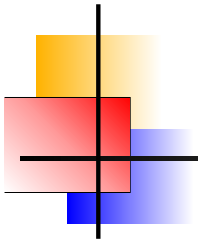


Programming Project:
Continuous Integration

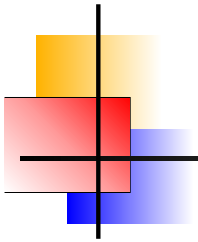
Guillermo Román Díez

groman@fi.upm.es

E.T.S. Ingenieros en Informática

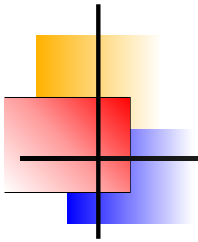
Universidad Politécnica de Madrid

2019-2020



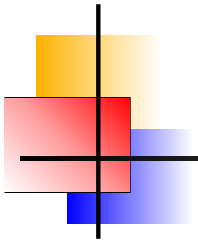
Continuous Integration

- Developing good software comes from carrying out good practices regardless of the particular technology used in the project
  - A great technology might help but it is not a *silver bullet*



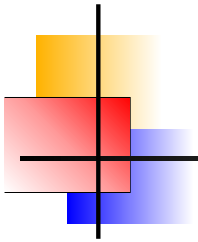
Continuous Integration

- Developing good software comes from carrying out good practices regardless of the particular technology used in the project
  - A great technology might help but it is not a *silver bullet*
- Software development requires an *infinite loop* of the following activities:
  - Planning and making changes: new features or maintenance
  - Test and observe the results
  - Based on the results, make corrections



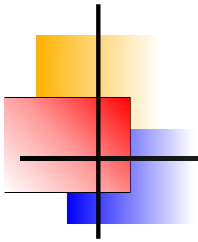
Continuous Integration

- Developing good software comes from carrying out good practices regardless of the particular technology used in the project
  - A great technology might help but it is not a *silver bullet*
- Software development requires an *infinite loop* of the following activities:
  - Planning and making changes: new features or maintenance
  - Test and observe the results
  - Based on the results, make corrections
- **Continuous Integration** (CI) aims at bringing together some tactics to ensure that, *after a change*, we will receive *immediate feedback* and know if we have broken something to fix it *immediately*



Continuous Integration

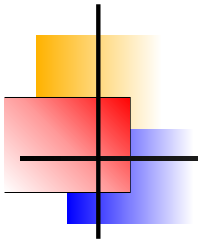
- Developing good software comes from carrying out good practices regardless of the particular technology used in the project
  - A great technology might help but it is not a *silver bullet*
- Software development requires an *infinite loop* of the following activities:
  - Planning and making changes: new features or maintenance
  - Test and observe the results
  - Based on the results, make corrections
- **Continuous Integration** (CI) aims at bringing together some tactics to ensure that, *after a change*, we will receive *immediate feedback* and know if we have broken something to fix it *immediately*
  - This rapid feedback allows to **immediately correct** and adjust the change
  - It **ensures the health** of our software through running a build after every change



A "bird's-eye view" of Continuous Integration

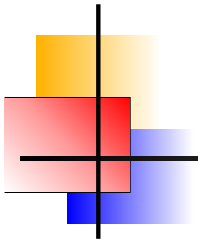
1. A programmer makes a change in the code
2. The programmer runs a private build and the tests against the newer code
3. The programmer commits (pushes) the changes to the repository
4. The CI server runs an integration build when the commit is detected
   - The server runs the integration tests
   - Applies the analyses to verify that the project adheres the quality standards
5. The team receives an email with the results of the integration build
   - The team inspects the report and checks whether the changes has a negative effect in the project

. . . and do it continuously



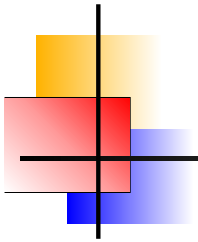
Continuous Integration Value

- CI **reduces risks**
  - Defects are detected and fixed sooner
  - The health of the software is tracked over time and can be kept under control
- CI **reduces repetitive manual processes**
  - Manual processes increases the risks
  - It guarantees that all steps of the process are executed following the correct order
  - Automatic processes starts after every commit (push) in the repository



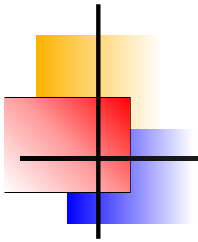
Continuous Integration Value

- **Generates deployable software** at any point in time (everywhere)
  - With CI you make small changes and integrate these changes with the rest of the code continuously
  - Problems are fixed as soon as possible and the code pass all tests throughout all phases in the project
- **Increases the confidence** in the software product
  - After every change all tests are passed, and thus, the team's confidence in the product developed is high
  - Delaying the integration process, programmers do not know the impact of their changes and are resistant to make changes



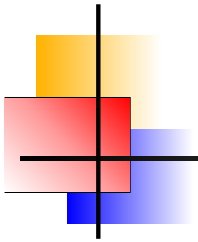
Continuous Integration Practices

1. Commit (push) code frequently
   - Make small changes: write the code and its corresponding tests
   - Do not wait to commit code and commit after each task
   - Frequent commits reduce the integration time and allow other users to use these changes
2. Don't commit broken code
   - All committed code must pass the tests
   - It is crucial to have fully automated tests that can be privately run before commit
   - Update your local copy before commit to run the tests with the last version of the code
3. Fix broken builds immediately
   - If the notification system reports an error it must be fixed immediately
   - As we are committing frequently, the error should be small and we can correct it quickly
   - Fixing bugs has maximum priority



Continuous Integration Practices

4. Write automated developer tests
   - Test must be fully automated
   - The team members and the integration server must be capable of running the tests in a fully automated way
5. All test and inspections must pass
   - Before a commit, 100% test must pass: tests are as important as compilation
   - Quality analyses must also be considered: when a change reduces the overall quality of the project, the team must immediately work on it
6. Avoid getting broken code
   - Do not get broken code as you will spend time in developing a work-around
   - The developers involved in the problem should be working in fixing it and commit the corrections to the repository as soon as possible



Continuous Integration with GitLab

- GitLab also works as continuous integration server
- A file `.gitlab-ci.yml` can be used to define the action to be done when a push is performed

```
maven_build:
  script: mvn test
```

  For instance, this command executes `mvn test` after each push
- The result of the build (success or fauilure) is sent by email to the users involved in the project
- By using the GitLab interface, you can access to the whole history of *builds* done in the project