

45426: Teste e Qualidade de Software

# Software quality models & Quality in the software process

Ilídio Oliveira

v2024-04-23

# Learning objectives

Define software quality and recognize the elements of software quality.

Explain the structure of ISO/IEC 25010:2011 Quality Model.

Distinguish between internal and external quality factors and give examples.

Explain the principles of agile testing

Distinguish Agile Testing from Continuous testing

Explain the insertion of QA in the software engineering process according to the TDD

Explain the TDD cycle and argue in favor its adoption

Which tools can be used to address the “quality dilemma”?

Software runs the world...

## THE RISKS DIGEST

Forum On Risks To The Public In Computers And Related Systems

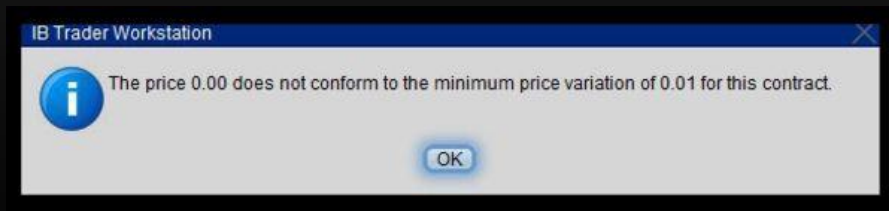
[ACM](#) Committee on Computers and Public Policy, [Peter G. Neumann](#), moderator

<http://catless.ncl.ac.uk/Risks/>

# Elements of software quality

We can recognize the lack of quality...

Can we be systematic about the **elements of software quality**?



Markets

## Oil Crash Busted Broker's Computers and Inflicted Big Losses

By [Matthew Leising](#)

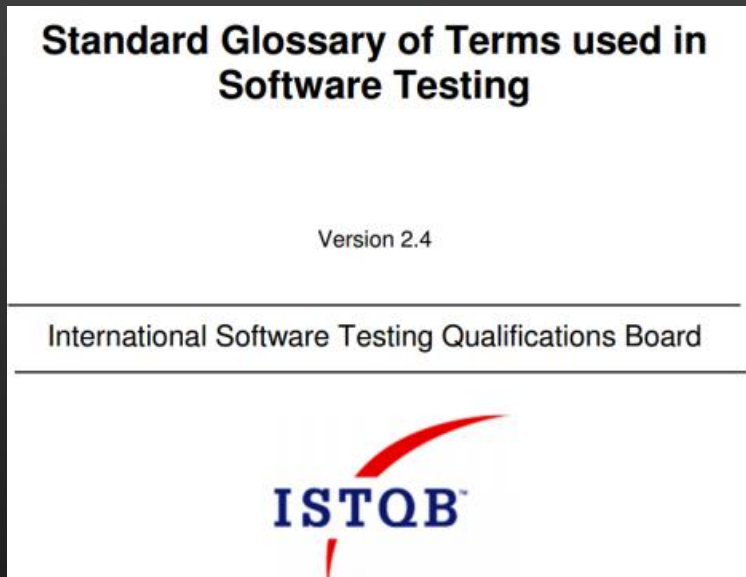
May 8, 2020, 3:48 PM GMT+1 Updated on May 8, 2020, 10:07 PM GMT+1

- Interactive Brokers users couldn't trade when oil broke zero
- Incident will cost firm more than \$100 million, chairman says

LIVE ON BLOOMBERG  
[Watch Live TV](#) >  
[Listen to Live Radio](#) >



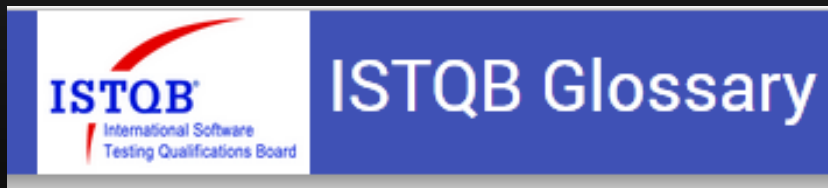
# Software quality defined



## ISTQB

Quality: The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations. [After ISO 25010]

Software quality: the totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs. [After ISO 9126]

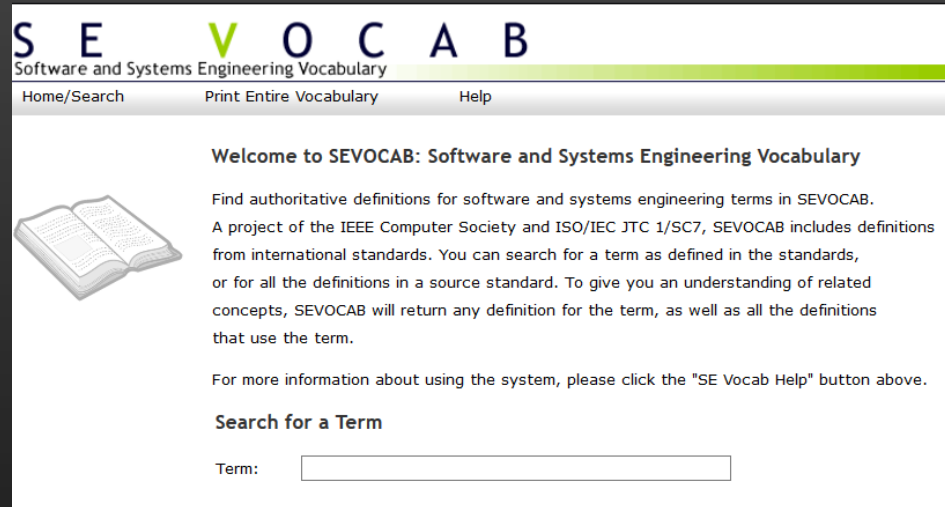


<https://glossary.istqb.org/>

# Software (product) quality defined

The degree to which a software product satisfies stated and implied needs when used under specified conditions

(ISO/IEC 25010:2011 Systems and software engineering--Systems and software Quality Requirements and Evaluation (SQuaRE)--System and software quality models, 4.3.13)

The screenshot shows the SEVOCAB website. At the top, the letters 'S E V O C A B' are displayed in a large, spaced-out font, with 'V' in green. Below this, the text 'Software and Systems Engineering Vocabulary' is written. A navigation bar contains three links: 'Home/Search', 'Print Entire Vocabulary', and 'Help'. The main content area has a heading 'Welcome to SEVOCAB: Software and Systems Engineering Vocabulary' and an illustration of an open book. The text explains that the site provides authoritative definitions for software and systems engineering terms, including definitions from international standards and source standards. It also mentions that related concepts and definitions using the term are provided. A search section titled 'Search for a Term' includes a text input field labeled 'Term:'.

