ANÁLISE DE SISTEMAS

Testes e *Quality Assurance* na Construção

Ilídio Oliveira

v2022/05/21

universidade de aveiro
departamento de eletrónica,
telecomunicações e informática
deti

Objetivos de aprendizagem

Identificar as actividades de validação e verificação no SDLC

Descrever as camadas da pirâmide de teste

Descrever o objecto dos testes de unidade, integração, sistema e aceitação

Explicar o ciclo de vida do TDD

Explicar como as actividades de QA são inseridas no processo de desenvolvimento numa abordagem clássica e em métodos ágeis

Relacionar os critérios de aceitação da história com testes ágeis

Explicar o conceito de especificações executáveis (e a relação com BDD).

Algumas ideias do desenvolvimento ágil

QUICK LOOK

What is it? Agile software engineering combines a philosophy and a set of development guidelines. The philosophy encourages customer satisfaction and early incremental delivery of software; small, highly motivated project teams; informal methods; minimal software engineering work products; and overall development simplicity. The development guidelines stress delivery over analysis and design (although these activities are not discouraged), and active and continuous communication between developers and customers.

Who does it? Software engineers and other project stakeholders (managers, customers, end users) work together on an agile team—a team that is self-organizing and in control of its own destiny. An agile team fosters communication and collaboration among all who serve on it.

Why is it important? The modern business environment that spawns computer-based systems and software products is fast-paced and ever-changing. Agile software engineering represents a reasonable alternative to conventional software engineering for certain classes of software and certain types of software projects. It has been demonstrated to deliver successful systems quickly.

What are the steps? Agile development might best be termed "software engineering lite." The basic framework activities—communication, planning, modeling, construction, and deployment—remain. But they morph into a minimal task set that pushes the project team toward construction and delivery (some would argue that this is done at the expense of problem analysis and solution design).

What is the work product? Both the customer and the software engineer have the same view—the only really important work product is an operational "software increment" that is delivered to the customer on the appropriate commitment date.

How do I ensure that I've done it right? If the agile team agrees that the process works, and the team produces deliverable software increments that satisfy the customer, you've done it right.

O dinamismo do mercado obriga a igual dinamismo do desenvolvimento. Especialmente quando os produtos de software assumem um papel fundamental na criação das vantagens competitivas.

A transformação digital (competitiva) obriga a uma eng.a de software competitiva.

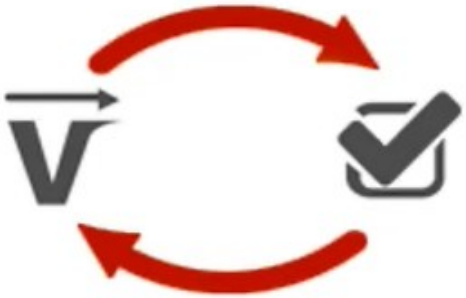
Credit: Pressmen'15
I Oliveira

Velocidade “furiosa”?

Greater speed may generate more risk and less quality...

... but **Velocity = Direction + Speed**

Para avançar depressa e com segurança, é preciso preparar a “máquina”: mexer no próprio processo de engenharia de sw.



quick feedback
improves direction
which improves quality which improves speed
which improves feedback

Crédito: Rui Gonçalves, Winning Consulting

É indispensável considerar as práticas que podem induzir ou medir a qualidade do produto

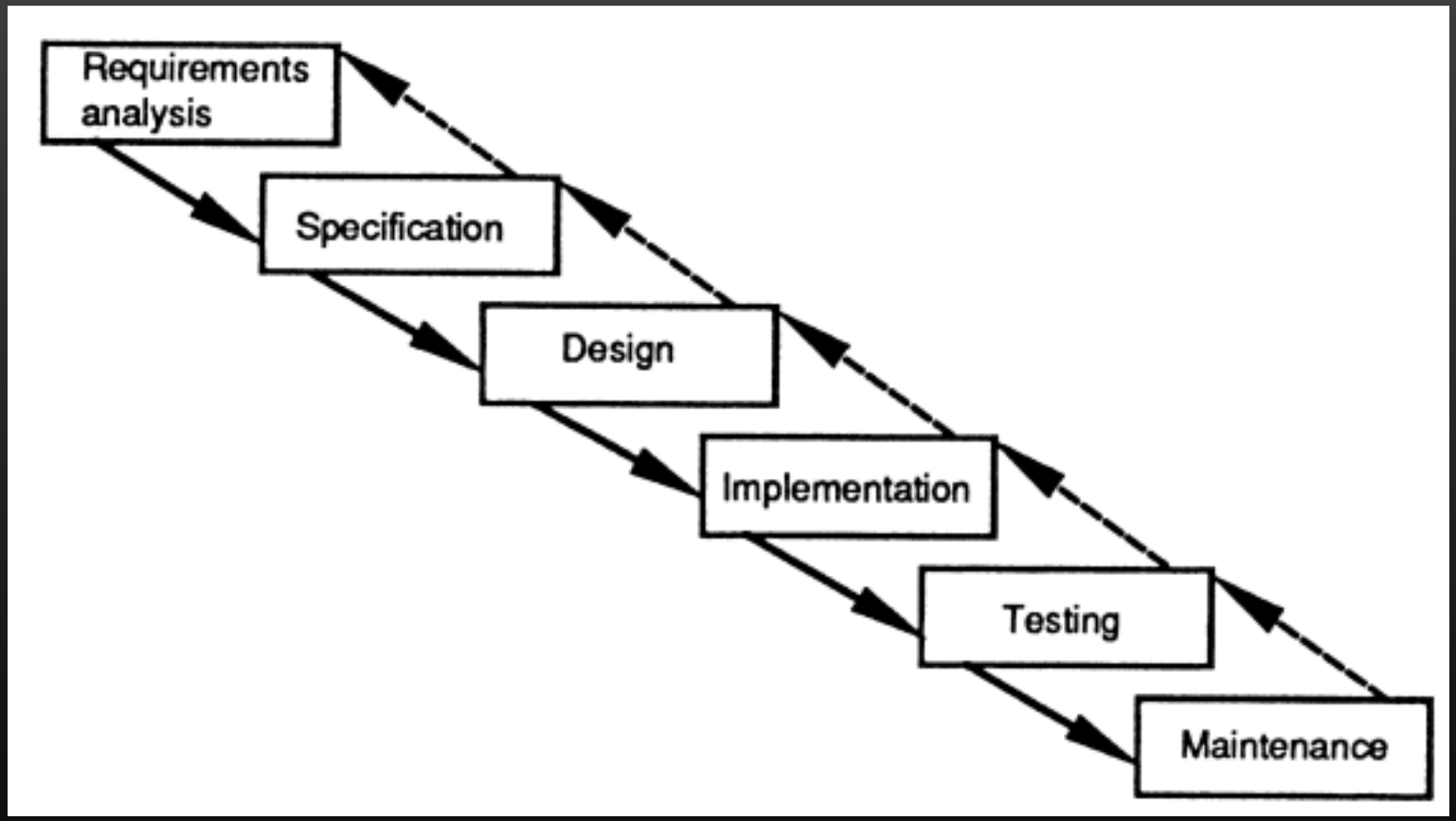
GARANTIA DE QUALIDADE DE SOFTWARE
conjunto de atividades (práticas) para controlar e monitorizar o processo de desenvolvimento de software para atingir os objetivos do projeto com um certo nível de confiança em termos de qualidade

