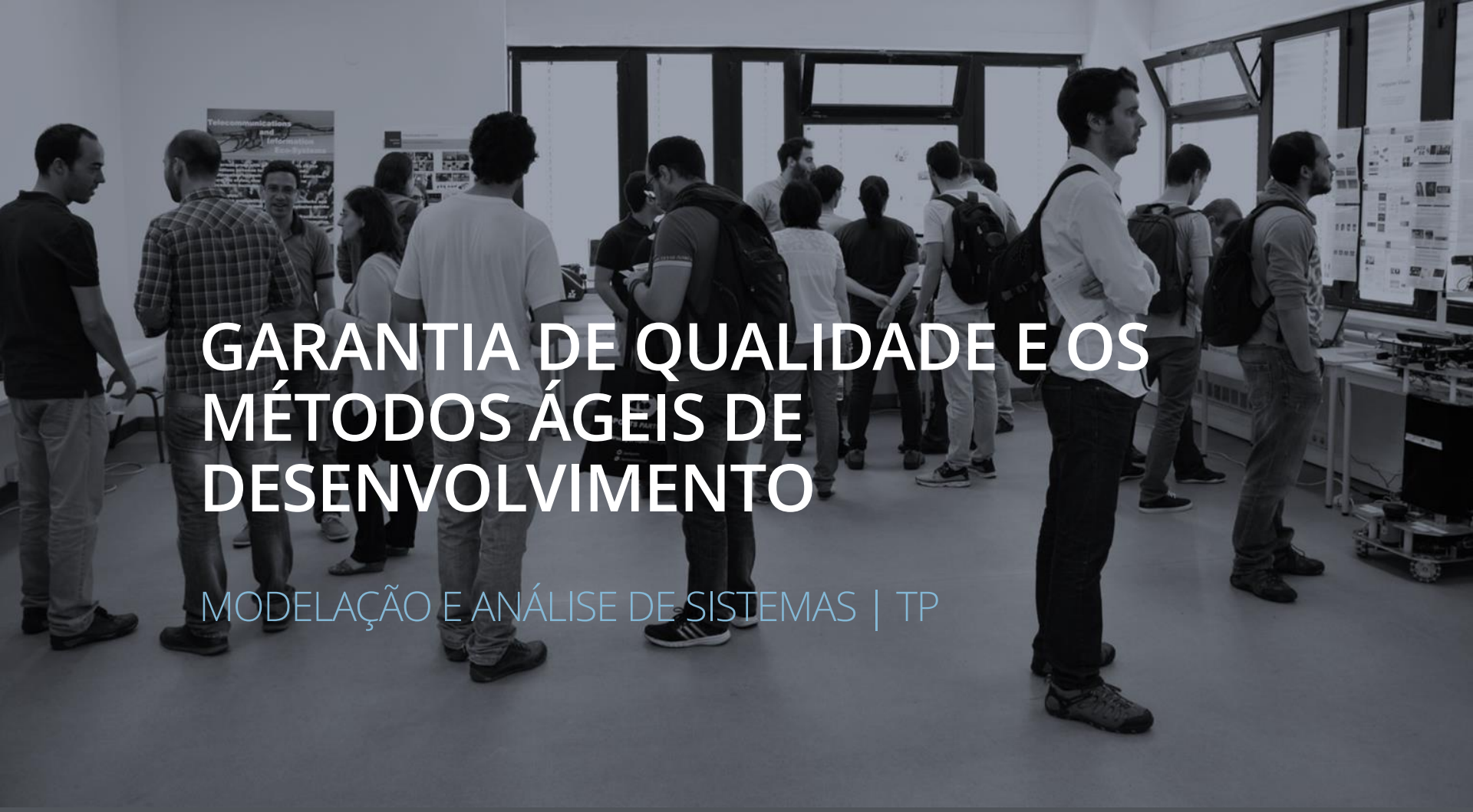


GARANTIA DE QUALIDADE E OS MÉTODOS ÁGEIS DE DESENVOLVIMENTO

MODELAÇÃO E ANÁLISE DE SISTEMAS | TP

ILÍDIO OLIVEIRA ico@ua.pt
v2018-05-24

universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

A PIRÂMIDE DOS TESTES

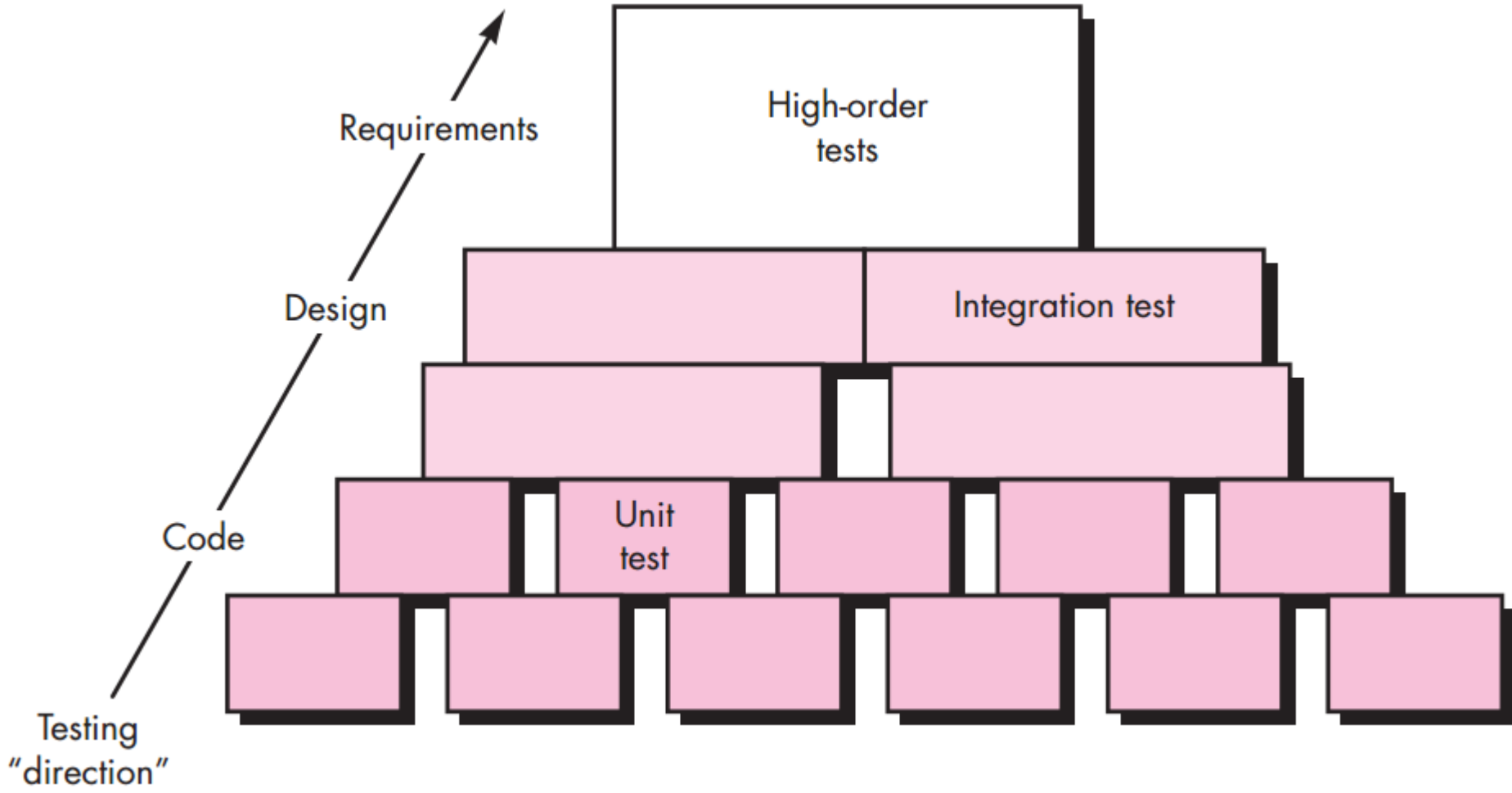
VERIFICATION: ARE WE DOING THE SYSTEM IN THE RIGHT WAY?

