# HAVING A WORKING SOFTWARE IS NOT ENOUGH

## THOUGHTS ABOUT SOFTWARE DESIGN

Lic. Yanet Morales

SG VIRTUAL CONFERENCE

# ABOUT ME

**01.** Bachelor degree in Computer Science

**02.** Developer, Scrum Master, Product Owner

**03.** Technical Delivery Manager at Improving Mexico

**04.** PSM II, PSPO II, PSK I, PSU I, SPS, PSFS

**yanet.morales@improving.com**
https://www.linkedin.com/in/yanetmoralesr


# AGENDA

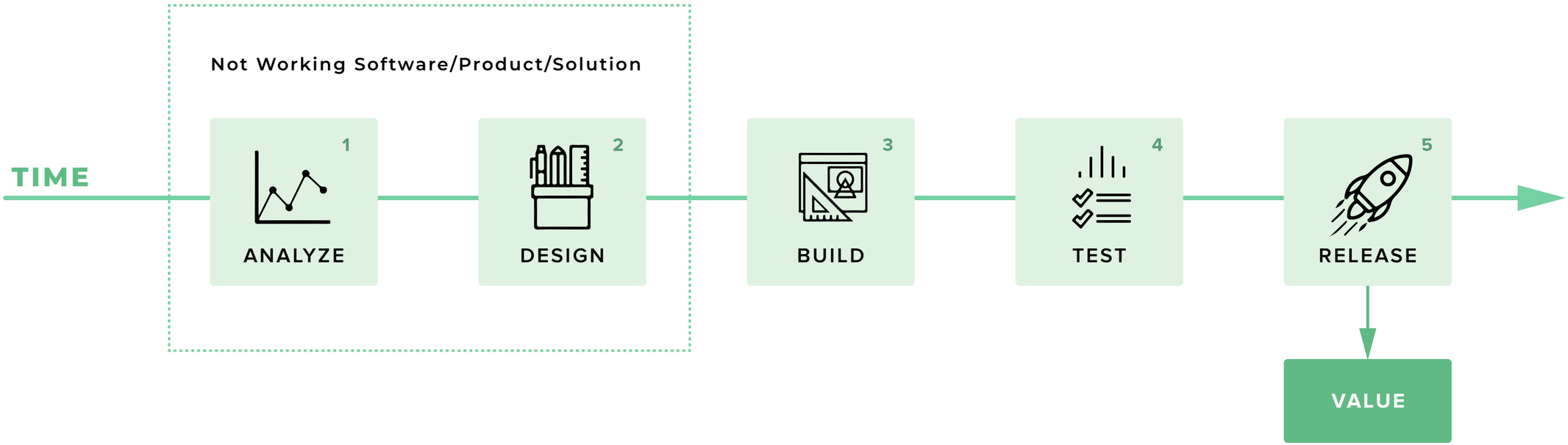
WHAT IS AGILE?

