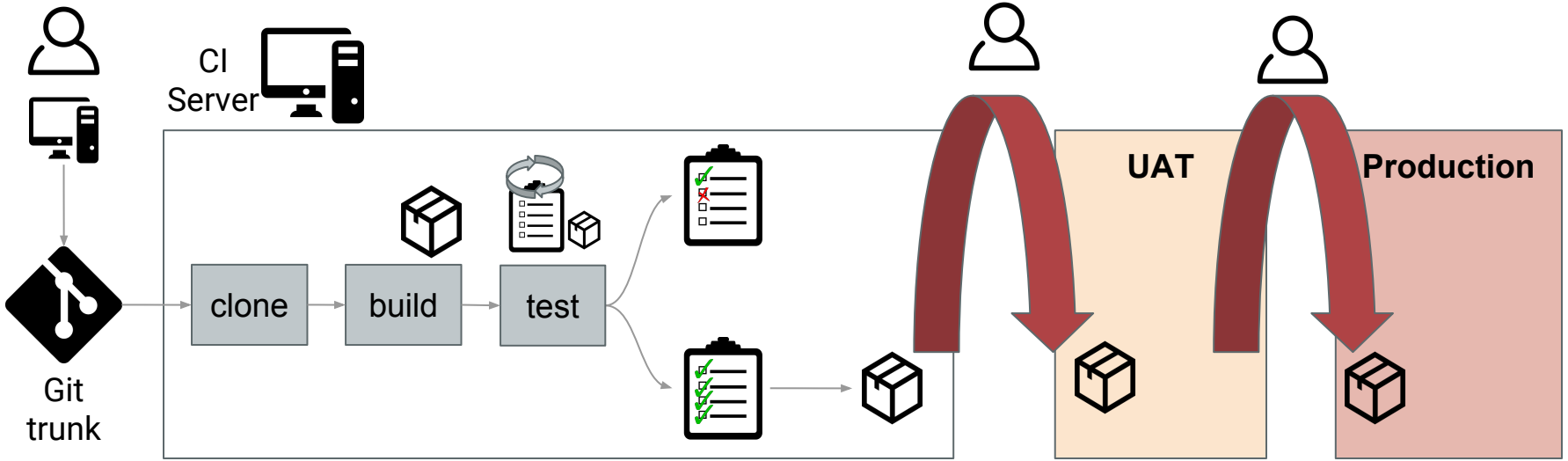
# We know what CI is what is CD?

- CI is great and typically solves a set of problems
- However, software is still not in the hands of users .....

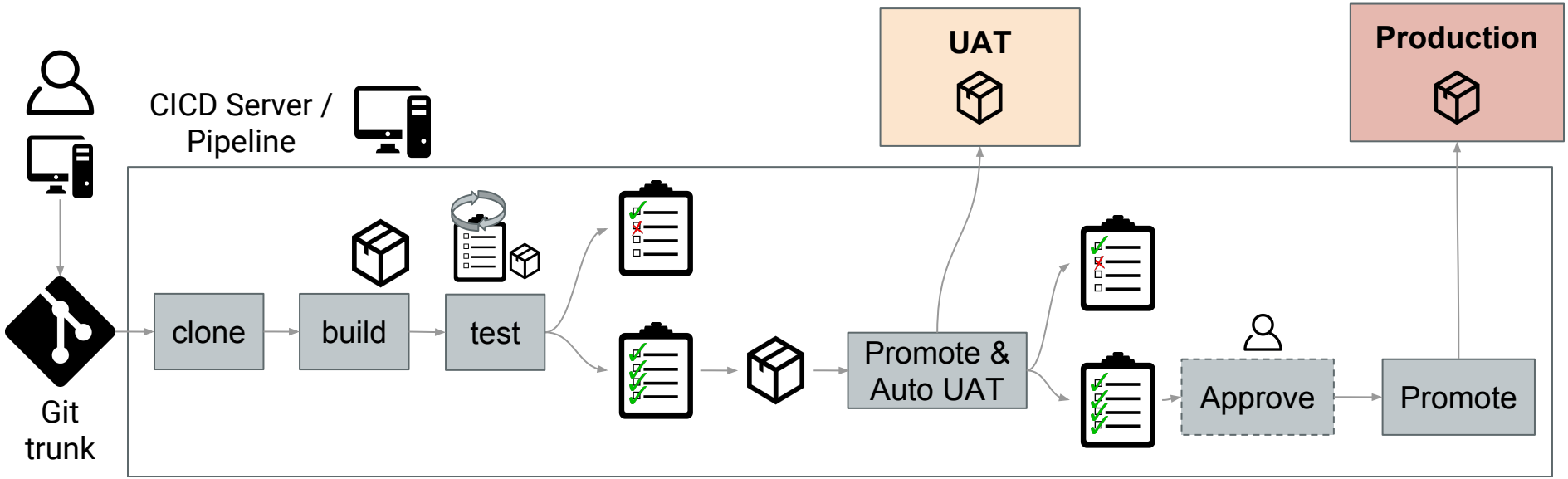



# Scope of CI

*After* CI the trusted artifact is manually promoted through environments (a bunch of copying and config fiddling)



