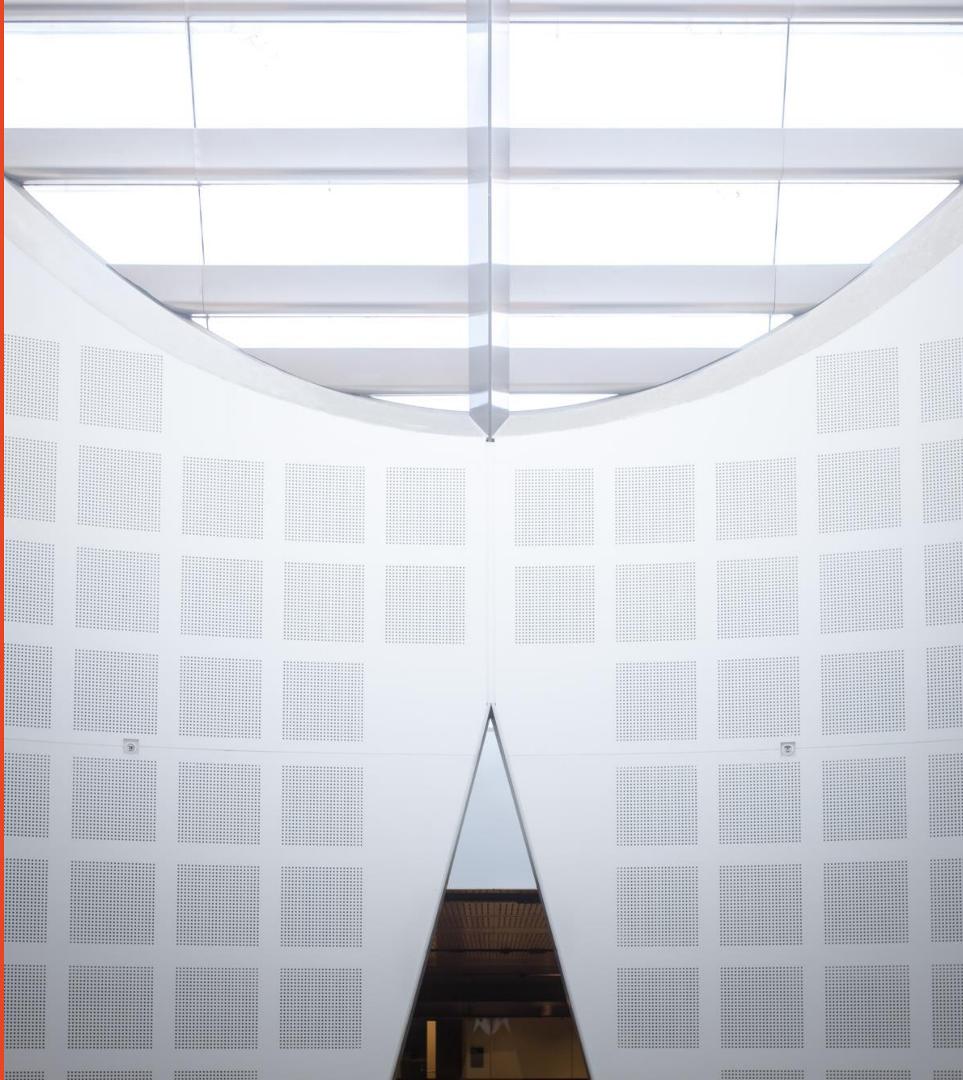
Agile Software Development Practices

SOF2412 / COMP9412

Continuous Integration

Dr. Basem Suleiman

School of Information Technologies



Copyright warning

Agenda

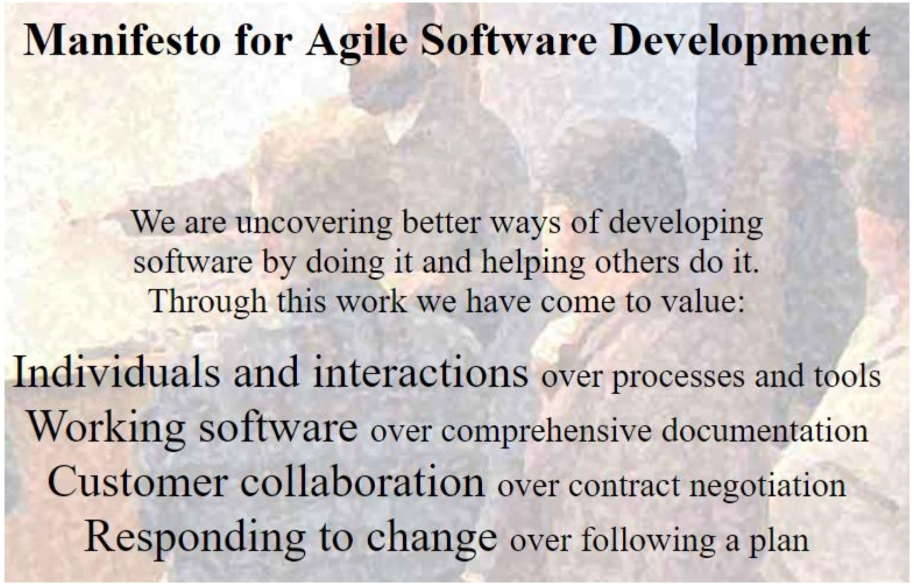
- Agile Development
- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Tooling – Jenkins

