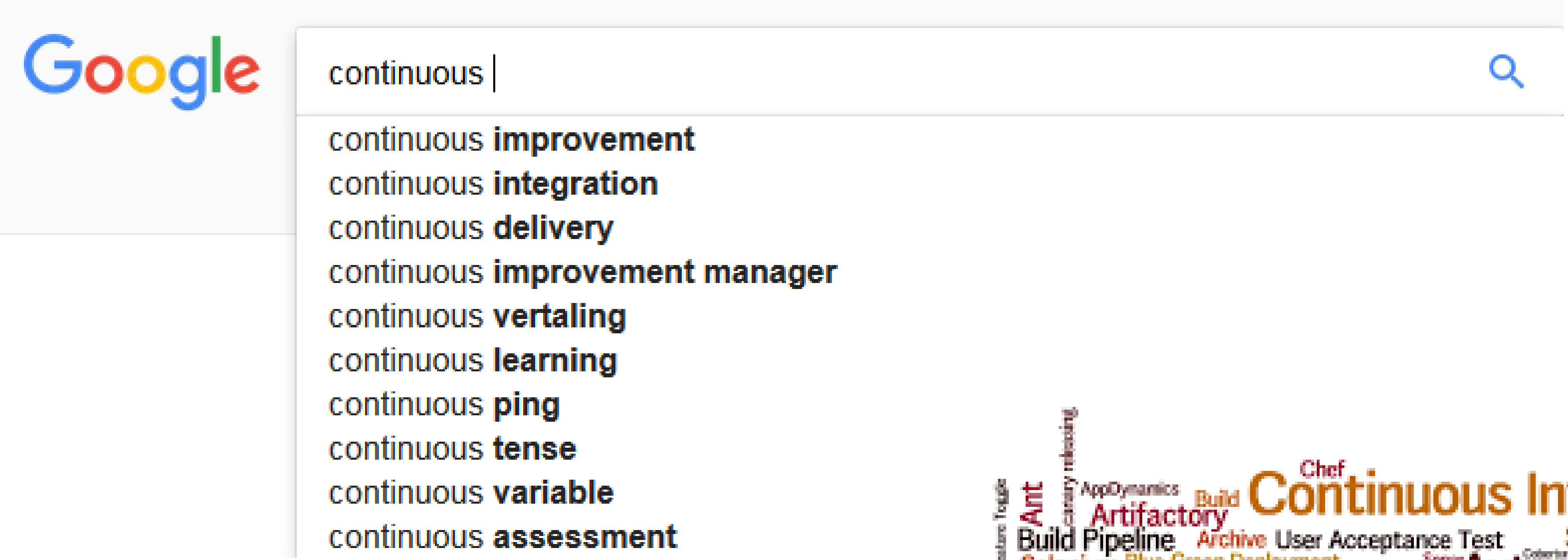
moveIT

FOR FASTER, BETTER AND MORE EFFICIENT IT

SHIFT LEFT

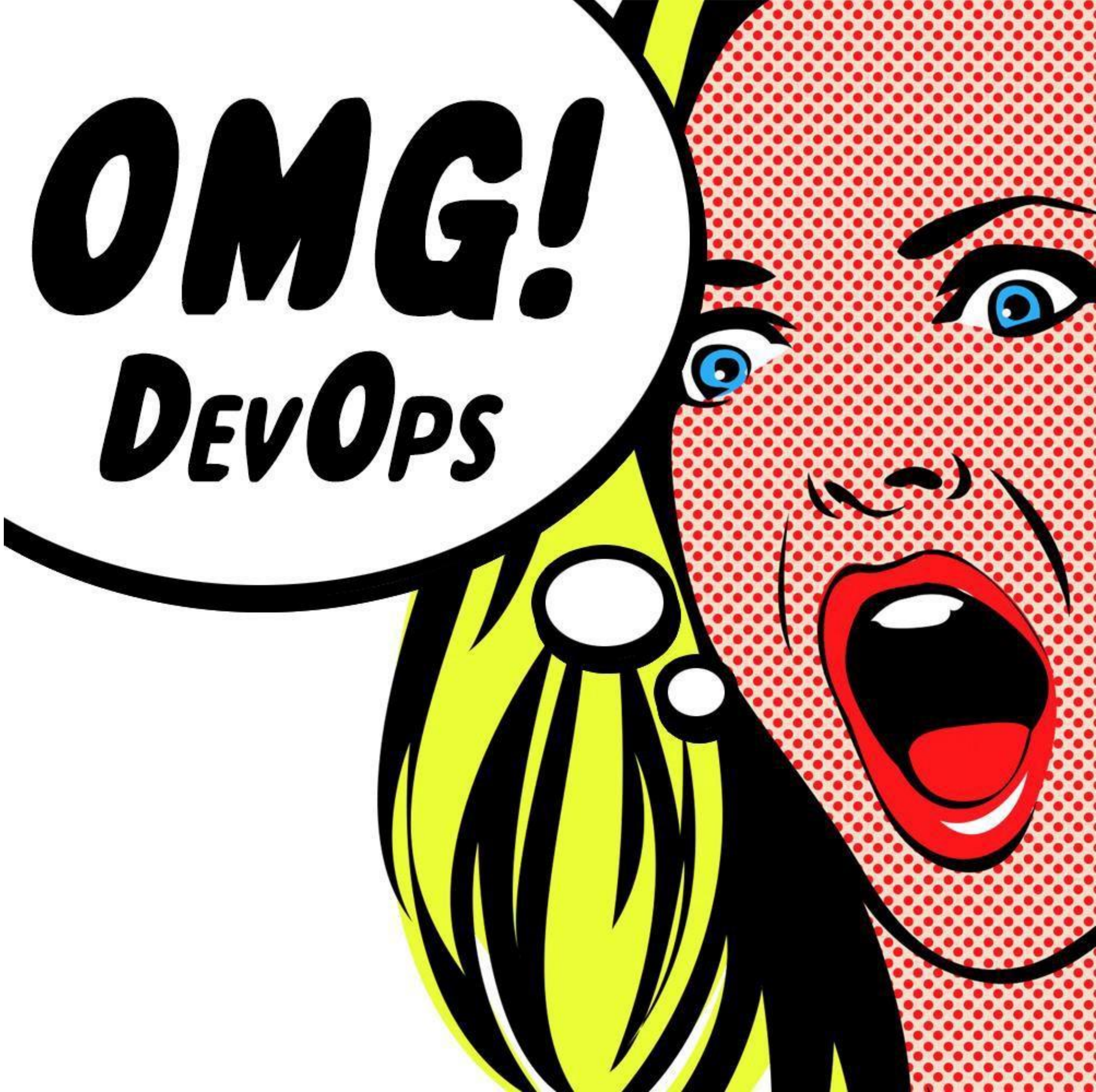
@mpilaeten
Michaël Pilaeten

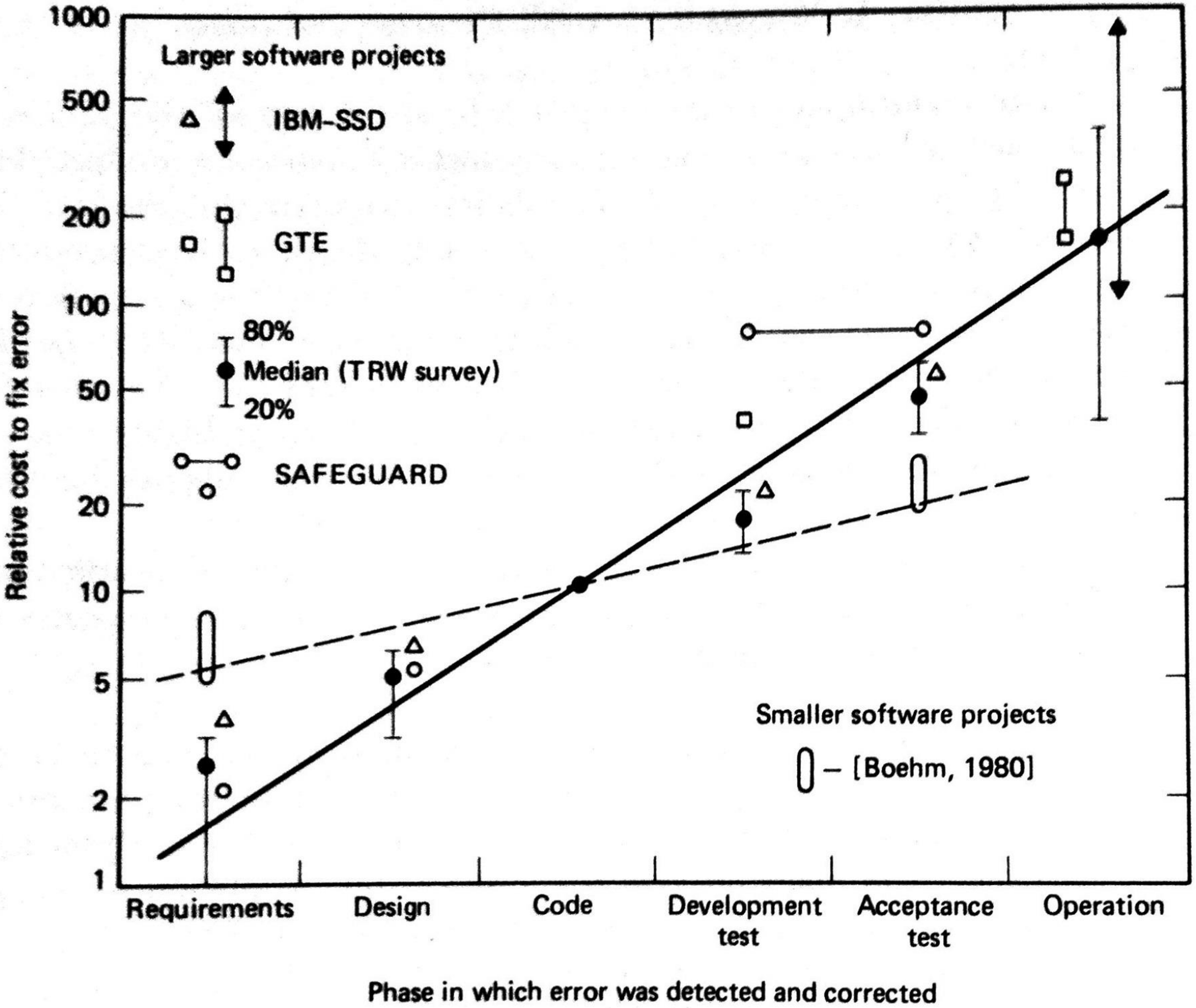


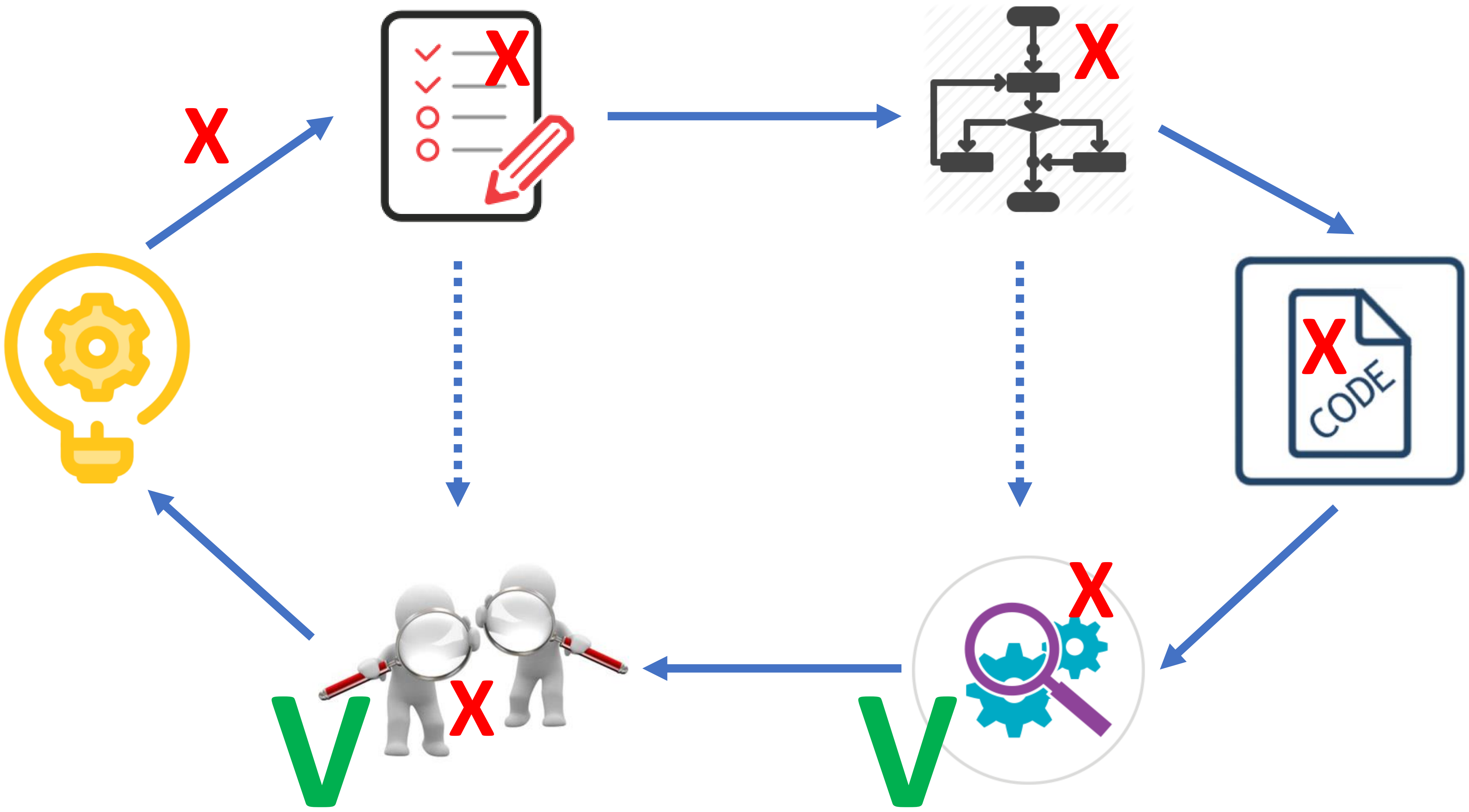
