



Entendendo Testes Unitários





Olá!

Sou o Leonardo Miranda

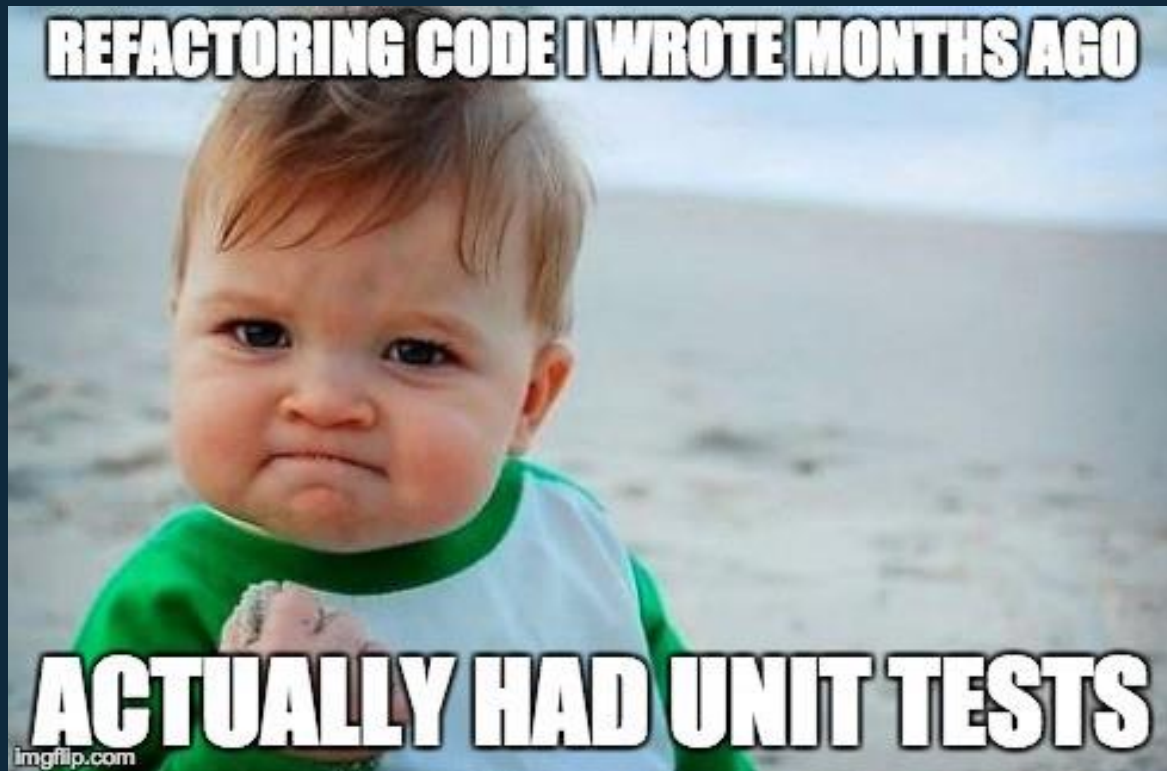
- Estudante e aprendiz.
- 4º semestre de Eng. Comp., Monitor e Representante.
- Estagiário na Anatel atuando com Engenharia de Software e Análise de dados.
- Aonde você pode me achar: Instagram(@leozinmiran), GitHub (0xl30z1n), leonardo.amaral@iesb.edu.br.



Por que usar testes?

1

- Verifica se o sistema faz o que foi determinado para fazer, e não faz nada que seja sem intenção, que não foi desenhado para fazer.
- Ajuda a encontrar bugs ou adição de uma funcionalidade;
- Diminuição de debugging;
- Aumenta a confiança ao mudar o software;
- Melhor documentação;
- Simples revisão.



