# Test-Driven Development

**Prof. Dr. Dirk Riehle**

**Friedrich-Alexander University Erlangen-Nürnberg**

## AMOS E03

# 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**
  - 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

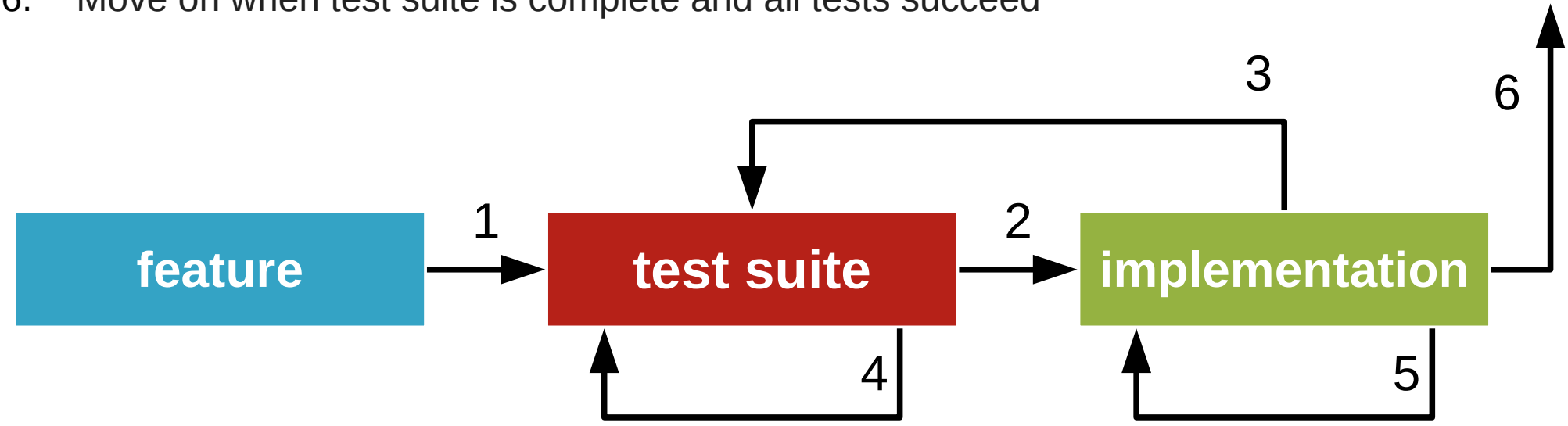
1. Red
2. Green
3. Refactor

# Test-driven Development (TDD) 1 / 3

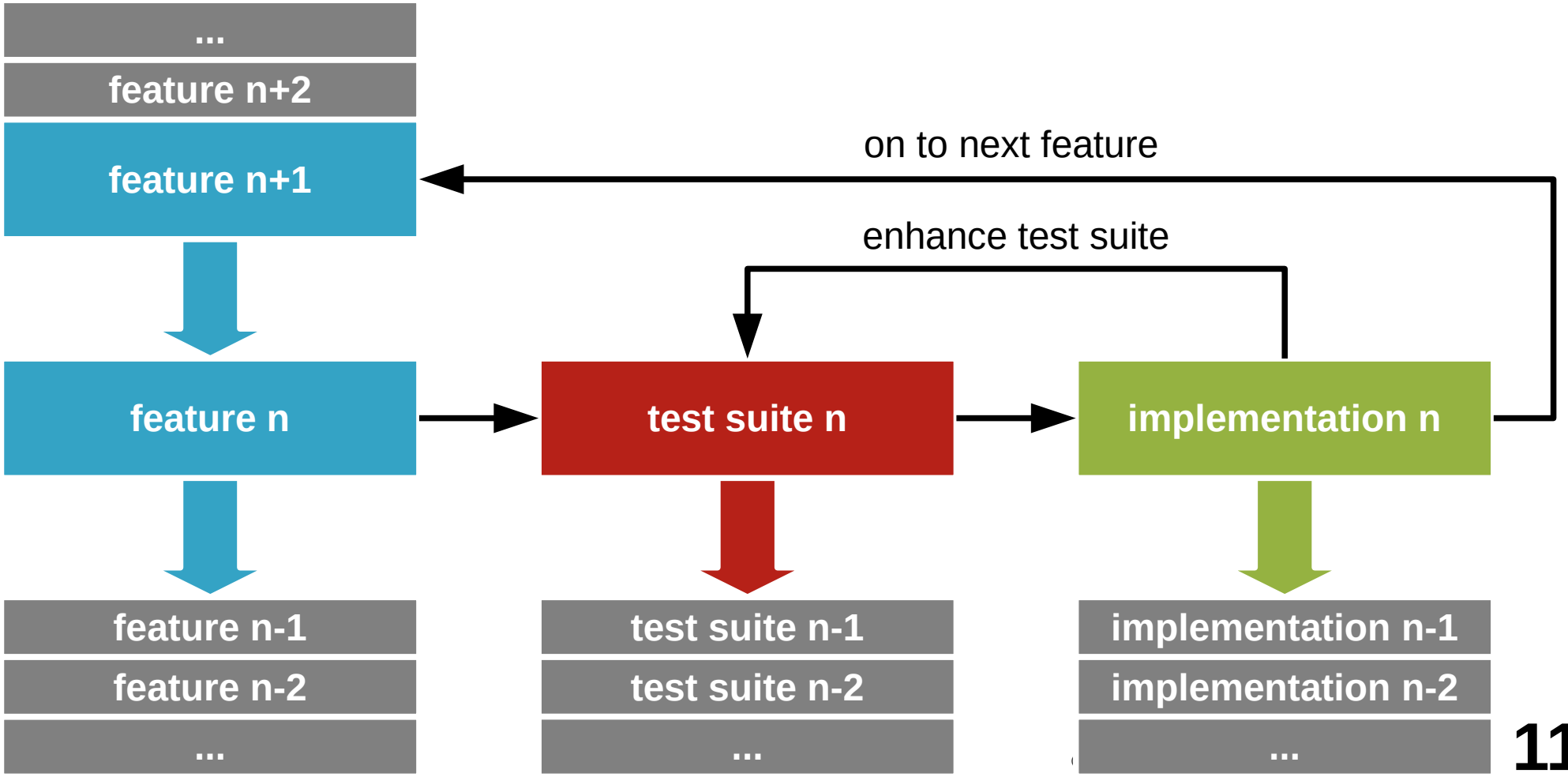
- Test-driven development
  - is a minimal development process based on test-first programming
  - turns feature requests into implementations
- Purpose of test-driven development
  - to grow the product incrementally and steadily
  - to be able to release after every feature implementation

# 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



# Review / Summary of Session

- Tests and testing
  - The basics of it
- Test-first programming
  - What it is, the rhythm of it
- Test-driven development
  - How this simplest of all process works

# Thank you! Questions?

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

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

# Credits and License

- Original version
  - © 2020 Dirk Riehle, some rights reserved
  - Licensed under Creative Commons Attribution 4.0 International License
- Contributions
  - None yet