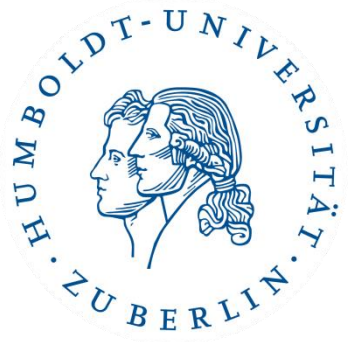


Semesterprojekt
Implementierung eines Brettspiels
(inklusive computergesteuerter Spieler)

Wintersemester 16/17

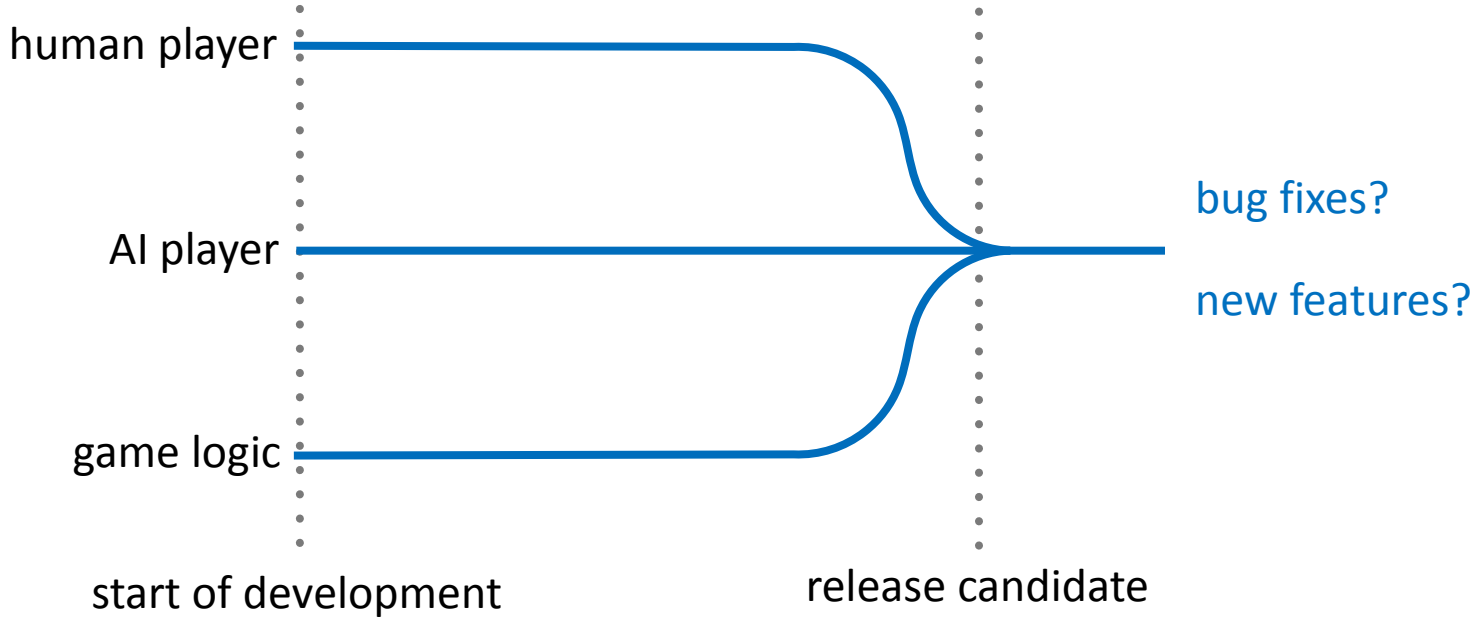
Unit Testing & Continuous Integration

Patrick Schäfer
patrick.schaefer@hu-berlin.de

Marc Bux
buxmarcn@informatik.hu-berlin.de

How Software Used to Be Developed

- software developed in teams
- software is divided in modules
- modules are assigned to teams
- modules have to be integrated at some point
- integration is done manually at release time



What Are the Issues?

- effects of changes (bugfixes, new features) are hard to predict
  → Unit Tests

- no feedback (by the product owner) possible before release
  → Continuous Integration

- unnecessarily complicated integration process
- bad software quality
- stress and frustration (“integration hell”) towards the end of a project

Unit Tests

unit test: systematic, automated test of a software component

- unit: smallest testable part of an application
  - in object-oriented programming, this is usually a method (or a class)
- the unit is tested in isolation (of other units)

- usually written in the same language as the tested software
- tests should be written by the developer (of the unit)
- can be written prior or concurrent to unit development
- tests can be successful or fail

