Agile Quality Assurance

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

AMOS F01

Agenda

1. Test-first programming
2. Test-driven development
3. Pair programming
4. Pre-commit code review
5. Build management
6. Continuous integration
7. Continuous delivery

1. Test-First Programming

Test-Driven Development in Context

1. Tests and testing
2. Test-first programming
3. Test-driven development

Tests and Testing

- Testing is a process
  - that tests some concern (the concern “under test”)
  - for correct and expected operation
  - according to a specification
  - usually as part of quality assurance
- Tests can be manual or automated
- Tests verify against a given specification
- Tests increase confidence in correct functioning
- However, tests can never proof a program correct

Types of Tests [1]

- **Components tests** (a.k.a. unit tests)
  - Focus on testing one component out of context
- **Acceptance tests** (a.k.a. functional tests)
  - Focus on testing one cross-cutting functionality
- **Integration tests** (a.k.a. system tests)
  - Focus on testing end-to-end system integrity

[1] This is a simplification for the purposes of this course.

Tests and Testing Terminology

- **Test (case)**
  - A single test for some particular aspect of the software, succeeds or fails
- **Test suite**
  - A set of related tests that cover a particular domain of the software
- **Test set-up**
  - The data and preparation necessary to run a test as intended
- **Test result**
  - The result of running a test, either succeed or fail, or a test error
- **Test harness (framework)**
  - A software, like JUnit, that is used to simplify the implementation of tests

Test-First Programming [B02]

- Test-first programming is a practice in which developers
  - Write a test before they implement the actual functionality and
  - Iterate over an “add new or enhance test, make test work” loop

- Functionality is a by-product of making the tests work
  - Test-first programming
    - clarifies code functionality and interfaces
    - improves code quality through second use scenario
    - builds up test suite for continuous integration (later)