CONTROLO DE QUALIDADE DE SOFTWARE
Avalia se os produtos de software estão dentro dos padrões de qualidade definidos recorrendo a inspeções e diferentes tipos de testes

SQA != SQC

A garantia de qualidade é parte integrante de um processo de engenharia

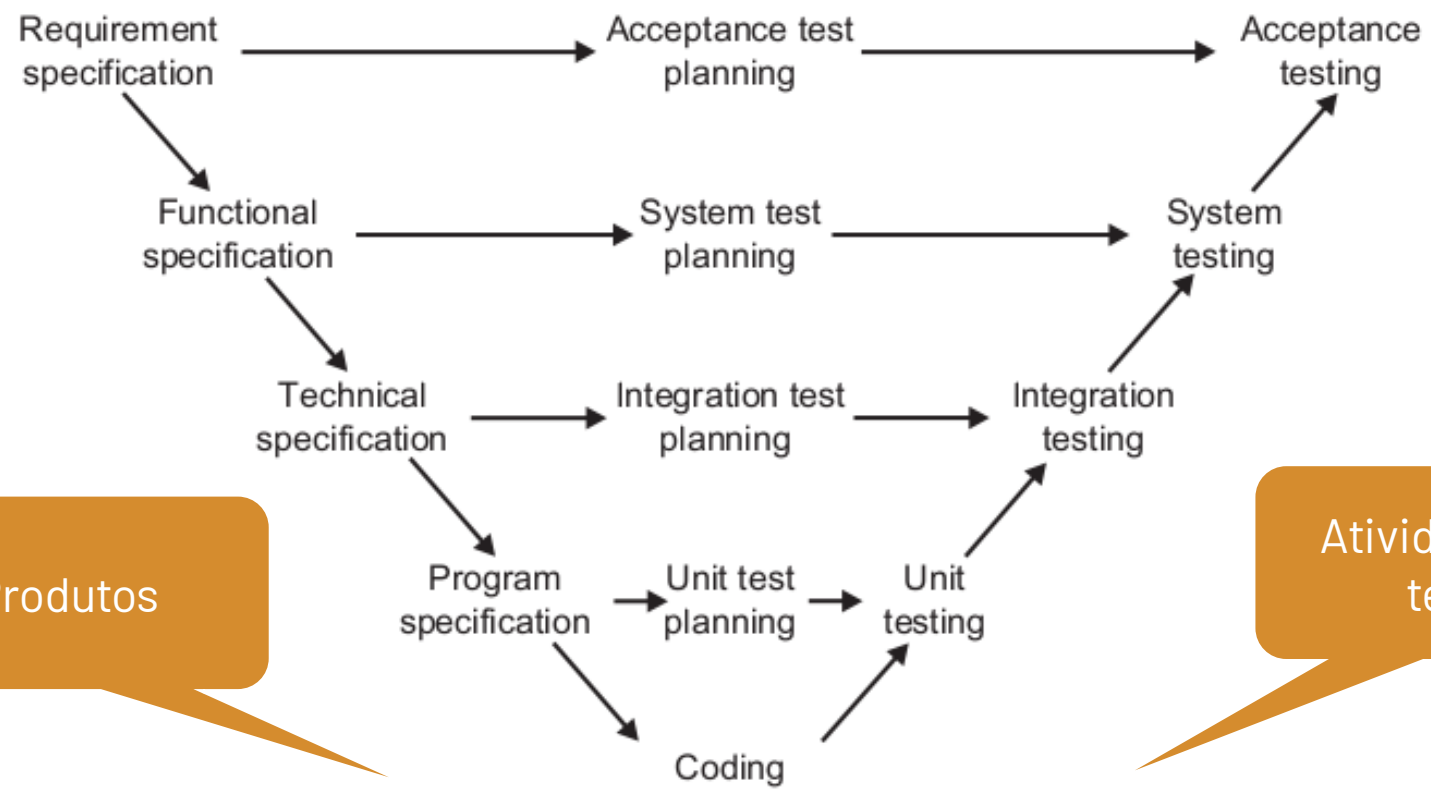
Abordagem de engenharia "clássica": Modelo waterfall



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

Ciclo de vida dos testes e o ciclo de vida do desenvolvimento do sw na abordagem sequencial

