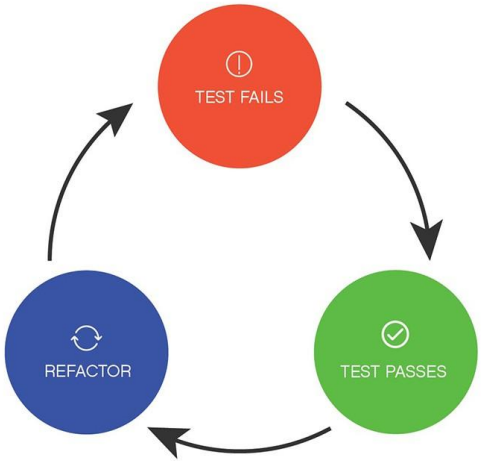
Test Driven Development

A short recap of what we learned

Benefits of TDD?

- Design emerges
- Test as documentation
- Faster debugging
- Courage through safety net

3 Basic Steps of TDD



1. Write a failing Test
2. Write just enough code to pass that failing test
3. Refactor your test

TDD Habits

F.I.R.S.T.

- **Fast**
  - Run Tests often -> they have to run fast
- **Isolate**
  - Tests work in any order
- **Repeatable**
  - Test always have the same result (no flaky tests)
- **Self validating**
  - No Human interpretation necessary -> green or red!
- **Timely**
  - Write at the right time -> before the code you want to test

Also:

Test should only check **ONE single behaviour** with only **ONE logical assertion** per test

How to approach TDD?

Baby steps

- Fake implementation
- Obvious implementation
- Triangulate with more tests

Behaviour not implementation

- We don’t care about the details -> details may change

Also:

- Commit often
- Let IDE take over refactoring

Evolve Code with TPP

Transformation Priority Premise -> What is the obvious implementation?

Providing Guidelines for Obvious (most simple) Implementation

- Start simple
- transform code to more complex code if needed

#	TRANSFORMATION	STARTING CODE	FINAL 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 arguments`
6	unconditional => **conditional**	`return arguments`	`if(condition)return arguments`
7	scalar => **array**	`dog`	`[dog, cat]`
8	array => **container**	`[dog, cat]`	`{dog = "DOG", cat = "CAT"}`
9	statement => **recursion**	`a + b`	`a + recursion`
10	conditional => **loop**	`if(condition)`	`while(condition)`
11	recursion => **tail recursion**	`a + recursion`	`recursion`
12	expression => **function**	`today - birthday`	`CalculateAge()`
13	variable => **mutation**	`day`	`var day = 10; day = 11;`
14	**switch case**		

Object Calisthenics

kalos and sthenos -> Beauty and Strength

Build “strong and beautiful” Code -> Make your code **easier to understand and maintain**

- ➔ Only one level of indentation per method
- ➔ Don’t use the ELSE keyword
- ➔ Wrap all primitives and strings
- ➔ First class collections (wrap all collections)
- ➔ Only one dot per line ~~`dog.Body.Tail.Wag()`~~ `=> dog.ExpressHappiness()`
- ➔ No abbreviations
- ➔ Keep all entities small
  [10 files per package, 50 lines per class, 5 lines per method, 2 arguments per method]
- ➔ No classes with more than two instance variables
- ➔ No public getters/setters/properties

Personal Conclusion

New approaches for me -> Mob and TDD

- In theory, tried out, but never with a “correct” approach

Big improvements in Mob

See the value of TDD more clearly thanks to 3 Step Rule and applying it myself

- Thinking Changed: Writing a Test first is not slower -> helps us do the right thing, in the right way
- Helpful Guidelines

Thank You!