Software Design and Construction

159.251

Continuous Integration, Delivery and Deployment

Amjed Tahir

a.tahir@massey.ac.nz

Overview

- Software systems are evolving fast

  ➔ require more flexible methods to cope with such evolution.
- Agile method are designed to build software systems *iteratively* and *incrementally*.
- This require dividing large components into smaller, more manageable pieces that integrate together as the software development progress.
- **Challenges**:
  - How to integrate those components?
  - How to manage the integration process?

Overview

- Continuous Integration (CI), Continuous Testing (CT) and Continuous Delivery/Deployment (CD) all work together …

- CI , CT and CD goals are similar
  - Speed up the development process.
  - Increase visibility which enables greater communication.
  - And most importantly, improve the quality!

Testing

The need for validation

- This early stage of the product lifecycle is awkward - we need to be able to test our code, and check that it integrates, even though it's not running yet.
- Testing can help - we can unit test our classes as we build them. This can go some way to validating them.
- What about integration? We need to test that our components work together.
  - We can do this with unit tests combining the components.
  - We can also make sure our main class is running early, so we can run the programme and do manual integration tests.

Why write tests?

- validate the system (immediate feedback that system works as expected, prevent regressions later in development)
- increase code coverage
- enables refactoring
- documents the system’s behavior
- Test driven development
- Customer acceptance
- ping pong pair programming

Testing

Many types of testing to ensure that code works according to expectations

Broadly categorised into:

- Manual tests
- Automated tests

Manual testing

Tests performed by a person, can be expensive as it requires setting up environment, susceptible to human error (missing steps in test script, input errors or misreading output)

Automated testing

Computer executes pre-written test automatically. Important for adopting CI/CD

Types of tests

Unit tests

Tests individual methods/function in the code. Inexpensive to execute and easy to automate.

Integration tests

Verifies if different components of the system work together.

