Coderetreat

by Coders TUG

Matteo Baglini
Freelance Software Developer (and so on...)
Coders TUG & DotNetToscana Co-Founder
@matteobaglini
matteo.baglini@gmail.com

Learning Through Sharing

Coders Tuscany User Group è una ***community di sviluppatori appassionati*** *che credono nel collaborative learning.*

Il nostro ***obiettivo*** *è creare un* ***network di coders*** *che desiderino alimentare la propria passione condividendo le proprie conoscenze ed esperienze, sperimentando,* ***imparando insieme****.*

CONOSCIAMOCI

Come ti chiami?

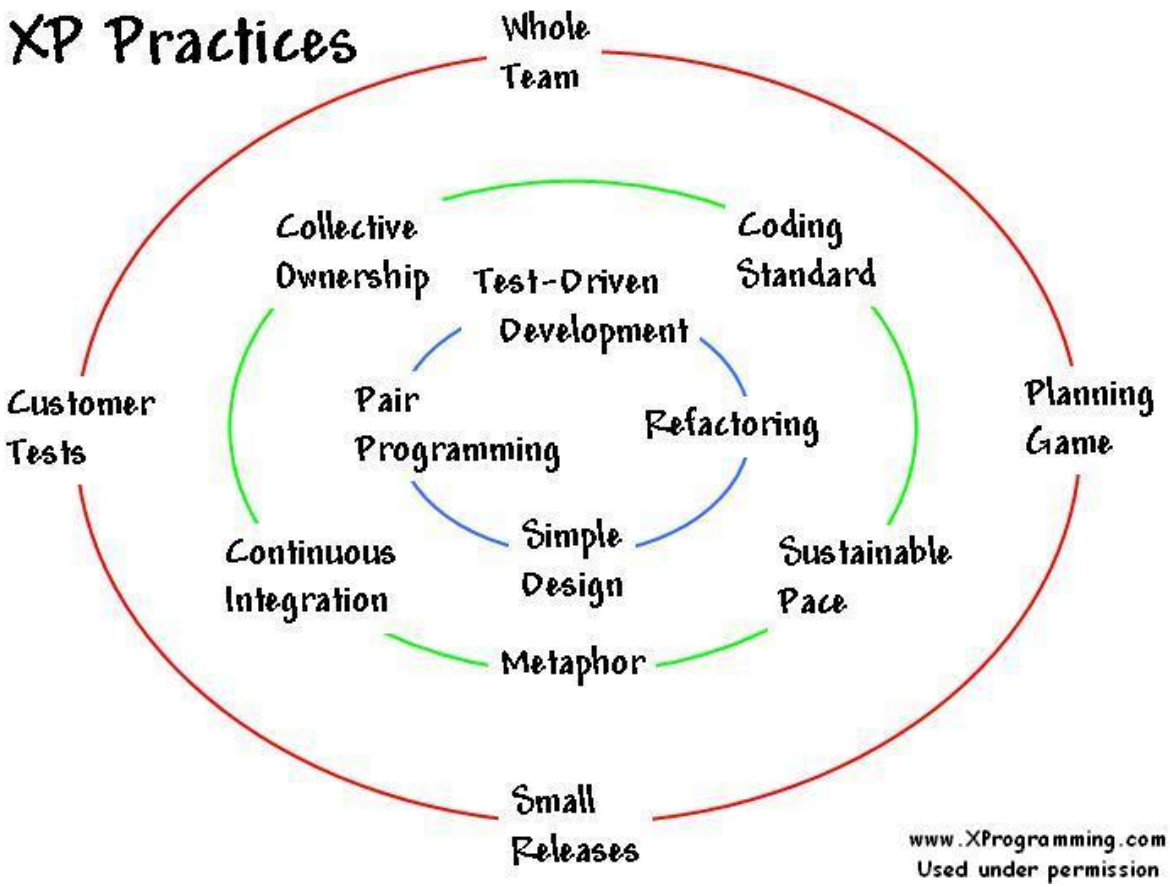
Da dove vieni?

Che lavoro fai?

Obiettivi della giornata?

