Test-driven Development

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

ADAP B04

Test-driven Development in Context

1. Tests and Testing
2. Test-first Programming
3. Test-driven Development

Test-First Programming [B02]

- Test-first programming is a practice in which developers
  - write a test before they implement the actual functionality
  - 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

Roman Numerals Example

CMLXXIV

100 less than 1000 plus
50 plus 10 plus 10 plus
1 less than 5 =

974

Roman Numerals Explained

Base Values

- 'I' = 1
- 'V' = 5
- 'X' = 10
- 'L' = 50
- 'C' = 100
- 'D' = 500
- 'M' = 1000

Parsing Rules

- Smaller base cases to the right are added to value
- Smaller base cases to the left are subtracted
- Rule 2 takes precedence over rule 1
- ...

Video on Test-Driven Development [1]

Roman Numerals

(A TDD Coding “Kata”)

Video Lessons

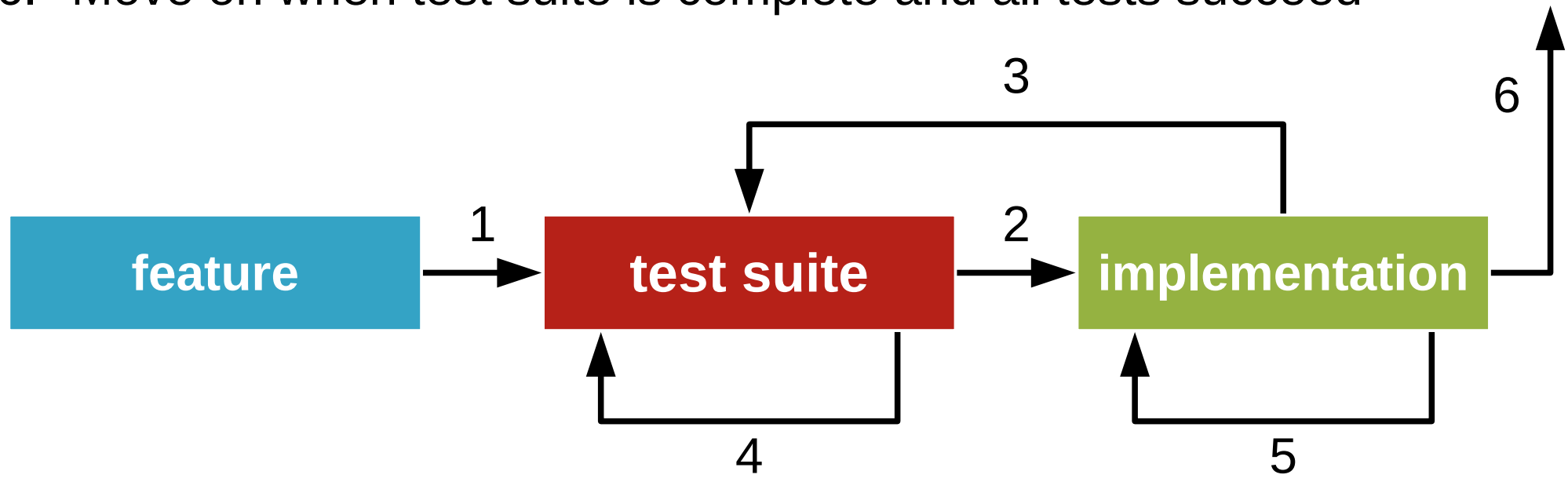
- Implements tests first, functions second
- Provides trivial implementations first
- Provides full implementations incrementally
- Programs with no slack at all, only progress
- Uses many IDE refactoring functions
- Views test code and function code in parallel windows
- Uses JUnitMax for unobtrusive feedback
- Deletes code after finishing coding kata