continuous

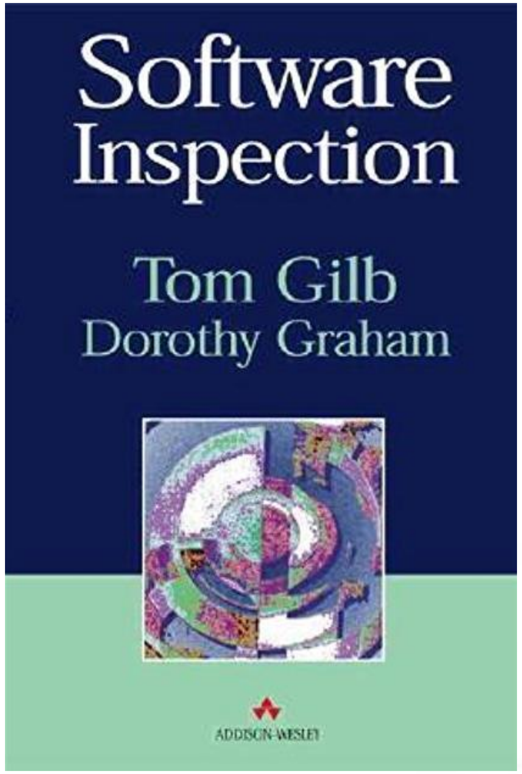
continuous **improvement**
continuous **integration**
continuous **delivery**
continuous **improvement manager**
continuous **vertaling**
continuous **learning**
continuous **ping**
continuous **tense**
continuous **variable**
continuous **assessment**

Feature Toggle
Ant
canary releasing
AppDynamics
Build
Chef
Continuous Integration
Artifactory
Build Pipeline
Archive
User Acceptance Test
Subversion
Selenium
Blue-Green Deployment
Sonar
Cobertura
FindBugs
Automation
Continuous Delivery
Checkstyle
Smoke Test
Zero-Downtime
soapUI
Review
Dynatrace
JaCoCo
Continuous Deployment
JMeter
Jenkins
JUnit
Monitoring
Commit Stage
Puppet
Git
Maven
ZAP