Check work products against their specifications
Check modules consistency
Check against industry best practices
...

VALIDATION: ARE WE DOING THE RIGHT SYSTEM?

Check work-products against the user needs and expectations

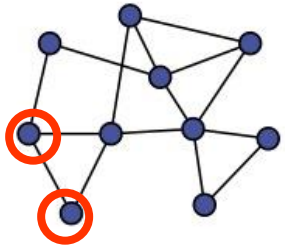
I Oliveira (2018)



Testing begins at component level and works outwards.

Different testing techniques are appropriate at different moments/software

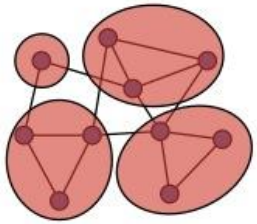
Unit testing

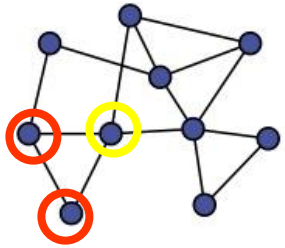


Each module does what it is supposed to do?

integration testing

managing complexity





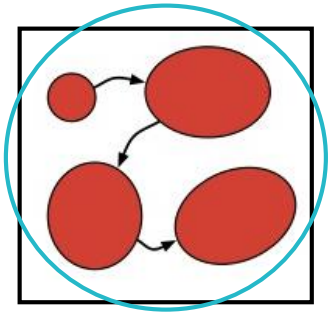
Do you get the expected results when the parts are put together?

Integration testing

Developer vs customer

Does the program satisfy the requirements?

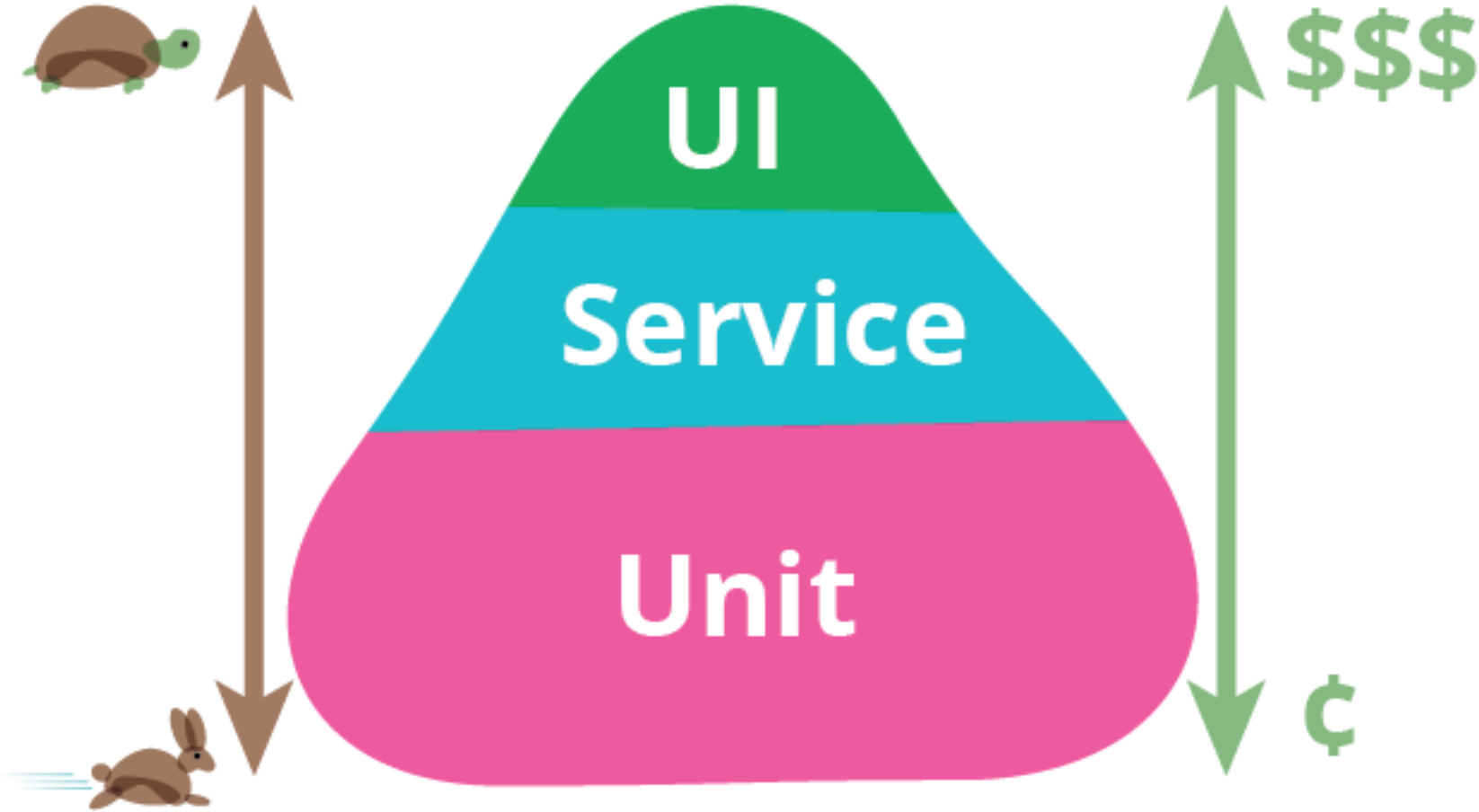
Acceptance / Functional Testing



System testing

The whole system functions as expected, in the target config?