# Continuous Delivery *extends CI*

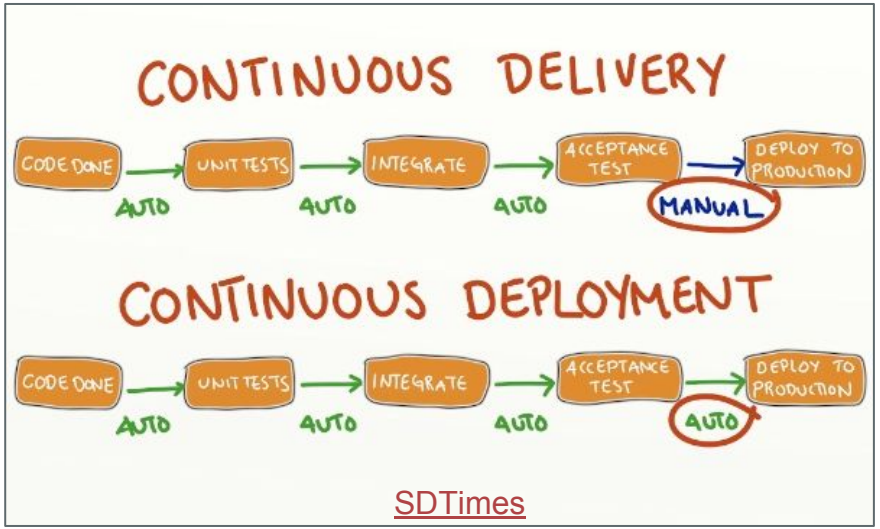


# Continuous *Delivery* vs *Deployment*

Delivery and Deployment are closely related.

**C Delivery:** Remains a manual ‘approval’ step to deploy to production

**C Deployment:** Every change that passes automated testing is pushed to prod



SDTimes

**Which one requires a higher quality test suite?**

# Project

1. In the last lecture we had a nice *local* development setup
2. I have provided an exemplar continuous ***deployment*** pipeline [based on ref arch]
    a. Teams can choose to use this as a starting point
    b. You will need to add stages going forward
    c. If you choose another toolset you need to establish basic CI
3. Demo
4. We have covered the *basics* of CD
    a. Have also not covered configuration management considerations
    b. Many more advanced techniques. (blue/green, canary, etc)

# Pop Quiz

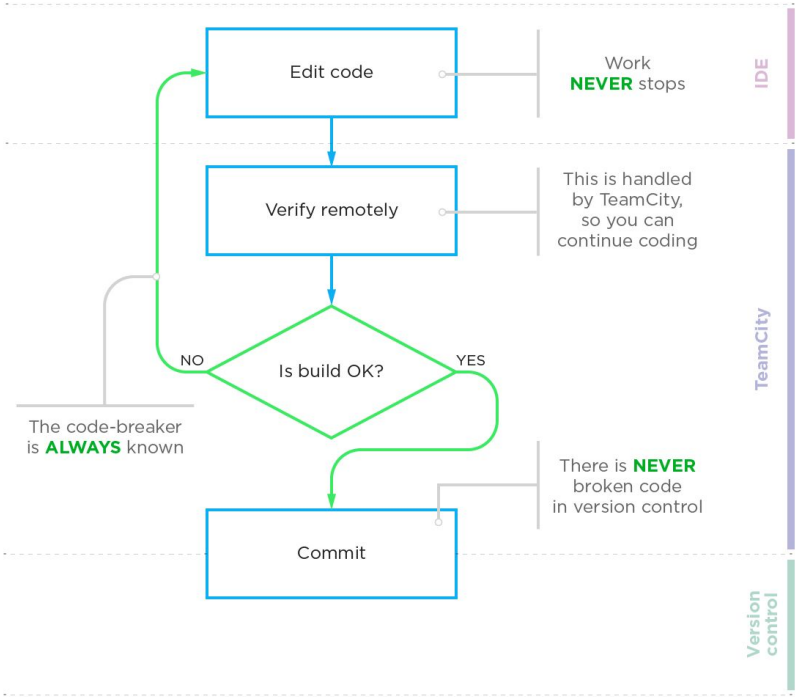
| Question | Answer |
| --- | --- |
| Continuous Integration is a <culture, process, tool support>? | |
| In Continuous Deployment the quality of the test suite doesn’t matter? | |
| In Continuous Delivery the quality of the test suite doesn’t matter? | |
| The difference between continuous delivery and deployment is? | |

# Reading

| Reading | Optionality |
| --- | --- |
| Jez Humble on CD or Fowler on CD | Required |
| Thoughtworks CI | Required |
| CD (understand the workflow diagram) | Test of Understanding |
| AWS CodePipeline | Required |
| CI @ Scale | Optional |
| First 15 minute, Advanced techniques reInvent CD | Optional |