Agile Development

Agile manifesto, principles and practices

THE UNIVERSITY OF SYDNEY

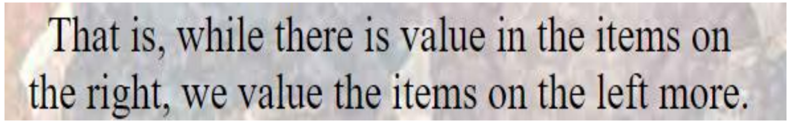
Agile Manifesto (2001) – Revisit



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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



That is, while there is value in the items on the right, we value the items on the left more.

Agile Manifesto: http://agilemanifesto.org/

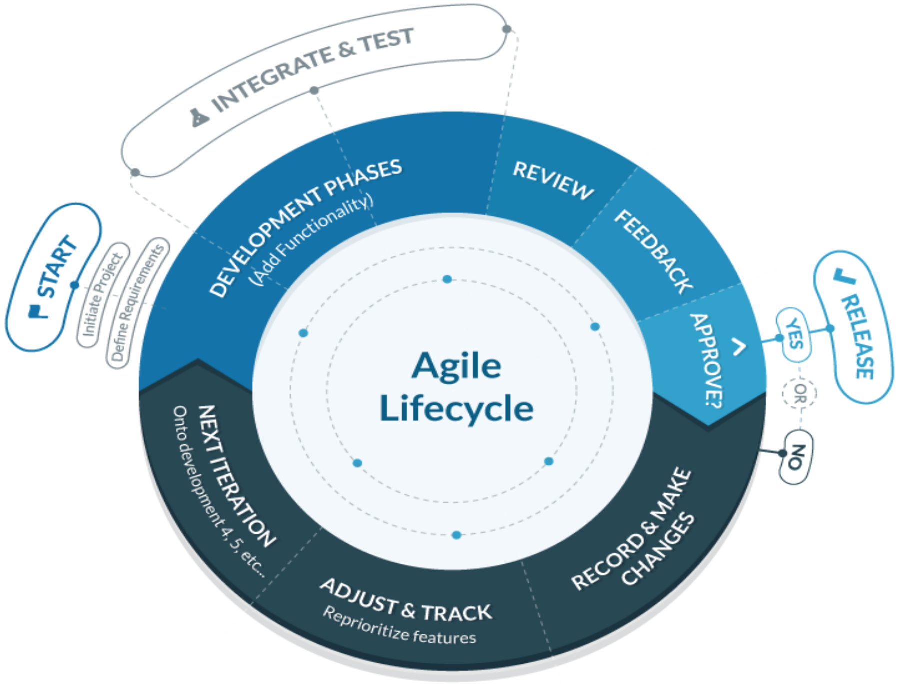
Agile Principles – Revisit

- 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**

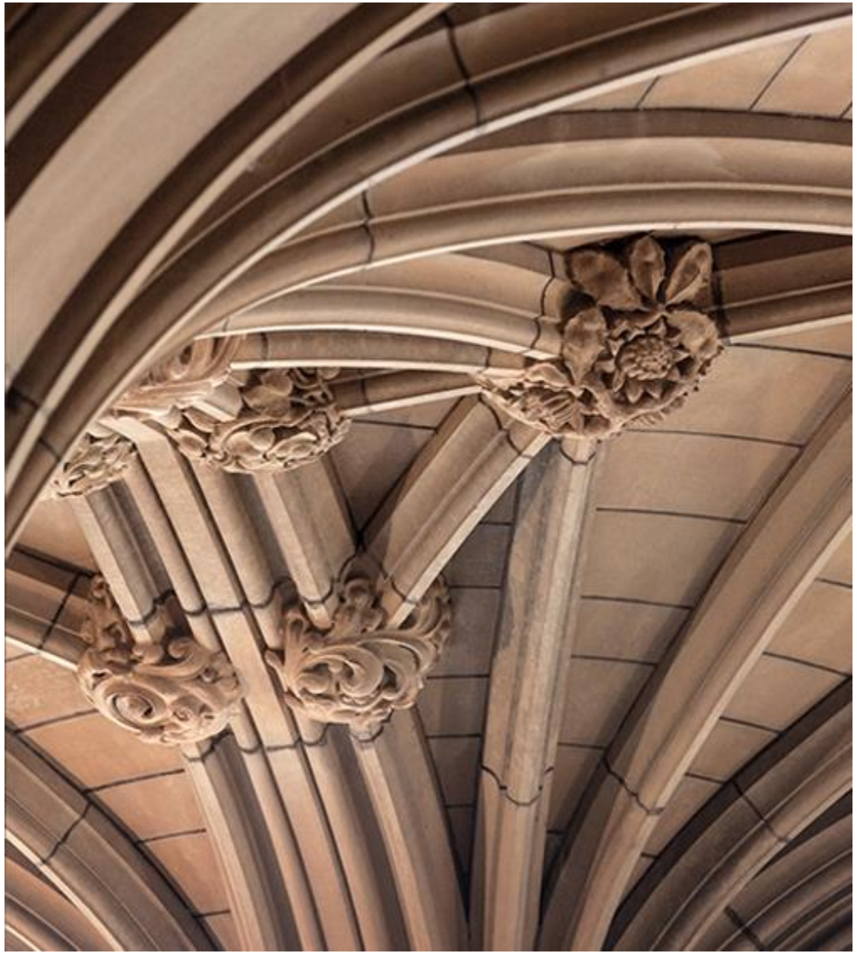
Agile Development – Integration

– Integrate and Test



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

What is Continuous Integration?



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*

- **Continuous Integration** : Software is developed in smaller, incremental change sets which are regularly integrated into the codebase of the complete product, where a process automatically builds and runs a test suite daily, hourly, or even per individual change .

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

Continuous Integration (CI)

- Suppose that two developers, or group of 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 and expected behavior of the system

