Agile Software Development Practices

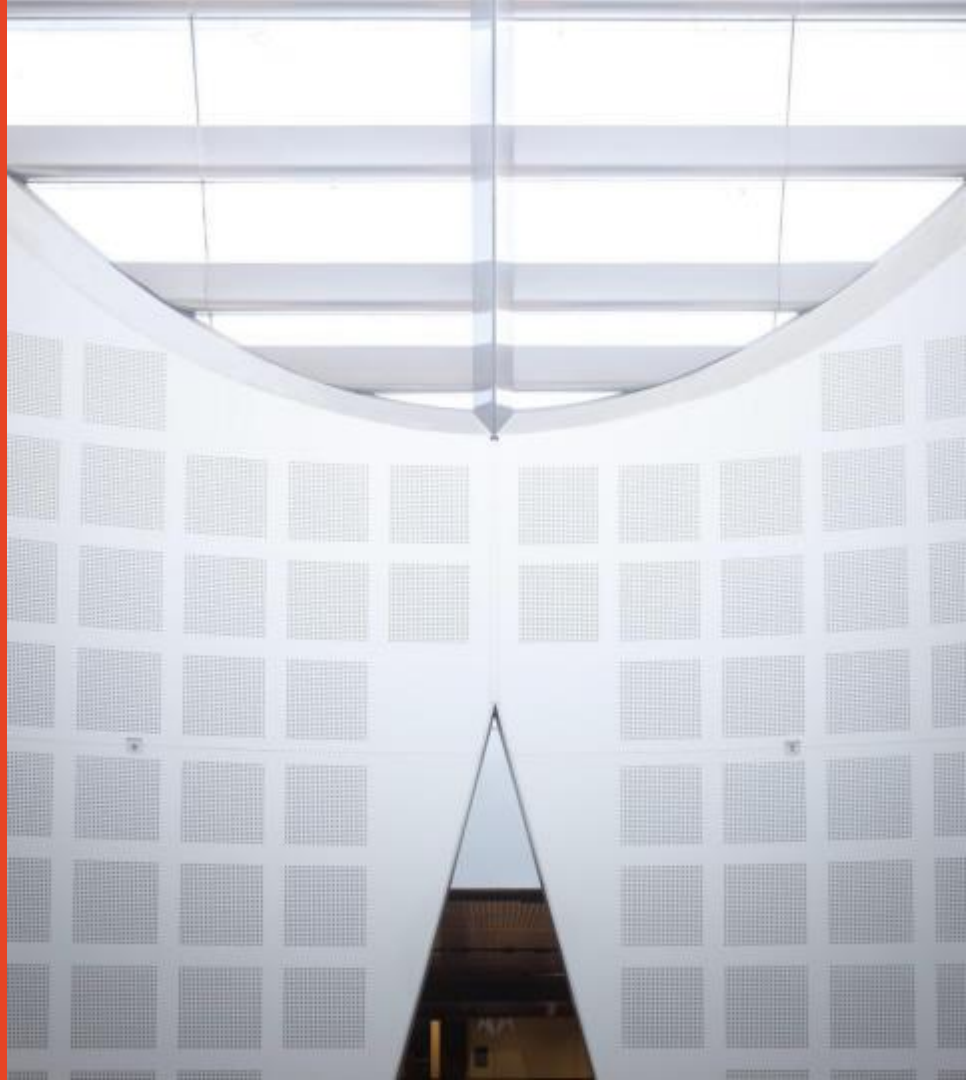
SOFT2412 / COMP9412

Continuous Integration, Continuous Delivery and Deployment

Xinyi Sheng

School of Computer Science

Slides based on Dr. Basem Suleiman and Dr. Farshid Hajati

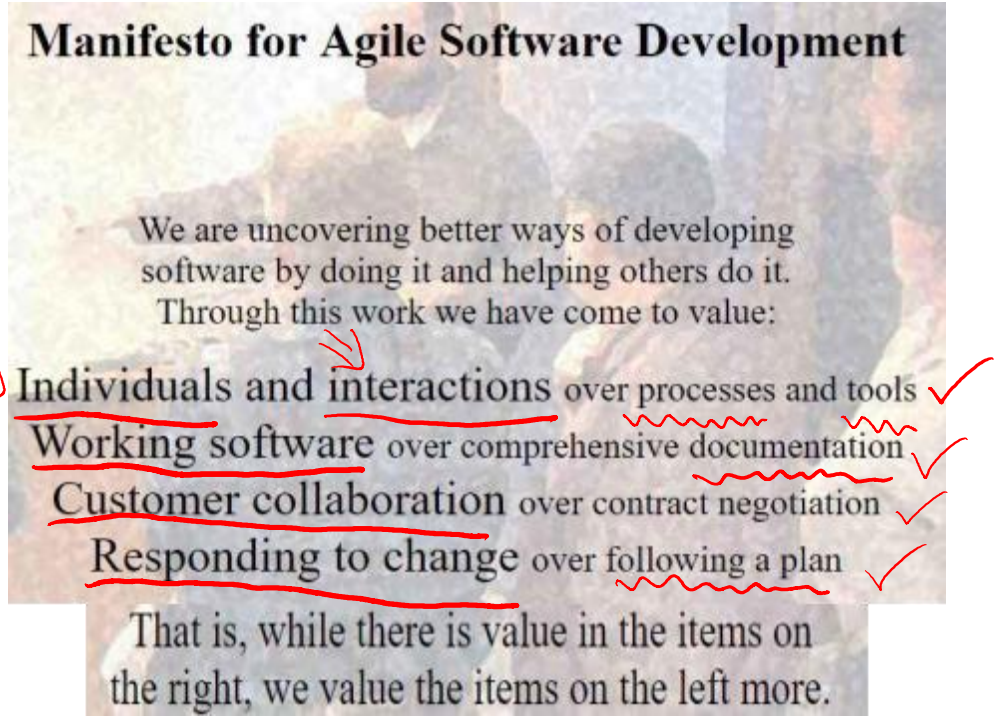


Agenda

- Agile Development
- Continuous Integration (CI)
- Effective CI
  - Automation practices
  - Team practices
- Continuous Delivery
- Continuous Deployment
- Tooling - Jenkins

Agile Development

Agile Manifesto (2001) - Revisit



Agile Manifesto: http://agilemanifesto.org/

Agile Principles - Revisit

- Our highest priority is to **satisfy the customer** through early and **continuous delivery of valuable software**.