<http://www.computer.org/sevocab>

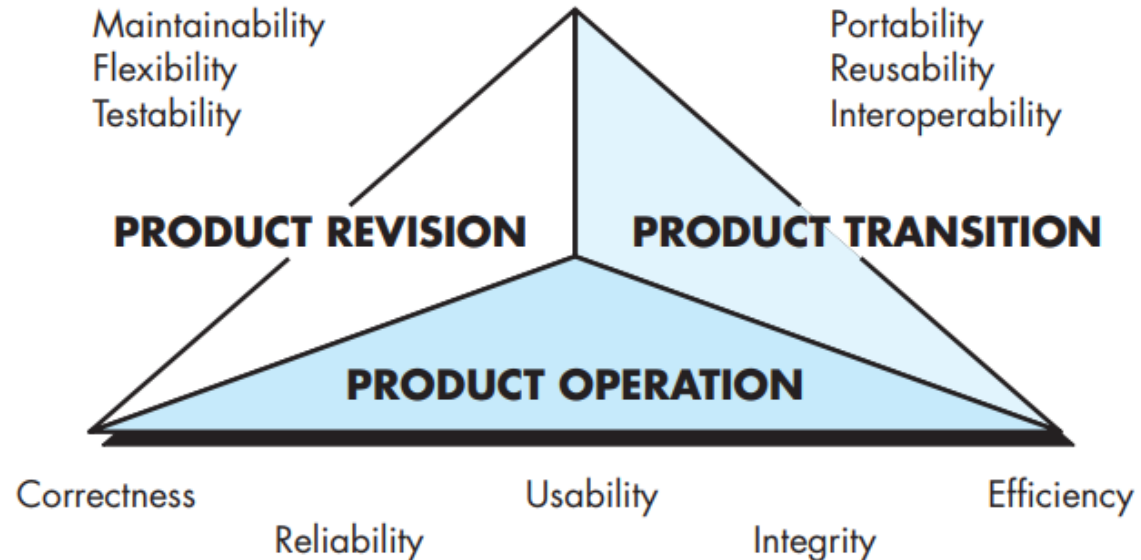
*“... the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value. These stated and implied needs are represented . . . by quality models that categorize product quality into characteristics, which in some cases are further subdivided into sub-characteristics.”*

# Software quality models

# McCall's software quality factors [McC77]

**FIGURE 19.1**

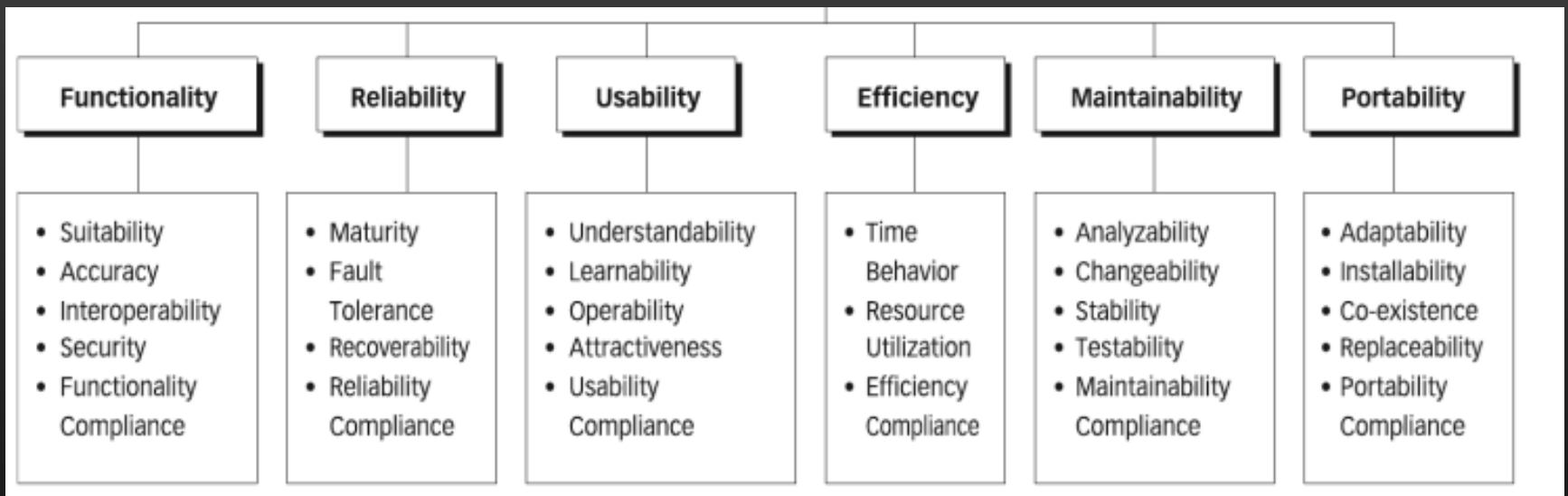
**McCall's  
software  
quality factors**



Useful categorization of factors that affect software quality. These software quality factors, focus on three important aspects of a software product: its operational characteristics, its ability to undergo change, and its adaptability to new environments.



# ISO 9126-1:2001 Software engineering – Product quality – Part 1: Quality model



See: Table of Characteristics and sub-characteristics for the ISO 9126-1 Quality Model

The quality model is the cornerstone of a product quality evaluation system. The quality model determines which quality characteristics/factors will be considered when evaluating the properties of a software product. Factors do not necessarily lend themselves to direct measurement

# Upgrade: ISO/IEC 25010:2011

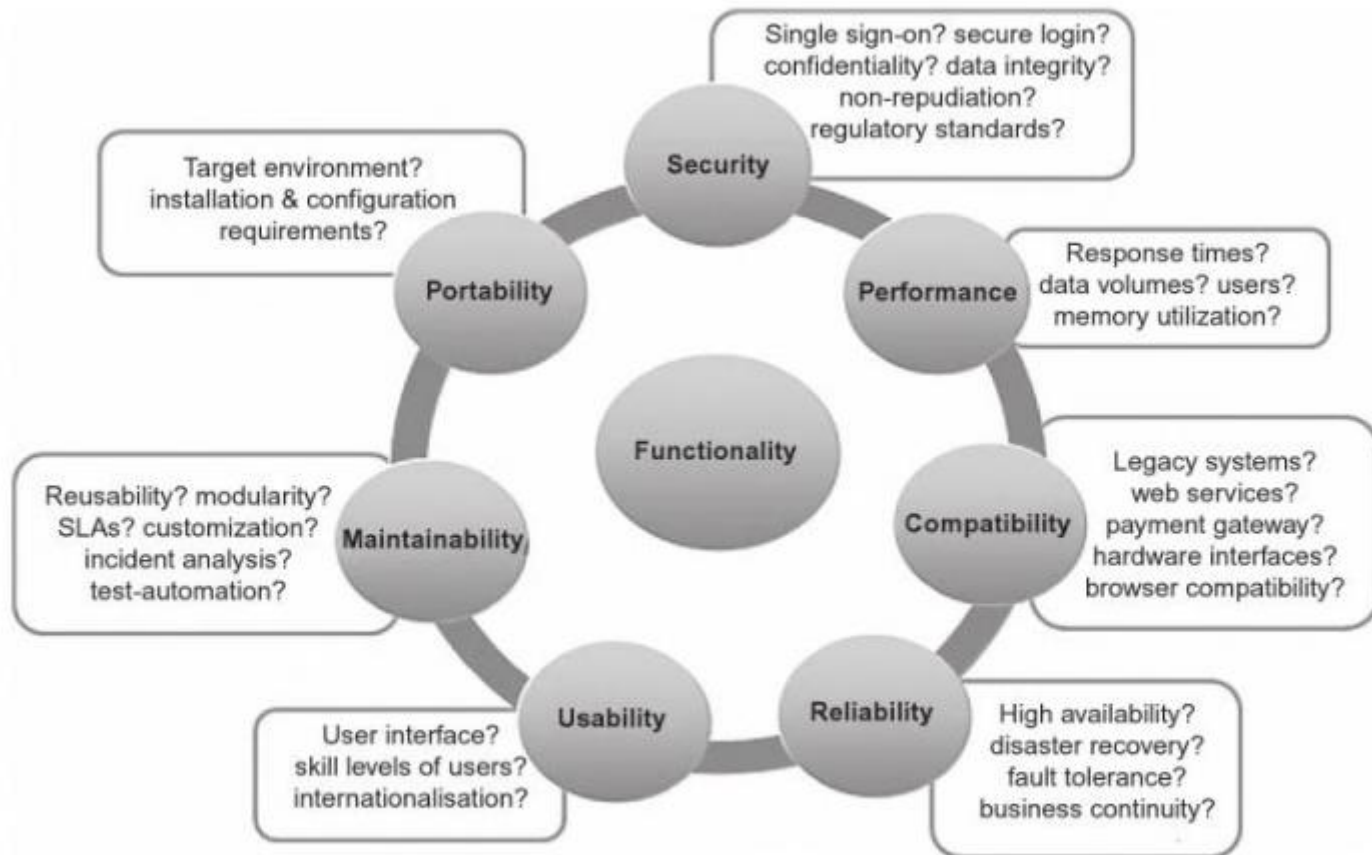
## Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models



<https://blog.codacy.com/iso-25010-software-quality-model/>

Functional quality characteristics/factors are just a part of the story...

