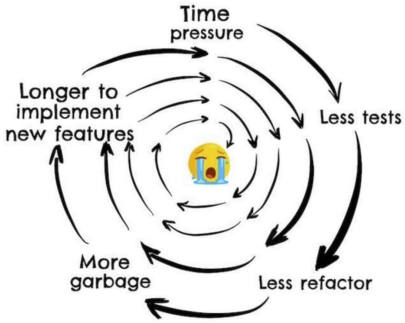
Looking Back

Luiza da Silva

01-06-2023

Technical Practices

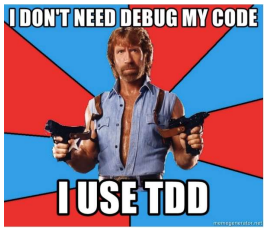


Why do software projects fail again and again?

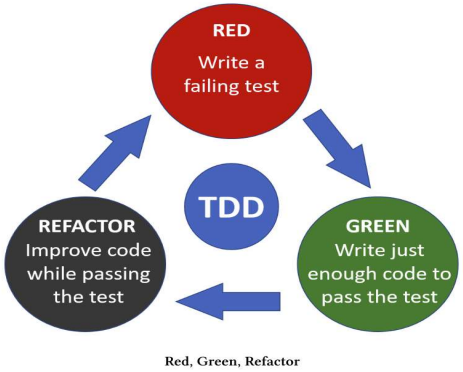
- **TECHNICAL DEBT** (accidental or intentional)

We need to address ***accidental complexity*** or ***tech debt*** as it emerges (or as it is found)

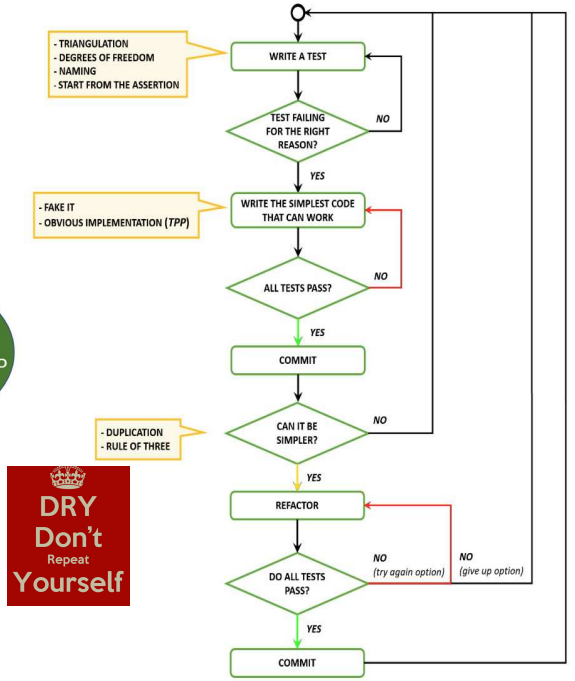
- Technical **Feedback** Practice
  - Test Driven Development
- Technical **Design** Practices
  - Refactoring
  - Simple Design



TDD – Start with the Tests



Red, Green, Refactor



Write test before implementation, test one behavior per test

Classic School TDD

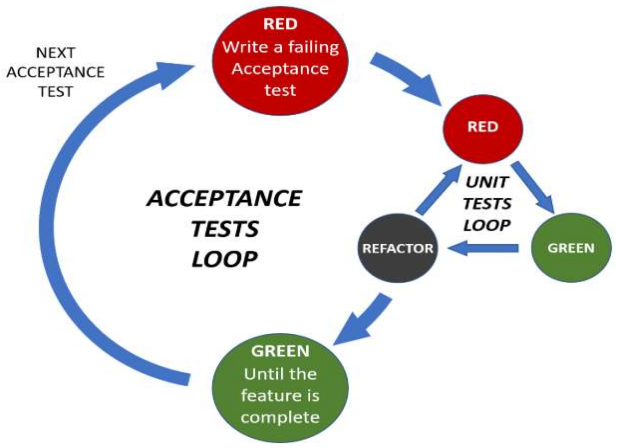
- Test small units
- Bottom-up
- Limited mocking
- Well suited for legacy code

Outside-In TDD

- Emphasis on end-to-end, start with Acceptance Testing (ATDD)
- Top-down
- Extensive Mocking (aka Test Doubles)
- Communication tool for between developers, testers and stakeholders

