Agile Programming

Dirk Riehle, FAU Erlangen

AMOS B05

Agenda

1. Agile programming
2. Refactoring
3. Test-driven development
4. Code review
5. Build processes
6. Mid-project review

1. Agile Programming

Programming Principles

1. KISS (keep it simple, silly)
2. YAGNI (you ain’t gonna need it)
3. DRY (don’t repeat yourself)

How to Approach a Programming Problem

1. Make it run
2. Make it right
3. Make it fast

Agile Methods are Business-Value-Driven

In agile methods, software developers work off features

- Features must have business value
- Features can cut across runtime tiers and code layers

Agile programming is challenging and requires broad competencies

Collective vs. Individual Code Ownership

Collective code ownership

- Everyone is equally responsible for the overall code base
- Everyone is both allowed to and should be able to fix anything
- Instills a feeling of overall responsibility, ensuring high quality

Individual code ownership

- Individuals are responsible for their own code
- Works best in a distributed setting, e.g. in open source

Agile programming assumes collective code ownership

Programming Standards (a.k.a Coding Guidelines)

A **programming standard** is a

- Set of rules and conventions for naming, formatting, and structuring code

A standard makes it easier to read code written by other people

- Nine times out of ten, code is read, not written

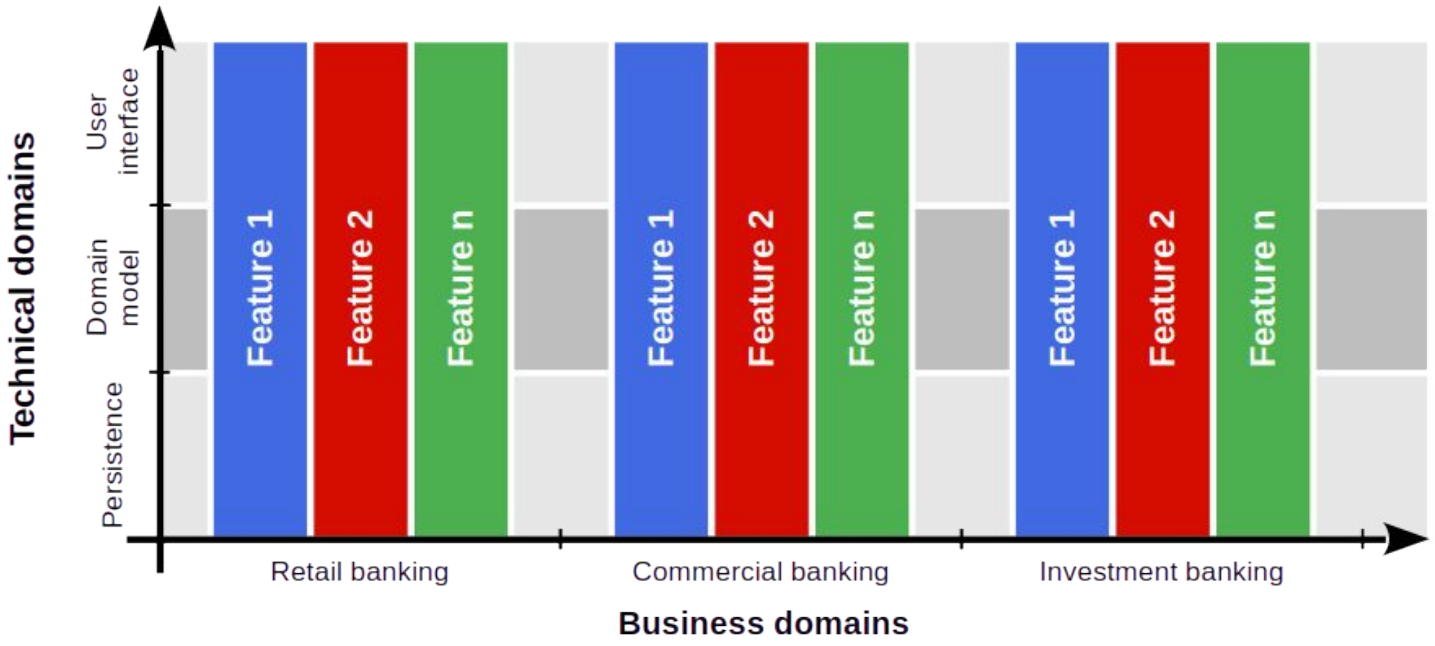
Programming standards should be mandatory

