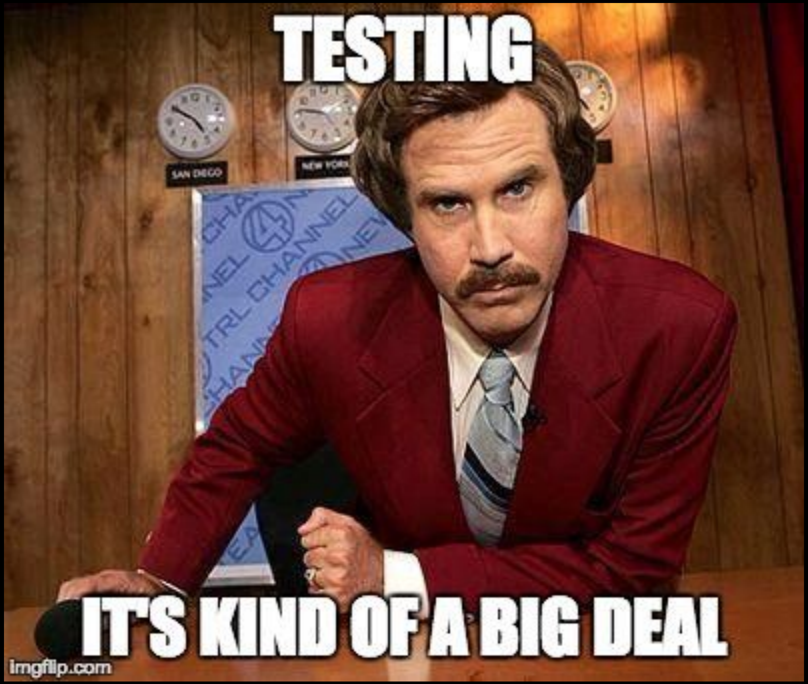
Testing in (scientific) programming

What are (unit) tests?

- **Short executable programs which “test” certain parts of your program**

```
# Implementation
mult(A,x) = [sum(A[i,j]*x[j] for j in 1:size(A,2)) for i in size(A,2)]

# Test
@test mult([1 2], [3, 4])       == [1*3+2*4]
@test mult([1 2; 3 4], [5, 6]) == [1*5 + 2*6, 3*5 + 4*6]
```