Test-First Rules 1 / 2

Only write new code

when a test fails

Then, eliminate waste

Test-First Rules 2 / 2

1. **Red**
2. **Green**
3. **Refactor**

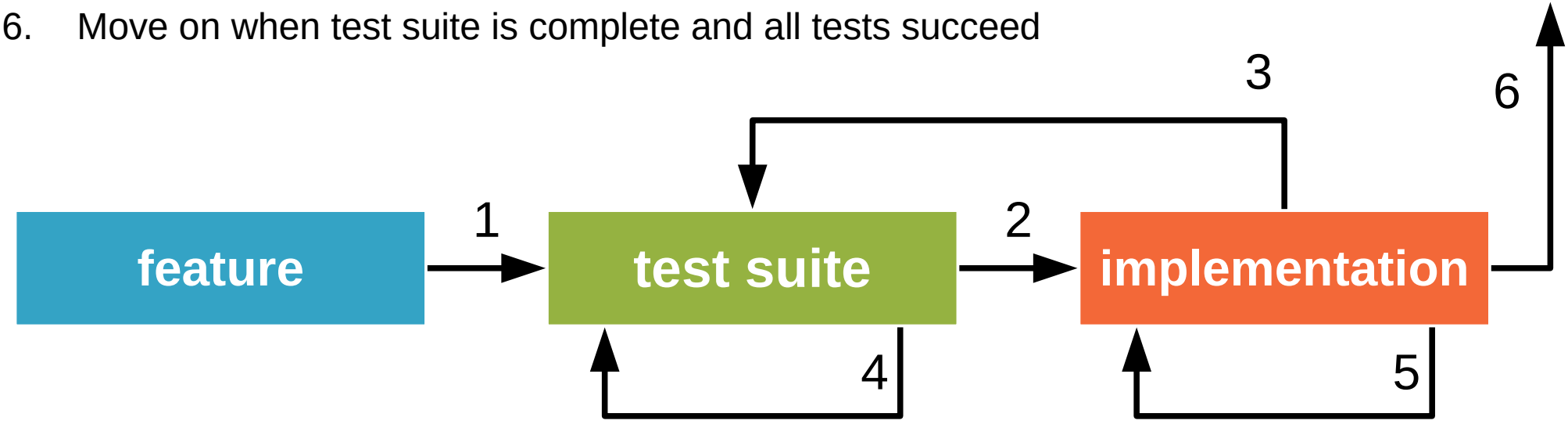
2. Test-Driven Development

Test-Driven Development (TDD) 1 / 3

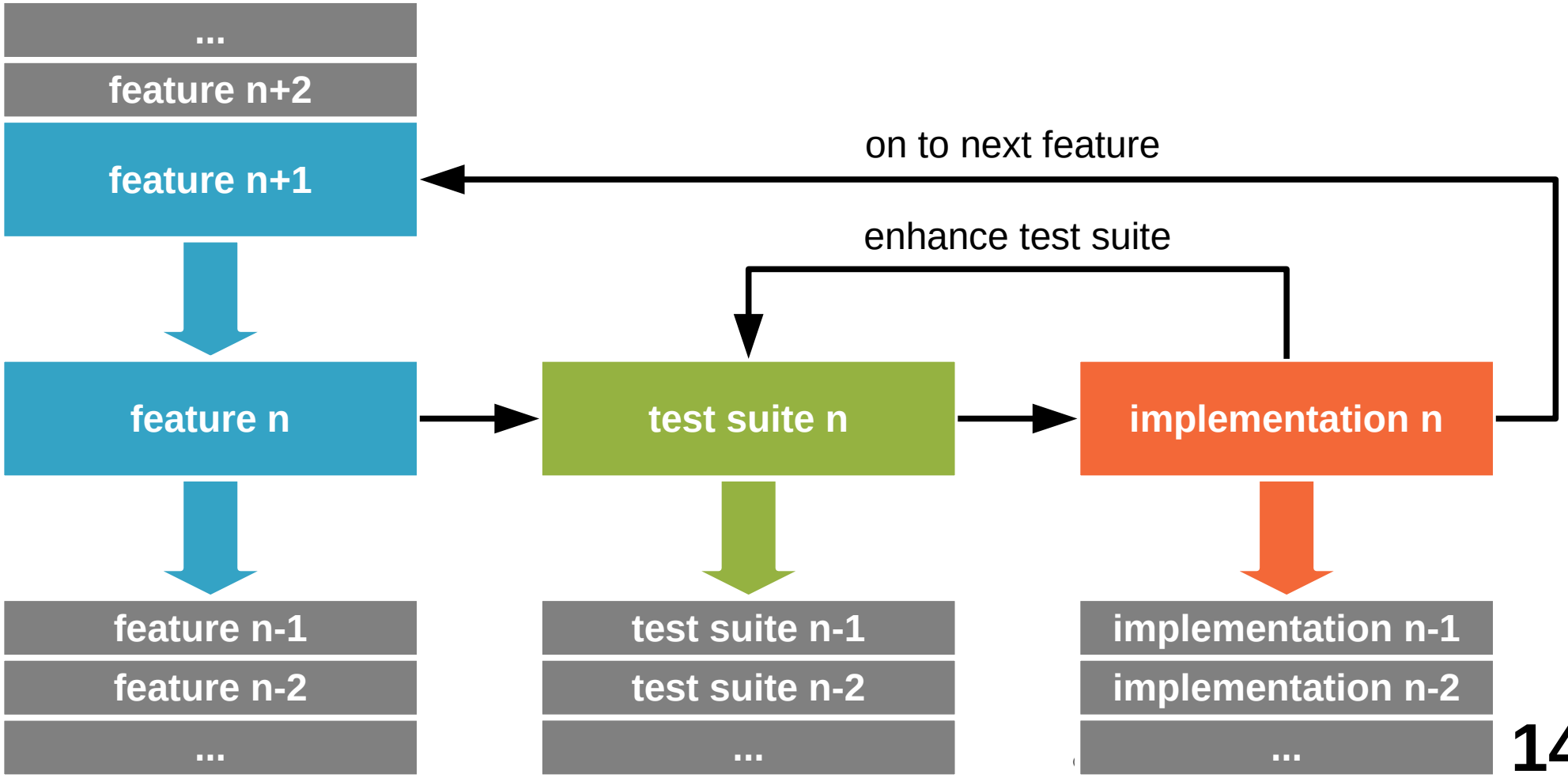
- Test-driven development
  - Is a minimal development process based on test-first programming
  - Turns feature requests into implementations
- Purpose of test-driven development
  - To grow the product incrementally and steadily
  - To be able to release after every feature implementation

Test-Driven Development 2 / 3

1. Translate partial or full feature description into test suite
2. Implement feature to fulfill (“green-bar”) test suite
3. Revise test suite from new insights
4. Refactor test suite to keep design and code clean
5. Refactor implementation to keep design and code clean
6. Move on when test suite is complete and all tests succeed



Test-Driven Development 3 / 3



3. Pair Programming

Code Review

- Definition and purpose
  - “Code review is systematic examination […] of computer source code.
    - It is intended to find and fix mistakes overlooked in the initial development phase,
      - improving both the overall quality of software
      - and the developers’ skills.
  - Reviews are done in various forms such as
    - pair programming,
    - informal walkthroughs, and
    - formal inspections.” [1]

[1] Adapted from https://en.wikipedia.org/wiki/Code_review [DR]

When to Review Code?

1. In the moment
2. Before commit
3. At another time
4. Before release

Agile Code Review Practices

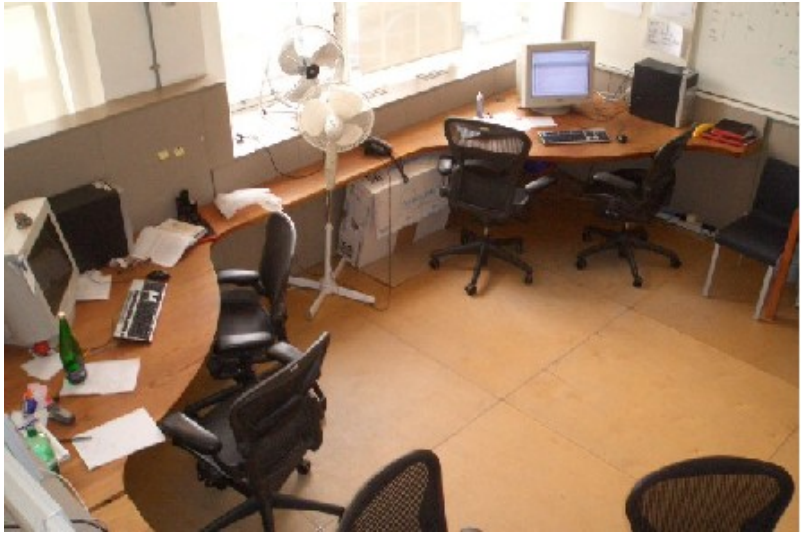
1. Pair programming
2. Pre-commit code review