- **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.

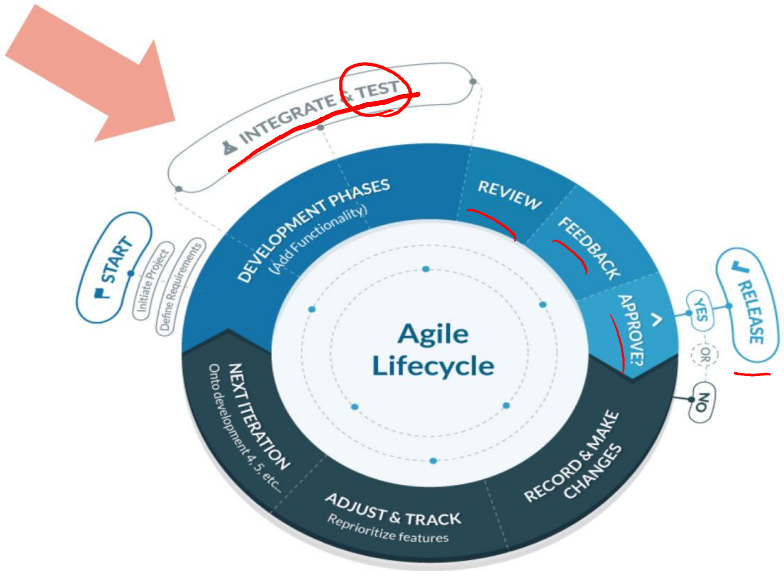
- **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the **shorter timescale**

- **Working software** is the primary **measure of progress**

Agile Alliance: http://www.agilealliance.org

Agile Development - Integration

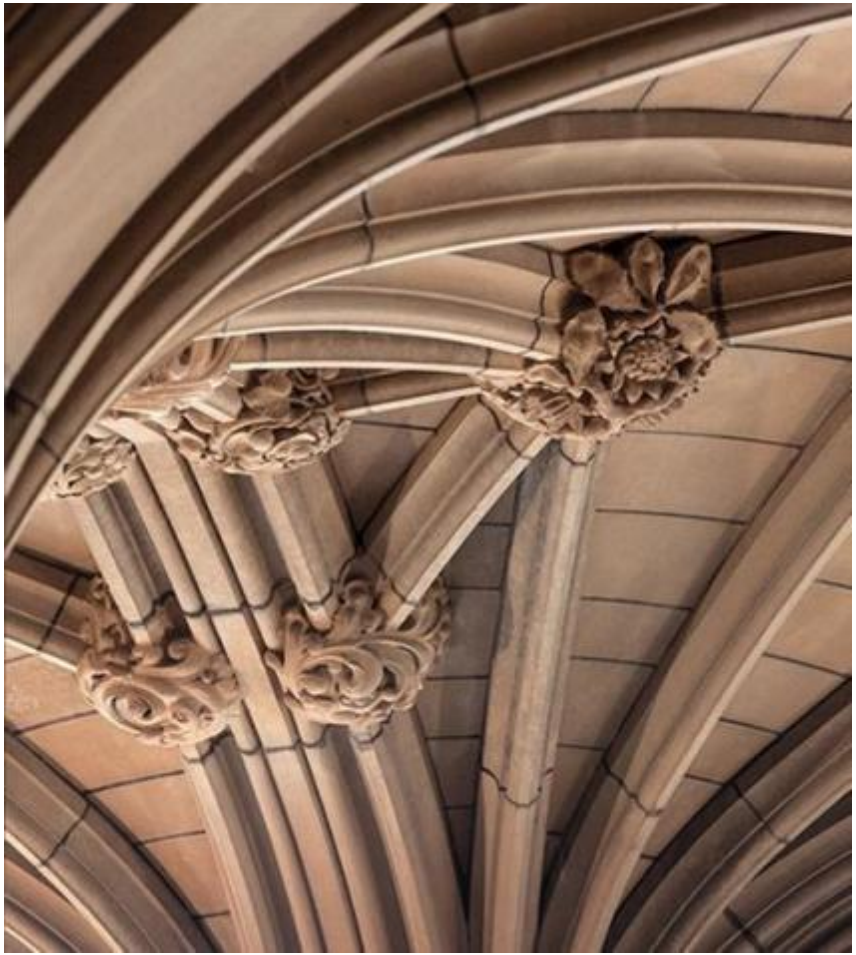
– Integration is a regular activity in agile development



Adding new features
fixing some bugs

https://blog.capterra.com/wp-content/uploads/2016/01/agile-methodology-720x617.png

What is Continuous Integration?



THE UNIVERSITY OF SYDNEY

Continuous Integration (CI)

*"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**" – Martin Fowler, 2006*

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

Continuous Integration (CI)

- Suppose that two developers working on two separate components; A and B. Each thinks they complete their work; code of both components need to be integrated and verified to deliver consistent & expected behavior of the system

different levels of integration

- Individual developers may work on components integrated into subsystems

- Sub-systems may be integrated with other team's subsystems for form a larger system

https://www.agilealliance.org/glossary/continuous-integration/

Continuous Integration (CI)

– The goal is to ensure that the software is always in a working state

- Every time somebody commits any change, the entire application is built, and a comprehensive set of automated tests is run against it