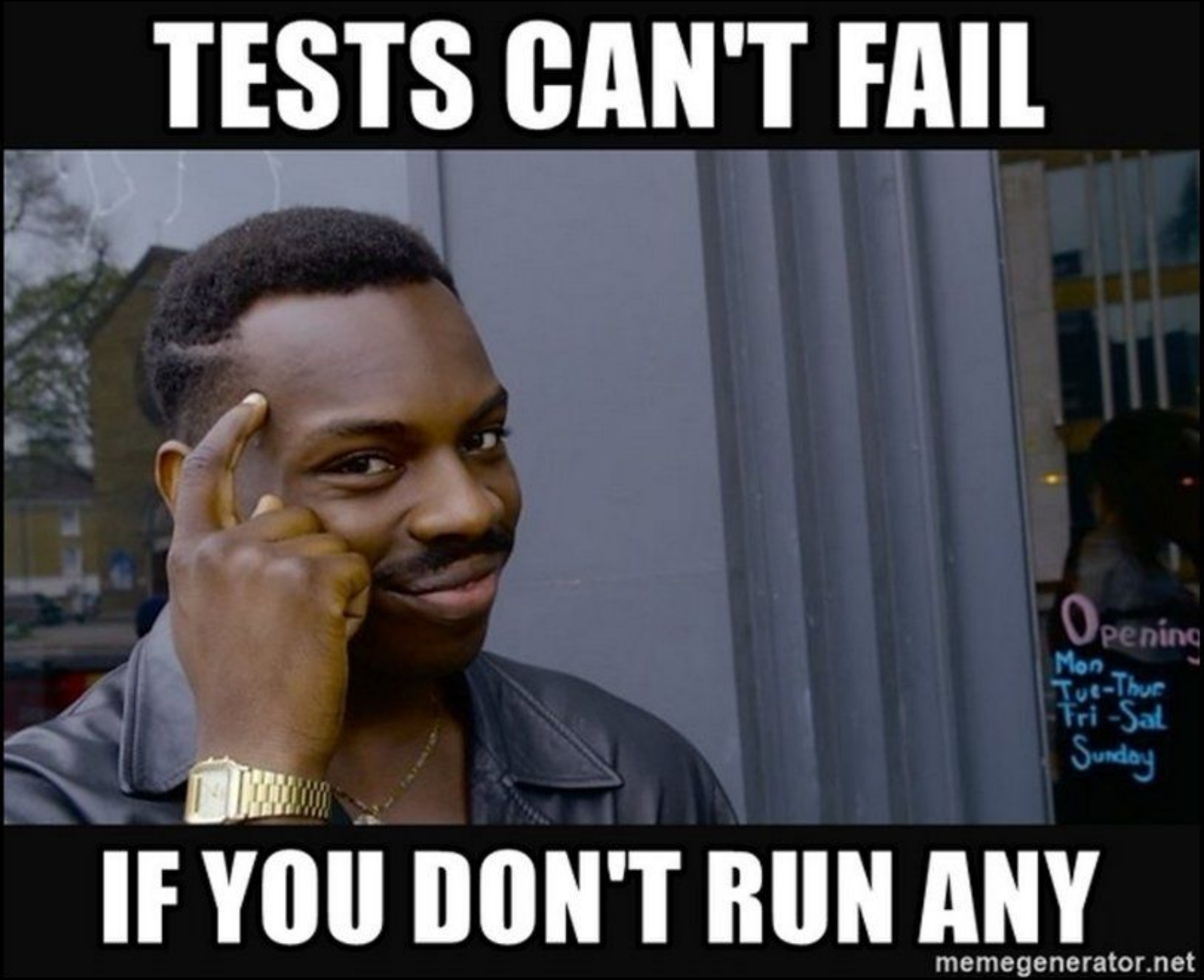

TESTING
IT'S KIND OF A BIG DEAL
imgflip.com

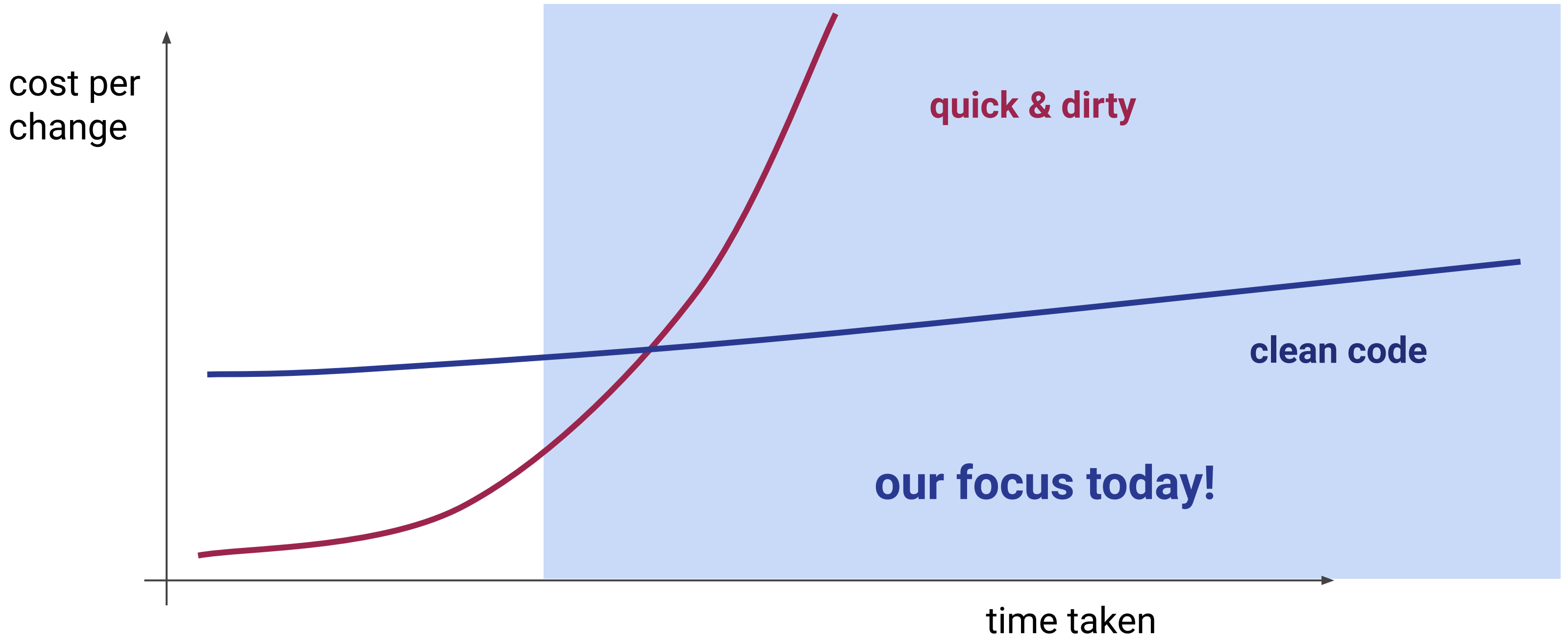
Fantastic bugs and how to find them...