Professional Language [1]

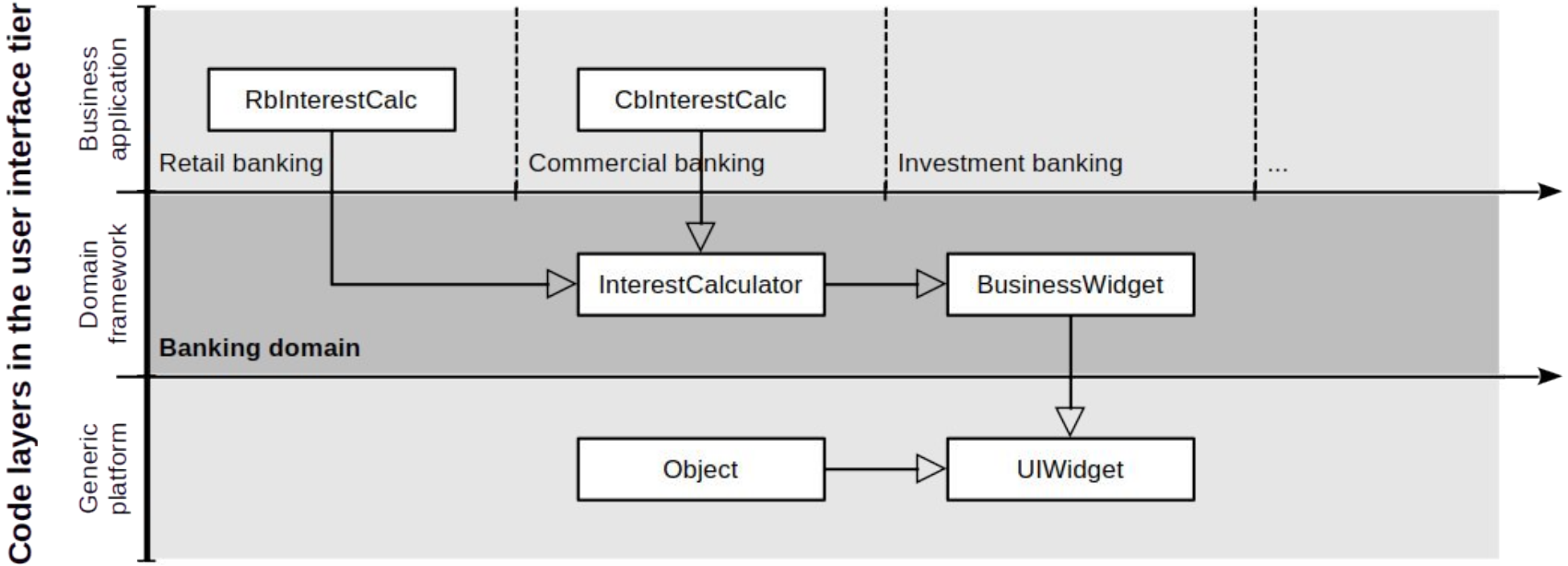
Query methods	Mutation methods	Helper methods
Get method (getter)	Set method (setter)	Factory method
Boolean query methods	Command method	Cloning method
Comparison method	Initialization method	Assertion method
Conversion method	Finalization method	Logging method
…	…	…

[1] See our course on advanced design and programming (ADAP)

Feature Teams (Dealing With Complexity)



Inner-Source Software Development



2. Technical Debt

Ward Cunningham [1] on Technical Debt [2]

[1] See https://en.wikipedia.org/wiki/Ward_Cunningham
[2] See https://youtu.be/Jp5japiHAs4

Technical Debt

Technical debt is a

- Lack of quality or comprehensiveness of code that
  - You accept to temporarily speed up development
  - Until you have to pay back the debt by refactoring

It is a metaphor used to communicate with managers

Managing Technical Debt

1. **Identify the technical debt**
   a. By so called “code smells”
2. **Determine the need to act**
   a. By correlating occurrences
3. **Know how to pay back**
   a. By refactoring your code

Code Smells [1]

Code smells are

- Identifiable structures in code that
  - Violate established design principles and
  - Reduce overall code quality

Code smells are not bugs (the code works, it just … smells)

[1] Fowler, M. (1999). Refactoring. Addison-Wesley Professional.

