Continuous integration

Angelo Gargantini

Sw Testing e verification

Continuous **Integration**

Commit code to shared repository frequently

- Purpose is to avoid big merge conflicts
  i.e. (“integration hell”)
- A build server automatically builds software and runs unit tests and integration tests to identify regressions
- Code does not have to be production ready when committed- e.g. use good branching practices (feature branches) to separate development and release code

Continuous **Delivery**

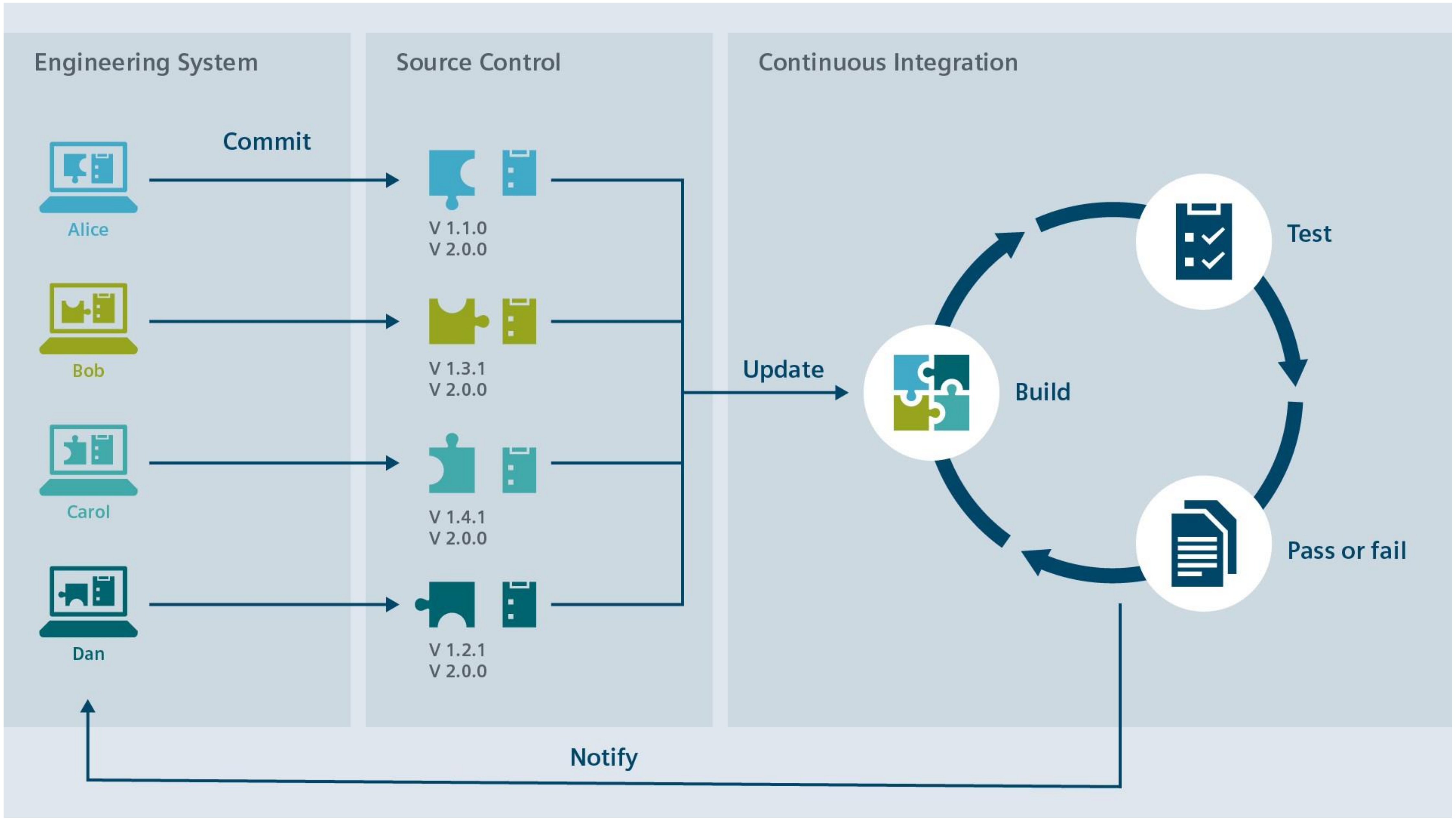
Publishable increments are “delivered” frequently, possibly automatically, as unit tests and acceptance tests pass