# Holistic view: need to consider quality characteristics in the product conception and development



**FIGURE 6.2**

Holistic View of Product Quality.

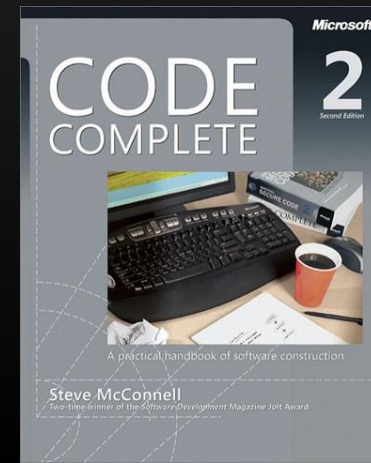
# Internal and external quality factors (McConnell, Code Complete)

External: those that the users/customers are aware of

- Correctness (constructed free from faults)
- Usability
- Efficiency (use of resources)
- Reliability (long times between failures)
- Integrity (state is always consistent)
- Adaptability (usable in new contexts)
- Robustness

Internal: those that the programmer is aware of

- Maintainability.
- Flexibility + portability (fits new uses or environments)
- Reusability (parts can be reused)
- Readability + understandability
- Testability



# Tests management

# Practices of SQA

## Testing

Software configuration management

Versions management

Code improvement

Reviews, shared practices, static analysis,...

Continuous integration/Continuous Delivery

Formal methods (out of scope for our course)

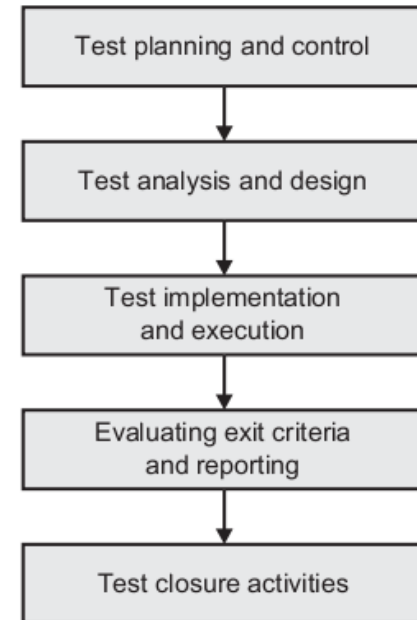
...

# Testing as a (parallel) process

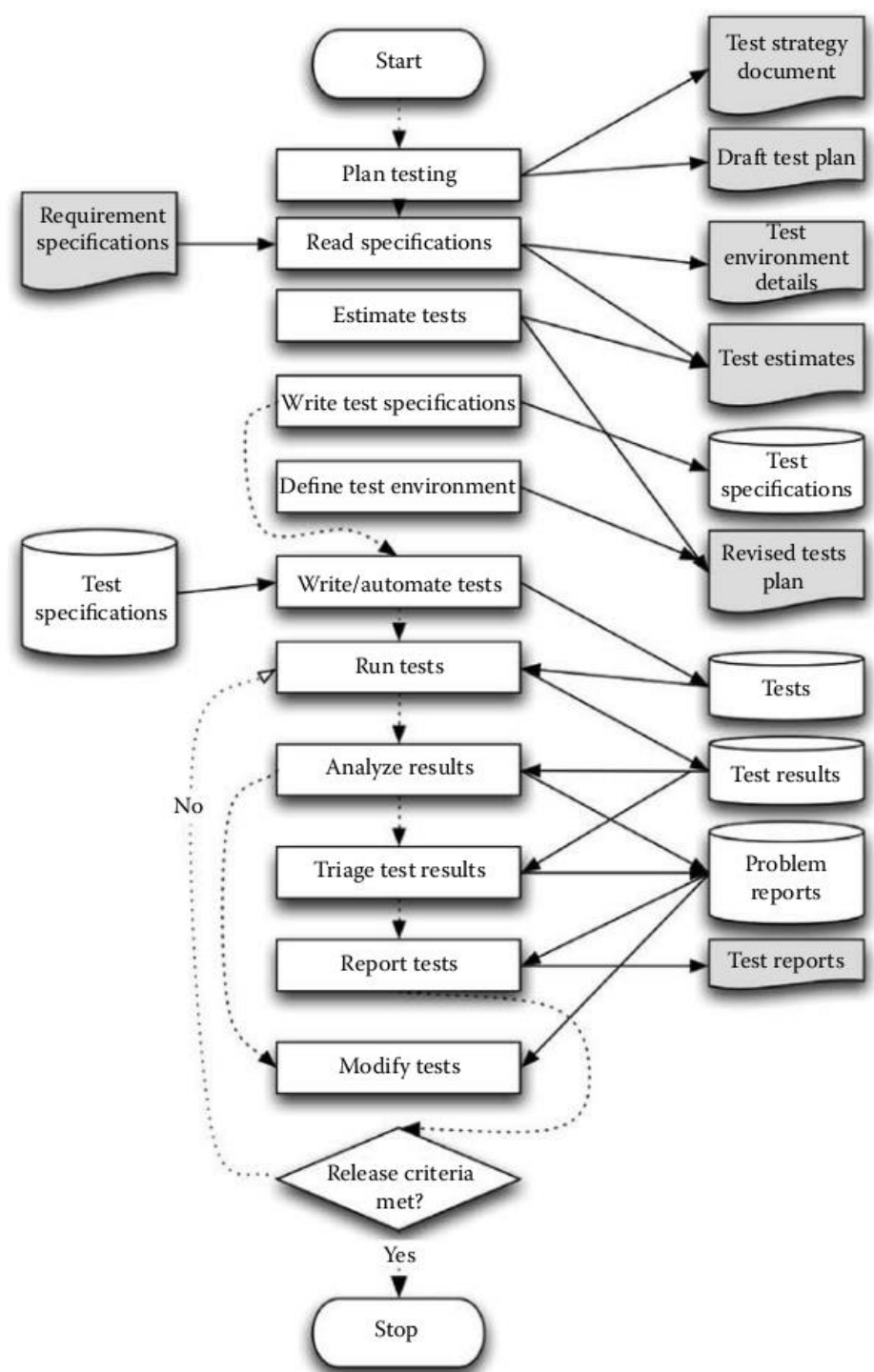
Testing methodology should answer:

What to test , how to test, when to stop, who does what

## Fundamental test process



# Detailed test process activities

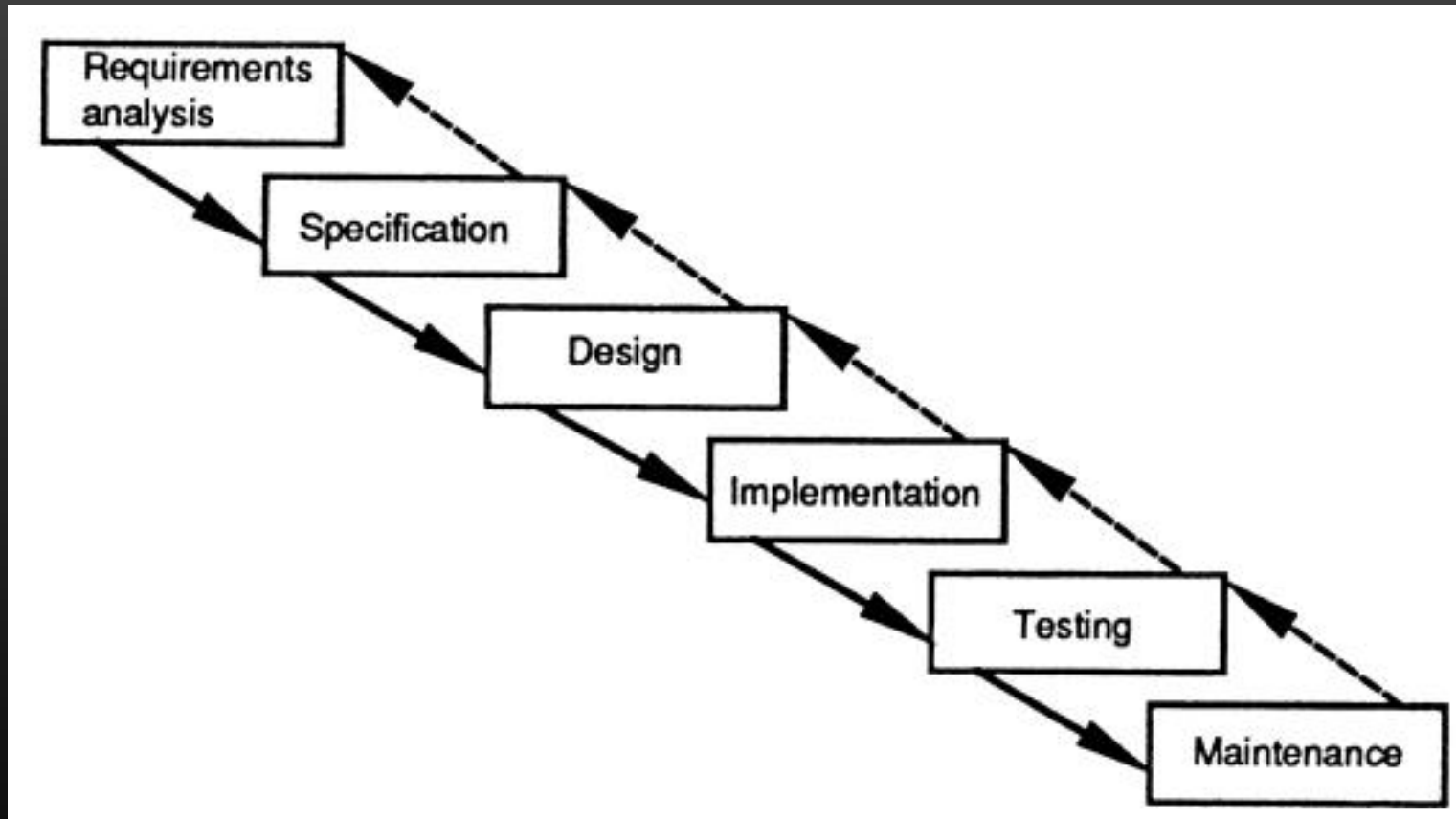


P. Farrell-Vinay, *Manage Software Testing*



# Tests in the (agile) engineering process

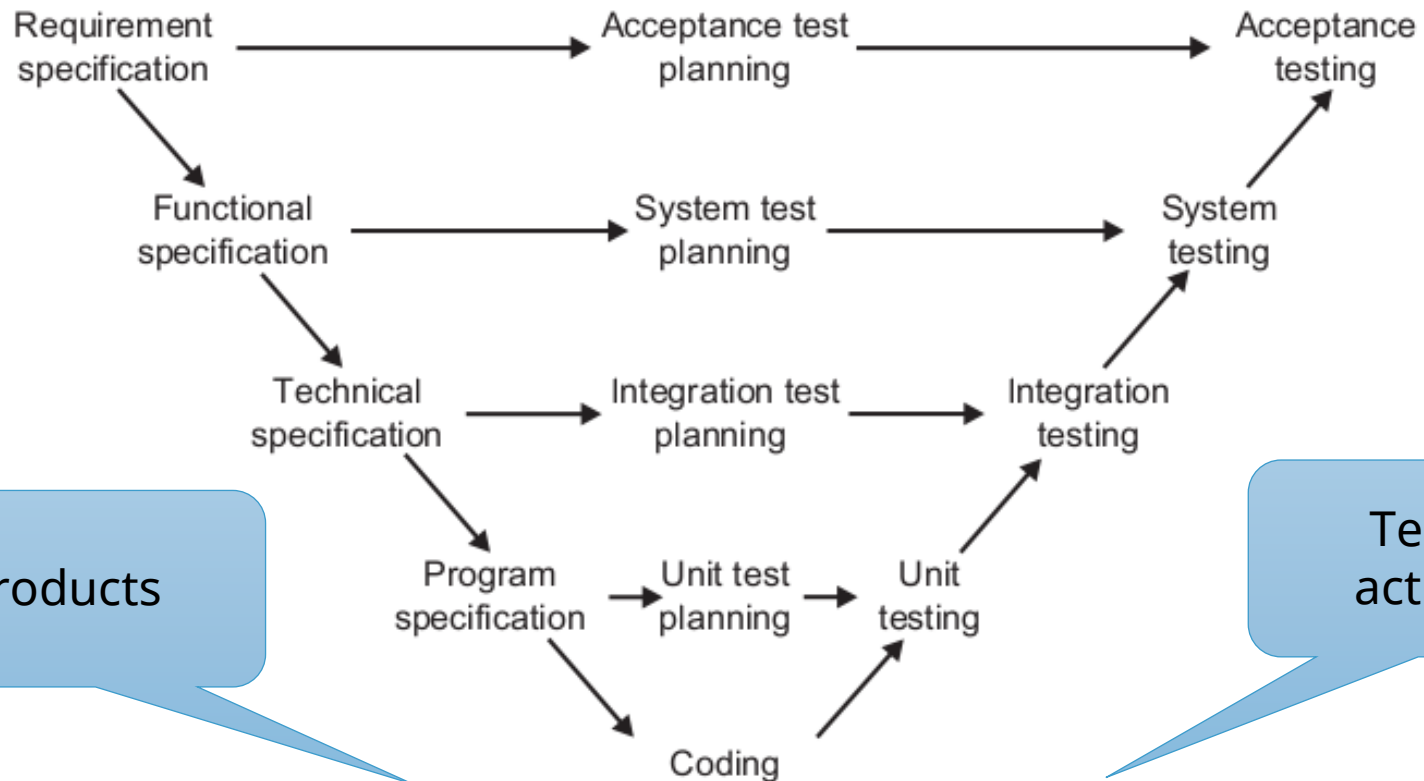
# “Classical” engineering approach: **Waterfall model**