(Fonte: Google)



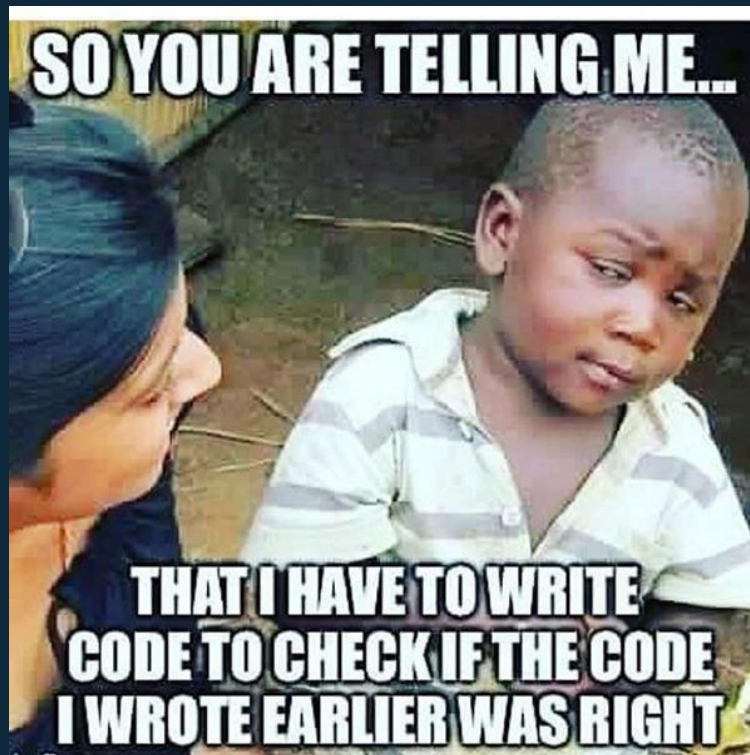
Testar é...

- ◇ Executar um programa com a intenção de achar erros;
- ◇ Testar um programa para achar o máximo de erros possíveis.

Testar não é...

- ◇ Mostrar que o programa funciona;
- ◇ Mostrar que não tem nenhum erro.





(Fonte: Google)

Guia para testagem de programa

Principle Number

Principle

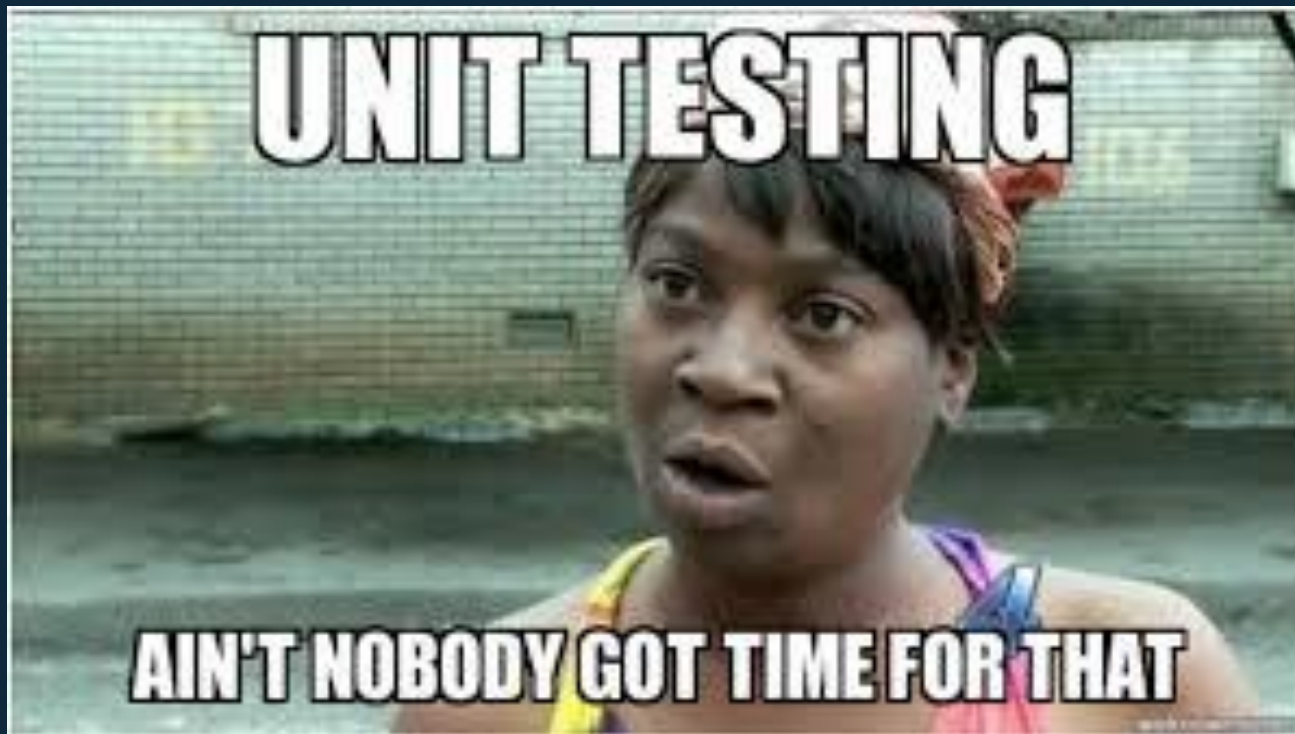
- | Principle Number | Principle |
|------------------|---|
| 1 | A necessary part of a test case is a definition of the expected output or result. |
| 2 | A programmer should avoid attempting to test his or her own program. |
| 3 | A programming organization should not test its own programs. |
| 4 | Any testing process should include a thorough inspection of the results of each test. |
| 5 | Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected. |
| 6 | Examining a program to see if it <i>does not do what it is supposed to do</i> is only half the battle; the other half is seeing whether the program <i>does what it is not supposed to do</i> . |
| 7 | Avoid throwaway test cases unless the program is truly a throwaway program. |
| 8 | Do not plan a testing effort under the tacit assumption that no errors will be found. |
| 9 | The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section. |
| 10 | Testing is an extremely creative and intellectually challenging task. |