Test-driven Development (TDD) 1 / 3

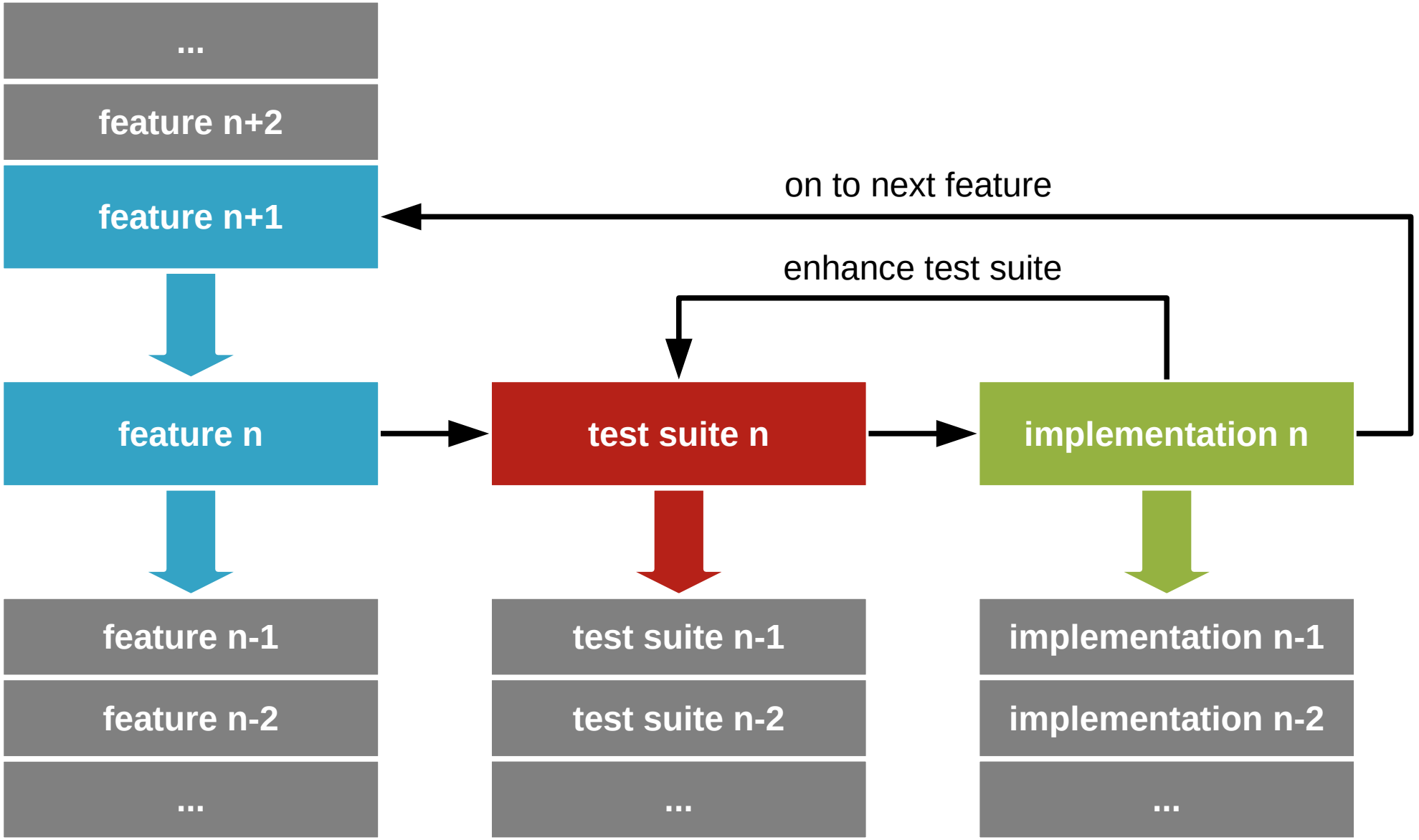
- Test-driven development
  - is a minimal development process based on test-first programming
  - turns feature requests into implementations
- Purpose of test-driven development
  - to grow the product incrementally and steadily
  - to be able to release after every feature implementation

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



Coding Humor: Life without Tests

YOU ARE IN A LEGACY CODEBASE

> RUN TESTS

YOU HAVE NO TESTS.

> READ SPEC

YOU HAVE NO SPEC.

> WRITE FIX

YOU ARE EATEN BY AN OLDER CODE HACK.

Review / Summary of Session

- Test-first programming
  - What it is, the rhythm of it
- Test-driven development
  - How this simplest of all process works

Thanks! Questions?