- If the build or test process fails, the development team stops whatever they are doing and fixes the problem immediately

https://www.agilealliance.org/glossary/continuous-integration/

Continuous Integration - Objectives

– Minimize the duration and effort required by each integration

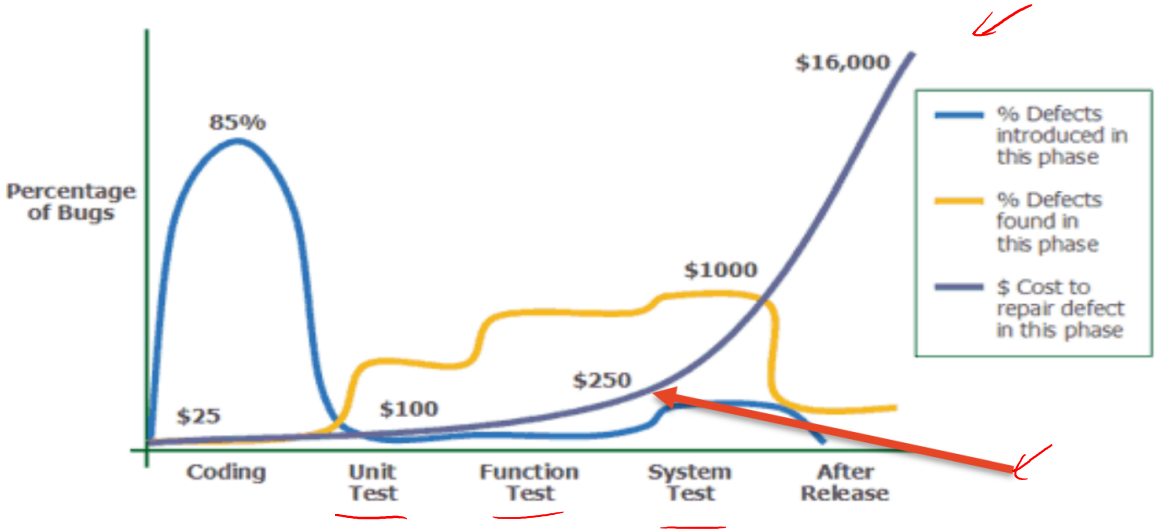
– Be able to deliver product version suitable for release at any moment

– To achieve these objectives:
  - Integration procedure which is reproducible
  - Largely automated integration

https://www.agilealliance.org/glossary/continuous-integration/

Continuous Integration - Costs

– Bugs are caught much earlier in the delivery process when they are cheaper to fix, providing significant cost and time savings



Capers Jones, Applied software measurement (2nd ed.): assuring productivity and quality, (1997), McGraw-Hill

Therac-25 Overdose*

- What happened?
  - Therac-25 radiation therapy machine
  - Patients exposed to overdose of radiation (100 times more than intended) - 3 lives!!

- Why did it happen?
  - Nonstandard sequence of keystrokes was entered within 8 seconds
  - Operator override a warning message with error code (“MALFUNCTION” followed by a number from 1 to 64) which is not explained in the user manual
  - Absence of independent software code review
  - **‘Big Bang Testing’**: software and hardware integration has never been tested until assembled in the hospital

*https://en.wikipedia.org/wiki/Therac-25#Problem_description

Discussion

Continuous Integration is a set of tools that help to automate software building and testing?

True/False.

Briefly explain why/why not?

Continuous Integration and Tooling

- CI should not be confused with the tools that assist it
  - CI is a development practice not tools
  - Different automation tools can be used including version control, system build and testing
- CI should lessen the pain of integration by increasing its frequency

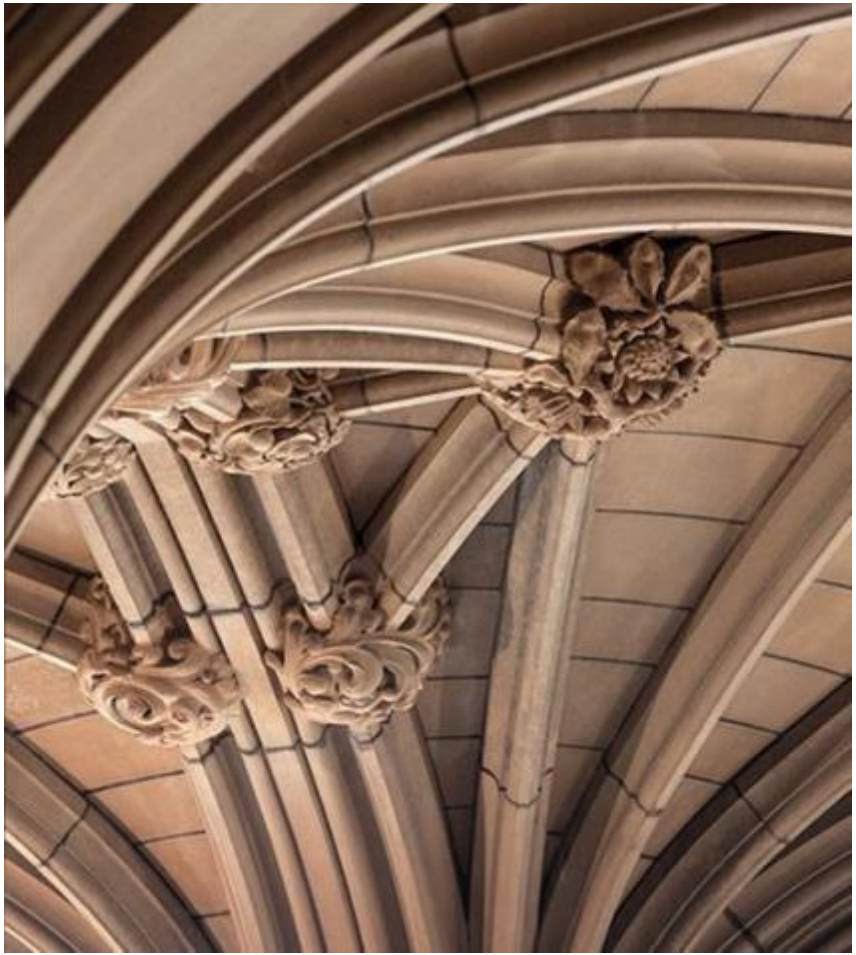
https://www.agilealliance.org/glossary/continuous-integration/

Continuous Integration - Implementation

- Use of a version control tool (e.g., CVS, SVN, Git)
- Use of automated build process
- Commitment and discipline from development teams (practice)
- Configuration of the system build and testing processes
- Use of CI server (Jenkins) to automate the process of integration, testing and reporting of test results (optional)