(Fonte: The Art of Software Testing. Page 13. Glenford J. Myers)



Dilemas de testagem de software :

- É impossível testar o programa completamente;
- Testagem de software é um exercício baseado no risco;
- Testes não mostram bugs que não existem;
- Bugs seguem bugs;
- Nem todos os bugs que achar serão arrumados;
- Velocidade x Qualidade.




(Fonte: Google)

A decorative pattern of hexagons in various shades of blue and teal. Some hexagons contain icons: a lightbulb, a thumbs up, a network of nodes, a smartphone, a magnifying glass, a gear, and a speech bubble. A large teal hexagon in the center-left contains the number '2'.

2

Tipos de testes

- Testes unitários;
- Testes de Integração;
- Testes sistêmicos;
- Testes de aceitação;
- Testes de segurança...



(Fonte: Software Testing and Continuous Quality Improvement. William E. Lewis.)

“

Technique	Brief Description
Acceptance testing	Final testing based on the end-user/customer specifications, or based on use by end-users/customers over a defined period of time
Ad hoc testing	Similar to exploratory testing, but often taken to mean that the testers have significant understanding of the software before testing it
Alpha testing	Testing of an application when development is nearing completion; minor design changes may still be made as a result of such testing. Typically done by end-users or others, not by programmers or testers
Basis path testing	Identifying tests based on flow and paths of a program or system
Beta testing	Testing when development and testing are essentially completed and final bugs and problems need to be found before final release. Typically done by end-users or others, not by programmers or testers
Black-box testing	Testing cases generated based on the system's functionality
Bottom-up testing	Integrating modules or programs starting from the bottom
Boundary value testing	Testing cases generated from boundary values of equivalence classes
Branch coverage testing	Verifying that each branch has true and false outcomes at least once
Branch/condition coverage testing	Verifying that each condition in a decision takes on all possible outcomes at least once
Cause-effect graphing	Mapping multiple simultaneous inputs that may affect others to identify their conditions to test
Comparison testing	Comparing software weaknesses and strengths to competing products
Compatibility testing	Testing how well software performs in a particular hardware/software/operating system/network environment
Condition coverage testing	Verifying that each condition in a decision takes on all possible outcomes at least once
CRUD testing	Building a CRUD matrix and testing all object creations, reads, updates, and deletions
Database testing	Checking the integrity of database field values
Decision tables	Table showing the decision criteria and the respective actions
Desk checking	Developer reviews code for accuracy



(Fonte: Software Testing and Continuous Quality Improvement. William E. Lewis.)