Functional tests

Checks implemented business functionality

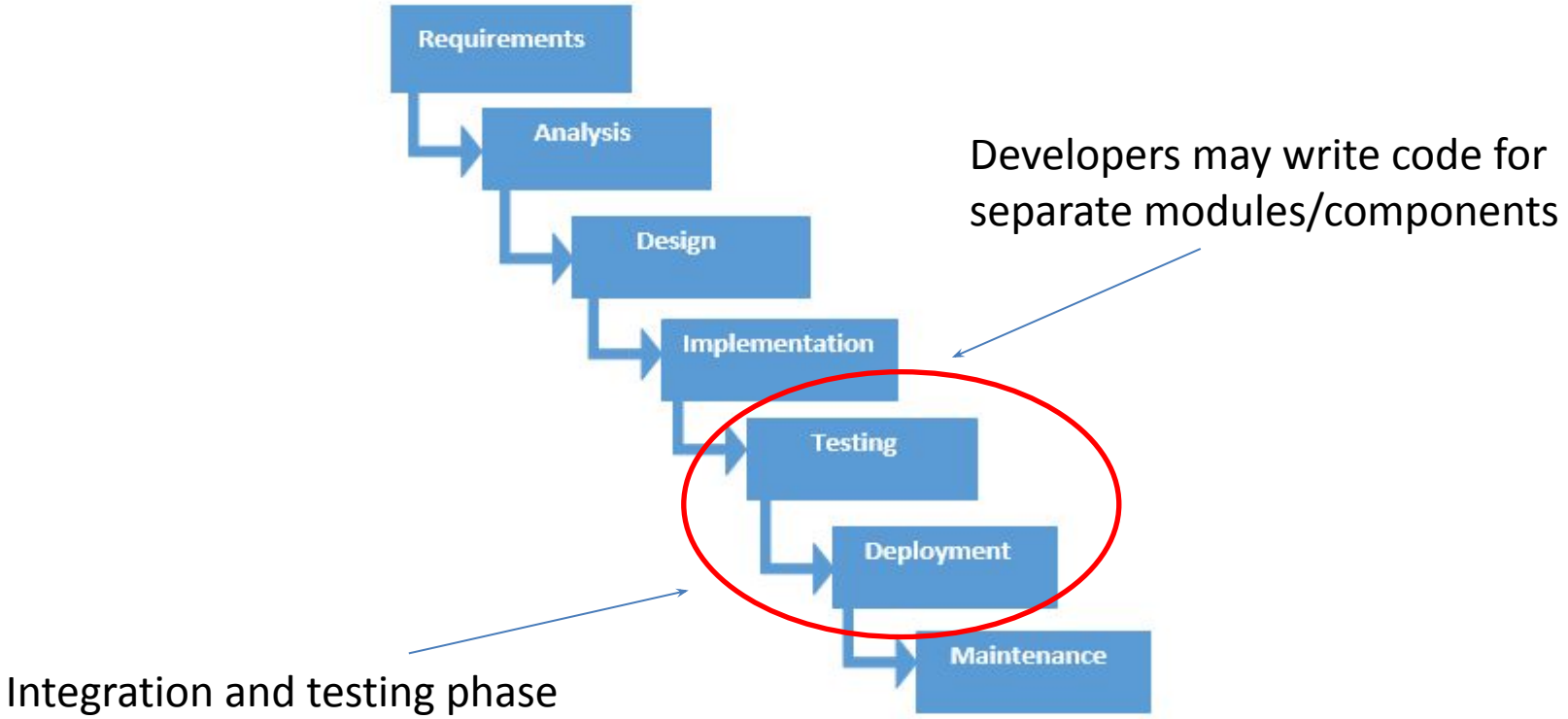
Continuous Testing

- First proposed as a way to reduce feedback time for developers via environment-triggered tests and traditional on-demand executed tests.
- Now referred to testing in all phases of the deployment pipeline (integration -> quality assurance (QA) -> stage -> production)
- Focus on business risks (what should immediately stop this release?)
- Reuse tests with environment-specific data

Continuous Integration - Testing's killer feature

Why do we need CT, CI and CD?

- Traditionally, integration, testing and deployment happen at the end of the development.



Traditional Integration

- All the features have to be completed before *Integration*, *Testing* and *Deployment* take place.
  - There are long periods between releases.
  - This can delay the process of finding and locating defects (late detected defects cost more to fix!).
  - Code review overhead – especially with large components.
  - Sometimes, poor control of the code when integrating large segments of the program (weeks/months of work).

Testing and Integration in Agile

- Agile values all forms of testing.
- Great focus on unit, integration and user acceptance testing.
- Agile depends highly on *Testing* and *Integration.*
- *Most Agile methods consider Unit Testing and User Acceptance Test as key practices.*

Continuous Development

remember this Scrum example

Continuous Integration (CI)

- *Continuous Integration* is the practice where members of a team integrate their work frequently leading to multiple integrations per release or per a specific timeframe.

Integrate -> build -> test

  - Integration can take place daily (or even in a shorter time frame).
  - Each integrated unit is verified by an automated build (including test) to detect possible integration errors/defects, as quickly as possible.
  - When tests fail, the developer's priority would be to fix the bug first!

Caution!!

"Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove"