Advantages of Unit Testing

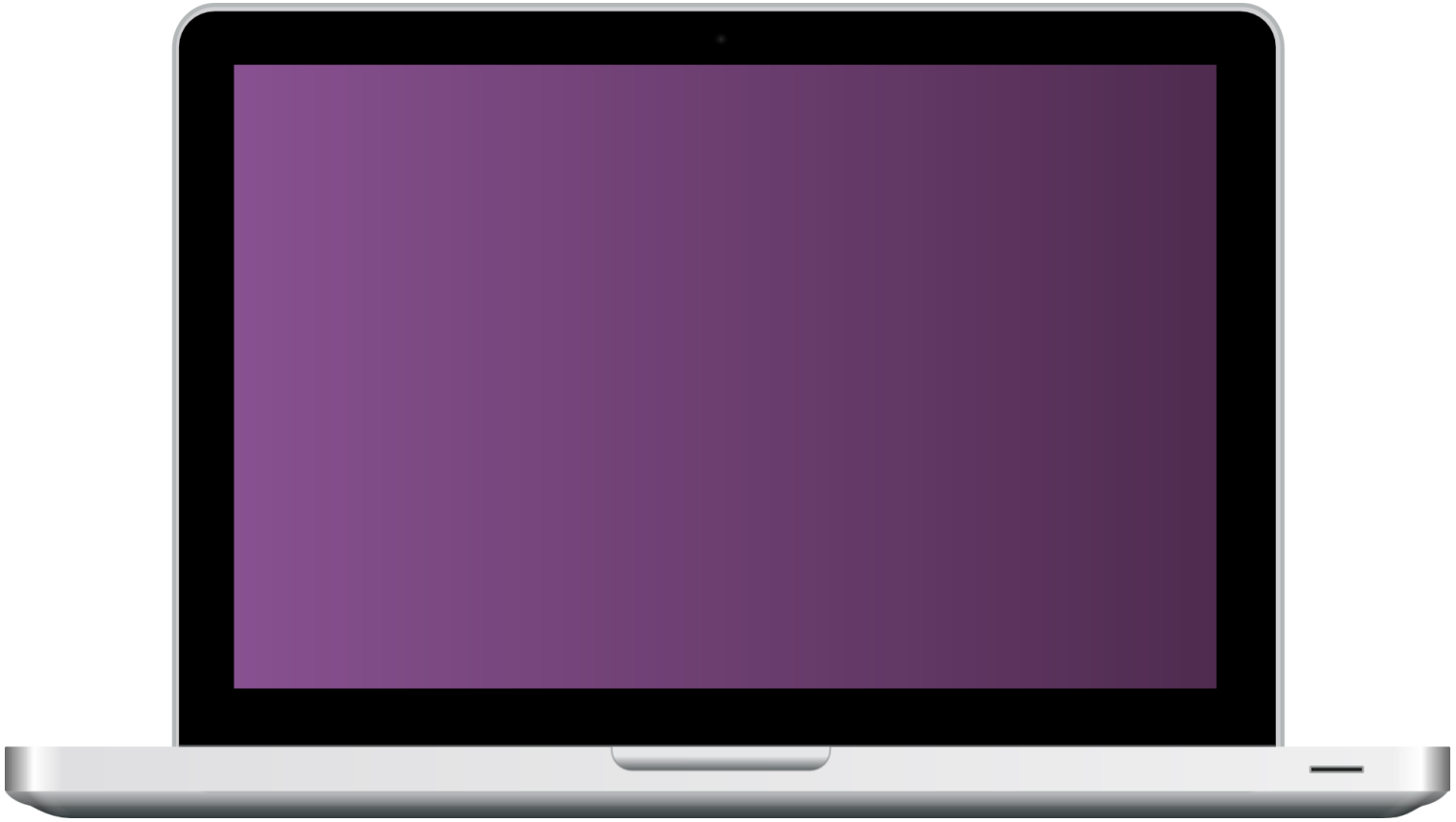
unit tests …

- … facilitate the design of robust code (bugs make it through only if the unit and its test are poorly designed)
- … provide immediate feedback on the effect of changes in the code base
- … serve as a to-do list subsequent to changes in the code base
- … help define what a piece of code is (and isn't) supposed to do

Best Practices

1. test the complete intended behavior of the unit, including
   a) expected cases (e.g., sort an unsorted array)
   b) special cases (e.g., sort already sorted array)
   c) boundary conditions (e.g., sort empty array)
2. test every behavior only once (no redundant test)
3. test only one unit at a time
4. design tests independent of the application‘s state
5. design tests independent of external resources
6. name unit tests clearly and consistently
7. whoever breaks a working unit is responsible for fixing it