Pair Programming

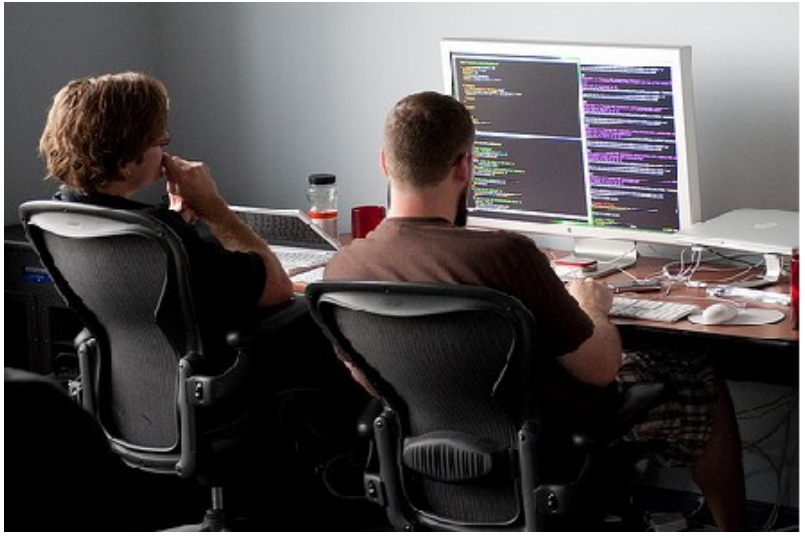
- Definition
  - Is programming carried out by pairs of programmers
  - One programmer implements, and the other programmer reviews
  - Effectiveness is debated; empirical studies show conflicting evidence
- Purpose
  - Quality assurance
  - Collaborative learning
  - Knowledge sharing
- Synonyms
  - Programmer and reviewer
  - Driver and co-driver
  - Pilot and navigator

Pair Programming (Practices)

- **Process**
  - Find comfortable partner
  - Switch roles often
  - Communicate regularly

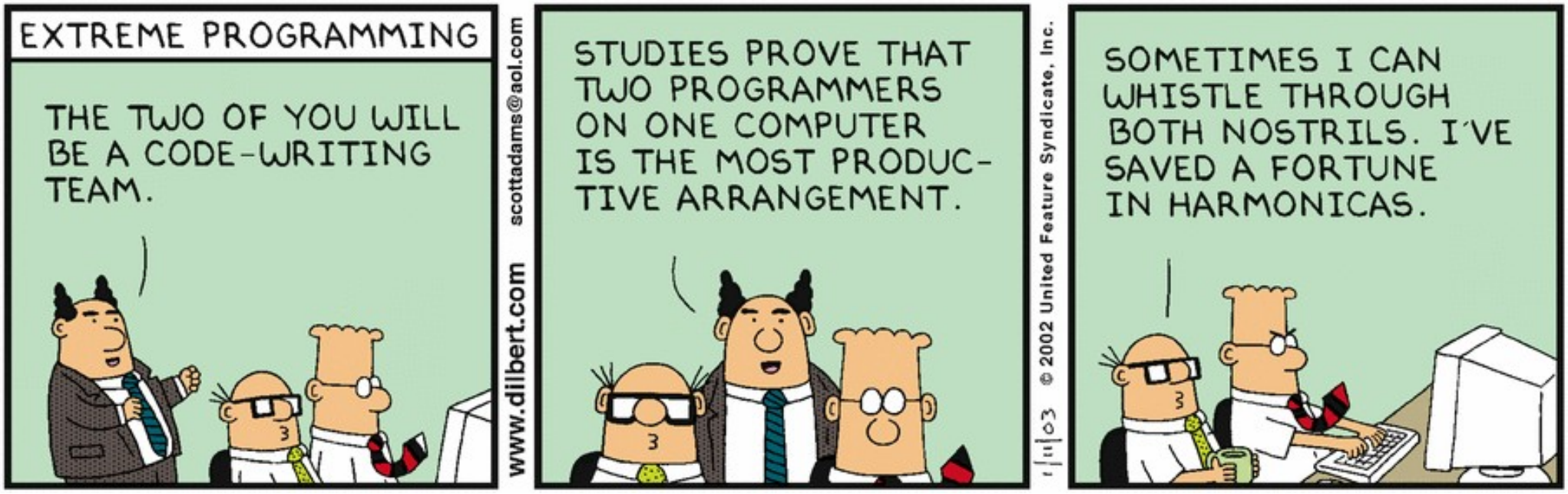


- **Advice**
  - Don’t force it for small stuff
  - Don’t overheat, take a break
  - Switch partners at times