CODEBASE ANALYSIS

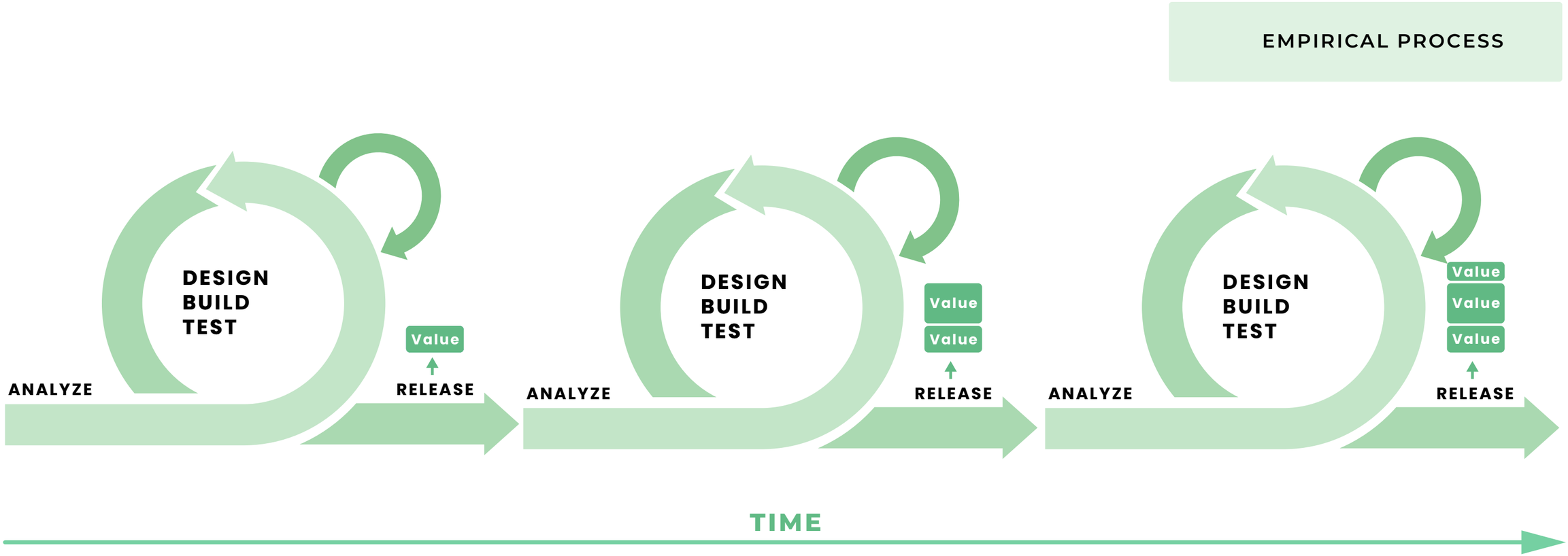
CODEBASE QUALITY

# 1 WHAT IS AGILE?

# TRADITIONAL APPROACH



# AGILE APPROACH



# HOW DOES YOUR CODEBASE LOOK?

High code complexity?

Business logic in the wrong places?

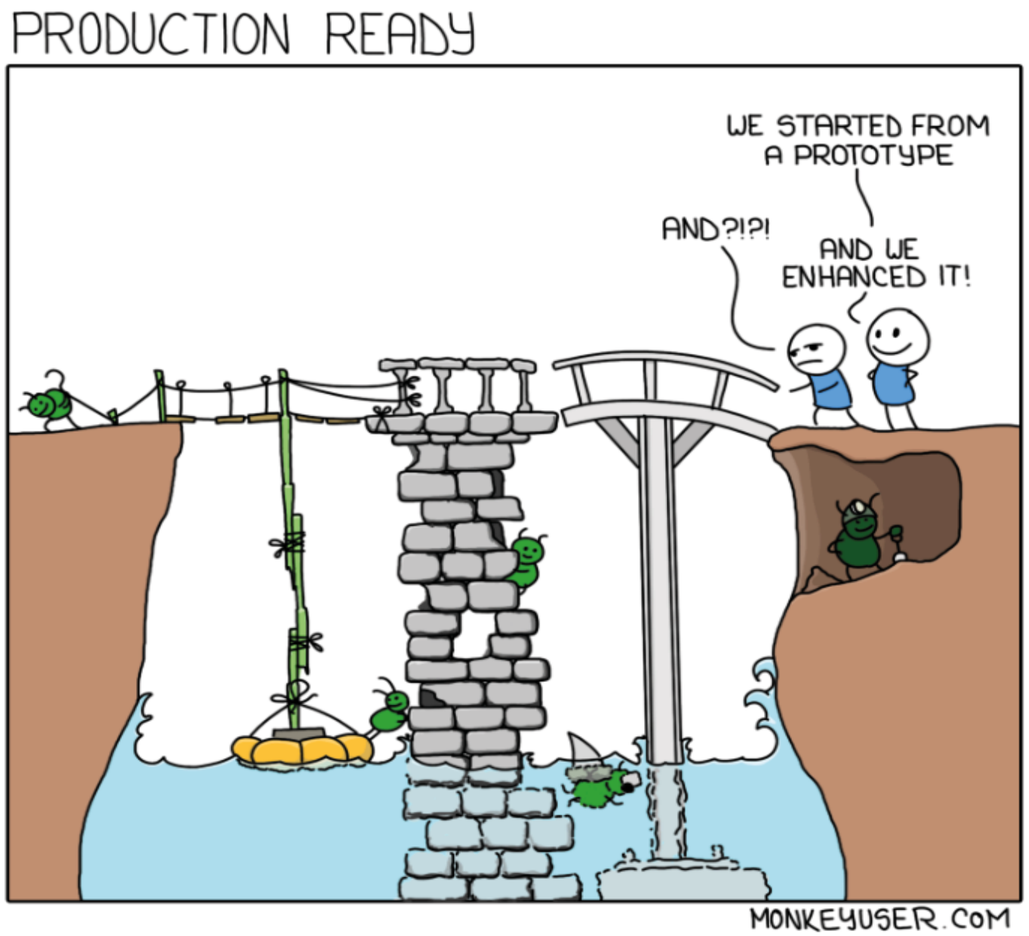
Unreadable names or algorithms?

