Code renovation training

Takeaways, lessons learned and thoughts...

pawel.hajda@css.ch

Approaching Legacy Code Refactoring

Power of MOB

Practice, practice, practice!

TDD+M

Approaching Legacy Code Refactoring

1. Identify code smells
2. Break dependencies
3. Put approval and mutation tests in place
4. Refactor and Unit Tests

Approaching Legacy Code Refactoring

1. Identify code smells

- Legacy code smells cheat sheet
- Categorise: application-level, class-level and method-level smells
- Rank: criticality level
- Static code analysis

Approaching Legacy Code Refactoring

2. Break dependencies

- IDE driven & provable refactorings
- “Subclass and Override” is tempting to use but do not forget other techniques

Inheriting

Subclass and override method
Extract implementer
Extract interface
Introduce static setter
Push down dependencies

Overriding

Extract and override call
Extract and override getter

Injection

Parametrize constructor

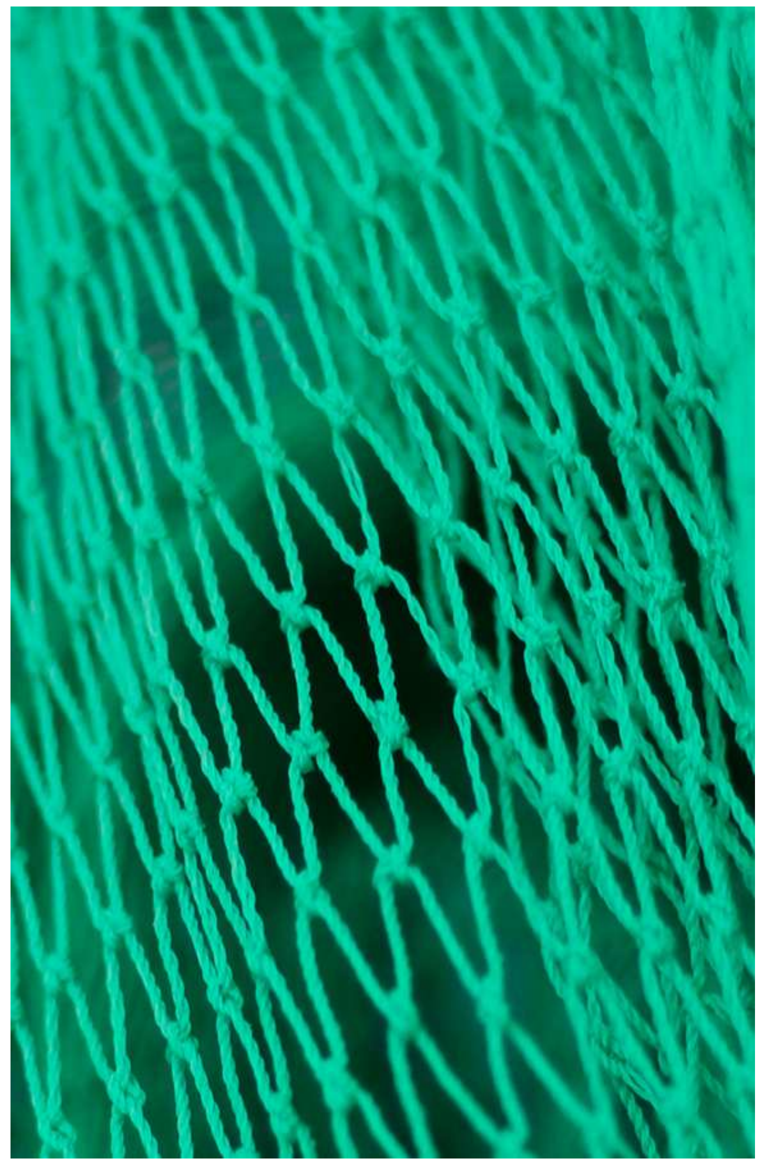
Static

Introduce instance delegator

Approaching Legacy Code Refactoring

3. Put approval and mutation tests in place

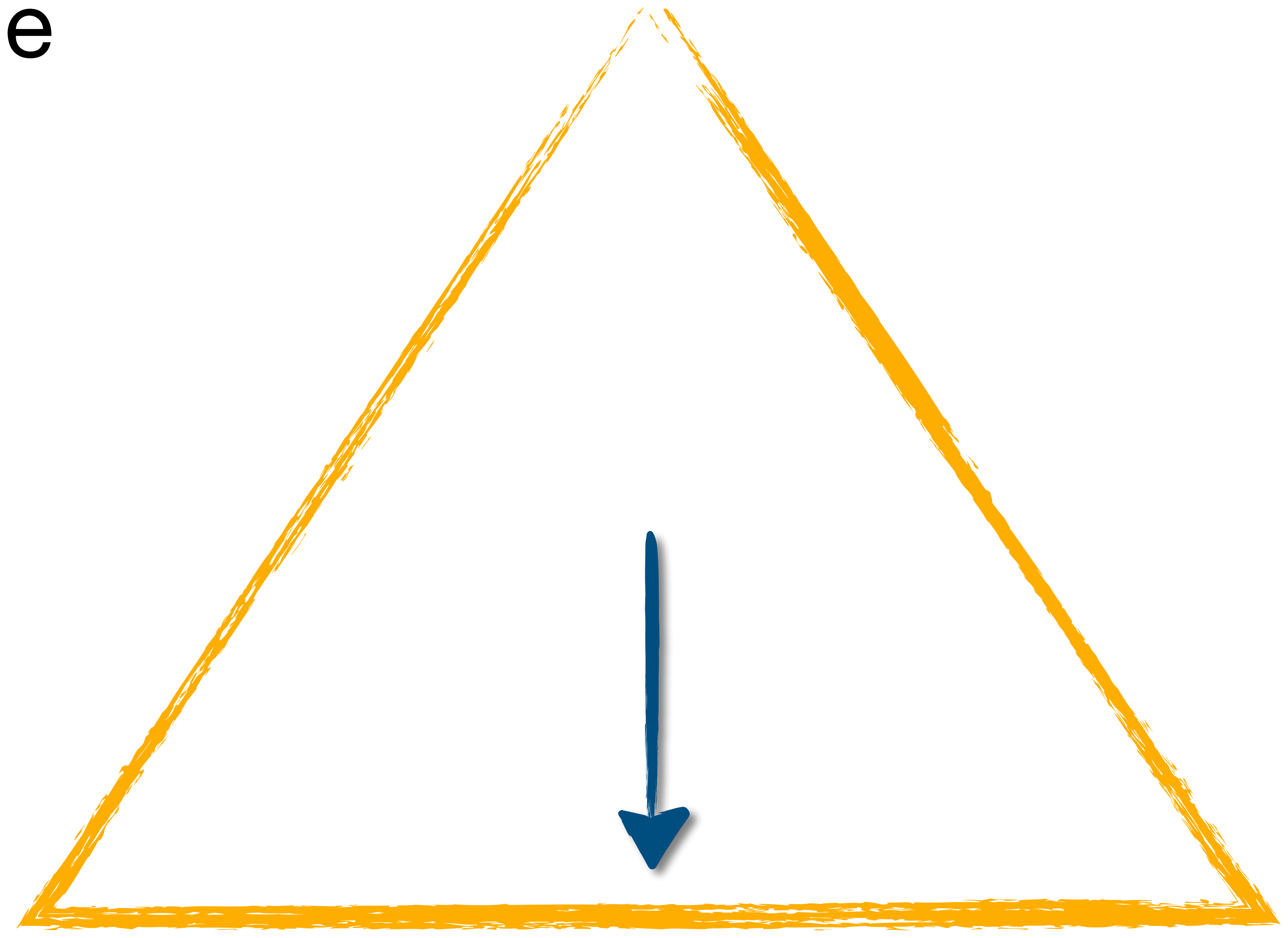
- Safety net, locking down the current behaviour, combinatorial tests give results with high test coverage
- Enable mutation tests to ensure if each statement is meaningfully tested and to test critical boundary cases