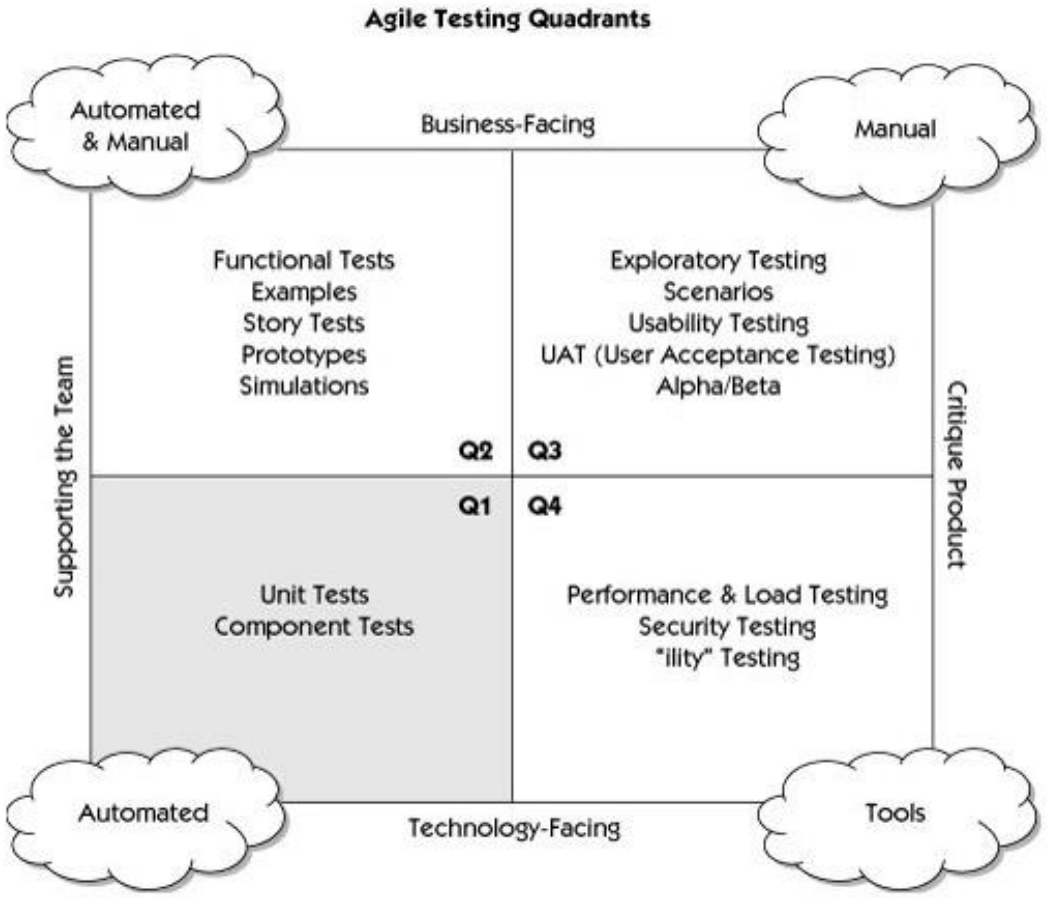
Test pyramid



https://martinfowler.com/bliki/TestPyramid.html

Agile testing quadrants

Agile Testing: A Practical Guide for Testers and Agile Teams



- **Functionality**
  - Correctness
  - Reliability
  - Usability

- **Engineering**
  - Efficiency
  - Testability
  - Documentation
  - Structure

Unit test example: Verifying the unit contract

A stack is empty on construction

A stack has size 0 on construction

After n pushes to an empty stack, n > 0, the stack is not empty && its size is n

If one pushes x then pops, the value popped is x, the size is decreased by one.

If one pushes x then peeks, the value returned is x, but the size stays the same

If the size is n, then after n pops, the stack is empty and has a size 0

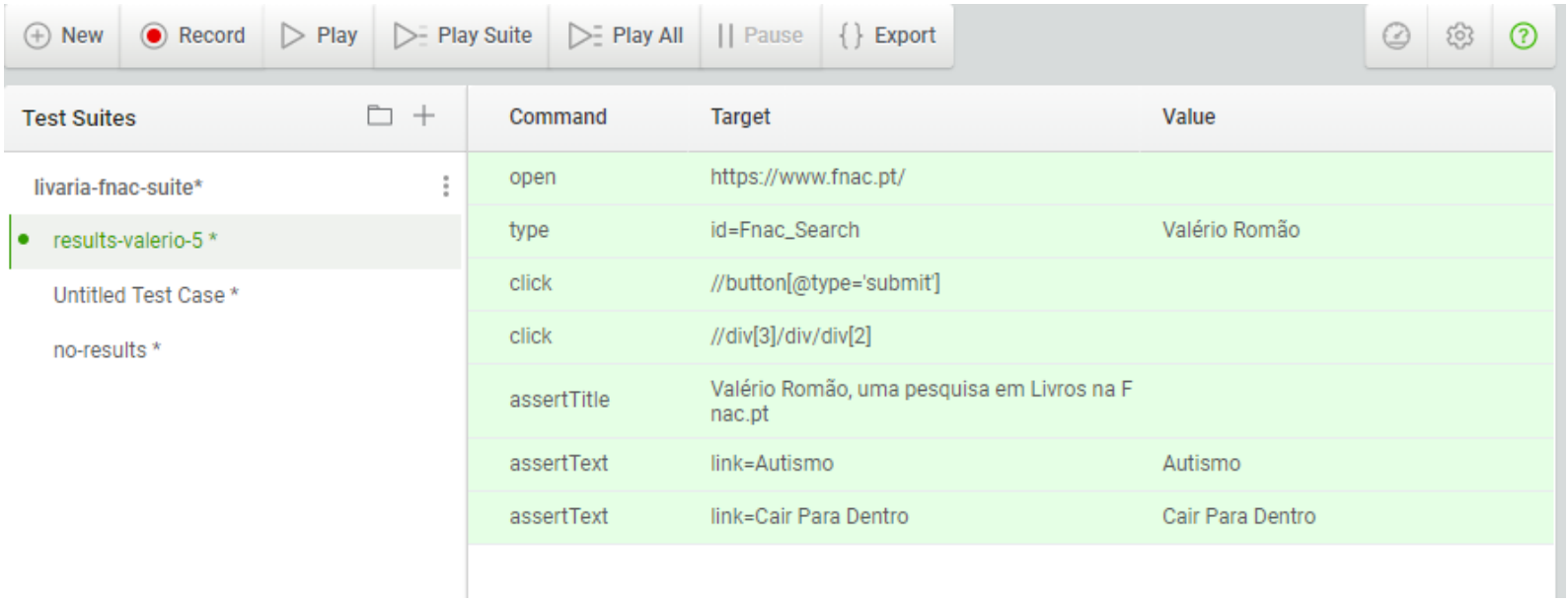
Popping from an empty stack does throw a NoSuchElementException

Peeking into an empty stack does throw a NoSuchElementException

For bounded stacks only, pushing onto a full stack does throw an IllegalStateException

→ See also: Ray Toal's notes.