- Individual developers may work on components integrated into subsystems
- Sub-systems may be integrated with other team's subsystems for forma a larger system

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

Continuous Integration (CI)

- The goal of CI is that the software is in a working state all the time
  - 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 – Why?

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

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
  - Build process to automatically trigger the unit testing every time any change is published to version control system
  - Alerting the team of a broken build in case of single test failing
- Use of CI server to automate the process of integration, testing and reporting of test results (optional)

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

Group Discussion

- CI is set of tools. Discuss Why/Why not?
- CI involves many activities, how should it help in Agile development?

THE UNIVERSITY OF SYDNEY

Continuous Integration and Tooling

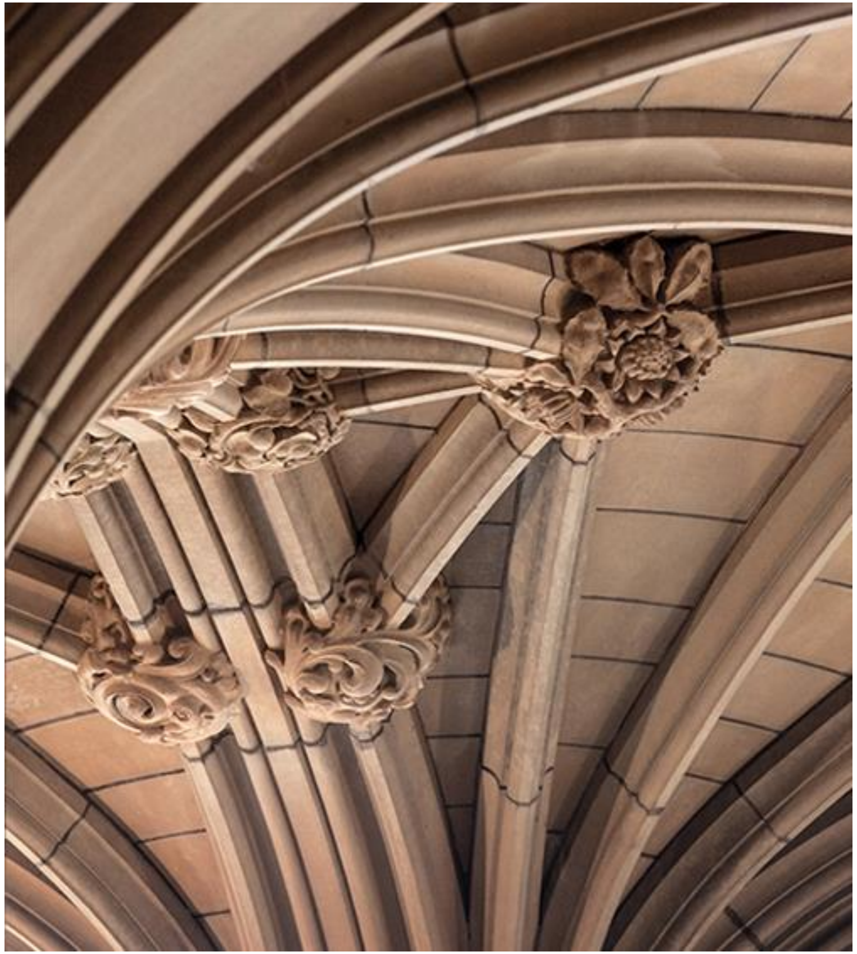
- CI should not be confused with the tools that assist it
  - CI is a development practice not tools
  - Different automation tools can be used including version control, system build and testing