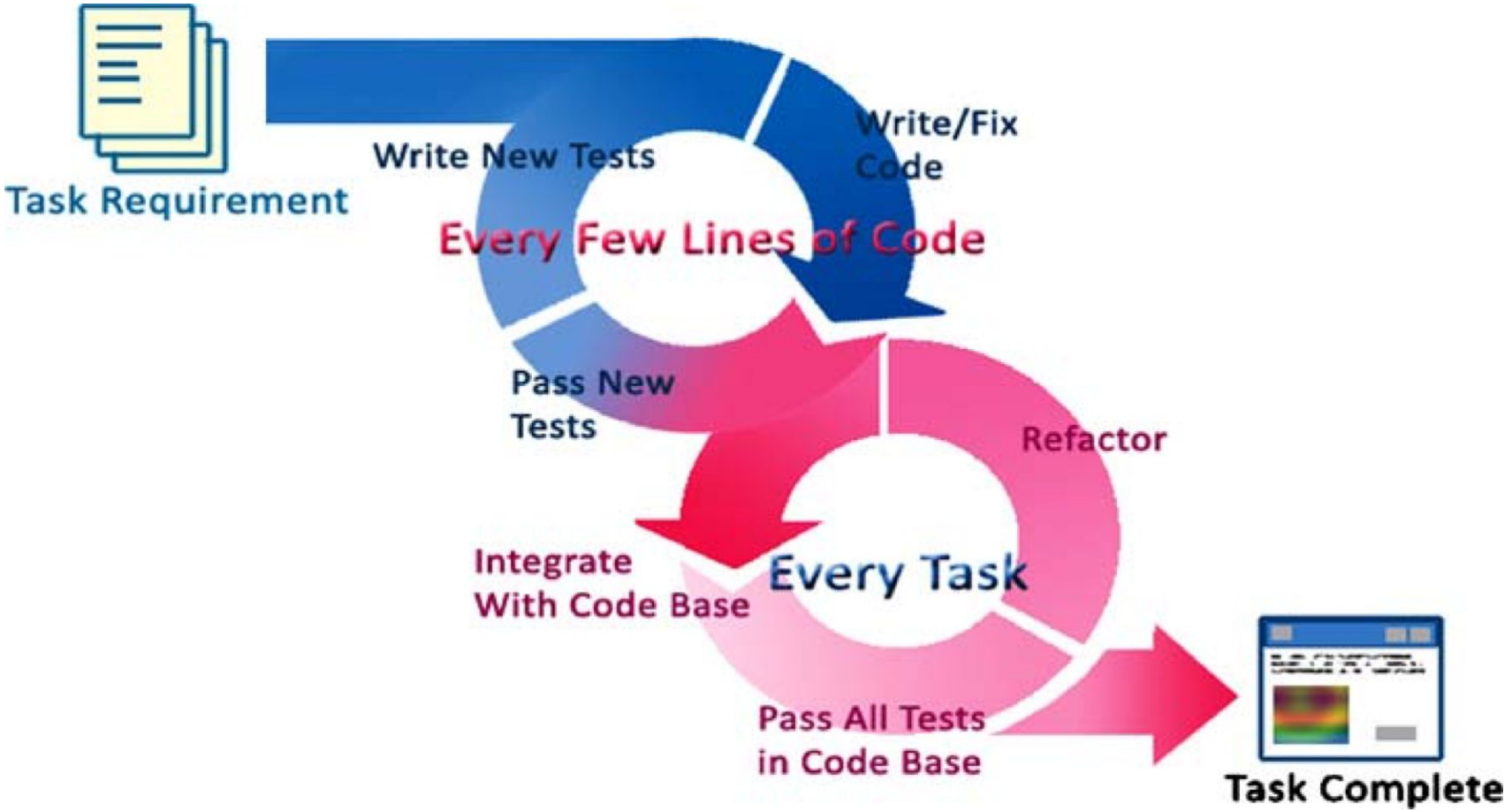
Martin Fowler

e.g. easier to fix bug closer to source than after release (or worse, after multiple releases. Which change caused the defect?)