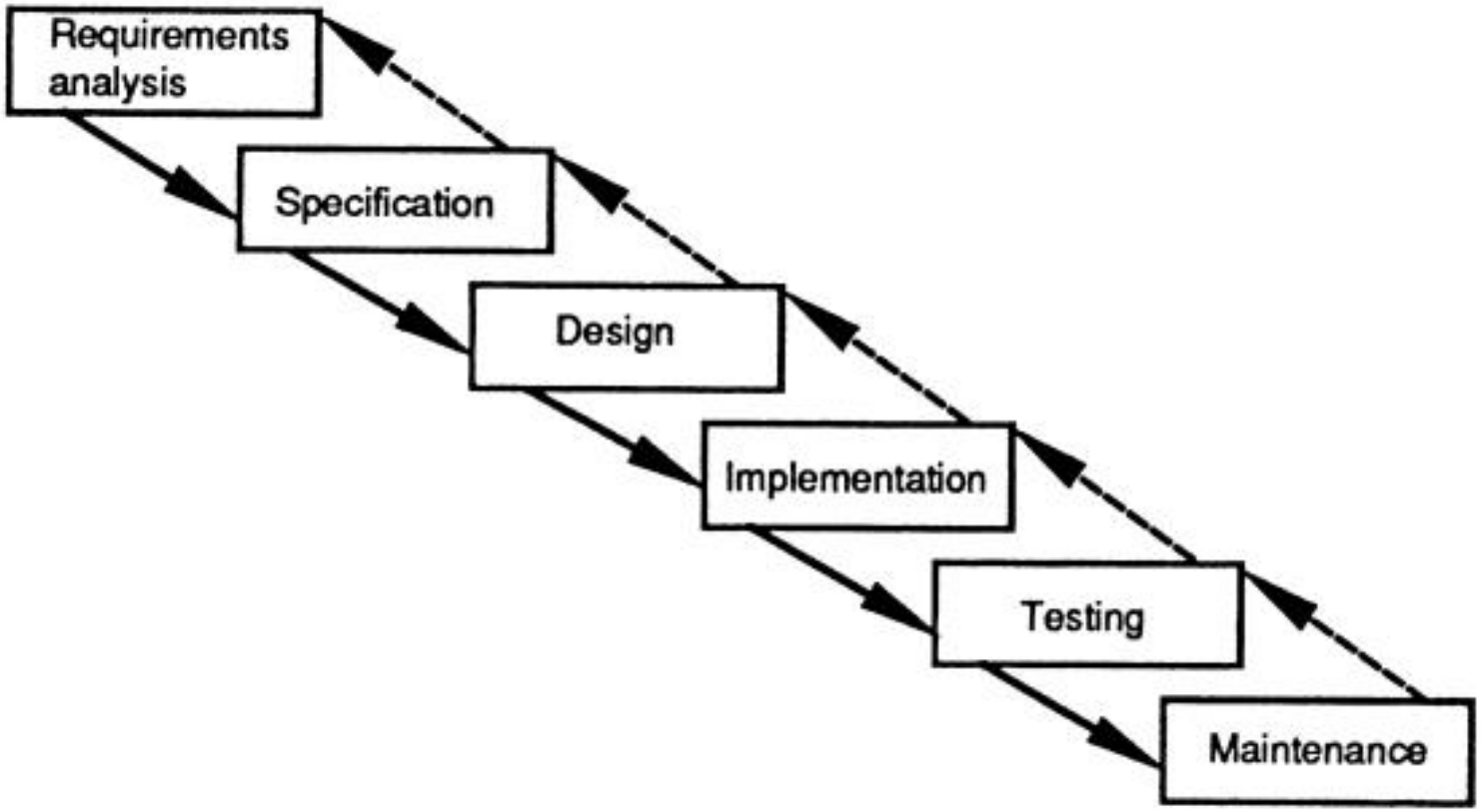
UI testing example: Web applications testing automation with Selenium



Katalon Recorder
Ultimate Selenium IDE to record, play, and debug app. Fast and extensible!

A GARANTIA DE QUALIDADE É PARTE INTEGRANTE DE UM PROCESSO DE ENGENHARIA

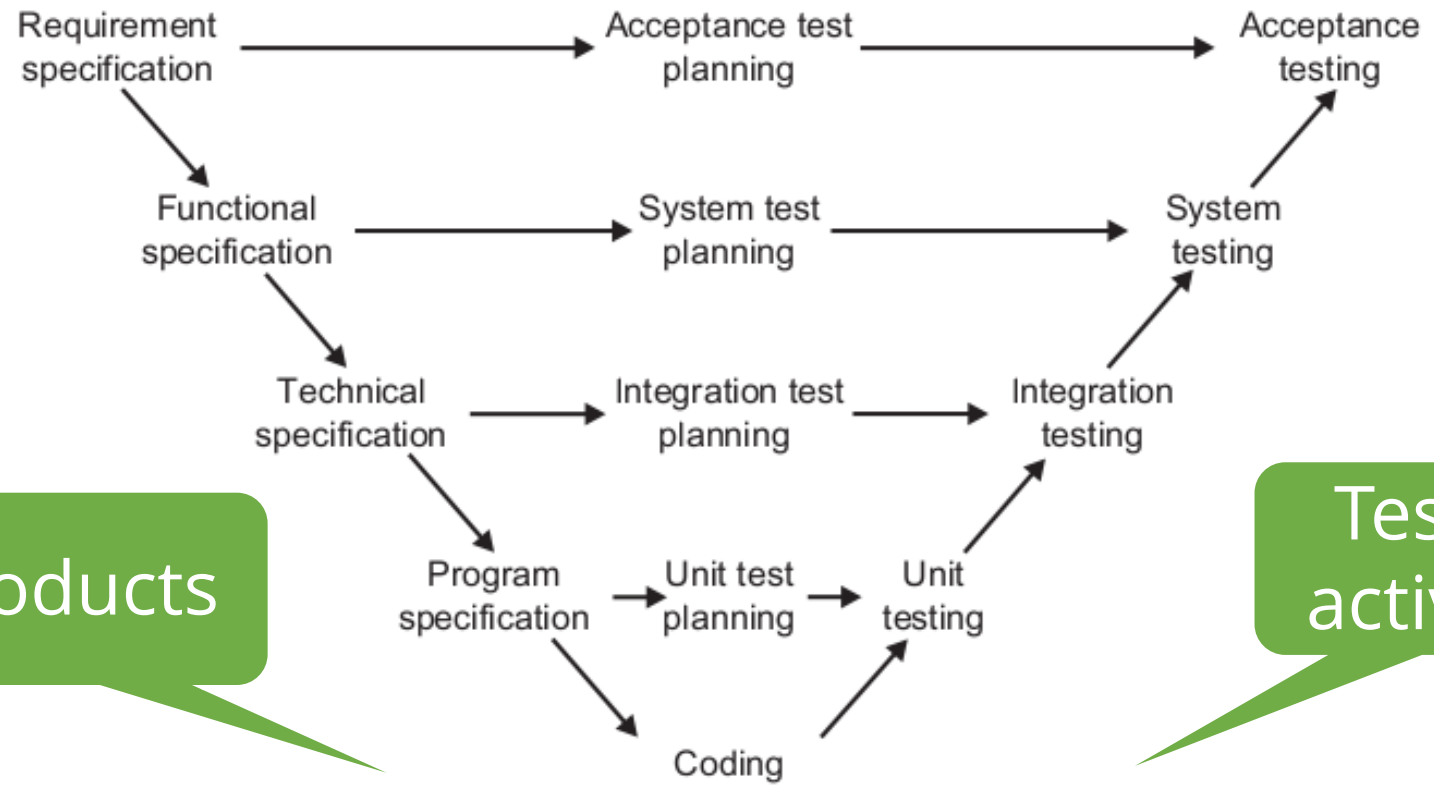
"Classical" engineering approach: **Waterfall model**