Approaching Legacy Code Refactoring

4. Refactor and Unit Tests

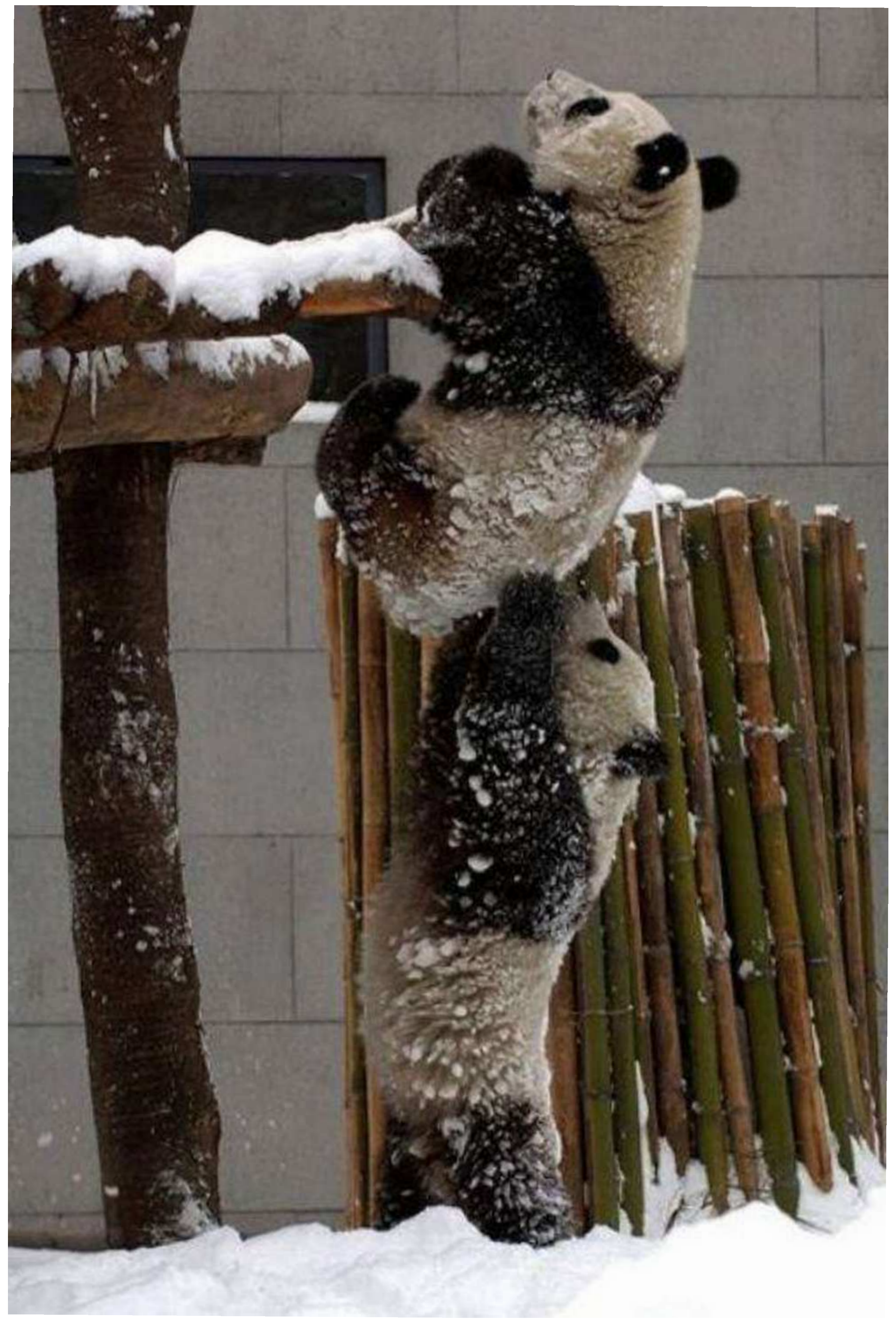
- Having first tests in place, cover the code with unit tests and refactor it.
  Go down in the testing pyramid



Power of MOB

Work effectively

- Gather ideas
- Ongoing review
- Everyone on the same baseline
- Effective decisions (*consent* instead of *consensus*)
- Make sure to have an intention



Practice, practice, practice!

- Kata's
- Master your IDE
  (shortcuts, live templates, etc.)
- Some of the refactoring patterns could be applied directly in your IDE

  (f.e. Peel & Slice technique can be completely IDE-Driven with IntelliJ)

ON THE ROAD TO SUCCESS, THERE ARE NO SHORTCUTS.
JOIN OUR TEAM
SHAFFER TRUCKING
OUR MOST VALUABLE RESOURCE SITS 63 FEET AHEAD.
C 240

Practice, practice, practice!

Breaking dependencies with Extract Interface Pattern (IntelliJ)

Adapter and Interface do not exist yet

1. Extract Method
2. Introduce parameter in extracted method (if necessary)
3. Create Adapter Class
4. Move instance method
5. Extract interface

Putting next methods to interface

1. Extract Method
2. Introduce parameter in extracted method (if necessary)
3. Move instance method
4. Push members down (with “keep abstract” option!)

