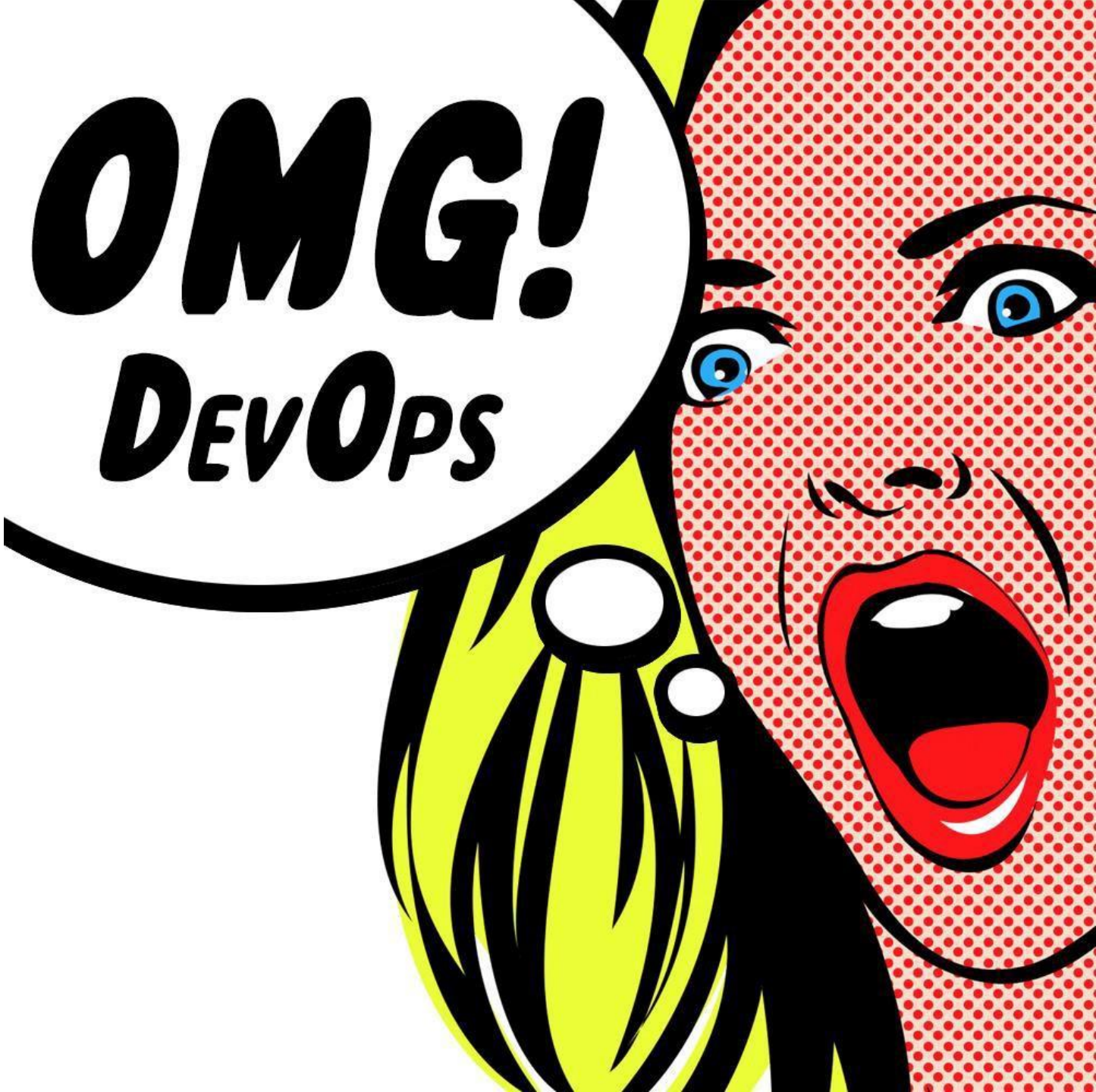
SHIFT LEFT

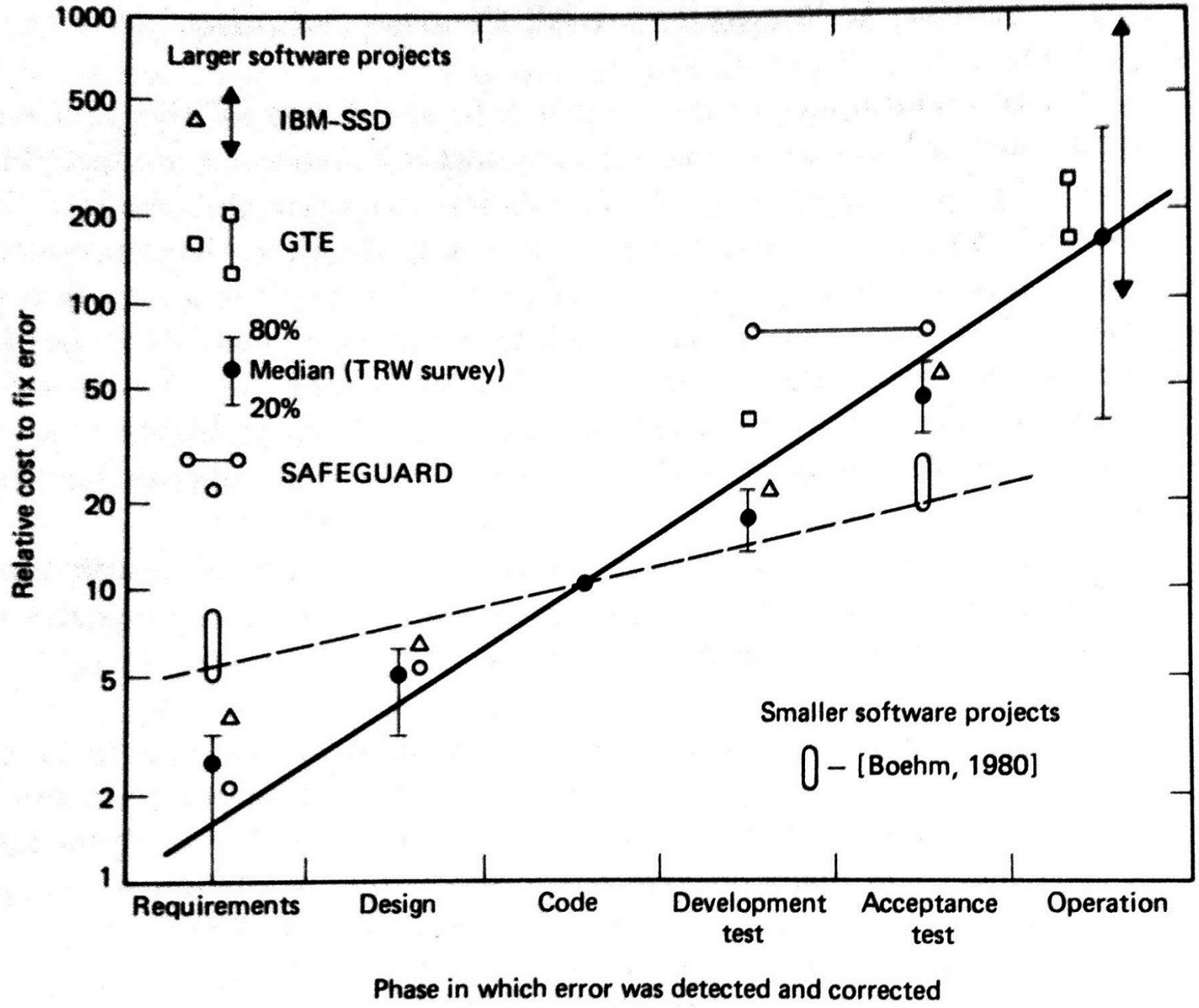
@mpilaeten
Michaël Pilaeten

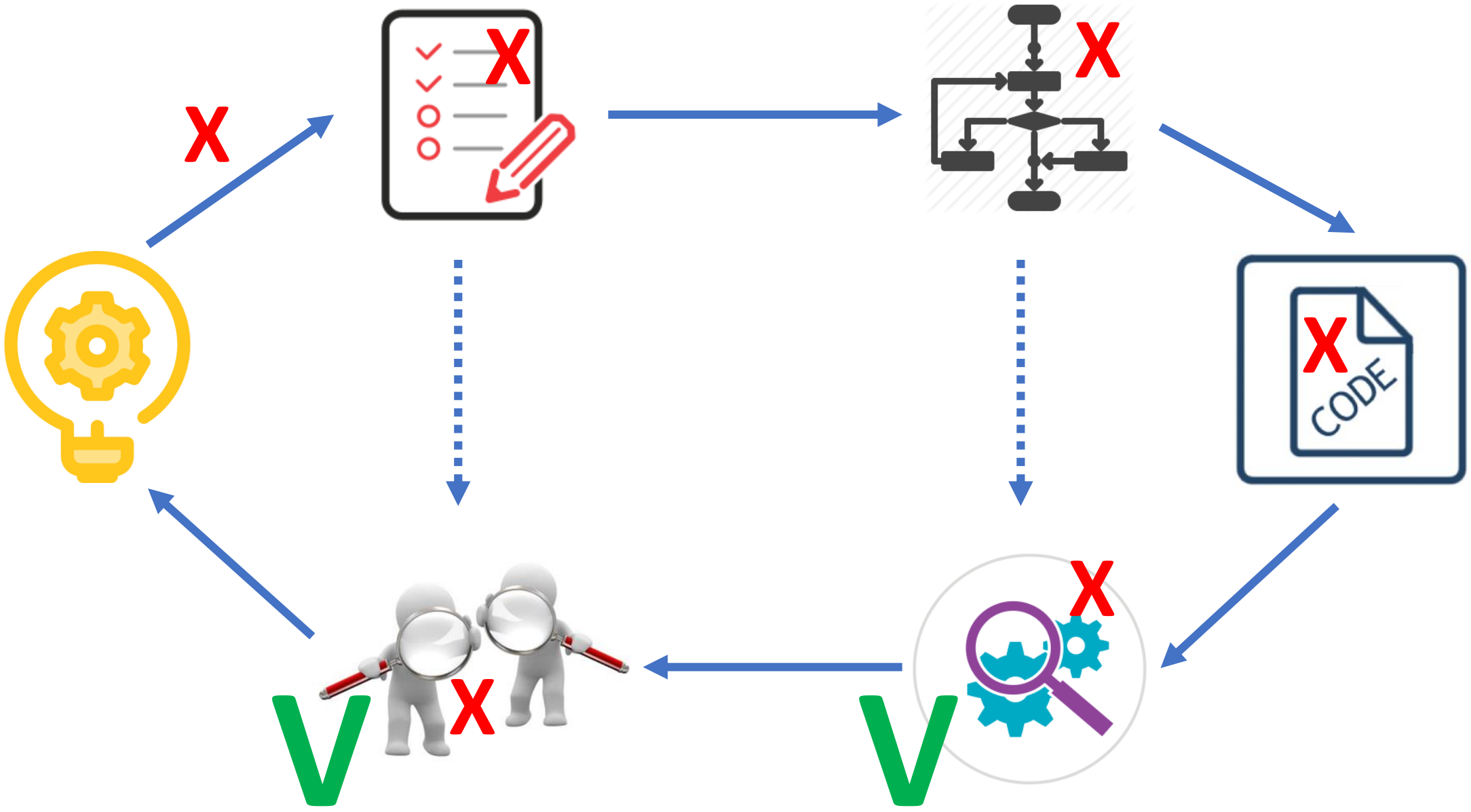
Google