- CI should lessen the pain of integration by increasing its frequency

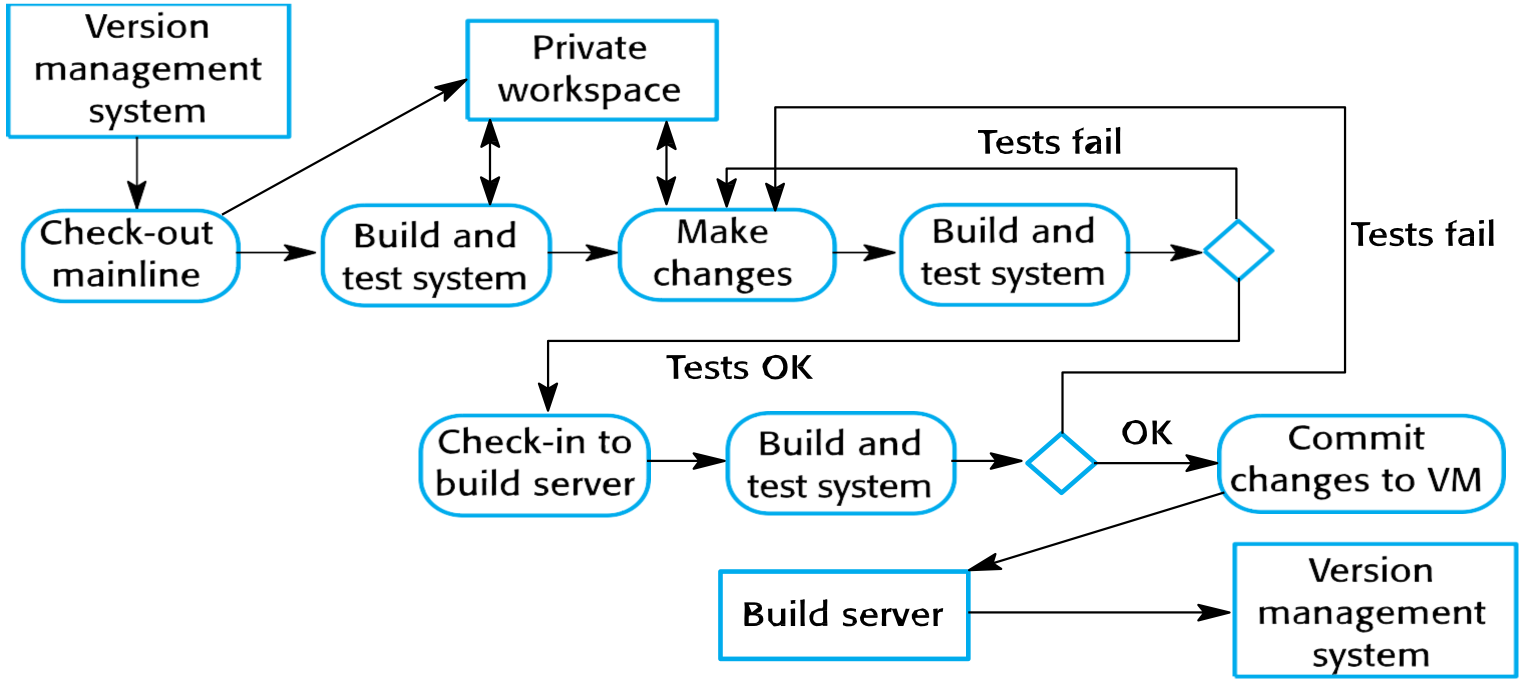
https://www.agilealliance.org/glossary/continuous-integration/

Continuous Integration

Workflow



Continuous Integration – Workflow



Continuous Integration – Workflow (1)

1. A developer check-out the system mainline from the VC system into their private workspace
2. They build the system and run automated tests to ensure the build system passes all tests
   – Tests not passed (build is broken) notify the developer who check-in the last baseline of the system to repair the problem
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)