Example Code Smells

1. Duplicate code
2. Long method
3. Large class
4. …

The “Three Strikes” Rule

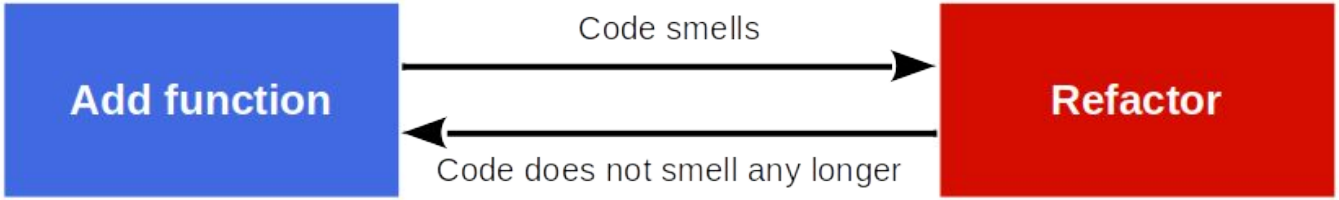
First time:	**Just do it**
Second time:	**Wince at duplication**
Third time:	**Refactor**

Refactoring

A **refactoring** is a

- Behavior-preserving transformation of code (with the goal of improving it)

Refactoring Process (“Two Hats”)



Example Refactorings

1. Extract method
2. Pull up field
3. Form template method
4. Decompose conditional
5. Extract class
6. Extract subclass
7. Extract interface
8. …