Transformation Priority Premise

#	Transformation	Start code	End code
1	{} -> nil	{}	[return] nil
2	Nil -> constant	[return] nil	[return] "1"
3	Constant -> constant+	[return] "1"	[return] "1" + "2"
4	Constant -> scalar	[return] "1" + "2"	[return] argument
5	Statement -> statements	[return] argument	[return] min(max(0, argument), 10)
6	Unconditional -> conditional	[return] argument	if(condition) [return] 1 else [return] 0
7	Scalar -> array	dog	[dog, cat]
8	Array -> container	[dog, cat]	
9	Statement -> tail recursion	a + b	a + recursion
10	If -> loop	if(condition)	loop(condition)
11	Statement -> recursion	a + recursion	recursion
12	Expression -> function	today – birth	CalculateBirthDate()
13	Variable -> mutation	day	var Day = 10; Day = 11;

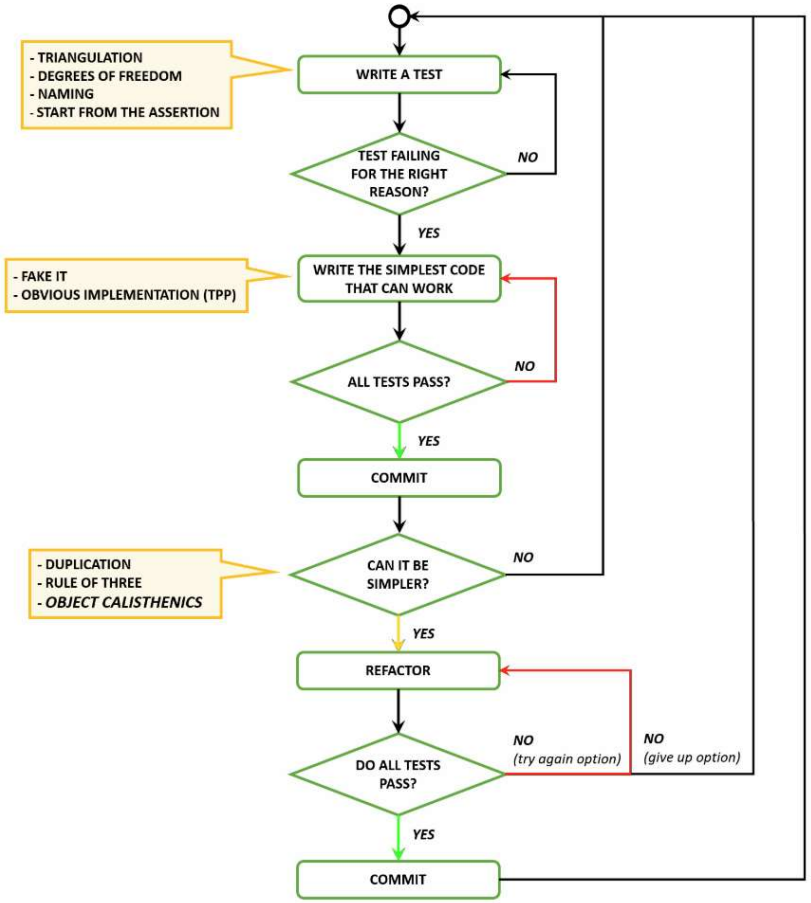


Simple Design

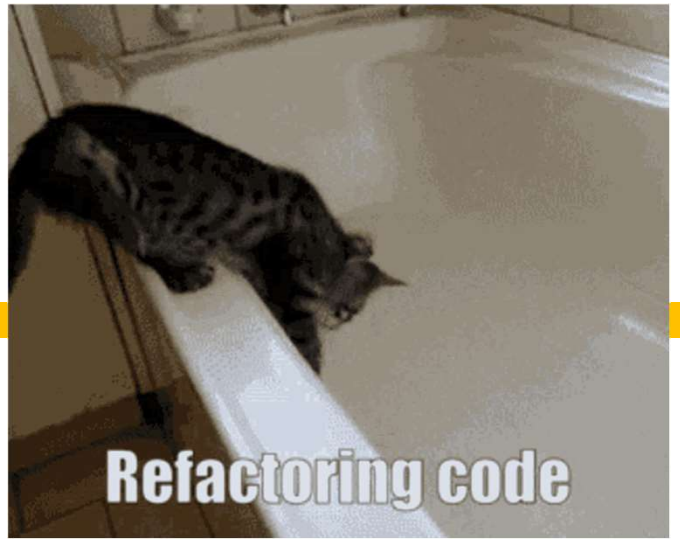
TDD and DRY is not enough! We need some help with the design