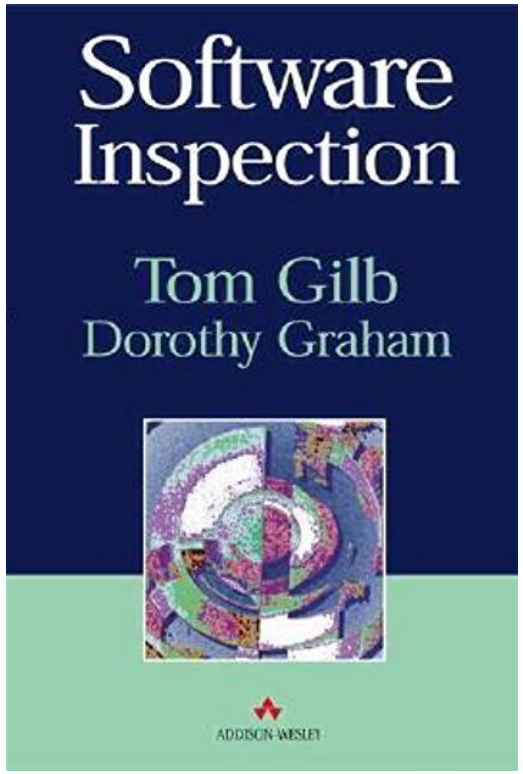
continuous

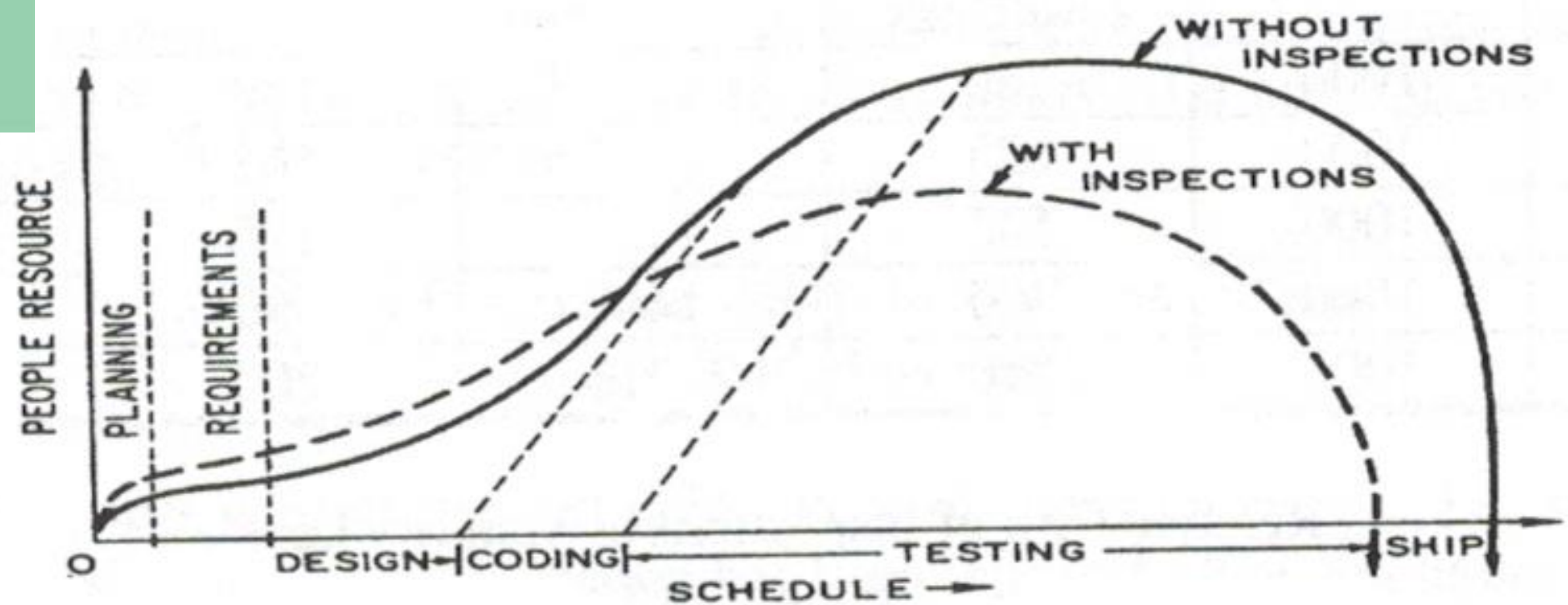
- continuous **improvement**
- continuous **integration**
- continuous **delivery**
- continuous **improvement manager**
- continuous **vertaling**
- continuous **learning**
- continuous **ping**
- continuous **tense**
- continuous **variable**
- continuous **assessment**