Dilbert on Pair Programming 1 / 2

Dilbert on Pair Programming 2 / 2



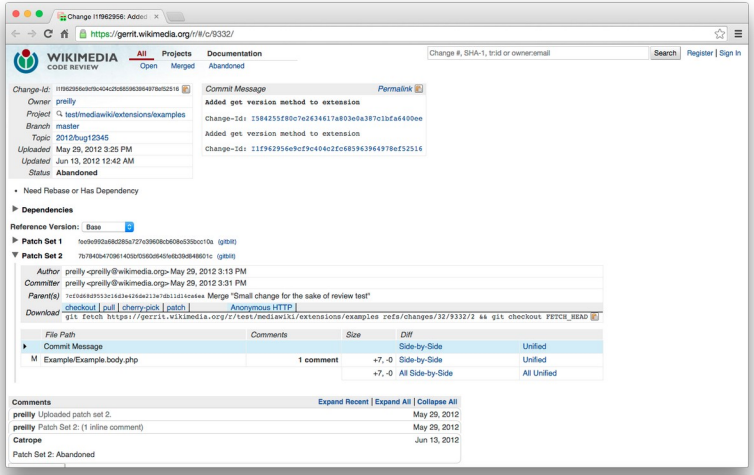
4. Pre-Commit Code Review

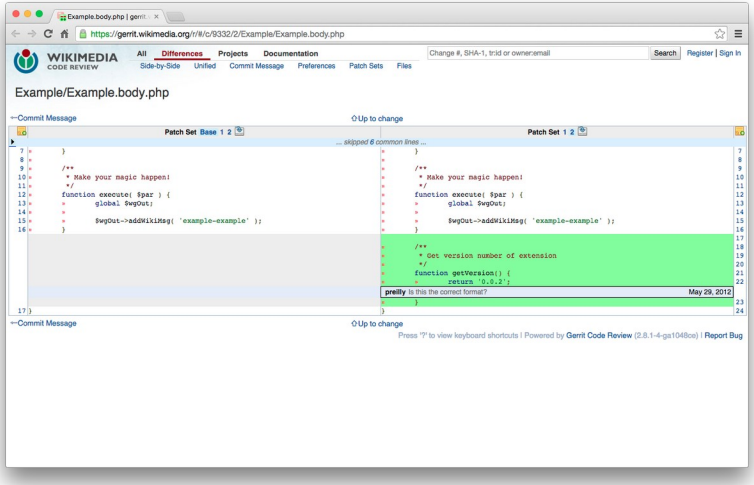
Pre-Commit Code Review

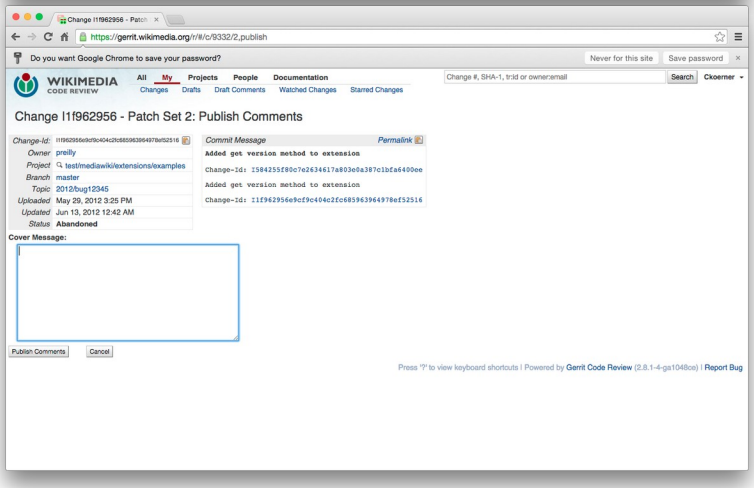
- Code review
  - Is the (peer) review of source code for quality criteria
  - Reviewer has accept or reject responsibility
  - Cf. “Vier-Augen-Prinzip” (in German)

- Pre-commit code review
  - Is the review of source code before it gets committed to a team repository
  - Typically facilitated by a software tool, e.g. Gerrit
  - May lead to back and forth between developers until “LGTM”