Technique	Brief Description
End-to-end testing	Similar to system testing; the "macro" end of the test scale; involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate
Equivalence partitioning	Each input condition is partitioned into two or more groups. Test cases are generated from representative valid and invalid classes
Exception testing	Identifying error messages and exception handling processes and conditions that trigger them
Exploratory testing	Often taken to mean a creative, informal software test that is not based on formal test plans or test cases; testers may be learning the software as they test it
Free form testing	Ad hoc or brainstorming using intuition to define test cases
Gray-box testing	A combination of black-box and white-box testing to take advantage of both
Histograms	A graphical representation of measured values organized according to the frequency of occurrence used to pinpoint hot spots
Incremental integration testing	Continuous testing of an application as new functionality is added; requires that various aspects of an application's functionality be independent enough to work separately before all parts of the program are completed, or that test drivers be developed as needed; done by programmers or by testers
Inspections	Formal peer review that uses checklists, entry criteria, and exit criteria
Integration testing	Testing of combined parts of an application to determine if they function together correctly. The "parts" can be code modules, individual applications, or client and server applications on a network. This type of testing is especially relevant to client/server and distributed systems.
JADs	Technique that brings users and developers together to jointly design systems in facilitated sessions
Load testing	Testing an application under heavy loads, such as testing of a Web site under a range of loads to determine at what point the system's response time degrades or fails
Mutation testing	A method for determining if a set of test data or test cases is useful, by deliberately introducing various code changes ("bugs") and retesting with the original test data/cases to determine if the "bugs" are detected. Proper implementation requires large computational resources.



(Fonte: Software Testing and Continuous Quality Improvement. William E. Lewis.)

Technique	Brief Description
Orthogonal array testing	Mathematical technique to determine which variations of parameters need to be tested
Pareto analysis	Analyze defect patterns to identify causes and sources
Performance testing	Term often used interchangeably with “stress” and “load” testing. Ideally “performance” testing (and any other “type” of testing) is defined in requirements documentation or QA or Test Plans
Positive and negative testing	Testing the positive and negative values for all inputs
Prior defect history testing	Test cases are created or rerun for every defect found in prior tests of the system
Prototyping	General approach to gather data from users by building and demonstrating to them some part of a potential application
Random testing	Technique involving random selection from a specific set of input values where any value is as likely as any other
Range testing	For each input identifies the range over which the system behavior should be the same
Recovery testing	Testing how well a system recovers from crashes, hardware failures, or other catastrophic problems
Regression testing	Testing a system in light of changes made during a development spiral, debugging, maintenance, or the development of a new release
Risk-based testing	Measures the degree of business risk in a system to improve testing
Run charts	A graphical representation of how a quality characteristic varies with time
Sandwich testing	Integrating modules or programs from the top and bottom simultaneously
Sanity testing	Typically an initial testing effort to determine if a new software version is performing well enough to accept it for a major testing effort. For example, if the new software is crashing systems every five minutes, bogging down systems to a crawl, or destroying databases, the software may not be in a “sane” enough condition to warrant further testing in its current state
Security testing	Testing how well the system protects against unauthorized internal or external access, willful damage, etc.; may require sophisticated testing techniques

A decorative graphic on the left side of the slide. It features a central large blue hexagon with white quotation marks. Surrounding it are several smaller hexagons in shades of blue and teal, some containing icons: a lightbulb, a thumbs up, a smartphone, a magnifying glass, and a gear. There are also some abstract shapes like a network of dots and a speech bubble.

(Fonte: Software Testing and Continuous Quality Improvement. William E. Lewis.)

