nagarro

Continuous integration

Exercise - Why Should We Integrate Continuously?

Build at least two groups, take a marker and some notes. Discuss and answer the following questions. Write your answers on the notes:

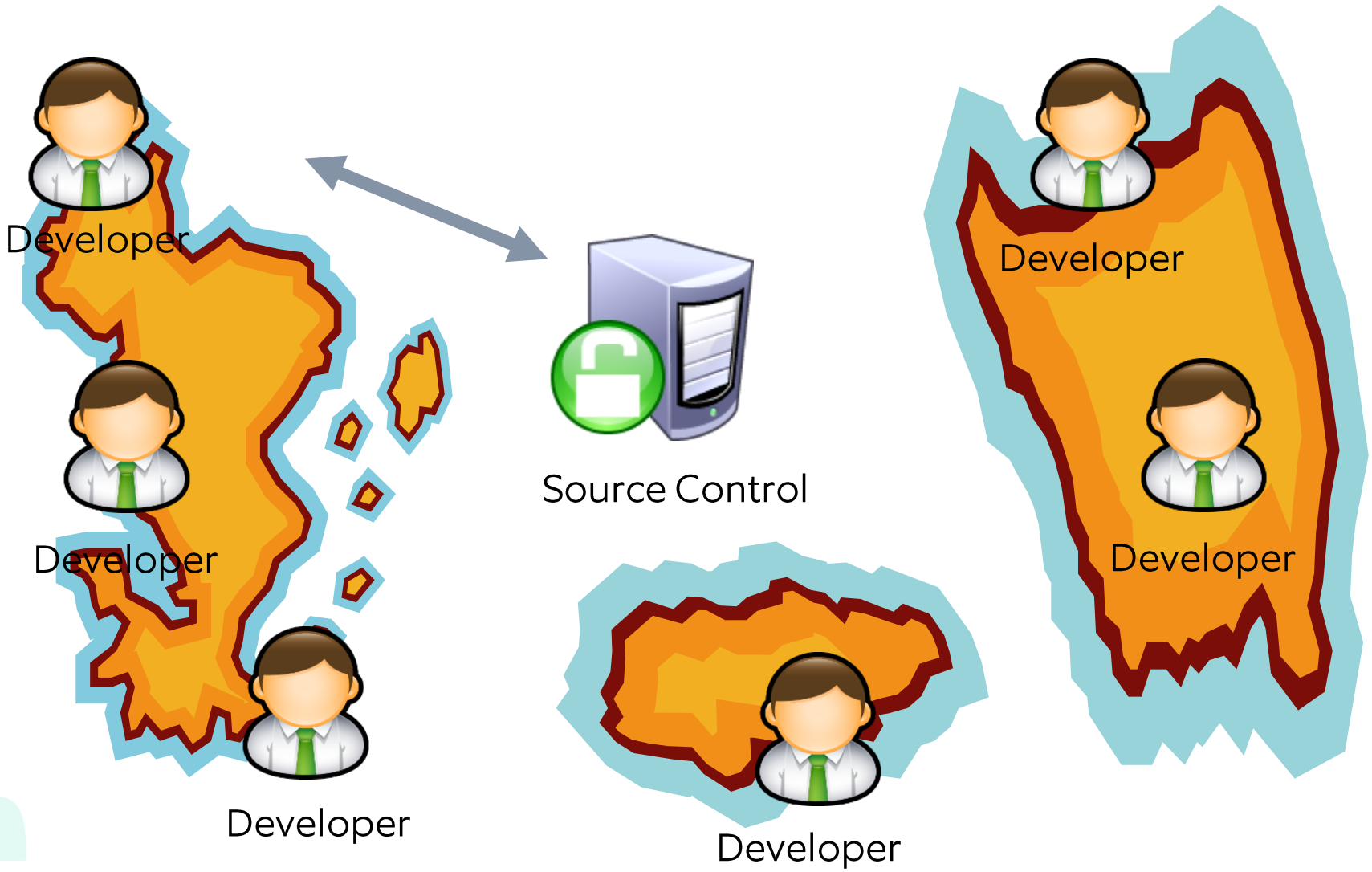
- Discuss about the beneficiaries of CI and their roles in the software development process. How many can you identify?
- What are the benefits of CI?

Remember the problems discussed on the previous slides.

Gather in a circle in front of the board. Each team sticks their notes on the board and presents their findings. Discuss about your most important insights. The trainer adds missing items.

Too Many Cooks Spoil the Broth!

Too Many Developers Spoil the Build?



Myth or Fact?

“It is better to integrate code at planned milestones to resolve potential integration problems at once.”

Integration Hell

Definition of continuous integration

C2 Wiki about Continuous integration

Development teams use Code Ownership to minimize conflicts among people editing. The longer engineers hold on to modules, the more important it is to minimize conflicts.