OMG!
DevOps


1000
500
200
100
50
20
10
5
2
1
Relative cost to fix error
Larger software projects
IBM-SSD
GTE
80%
Median (TRW survey)
20%
SAFEGUARD
Smaller software projects
– [Boehm, 1980]
Requirements
Design
Code
Development test
Acceptance test
Operation
Phase in which error was detected and corrected


X
X
X
X
CODE
X
V
X
V


Software
Inspection
Tom Gilb
Dorothy Graham
ADDISON-WESLEY


WITHOUT
INSPECTIONS
WITH
INSPECTIONS
PEOPLE RESOURCE
PLANNING
REQUIREMENTS
O
DESIGN
CODING
TESTING
SHIP
SCHEDULE

SHIFT LEFT

WHY ?

- 21% of projects fail due to requirements
  Standish Group CHAOS Report

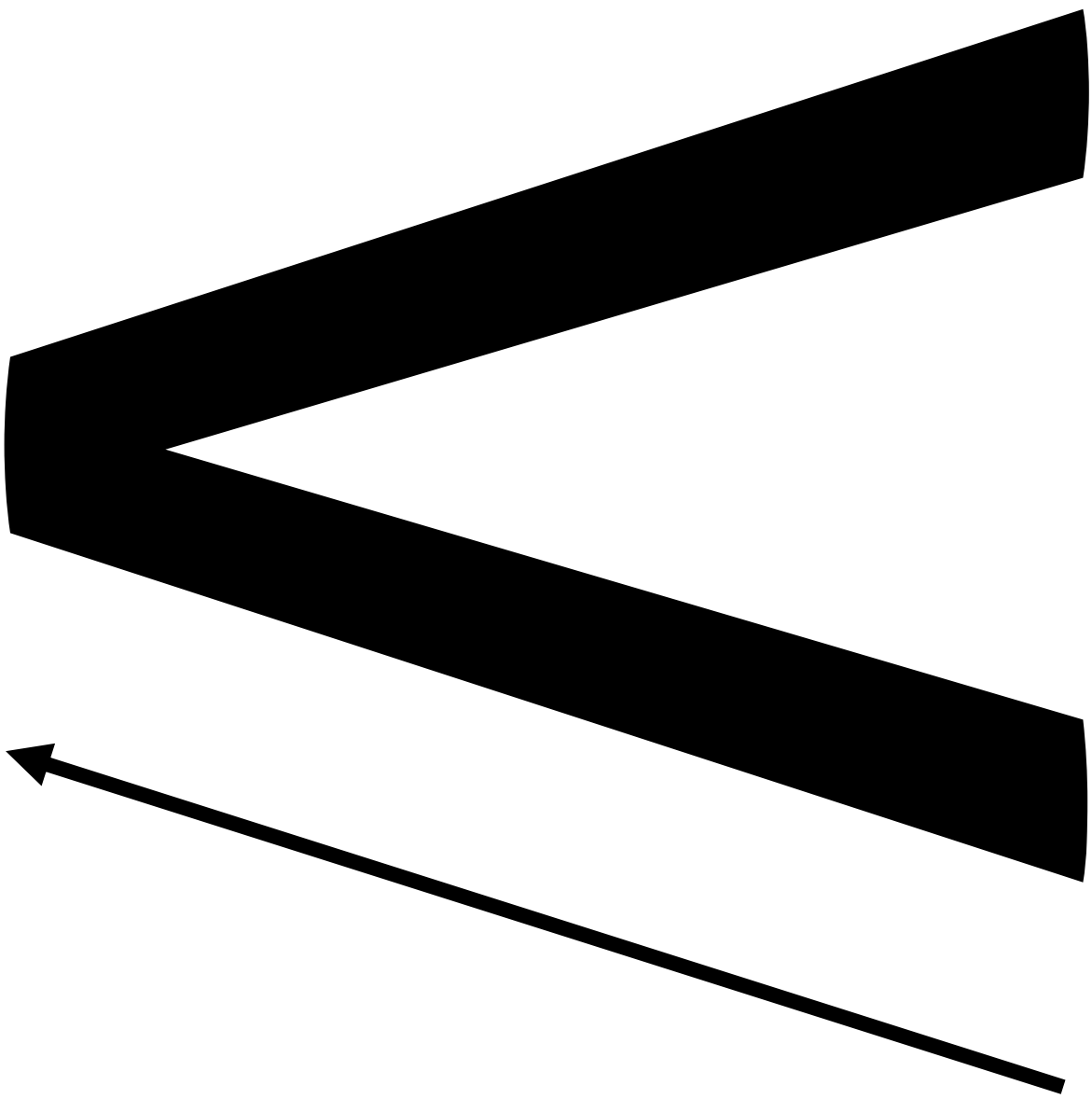
- 57% of defects are made before coding starts
  Lauesen & Vinter – Preventing Requirements Defects