Test-Driven Development

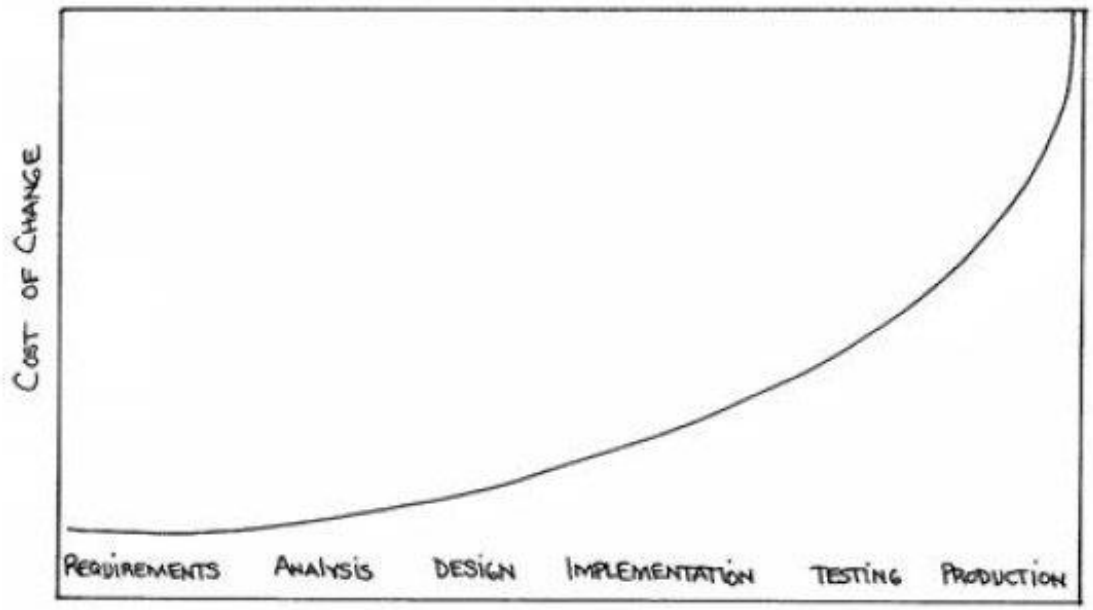
Extreme Programming

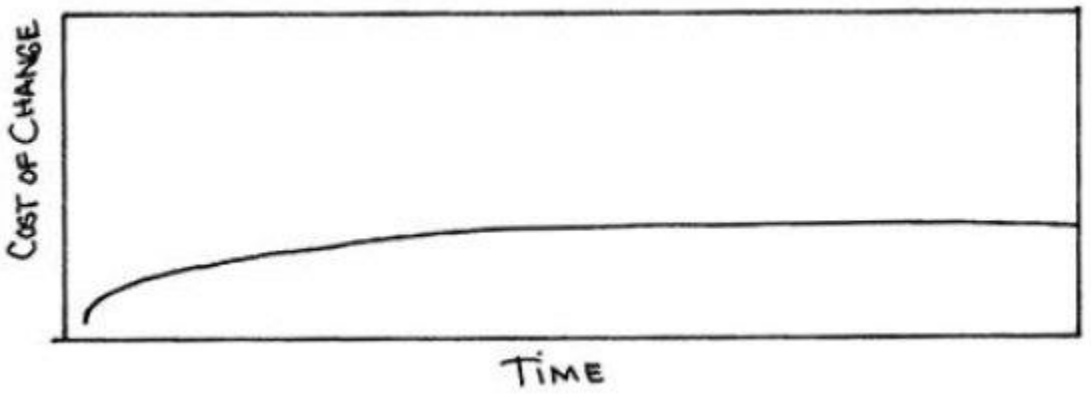


Cost of Change

Barry Boehm

Kent Beck





Clean code that works

Simple Design

The team keeps the design exactly suited for the current functionality of the system.

1. Passes all the tests
2. Contains no duplication
3. Express developer intent
4. Contains as little code as possible

(In this order)

“TDD doesn't drive good design. TDD gives you immediate feedback about what is likely to be bad design. If a test is hard to write, if a test is non-deterministic, if a test is slow, then something is wrong with the design.”

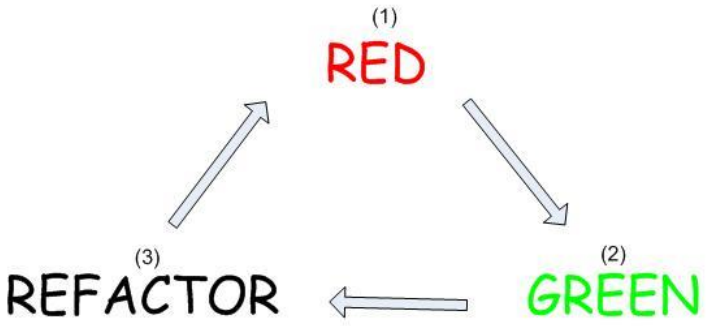
Kent Beck

Mechanics

TDD Rules

- Write new code only if an automated test has failed
- Eliminate duplication

TDD Phases



1. Write a test.
2. Make it compile.
3. Run it to see that it fails.
4. Make it run.
5. Remove duplication.

The **different phases** have **different purposes**. They call for **different styles of solution**, different aesthetic viewpoints. **The first three phases need to go by quickly**, so we get to a known state with the new functionality. We can commit any number of sins to get there, because speed trumps design, just for that brief moment.

Now I'm worried. I've given you **a license to abandon all the principles of good design**. *[cut]* The cycle is not complete. A four-legged Aeron chair falls over. **The first four steps of the cycle won't work without the fifth. Good design at good times. Make it run, make it right.**

There, I feel better. **Now I'm sure you won't show anyone except your partner your code until you've removed the duplication**.

Kent Beck, Test-Driven Development by Example

Clean code that works

Make it run, make it right

TDD & EMERGENT DESIGN

MECHANICS

MOTIVATIONS

GOALS

RUNNING TESTED FEATURES

RED — ALWAYS SEE IT FAIL!

GREEN — EVERYTHING IS ADMITTED, EVEN BLACK MAGIC

REFACTOR — HERE THE MAGIC HAPPENS!

→ HOW TO CHOOSE THE NEXT TEST? IT SHOULD...