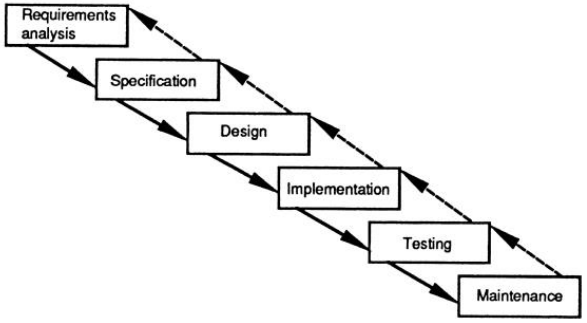
W. Royce, "Managing the Development of Large Software Systems," *Proc. Westcon*, IEEE CS Press, 1970, pp. 328-339.

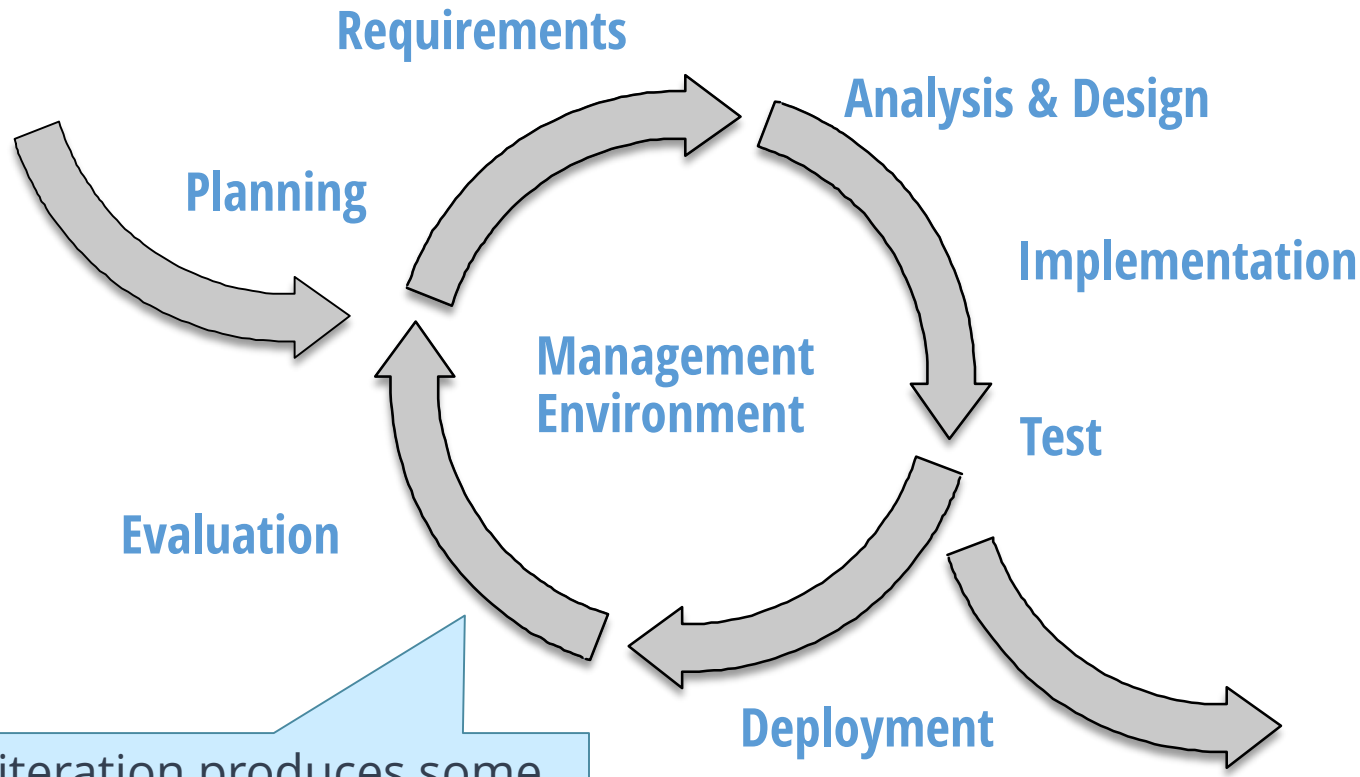
Test lifecycle and sw development lifecycle – the sequential approach

Figure 2.2 V-model for software development



O desenvolvimento iterativo foca a entrega de valor orientada por **ciclos curtos**





Each iteration produces some executable result

I Oliveira (2018)

Being systematic about software quality

SOFTWARE QUALITY ASSURANCE

set of activities (methodology) to control and monitor the software development process to attain the project goals with a certain level of confidence in quality terms.

SOFTWARE QUALITY CONTROL

evaluates if software products are within the defined quality standards resorting to inspections and different kinds of tests

SQA != SQC

SQC is aims at detecting and fixing defects. SQA aims at preventing them.

Practices of SQA

Testing

Software configuration management

Versions management

Code improvement

Reviews, shared practices, static analysis,...