Object calisthenics

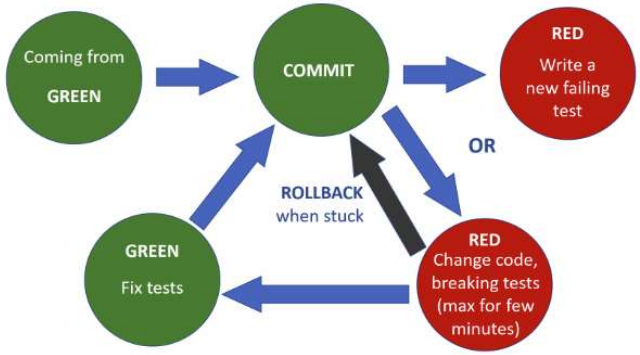
1. Only one level of indentation per method
2. Don’t use the ELSE keyword
3. Wrap all primitives and strings (wrap primitive types in classes)
4. First class collections (wrap collections in classes)
5. One dot per line
6. Don’t abbreviate
7. Keep all entities small
8. No classes with more than two instance variables
9. No getters/setters/properties
10. All classes must have state



Refactoring



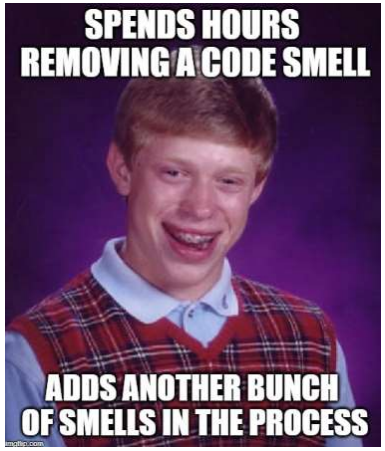
- Change design without changing behavior
- Find and shape abstractions
- Stay green while refactoring
- Don’t change production code that is not covered by tests
- Learn the shortcuts of your IDE
- Commit often
- Refactor for readability before design
- Parallel Change (Expand, Migrate and Contract