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

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

Credits and License

- Original version
  - © 2012-2018 Dirk Riehle, some rights reserved
  - Licensed under Creative Commons Attribution 4.0 International License
- Contributions
  - ...

Test-driven Development

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

ADAP B04

It is Friedrich-Alexander University Erlangen-Nürnberg – FAU, in short.

Corporate identity wants us to say “Friedrich-Alexander University”.

Test-driven Development in Context

1. Tests and Testing
2. Test-first Programming
3. Test-driven Development

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**

Roman Numerals Example

CMLXXIV

100 less than 1000 plus
50 plus 10 plus 10 plus
1 less than 5 =

974

Roman Numerals Explained

Base Values

- 'I' = 1
- 'V' = 5
- 'X' = 10
- 'L' = 50
- 'C' = 100
- 'D' = 500
- 'M' = 1000

Parsing Rules

- Smaller base cases to the right are added to value
- Smaller base cases to the left are subtracted
- Rule 2 takes precedence over rule 1
- ...

Video Lessons

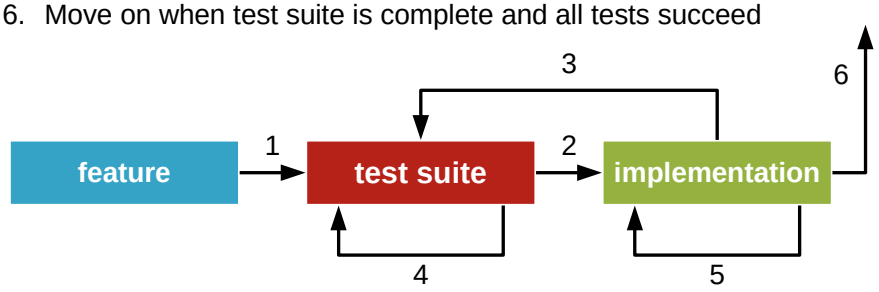
- Implements tests first, functions second
- Provides trivial implementations first
- Provides full implementations incrementally
- Programs with no slack at all, only progress
- Uses many IDE refactoring functions
- Views test code and function code in parallel windows
- Uses JUnitMax for unobtrusive feedback
- Deletes code after finishing coding kata

Test-driven Development (TDD) 1 / 3

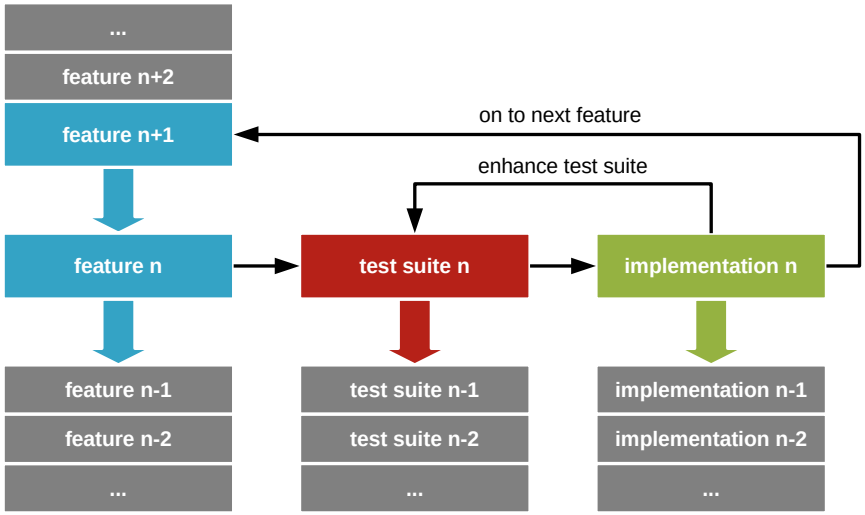
- Test-driven development
  - is a minimal development process based on test-first programming
  - turns feature requests into implementations
- Purpose of test-driven development
  - to grow the product incrementally and steadily
  - to be able to release after every feature implementation

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



Coding Humor: Life without Tests

YOU ARE IN A LEGACY CODEBASE

> RUN TESTS

YOU HAVE NO TESTS.

> READ SPEC

YOU HAVE NO SPEC.

> WRITE FIX

YOU ARE EATEN BY AN OLDER CODE HACK.