Example Code Review with Gerrit

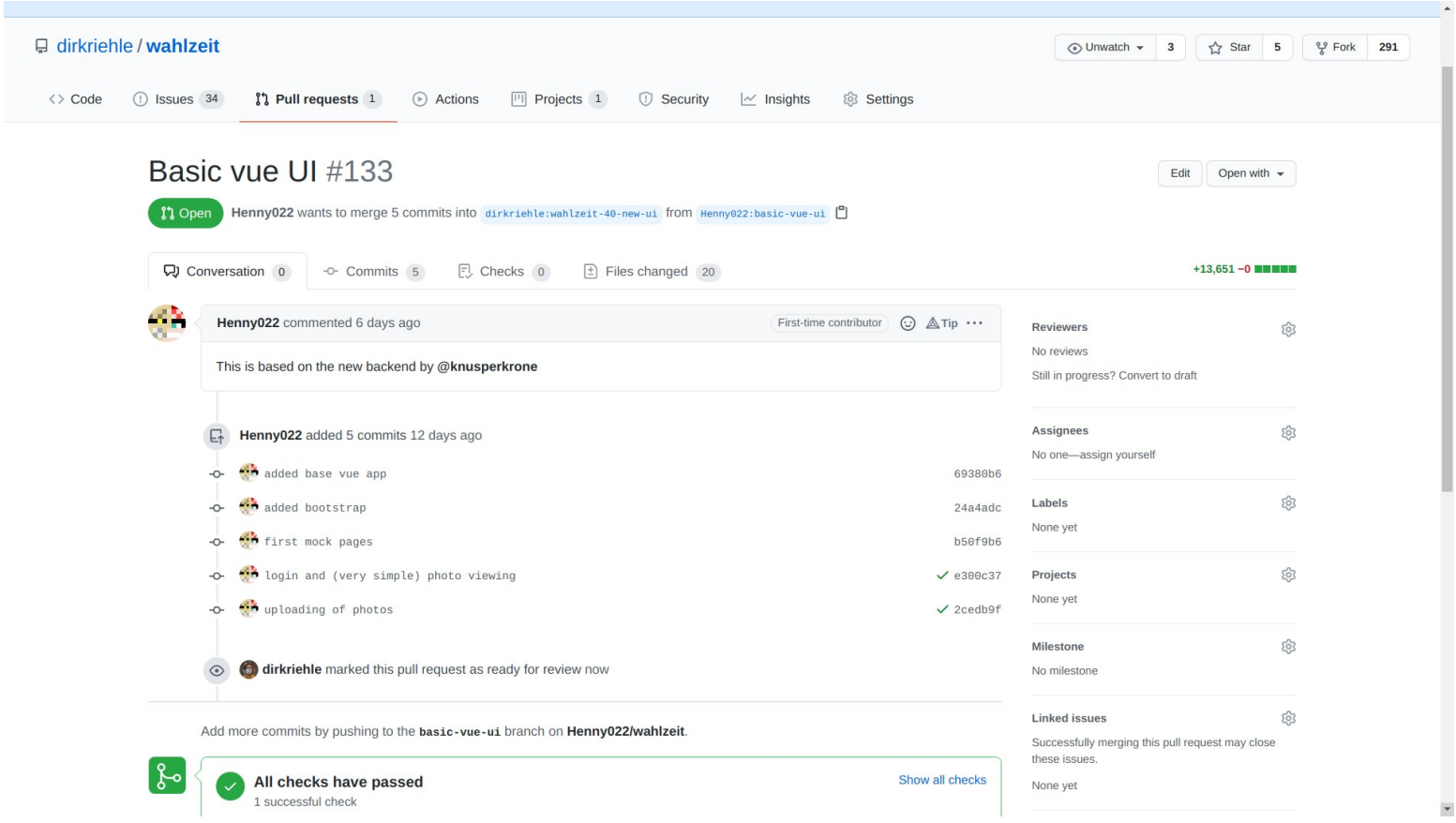






[1] See https://www.mediawiki.org/wiki/Gerrit/Tutorial

Example Code Review in Distributed Work (Merge Requests)



Benefits of Pre-commit Code Review

- Collaboration
  - Improves knowledge sharing and teamwork
  - Makes it easier to establish topics like security
- Quality assurance
  - Leads to more disciplined developers
  - Prevents (some) errors before they happen
  - Raises overall quality standards
- Feeling of responsibility
  - Specifically, supports collective code ownership
  - Strengthens overall feeling of responsibility

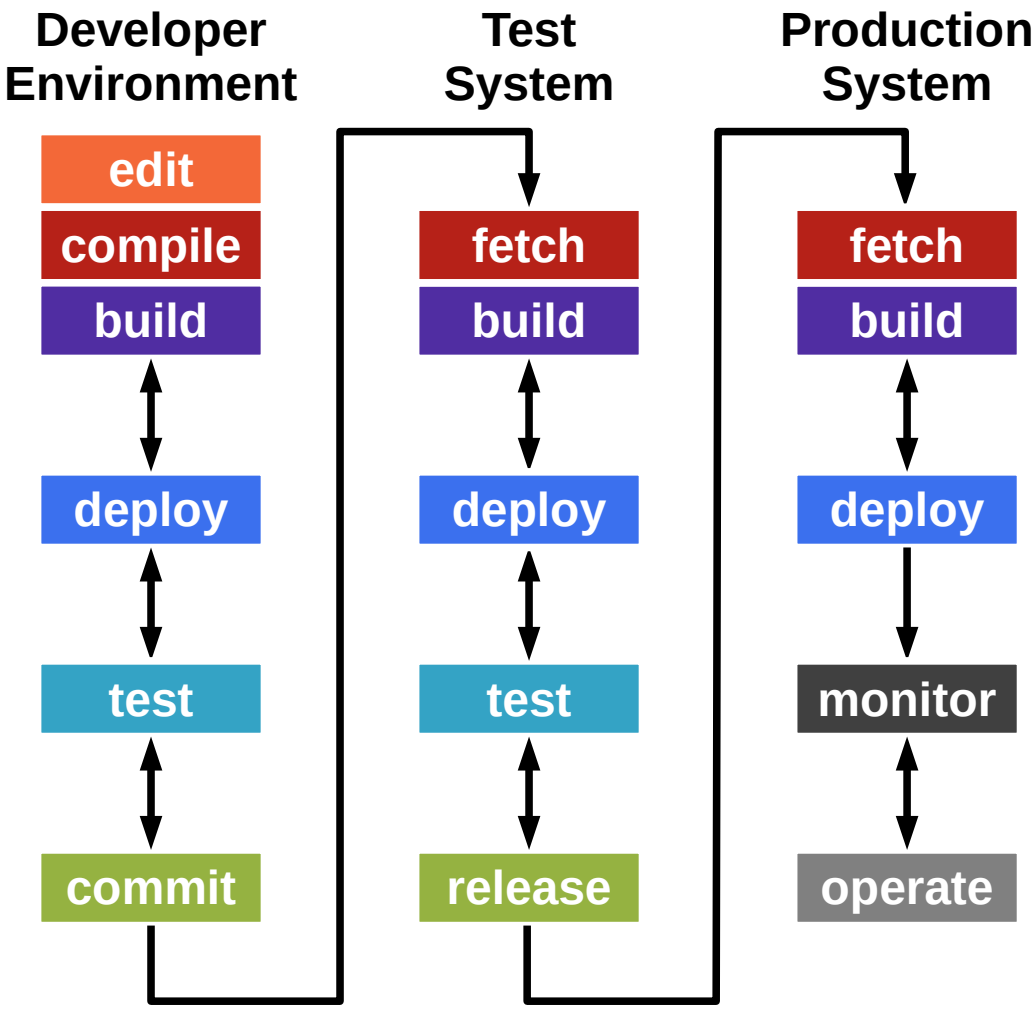
Agile vs. Open Source Code Review

- **Agile methods**
  - Programming guidelines
    - Code reading >> writing
    - Make it easy to get acquainted
  - Collective code ownership
    - Feature-oriented development
    - Typically co-located development
    - Everyone has write access
  - Pair programming
    - Changes are reviewed directly
    - Everyone is a peer