Demo: Unit Tests



Repository:
https://github.com/hu-berlin-semesterprojekte/cidemo

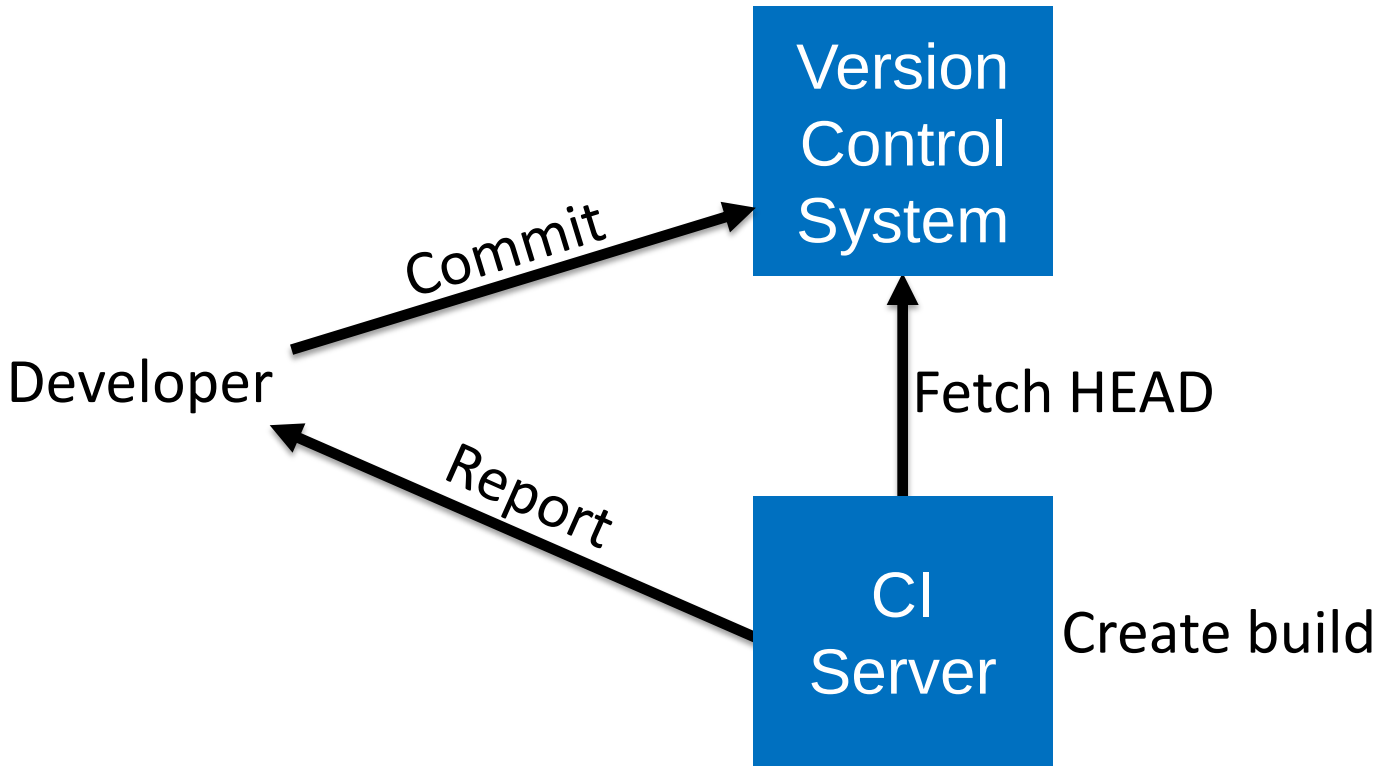
Continuous Integration (CI)

continuous integration: automatically test and merge all units into an integrated software (multiple times a day)

- every change (e.g., git push) in the software triggers a new build
- unit tests are executed to determine the success of a build
- gives feedback in form of reports

- requires version control and build automation for downloading dependencies, compiling code, and running tests
  - build automation tools for Java: Maven, Ant, Gradle
- builds can be successful or fail

CI in Practice



Advantages of Continuous Integreation

with continuous integration, we …

- prevent “integration hell” early
- always know the latest stable version of our software
- instant feedback if a developer‘s work in progress breaks the stable version
- can automatically test different setups
  - different databases
  - multiple versions of 3rd party libraries
  - different configurations