- Working Software over Comprehensive Documentation
  Agile Manifesto

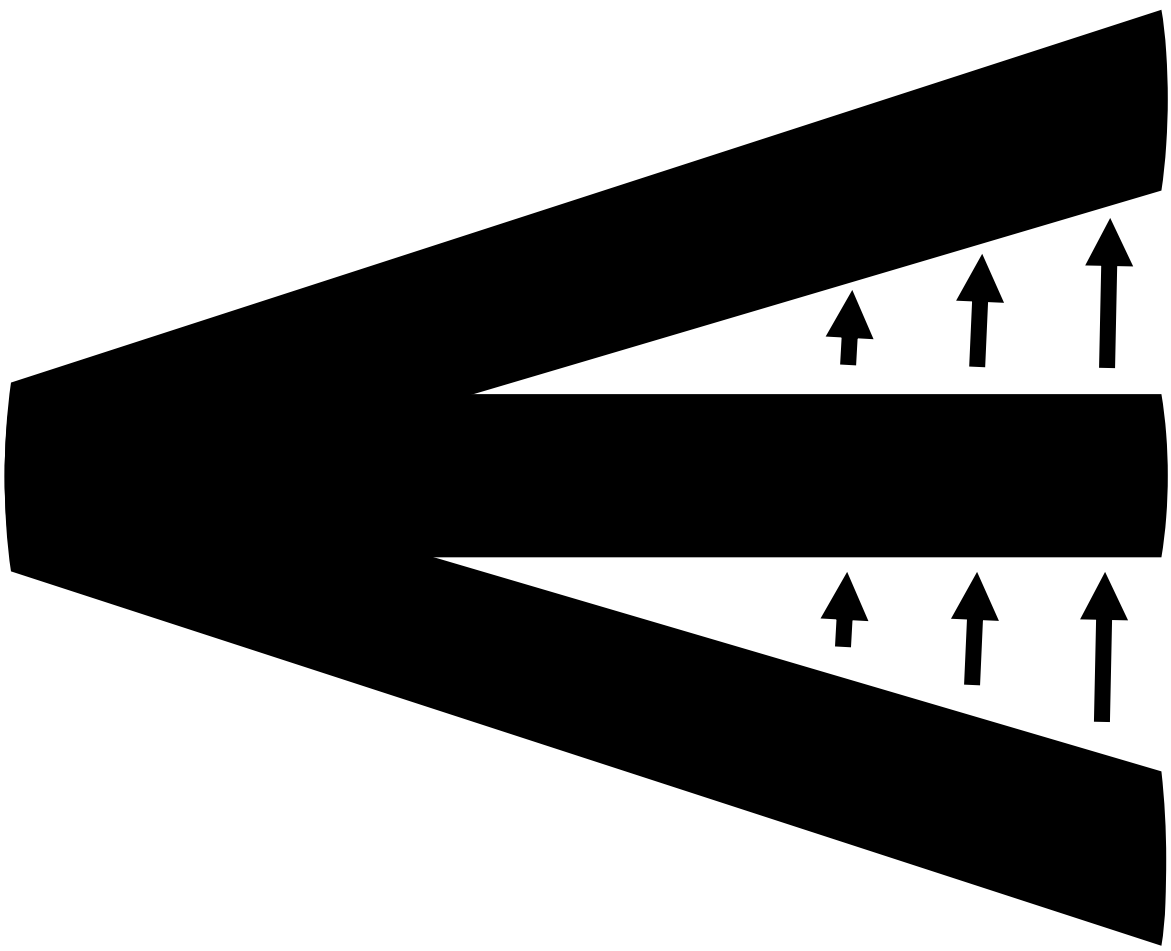
SHIFT LEFT

3 TYPES

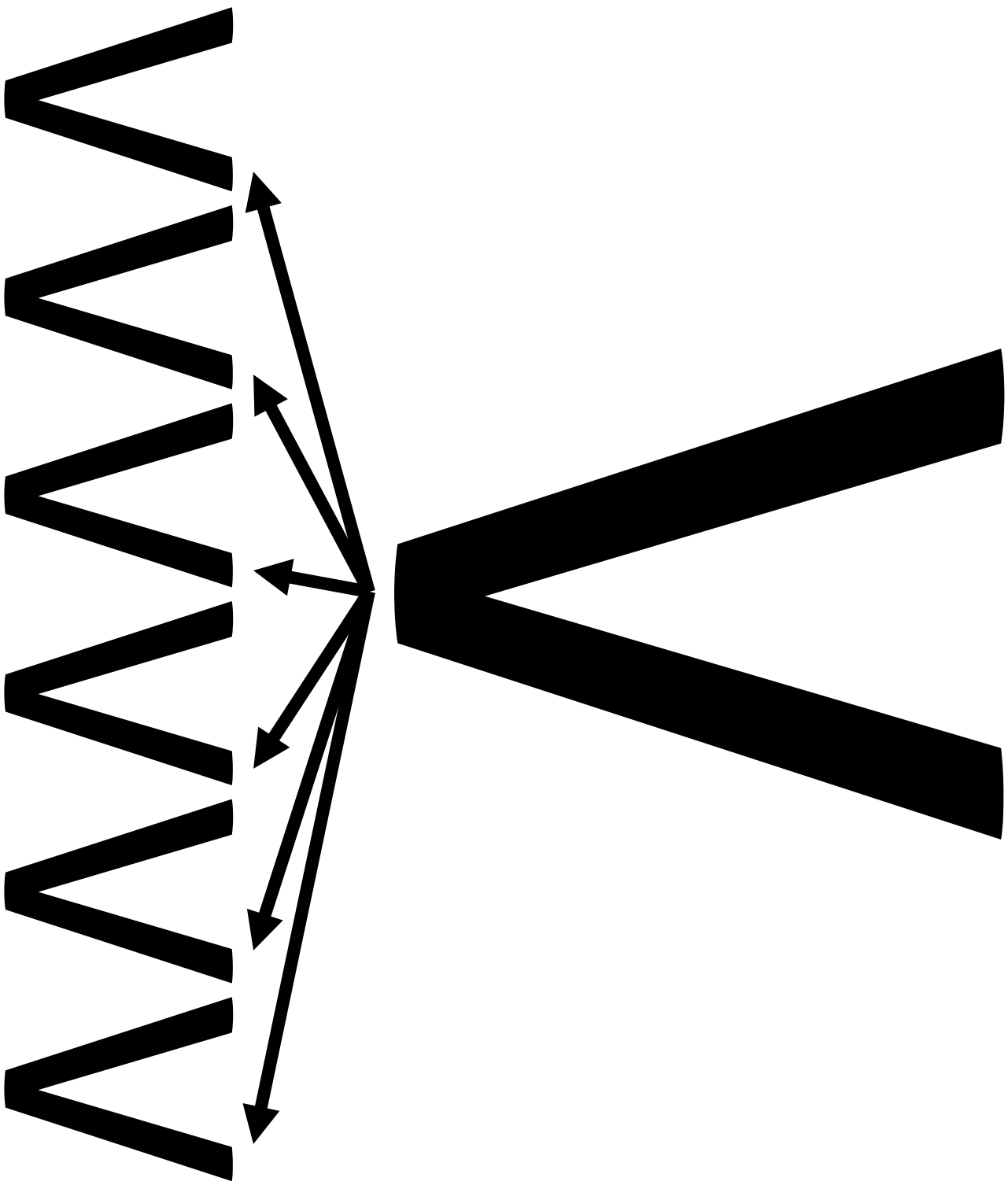
TRADITIONAL



MODEL-BASED

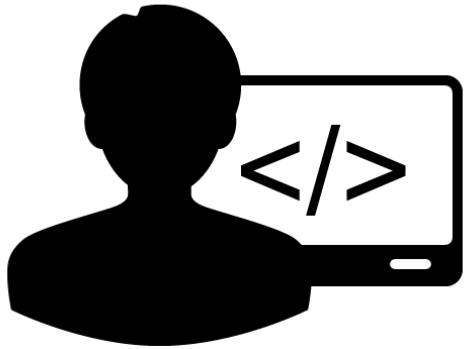


INCREMENTAL



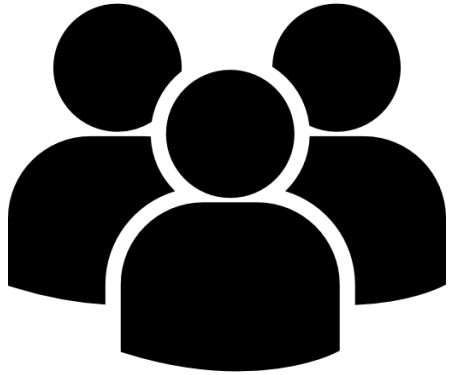
SHIFT LEFT

3 FLAVOURS



Developer Testing

TDD



User Testing

ATDD

BDD

TDD

UNIT TESTING

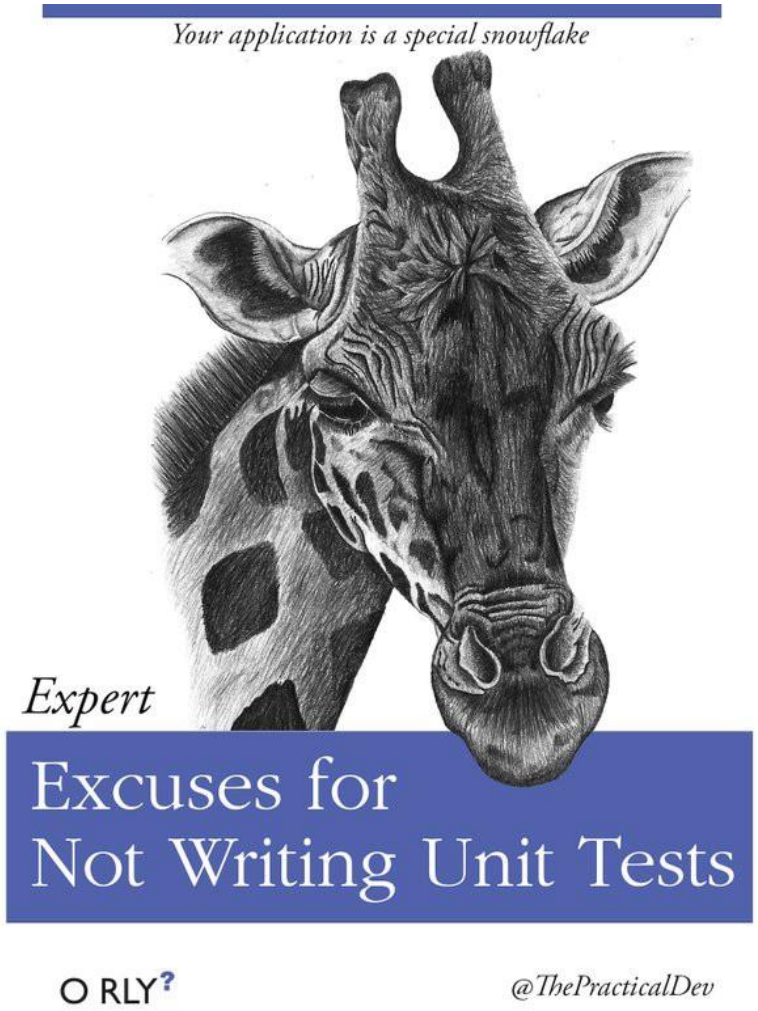
TEST FIRST APPROACH

RED / GREEN / REFACTOR

OBJECTIVE: HIGH QUALITY CODE

- Unit Tests are constructed with assertions that can check
  - That a function performs a calculation and returns an expected result
  - That a function changes the state of an application
  - That a function calls another function

- Unit Tests should be:
  - Automated
  - Easy to implement and maintain