- **Open source**
  - Programming guidelines
    - Code reading >> writing
    - Showing respect for project
  - Individual code ownership
    - Component-oriented development
    - Typically distributed development
    - Strictly regulated write access
  - Patch review
    - Changes are submitted for review
    - Two-class reviewing hierarchy

5. Build Management

Professional Development Cycle



- QA engineer
  - Fetches code
  - Builds full system
    - Deploys in test system
  - Tests full system
    - Automated and by-hand
      - Component tests
      - Acceptance
      - Integration tests
  - Deploys full system
  - Operates system

Build Process

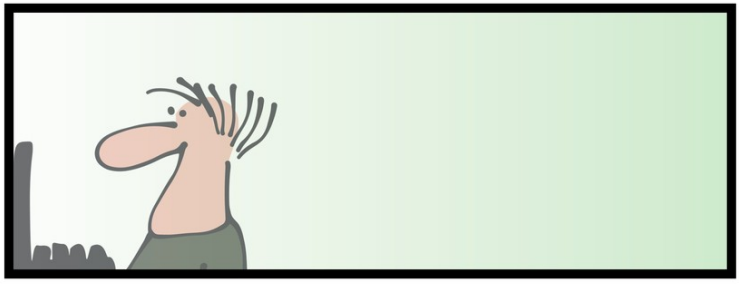
- Definition
  - Is the defined and (ideally automated) process of deriving an installable product from its source artifacts
- Purpose
  - Defines a standard environment
  - Provides developers with setup
  - Defines clear commit rules
  - Manages test data etc.

Developer Responsibilities

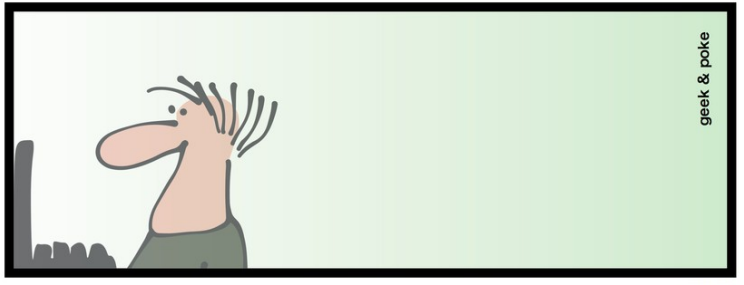
- Only work in defined standardized environment
  - This ensures no subtle differences to central build process
- Only commit / push code that compiles and works
  - “Breaking the build” through non-compiling code hurts team
- Only commit / push code that passes all the tests
  - Failing tests quickly degenerate system (hard to catch-up)

DEVELOPMENT CYCLE

FRIDAY EVENING EDITION



COMMIT



PUSH

RUN

Build Asset Management

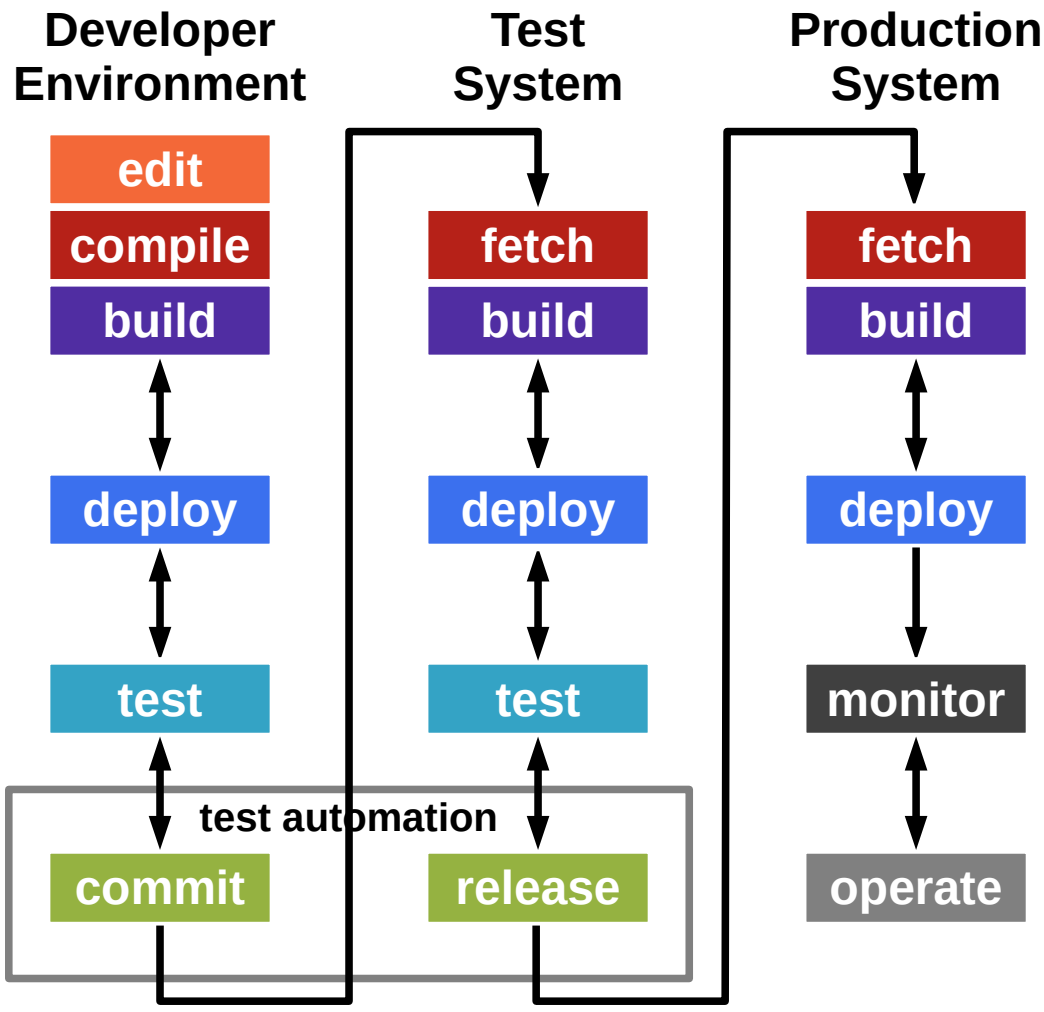
- Management by hand
  - Libraries are curated by hand
- Automated management
  - Libraries are pulled in automatically
- Bill of materials
  - Possibly generated automatically