Issue tracking and task management

Continuous integration

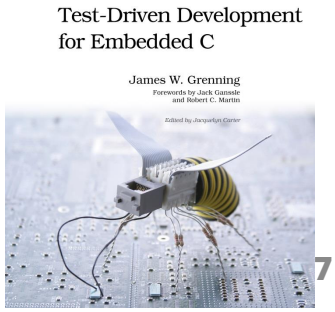
TEST-DRIVEN DEVELOPMENT

Debug Later Programming

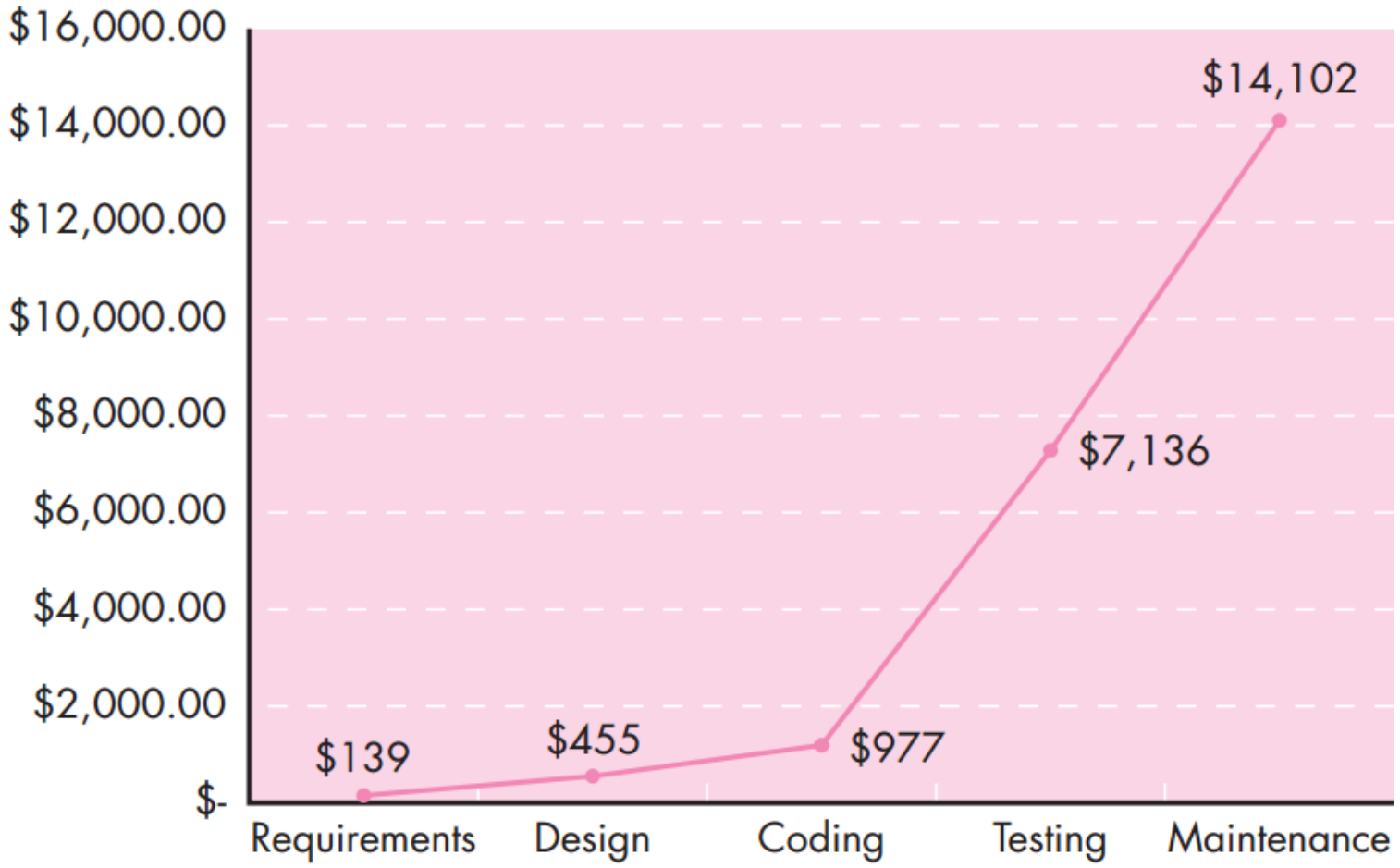
We've all done it—written a bunch of code and then toiled to make it work. Build it and then fix it. Testing was something we did after the code was done. It was always an afterthought, but it was the only way we knew.

We would spend about half our time in the unpredictable activity affectionately called *debugging*. Debugging would show up in our schedules under the disguise 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

