COMP0104 Software Development Practice: Continuous Integration

Jens Krinke

Centre for Research on Evolution, Search & Testing
Software Systems Engineering Group
Department of Computer Science
University College London

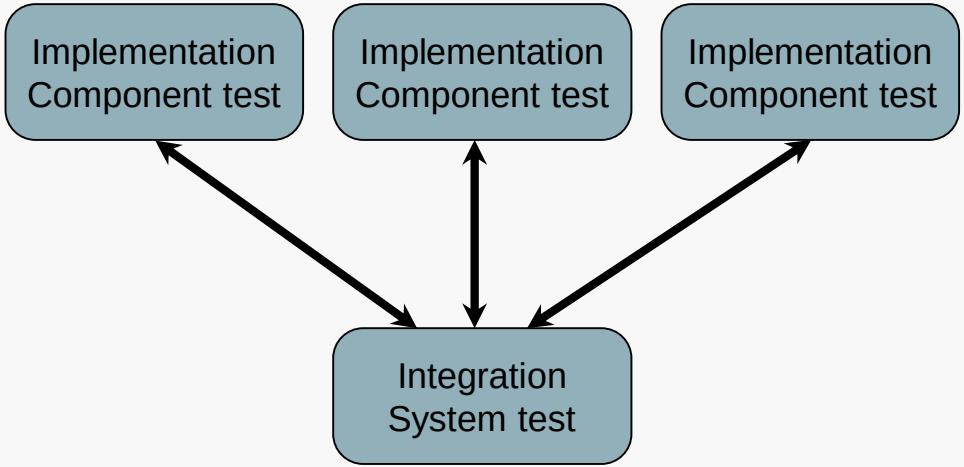
What is this module about?

- Version Control
- Building Programs
- (Testing)
- Analysing Programs
  - checking style
  - detecting clones
  - finding problems

Integration into an IDE (Eclipse)

Tool	Works on...	Needs...
Diff	Source code	-
Subversion/Git	All artefacts	Properties
Make/ANT/Maven	All artefacts	Makefile
JUnit	Source code	Test code
Checkstyle	Source code	Configuration
Simian	Source code	Configuration
…	…	…
…	…	…

Old-style development



- Integration and system test could take a huge amount of time.
- Problems discovered during integration could block other teams

Component Test vs Unit Test

Component Testing and Unit Testing
are usually used interchangeably.

The main difference is the scope:

- Unit Testing is more low-level,
  units are not subdivided into other components.
- Units can be any fragment of code,
  usually methods and classes
- Components are classes, packages,
  up to complete sub-systems

Integration Testing

- Progressive linking and testing of programs or modules to ensure their proper functioning in the complete system.
- Testing in which components are combined and tested to evaluate the interaction among them.

xUnit Frameworks

- The meaning of unit tests and integration tests has become blurry and overlaps.
- Unit testing and integration testing is done within xUnit frameworks.
- Fowler uses the term **solitary** and **sociable** unit tests.

Do integration as soon as possible!

- Integrate all components right from the start.
- Replace not yet implemented components by mock-ups.
- Always have a fully integrated version, even if it has only partial functionality.

Incremental Integration Approaches

- Top-Down
- Bottom-Up
- Vertical Slice
- Sandwich
- Risk-Oriented
- Featured-Oriented
- T-Shaped

Integration Approach and Order

- The integration approach affects the order in which components are developed.
- A well-thought-out integration order reduces testing effort and eases debugging.
- No integration approach is best in every case, the best approach varies from project to project.

Integration per developer

- Having a fully integrated version per developer does not guarantee no problems for others.
- The integrated version depends on environment, configuration, available plugins, etc.
- The integrated version may not be deployable.
- How to get the version out of the IDE?

Automation is everything...

- Most of the presented tools can be automated and integrated into an IDE.
- The automation can be used to always have a fully integrated individual workspace (a running and tested program).
- Only commit changes when you are sure not to break the fully integrated workspace.
- What happens with the changes of the other team members? And the complete project?

Daily Build and Smoke Test

- Every file is compiled, linked, and combined into an executable program every day.
- The program is then put through a “smoke test”, a relatively simple check to see whether it runs.
- It is easy to pinpoint why a product broke: it happened since the last successful build.
- Daily build without smoke tests have little value.
- A broken build should be the exception, breaking the build may be penalised.

heb@Wikimedia Commons, CC BY-SA 2.5

Challenges of Daily Build and Smoke Test

- Keep the build fast
  (Windows 2000, 50mLOC, 19h for a build)
- Managing daily build and smoke test
  may become a large task on its own.
- What is added to the daily build?
  - adding every new feature
    increases the risk of breaks
  - delaying new features makes the build outdated
- Code is tested twice
  (by the developer and the daily build)

Continuous Integration (CI)
Martin Fowler:

- Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily.
- Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

Continuous Integration (CI)
Martin Fowler:

Many teams find that this approach leads to significantly reduced integration problem and allows a team to develop cohesive software more rapidly.