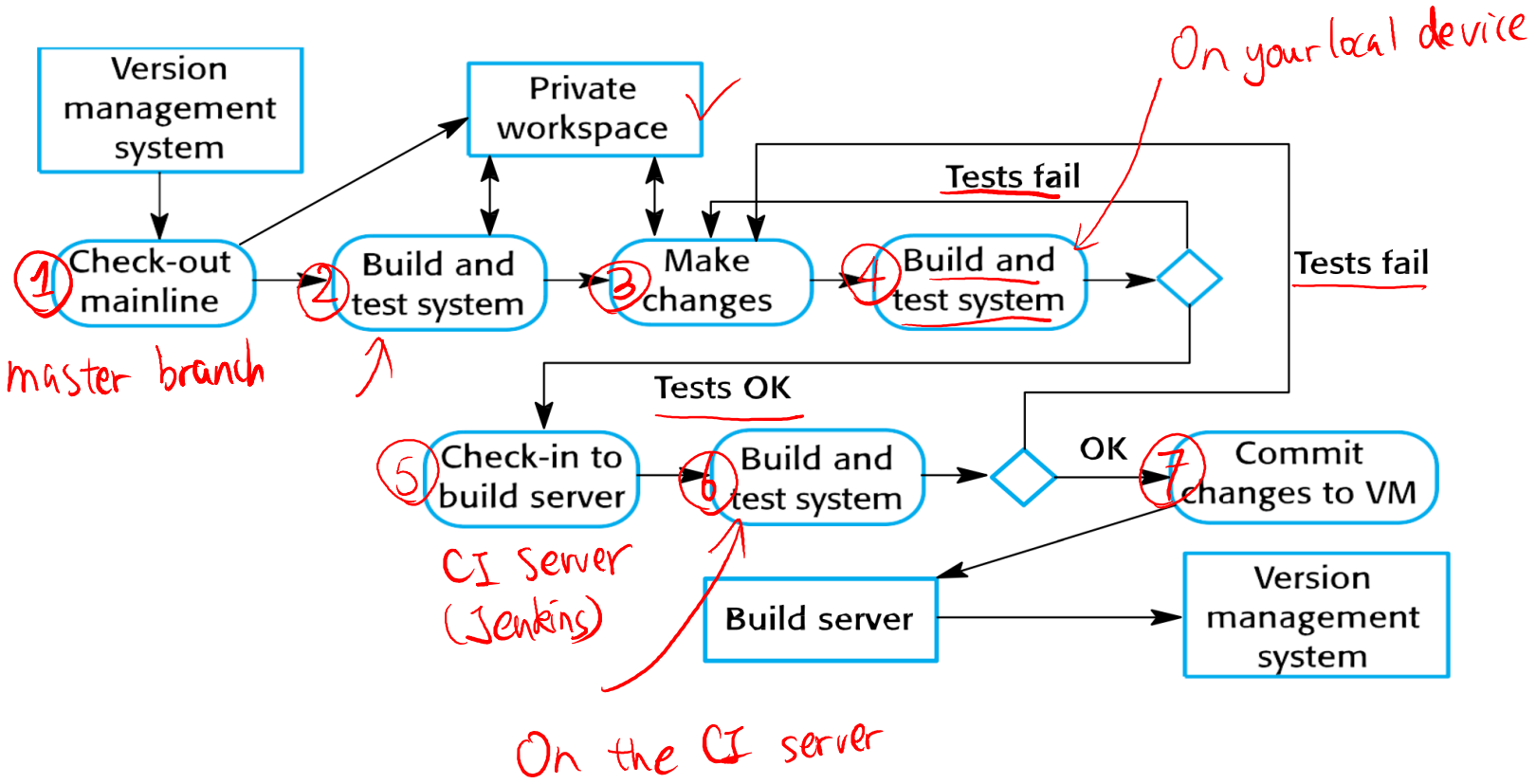
https://www.agilealliance.org/glossary/continuous-integration/

Continuous Integration Workflow



THE UNIVERSITY OF SYDNEY

Continuous Integration - Workflow



Continuous Integration - Workflow (1)

1. A developer check-out the system mainline from the VC system
2. Build the system and run automated tests to ensure the build system passes
   - Tests not passed, notify the developer who check-in the last baseline of the system
3. They make the changes to the system components (e.g., add new feature)
4. They build the system in a private workspace and rerun system tests
   - If the tests fail, they continue editing

Continuous Integration - Workflow (2)

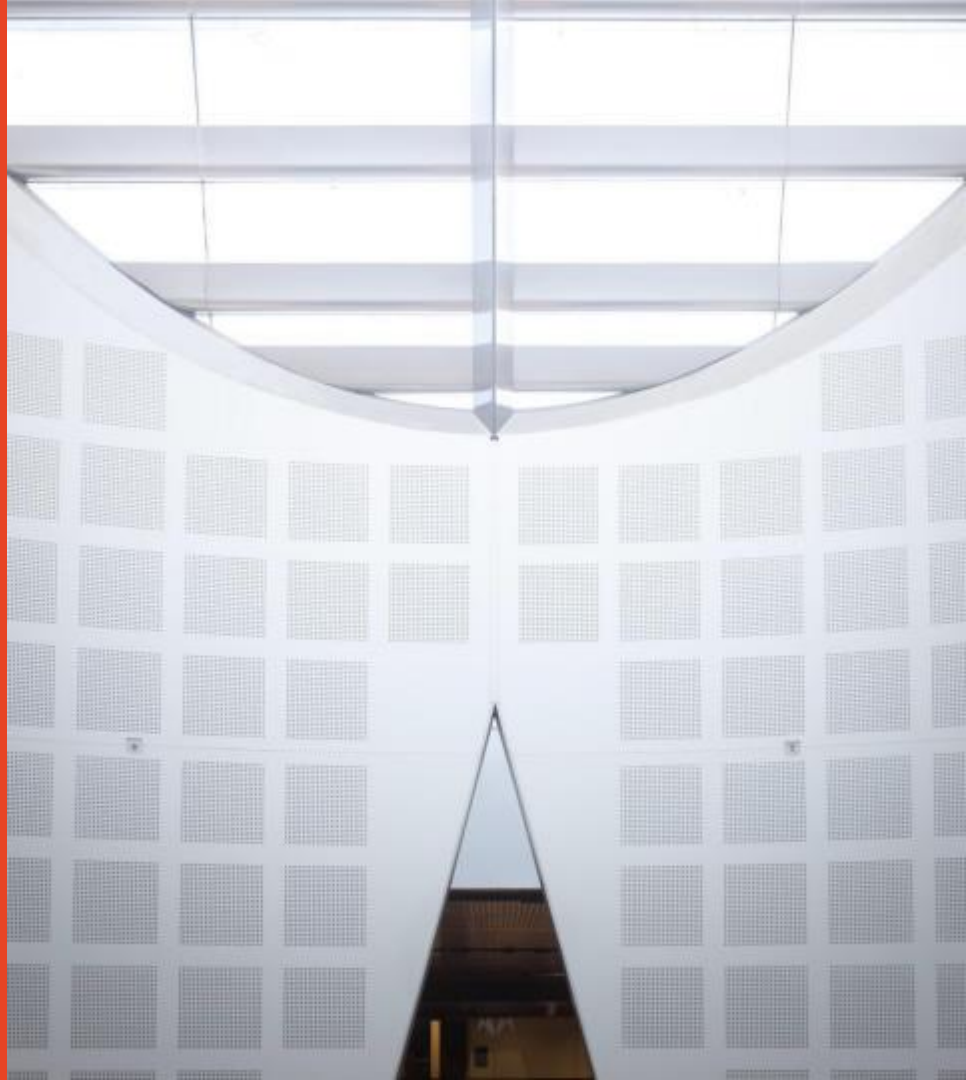
5. System passed tests; they check it into the build system server