1. TAKE 5/15 MIN FROM TEST TO PASS OTHERWISE IT'S TOO BIG
2. MAKE ME PROGRESS TOWARDS A BETTER UNDERSTANDING OF THE SYSTEM
3. FORCES A REFACTORING WHICH SOLVES A SMELL

POSSIBLE CHOICES

1. NO TESTS & NO DESIGN ↓ LOTS OF BUGS — JUST HOPE YOUR CODE DOESN'T NEED TO EVOLVE
2. NO TESTS, DESIGN A LITTLE BIT AT A TIME — DIFFICULT TO FOLLOW WHEN THE CODEBASE GETS BIGGER — LEGACY CODE ACCUMULATE. YOU'RE NOT ABLE TO CHANGE IT FAST ENOUGH ↓ OFTEN LEAD TO BIG REWRITES
3. USE TDD. DESIGN EMERGE TEST AFTER TEST → HOW? THE SAFETY NET BUILT INTO TESTS GIVES MORE CONFIDENCE WHICH LEADS TO MORE AMBITIOUS REFACTORINGS → WHICH KEEPS THE DESIGN FLUID AND ABLE TO CHANGE

IT ALSO SCALES NICELY ON BIG CODEBASES ↑ THE POWER OF TDD COMES FROM ITS LAST STEP: REFACTORING ↑

SIMPLE DESIGN →