What if engineers didn't hold on to modules for more than a moment?

What if they made their (correct) change, and presto! everyone's computer instantly had that version of the module?

You wouldn't ever have Integration Hell, because the system would always be integrated. You wouldn't need Code Ownership, because there wouldn't be any conflicts to worry about.

Continuous integration

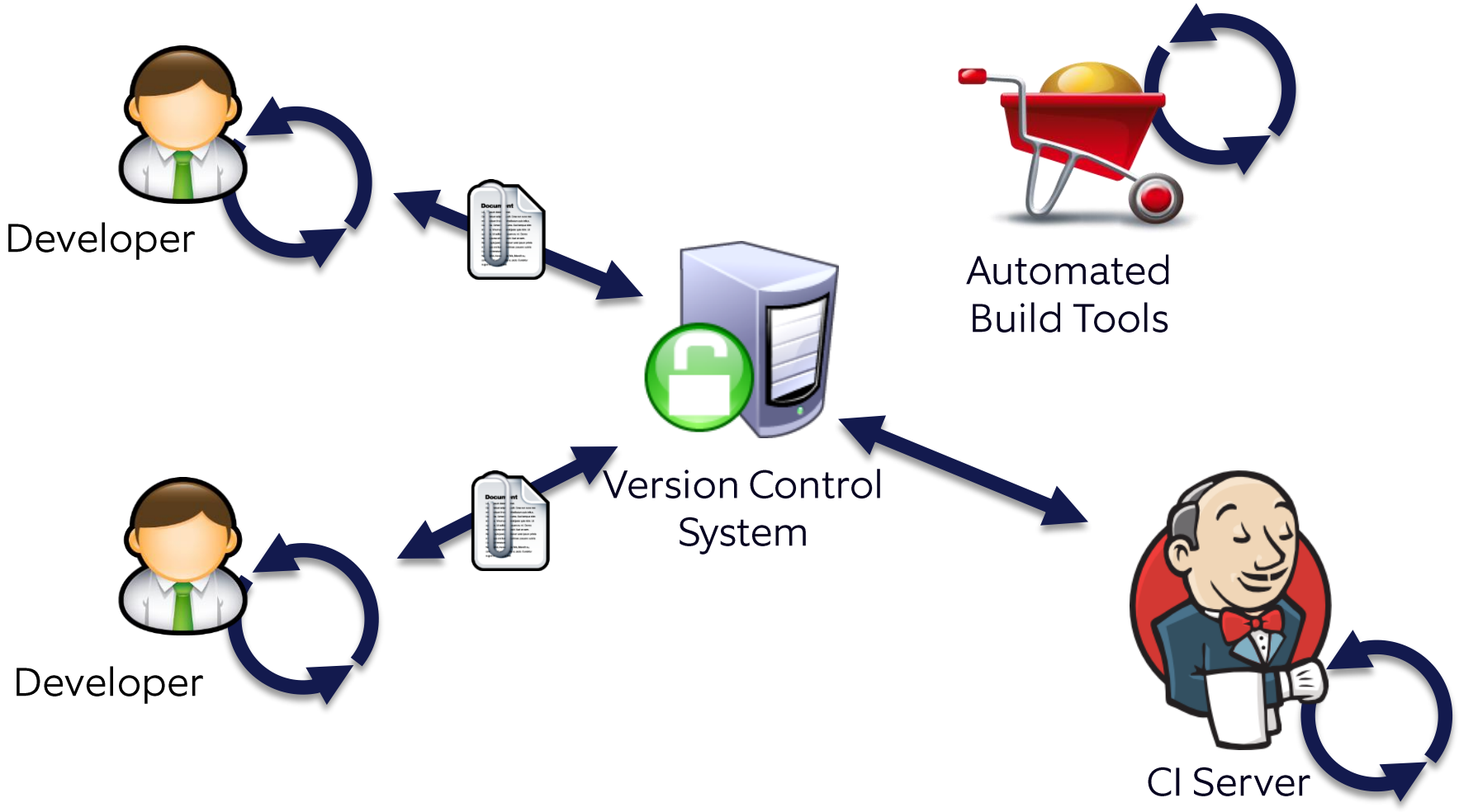
Overview

Continuous integration is a **software development practice** where members of a team **integrate their work frequently**, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is **verified by an automated build** (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