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

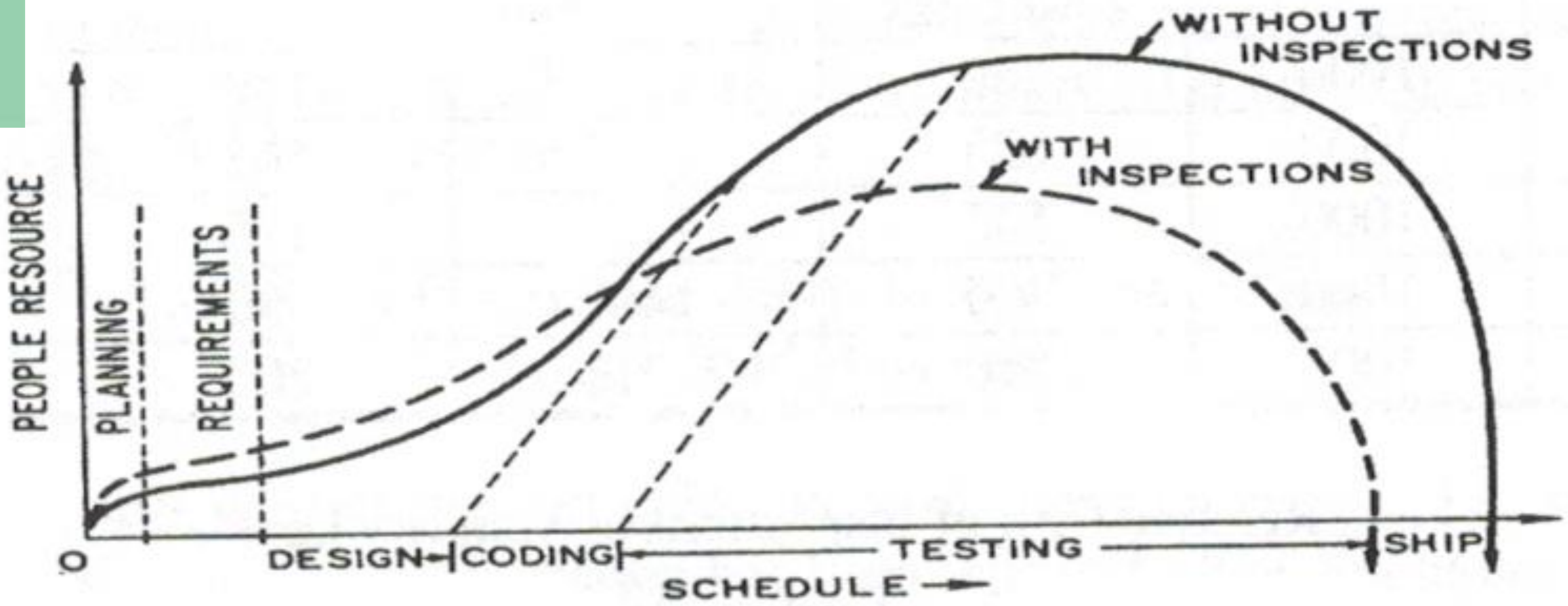

X
X
X
X
CODE
X
V
X
V



Software Inspection

Tom Gilb
Dorothy Graham

ADDISON-WESLEY



SHIFT LEFT

WHY ?

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

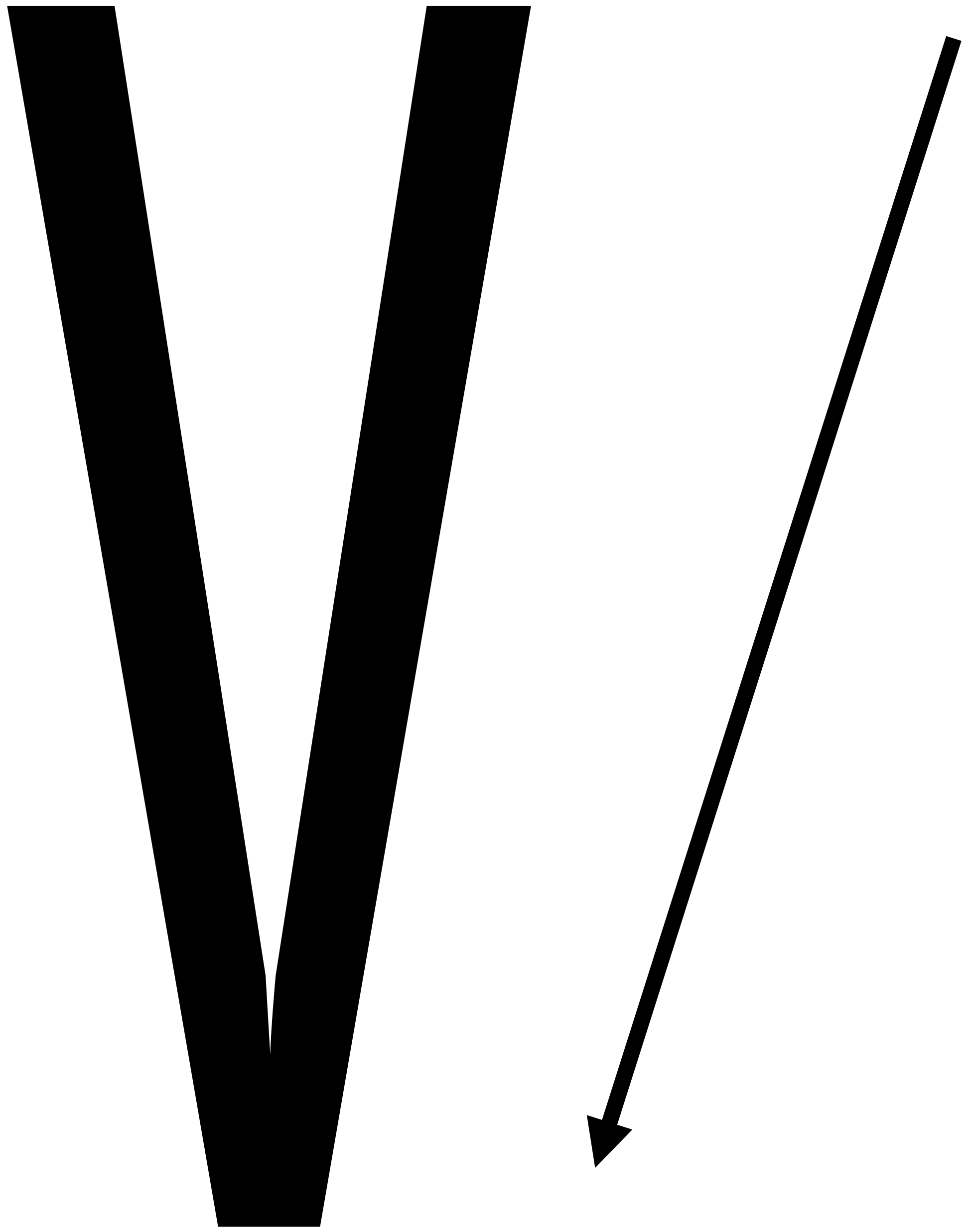
- 57% of defects are made before coding starts
  Lauesen & Vinter – Preventing Requirements Defects