6. Build the system on the build server and run the tests
   – Changes committed by other developers since last check-out, these need to be checked out and tests should pass on the developer's private space

7. Commit the changes they have made in the system mainline

Effective Continuous Integration

Automation practices



Effective CI - Automation Practices (1)

Regular Check-in

- Developers to check in their code into the mainline regularly (at least couple of times a day)
  - Small changes less likely to break the build
  - Easier to revert back to last committed version
- Check-in vs branching
  - Working on branch means your code is not being integrated with other developer's code, especially long-lived branches

Effective CI - Automation Practices (2)

Create a comprehensive automated test suite

Three types of automated tests should considered in CI build:

- *Unit Testing*: test the behavior of small pieces in isolation
- *Component Testing*: test the behavior of several components
- *Acceptance Testing*: test if the application meets the acceptance criteria set by end users (functional and non-functional)

- These sets of tests should provide high level of confidence that any introduced change has not broken existing functionality.

Effective CI - Automation Practices (3)

Keep the build and test process short

- Developers may stop doing full build and running the test before they check-in
- CI will likely miss multiple commits if it takes too long
- Lead to less check-ins as developers wait for the software to build and tests to run

– Should aim for 1.5 to 10 minutes (depending on the application size)
– Optimize your tests to achieve same coverage
– Split testing into two stages:
  - Compile software and run suite of unit tests and create deployable binary
  - Use binary to run acceptance, integration and performance tests

Effective CI - Automation Practices (4)

Developers to manage their development workspace

- They should be able to build, run automated tests and deploy on their local machines using the same processes used in the CI

Juran and Gryna 1998

Effective CI - Automation Practices (5)

Use CI software

- CI activities often has 2 components;
  - 1 CI Workflow execution
    - It polls VCS, check out a copy of the project to a directory on the server if it detects changes
    - It execute commands to build the application and run automated tests
  - 2 CI results visualization and reporting
    - Results of the processes that have been run, notifies you of the success or failure of the build and automated tests and provide access to test reports

Effective Continuous Integration

Essential practices - development teams

Effective CI – Team Practices (1)

Don’t Check in on a broken build

- If the build fails, responsible developers will need to find the cause of breakage and fix it quickly
- CI process will always identify such breakage and will likely lead to interrupting other developers in the team

Effective CI - Team Practices (2)

Always run all commit tests locally, or get your CI server to do it for you

- Other check ins before your last update from the VCS
- Check-in indicates that either a developer forgot to add some new artifacts to the repository, or someone checked in in the meantime
- Modern CI servers offer pretested commit, personal build
  - The CI server will take your local changes and run a build with them on the CI environment
    . Either check-in your changes or notifies you of build failure

Effective CI - Team Practices (3)

Wait for commit Tests to pass before moving on

- CI is shared among all developers
- Keep the system in consistent and stable state
- Fix failure without interrupting the workflow

Effective CI - Team Practices (4)

Never Go Home on a Broken Build

- No broken build
- Check-in regularly and early enough to give yourself time to deal with problems
- If fails, revert your change from source control and leave it in your local working copy

Effective CI - Team Practices (5)

Always be prepared to revert to the previous revision

"*Airplane pilots are taught that everytime they land, they should assume that something will go wrong, so they should be ready to abort the landing attempt and go around to make another try*"

- Fix it quickly or revert and fix it locally
- One of the VCS benefits
- Mindset – be prepared

Effective CI - Team Practices (6)

Time-box fixing before reverting

- Set a team rule; fixed it within X-minute or revert
- Think of other team members

Effective CI - Team Practices (7)

Do not comment out failing tests

- Often a result of the time-box fixing

- It takes time to find the root cause of tests failing
  - Are test cases no longer valid given many changes made to the application?
  - You may need to talk to other developers

Effective CI - Team Practices (8)

Take responsibility for all breakages that result from your changes

- It is your responsibility, because you made the change, to fix all tests that are not passing as a result of your changes
- To fix breakages, developers should have access to any code that they can break through their changes

Continuous Integration

Case Study

Continuous Integration at Scale - Google

- 20,000+ developers in 40+ offices
- 4,000+ projects under development
- Single code tree (billions of files)
- 30,000 check-ins per day
- Everyone develops and releases from head
- All builds from source
- >100 million test cases executed per day
- can roll back anyone else's code change if it's causing problems

http://www.infoq.com/presentations/google-test-automation

Continuous Delivery (CD)

Continuous Delivery

– “Continuous Delivery is a software development discipline where you build software in such away that the software can be **released** to production **at any time**.” – *Martin Fowler, 2013*

Continuous Delivery

- Release new changes to customers quickly and in reproducible way

- Automate the release process so you can deploy your application at any time
  - Packaging artifacts (using automated building tools) to be delivered to end users

Why Continuous Delivery?