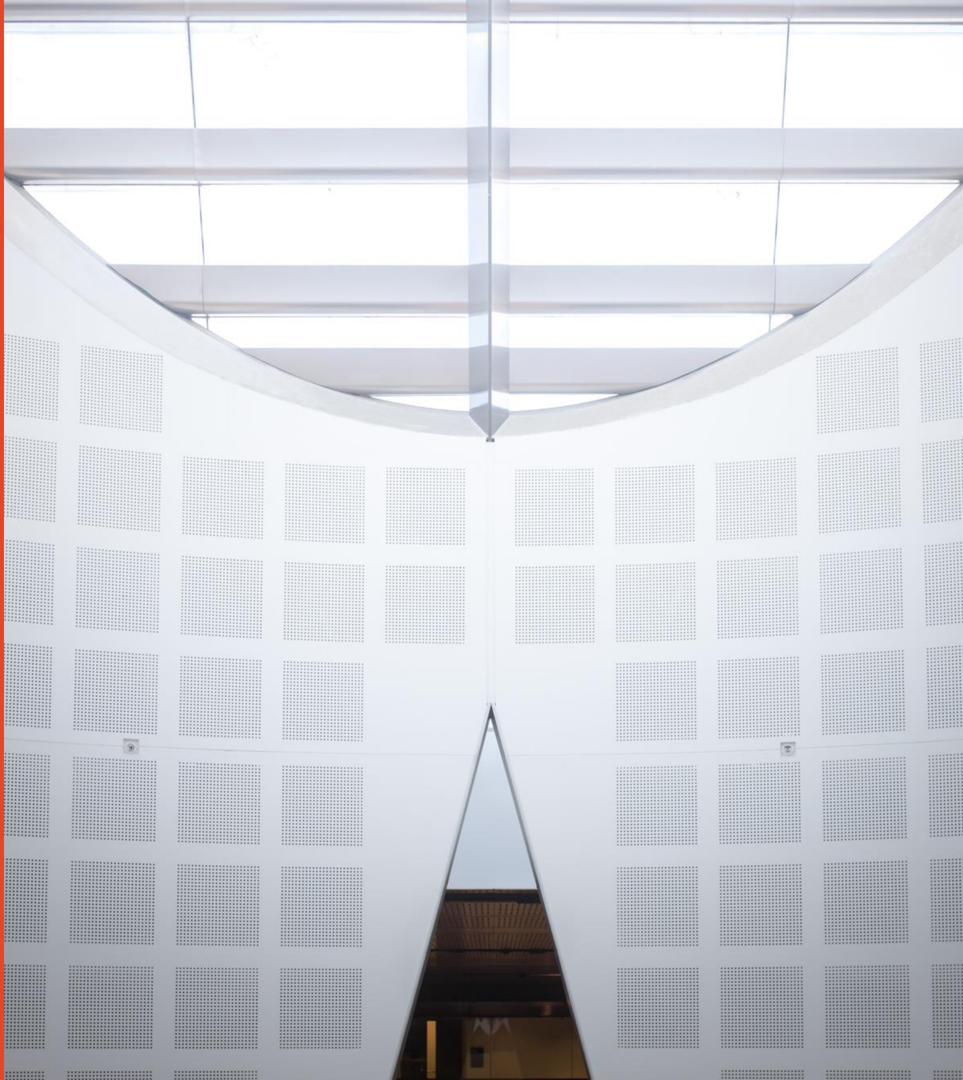
5. Once the system has passed its tests, they check it into the build system server
   - They do not commit it as a new system baseline in the VC system

6. They build the system on the build server and run the tests (if using Git, pull recent changes from the server to their private workspace)
   - If any changes were committed by other developers since last check-out, then these need to be checked out and tests should pass on the developer's private space

7. They commit the changes they have made as a new baseline in the system mainline

Effective Continuous Integration

Automation practices



Effective CI – Automation Practices (1)

Regular Check-in

- Developers to check in their code to the mainline regularly (at least couple of times a day)
  - Stick to small changes which are 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

Juran and Gryna 1998

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, combined, should provide an extremely high level of confidence that any introduced change has not broken existing functionality

Juran and Gryna 1998

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
  - Will likely lead to less check-ins as developers will have to 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

Juran and Gryna 1998

Effective CI – Automation Practices (4)

Developer's to manage their development workspace

- They should be 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

- Many software provide the infrastructure for CI activities, often has 2 components
- CI Workflow execution
  - It polls VCS at regular intervals, check out a copy of the project to a directory on the server if it detects changes
  - It then execute pre-specified commands to build the application and run automated tests
- 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

Juran and Gryna 1998

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 as soon as possible
- IC 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 before committing, or get your CI server to do it for you

- Other developers may have checked in before your last update from the VCS
- Check-in indicates that either a developer forgot to add some new artefacts 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

Never Go Home on a Broken Build

- Try to fix so the system is not on broken build
- Check-in regularly and early enough to give yourself time to deal with problems
  - Many experienced developers make a point of not checking in less than an hour before the end of work, or they leave that to do first thing the next morning
- If all else fails, simply revert your change from source control and leave it in your local working copy
  - Some version control systems make this easier by allowing you to accumulate check-ins within your local repository without pushing them to other users

Effective CI – Team Practices (4)

Always be prepared to revert to the previous revision

"*Airplane pilots are taught that every time 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*"

- Cannot fix the problem quickly, a developer should revert to the previous change-set held in the VCS and remedy the problem in their local environment
- One of the VCS benefits is to allow us precisely this freedom to revert
- Mindset – be prepared

Effective CI – Team Practices (5)

Time-box fixing before reverting

- Set a team rule: if the build breaks try to fix it for (ten) minutes, revert to the previous solution if you're not able to fix it in 10 minutes
- Think of other team members

Effective CI – Team Practices (6)

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 (7)

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
  - In some projects, access to the whole codebase is not affordable, so use team collaboration

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(2013)

Continuous Delivery

Continuous Delivery

- “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, 2013

- Continuous Delivery: the automated implementation of an application’s build, deploy, test, and release process