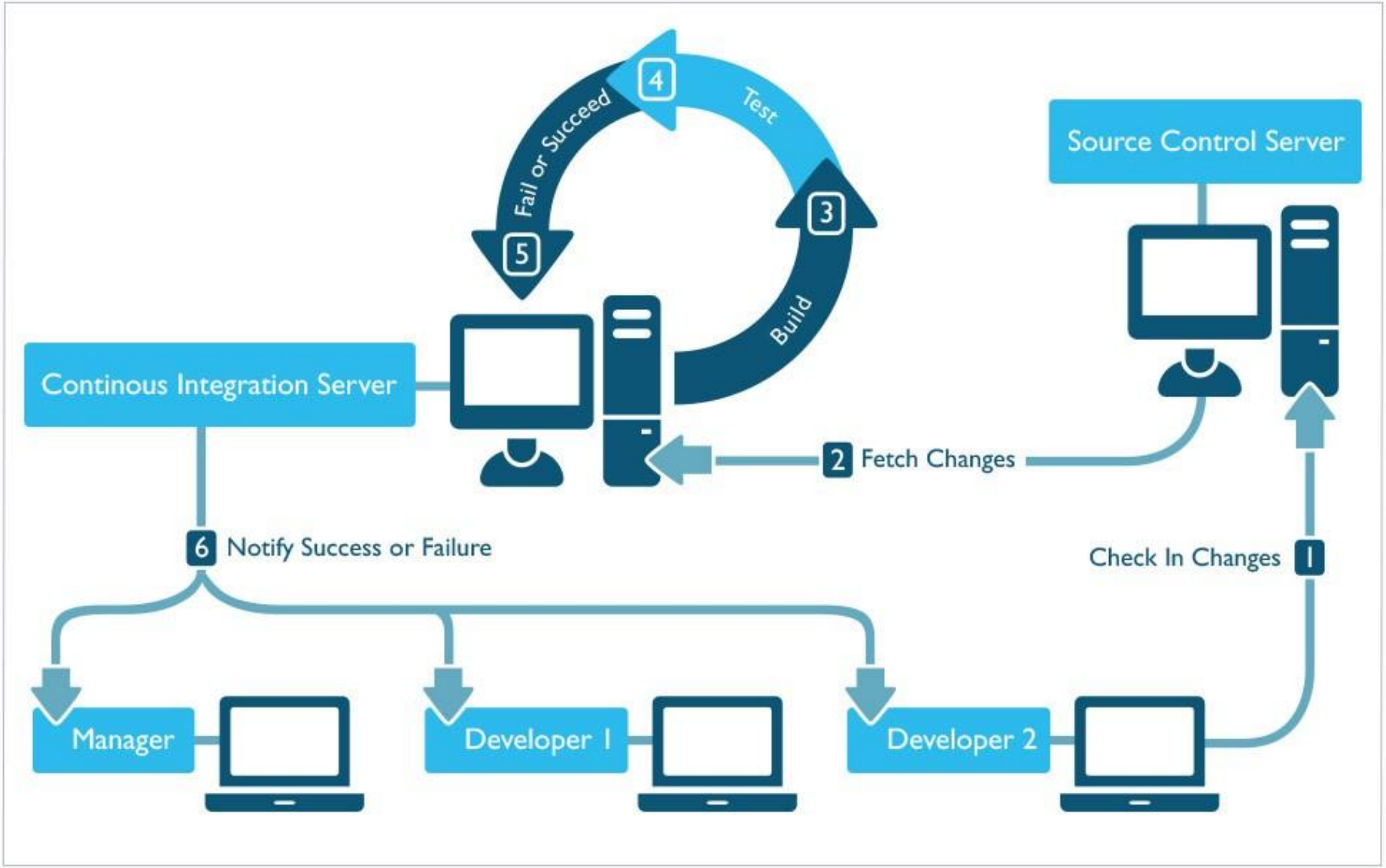
Integration workflow

- A typical continuous integration workflow will contain:
  - Setting up the automated build and unit testing.
  - Queuing the build and execute tests.
  - Configuring the whole workflow to execute on a continuous basis.

An example from Test Driven Development



Source: Nagappan et al. 2008



http://https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html

Testing makes CI robust

Continuous Integration without testing would be dangerous. New versions of a project are released regularly - if there is no way to verify that the changes are not buggy, this could compromise the stability of the project!

With testing as part of CI, any problems are immediately identified and can be fixed.

Use of git in CI

Continuous Integration makes sure that our programme can run at **all** times in the development process. Integration issues are sorted right at the beginning while they are still small issues.

Git branching can **delay** CI being run, which limits effectiveness.

- Keep branches small and short lived.
  - This allows code to go into the master branch sooner.
  - CI only runs when the master branch is updated
  - Using prolonged branching cuts down on the effectiveness of CI as it is longer between tests being run.