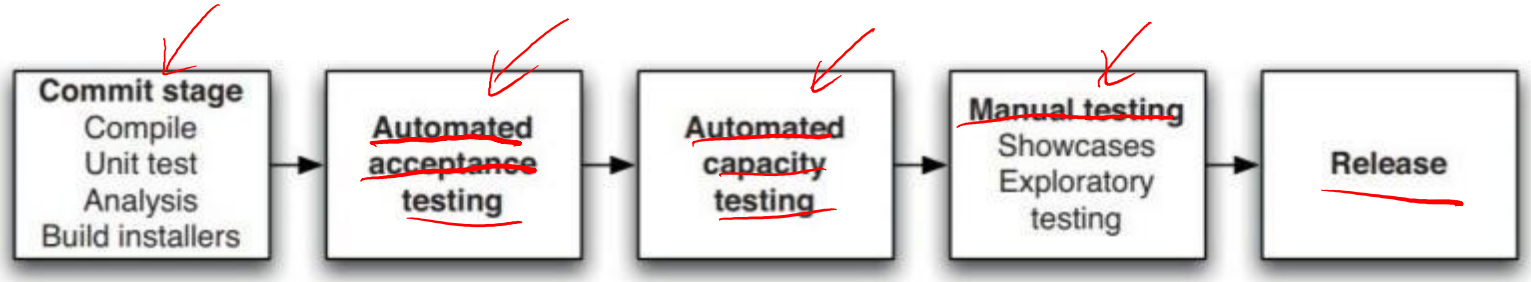
- "Our highest priority is to satisfy the customer through ***early and continuous delivery of valuable software***." – Agile Principle
- Cycle time in hours not months
- Quality should be built-in to the process
- Late feedback is expensive!

Continuous Delivery - Deployment Pipeline

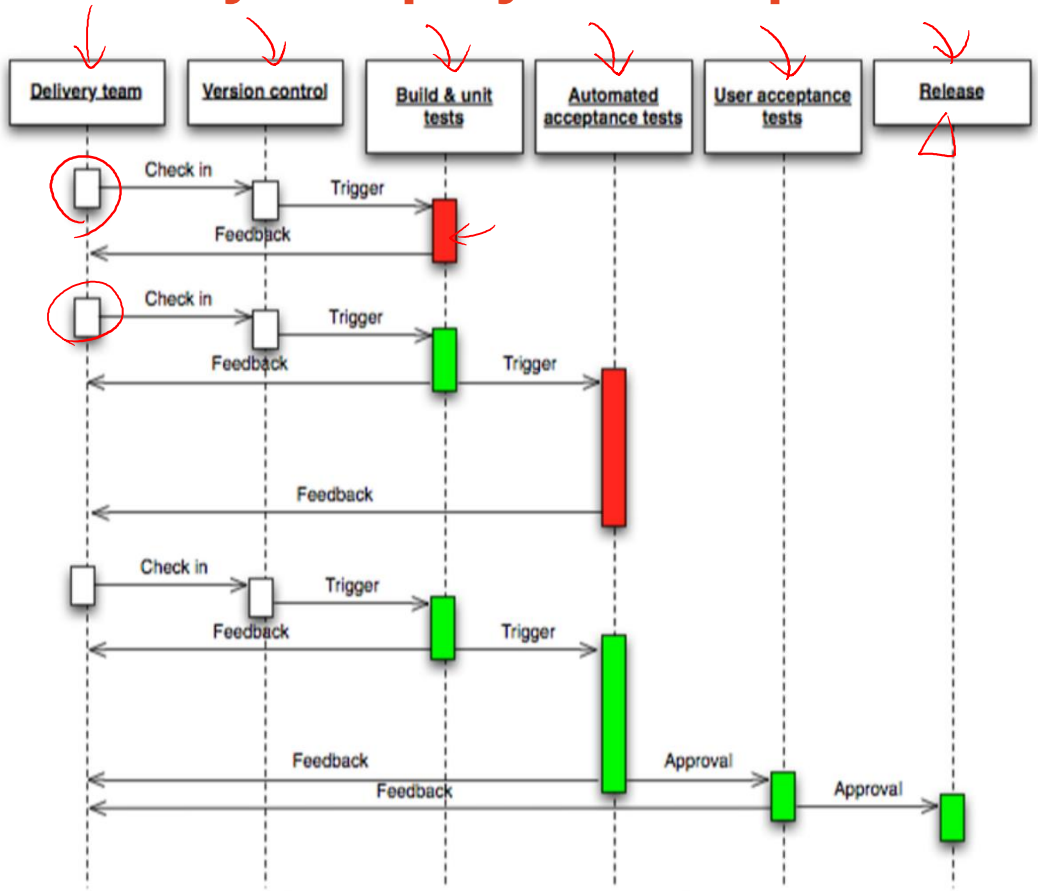
- An automated implementation of the application's build, deploy, test and release process
- Has its foundations in the process of CI