The Pragmatic Programmers

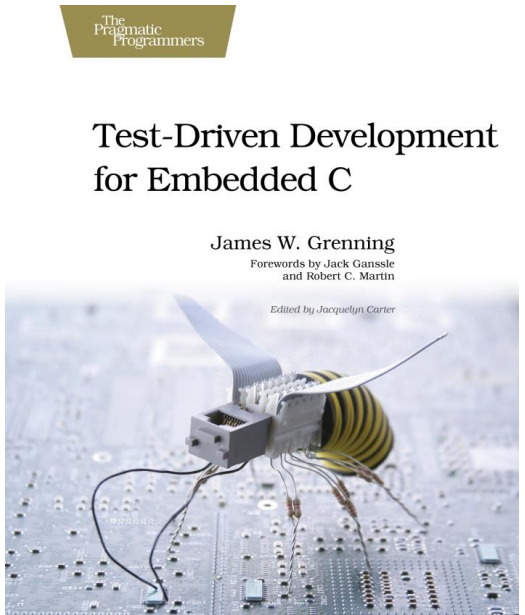


The cost of correcting an error raises exponentially along the sw lifecycle

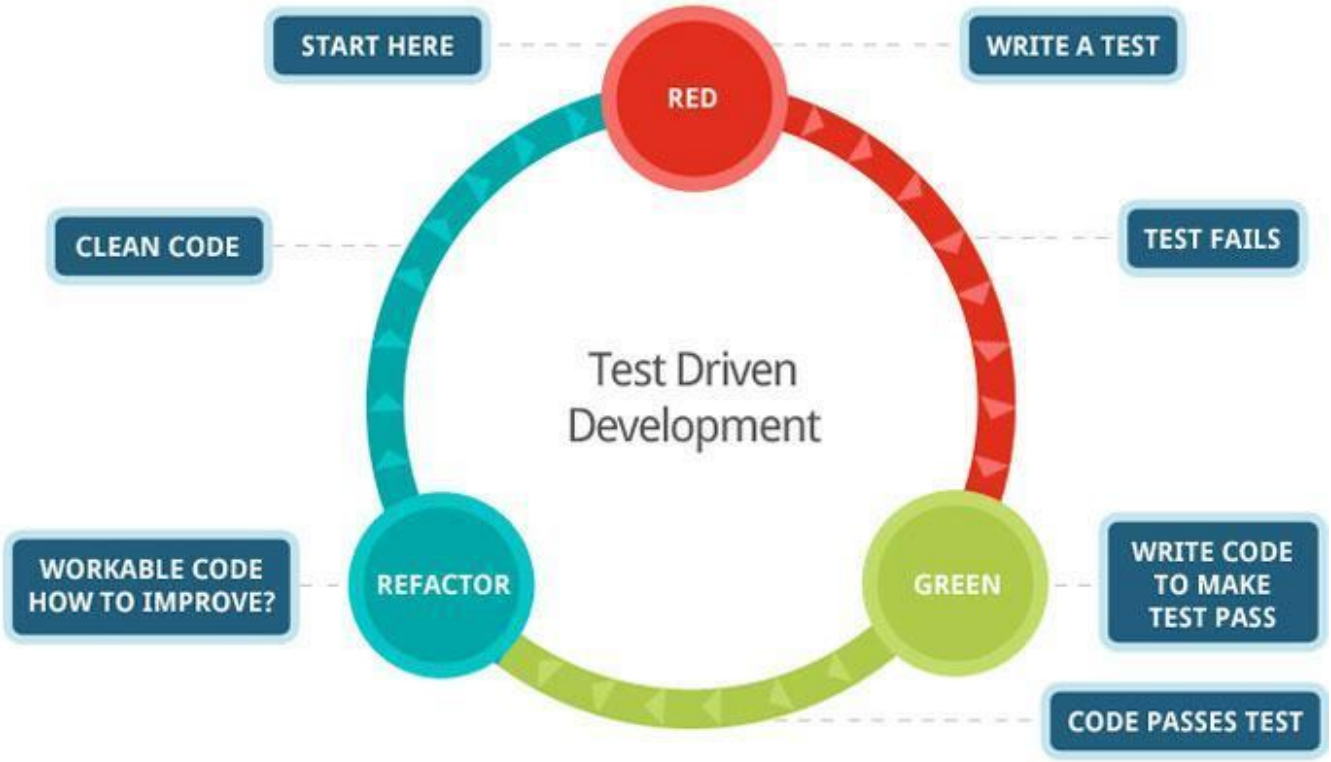


Boehm, B., and V. Basili, "Software Defect Reduction Top 10 List," IEEE Computer, vol. 34, no. 1, January 2001, pp. 135–137. http://doi.ieeecomputersociety.org/10.1109/2.962984

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.

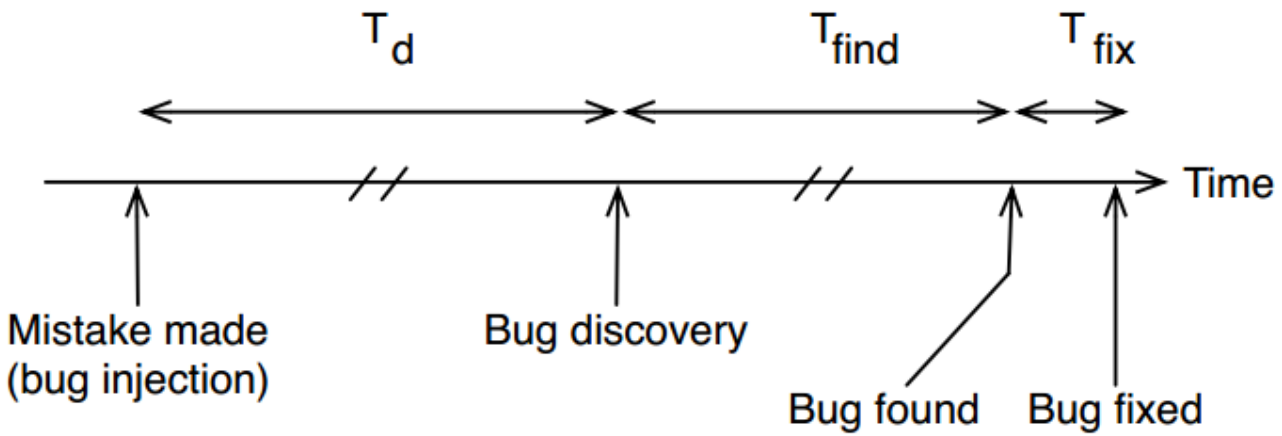


Figure 1.1: Physics of Debug-Later Programming

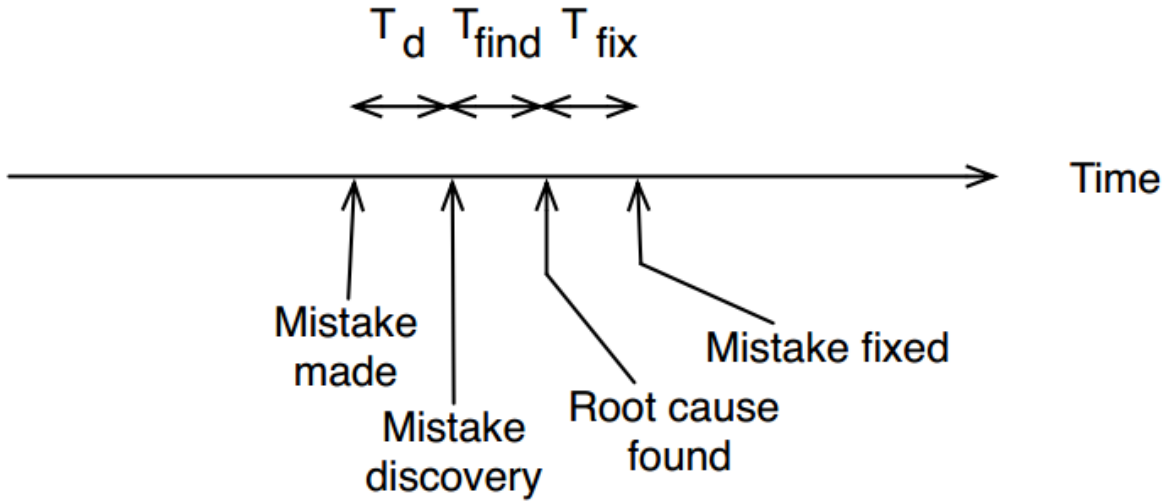


Figure 1.2: Physics of Test-Driven Development

I Oliveira (2018)

Story Testing

Executable Use Cases

I Oliveira (2018)