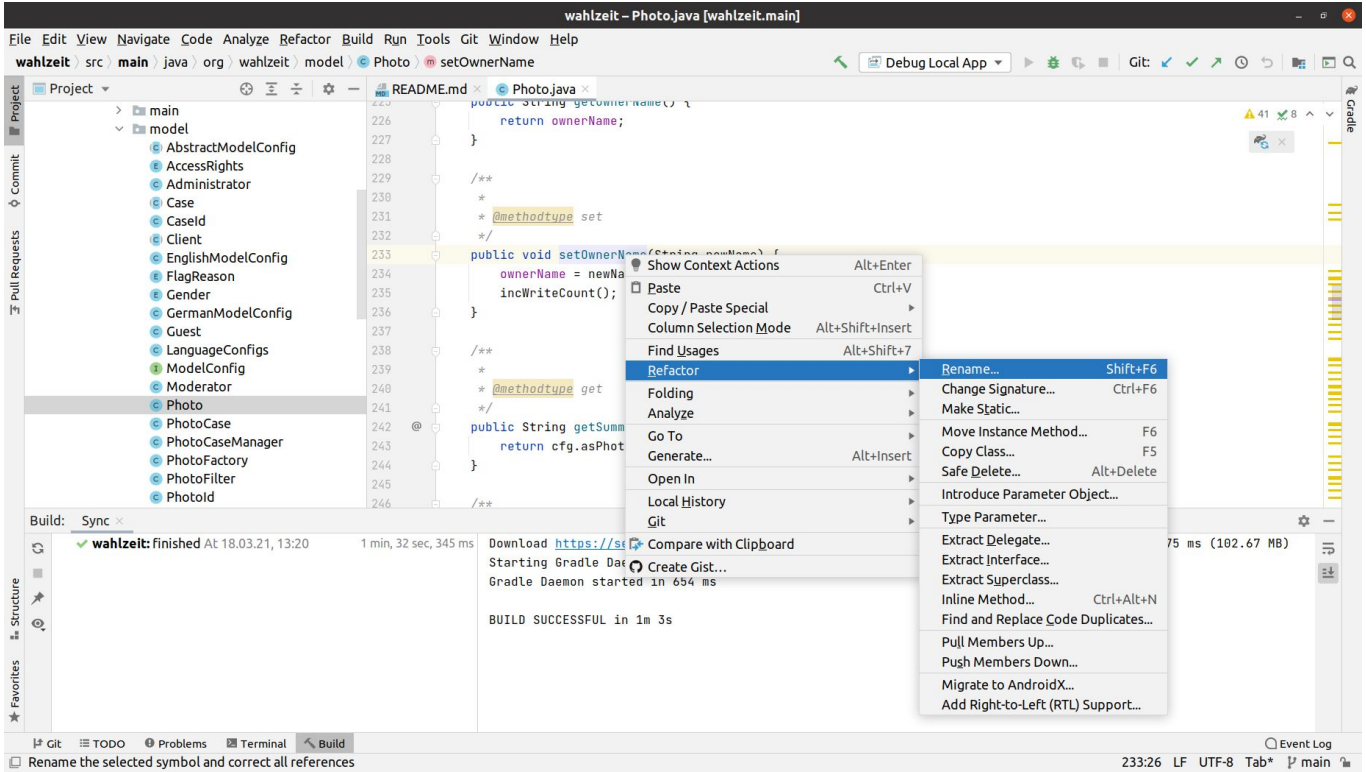
Example Smell Removal by Refactoring

Remove **Duplicated Code** by **Extract Method** or **Pull Up Field** or …

Remove **Long Method** by **Extract Method** or **Decompose Conditional** or …

Remove **Large Class** by **Extract Class** or **Extract Subclass** or …

IDEs Readily Support (Some) Refactorings



3. Test-Driven Development

Test Terminology

Testing is a process for assessing correct operation according to a specification

A test is the instructions to perform a specific assessment

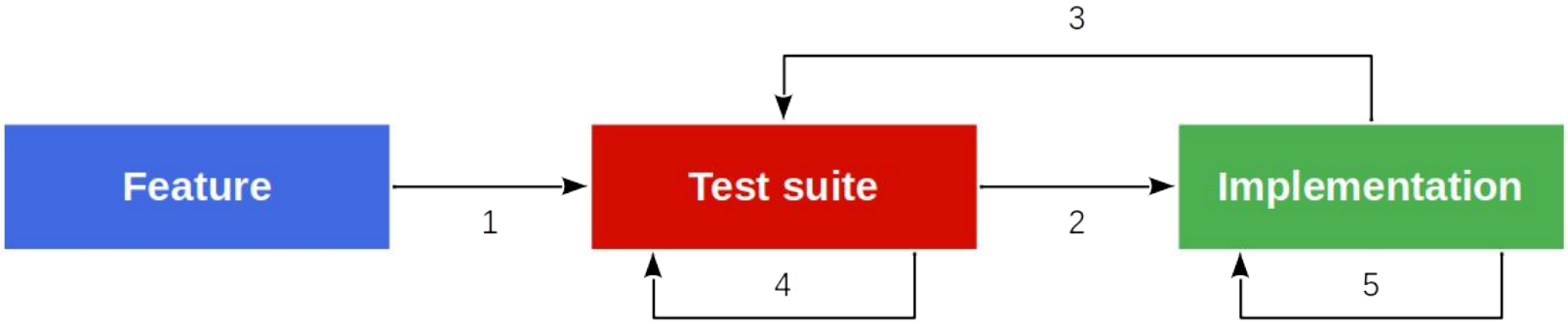
- Component tests (unit tests) test a particular component in isolation
- Acceptance tests (functional tests) test a cross-cutting function
- Integration tests (end-to-end tests) test the interaction of several components