Continuous Delivery - Deployment Pipeline

– Example of deployment pipeline



Continuous Delivery - Deployment Pipeline



http://continuousdelivery.com/2010/02/continuous-delivery/

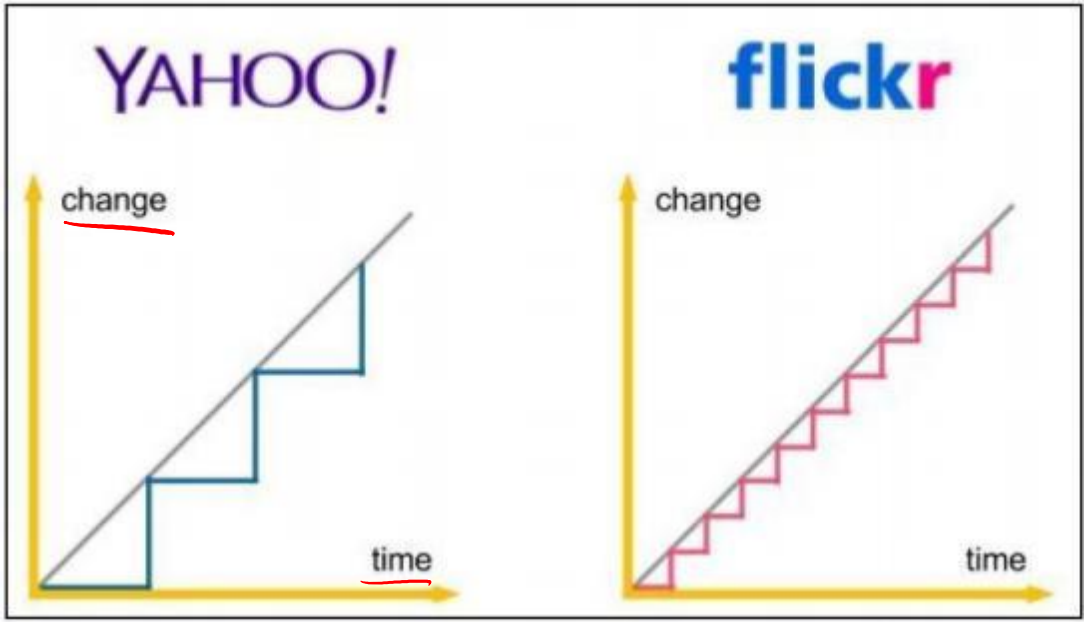
Continuous Delivery

Case Study

Case Study - Yahoo and Flicker

Yahoo	Flicker
Well-established corporation	Start-up
Rigid development processes	Agile development process
Heavy delivery process	Many releases per day (CD)
Very experienced developers	Treated as junior developers
Higher downtime	Lower downtime

Case Study - Delivery Processes



Continuous Delivery - Industry Examples

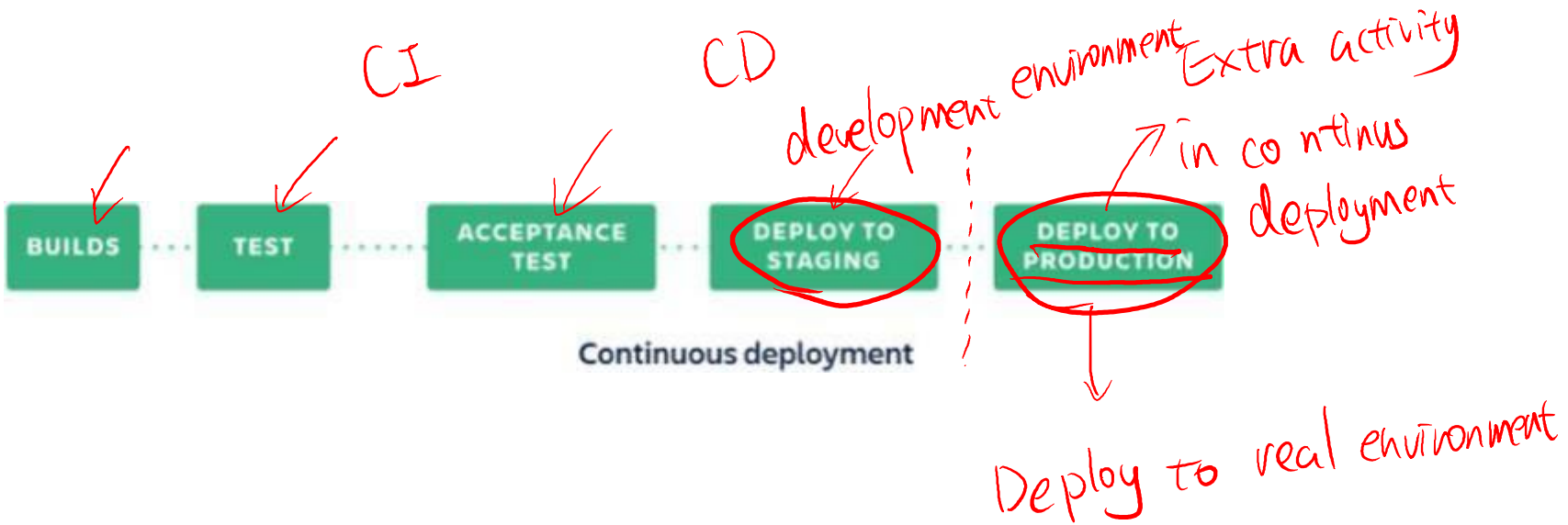
- Amazon (2011): 11.6 seconds between deployment
- Facebook (2013): 2 deployments a day
- HubSpot (2013): 300 deployments per day!
- Atlassian (2016): 65% of customers practice continuous delivery

Continuous Deployment

Continuous Deployment

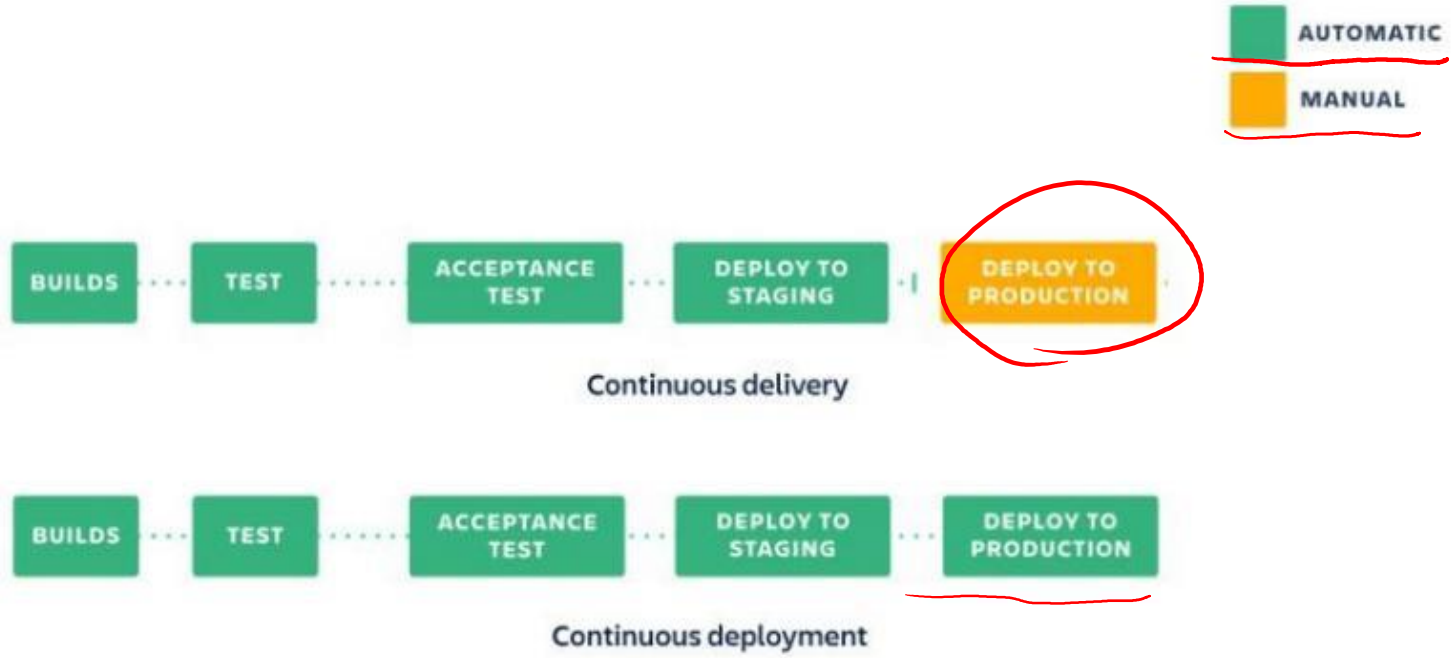
“The practice of releasing every good build to users – a more accurate name might have been 'continuous release’” ---- Jez Humble, 2010