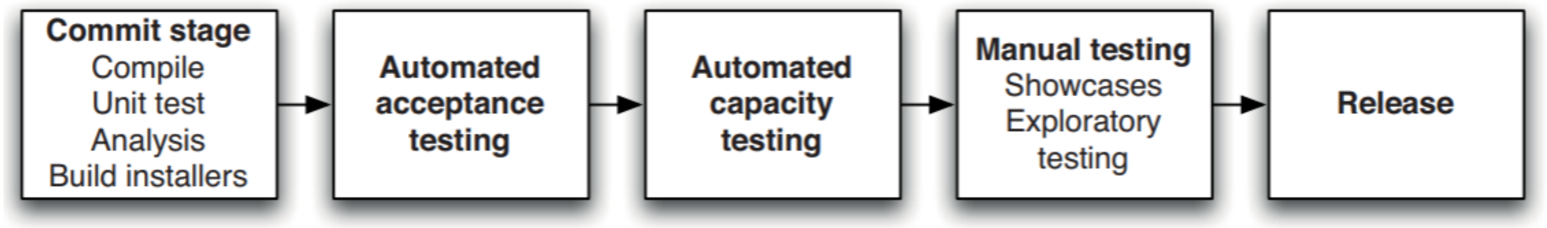
- Why Continuous Delivery:
  - Agile Manifesto – “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.”

Why Continuous Delivery?

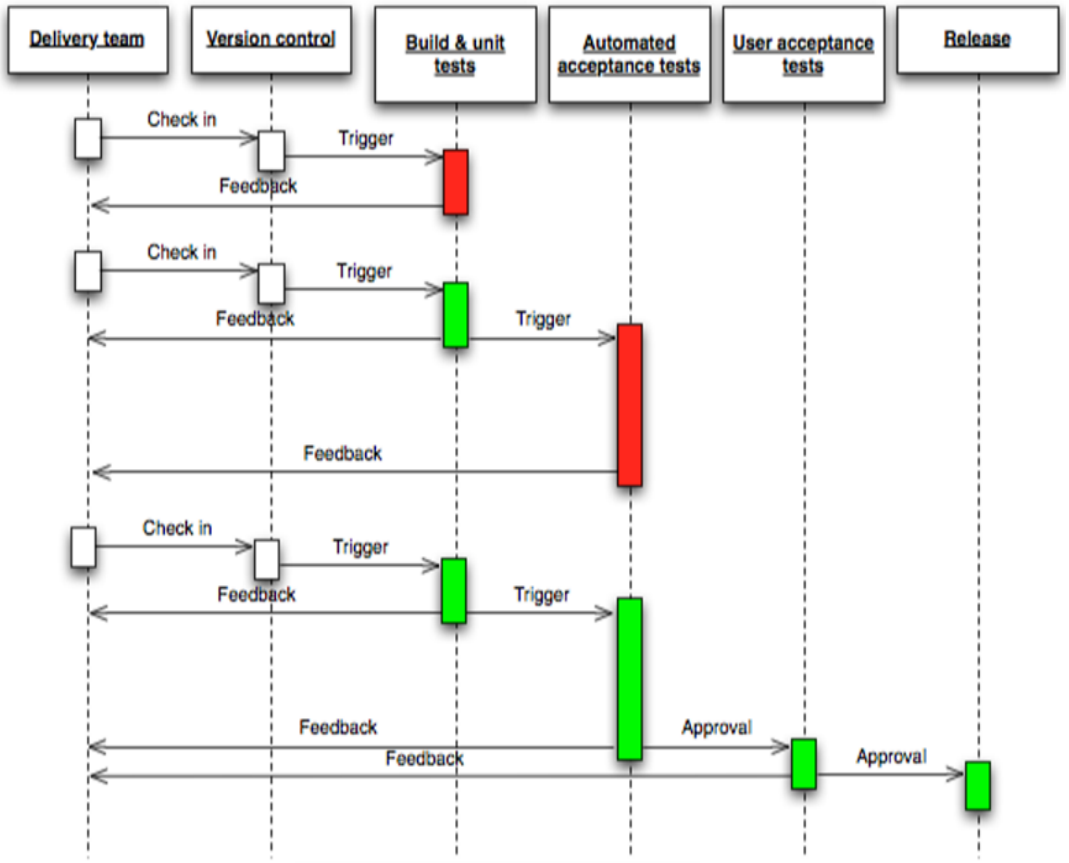
- “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.” – Agile Manifesto
- Cycle time should be measured in hours –not months – How long does a single change take to release?
- Quality should be built-in to the process – Not an afterthought of manual inspection
- Feedback should be close as possible to point of failure – 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