Figure 2.2 V-model for software development



Produtos

Atividades de teste

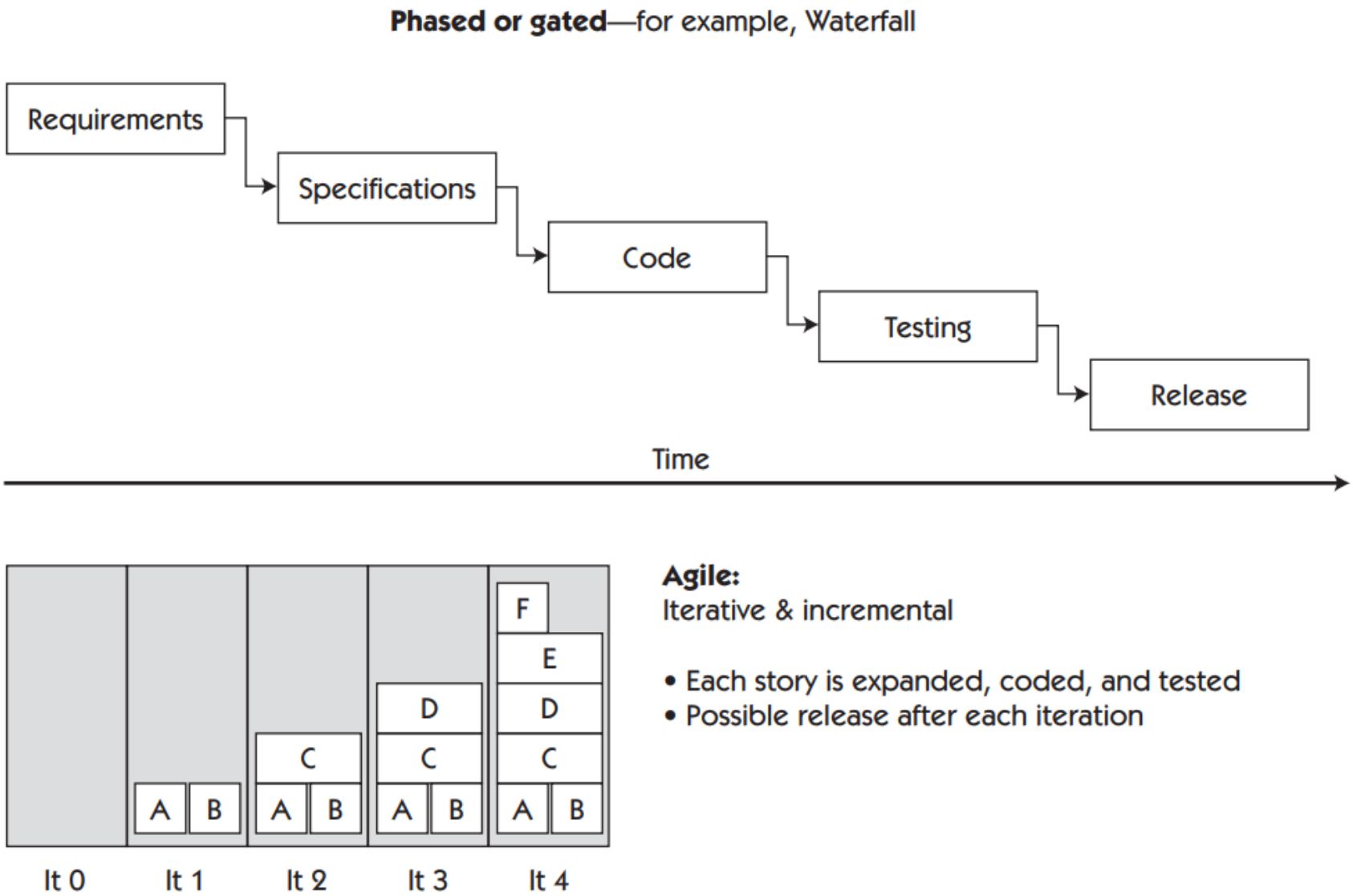
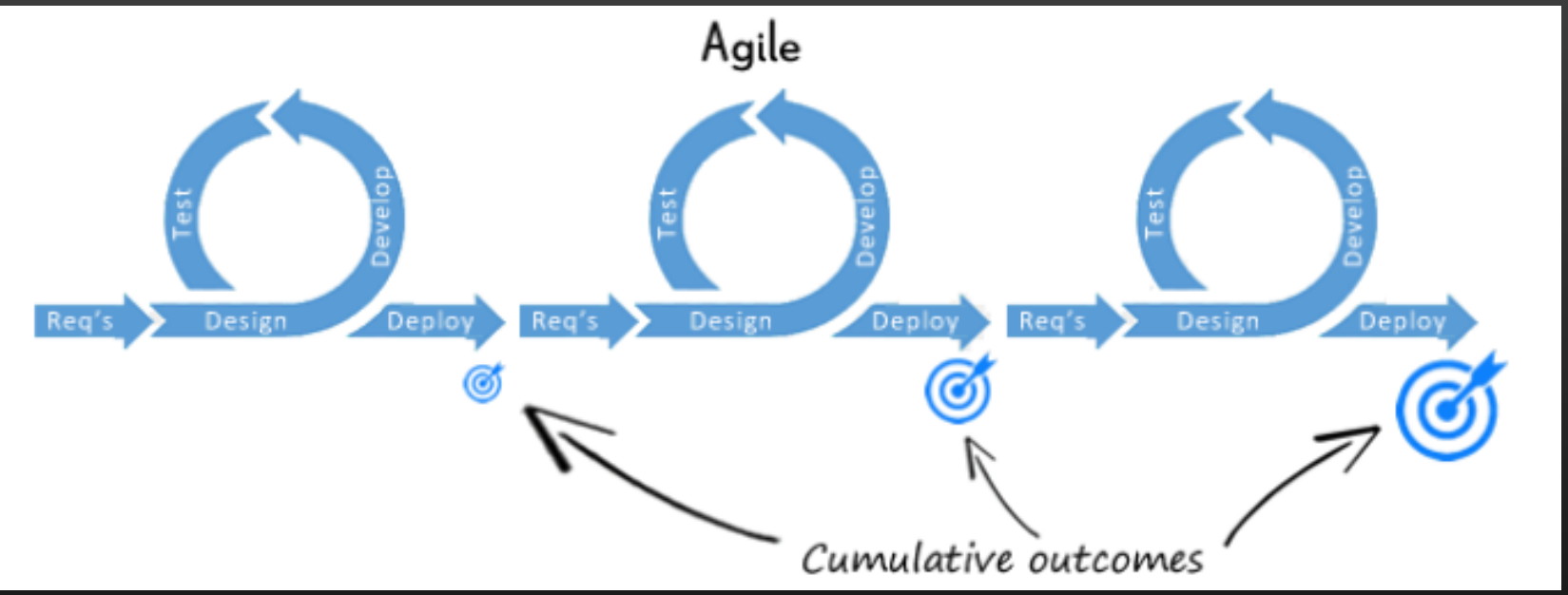
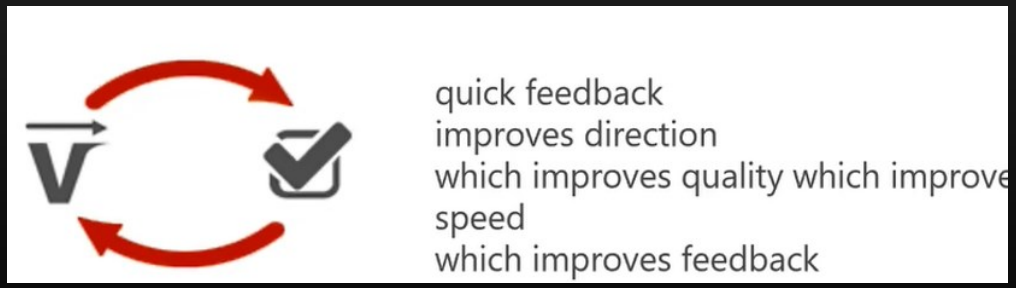