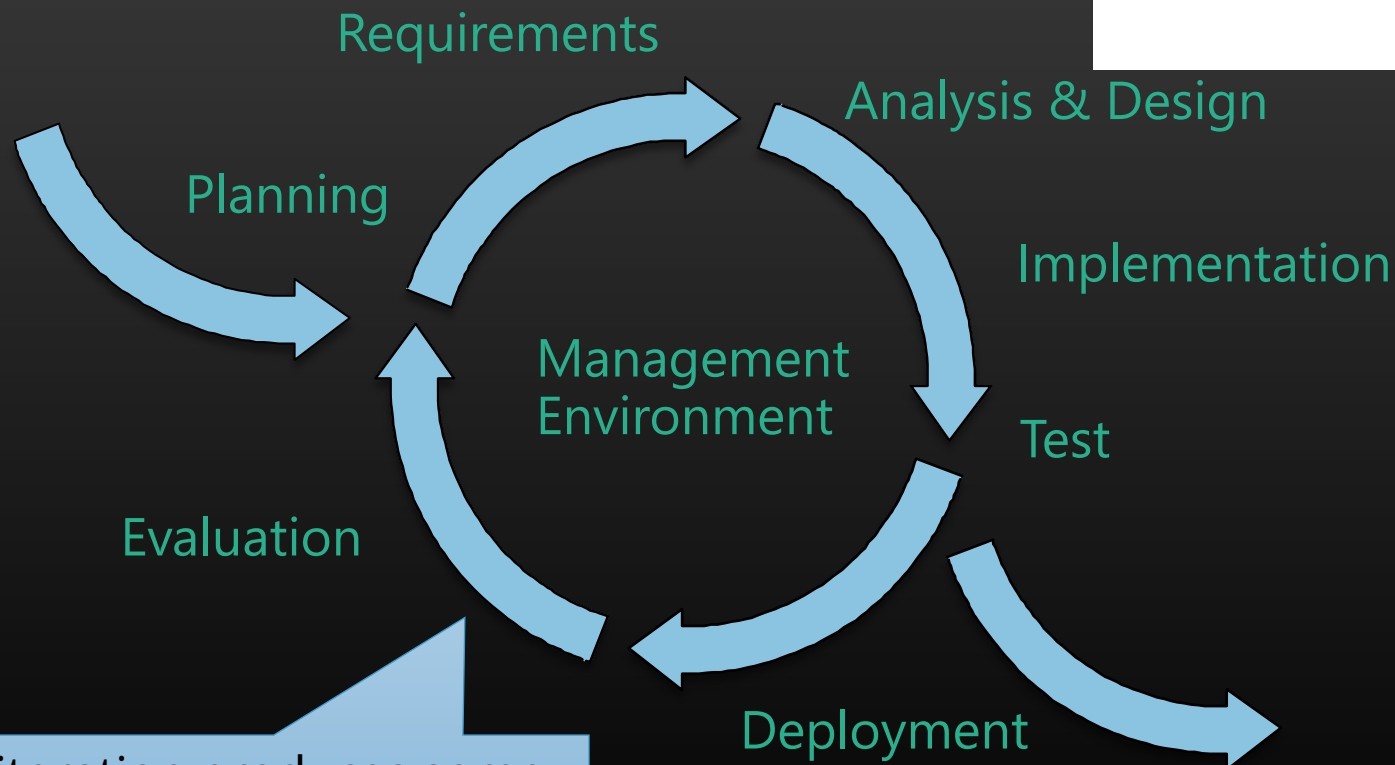
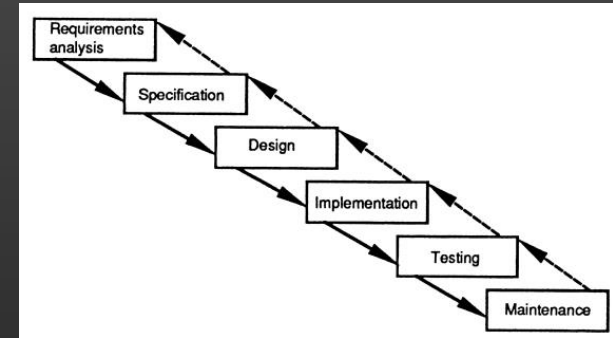
W. Royce, “Managing the Development of Large Software Systems,” *Proc. Westcon*, IEEE CS Press, 1970, pp. 328-339.