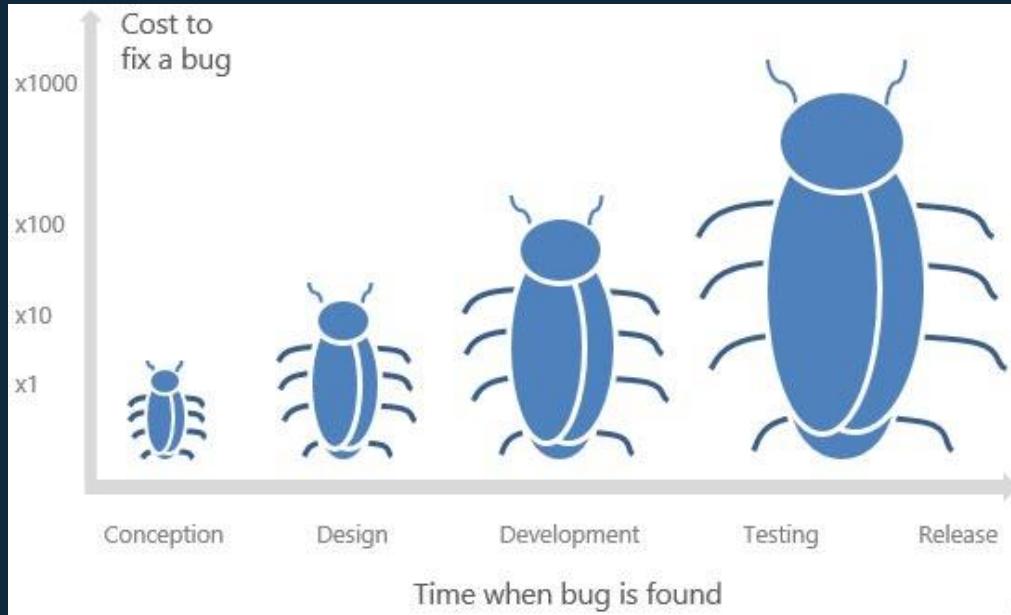
Technique	Brief Description
State transition testing	Technique in which the states of a system are first identified and then test cases are written to test the triggers to cause a transition from one condition to another state
Statement coverage testing	Every statement in a program is executed at least once
Statistical profile testing	Statistical techniques are used to develop a usage profile of the system that helps define transaction paths, conditions, functions, and data tables
Stress testing	Term often used interchangeably with “load” and “performance” testing. Also used to describe such tests as system functional testing while under unusually heavy loads, heavy repetition of certain actions or inputs, input of large numerical values, or large complex queries to a database system
Structured walkthroughs	A technique for conducting a meeting at which project participants examine a work product for errors
Syntax testing	Data-driven technique to test combinations of input syntax
System testing	Black-box type testing that is based on overall requirements specifications; covers all combined parts of a system
Table testing	Testing access, security, and data integrity of table entries
Thread testing	Combining individual units into threads of functionality which together accomplish a function or set of functions
Top-down testing	Integrating modules or programs starting from the top
Unit testing	The most “micro” scale of testing; to test particular functions or code modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. Not always easily done unless the application has a well-designed architecture with tight code; may require developing test driver modules or test harnesses
Usability testing	Testing for “user-friendliness.” Clearly this is subjective, and will depend on the targeted end-user or customer. User interviews, surveys, video recording of user sessions, and other techniques can be used. Programmers and testers are usually not appropriate as usability testers
User acceptance testing	Determining if software is satisfactory to an end-user or customer
White-box testing	Test cases are defined by examining the logic paths of a system

A decorative graphic on the left side of the slide consists of a cluster of hexagons in various shades of blue and cyan. Some hexagons contain white icons: a lightbulb, a thumbs-up, a network of nodes, a smartphone, a magnifying glass, and a gear. A large cyan hexagon in the center of this cluster contains the white number '3'.

Classificações

- Static and Dynamic Testing;
- Manual and automated Testing;
- Black-box and White-box Testing;
- Functional and Structural Testing.

Custo de correção de bug



(Fonte: Google)



4

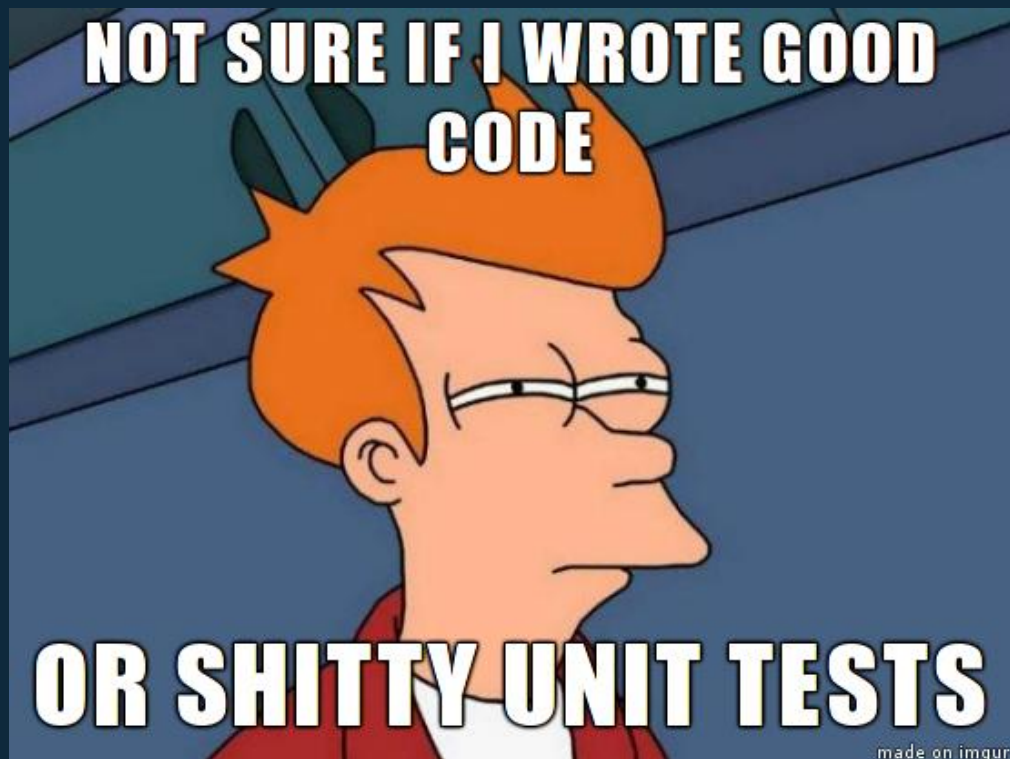
Testes Unitários:

- Busca cobrir uma funcionalidade da aplicação em seu nível mais básico.
- Foca nas menores unidades do código.
- Testa cada pedaço unitário de código, sendo geralmente um método ou classe individual, em isolamento para ver se determinadas condições respondem conforme esperado. São fáceis de escrever. São testados rapidamente.
- Ajuda a descobrir erros. Diminui o tempo para arrumar erros.
- Ajuda a moldar o design e a qualidade do código.
- Explícito é melhor do que implícito (The Zen of Python). Ajudam a conhecer de forma explícita qual o comportamento do sistema.

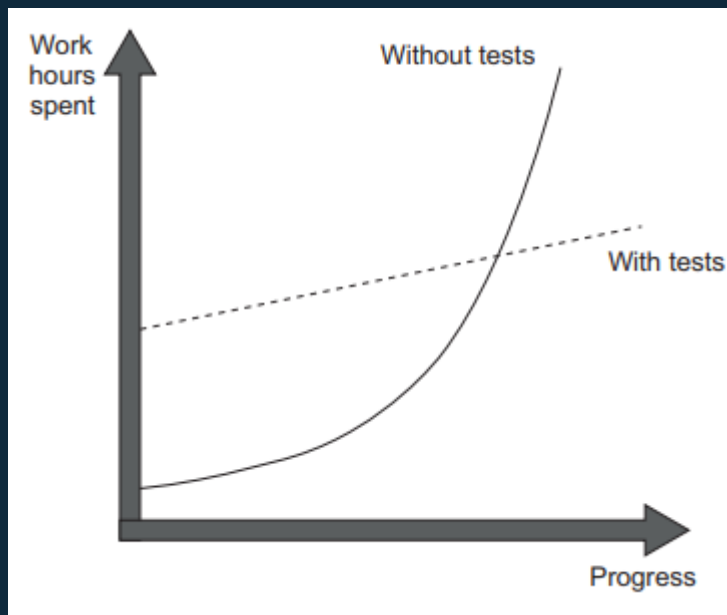


Objetivos:

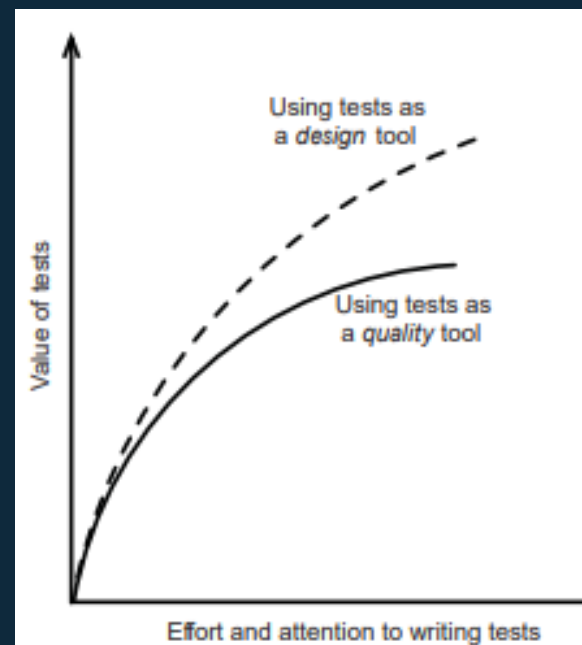
- AAA – Arrange , Act, Assert;
- 100% de cobertura do código não é o objetivo, o que deve ser analisado é a diferença do valor desses testes para você. Depende de que parte do código não é coberto pelos testes e se estes testes são capazes de ressaltar os erros de programação;
- Um bom test suite é integrado ao ciclo de desenvolvimento, foca apenas nas partes mais importantes do código e retorna o máximo valor sobre menor custo de manutenção;
- Código legível é código sustentável.