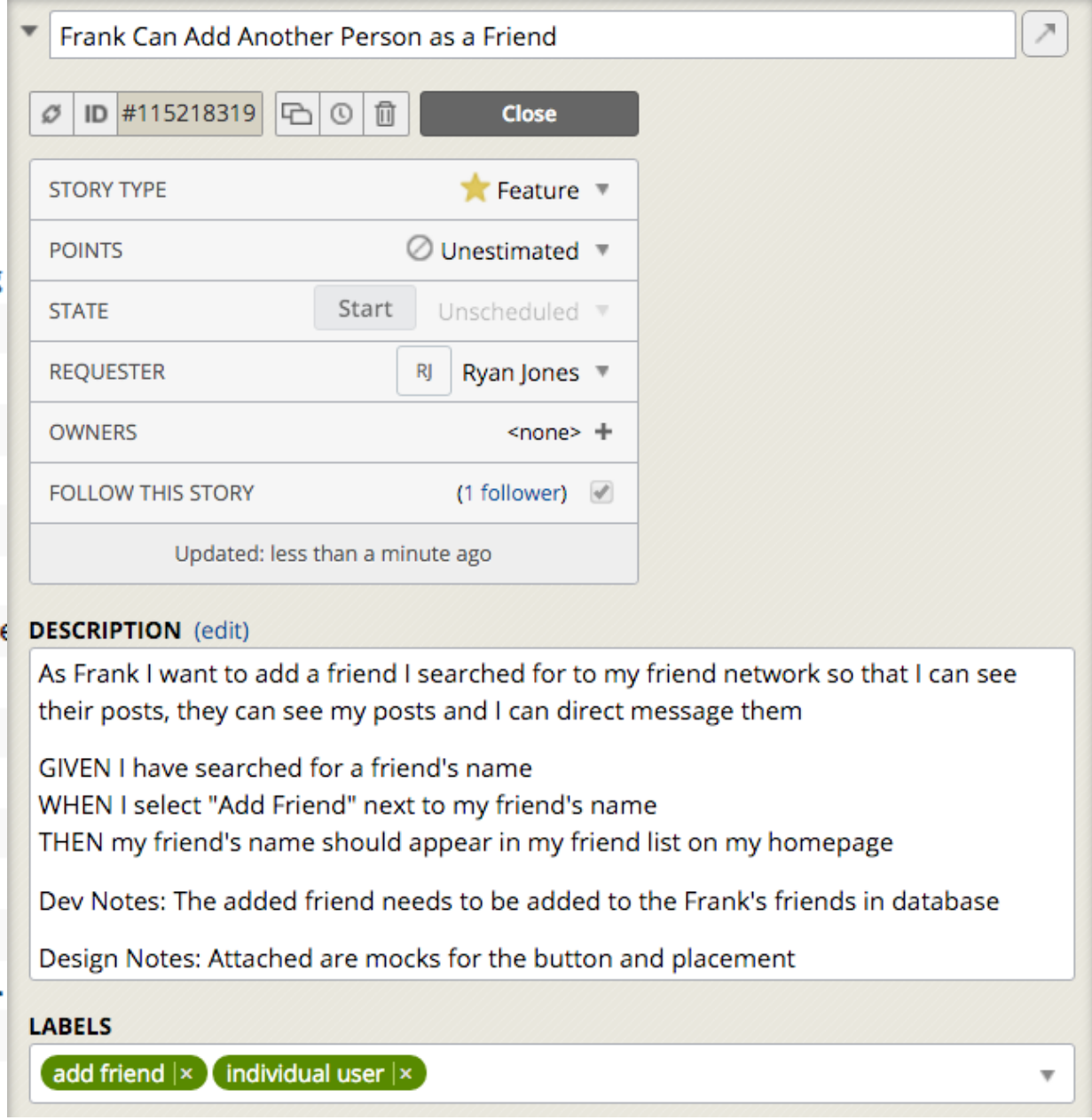
Stories, use cases, scenarios

FIGURE 8:
THE RELATIONSHIP BETWEEN THE FLOWS AND THE STORIES

I Oliveira (2018)

A story and tests...

```
Title (one line describing

Narrative:
As a [role]
I want [feature]
So that [benefit]

Acceptance Criteria: (prese

Scenario 1: Title
Given [context]
  And [some more context].
When  [event]
Then  [outcome]
  And [another outcome]...

Scenario 2: ...
```



→ Principles for user stories content

I Oliveira (2018)

Behaviour 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).

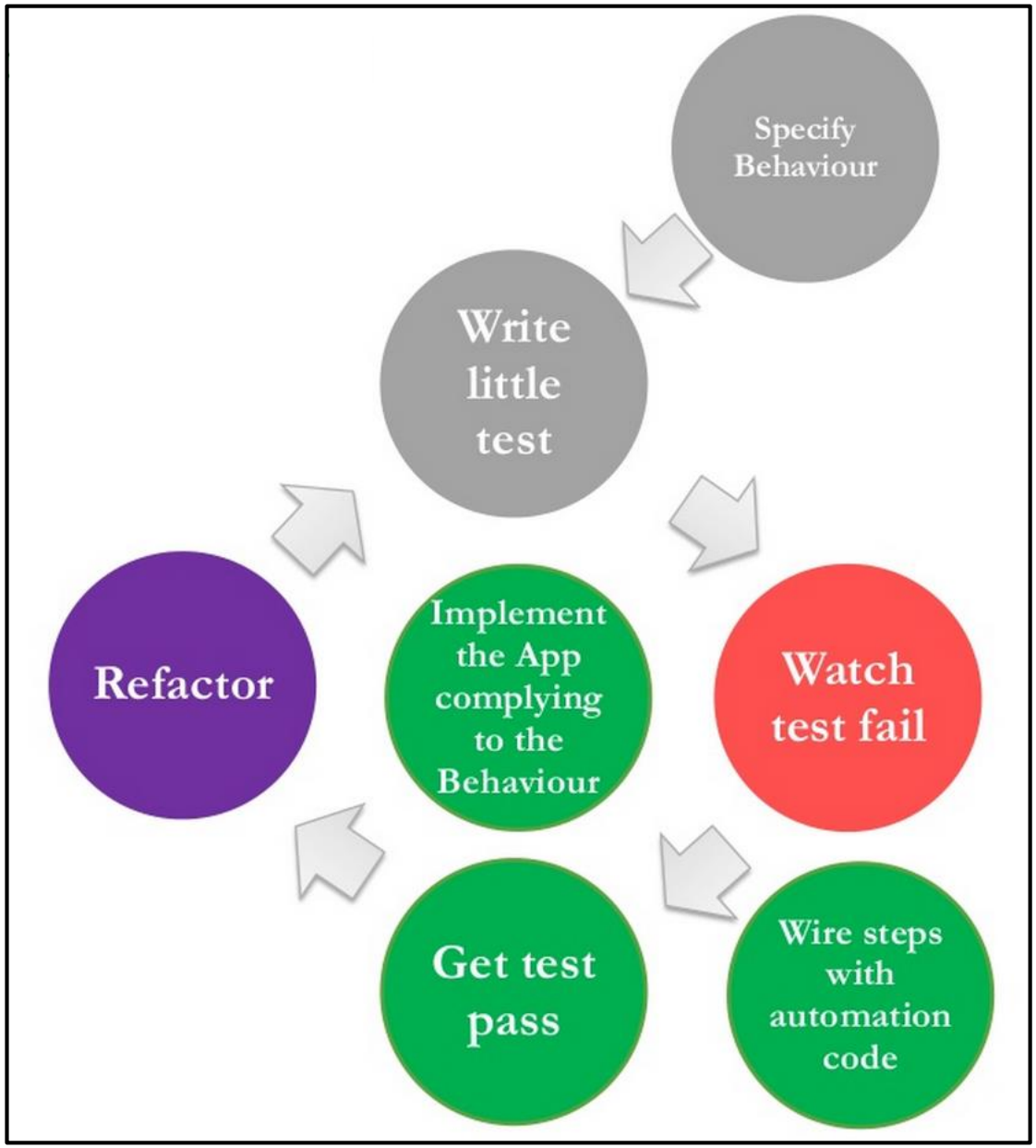
And...

```
Feature: Multiple site support

  Background:
    Given a global administrator named "Greg"
    And a blog named "Greg's anti-tax rants"
    And a customer named "Wilson"
    And a blog named "Expensive Therapy" owned by "W

  Scenario: Wilson posts to his own blog
    Given I am logged in as Wilson
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."

  Scenario: Greg posts to a client's blog
    Given I am logged in as Greg
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."
```

→ Example from behat documentation.

Acceptance criteria should be executable

BDD: Behaviour-driven development



Credit: Nalin Goonawardana