In an ideal world, test reduce the time you need for finding bugs!

Short term:
You test the things which you code **should** do. If it fails, you fix it immediately.

Long term:
If a new change has unexpected side effects, your tests **might** catch it.


TESTS CAN'T FAIL
IF YOU DON'T RUN ANY
Opening
Mon
Tue-Thur
Fri -Sat
Sunday
memegenerator.net



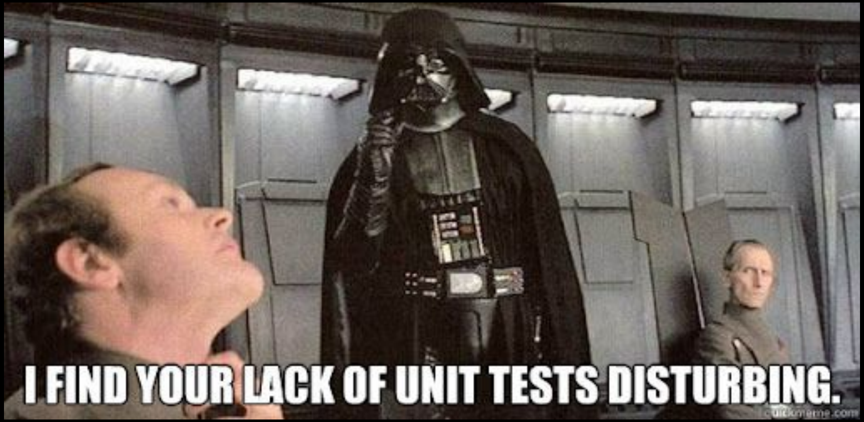
The bigger picture.

QA: Guys, we're finding way too many bugs

Should we test?

Just one example:

There are reports of scientists who believed that they needed to modify the physics model or develop new algorithms, but later discovered that the real problems were small faults in the code.