1. ALL TESTS RUN
2. NO DUPLICATION
3. EXPRESS DEVELOPER INTENT
4. USE THE MINIMUM NUMBER OF CLASSES AND FUNCTIONS

↓

SIMPLE != EASY

CLEAN CODE THAT WORKS

↓

DIFFICULT TO ACHIEVE AT THE SAME TIME EVEN FOR THE BEST PROGRAMMERS

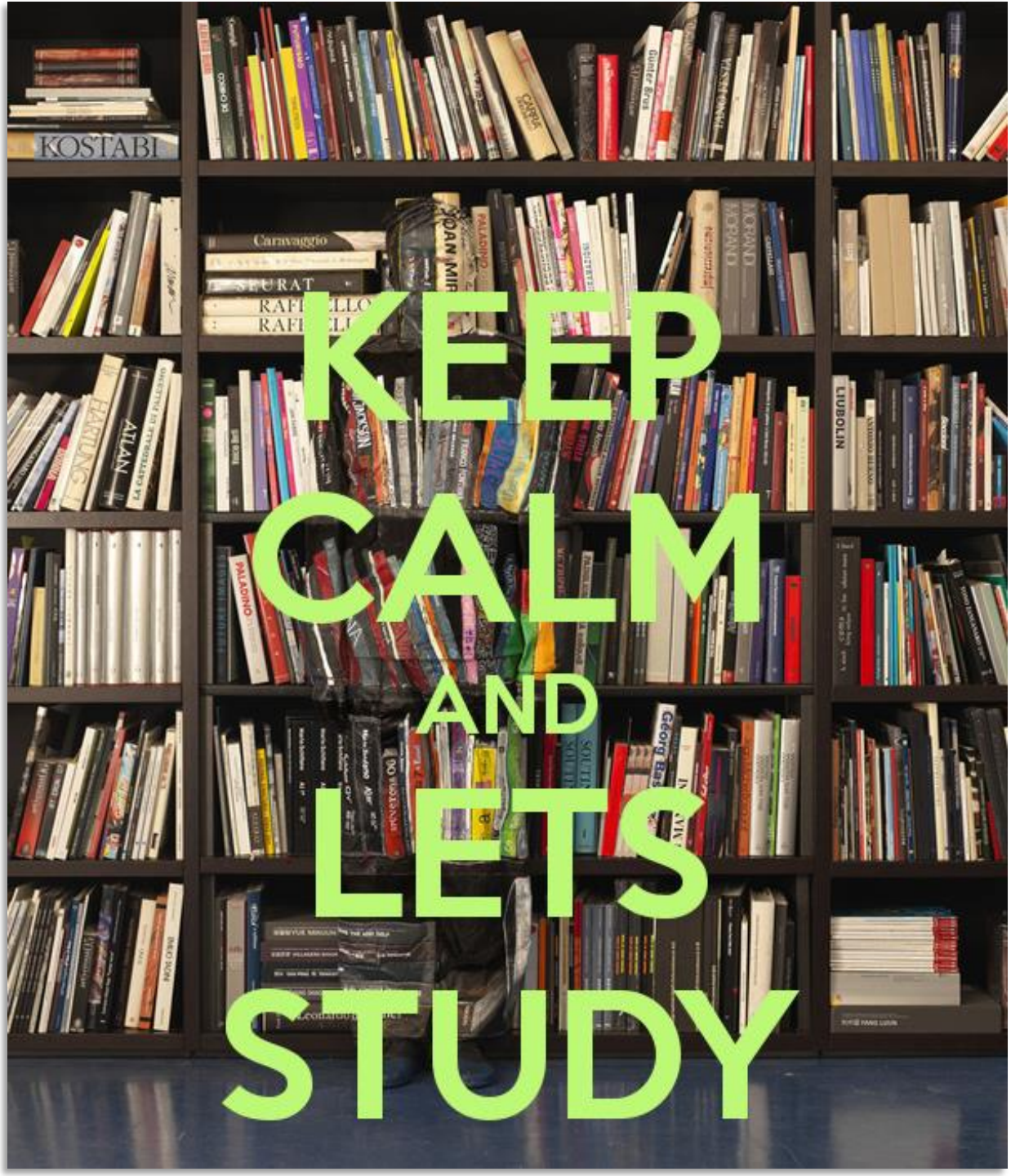
↓

TDD MECHANICS IMPOSE US TO SPLIT THE CODING PHASE IN TWO PARTS

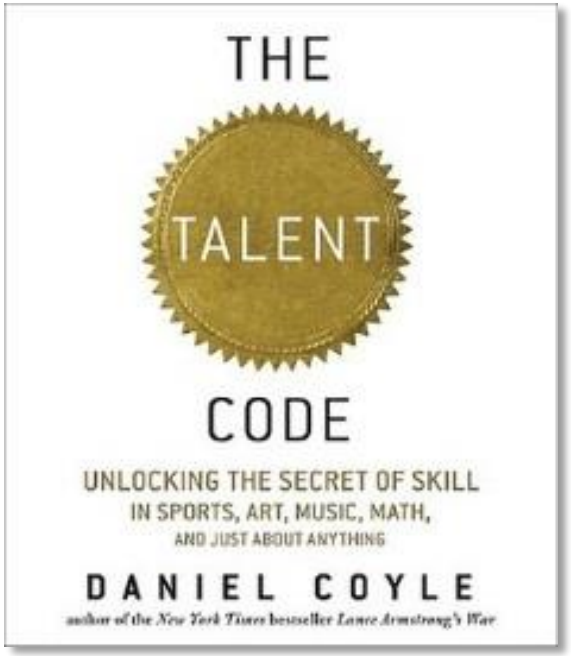
1. MAKE THE CODE WORKS — GO GREEN
2. MAKE THE CODE CLEAN — GO REFACTOR

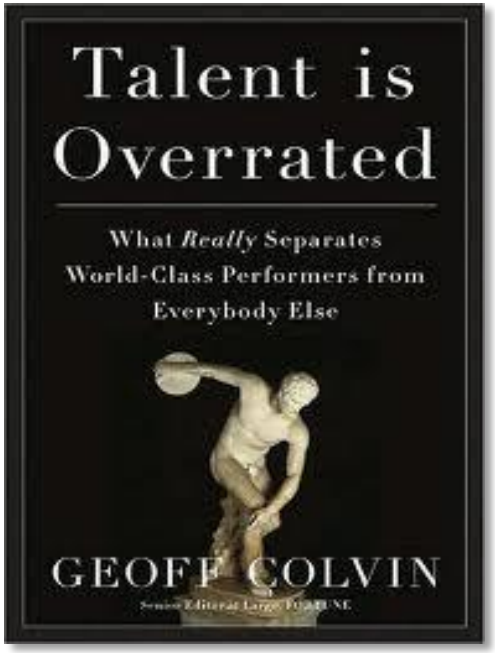
@iacoware

Learn TDD


KOSTABI
Caravaggio
SEURAT
PALADINO
LIUBOLIN
KEEP
CALM
AND
LETS
STUDY

It's all about Deliberate Practice





The secret is life-long period of deliberate effort to improve performance in a specific domain. The secret is what researcher calls ***Deliberate Practice****.*