Highly coupled code?

Combination of legacy and new applications?

**Having a working software is not enough!**

**Responding rapid to change is not enough!**



# RAISING THE BAR

| | AGILE MANIFESTO (2001) | MANIFESTO FOR SOFTWARE CRAFTSMANSHIP (2009) |
|---|---|---|
| Processes and tools | **Individuals and interactions** | **A community of professionals** |
| Comprehensive documentation | **Working software** | **Well crafted software** |
| Contract negotiation | **Customer collaboration** | **Productive partnerships** |
| Following a plan | **Responding to change** | **Steadily adding value** |

# 2 CODEBASE ANALYSIS

# IS YOUR CODE COMPLEX?

"Complexity is anything related to the structure of a software system that makes it **hard to understand and modify**"

**It takes a lot of work** to implement even small improvements.

The incremental nature of complexity makes it **hard to control**

Complexity makes it **difficult and risky to modify an existing code base**

**Symptoms of code complexity:**

**Change amplification:**
A simple change requires code modifications in many places.

**Cognitive load:**
How much a developer needs to know about the code to complete a task?

**Unknown unknows:**
There is something you need to know, but there is no way for you to find out what it is, or even, whether there is an issue.


John Ousterhout, A philosophy of Software Design, 2018

# WHAT IS THE SMELL OF YOUR CODEBASE?

Smells (especially code smells) are warning signs about potential problems in code. Not all smells indicate a problem, but most are worthy of a look and a decision.

## SMELLS WITHIN CLASSES

**Measured smells**
- Comments
- Long method
- Large class
- Long parameter list

**Names**
- Type embedded in name
- Uncommunicative name
- Inconsistent names

**Unnecessary complexity**
- Dead code
- Speculative generality

**Duplication**
- Magic numbers
- Duplicated code
- Alternative classes with different interfaces.

**Conditional logic**
- Null check
- Special case
- Complicated boolean expression
- Simulated inheritance (Switch statement)

## SMELLS BETWEEN CLASSES

**Data**
- Primitive obsession
- Data class
- Data clump
- Temporary field

**Inheritance**
- Refused bequest
- Lazy class
- Inappropriate intimacy (Subclass Form)

**Responsibility**
- Feature envy
- Middle man
- Message chains
- Inappropriate intimacy (General Form)

**Accommodating change**
- Divergent change
- Shotgun surgery
- Parallel inheritance hierarchies
- Combinatorial explosion