I FIND YOUR LACK OF UNIT TESTS DISTURBING.
quickmeme.com

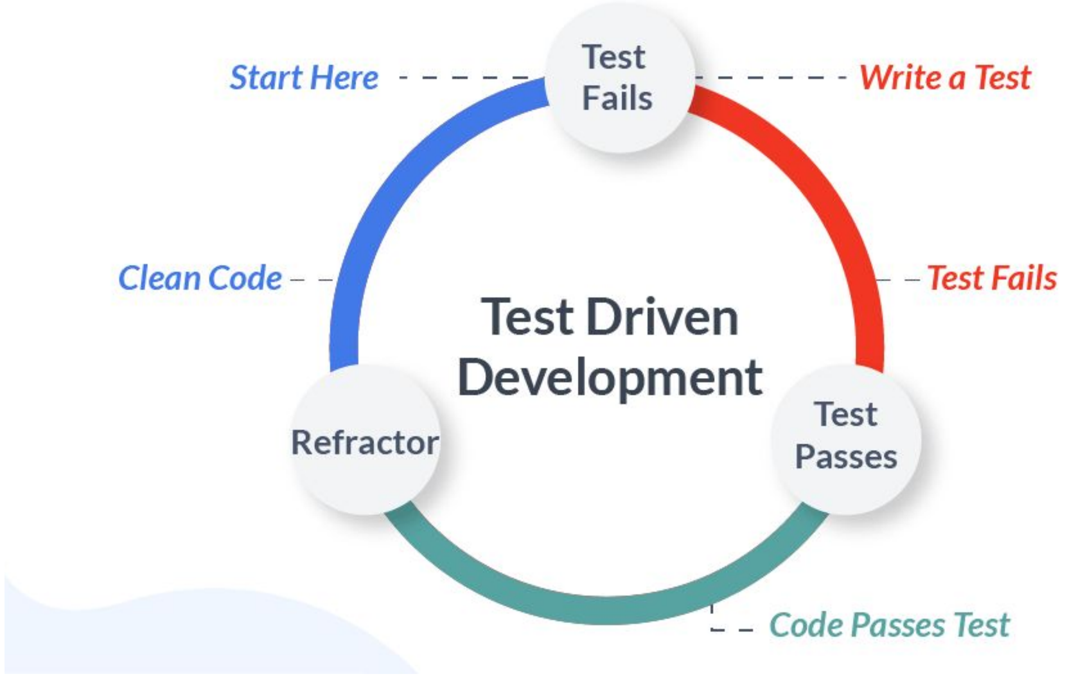
Example

How to write good tests?

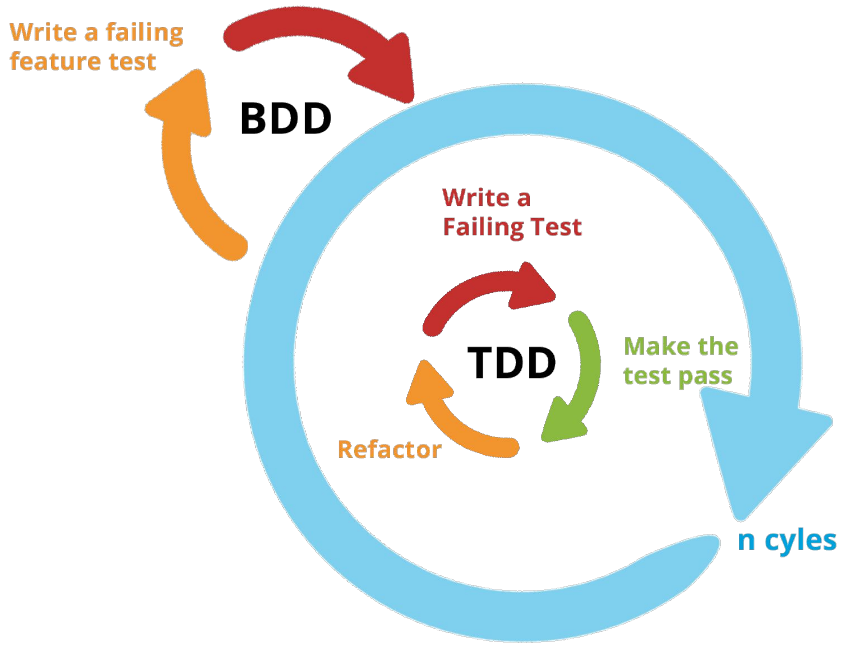
Guidelines:

1. All the tests pass.
2. There is no duplication.
3. The code expresses the intent of the programmer.
4. Classes, and methods are minimized.

Test **driven**-development



source: https://www.xenonstack.com/blog/test-driven-development



Write **specifications**

- in **natural language**

- in collaboration with all collaborators (also **non-developers**)