- Working Software over Comprehensive Documentation
  Agile Manifesto

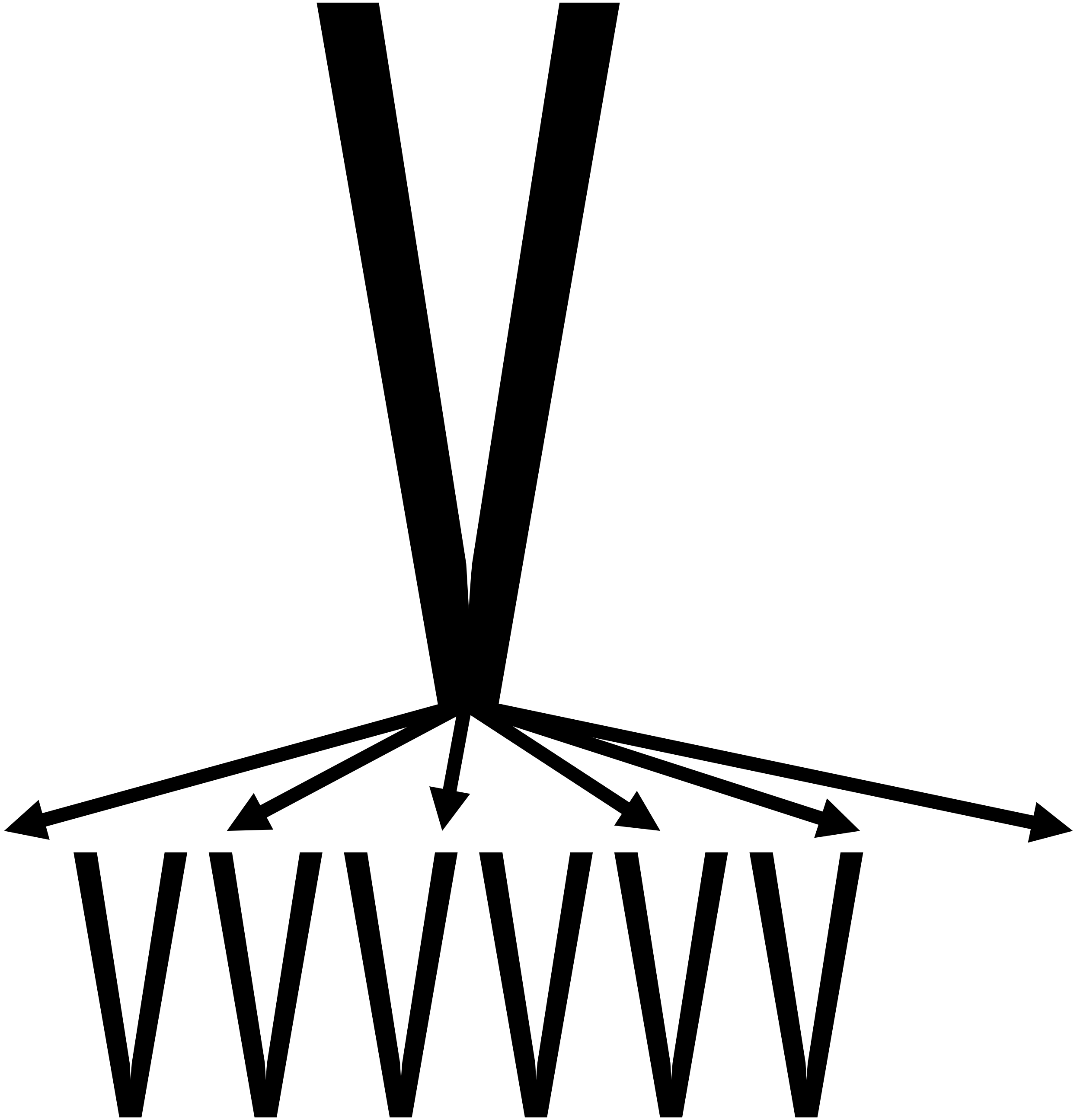
SHIFT LEFT

3 TYPES

TRADITIONAL



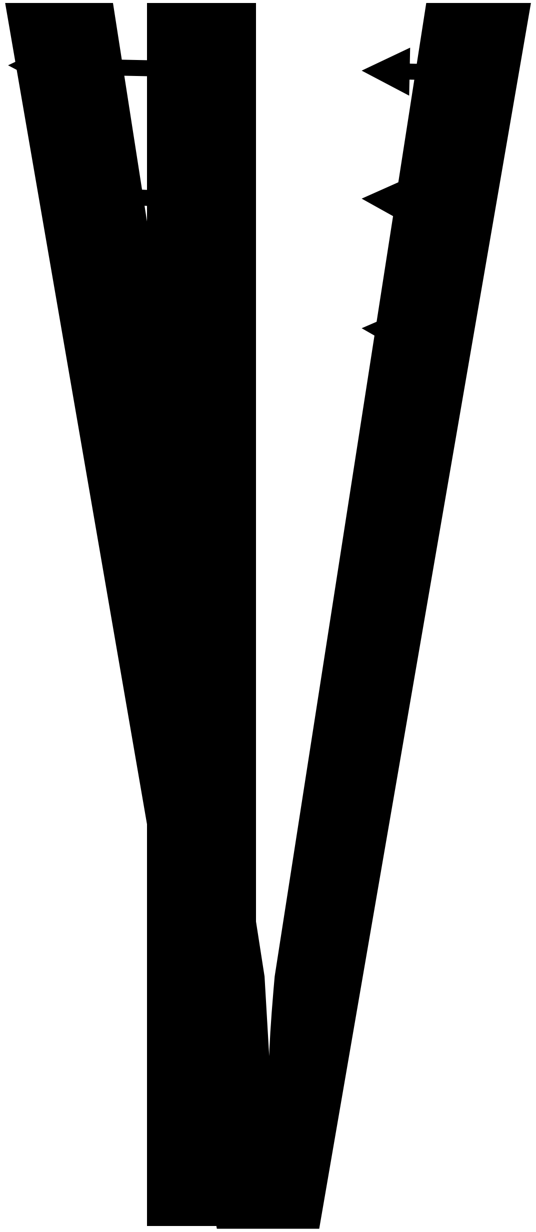
INCREMENTAL



V

VVVVVV

MODEL-BASED



SHIFT LEFT

3 FLAVOURS

Developer Testing

TDD

User Testing

ATDD

BDD

TDD

TDD

ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT

UNIT TESTING

TEST FIRST APPROACH

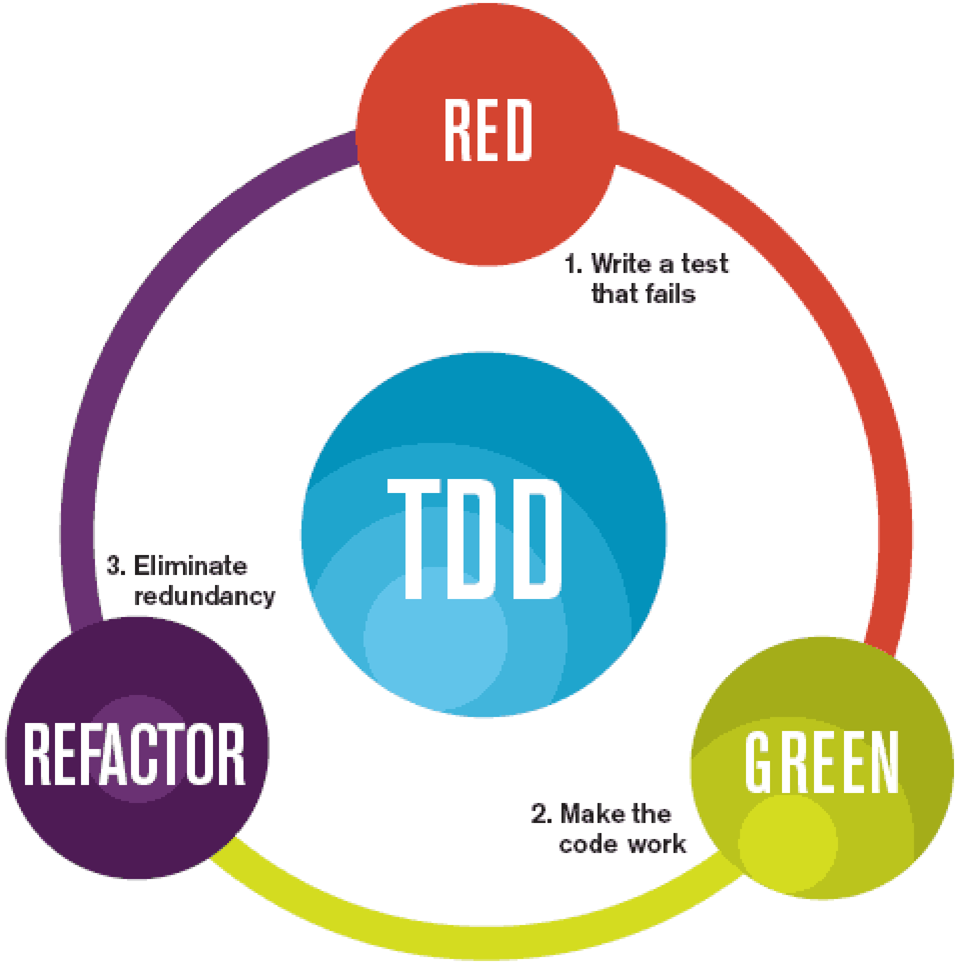
RED / GREEN / REFACTOR

OBJECTIVE: HIGH QUALITY CODE

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

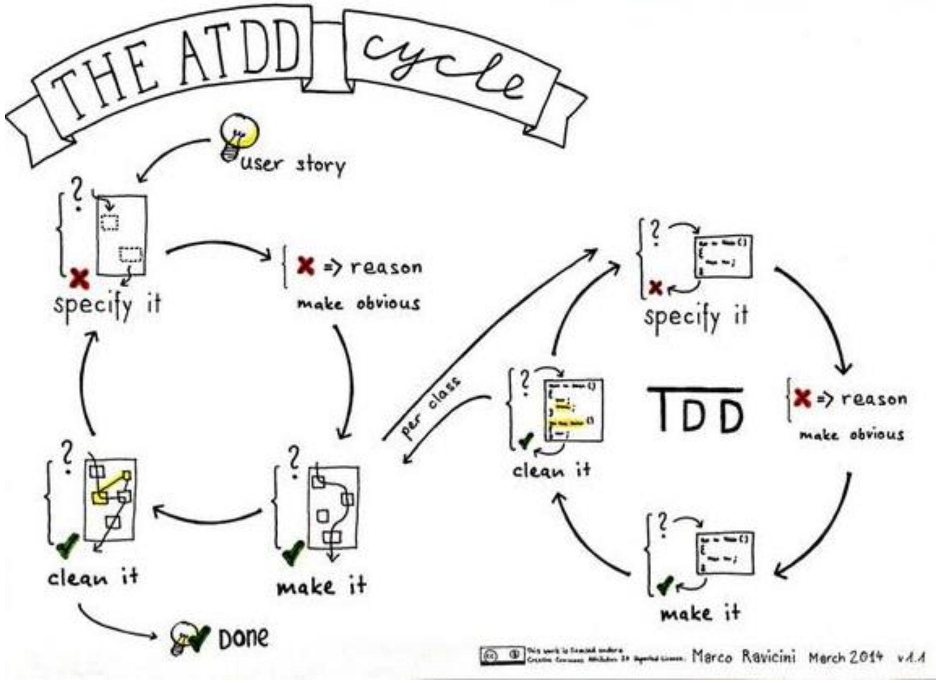
Refactored code

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