**Library classes**
- Incomplete library class

William C. Wake, Refactoring Workbook, 2003

# TECHNICAL DEBT: IT'S NOT JUST ABOUT CODING

## SOFTWARE DESIGN

**01.** High code complexity

**02.** Business logic in the wrong places

**03.** Unreadable names or algorithms

**04.** Highly coupled code

**05.** Combination of legacy and new applications

## TECHNICAL PRACTICES

**01.** Improper software customization

**02.** Releases with lots of bugs

**03.** Lack of unit tests

**04.** Lack of automated deployment

**05.** Lack of automated build

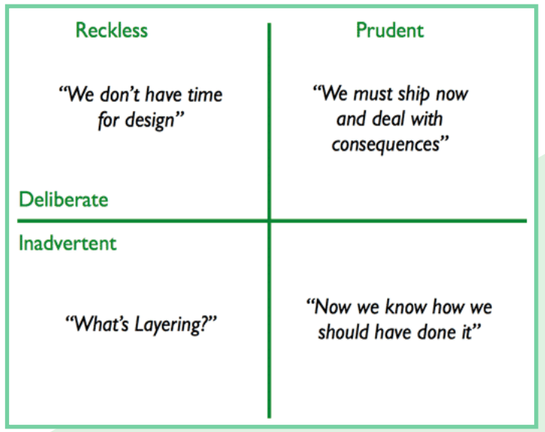
## BUSINESS DECISIONS

**01.** Prioritization of product constraints, like deadlines, over code design

**02.** To many steps in the process

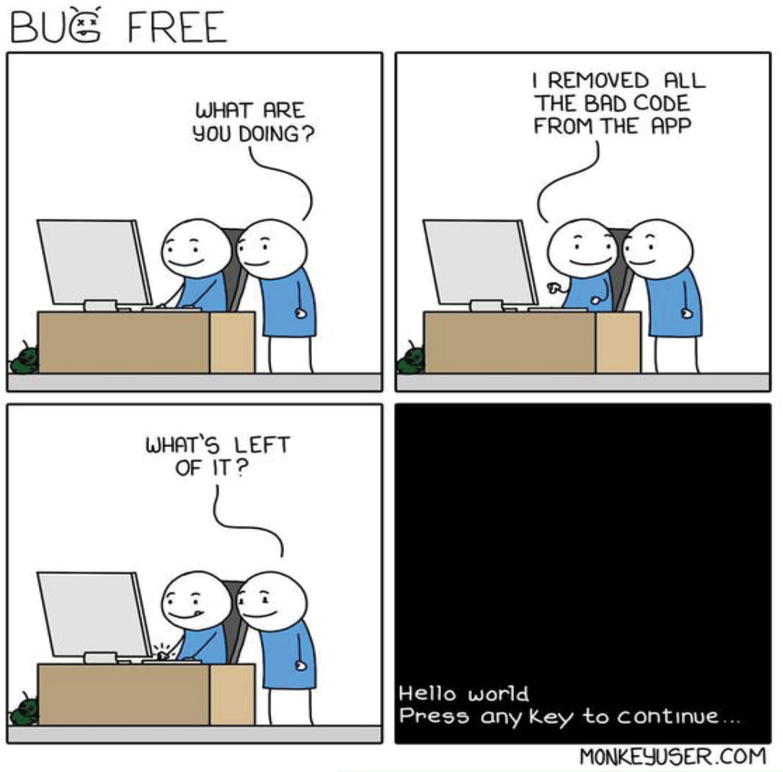
**03.** Manual approval steps that are not reviewed

**04.** High business logic complexity




Martin Fowler, TechDebt Quadrant

# HOW DO YOU KNOW THAT THE CODE YOU ARE WRITING HAS GOOD QUALITY?



# SAMPLE

javascript

```javascript
function addNumbers(num1, num2) {
  if (isNaN(num1) || isNaN(num2)) {
    return "Invalid input";
  } else {
    let sum = num1 + num2;
    if (sum > 100) {
      return "Result is too large";
    } else {
      return sum;
    }
  }
}
```

javascript

```javascript
function addNumbers(num1, num2) {
  if (!isNumeric(num1) || !isNumeric(num2)) {
    throw new Error("Invalid input");
  }

  const sum = num1 + num2;

  if (sum > 100) {
    throw new Error("Result is too large");
  }

  return sum;
}

function isNumeric(num) {
  return !isNaN(parseFloat(num)) && isFinite(num);
}
```

# 3 CODEBASE QUALITY

# SOME WORDS ABOUT SOFTWARE DESIGN...

## Software Design

- Readability
- Modularity
- Portability
- Scalability
- Maintainability
- Compatibility
- Fault-tolerance
- Robustness
- Extensibility
- Reusability
- Security
- Performance
- Usability
- Reliability