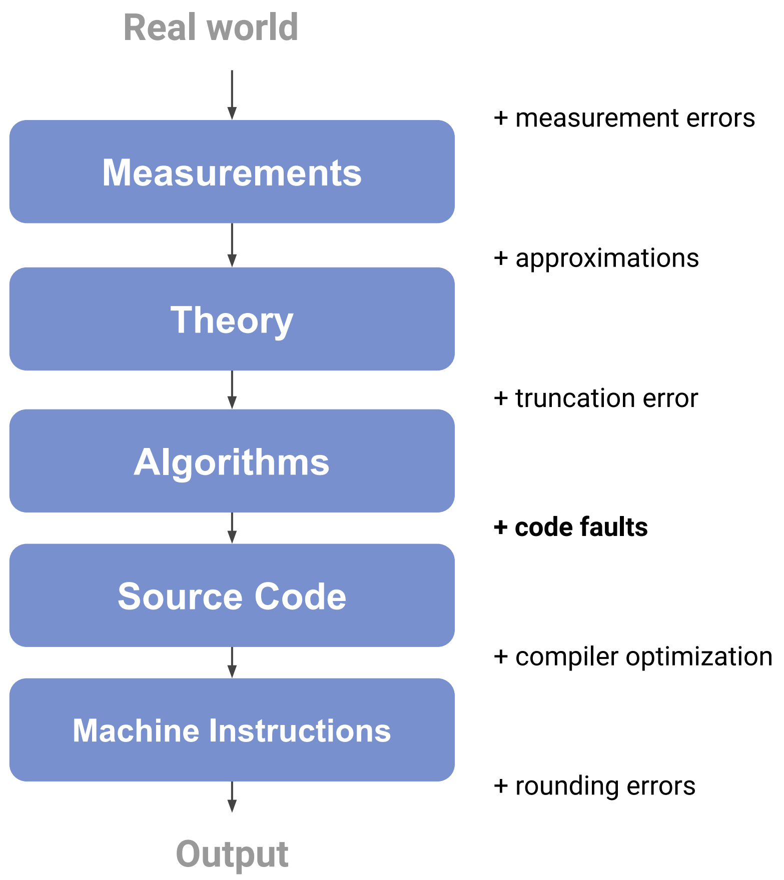
- focus on what the program **should** do and not how it is implemented

Behaviour driven development (BDD)

source: https://www.oreilly.com/library/view/oracle-jet-for/9781787284746/582c3888-7453-4ea8-87d4-d439fd47de73.xhtml

Testing of scientific software

…is not easy


Real world
Measurements
+ measurement errors
Theory
+ approximations
Algorithms
+ truncation error
Source Code
+ code faults
Machine Instructions
+ compiler optimization
+ rounding errors
Output

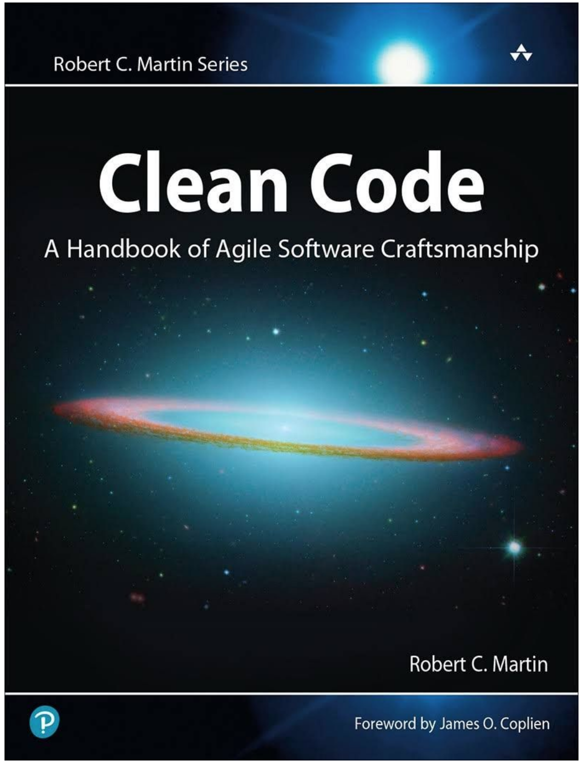
Challenges for testing of scientific software?