Figure 1-4 Traditional testing vs. agile testing

Na abordagem ágil, a garantia de qualidade tem de ser aplicada em cada ciclo





O papel dos testes de software

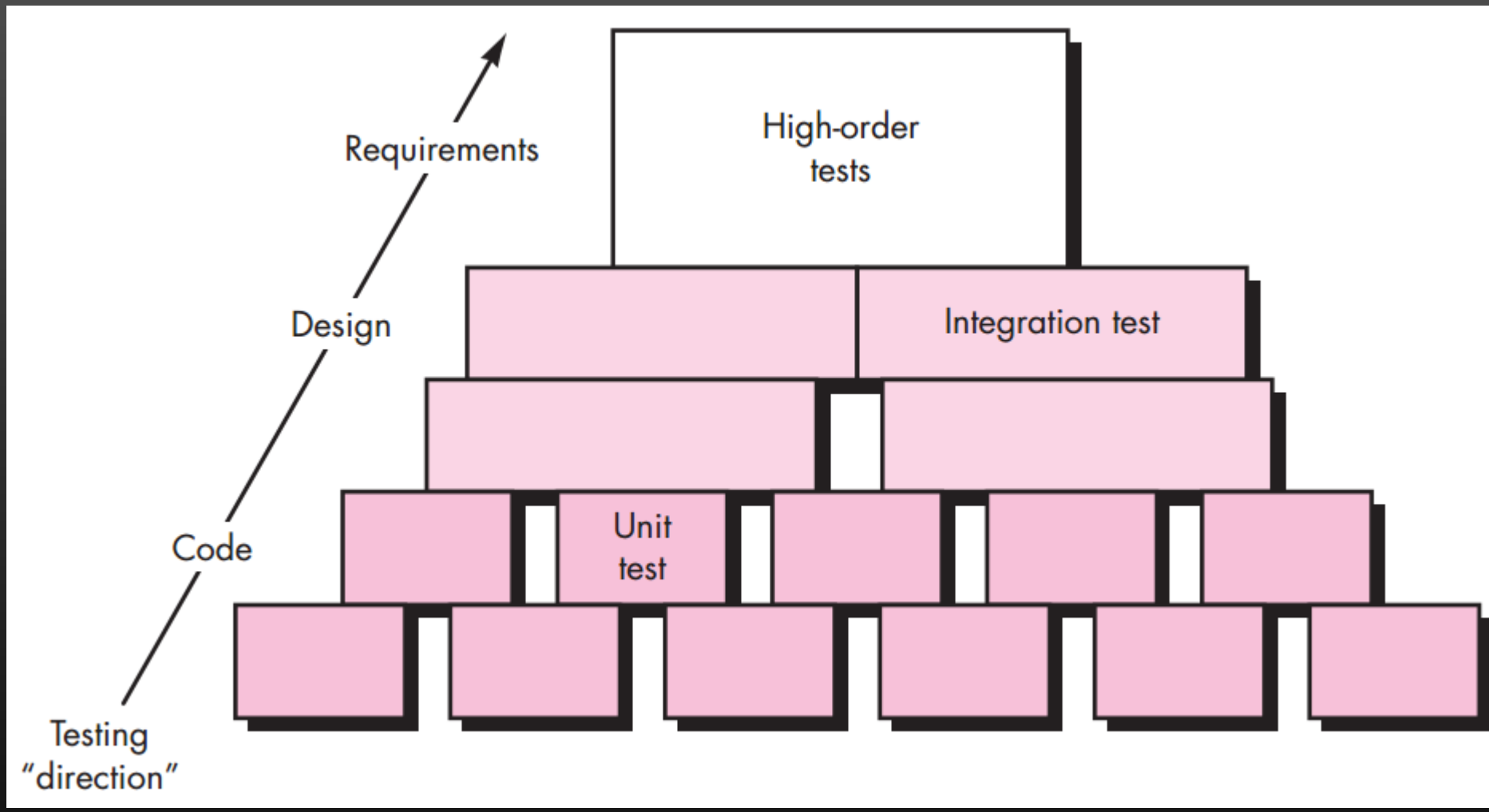
Verificação vs Validação

VERIFICAÇÃO: ESTAMOS A FAZER O SISTEMA DA FORMA CORRETA?

Verificar os produtos em relação às especificações
Verificar a consistência dos módulos
Comparar com as melhores práticas da indústria...

VALIDAÇÃO: ESTAMOS A FAZER O SISTEMA ADEQUADO?

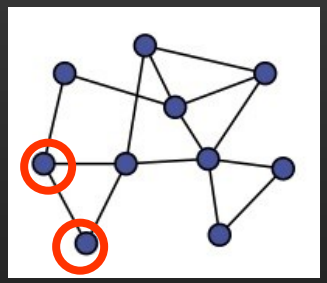
Avaliar os produtos em função das necessidades e expectativas dos utilizadores



Os testes começam ao nível dos componentes e progridem para "fora"

Different testing techniques are appropriate at different moments/software

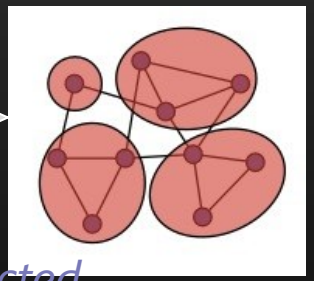
Unit testing

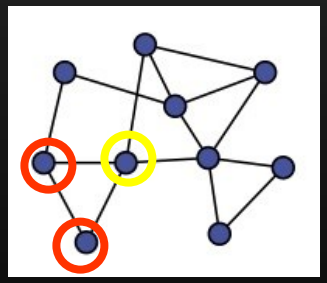


Each module does what it is supposed to do?

managing complexity

integration testing





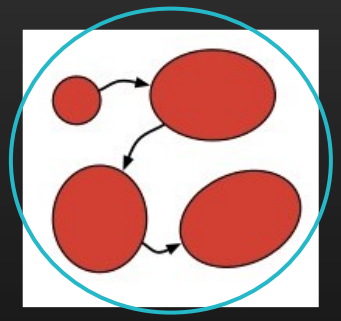
Do you get the expected results when the parts are put together?

Integration testing

Developer vs customer

Does the program satisfy the requirements?

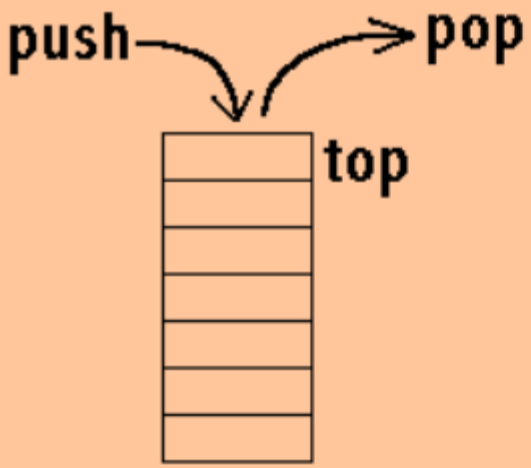
Acceptance / Functional Testing



System testing

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

Teste unitário: o contrato de uma *stack*



Operations

- `push(x)`: add an item on the top
- `pop`: remove the item at the top
- `peek`: return the item at the top (without removing it)
- `size`: return the number of items in the stack
- `isEmpty`: return whether the stack has no items

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.