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

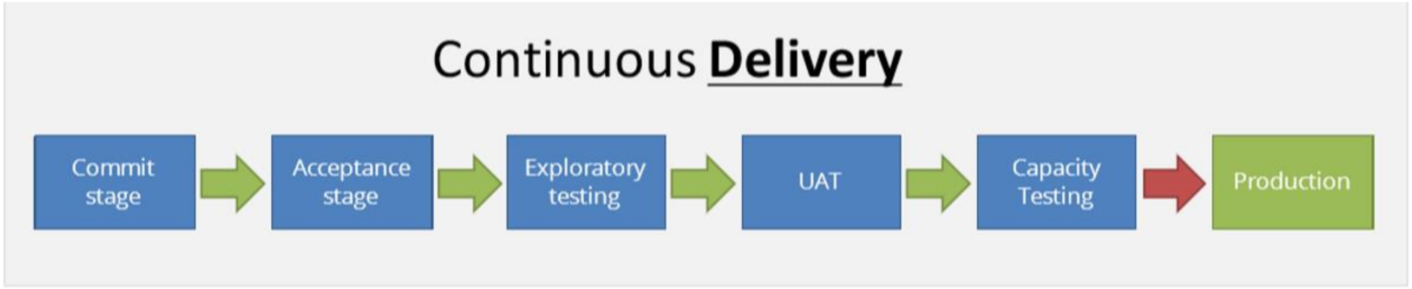
Continuous Deployment

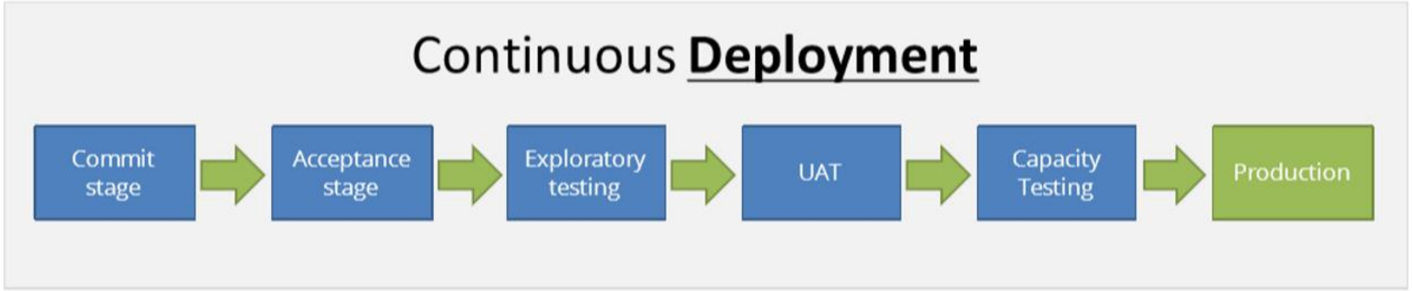
Continuous Deployment

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

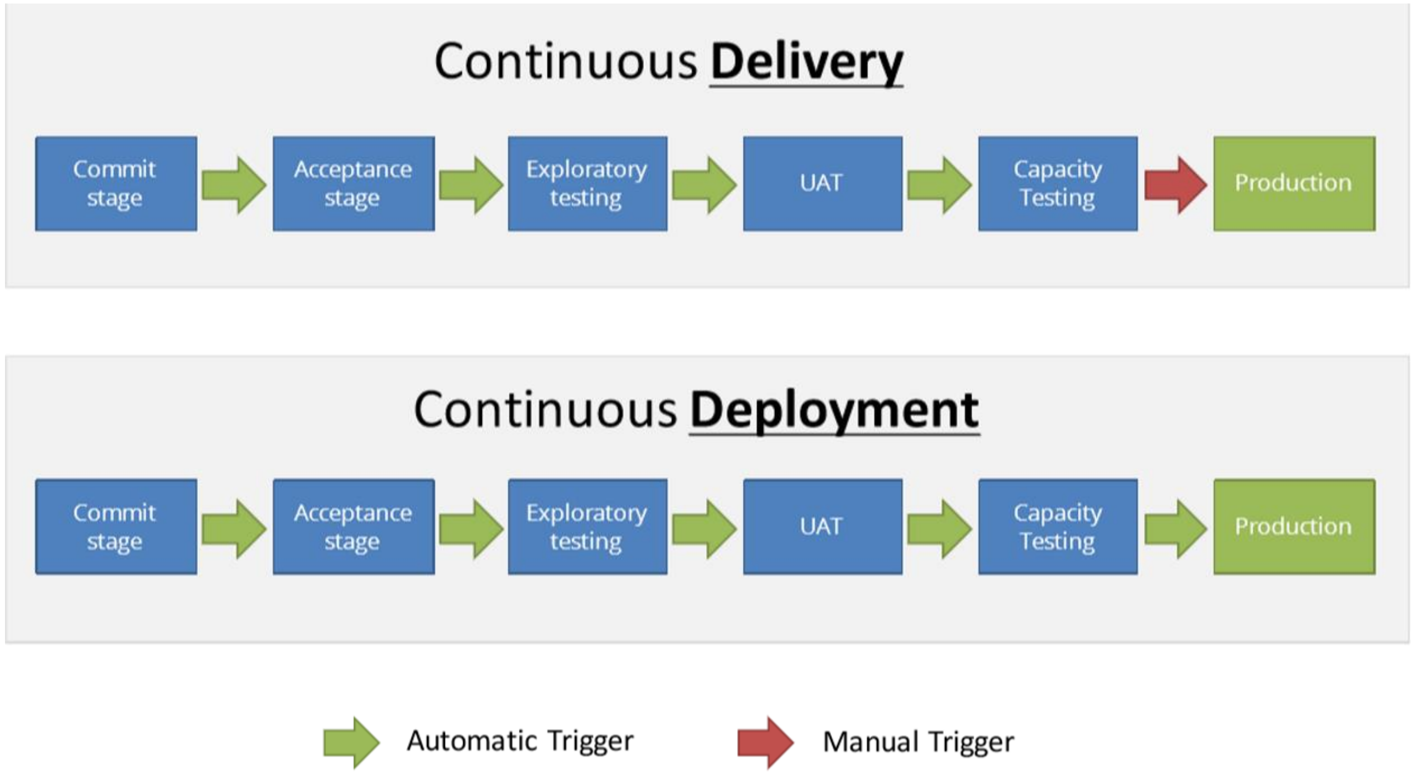
- **Continuous Deployment:** a continuation of the continuous delivery process, where the application or service is automatically deployed to the customer