**Software Architecture**

Decisions that are hard to be changed later
High cost
High risk

Decisions regarding with code structure

# REMOVE THE SMELL OF YOUR CODE

## Code Smells

***Simple Smells*** *(magic numbers, long parameter list, etc)*

***Composite Smells*** *(God Class/Large Class, Blobs, etc)*

***Approaches to Detect*** *(metrics, strategies/rules, etc)*

***Support Tools***

## Quality

***Technical Debt*** *Management, Prioritizing, Mitigate...*

***External Quality Attributes*** *Maintainability, Testability, Functionality...*

***Internal Quality Attributes*** *Cohesion, Coupling, Size...*

## Refactorings

***Primitive Refactorings*** *(renames, extracts, etc)*

***Composite Refactorings*** *(Replace conditional with, etc)*

***Approaches to Opportunities*** *(metrics, strategies/rules, search, etc)*

***Support Tools***

THE REFACTORING CYCLE

Start with a working program.

While smells remain:

- Choose the worst smell.
- Select a refactoring that will address the smell.
- Apply the refactoring.

Lacerda, Guilherme & Petrillo, Fabio & Pimenta, Marcelo & Guéhéneuc, Yann-Gaël. (2020). Code Smells and Refactoring: A Tertiary Systematic Review of Challenges and Observations. Journal of Systems and Software. 167. 110610. 10.1016/j.jss.2020.110610.

# CLEAN YOUR CODE

## DESIGN PATTERNS

General repeatable solution to a commonly occurring problem in software design. Description or template for how to solve a problem that can be used in many different situations.

| Creational | Structural | Behavioral |
|---|---|---|
| Singleton | Adapter | Template |
| Factory | Bridge | Mediator |
| Prototype | Decorator | Observer |
| Builder | Composite | Strategy |
| | Proxy | Iterator |

## DESIGN PRINCIPLES

Set of guidelines that helps developers to make a good system design.

**Composition over inheritance - Encapsulate what varies - SOLID**

**YAGNI (You ain't gonna need it) Principle - DRY**

## CLEAN CODE

Name, construct, structure, style, readability

**Meaningfull name - Keep methods, classes, files small**

**Consistency - Correct constructs - Refactor often - Code Style**

# CREATE A PLAN TO START PAYING BACK YOUR DEBT!

### SHORT-TERM GOALS (1 SPRINT OR LESS)

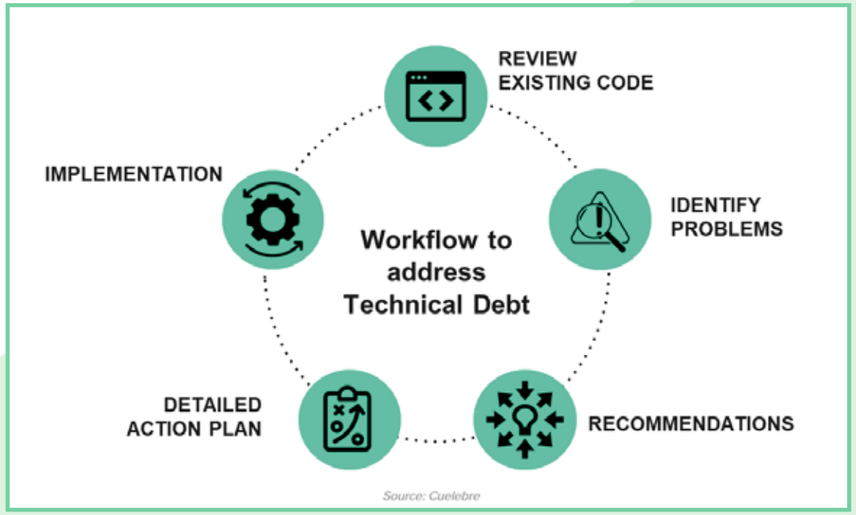
- **01.** Code review with pull request
- **02.** Static code analysis
- **03.** Following naming conventions
- **04.** Code refactoring