Tests can be automated or manually performed

Test-First Programming

Test-first programming is the

- Practice of first writing a test and then making the system pass the test



Rules of Test-first Programming

Only write new code if a test fails, then eliminate waste

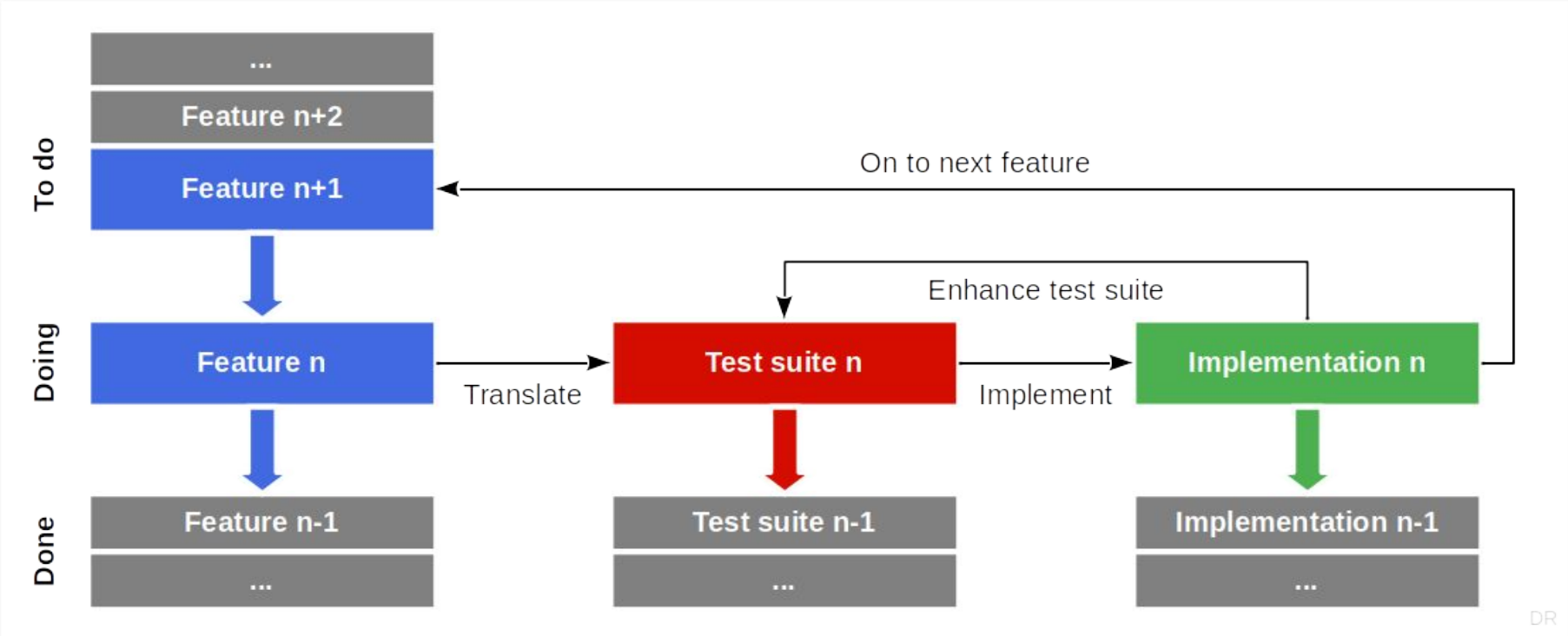
Red, green, refactor

Test-Driven Development

Test-driven development is a

- Minimal development process based on test-first programming

Test-Driven Development Process



4. Code Review

Code Review

Code review is the practice of

- Having someone else assess your code for feedback and approval

More formally, a code review is

- The systematic examination [...] of computer source code

Common forms of code review are

- Pair programming
- Walkthroughs
- Inspections

When to Perform Code Reviews

1. **As code is written (→ Pair programming)**
2. **Before a commit (→ Pre-commit code review)**
3. Before a release
4. At some other time

Pair Programming

Pair programming is the practice of

- Two people in front of the same display, with
  - One person implementing, i.e. writing code (acting in the moment), and
  - One person reviewing, i.e. watching, thinking, commenting, steering