  - Written in the same language as the production code
  - Simple (easy) to run
  - Very fast to execute



- Effective Unit Tests = Trustworthy Unit Tests

- To be trustworthy, they are expected to be FRAID
  - F - Fast
    For immediate feedback. It should be possible to run many tests in very little time.
  - R - Repeatable
    The test can be ran at any time, automatically or user triggered
  - A - Atomic
    All unit tests or a subset can be ran in any random order, without affecting the results
  - I - Isolated
    The Unit Test should strive to exercise only the intended code
  - D - Deterministic
    Every time the test is ran, it should produce the same result

- 3-step pattern for Unit Testing: **Arrange - Act - Assert**

- Arrange:
  - Prepare the execution environment
  - Not always necessary

- Act:
  - Execute the operation being tested
  - Mandatory

- Assert:
  - Check the result produced by the operation
  - Mandatory
  - 1 unit test = 1 assertion

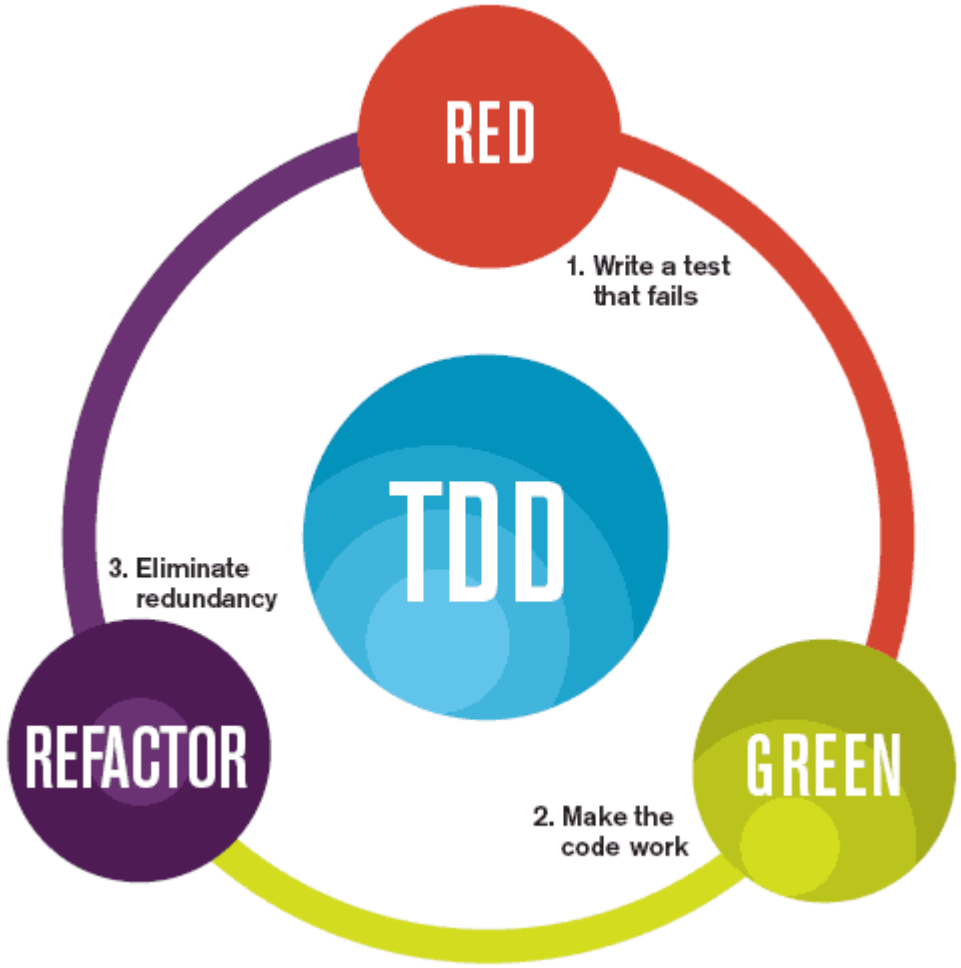
```
[TestMethod]
public void AddTwoNumbers_Success()
{
    // Arrange
    const int firstNumber = 1;
    const int secondNumber = 2;
    var calculator = new Calculator();

    // Act
    var result = calculator.Add(firstNumber, secondNumber);

    //Assert
    Assert.AreEqual(3,result);
}
```

Single iterations of a few minutes, in steps:

1. DEV considers the change up hand
2. DEV identifies the smallest change
3. DEV writes a (failing) unit test
   Describing and identifying an example of the code behavior needed for the change
4. DEV writes the code
5. DEV runs the unit test to verify the change
6. DEV refactors
7. DEV improves the design