### MID-TERM GOALS (MULTIPLE SPRINTS)

- **01.** Setting up a CI and CD approach
- **02.** Implementing a coding guideline
- **03.** Removing duplicate code
- **04.** Implement test automation

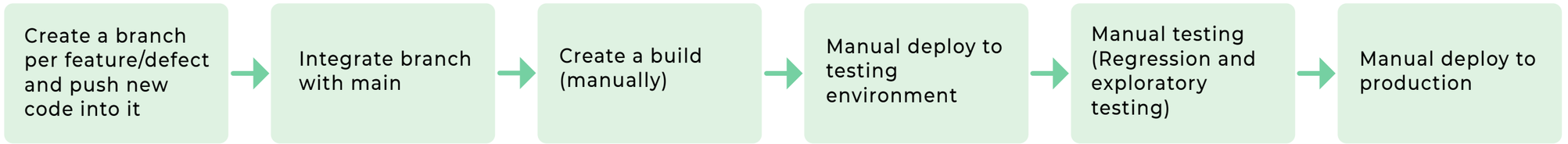
### LONG-TERM GOALS (LONG PERIOD TIME, YEARS)

- **01.** Adopting new technologies

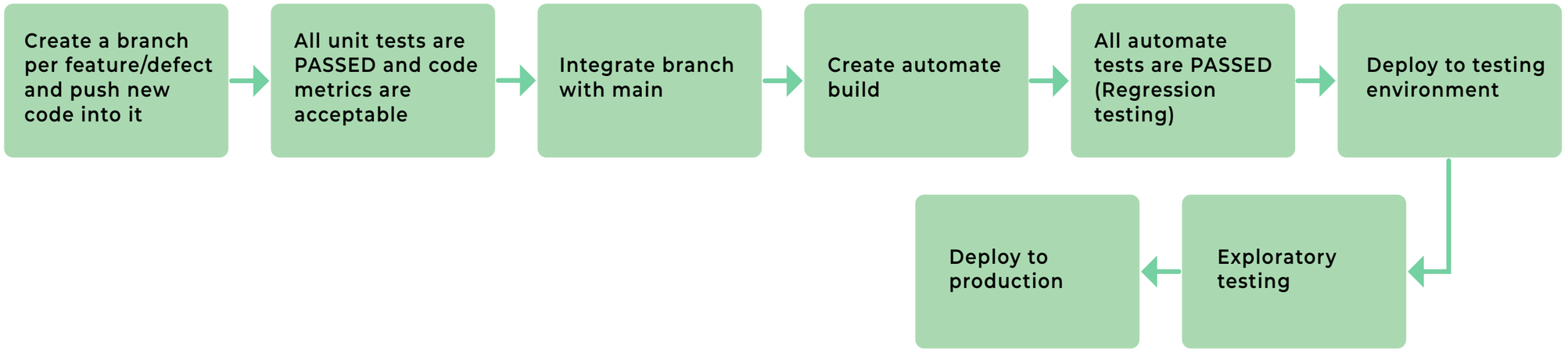


# IMPROVE QUALITY IN YOUR DELIVERY PROCESS

DEFINITION OF DONE

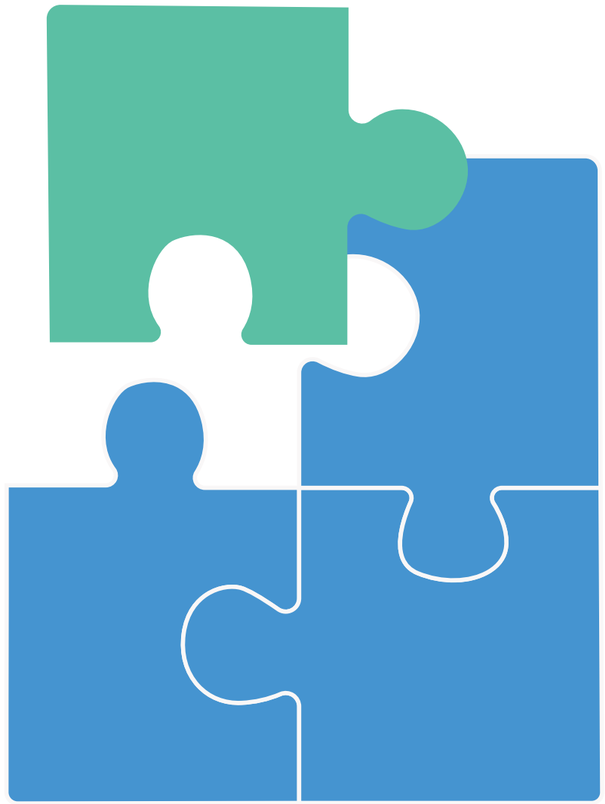



DEFINITION OF DONE



# IMPROVE TECHNICAL PRACTICES

- Definition of Done