- Purpose is to shorten release cycles while ensuring that production code is always shippable
- “Delivery” is whatever mechanism is used to release code- could be as simple as a merge to the official release branch
    - Important part is the customer can test and accept code

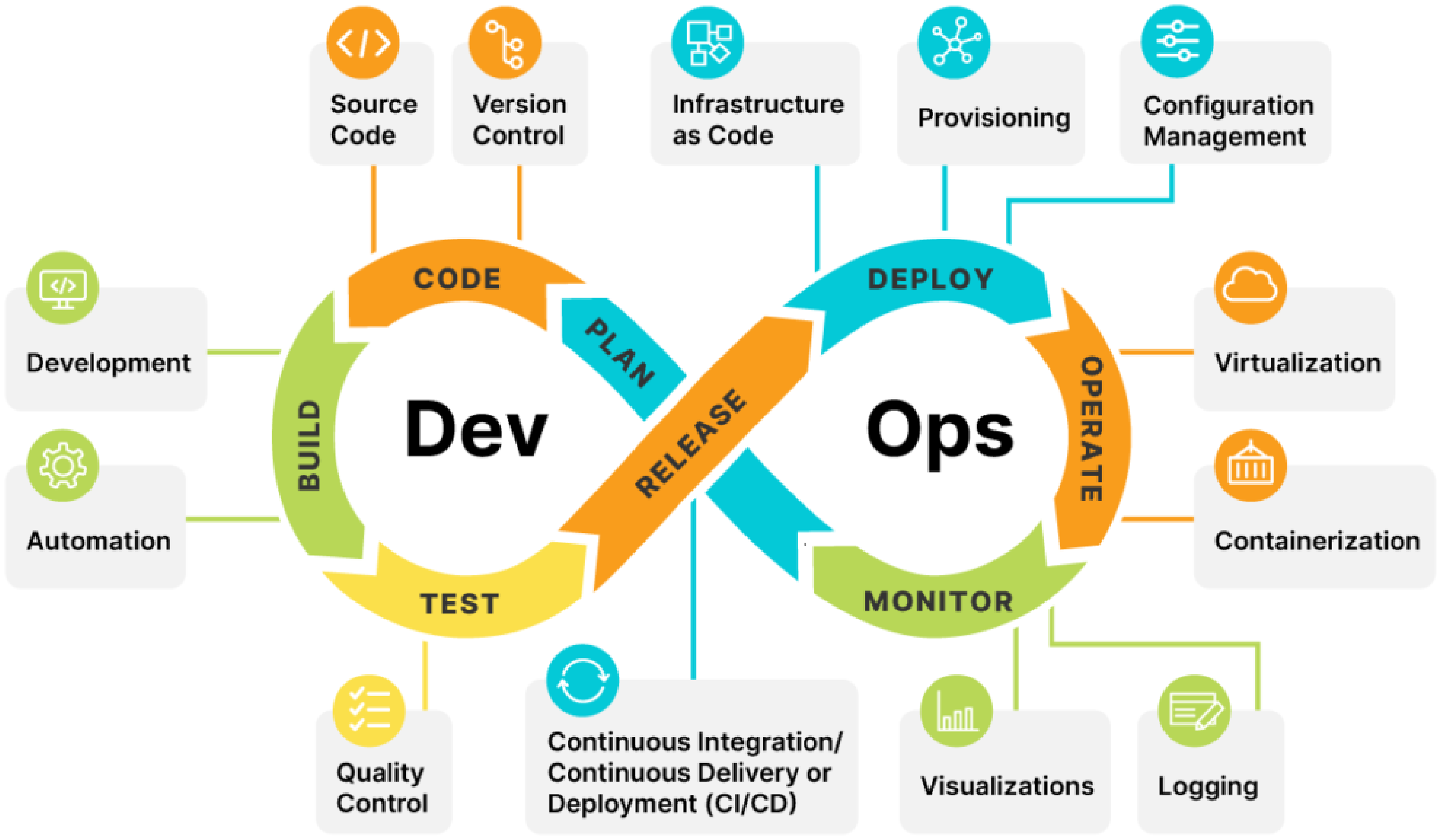
Continuous **Deployment**

Features that pass customer's acceptance tests are automatically deployed to production

- Reduces software development latency to production
- Minimizes changes released in each version of software
- Example: Gitlab allows automatic merging of a branch to master when automated test suite completes successfully