Exemplo de teste UI: Automatização de testes de aplicações Web com Selenium IDE

Isaac Asimov - Bertrand Livreiro

https://www.bertrand.pt/autor/isaac-asimov/7861

BERTRAND LIVREIROS

Pesquisar

LIVROS

EM PORTUGUÊS

EM INGLÊS

Isaac

Extension: (Selenium IDE) - Selenium IDE - as1 — Mozilla Firefox

Project: as1

Executing

✓ comprar-livro

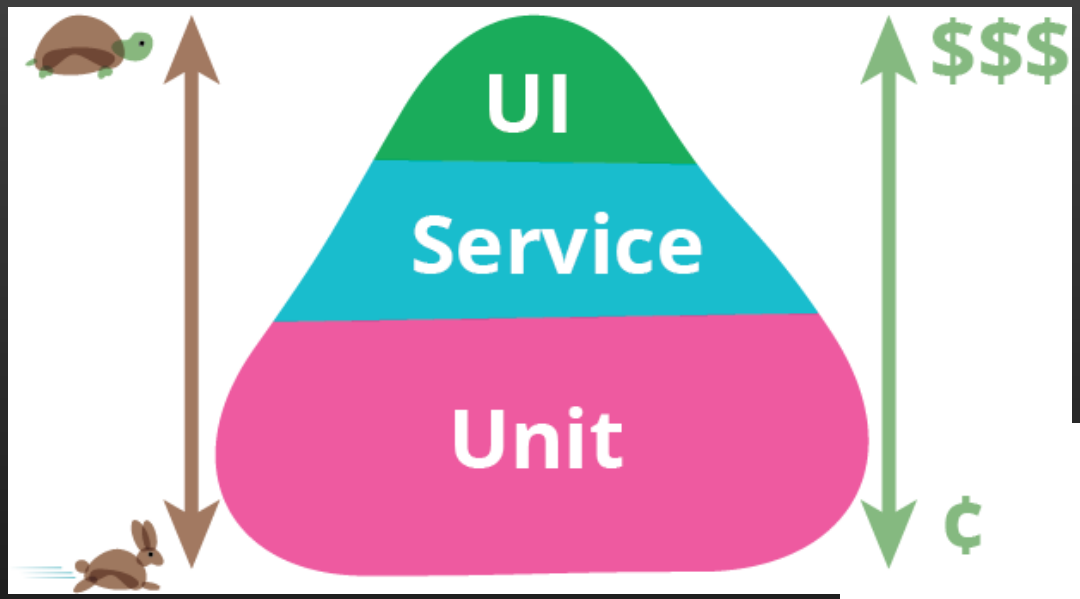
https://blazedemo.com

	Command	Target	Value
1	✓ *open*	https://www.bertrand.pt/	
2	✓ *click*	id=form-searchform-palavra	
3	✓ *type*	id=form-searchform-palavra	isaac asimov
4	✓ *click*	css=.item-auto-complete-img	
5	✓ *click*	css=.product-title	
6	✓ *assert title*	Isaac Asimov - Bertrand Livreiros - livraria Online	
7	✓ *assert text*	css=.product-title	Eu, Robô

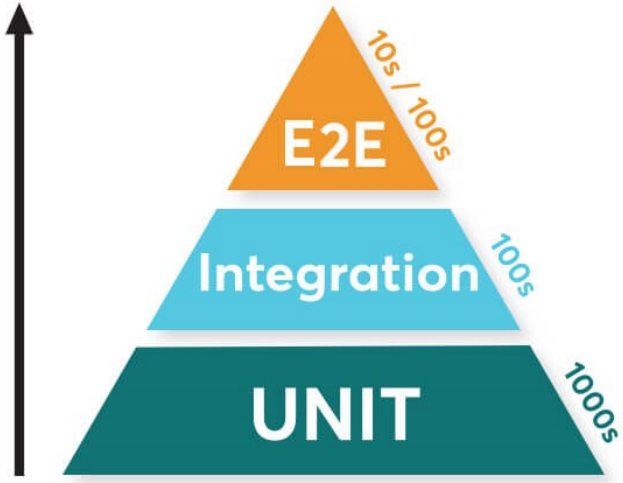
0 comentários)

1920 e foi para os Estados Unidos c
Bioquímica, é um dos autores ma
da ficção científica e da divulgação cie
er a série *Fundação* com 21 anos e a
s de 470 livros sobre temas variados

Pirâmide dos testes



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



https://www.blazemeter.com/blog/agile-development-and-testing-an-introduction

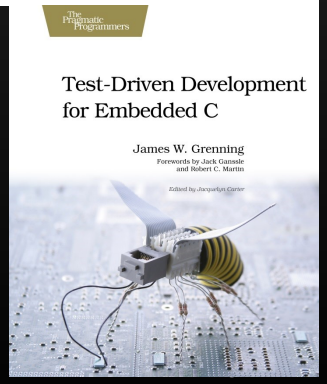
Test-driven development

Desenvolvimento conduzido pelos testes.

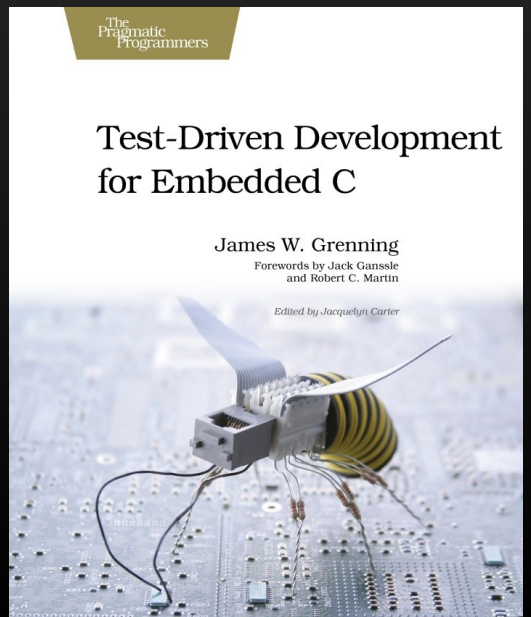
Debug Later Programming

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.

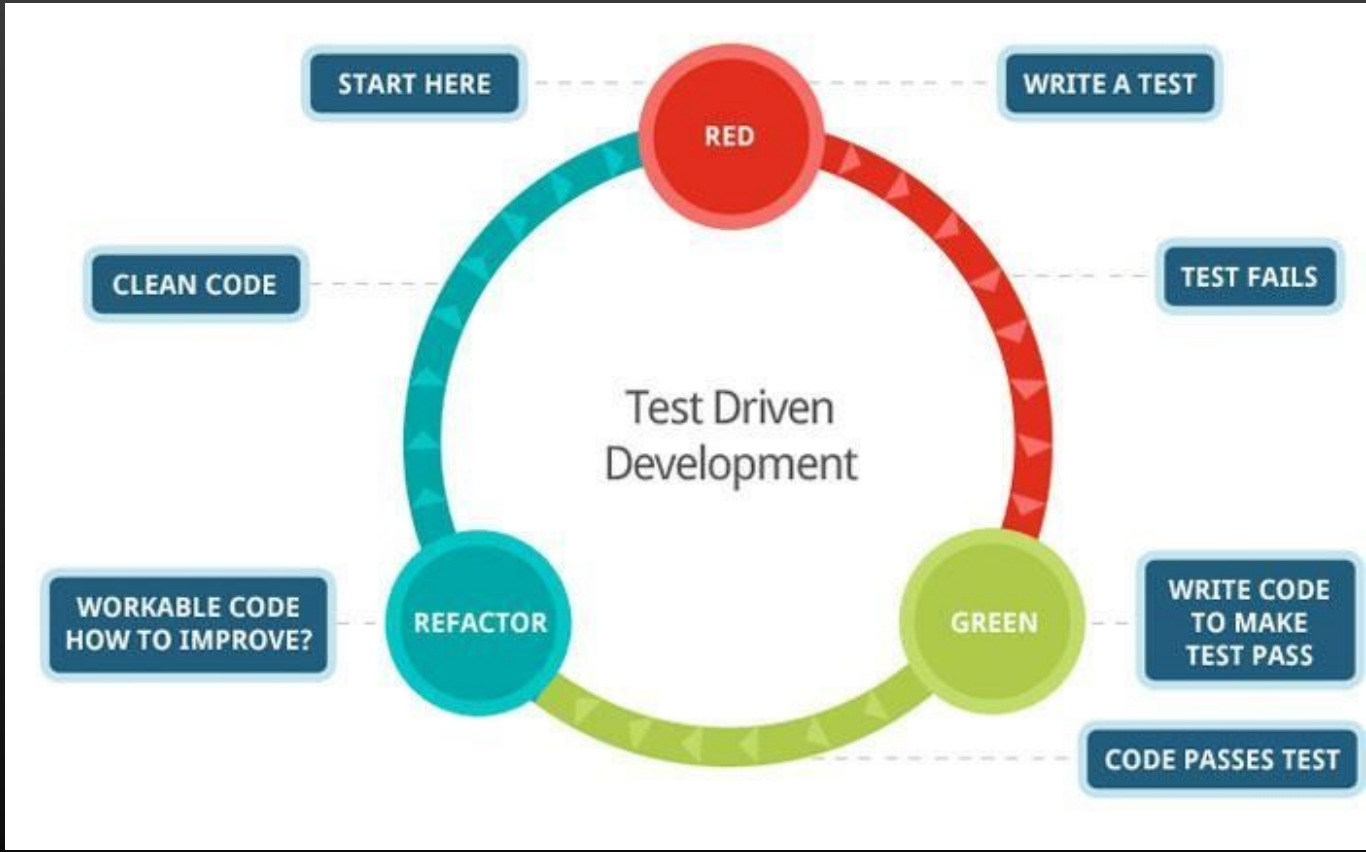
We would spend about half our time in the unpredictable activity affectionately called *debugging*. Debugging would show up in our schedules under the disguise of test and integration. It was always a source of risk and uncertainty. Fixing one bug might lead to another and sometimes to a cascade of other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