(Fonte: Google)



(Fonte: Unit Testing. Principles, Practices, and Patterns. Vladimir Khorikov.)



(Fonte: Effective Unit Testing. Lasse Koskela)



TDD é...

- ◇ Um processo formado pela repetição de um pequeno ciclo de desenvolvimento. Red, Green, Refactor cycle;
- ◇ É um processo, pode ser usado com diversas categorias de testes, como testes unitários, testes de integração e dentre outros;
- ◇ TDD é uma metodologia de escrita de testes;
- ◇ É uma habilidade de pensar sobre o problema que está tentando resolver antes de tentar resolver.





Teste Unitário é...

- ◇ Apenas um tipo de teste;
- ◇ Pode ser abordado como test-last (o código de produção é escrito primeiro e depois os testes são adicionados, às vezes, uma tarefa inteira é completada antes de algum teste ser escrito) ou test-first (envolve escrever muitos testes e depois escrever muitas linhas de código de produção), este sendo a abordagem mais recomendável.



A decorative pattern of hexagons in various shades of blue and cyan on the left side of the slide. Some hexagons contain icons: a lightbulb, a thumbs up, a network node, a smartphone, a magnifying glass, a gear, and a speech bubble.

5

DEMO



Unittest framework

- Class, herança;
 - Devem começar com test_...
 - Fixtures(setup e teardown);
 - Assert;
 - Test suite;
 - Test runner;
 - Skip;
 - Raise exception.
 - Pode fazer integration tests.
- `py -m unittest name_file.py;`
 - `py -m -v unittest name_file.py;`
 - Running single test.





Pytest framework

- Funções e métodos;
- Devem começar com test_...
- Assert;
- Parametrize;
- Fixture (classe);
- Markers (fail, skip);
- Pode fazer integration tests.
- pip install pytest;
- pytest file_name.py;
- plugins (pytest-cov).





Diferença entre unittest e pytest

Unittest

- Mais rápido para poucos testes;
- Necessita de menos conhecimento;
- Já vem na standard library;
- Segue o padrão xUnit;
- Utiliza de herança.

Pytest

- Diversos plugins (integration tests, paralelismo, concurrency, coverage...);
- Pode ser usado como função;
- Pode seguir o padrão xUnit ou usar os próprios fixtures;
- Parametrization.



Referências

6

- Software Testing. Ron Patton.
- The Art of Software Testing. Glenford J. Myers.
- Software Testing and Continuous Quality Improvement. William E. Lewis.
- Effective Unit Testing. A guide for Java developers. Lasse Koskela.
- Unit Testing. Principles, Practices, and Patterns. Vladimir Khorikov.
- xUnit Test Patterns. Refactoring Test Code. Gerard Meszaros.
- The Art of Unit Testing with examples in C#. Roy Osherove.
- JUnit in action. Vincent Massol.
- Testing Python. Applying Unit Testing, TDD, BDD, and Acceptance Testing. David Sale.
- Python Testing Cookbook. Greg L. Turnquist.
- Python Testing with pytest. Simple, Rapid, Effective, and Scalable. Brian Okken.
- Python Unit Test Automation. Ashwin Pajankar.
- The Art of Unit Testing with examples in .NET. Roy Osherove.
- <https://docs.python.org/3/library/unittest.html>
- <https://docs.pytest.org/en/6.2.x/contents.html>
- https://docs.pytest.org/en/latest/reference/plugin_list.html
- <https://github.com/renzon/pytest-vs-unittest>

Recursos Extras



- ✓ Mock e patch.
- ✓ TDD.
- ✓ Refatoração.
- ✓ OOP.
- ✓ Software Engineering at Google. Titus Winters, Tom Manshreck and Hyrum Wright.
- ✓ <https://github.com/PlatziMaster/challenge-python-05>
- ✓ <https://github.com/AndyLPK247/python-testing-101>
- ✓ <https://github.com/MaartenGr/UnitTesting>
- ✓ https://github.com/htorrence/pytest_examples





Obrigado!

Alguma pergunta?