Continuous Deployment

Continuous Deployment (CD)

- *CD is the deployment or release of code to Production as soon as it is ready.*

- Follows the testing that happens during CI, and pushes changes to the production system.

- It aims at building, testing, and releasing software faster and more frequently which can reduce the time and cost of delivering changes by allowing for more incremental update to an application in production.

Continuous Deployment cont'd

- CD helps to make sure that a version of your code is accessible at all times.
- Any testing is done prior to merging to the base code and is performed on Production-like environments.
- CD requires CI - otherwise, you will get errors in the release.

Continuous Deployment Process

- CD Process = CI + deploying the code to production.
- Every change made to the code (by ALL developers) has to be deployed to production.
- This can result in several deployments per day.
- The problem is with business decision – not allowing the code to be deployed.

Benefits of CI and CD

- Frequent reviews and short feedback cycles.
- Frequent refactoring and updates to the code.
- Frequent testing in small chunks.
- Smooth integration process.
- Increase visibility which enables greater communication
- Catch bugs/defects faster.
- Spend less time debugging and more time adding features.
- Stop waiting to find out if your code’s going to work.
- Reduce integration problems allowing you to deliver software more rapidly.

All related

- You usually starts with single *test* – then you *integrate* separate units that have been individually tested and conduct an integration tests .. Then *deliver* and/or *deploy*.

- Checkout the source code -> write your code -> test -> build ➔ commit -> test -> build again! ➔ Test again! .

CT, CI and CD best practice

- '*Automation*' is the key!

- Configuration Management in place!
  - Maintain a code repository (all the things you have learned about version control)
- CI does not make sense without CT.
- Everyone should commit to the base code, everyday
  - or preferably, multiple times a day!
- Make it accessible - everyone can see the results of the latest build - use a tracking system!

As a developer……

- Developers play a central role in CI, CT and CD.
- Follow *best practice* to avoid problems.
- You need to make sure that
  - Check in the code as frequently as possible (long delays can be harmful!).
  - Do not check in broken or untested code.
  - Do not check in when the build is broken.

Tools that help the process

- Specific CI tools e.g. Travis, Jenkins and GH Actions.
- Version control tools e.g. Git or SVN.
- Issue (change) tracking systems e.g. JIRA or Bugzilla.
- Build tools e.g. Maven or Gradle.
- Unit testing e.g. XUnit framework and/or Selenium.