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.



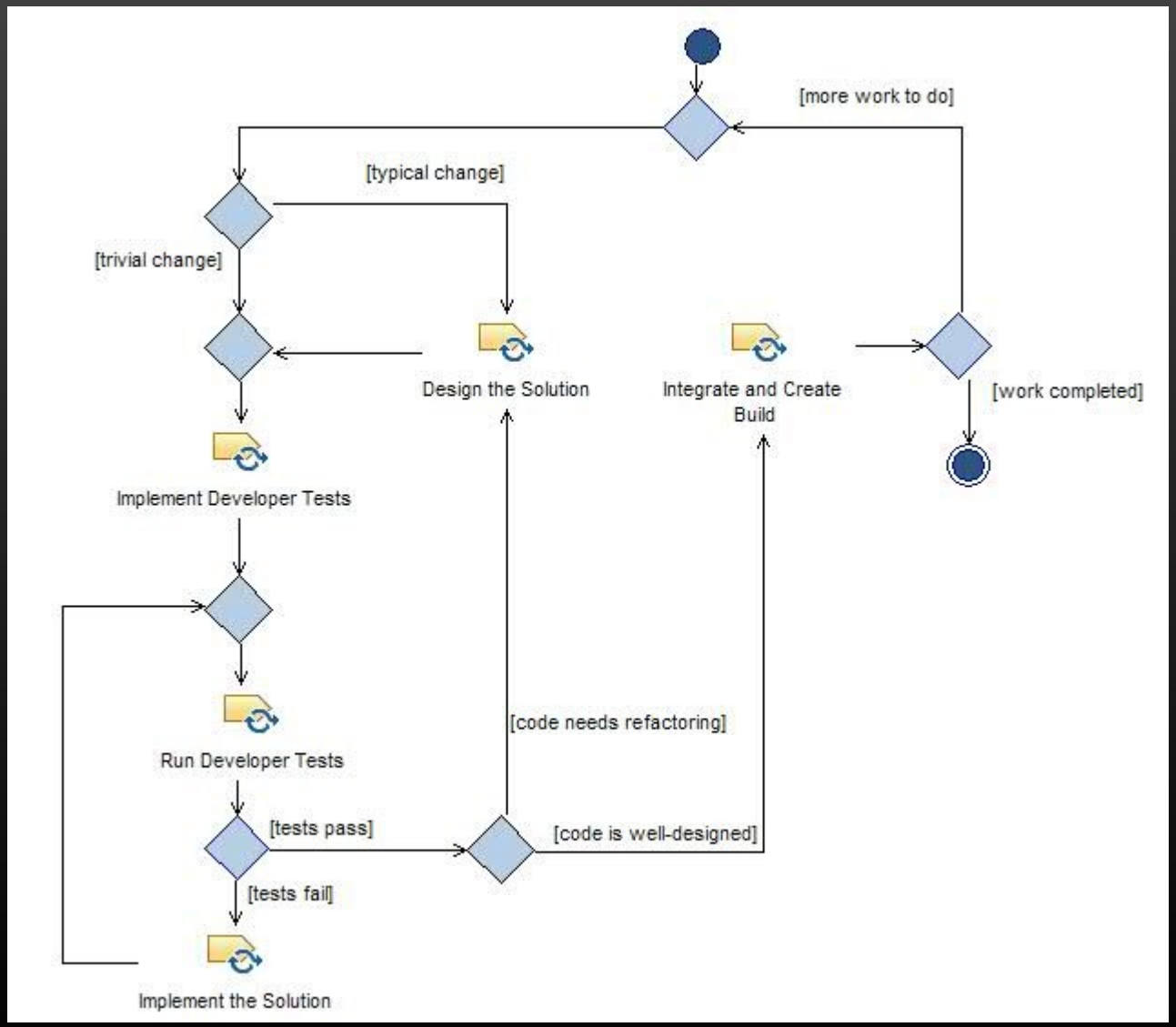
TDD: Test Driven Development



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.


[more work to do]
[typical change]
[trivial change]
Design the Solution
Integrate and Create Build
[work completed]
Implement Developer Tests
Run Developer Tests
[code needs refactoring]
[tests pass]
[code is well-designed]
[tests fail]
Implement the Solution