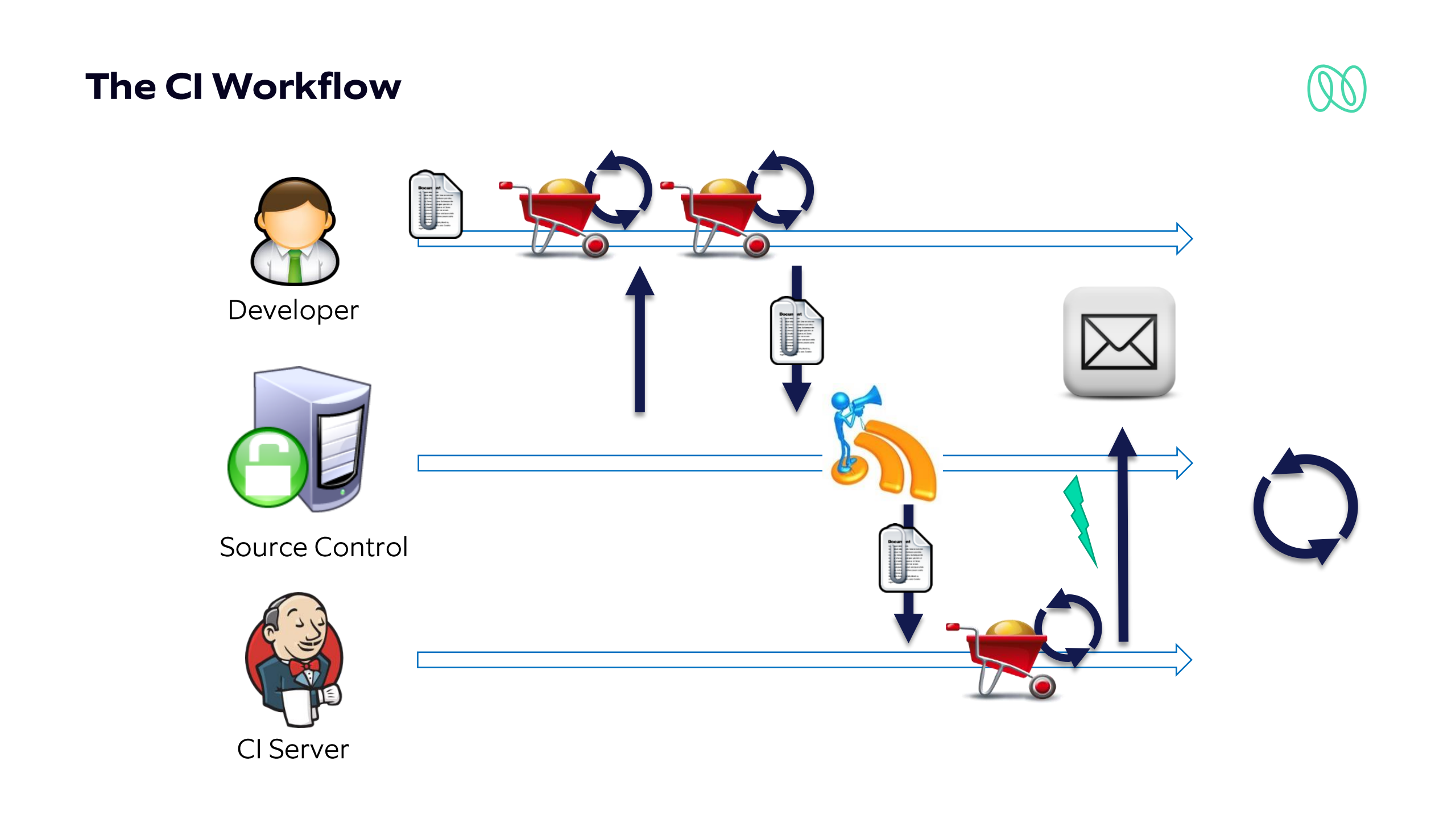
- Martin Fowler

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

Elements of CI



The CI Workflow



Automated Build Tools

1. Avoids „Works on My Machine“
2. Reducing the manual effort
   - Automation wastes less time
3. Not an IDE Build
4. Common Build-Tools:
   - Ant, Maven, Gradle
   - MSBuild

Anything Else?

- Code-Quality checks
- Archiving and
  Release Management
- Test Automation
- Continous Delivery and Deployment

Continuous integration

Principles & Practices

CONTINUOUS INTEGRATION

- Have a **single place** where all the code lives
- Everyone commits to the mainline **every day**
- **Automate** the build process
  - Fix broken build immediately
  - Make and keep the build fast
- Every commit **triggers a build**
- Automate the **testing** process
- Everyone has **access** to the latest result
- Everyone can **see** everything

Continuous integration

Benefits

CONTINUOUS INTEGRATION

- Integration takes **less effort**
- Issues will come up **earlier**
- Automation means **less issues**
- The process is **more visible**
- Improved **team communication**
- Short integration iterations means **more flexibility**
- The code is **ready to be delivered** more often