Continuous Delivery Vs Continuous Deployment





Continuous Delivery Vs Continuous Deployment

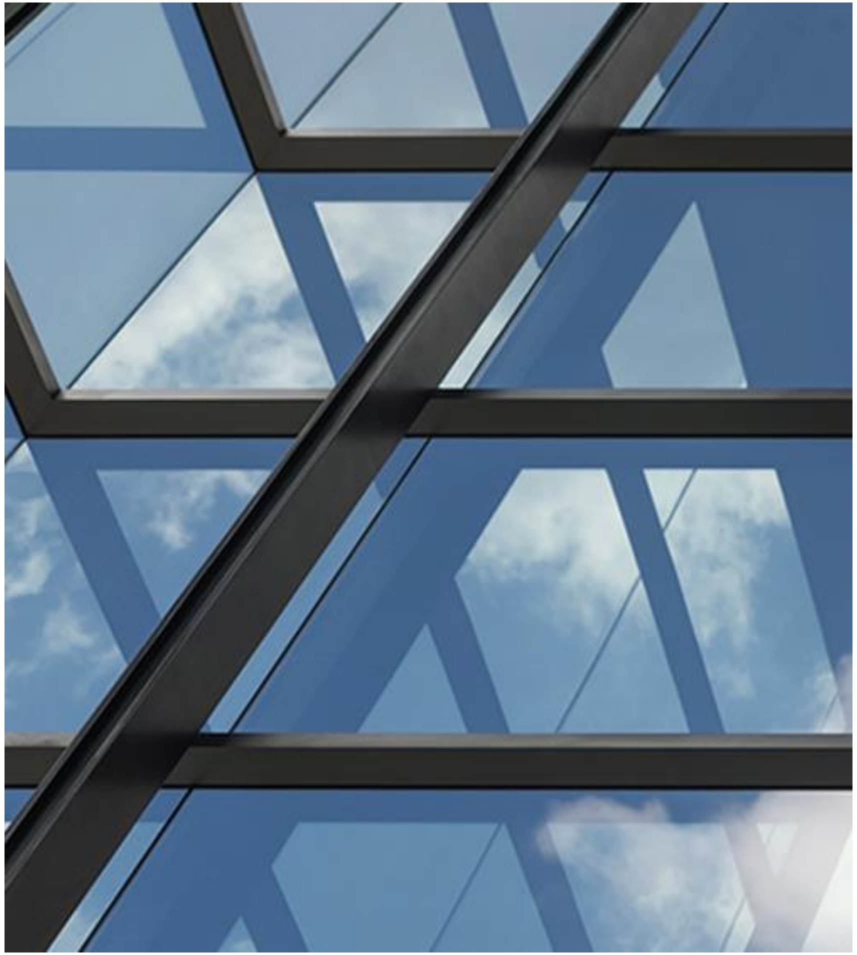


Deployment Automation

- 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 / Deployment Tools

Jenkins



Jenkins

- "Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software."

- *Jenkins pipeline* is a suite of plugins which supports implementing and integrating *continuous delivery pipelines* into Jenkins
  - A *continuous delivery pipeline* is an automated expression of your process for getting software from version control right through to end users/customers
  - Typically written in *Jenkinsfile* which is checked in a project's source code repository

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 to help the automation;
  - **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, the steps will start to execute as soon as an executor is available
  - 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
  - This means the pipeline can define the environment and tools required without having to configure various tools and dependencies on agents manually
  - Any tool which can be prepacked in a Docker container can be used

https://jenkins.io/

Jenkins – Recording Tests and Artifacts

- Junit step is already bundled with Jenkins

- Jenkins can record and aggregate all test results and artefacts
  - 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 (aftefacts)
  - This built-in support is 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 Hipchat

- 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
- 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