# Test lifecycle and sw development lifecycle: the V-Model approach

**Figure 2.2** V-model for software development

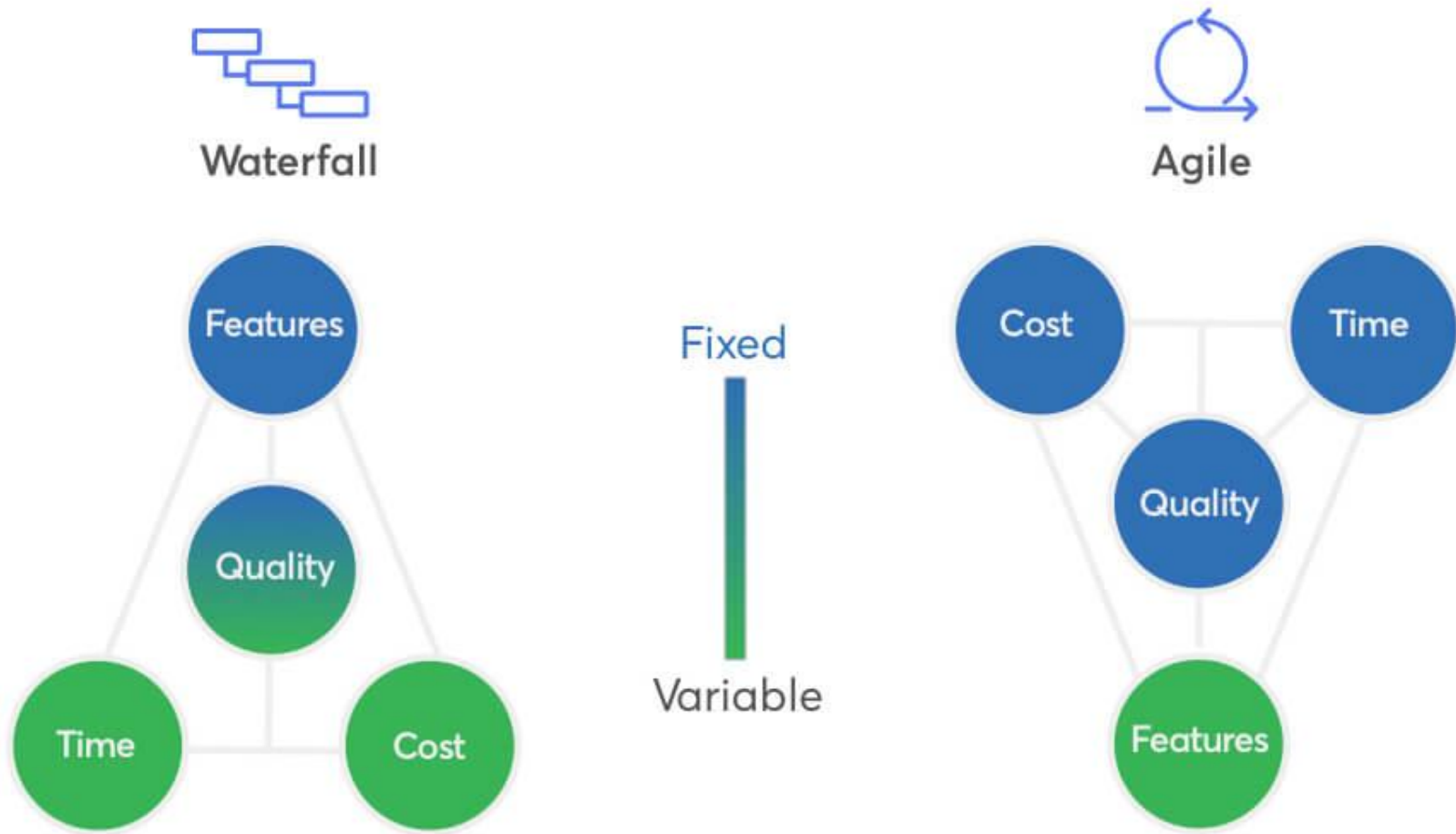


# Evolutionary development “conquers” the product along short development cycles



Each iteration produces some executable result

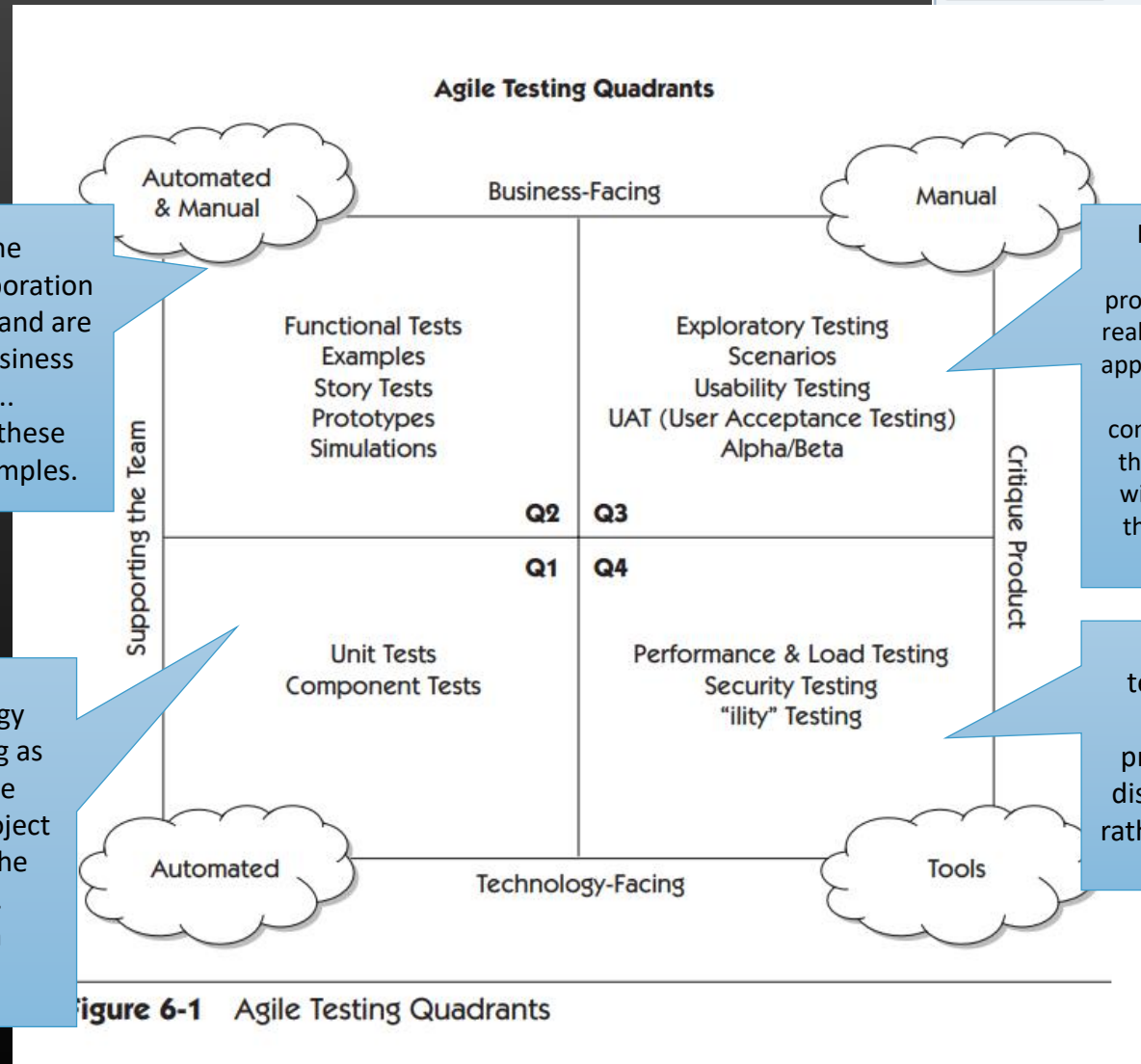
# Comparing old-school and incremental approaches



# Agile testing quadrants



**Agile Testing: A Practical Guide for Testers and Agile Teams**



Often conducted by the development team in collaboration with business stakeholders and are aimed at validating the business value of the software..  
With agile development, these tests are derived from examples.

business-facing tests to critique the product, emulating the way a real user would work with the application (tests by humans).  
Exploratory testing are conducted spontaneously by the testers as they interact with the software (explore the software's behavior in unexpected way)

Ensure that the technology stack and tools are working as expected. Focused on the technical aspects of the project and often conducted by the developers themselves.  
Q1 tests fits test-driven development (TDD).

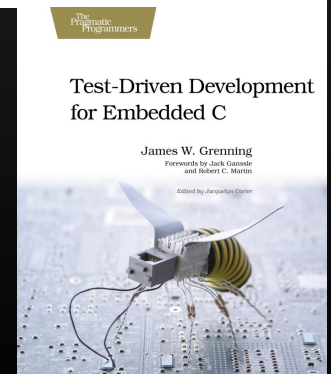
technology-facing tests meant to review the product "qualities"; we discuss them in technical rather than business terms

# Test-driven development (TDD)

# Debug Later Programming

We've all done it—written a bunch of code and then toiled to make it work. **Build it and then fix it.** Testing was something we did after the code was done. It was always an afterthought, but it was the only way we knew.

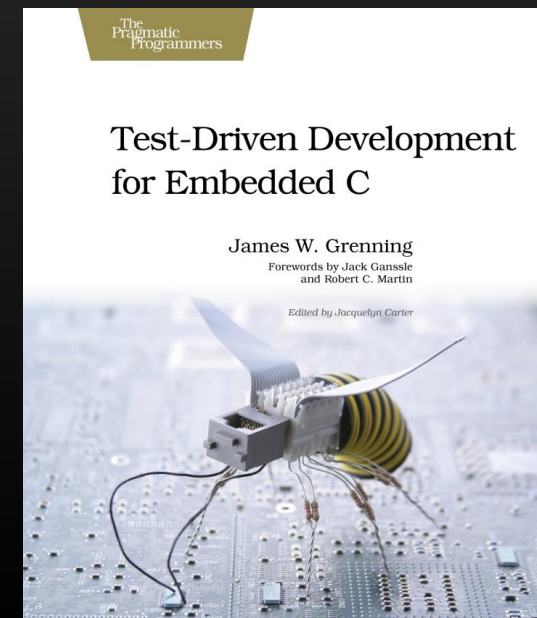
We would spend about half our time in the unpredictable activity affectionately called *debugging*. Debugging would show up in our schedules under the guise of test and integration. It was always a source of risk and uncertainty. Fixing one bug might lead to another and sometimes to a cascade of other bugs. We'd keep statistics to help predict