Best Practices

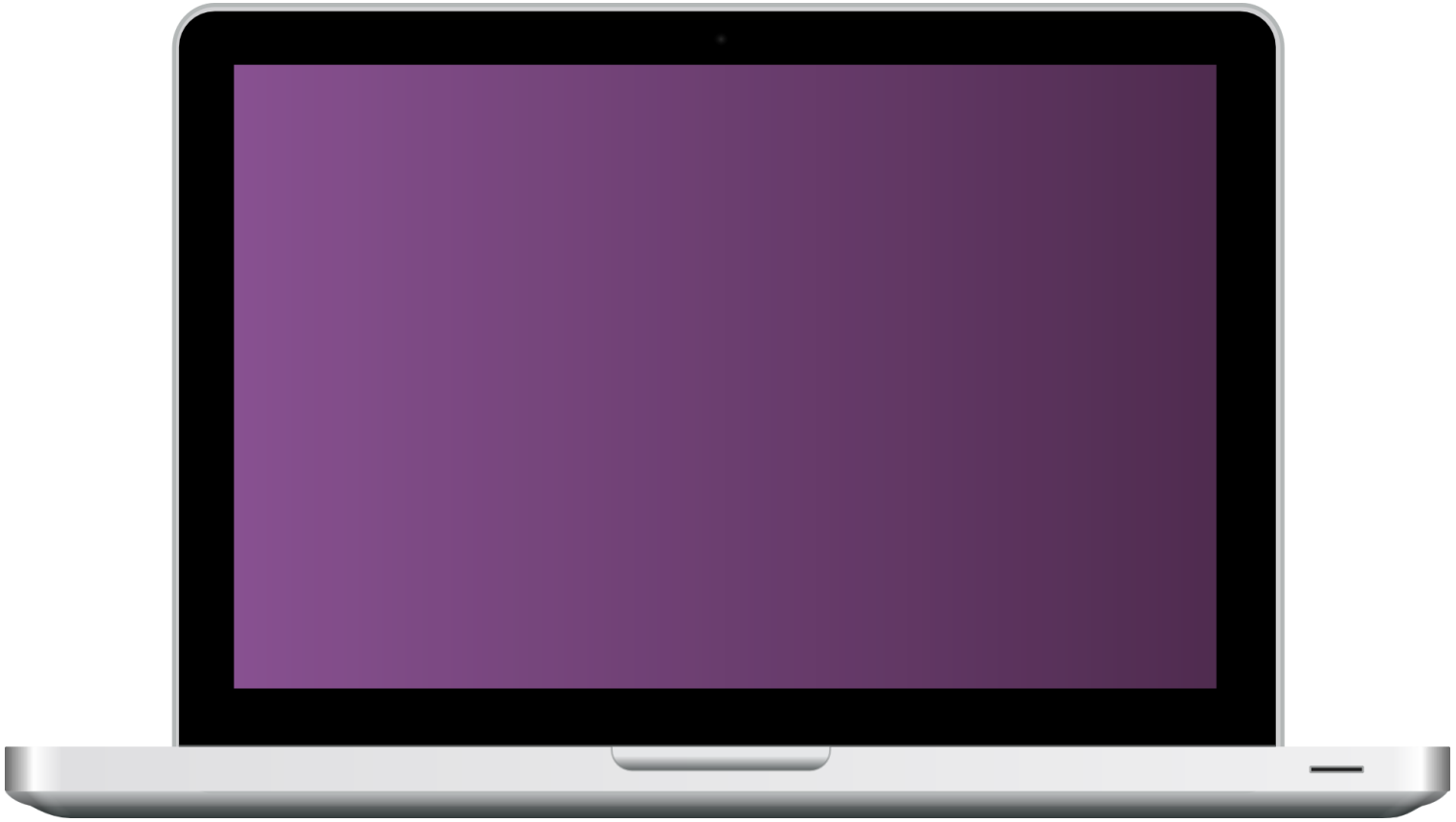
1. design meaningful unit tests for your software modules
2. commit frequently; keep iterations small
3. keep your tests fast; keep the build fast
4. don’t (ever) commit into a stable branch when the build is broken

Travis CI

- open-source continuous integration service / server
- website: https://travis-ci.org
- coupled to GitHub

- easy to set up:
  1. sign in using your GitHub account
  2. select repositories that Travis should build
- build is configurable via .travis.yml file
  - YAML is a popular data serialization file format, similar to XML or JSON

Demo: Continuous Integration



Repository:
https://github.com/hu-berlin-semesterprojekte/cidemo

Further Reading

- unit tests in Java using JUnit:
  http://www.frankwestphal.de/UnitTestingmitJUnit.html
- build automation in Java using Maven:
  https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

- unit tests in Unity using Unity Test Tools:
  https://unity3d.com/learn/tutorials/topics/production/unity-test-tools
- continuous integration in Unity using GitHub and Travis CI:
  https://jonathan.porta.codes/2015/04/17/automatically-build-your-unity3d-project-in-the-cloud-using-travisci-for-free/

Next steps

- familiarize yourself with unit tests & continuous integration
  - further reading (→ last slide)
  - start testing and integrating (→ user story “Continuous Integration”)
- this week (w/o POs)
  - finalize tasks in your sprint backlog (incl. tests, code review)
  - twice: “Daily” Scrum
  - mid-week: technical refinement for new user stories
- before Monday, 14:00 (w/o POs): Sprint #2 Planning
- next Monday (w/ POs), 14:00
  - Sprint #1 Review Meeting; bring a laptop & presentable prototype
  - Sprint #2 kickoff; present your Sprint Backlog
  - short talk on coding guidelines?
- Questions?