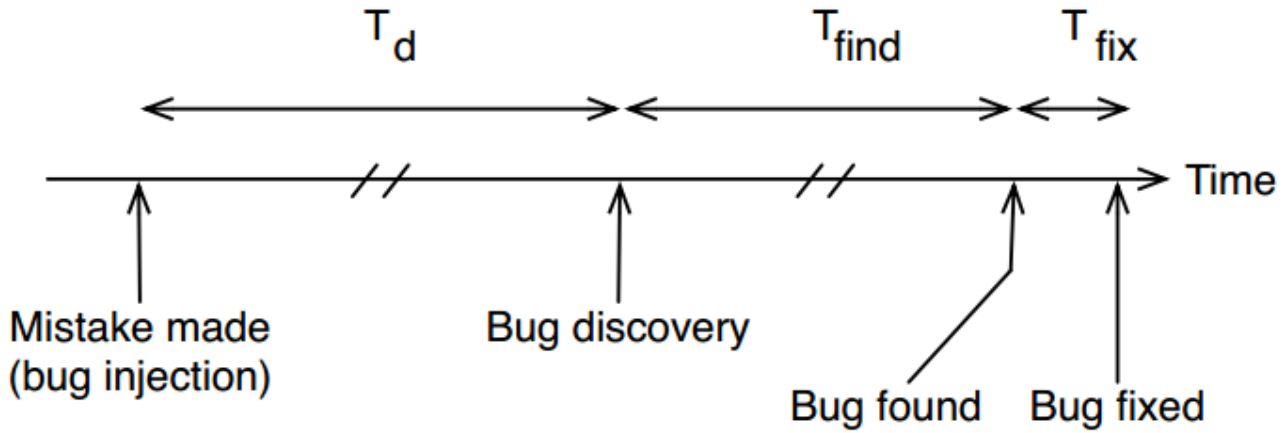


Figure 1.1: Physics of Debug-Later Programming

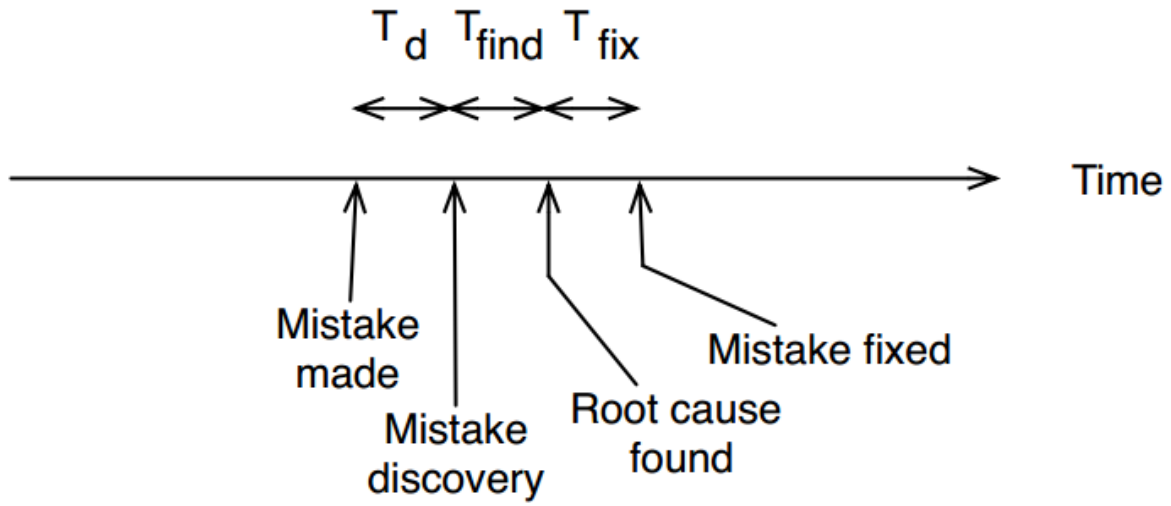


Figure 1.2: Physics of Test-Driven Development

The benefits of TDD: an investigation

A summary of selected empirical studies of test-driven development: industry participants*

Family of studies	Type	Development time analyzed	Legacy project?	Organization studied	Software built	Software size	No. of participants	Language	Productivity effect
Sanchez et al.[6]	Case study	5 years	Yes	IBM	Point-of-sale device driver	Medium	9–17	Java	Increased effort 19%
Bhat and Nagappan[7]	Case study	4 months	No	Microsoft	Windows networking common library	Small	6	C/C++	Increased effort 25–35%
	Case study	≈7 months	No	Microsoft	MSN Web services	Medium	5–8	C++/C#	Increased effort 15%
Canfora et al.[8]	Controlled experiment	5 hours	No	Soluziona Software Factory	Text analyzer	Very small	28	Java	Increased effort by 65%
Damm and Lundberg[9]	Multi-case study	1–1.5 years	Yes	Ericsson	Components for a mobile network operator application	Medium	100	C++/Java	Total project cost increased by 5–6%

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4163024

WINGMAN SOFTWARE

Story Testing

Executable Use Cases

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: (prese

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 × individual user ×

→ Principles for user stories content

Acceptance criteria: 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).

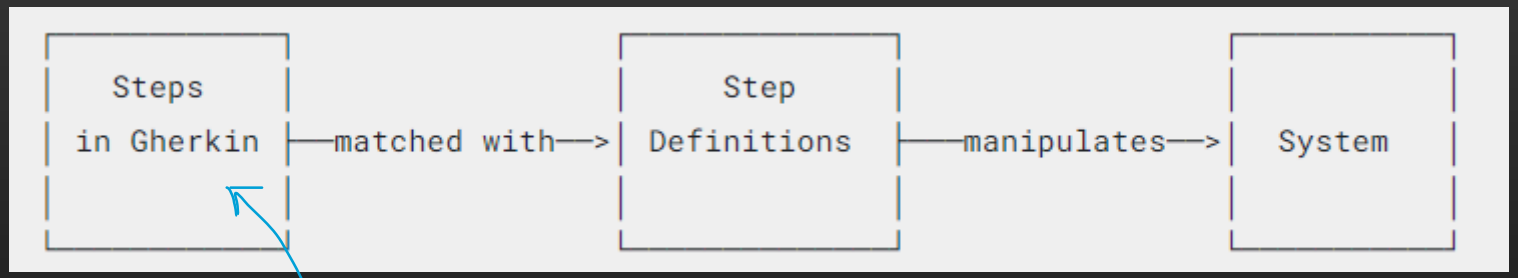
- And...

```
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
```

Cucumber framework




```
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
```