Continuous Deployment



https://www.atlassian.com/continuous-delivery/continuous-deployment

Continuous Delivery Vs Continuous Deployment



https://www.atlassian.com/continuous-delivery/continuous-deployment

CD → Continuous delivery

Deployment Automation

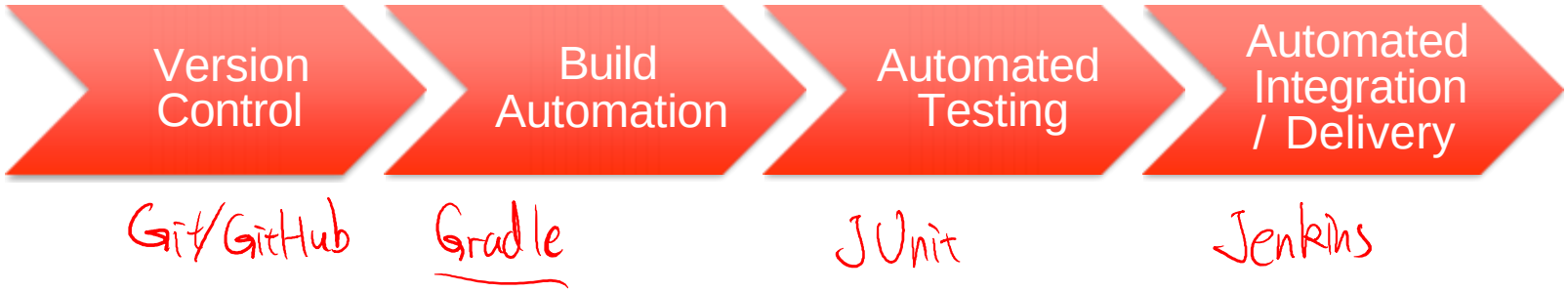
- Deployments should tend towards being fully automated
  - Pick version & environment
  - Press “**deploy**” button
- Automated deployment scripts should be up-to-date
- Don’t depend on the deployment expert
- Automated deployment process:
  - Cheap and easy to test
  - Fully auditable
  - Should be the only way in which the software is ever deployed

Continuous Integration / Delivery Tools

Jenkins

THE UNIVERSITY OF SYDNEY

Tools for Agile Development



Jenkins

- Jenkins is an automation server to automate tasks related to building, testing, and delivering or deploying software.

- *Jenkins pipeline* supports implementing and integrating *CD pipelines* into Jenkins
  - CD pipeline is an automated expression of your delivery process
  - Written in *Jenkinsfile*

https://jenkins.io/

Jenkins - Running Multiple Steps

- Building, testing and deploying activities are defined as stages and steps
- Jenkins allow composing different steps (commands) to model simple and complex automation processes

- Special steps:
  - **Retry:** retrying steps a number of times (e.g., *retry(4)*)
  - **Timeout**: exiting if a step takes long time (*timeout (time: 3, unit: MINUTES)*)
  - **Clean-up**: to run clean-up steps or perform **post** actions based on the outcome of the pipeline when it finishes executing

https://jenkins.io/

Jenkins - Execution Environment

- The ***agent*** directive specifies how to execute pipeline
  - All steps contained within the block are queued for execution
  - A workspace is allocated which will contain files checked out from source control and any additional files for the pipeline

- Pipeline is designed to use ***Docker*** images and containers to run inside
  - No need to configure various tools and dependencies on agents manually

https://jenkins.io/

Jenkins - Recording Tests and Artifacts

- JUnit already bundled with Jenkins
- Jenkins can record and aggregate all test results and artifacts
  - Reported through using the **post** section
  - A pipeline that has failing tests will be marked as **UNSTABLE**
- Jenkins stores files generated during the execution of the pipeline (aftifacts)
  - Useful for analyzing and investigation in case of test failures

https://jenkins.io/

Jenkins - Cleaning and Notifications

- **Post** section can be used to specify clean up tasks, finalization or notifications
  - Post section will run at the end of the pipeline execution
  - E.g., lean up our workspace, delete unneeded directories

- Notifications can be set-up to
  - Send emails
  - Post on Slack or other

- Notification can set-up when things are failing, unstable, or even succeeding

https://jenkins.io/

References

- Humble, J., Farley, D., 2010. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional

- Ian Sommerville 2016. Software Engineering: Global Edition (3rd edition). Pearson, England

- Rafal Leszko. 2017. Continuous Delivery with Docker and Jenkins: Delivering Software at Scale. Packt Publishing.

- Kevic, Katja; Murphy, Brendan; Williams, Laurie; et, al. Characterizing Experimentation in Continuous Deployment: A Case Study on Bing,2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2017

Tutorial: Continuous Integration with Jenkins

Lecture: Group Dynamics; Tools and Technologies for Teamwork, Issue Tracking

- Teams in Software Development Models
- Teams in Agile Development
- Effective Teams and Teamwork