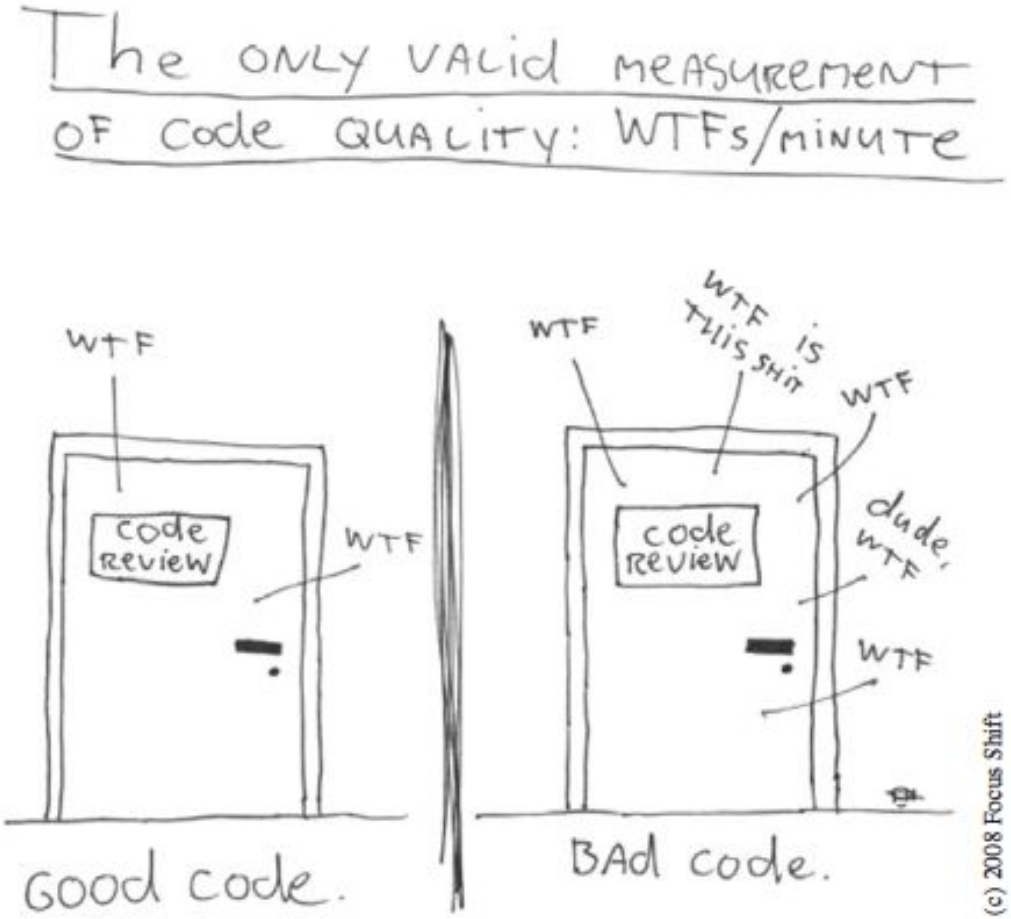
1. **The Oracle Problem:**
   How can we design a test, if we don't know the desired output?

2. **The Tolerance Problem:**
   Faults can be masked by round-off errors, truncation errors and model simplifications.

How much shall we test?

The bigger picture: **Clean code**


Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Robert C. Martin
Foreword by James O. Coplien


The only valid measurement of code quality: WTFs/minute
WTF
code review
WTF
Good code.
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift

- **Avoid fancy coding.** Clearly written code will be fast and small enough.

- **Use available libraries.** The easiest way to not have bugs writing a utility routine is to not write it.

- Learn a few formal techniques for the more complicated stuff. If there's complicated conditions, nail them down with pen and paper.

- For existing code, learn how to **refactor**:
  how to make small changes in the code, often using an automated tool, that **make the code more readable without changing the behavior.**

- **Don't do anything too quickly.**
  Taking a little time up front to do things right, to check what you've done, and to think about what you're doing can pay off big time later.

- Once you've written the code, use what you've got to make it good.
  Unit tests are great. You can often write tests ahead of time, which can be great feedback (if done consistently, this is **test-driven development**). Compile with warning options, and pay attention to the warnings.

- **Get somebody else to look at the code.**
  Formal code reviews are good, but they may not be at a convenient time. Pull requests, or similar if your scm doesn't support them allow for asynchronous reviews. Buddy checking can be a less formal review. Pair programming ensures two pairs of eyes look at everything.

Advanced topics: Continuous Integration

CI (Continuous Integration) **→ Run tests on a server**

- Free for open source projects on GitHub!

Advantages:

- Every commit gets checked.
- Tests run automatically (and not on your own computer)

the End

References

Testing Scientific Software: A Systematic Literature Review
Upulee Kanewala, James M. Bieman

https://arxiv.org/pdf/1804.01954.pdf

Clean Code: A Handbook of Agile Software Craftsmanship
Robert C. Martin