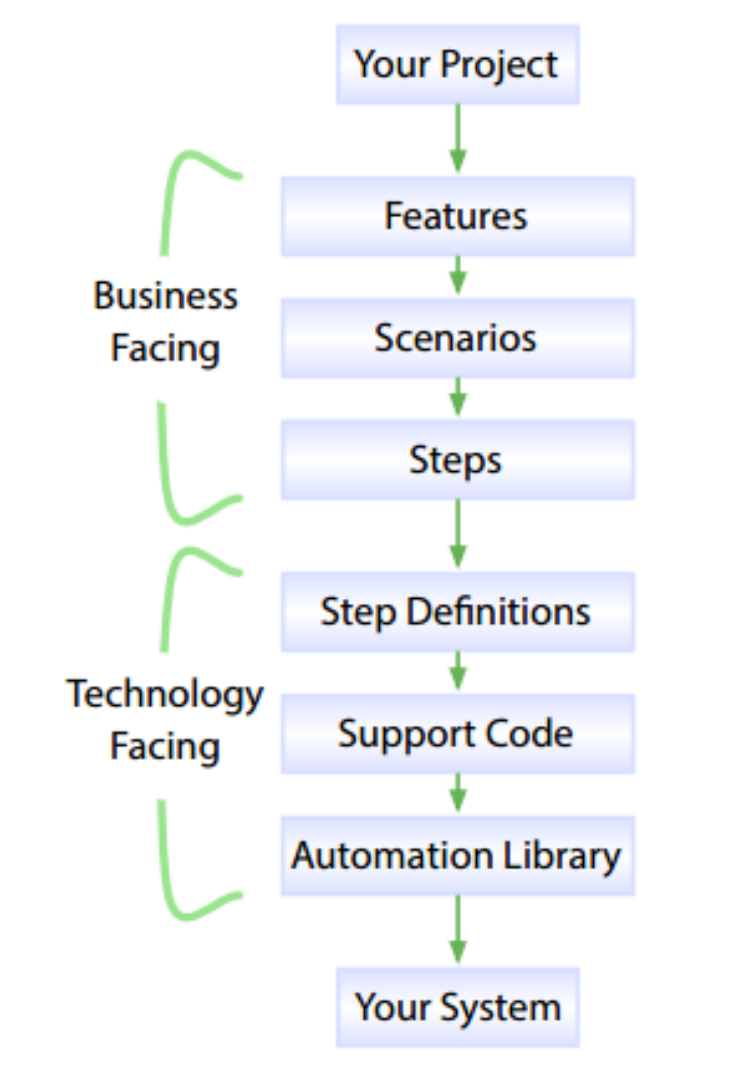
Cucumber reads specifications from plain-language text files called ***features***, examines them for ***scenarios*** to test.

Each scenario is a list of ***steps*** for Cucumber to work through.

Along with the features, you give Cucumber a set of ***step definitions***, which map the business-readable language of each step into code to carry out whatever action is being described by the step.

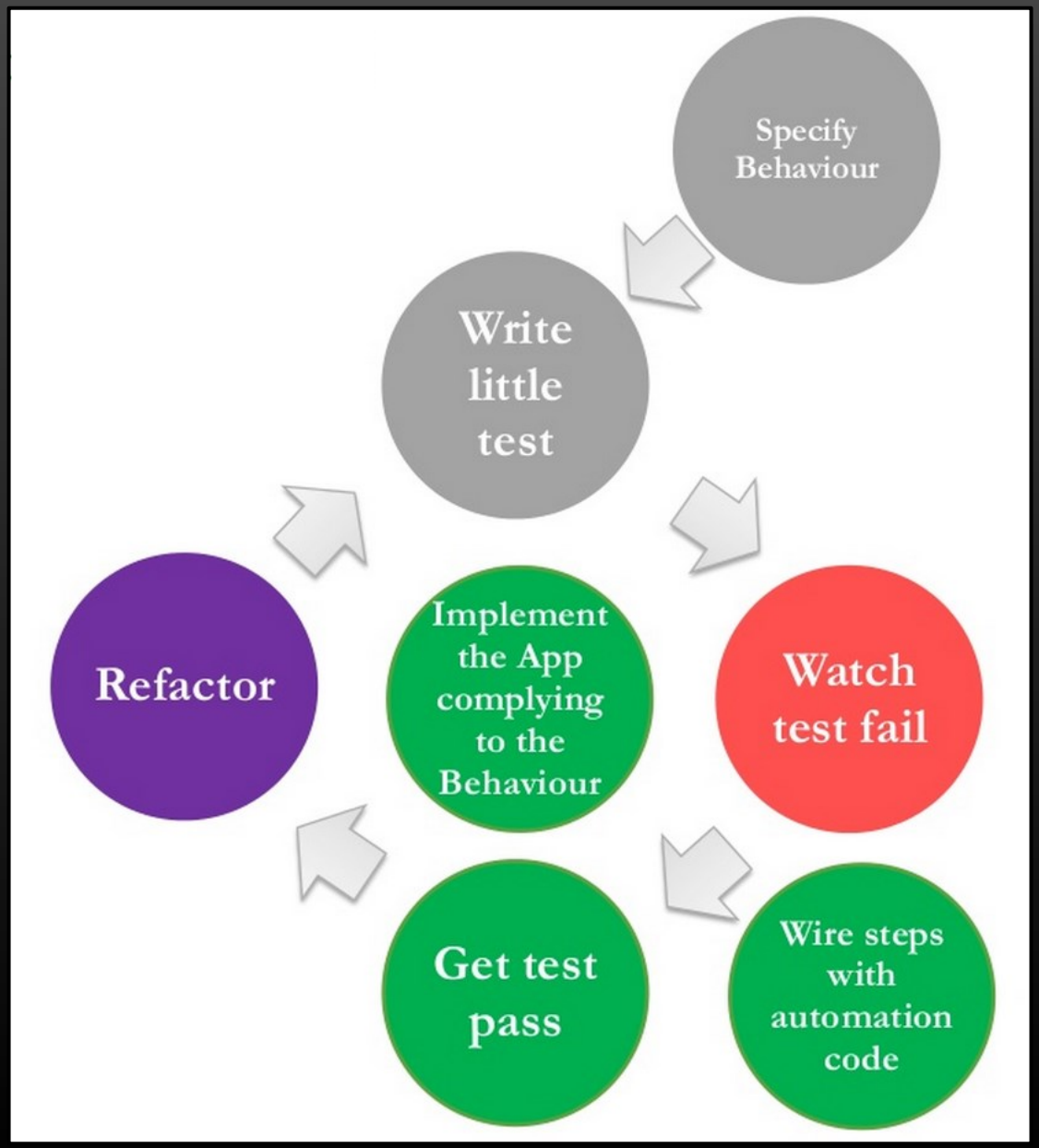
The **step definition** itself will probably just be one or two **lines of code**, specific to the domain of your application.

Sometimes that may involve using an ***automation library***, like the browser automation library Selenium.



(cucumber demo)

BDD: Behaviour-driven development



Credit: Nalin Goonawardana

Uncomplicate TDD and BDD

by JEFF NYMAN posted on 17 SEPTEMBER 2017

BDD IS AN ABSTRACTION OF TDD

So here's how I see it. The key value of TDD is that at each step of the way, you have demonstrably relevant working software as well as an itemized set of what we can call "executable specifications" that illustrate aspects of behavior. And we do this at the appropriate level of abstraction. Which takes us to BDD.

BDD is really just the addition of business concerns to the technical concerns that we deal with in TDD. BDD wasn't a reaction to TDD, as is often stated. BDD was simply an approach that let us move up the abstraction chain *as people became more comfortable with TDD*.

Simply put, BDD is about writing those conditions we talked about in the context of scenarios such that they will tell us the kind of behavior change they affect. A good barometer for adding a scenario might be asking if the scenario you are writing would be worth explaining to a business stakeholder. And you can frame that by asking what value it provides them. What aspect of the overall user experience is being captured in that scenario?

And not just "failure" in a singular sense, but failure modes based on the likely sensitivities of

http://testerstories.com/2017/09/uncomplicate-tdd-and-bdd/

context of your immediate work.

References

Core readings	Suggested readings
• "Story testing - executable use cases - for embedded systems", J. Grenning	• [Dennis]- Chap. 12. • [Pressman] – Chap. 17 ("Software Testing Strategies)