Continuous integration

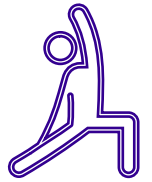
What CI can thus accomplish is

CONTINUOUS INTEGRATION

Higher quality

Faster delivery

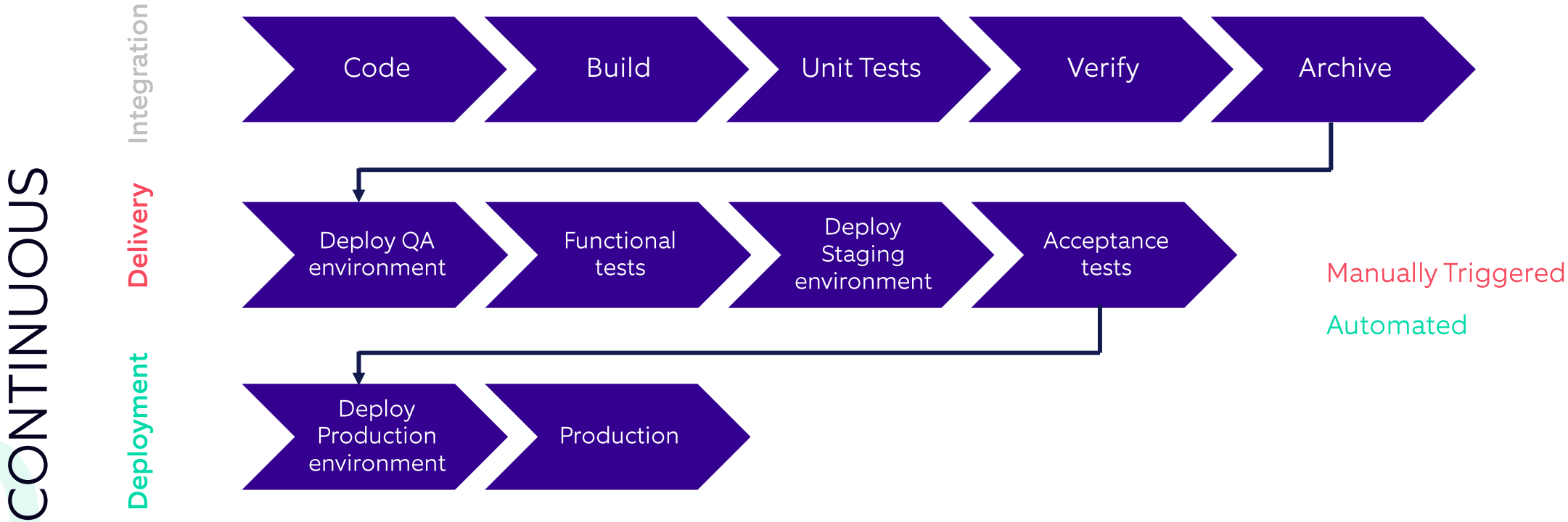
Lower costs



More flexibility

Continuous ...

Integration vs. Delivery vs. Deployment



- Continuous delivery
  - Software **can** be deployed to production at any time

- Continuous deployment
  - Software **is automatically** deployed to production all the time

Continuous delivery

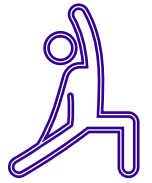
What CD can thus accomplish is

CONTINUOUS DELIVERY

Higher quality

Faster delivery

Lower costs



More flexibility

Definition of continuous delivery

Continuous delivery

Overview

Continuous delivery is a **software development discipline** where you build software in such a way that the software can be **released to production at any time**.

- Martin Fowler

https://martinfowler.com/bliki/ContinuousDelivery.html

Distributed vs. Centralized Version Control Systems

1. Centralized (client/server) systems
   - E.g. CVS, SVN
   - Basically all actions are performed on the server
   - Requires a server to function

2. Distributed systems
   - E.g. Git, Mercurial
   - Basically all actions are performed locally
   - Can work with remote repositories, but do NOT require a remote server to function

3. Discussion: Pros and Cons of Git vs. SVN?

Git vs. SVN

GIT	SVN
(-) Steep learning curve	(+) Smoother learning curve
(-) More complex set of commands (different commands to achieve the same result)	(+) Less complex set of commands
(-) GUI based tools not mature as SVN	(+) Lots of mature GUI based tools
(-) Not suited for large files (2GB limit)	(+) File size only limited by file system
(+) Distributed nature allows for different replication approaches (master-master, master-slave, peer-to-peer)	(-) Centralized nature allows for mirroring-only replication appraoch
(+) Local actions allow for faster interaction	(-) Remote actions slow down process
(+) Local history manipulation allows rollbacks	(-) Remote commits-only can lead to rare commit frequency
(+) First class branches	(-) Branches are just separate folders
...	...