- Test Driven Development
- CI / CD
- Pair/Mob Programming
- Agile Testing



- Monitoring
- Code Reviews
- DevOps
- Build/Release pipelines
- Code Metrics

# SOME TOOLS YOU CAN CHECK...

**Static code analyzers** (look for common mistakes, catch syntax errors in non-compiled languages, identify code smells)

**Code style checkers** (ensure all code is formatted in the same way)

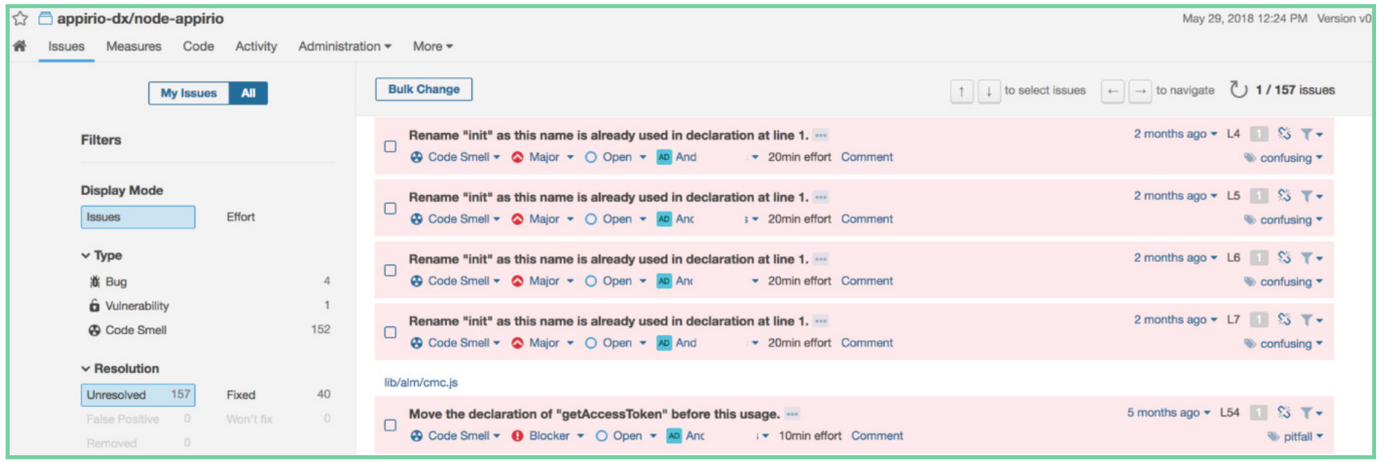
**Code complexity tools** (guard against overly complex logic by calculating cyclomatic complexity)