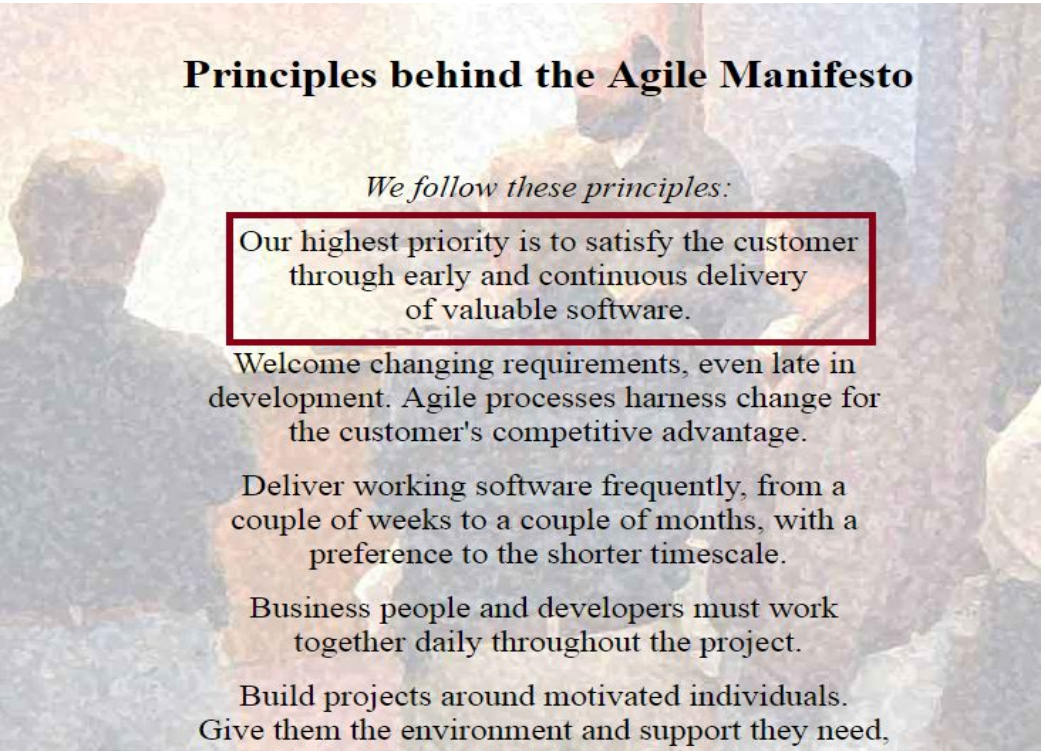
Moving towards Continuous Delivery

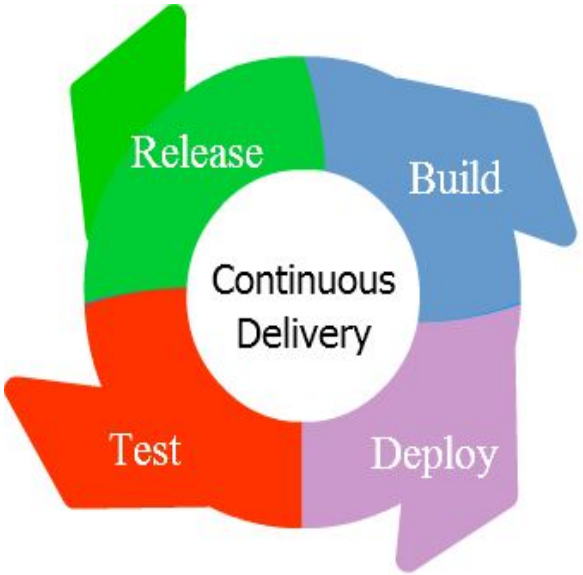
Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, *safely* and *quickly* in a *sustainable* way.

It means that you **can** do frequent deployments. Businesses may prefer a slower rate of deployment - this becomes a business decision rather than a technical one.

Continuous Delivery cont'd

- CD has been mentioned as the first principle in the Agile key principles!

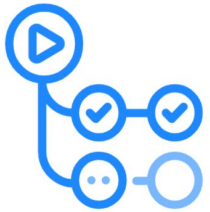




Example of CI tool

Standalone

Integrated with VC



GitHub Actions

How it works?

