Coderetreat

<DATE>

<LOCATION>

Me

You?

What do we learn?

- Design skills
  - minute-by-minute decisions we do daily
- Delete code

What is (a typical) coderetreat

- 1 day of coding
- Conway's game of life
  - (simple rules, learning to program - not the domain)
- 6 pairing partners
- ~5 constraints
- Language agnostic

What to expect todav?

- Practice the fundamentals of programming
  - OO, functional, clean code, TDD, refactoring
- An intense day of coding
- And a lot of fun

Structure of the day

- 15 min intro
- 3 sessions of 45minutes + 15min retro
- 1 hour lunch
- 3 sessions of 45min + 15min retro
- 15 min closing circle

principles

- We are all valued - everyone counts
- We are all here to learn.
- Focus on practice & experiment
- Try not to finish the problem
- DELETE YOUR CODE after each session
- Have Fun

prerequisites

- Computer
- Coding environment
- Test environment
  - unless you want to start by creating one of your own
- Source control (git, SVN, mercurial, etc.)

The 4 elements of simple design

1. Passes its tests
2. Maximizes clarity
3. Minimizes duplication
4. Has fewer elements

1. Passes its tests

Not “automated tests”

About correctness and verification.

Faster feedback cycle is better.

“If you are to ask how fast your test suite should be, it should be faster”

2. Expresses Intent

How quickly can you find the part that should be changed?

A source of code smells - if it is hard to give an expressive name, maybe the (unit) is doing too much

3. No Duplication

Not about code, but knowledge.

“Every piece of knowledge should have one and only one representation”

4. Small

Code that is no longer used?

Duplicate abstractions - could I combine

Duplication in behaviour -> sign of a missing Abstraction?

Conway's game of life

Having an **infinite 2D orthogonal** universe. Being given an initial generation called a **seed**. The following rules are applied **simultaneously**

The rules

A live cell having less than 2 live neighbors dies

A live cell having 2 or 3 live neighbors lives

A live cell having more than 3 neighbors dies

A dead cell having 3 neighbors becomes alive

First session

- Find a pair
- Choose programming language
- Setup the environment
- Make a decision on how to start
- We’ll start in 5 minutes.

Why coderetreats?

- Learn through pairing
  - … out of comfort zone
- deliberate practice
- experiment
  - ...safely in safe environment

pair programming
no naked primitives

- TDD Ping-pong

- No naked primitives
  - a.k.a primitive obsession

3nd session - no naked primitives

- No conditionals
- only 4 lines per method
- No naked primitives
  - a.k.a primitive obsession

Lunch

- 1h

4th session - single responsibility

- Precise names split (no and/or)
- No ifs, switch
- No try/catch
- One assert per test
- One behaviour per test

5th session
OO or FP?

- Use only immutable objects
  OR
- Tell - Don’t Ask.
  - That is, Not getters / setters.

6th session - wrap up

- Your choice of today's sessions:
  - TDD (mute/evil) ping-pong
  - no naked primitives/conditionals
  - Single responsibility
  - Only Immutables
  - Tell-Don't-Ask

Thank you

- Thanks for sponsor (Agile Finland ry)
- closing circle:

Closing circle

- What, if any, did you learn today?
- What, if any, did surprise you?
- What, if any, will you take to work with you ‘tomorrow’?

3nd Session - Sapir-Whorf hypothesis

- Every test name should describe behaviour

4th session - taking baby steps

1) Initialize source control repository
2) Start a timer for 2 minutes
3) Write exactly one test
    a) Timer rings, the test is red, then revert and go to 2)
    b) The test is green before the timer rings, then commit
4) Restart timer
5) Refactor
    a) Timer rings, the refactoring is not complete then revert and restart
    b) The refactoring is complete before the timer rings, commit and go to 2)

Note.

The timers must run continuously, don't stop to talk!

5th session - obiect calisthenics

1. One level of indentation per method
2. Don’t use the ELSE keyword
3. Wrap all primitives and Strings
4. First class collections
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

Nth session - TDD-as-if-you-meant-it

1. Write exactly **one** failing **test**
2. Make the test pass by writing implementation code in the test method
3. When duplication is spotted extract the implementation from tests to:
    1. a new method in the test class
    2. an existing method in the test class
4. When more methods belong together extract them into a new class
5. Refactor as required

5th session - brutal refactoring game

1. Lack of tests
2. Name not from domain
3. Name not expressing intent
4. Unnecessary if
5. Unnecessary else
6. Duplication of constant
7. Method does more than one thing
8. Primitive obsession
9. Feature envy
10. Method too long (> 6 lines)
11. Too many parameters (> 3)
12. Tests: not unitary
13. Tests: setup too complex
14. Tests: unclear action
15. Tests: more than one assert
16. Tests: no assert
17. Tests: too many paths

2nd Session - verbs instead of nouns

- TDD ping-pong
- Every class name and variable name needs to be a verb.
  - CreatesCellGeneration
  - AppliesRuleNumberOne

2nd Session - Sapir-Whorf hypothesis

- Every test name should describe behaviour
- No assertEqual, rather
  - assertThat(foo, is(equalTo(bar));
  - specify(actual, is.not.expected)
  - foo.should eq bar

Nth session

- only immutable objects