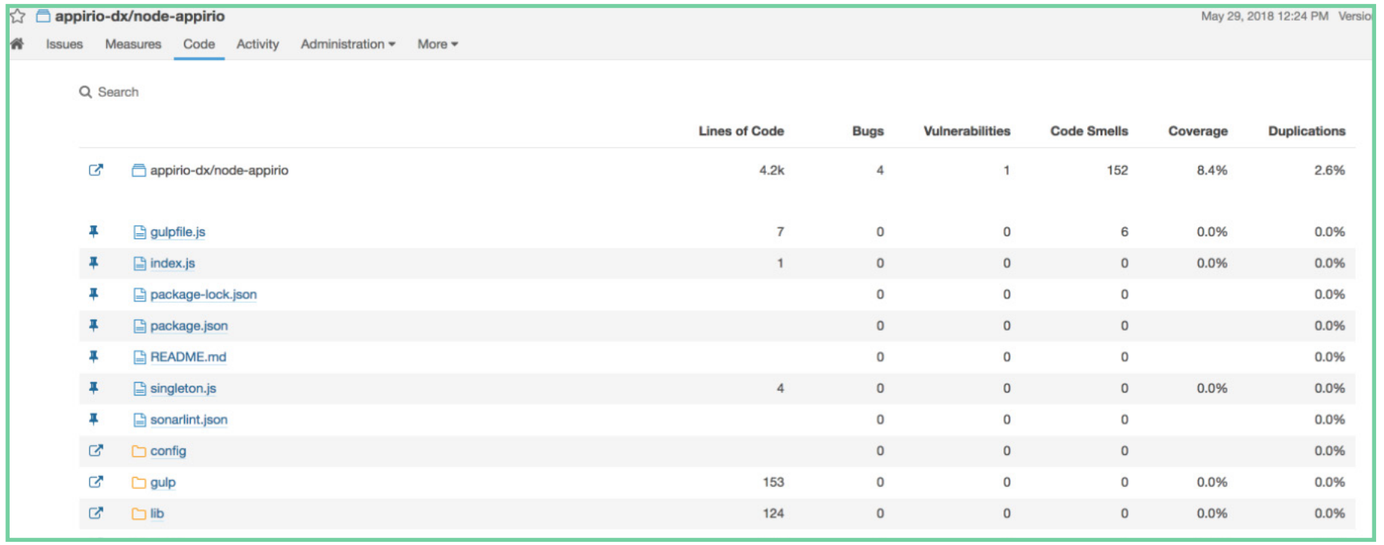
**Code coverage tools** (measure how many lines of code were exercised by the test suite)

SONARQUBE SONARLINT

CODACY ESLINT

DEEPSOURCE



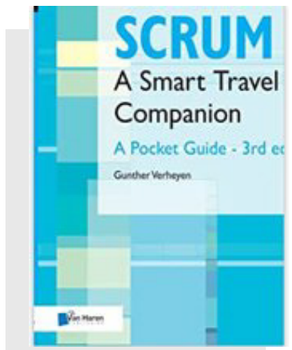



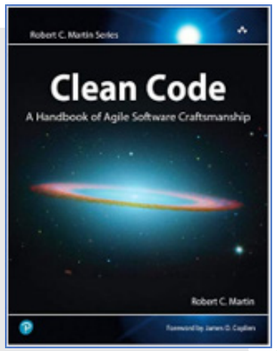
# 4

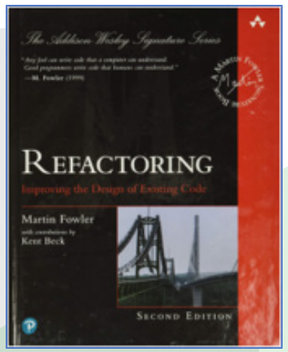
## WHAT WOULD YOU DO WITH WHAT YOU LEARNED TODAY?

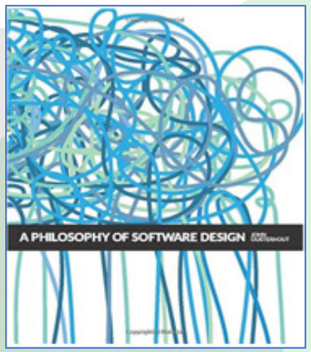
# NEXT STEPS

- Read the Scrum Guide
- Improve your technical skills
- Participate in webinars related with Agile topics
- Read some books

# ¿QUESTIONS?

CONNECT WITH ME

**YANET MORALES**
yanet.morales@improving.com
https://www.linkedin.com/in/yanetmoralesr

SG VIRTUAL CONFERENCE