Quiz: Supply Chain Effects

- Last week you delivered your product release to your client. The phone rings and an angry client is on the line, complaining about missing their schedule. What might have gone wrong?
  1. The client's development team objected
  2. The client's quality assurance unit objected
  3. The client's legal team objected
  4. All of the above

6. Continuous Integration

Continuous Integration Development Cycle



- Release (QA) engineer
  - Fetches code
  - Builds full system
    - Deploys in test system
  - Tests full system
    - Automated and by-hand
      - Component tests
      - Acceptance
      - Integration tests
  - Deploys full system
  - Operates system

Test Automation

- Test automation ...
  - automatically carries out all available tests
    - Component tests (unit tests)
    - Acceptance tests (functional tests)
    - Integration tests and system tests
  - provides feedback to development and QA

Continuous Integration

- Continuous integration (CI) is a code integration process
  - Upon trigger (commit to official repository)
    - the system under construction is fetched, built, deployed, and tested
    - in a fully automated way (no human intervention)
  - Feedback upon system status is provided to both
    - developers and
    - managers
- The purpose of continuous integration is to
  - always know where you are standing with respect to the project
  - ideally improve quality such that you can deploy at any time
- Continuous integration requires test-driven development

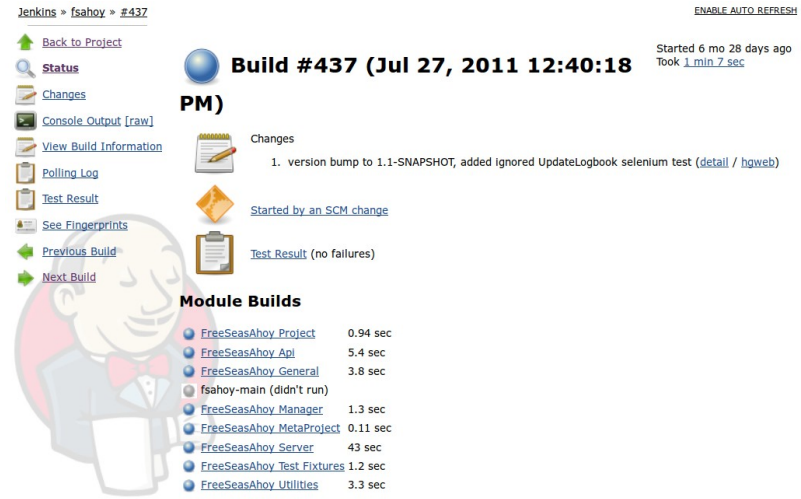
Continuous Integration and Lava Lamps

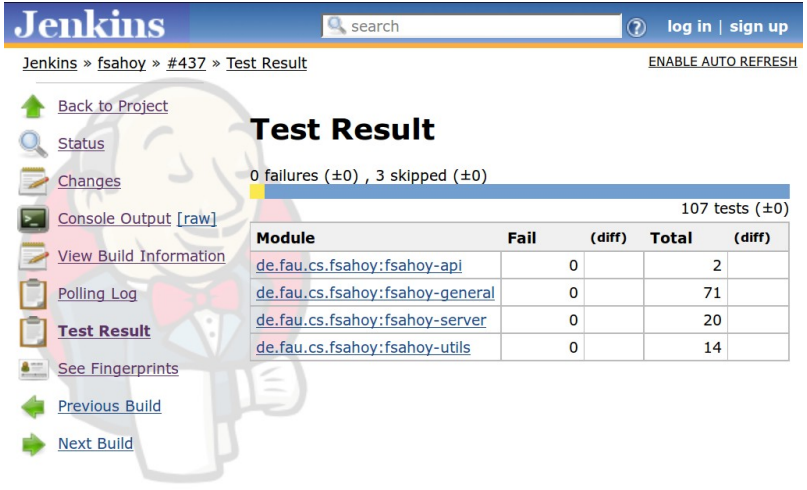
In the early days, lava lamps were used to signal whether the project could be deployed to production or not.

Continuous Integration Dashboards

Continuous integration quickly evolved to build and test status dashboards hung on office walls for everyone to see.