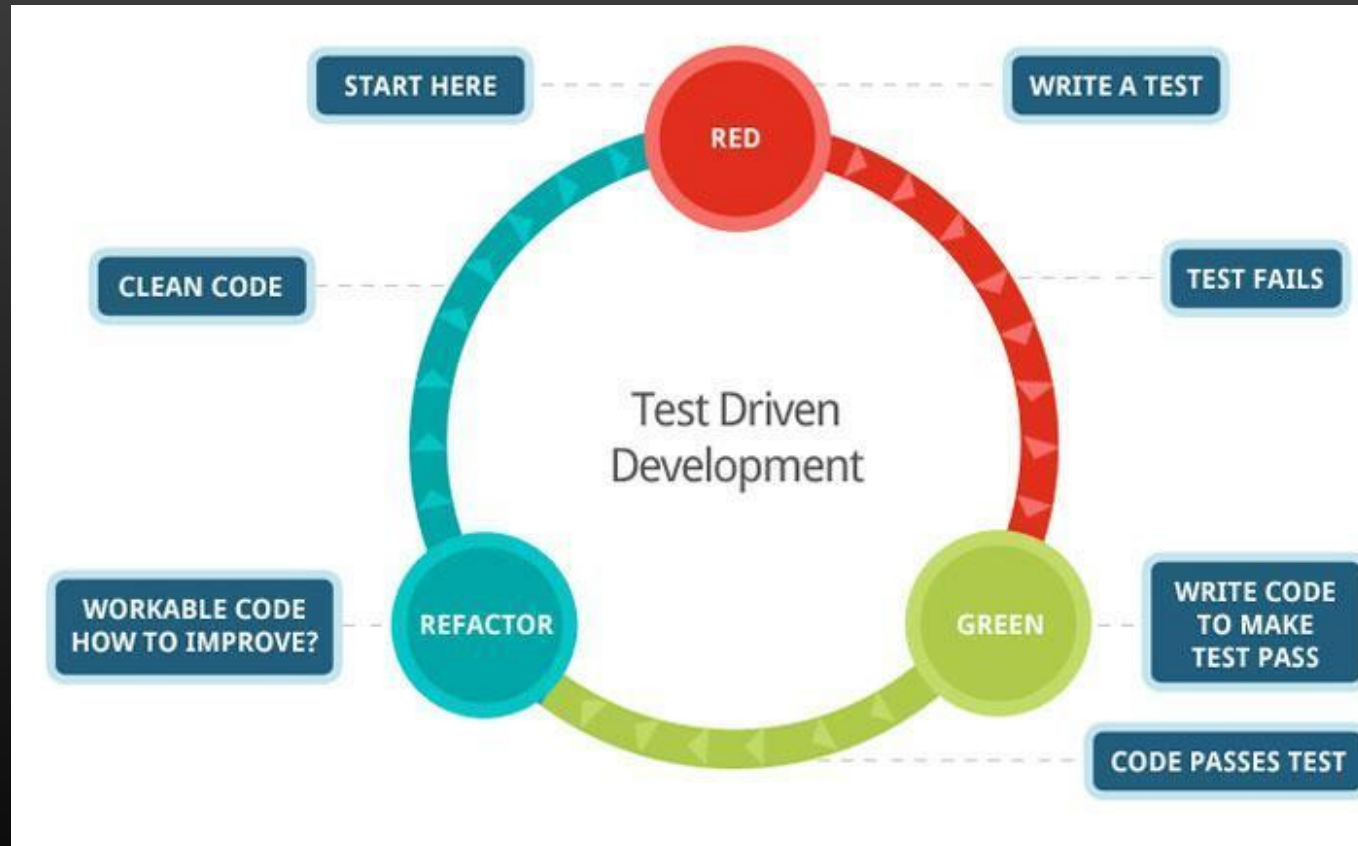


# Test-driven mindset

Test-Driven Development is a technique for building software incrementally. Simply put, no production code is written without first writing a failing unit test. Tests are small. Tests are automated. Test-driving is logical. Instead of diving into the production code, leaving testing for later, the TDD practitioner expresses the desired behavior of the code in a test. The test fails. Only then do they write the code, making the test pass.



# TDD: Test Driven Development



At the core of TDD is a repeating cycle of small steps known as the TDD microcycle. Each pass through the cycle provides feedback answering the question, does the new and old code behave as expected? The feedback feels good. Progress is concrete. Progress is measurable. Mistakes are obvious.

The steps of the TDD cycle in the following list are based on Kent Beck's description in his book *Test-Driven Development* [Bec02]:

1. Add a small test.
2. Run all the tests and see the new one fail, maybe not even compile.
3. Make the small changes needed to pass the test.
4. Run all the tests and see the new one pass.
5. Refactor to remove duplication and improve expressiveness.

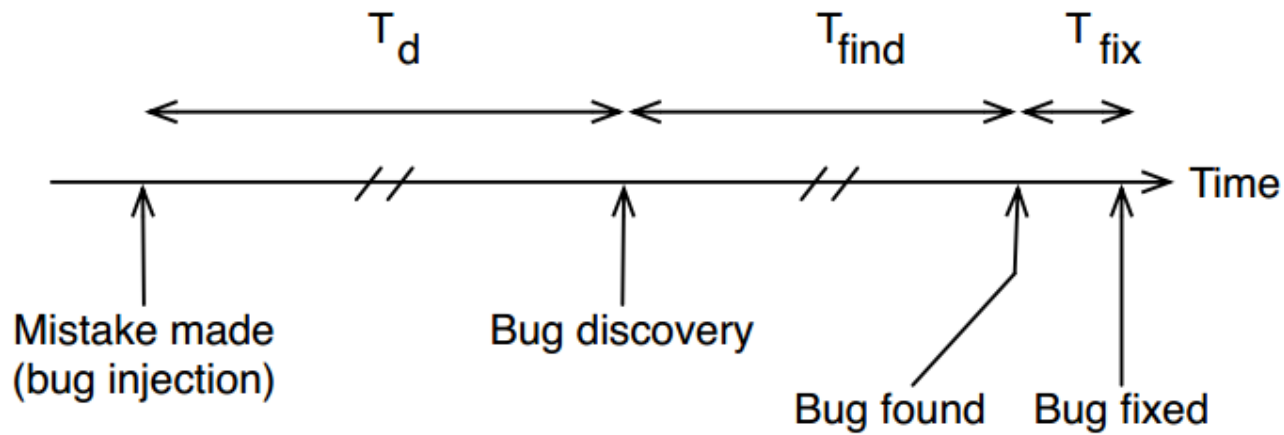


Figure 1.1: Physics of Debug-Later Programming

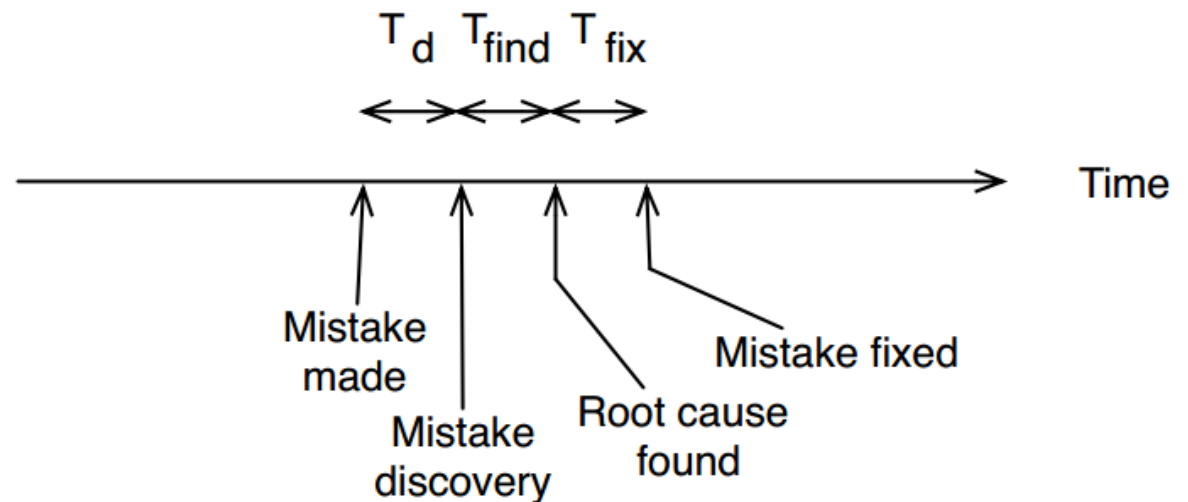


Figure 1.2: Physics of Test-Driven Development

# Acceptance test driven development (ATDD)

# Stories, use cases, scenarios



**FIGURE 8:**  
**THE RELATIONSHIP BETWEEN THE FLOWS AND THE STORIES**

# A story and tests...

Title (one line describing

Narrative:

As a [role]

I want [feature]

So that [benefit]

Acceptance Criteria: (presented as a list of

Scenario 1: Title

Given [context]

And [some more context].





When [event]

Then [outcome]


And [another outcome]...


Scenario 2: ...

Frank Can Add Another Person as a Friend

 ID #115218319   

Close

STORY TYPE  Feature ▼

POINTS  Unestimated ▼

STATE 

Start

 Unscheduled ▼

REQUESTER 

RJ

 Ryan Jones ▼

OWNERS <none> +

FOLLOW THIS STORY (1 follower) ☒

Updated: less than a minute ago

**DESCRIPTION** [\(edit\)](#)

As Frank I want to add a friend I searched for to my friend network so that I can see their posts, they can see my posts and I can direct message them

GIVEN I have searched for a friend's name  
WHEN I select "Add Friend" next to my friend's name  
THEN my friend's name should appear in my friend list on my homepage

Dev Notes: The added friend needs to be added to the Frank's friends in database

Design Notes: Attached are mocks for the button and placement

**LABELS**

add friend | x individual user | x ▼

→ Principles for user stories content

# Behavior Driven Development: Given, When, Then style

Structured syntax ([Gherkin](#)) to describe a feature (for testing):

**Feature:** what

**Scenario:** some determinable business situation

**Given:** preparation/setup (e.g.: required data)

- And...