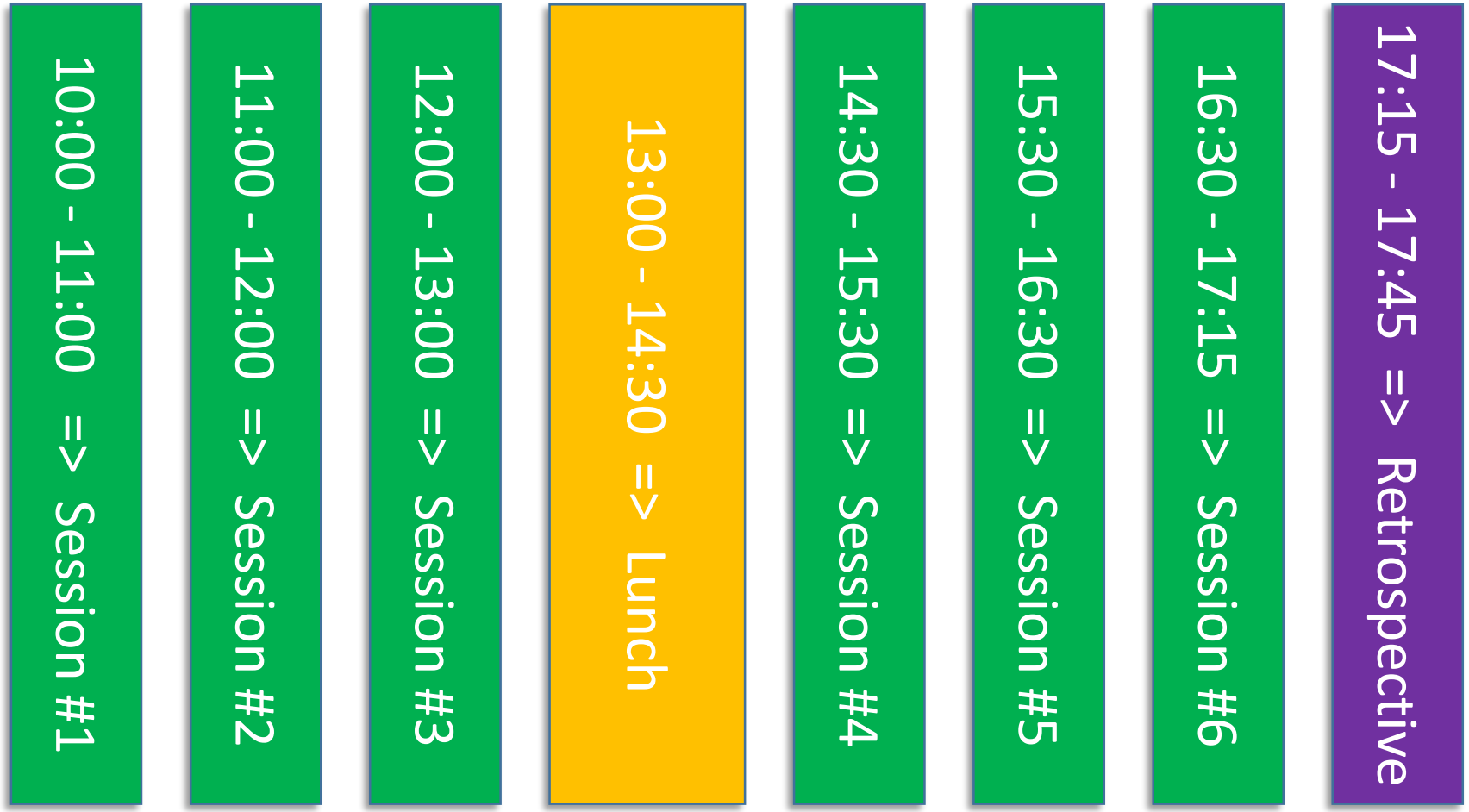

STRESS


Law of Demeter
Polymorphism
Small Tupo


RELAX


EepyBird.com
Entertainment
Diet Coke

Structure of the day



Structure of session

45’ Coding

10’ Retrospective

5’ Break

RULES!

1. You SHALL!
2. You WILL!
3. You MUST!

Pair Programming

Test-Driven Development

ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT

Simple Design

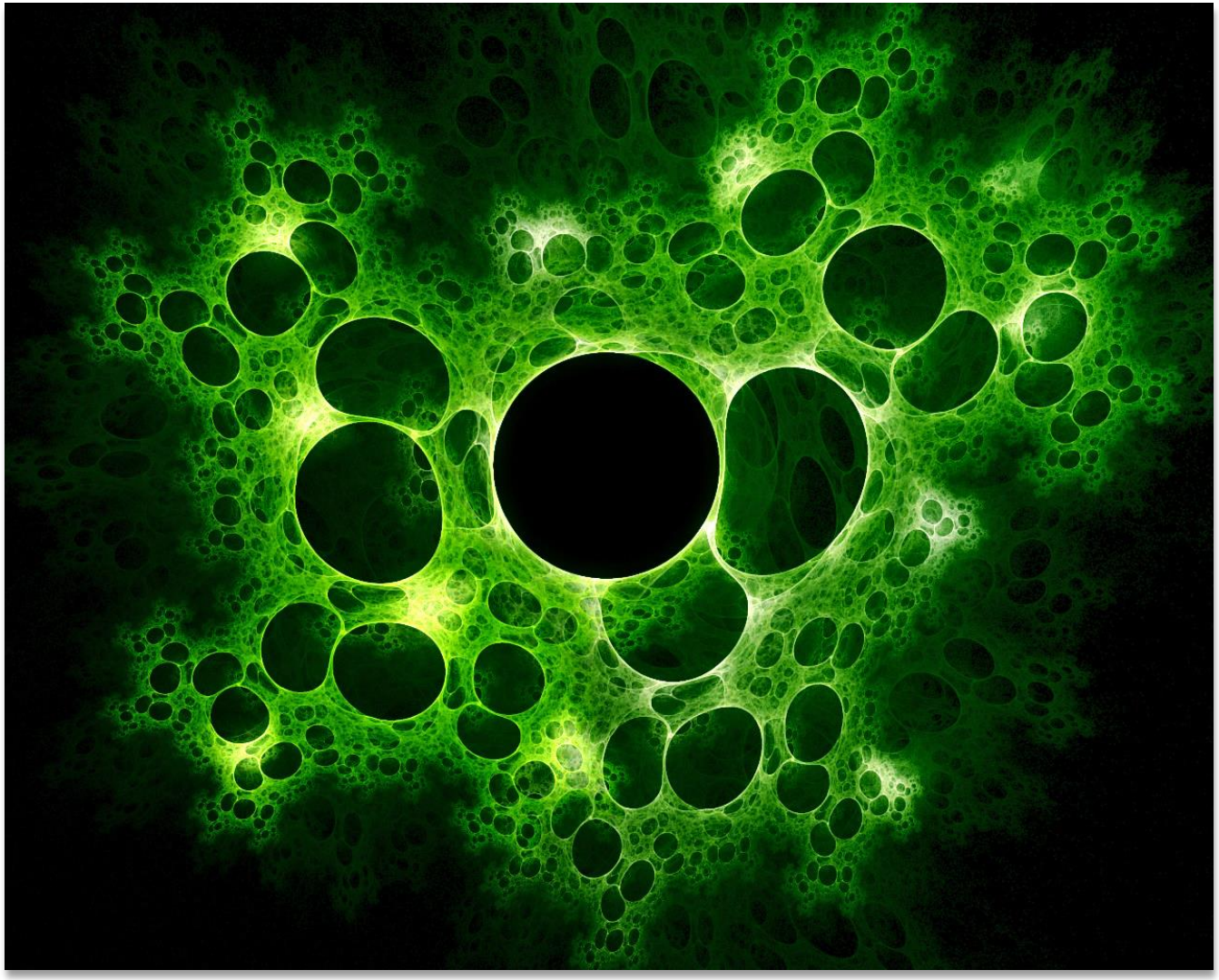
The team keeps the design exactly suited for the current functionality of the system.