Goals of CI

- discover bugs early
- have a cohesive team
- high quality code base
- high project visibility
- always have a deployable version

Results

- Better quality software
- Frequently tested software
- Tests are parallel to development
- Integration and release builds are easy and fast

How to build

- After every check-in to the repository.
- Build with a single script, using MAKE, Maven, ...
- The build script is part of the source repository.
- Run tests and checks.
- Tests and configurations
  must be in the source repository.
- What is not in the repository, does not exist.
- Best on a dedicated CI server

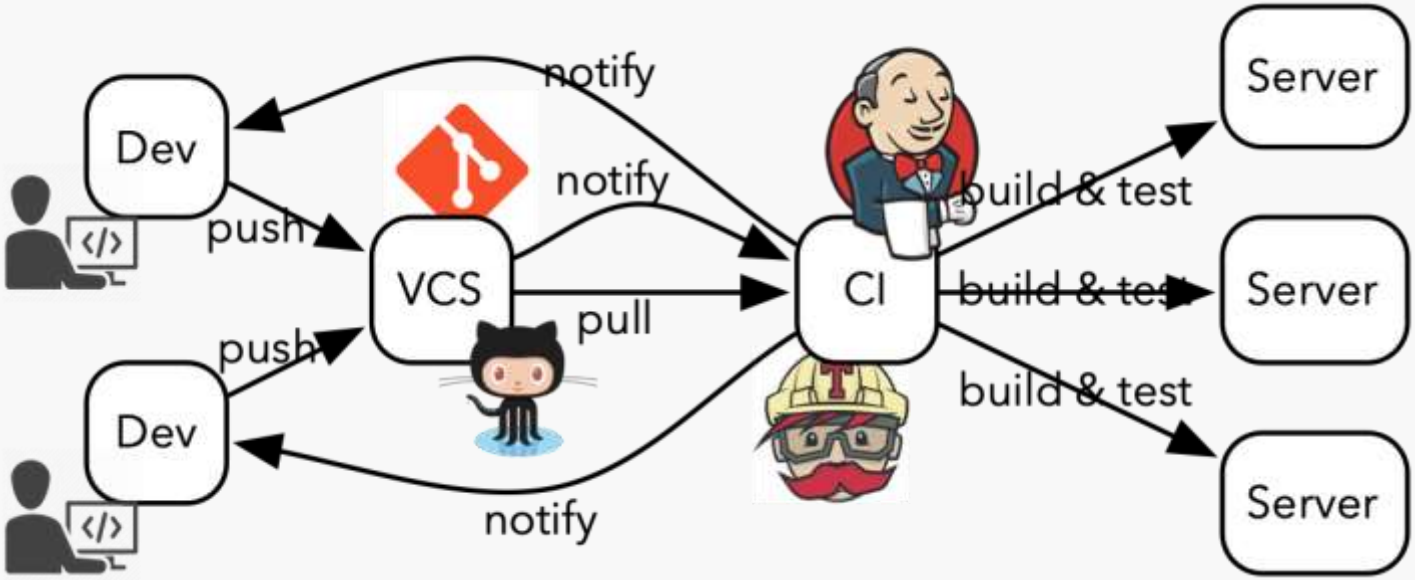
Available CI servers

- Cruise Control
- Tinderbox
- Apache Continuum
- Hudson
- Jenkins
- Travis
- …

Travis vs. Jenkins

Jenkins	Travis
Open Source	Commercial
Application	Service (Hosted)
Configuration	Convention
Flexible	Easy to use

Continuous Integration in Practice



Feedback is essential

Developers are notified as soon as a build breaks.