These are the programmer and the reviewer, also

- Driver and co-driver
- Pilot and navigator

Advice on Pair Programming

Find comfortable partner

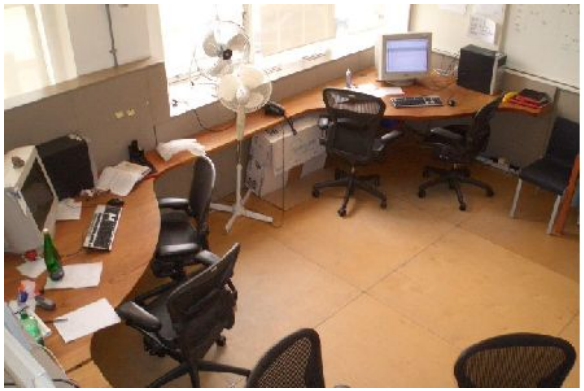
Switch roles regularly

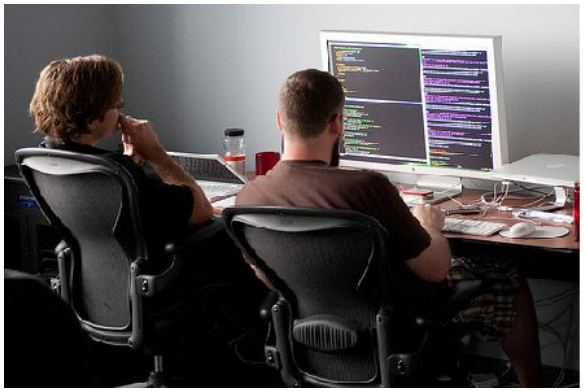
Communicate regularly

Don’t force it for small stuff

Don’t overheat, take breaks

Switch partners at times





uni1.de/amos

Pre-Commit Code Review

Pre-commit code review is the practice of

- Having a peer review code for feedback and approval before it gets committed

The benefits of pre-commit code review

- Strengthens feeling of collective responsibility
- Improves knowledge sharing and teamwork
- Catches bugs and problems at the right time
- Leads to more disciplined developers
- Raises overall quality cost-efficiently

Made easy through distributed version control and merge requests

Source Code Traceability

Traceability of something is the

- Ability to trace the something back to its roots / predecessors

Traceability of source code (a.k.a. post-RS traceability) is the

- Ability to trace back the source code to the requirement it fulfills
- Realized on GitHub by linking pull requests to their issue

Benefits of source code traceability

- Makes clear why the code is there in the first place
- Helps fulfill standards and certification requirements
- Ensures commits are focused / semantically closed

5. Build Processes

Build Process

A **build process** is the process of

- Creating an installable software from its source artifacts

Quality criteria of a build process are

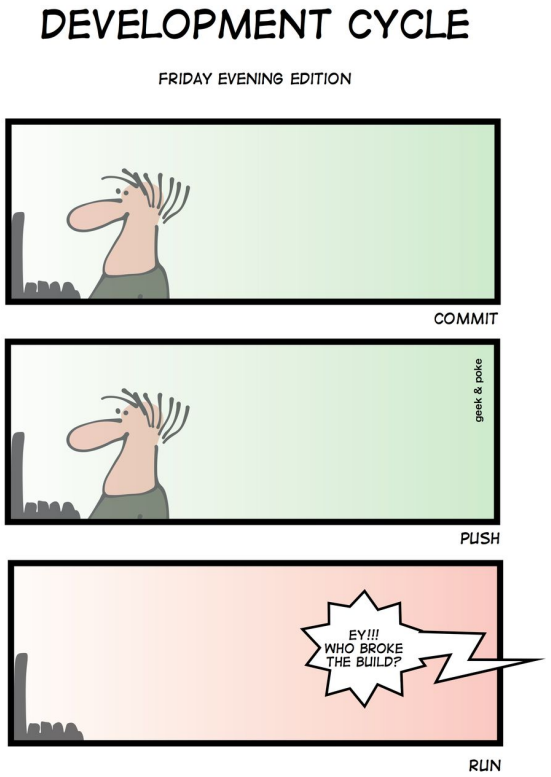
- Fully automated
- Reentrant and deterministic
- With a defined context independent environment

All build assets need to be managed properly

Developer Responsibilities

Don’t break the build (where it affects others)

- Only commit code that compiles
- Only commit code that passes all tests
- Only work in a standardized work environment



Continuous Integration

Continuous integration is the practice of

- Automatically building and testing the software upon defined triggers

Usually with every commit of a developer

- Sometimes only once per day (nightly builds)

Continuous integration may have different scopes

- Depending on the size of the software under development

The goal is to always know whether the software is in good working order

- The faster you can react to an issue, the better (more cost-efficient)