1. Passes all the tests
2. Contains no duplication
3. Express developer intent
4. Contains as little code as possible

(In this order)

Change Partner

Delete Your Code



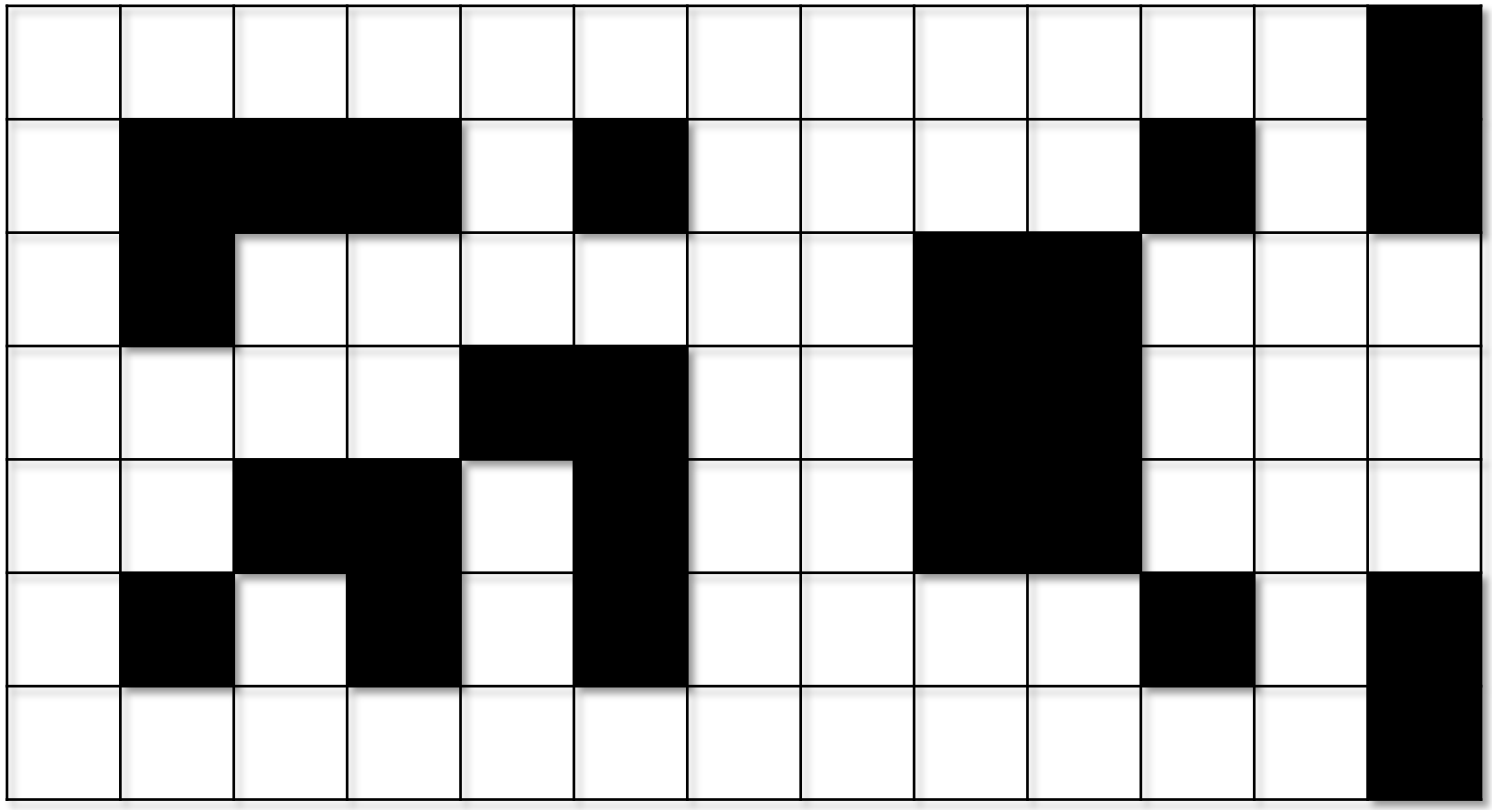
The Game of Life

*The Game of Life is a **cellular automaton** devised by the British mathematician **John Conway** in 1970.*