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

All unit tests automated before coding starts

Immediate feedback

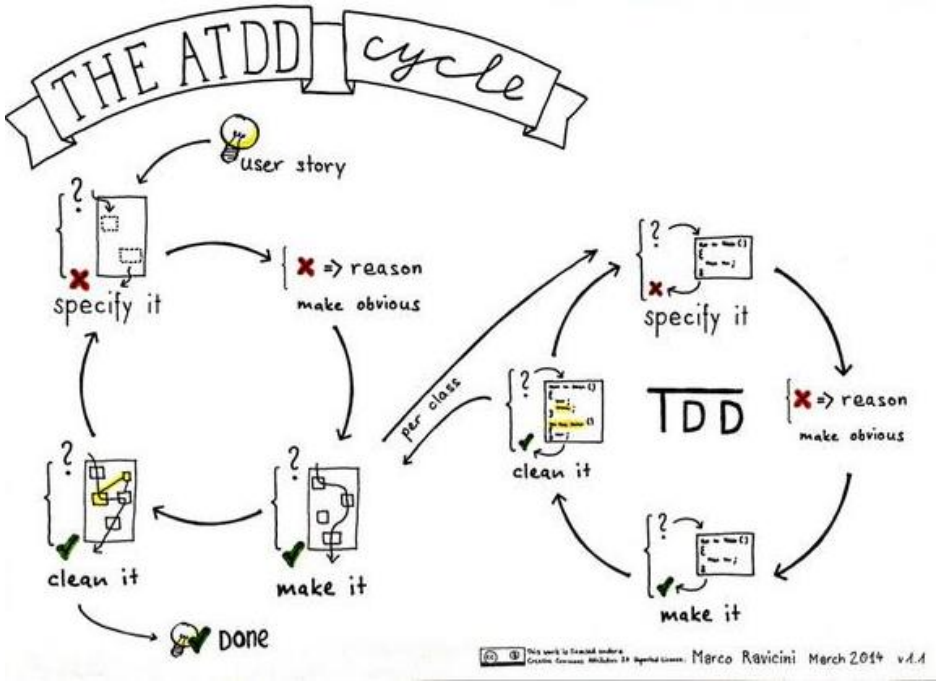
Refactored code

Legacy code

Large bodies of Unit Tests

We have build the thing right

ATDD



DEV METHODOLOGY

ACCEPTANCE TESTING

SPECIFICATION BY EXAMPLE

COLLABORATIVE WORKSHOPS

BUILD THE
THING
RIGHT

TDD

ATDD

BUILD THE
RIGHT
THING

All acceptance tests ready before coding starts

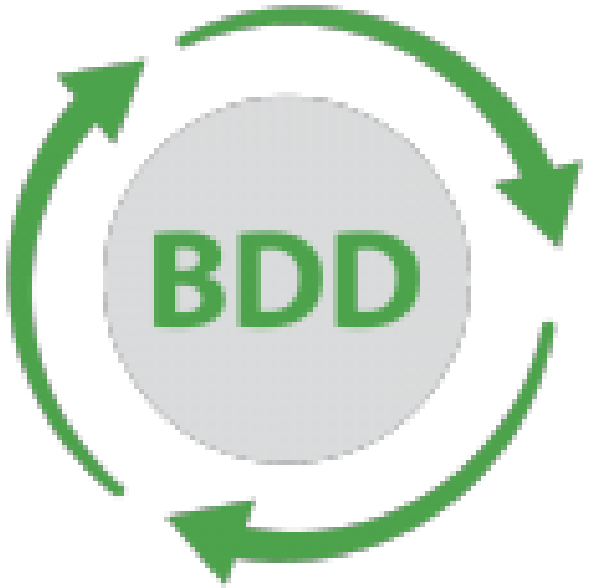
Automation not required

Frameworks

Doesn’t embrace change

Duplicate automation (UT & AT)

BDD



DEV METHODOLOGY

USER STORIES & SCENARIOS

DOMAIN SPECIFIC LANGUAGE

CUSTOMER CENTRIC APPROACH

- Feature <title>
  - AS A <role>
  - I WANT <action>
  - SO THAT <business value>

- Scenario <title>
  - GIVEN <context>
    - AND <more context>
  - WHEN <action>
    - AND <other action>
  - THEN <outcome>
    - AND <more outcomes>

- Description of the Feature
  - Stakeholder and/or user role
  - Action to be undertaken
  - Business value provided (rationale)

- Description of the Scenario
  - Preconditions
  - Actions
  - Expected outcomes

User Stories vs. BDD Scenarios

- User Stories capture business requirements & HL functionality
- Scenarios are derived from User Stories, with more details and specifics

- User Stories (and acceptance criteria) may be automated
- Scenarios must be automated

- User Stories are documented visually (task board)
- Scenarios are documented in code (or phrase templates)

Less documentation

Link with iterative SDLC

Frameworks

Learning curve

Change documentation style

SHIFT LEFT

SUMMARY

WTFM?

- TDD : by developers
  ATDD & BDD : by users
- TDD & BDD require automation
  ATDD: automation optional
- TDD not prioritized
  ATDD & BDD: priority by user
- ATDD & BDD support collaboration
- D = Development
  Automated tests are NOT the goal
- GOAL: push defect curve to the left