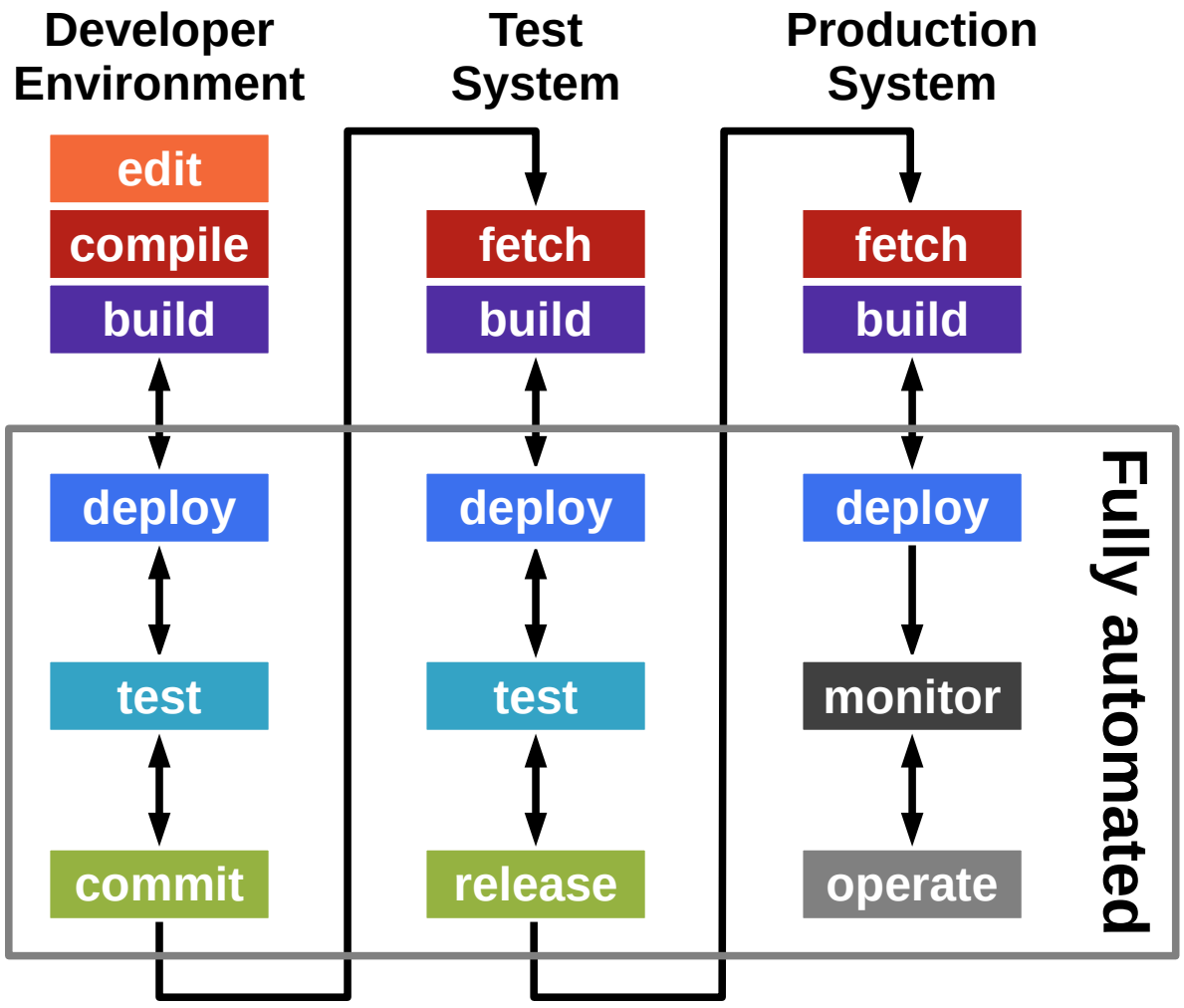
Example CI Dashboard (Jenkins)





7. Continuous Delivery

Continuous Deployment Development Cycle



- Fully automated
  - Compile and build
  - Deployment
    - To test environment
    - To production
  - Test execution
- Partially automated
  - System monitoring
  - Automated rollback
- Human decisions
  - Commit decision
- No release decision

Continuous Delivery

- Continuous delivery is a delivery process
  - Upon trigger (commit to official repository)
    - the system is integrated, tested, and **deployed to production**
    - in a fully automated way (no human intervention)
  - A poorly functioning system may be rolled back
    - Requires monitoring and rollback facility of deployed system
    - System status is assessed using key figures
- The purpose of continuous delivery is to
  - put development results into production as fast as possible
  - improve quality by holding the team to high operational standards

Continuous Delivery 1 / 2

1. Test automation
2. Continuous integration
3. Continuous deployment

Continuous Delivery 2 / 2

- Test automation =
  - Tests and testing
- Continuous integration =
  - Test-driven development +
  - Automated building +
  - Test automation
- Continuous deployment =
  - Continuous integration +
  - Deploy to production +
  - Monitoring and rollback
- DevOps [1]
  - Continuous deployment +
  - Operations and culture

[1] Short for “development operations”

Review / Summary of Session

- Test-first programming
- Test-driven development

- Code review practices
  - Pair programming
  - Pre-commit code review

- Continuous integration
- Continuous delivery

Thank you! Questions?

dirk.riehle@fau.de – http://osr.cs.fau.de

dirk@riehle.org – http://dirkriehle.com – @dirkriehle

Credits and License

- Original version
  - © 2021 Dirk Riehle, some rights reserved
  - Licensed under Creative Commons Attribution 4.0 International License
- Contributions
  - None yet