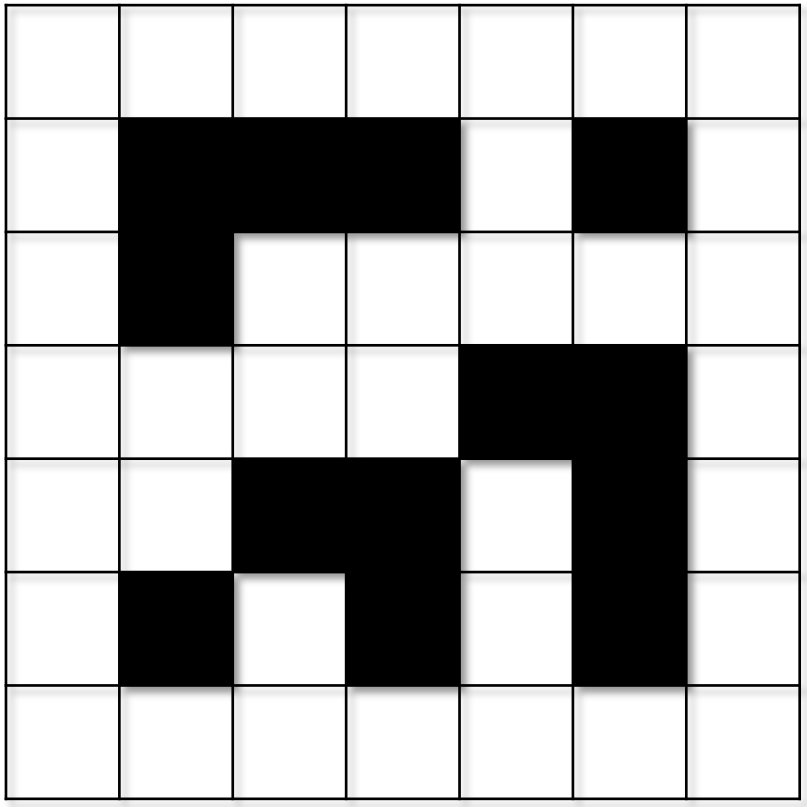
*It's a **zero-player game**.*

*You **create an initial state** and **watch how it evolves**.*

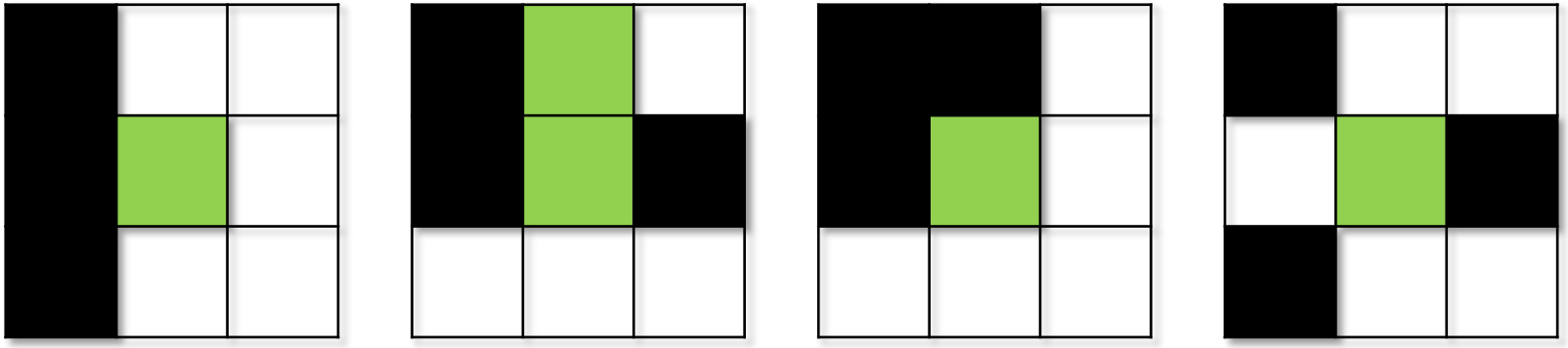
*The **universe** Game of Life is an infinite 2D grid of square **cells**, each of which is in one of two possible states, **alive** or **dead**.*



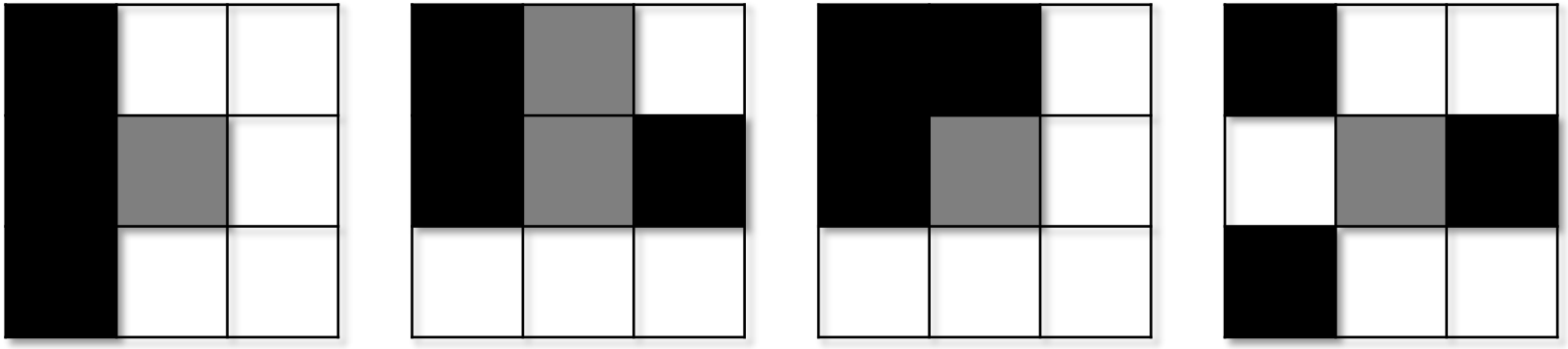
At each **generation**, every cell interacts with its **eight neighbours**, following **three rules**.