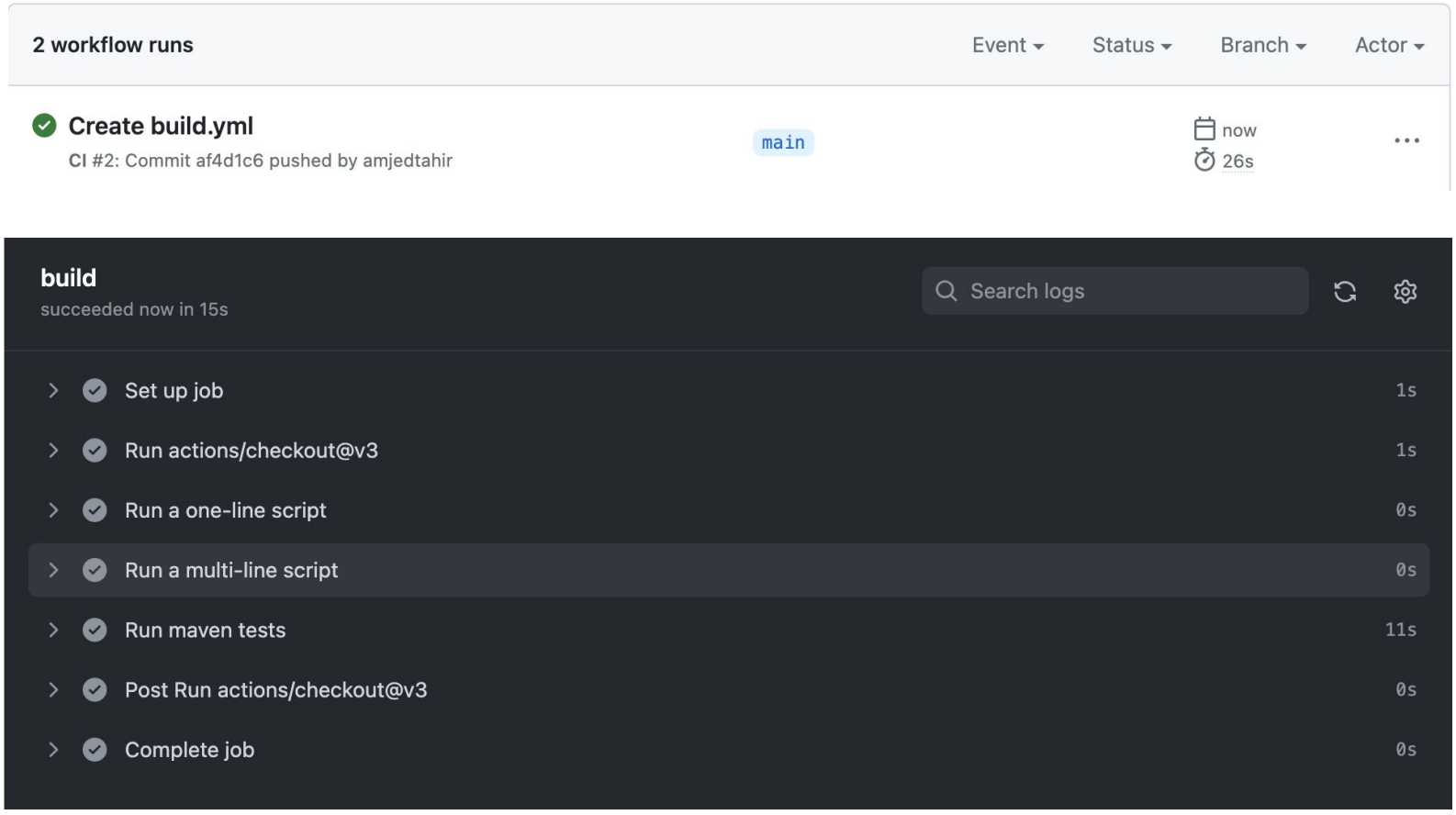
- All you need is a configuration file (e.g, .ymal) to set up the workflow
- For standalone application, you need to connect your CV with your CI tool (e.g., github $\longrightarrow$ Jenkins).

Example with GitHub Actions

Create a build in GH actions to run simple maven tasks

```
    # Run tests
        - name: Run maven tests
          run: mvn clean test
```

https://github.com/SE-Design-and-construction/taxcalculator


2 workflow runs
Event
Status
Branch
Actor
Create build.yml
CI #2: Commit af4d1c6 pushed by amjedtahir
main
now
26s
build
succeeded now in 15s
Search logs
Set up job 1s
Run actions/checkout@v3 1s
Run a one-line script 0s
Run a multi-line script 0s
Run maven tests 11s
Post Run actions/checkout@v3 0s
Complete job 0s

References and interesting reads

- Fowler, M . “Continuous Integration”. http://www.martinfowler.com/articles/continuousIntegration.html, Retrieved on 23/7/2019.

- Stolberg, S. "Enabling agile testing through continuous integration." *Agile Conference, 2009. AGILE'09.*. IEEE, 2009

- Richardson, J. “Continuous Integration: The Cornerstone of a Great Shop”. http://www.methodsandtools.com/archive/archive.php?id=42 Retrieved on 23/7/2019.

- Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education.