# Advanced Considerations

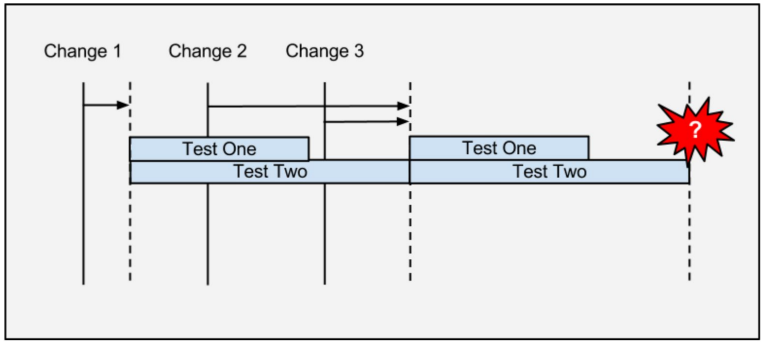
# CI In Industry (1)

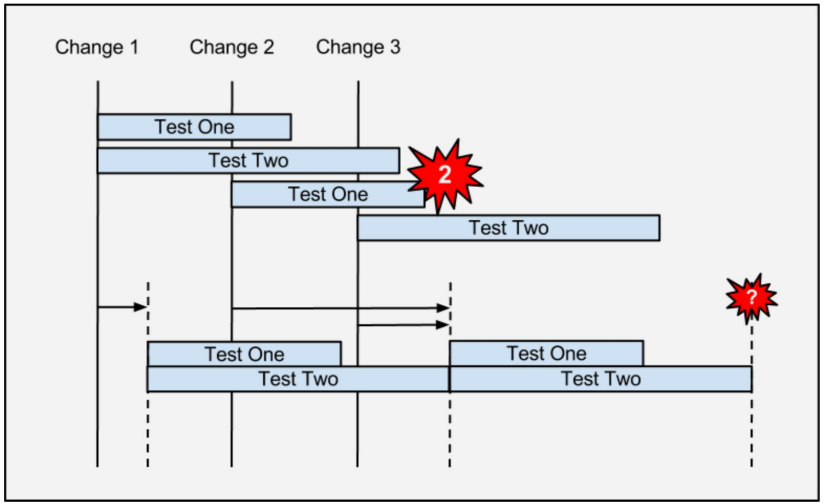



Jetbrains

- You may guess there is a race condition on two simultaneous commits
- Many products are set up to run tests *pre-commit* to handle the verification process and control race conditions of commits
- Often implemented with pre-commit hooks in git

# CI In Industry (2)

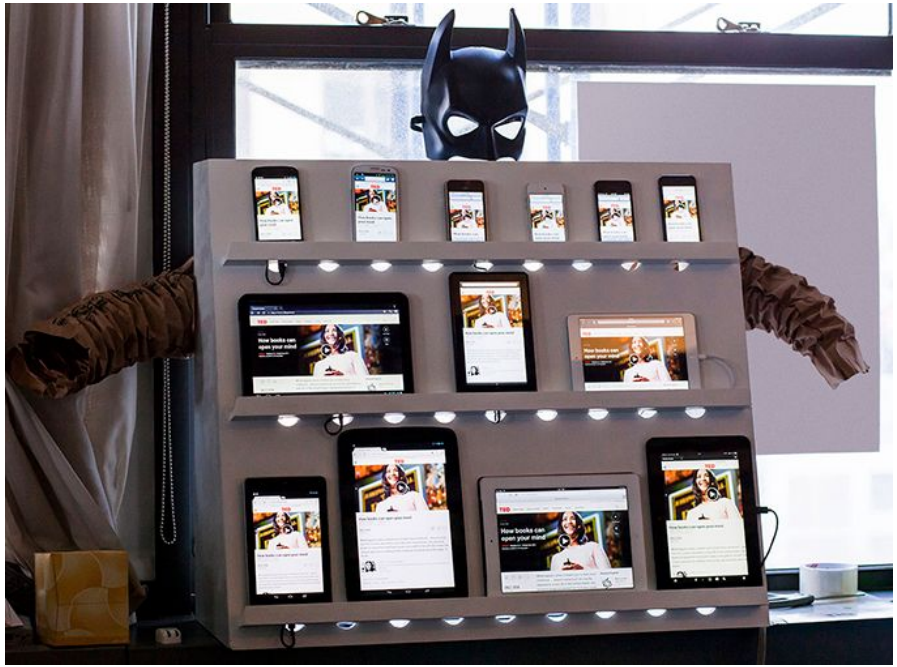



- Large number of commits and test duration > commit interval

http://eclipsecon.org/2013/sites/eclipsecon.org.2013/files/2013-03-24%20Continuous%20Integration%20at%20Google%20Scale.pdf

# CI In Industry (3)



- For those who say ‘my code is difficult to test’
- Testability is quality attribute
- Engineering skill and effort to build testable systems
- If people can CI hardware we can CI software

https://hello.ted.com/2013/11/05/holy-mobile-devices-batstand-how-we-test-pages-across-screen-sizes/