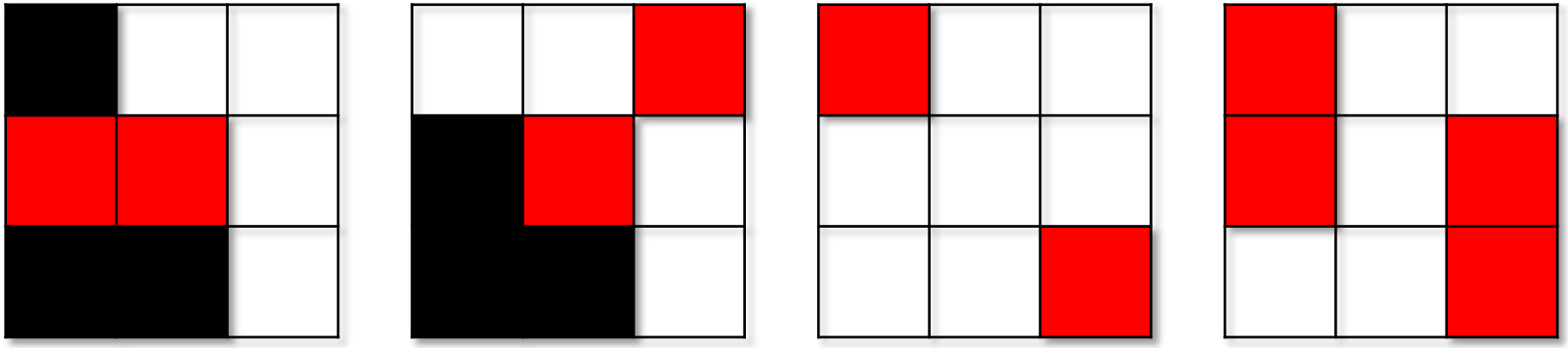
Any **dead** cell with **exactly three live neighbours** becomes **a live** cell, as if by reproduction.



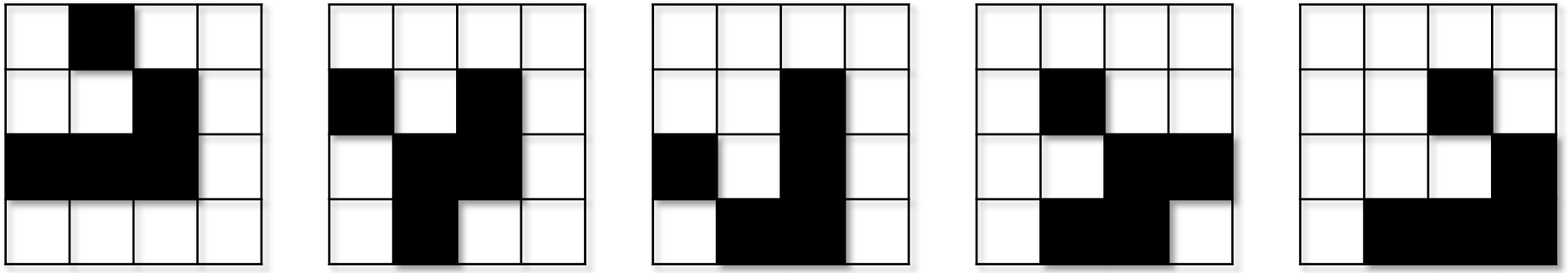
Any live cell with two or three live neighbours lives on to the next generation.



Otherwise, the cell dies from either loneliness or overcrowding.

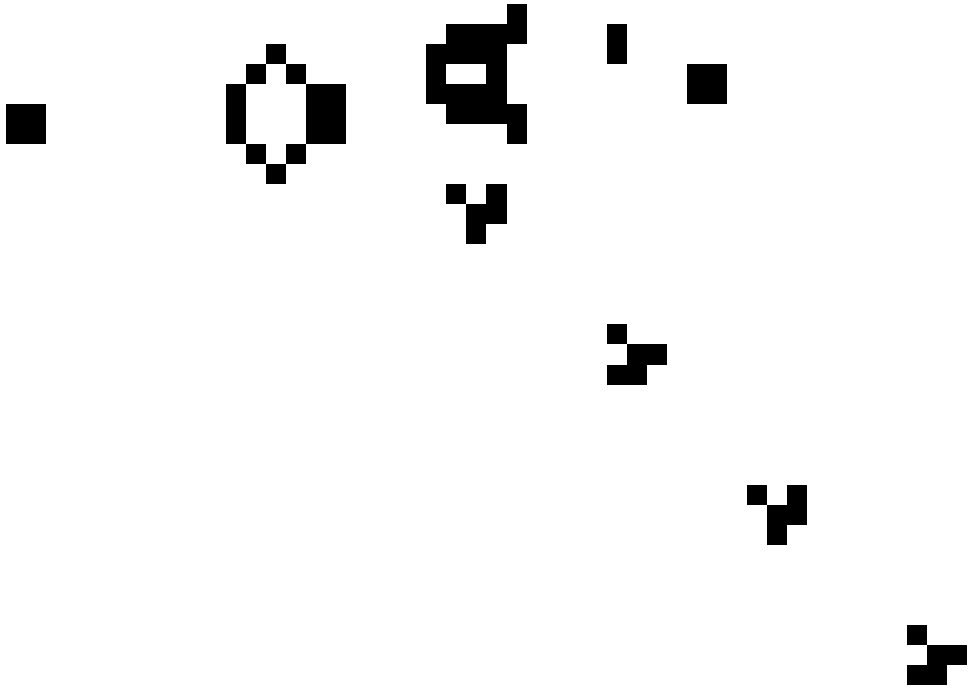


Depending on **how it's seeded**, the game board **exhibits** **remarkable**, very lifelike, **behavior**. Like a **glider**.



The Game of Life demonstrates emergent behavior.

The behavior of the system as a whole can't be predicted solely by looking at the behavior of the single objects that comprise the system.




{let's code
together;}