Continuous Deployment

Continuous deployment is the practice of not only building and testing but of

- Deploying the software into production

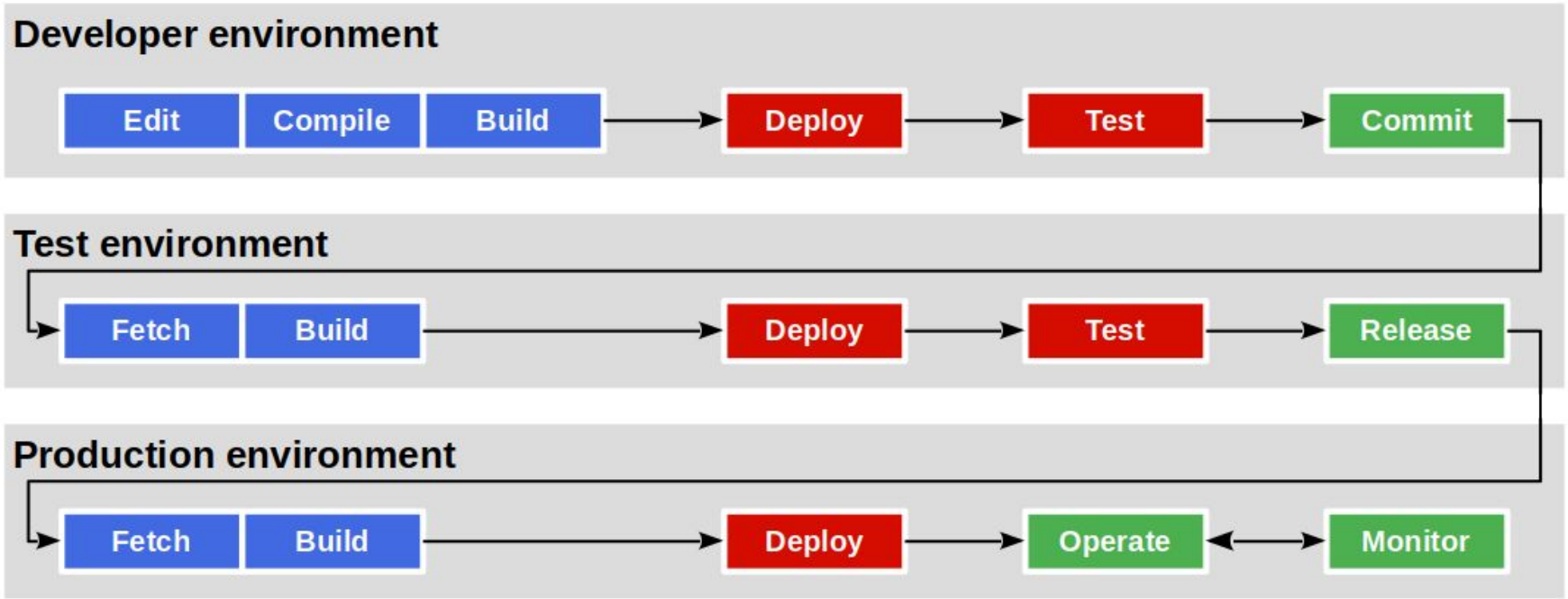
Users are the final deciders of

- Whether the software does what is expect of it

Continuous deployment requires monitoring the performance of the software

- Beyond system tests, you need to watch key metrics of performance

Separation of Environments



One-time Deliverable: Build Process Video

Please create a recorded from-scratch demonstration of your full build process

If you run a continuous integration process, great! But it is not required

6. Mid-Project Review

Milestone: Mid-Project Review

We expect you to demo your work

- One command to start the software for demoing purposes
- Have a script for the most common use-case and demo it
- We will call arbitrarily on people in the team to show this

Feel free to coordinate with and learn from other teams

Summary

1. Agile programming
2. Refactoring
3. Test-driven development
4. Code review
5. Build processes
6. Mid-project review

Thank you! Any questions?

dirk.riehle@fau.de – https://oss.cs.fau.de

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

Legal Notices

License

- Licensed under the CC BY 4.0 International license

Copyright

- © Copyright 2009, 2024 Dirk Riehle, some rights reserved