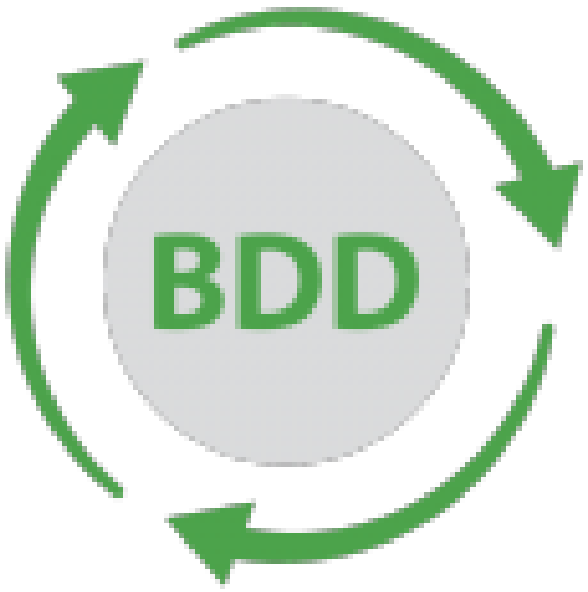
Automation not required

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

Less documentation

Link with iterative SDLC

Frameworks

Learning curve

Change documentation style

SHIFT LEFT

SUMMARY

WIIFM?

- 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