TDD+M

Test-driven development with mutation testing

Test-driven development with mutation testing – an experimental study

https://www.researchgate.net/publication/346533953_Test-driven_development_with_mutation_testing_-_an_experimental_study

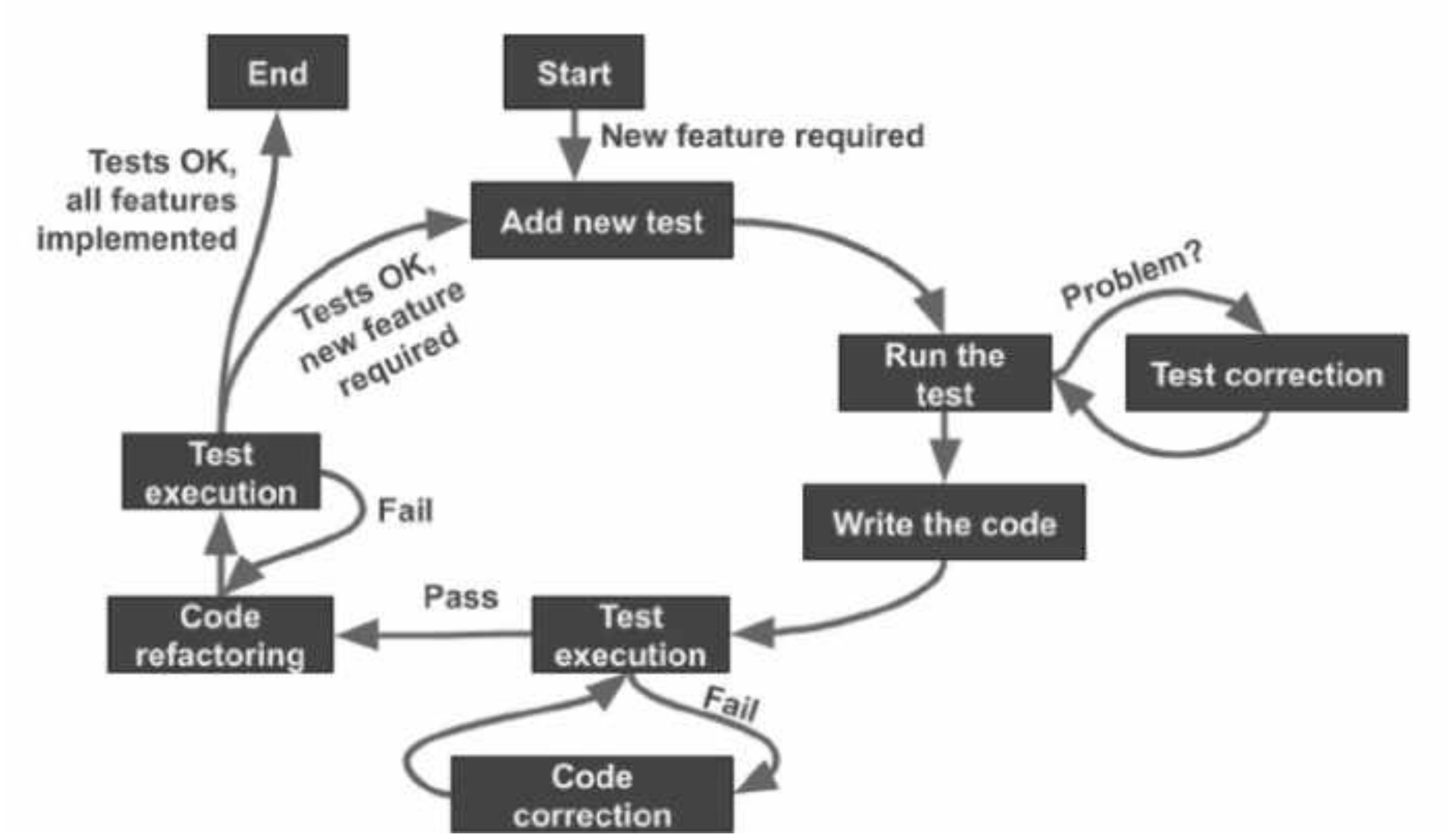
The experiment showed that adding mutation into the TDD process allows
the developers to provide better, stronger tests and to write a better quality code

*“The novelty of this paper, comparing to the studies previously cited, is that we do not focus
on the coverage criteria themselves, but on the role of mutation in the TDD process:
we investigate if mutation testing improves the quality of code developed within the TDD approach”*

TDD+M

Classic TDD and questions about:

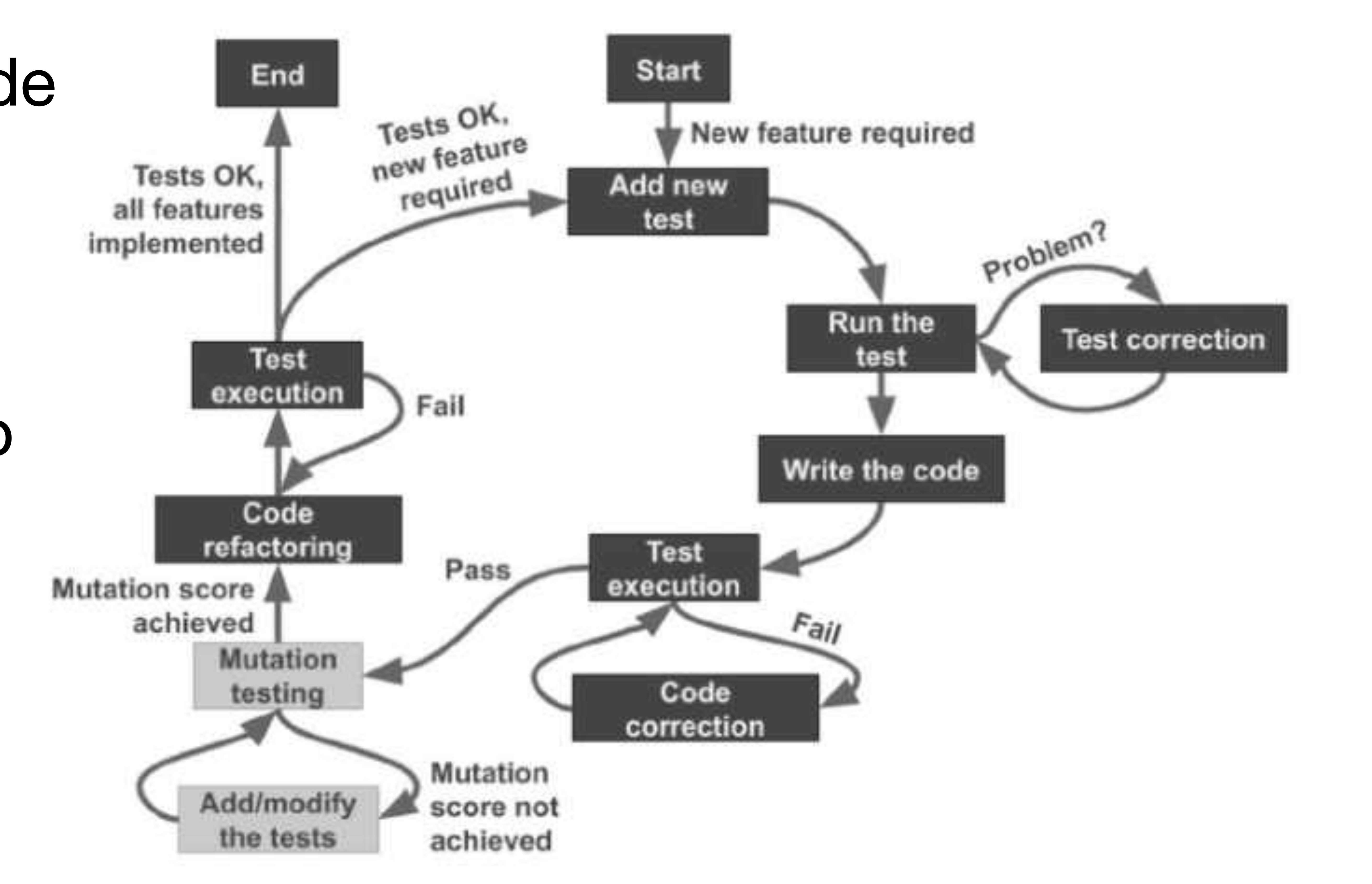
- Sufficiency
- Coverage
- Semantic Stability



TDD+M

TDD with mutation tests

- Mutation tests are inserted between test execution and code refactoring
- When the tests fail during the mutation phase, then we know that test cases are weak and do not detect defects
- Is it worth it?



Thank you for your attention

Questions?

References

- https://alcor.academy
- https://www.researchgate.net/publication/346533953_Test-driven_development_with_mutation_testing_-_an_experimental_study
- https://blog.cleancoder.com/uncle-bob/2016/06/10/MutationTesting.html