Review / Summary of Session

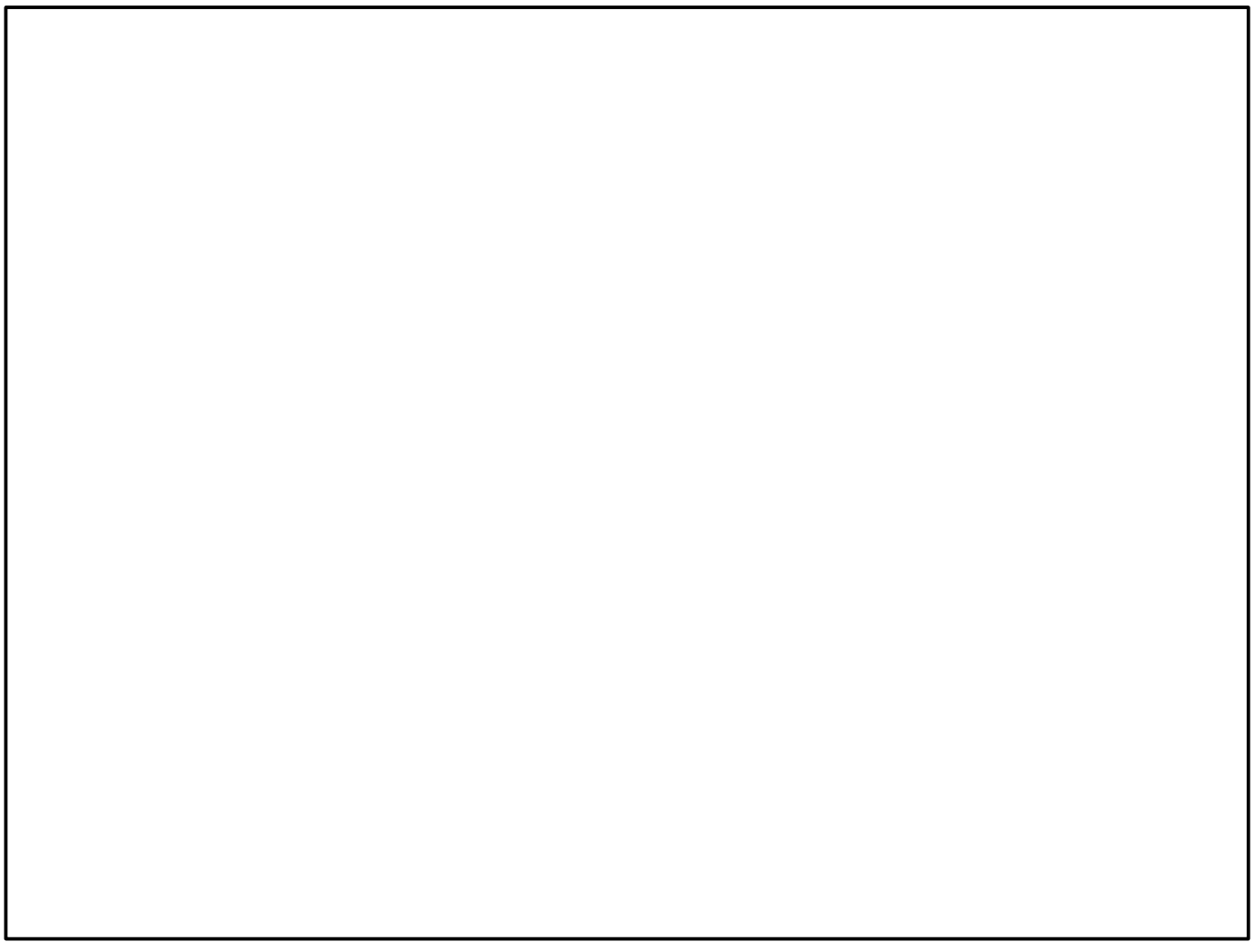
- Test-first programming
  - What it is, the rhythm of it
- Test-driven development
  - How this simplest of all process works

Thanks! Questions?

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

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

DR



Credits and License

- Original version
  - © 2012-2018 Dirk Riehle, some rights reserved
  - Licensed under Creative Commons Attribution 4.0 International License
- Contributions
  - ...