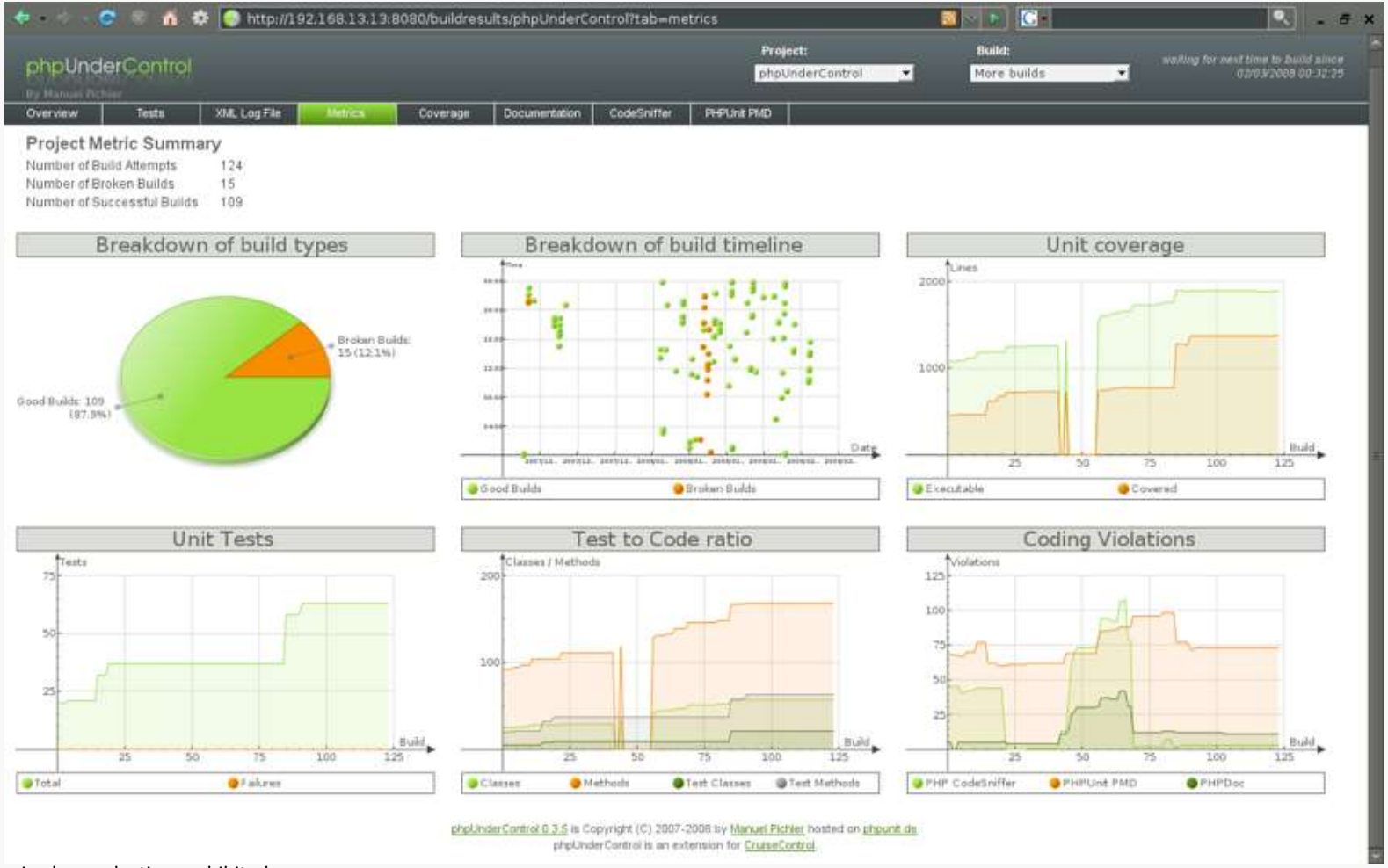
- Fixing the problem has the highest priority.
- The same holds for the tests.
- A broken build should be fixed within a short time.
- Don’t leave after a check-in, wait for the build to complete.
- Don’t go home if your check-in broke the build!

Measure the software

Identify key metrics and track them visually

- Number of failed tests
- Number of classes, LOC
- How many style violations?
- How many code smells?
- Duration of build?

Not only nice graphs for managers...



Principles of Continuous Integration

- commit early and often
- never commit broken code
- fix build failures immediately
- fail fast
- act on metrics
- build in every target environment
- create artefacts from every build

Problem: Keep the build fast!

- Optimise your test suite (COMP0103)
- Separate into staged builds
  - check-in build and test
  - full test (overnight)
- Separate into multiple projects
  - use external projects ‘as-is’
  - use the last fully tested external project
- Concurrently execute tests and checks.

Continuous Delivery

- Continuous Integration (CI) ensures that changes are integrated and checked so that an artefact can be built.
- Continuous Delivery (CD) extends CI by ensuring that the built artefacts can be deployed and released.
- CD is harder to adopt than CI
  as it needs extensive automation:
  - Deployment into testing environment
  - Extensive testing in different environments
  - Actual deployment: Continuous Deployment

Pipelines

- The steps of CI/CD are realised in a pipeline.
- Some steps are usually realised through a build infrastructure like Maven or Gradle.
- Most deployment steps are realised through separate tools or scripts.
- The CI pipeline is applied to all branches, including feature branches.
- The CD pipeline is applied only to the trunk (master and development branches, not feature branches).

Concepts

- The integration approach affects the order in which components are developed.
- Continuous integration can help to build better quality software faster, with more confidence.
- Continuous integration builds a fully integrated, tested, and checked system after every build.
- Problems are discovered early, and must be fixed fast.
- The system is build fresh from the repository, outside an IDE, best on a dedicated server.