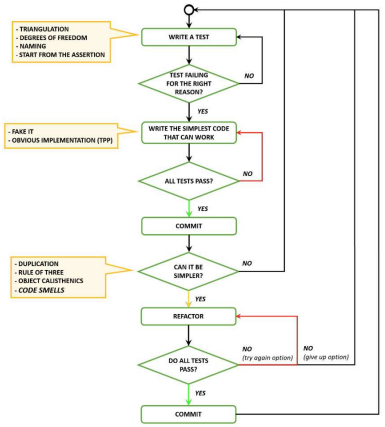


Extended TDD cycle: the Refactor cycle

Simple Design and Refactoring

Code Smells and Refactoring





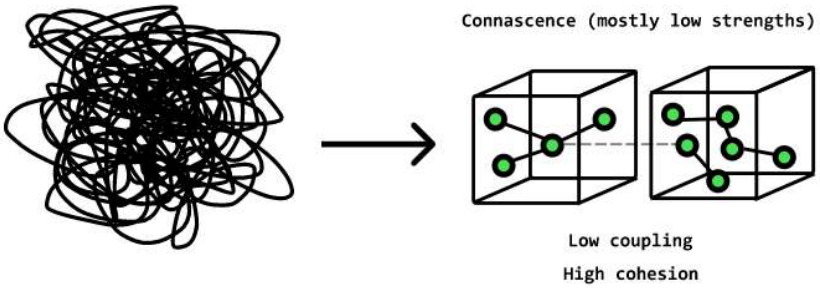
Switch Statements OOA Replace Conditional with Polymorphism Replace Type Code with Subclasses Replace Type Code with State/Strategy Move Accumulation to Visitor Replace Conditional Dispatcher with Command Replace Parameter with Explicit Methods Introduce Null Object	**Inappropriate Intimacy** COU Move Method Move Field Extract Class Hide Delegate Replace Inheritance with Delegation	**Large Class** BLO Extract Class Extract Subclass Extract Interface Replace Data Value with Object Replace Conditional Dispatcher with Command Replace Implicit Language with Interpreter Replace State-Altering Conditionals with State
Primitive Obsession BLO Replace Data Value with Object Encapsulate Composite with Builder Introduce Parameter Object Extract Class Move Embellishment to Decorator Replace Conditional Logic with Strategy Replace Implicit Language with Interpreter Replace Implicit Tree with Composite Replace State-Altering Conditionals with State Replace Type Code with Class Replace Type Code with State/Strategy Replace Type Code with Subclasses Replace Array With Object	**Duplicated Code** DIS Chain Constructors Extract Composite Extract Method Extract Class Form Template Method Introduce Null Object Factory Method Pull Up Method Pull Up Field Substitute Algorithm Adapter	**Long Method** BLO Extract Method Compose Method Introduce Parameter Object Move Accumulation to Collecting Parameter Move Accumulation to Visitor Decompose Conditional Preserve Whole Object Replace Conditional Dispatcher with Command Replace Conditional Logic with Strategy Replace Method with Method Object Replace Temp with Query
Divergent Change CHP Extract Class	**Shotgun Surgery** CHP Move Method Move Field Inline Class	**Feature Envy** COU Extract Method Move Method Move Field
Long Parameter List BLO Replace Parameter with Method Introduce Parameter Object Preserve Whole Object	**Data Clumps** BLO Extract Class Preserve Whole Object Introduce Parameter Object	**Parallel Inheritance** Hierarchies CHP Move Method Move Field
Middle Man COU Remove Middle Man Inline Method Replace Delegation with Inheritance	**Data Class** DIS Move Method Encapsulate Field Encapsulate Collection	**Message Chains** COU Hide Delegate Extract Method Move Method
Speculative Generality DIS Collapse Hierarchy Rename Method Remove Parameter Inline Class	**Temporary Field** OOA Extract Class Introduce Null Object	**Lazy Class** DIS Collapse Hierarchy Inline Class
Refused Bequest OOA Push Down Field Push Down Method Replace Inheritance with Delegation	**Alternative Classes with Different Interfaces** OOA Unify Interfaces with Adapter Rename Method Move Method	**Incomplete Library Class** COU Introduce Foreign Method Introduce Local Extension
Comments DIS Rename Method Method Introduce Extract Assertion	Dead Code DIS	

Refactor code smells table

BLO – Bloater, CHP – Change preventer, COU – Coupler, DIS – Dispensable, OOA – Object Orientation Abuser

Object calisthenics violation	Code smells consequence
Only one level of indentation per method	Long Method
Don't use the ELSE keyword	Long Method / Duplicated Code
Wrap all primitives and strings	Primitive Obsession / Duplicated Code / Shotgun Surgery
First class collections	Divergent Change / Large Class
One dot per line	Message Chains
Don't abbreviate	NA
Keep all entities small	Large Class / Long Method / Long Parameter List
No classes with more than two instance variables	Large Class
No getters/setters/properties	NA
All classes must have state, no static methods, no utility classes	Lazy Class / Middle Man / Feature Envy

Simple Design



Core Principles

- **Four rules of simple design**
  - Passes tests
  - Reveals intention
  - No duplication
  - Fewest Elements
- **SOLID++**
  - Single Responsibility
  - Open/Closed
  - Liskov Substitution
  - Interface Segregration
  - Dependency Inversion
  - Balanced Abstraction
  - Least Astonishment (WTF)