Engineering System
Source Control
Continuous Integration
Commit
Alice
Bob
Carol
Dan
V 1.1.0
V 2.0.0
V 1.3.1
V 2.0.0
V 1.4.1
V 2.0.0
V 1.2.1
V 2.0.0
Update
Build
Test
Pass or fail
Notify

devops


Source Code
Version Control
Infrastructure as Code
Provisioning
Configuration Management
Development
Automation
CODE
PLAN
BUILD
Dev
TEST
RELEASE
DEPLOY
Ops
OPERATE
MONITOR
Virtualization
Containerization
Quality Control
Continuous Integration/ Continuous Delivery or Deployment (CI/CD)
Visualizations
Logging

Gitlab CI/CD

Basic feature: allows shell commands to run in response to commits

You can define and run your own:

- Build process
- Test suite
- Packaging/deployment/merge process
- Etc.

CI/CD Execution

- Defined in root of repository in hidden file
  .gitlab-ci.yml
- A commit action results in running a *pipeline*
- A pipeline has one or more *stages*
- A stage has one or more *jobs*
- Each job is farmed out to a *runner*, which executes the desired commands
- If all jobs succeed, the pipeline stage succeeds and the next stage starts. Otherwise the stage fails and the whole pipeline stops
- Pipeline results are logged and email is sent in the case of failure

Gitlab Notes

- Using CI/CD requires a runner to be enabled, SLU’s Gitlab installation currently does not have any runners, but a free account at gitlab.com allows 2000 minutes of execution time per month (as of fall 2019)
- yml files stand for “Yet Another Markup Language,” yes it’s redundant, but there are plenty of tutorials
    - The top-level elements are usually *jobs*
    - Each job must define a *script* element that defines what commands are run as a part of that job
    - The *stages* top level element can define a set of pipeline stages. By default, the stages *build*, *test*, and *deploy* exist. By default, jobs belong to the test stage unless specified
    - Lots of other features, see reference for details: https://docs.gitlab.com/ee/ci/yaml/README.html

Usando GITHUB

- https://github.com/garganti/esempio_ci_maven

- Configurazione nel file
- .github\workflows\maven.yml

- run: mvn --batch-mode --update-snapshots verify

Non solo tests

- Anche strumenti di analisi statica (che vedremo) possono essere integrati in CI
  - Esempio: violazioni di convenzioni del linguaggio
  - Violazioni delle dipendenze ..
  - …

Technical debt e CI

- Technical debt (also known as tech debt or code debt) describes what results when development teams take actions to expedite the delivery of a piece of functionality or a project which later needs to be refactored.
  - In other words, it’s the result of prioritizing speedy delivery over perfect code.
- Technical debt is a phrase originally coined by software developer, Ward Cunningham, who in addition to being one of 17 authors of the Agile Manifesto, is also credited with inventing the wiki. He first used the metaphor to explain to non-technical stakeholders at WyCash why resources needed to be budgeted for refactoring.
- **CI un modo per valutare/pagare il techinal dept**

Come introdurre CI in un processo

- **Source control version management**
  - Probably the most important foundational pillar of continuous integration is source control version management. It is used to communicate and resolve editing conflicts between multiple developers working in the same codebase. Source control version management comes in a variety of tools, the most popular being Git and Subversion. CI-as-a-service products center around the version control system.
- **Automated testing**
  - Most serious software projects include an additional code base that is not explicitly responsible for the business product and features. This secondary code base is a test suite and acts as a set of assertions that assures the primary code base is working correctly without bugs. During development, these tests are run by developers to validate that new code has not caused any regression on existing features. These test cases can also be run by extraneous tools to automate this validation process. CI service products will automatically run the test cases for a project on user-specified events. Generally, when a developer pushes code using the version control system an event will trigger the full test suite to run automatically.
- **Build automation**
  - “Builds” are the artifacts created to snapshot the current release version of a software project. Builds are distributed to end users through various networks. There is generally a set of scripted steps a project will take to create a build artifact. CI tools help streamline this build process with the use of automatic triggers from the version control system. One example trigger would be when new code is merged to the production branch of the codebase upload the build to a remote server for users to download.
- **Automated Deployments**
  - When builds are ready to be distributed they go through a deployment process. Depending on the project deployment can have a variety of outcomes. For example, web projects are deployed to publicly accessible web servers. During this deployment, the artifact that was generated in the build phase is copied onto the web servers. The deployment process for mobile and desktop vary and may involve uploading to a ‘store’ so users may download the app.