**When:** the set of actions (execute).

- And...

**Then:** specifies the expected resulting state (assert).

- And...

```
Feature: Check Covid-19 stats

Scenario: Obtain world data
  Given I am in the Home page
  When I check to obtain world data
    And I choose the date 2021-01-01
    And I click 'Submit' under the covid section
  Then I should receive covid stats from the 'world'
    And the date 'at' field should be 2021-01-01
    And no other date field should appear

Scenario: Obtain country data
  Given I am in the Home page
  When I uncheck to obtain world data
    And I choose stats after 2021-12-12
    And I choose the country 'Portugal'
    And I click 'Submit' under the covid section
  Then I should receive covid stats from 'Portugal'
    And the date 'after' field should be 2021-12-12
    And no other date field should appear
```

Acceptance criteria should be executable



# Integrate Continuous testing in the development process

# Towards Continuous Testing

“We’re on the border between traditional testing and modern and continuous testing.

We’re moving...

- from the structure of the large, siloed Dev team with a centralized QA, and bottlenecks and delays in handoffs
- to a new structure: consists of small, autonomous and self-contained teams, which frequently ship software, use Continuous Integration tools to automate, and manage their own build environment to minimize bottlenecks.”

<https://www.testorigen.com/key-best-practices-for-continues-testing-in-agile/>

Practices for Continuous Testing:

Invest in continuous testing tools

Play well with continuous integration (Git, I, GitHub Actions, Sonarqube,...) Travis C

Opensource, documentation,...

Automate anything you can

The more tedious tasks are the more likely to be ready for automation

Collaborate

No silos; shared dashboards, shared ownership, Slack-like feedback on QA events

Define & publicize results

could include test coverage, number of pass-fail builds, average response time, etc.

“Dashboards in social areas” style of feedback

# Continuous testing

Continuous Testing is an effective testing technique carried out to automate the testing process, so it happens simultaneously with the development process

Scope is vast, with the validation of the functional and non-functional requirements of the product.

Key components:

**Continuous Integration (CI)**

**Continuous Delivery (CD)/test environments**

**Test Automation**



<https://www.professionalqa.com/continuous-testing>

# Quality concerns in the Definition of Done

## Definition of Done in Agile

The development team needs to decide what Done means (DoD)

DoD → checklist of assertions that must be true for every increment of software that is ready for deliver

## DoD “properties”:

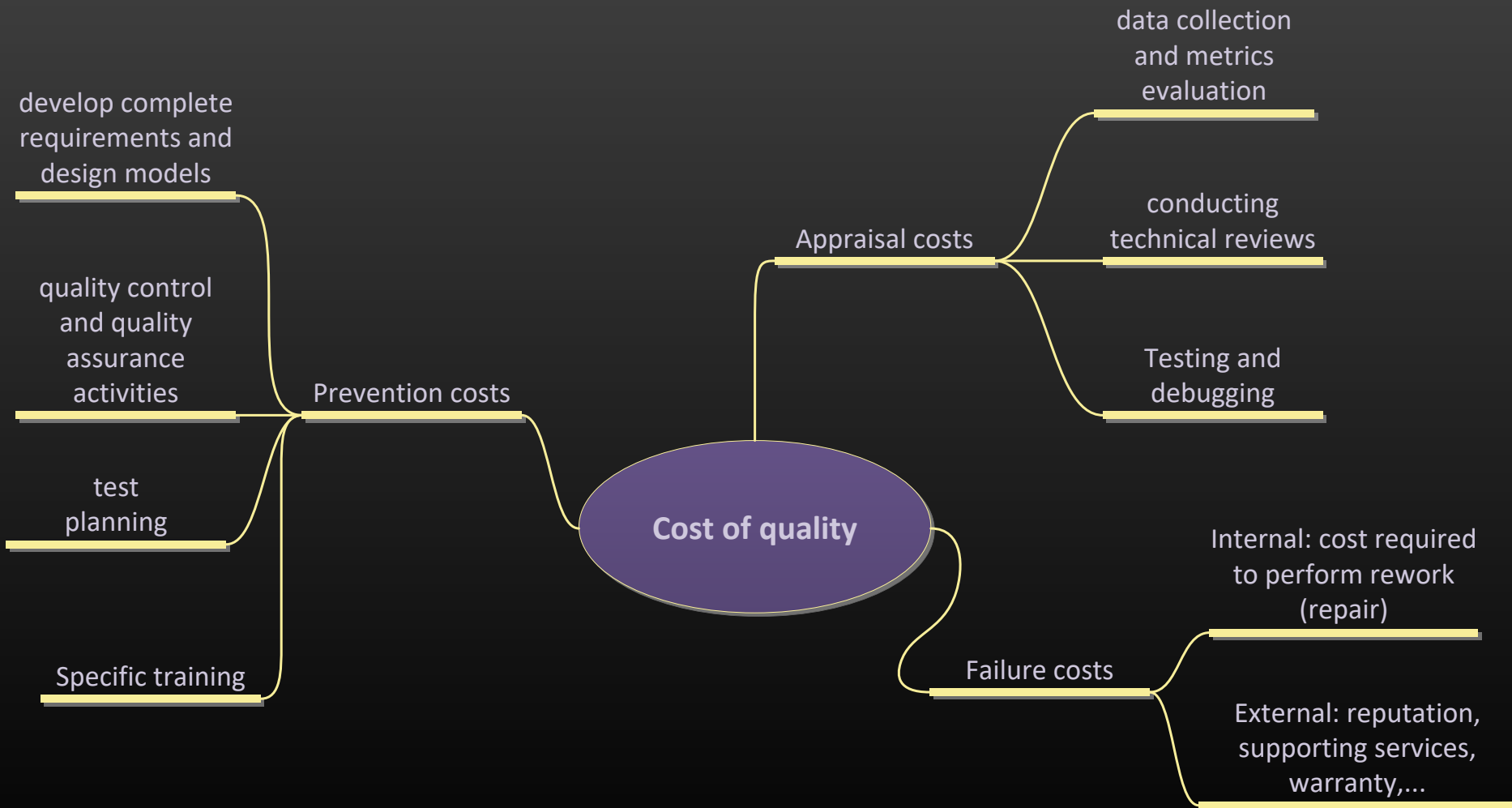
- A short, measurable checklist
- Assess shippable-readiness
- Yet, realistic...

## Sample topics:

- Story is estimated
- Acceptance criteria agreed
- Code reviewed
- Acceptance Tests for Increment exist and are Automated
- Code Coverage meets required level
- Security Checks Pass on Increment
- Increment meets agreed UX standards
- Increment has been demonstrated to PO

# The quality dilemma

# Costs vs investments in software quality



# Quality dilemma: when good is good enough?

If you produce a software system that **has terrible quality**, you lose because no one will want to buy it.

If on the other hand you spend infinite time, **extremely large effort**, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway. Either you missed the market window, or you simply exhausted all your resources.

So people in industry try to get to that magical middle ground where the product is good enough not to be rejected (...) but also not the object of so much perfectionism (...) that it would take too long or cost too much to complete.

*B. Meyers*



More  
resources,  
time to  
market?,...



Faster,  
reputation  
risks,...



<http://www.artima.com/intv/serious2.html>



"I worry that one day all of humanity will  
be maintaining code of our forefathers...  
think of the children!"  
*@stackexchange*