Cohesion - Maximize

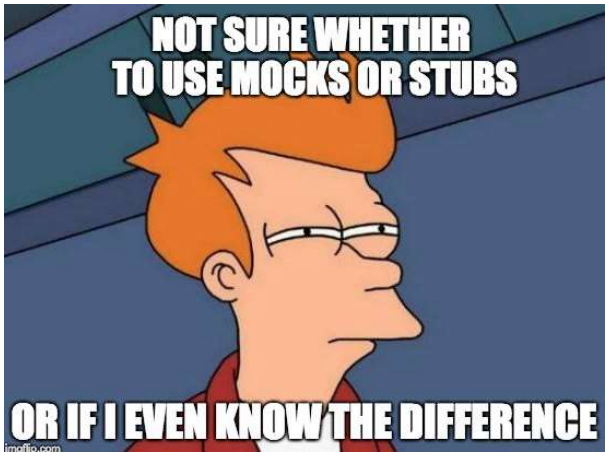
Coupling - Minimize

Connascence - Optimize

Connascence

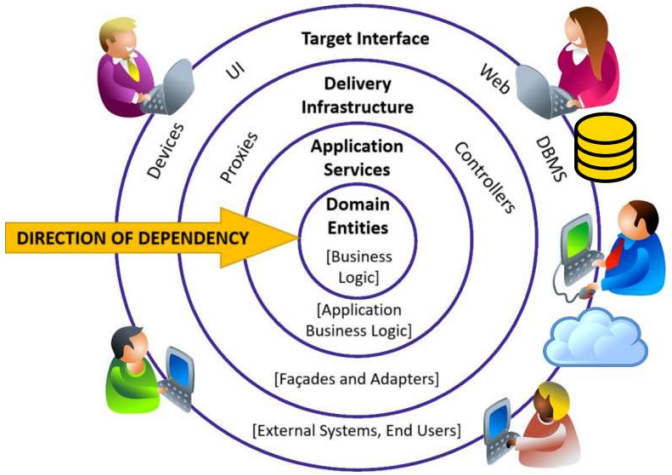
Outside-In TDD: Test Doubles

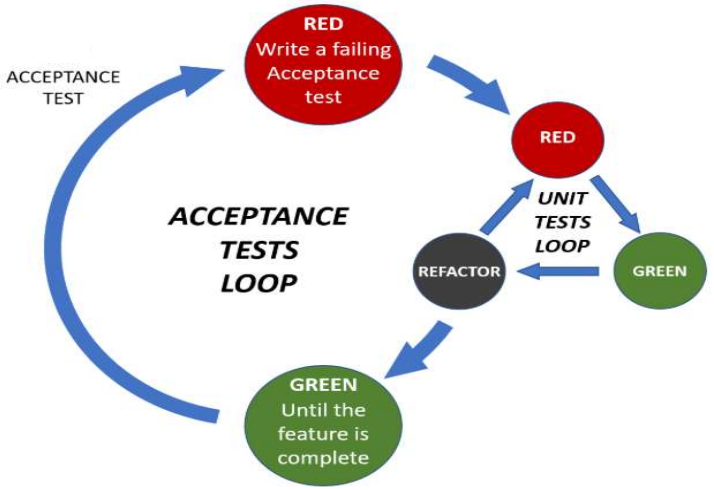
- Stub/Fake
- Mock/Spy
- Command-Query Separation
  - Command – modify state but does not return it
    - Use Mock/Spy in Assert part
  - Query – returns state but does not modify it
    - Use Stub/Fake in Arrange part
- Guidelines
  - Only for classes we own
  - Verify as little as possible in a test
  - Don’t use test doubles for isolated objects
  - Don’t add behavior inside test doubles
  - Only use test doubles for immediate neighbors
  - Same class can act both as stub and mock



Beyond Design

- Outside-In Mindset
  - Onion Architecture
  - Modular, loosely coupled architecture
  - Business-first view
  - YAGNI
  - Focus on public interfaces => minimizes **entropy**
  - Encourages readability

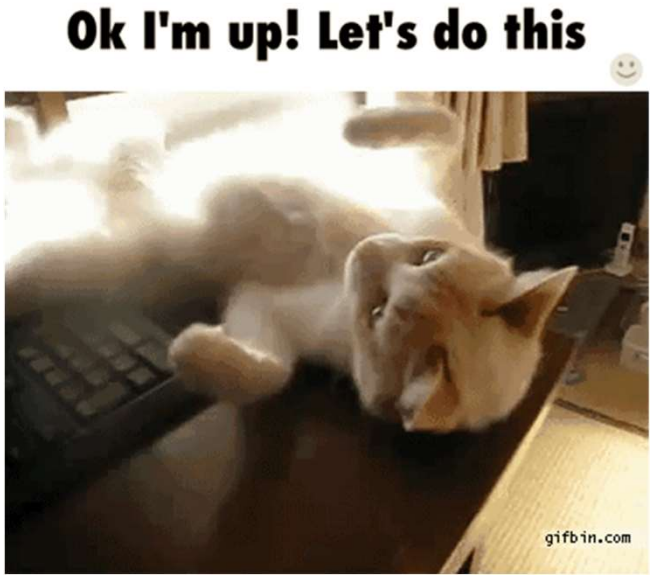




1. Define Acceptance Test
2. Make Acceptance Test FAIL outer loop
3. Make Acceptance Test PASS inner loop

Summary

- We now have the tools to start a different software journey
- What are we waiting for?



Questions?

Thank you!

Luiza Helena da Silva
luizahs@hotmail.com