# CODE QUALITY

CODE ENGINE
STUDIO


QUALITY CONTROL
APPROVED

# What is clean code?

How do you define good quality code?

# What is Clean Code?

*I like my code to be **elegant and efficient**. Clean code does one thing well*

Bjarne Stroustrup, inventor of C++

*Clean code always looks like it was written by someone who **cares***

Michael Feathers

*If you want your code to be easy to write, make it **easy to read***

Robert C. Martin, Co-author of Agile Manifesto

# Why should you care?

# 1. Personally


Excellent
Good
Average
Poor
Very Poor

*Good enough,*
*is **not** good enough.*


EXCELLENCE

# 2. Cost

# Cost of fixing bugs

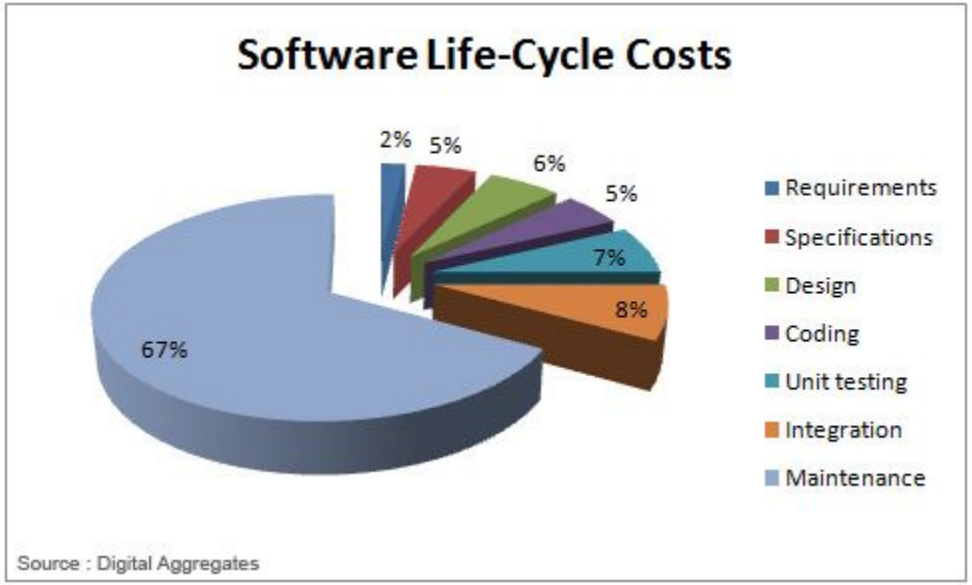
| Design and architecture | Implementation | Integration testing | Customer beta test | Postproduct release |
|---|---|---|---|---|
| 1X* | 5X | 10X | 15X | 30X |

*X is a normalized unit of cost and can be expressed in terms of person-hours, dollars, etc.
Source: National Institute of Standards and Technology (NIST)†

*By catching defects as early as possible in the development cycle, you can significantly reduce your development costs.*

# 3. Time

# Software life-cycle



# 4. Professionally

# Why should you care?

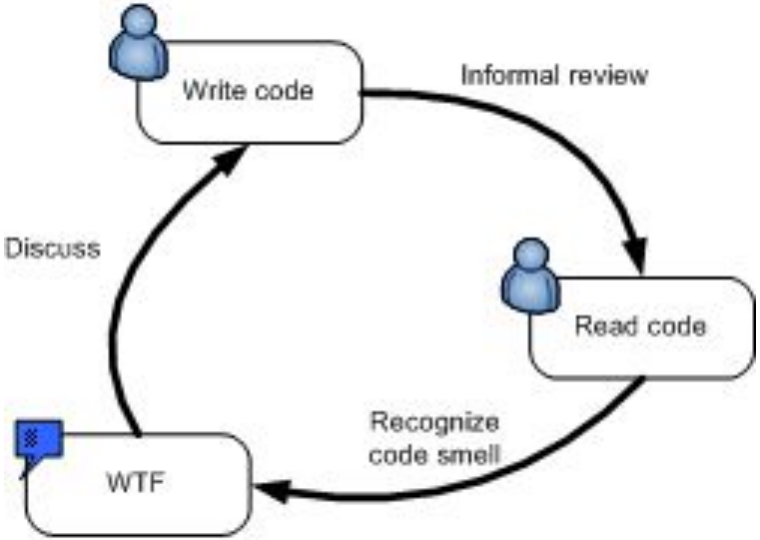
**For yourself**

Dev time = 60% reading and 40% writing

Easier to fix bugs

Easier to estimate new features

Easier to maintain



# Why should you care?

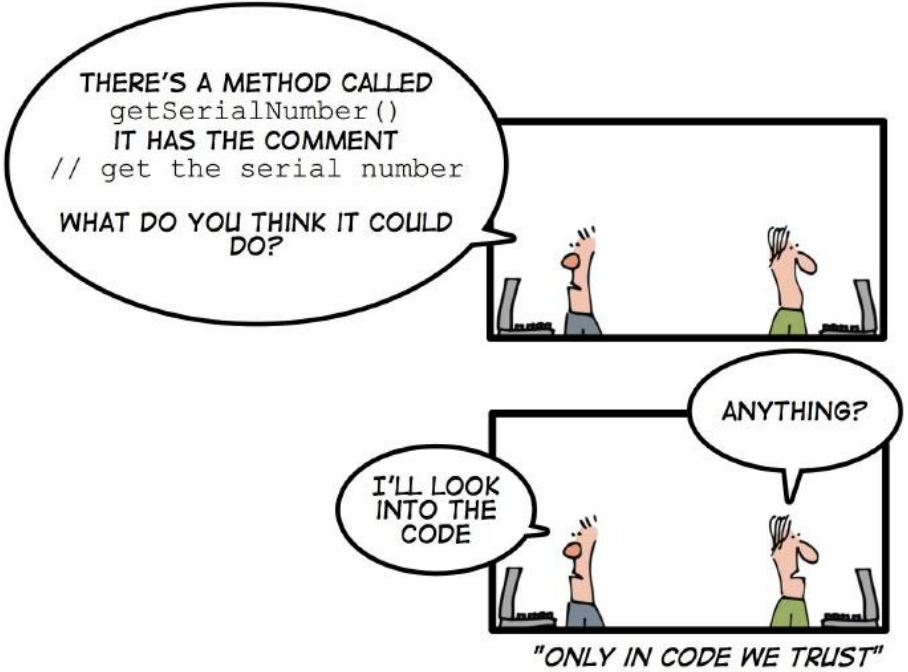
**For other co-workers**

New member joins

Maintenance

Hand-over

Making the life easier for everyone involved in the project




http://geekandpoke.typepad.com


CODING IS NOT A SPRINT,
IT'S A MARATHON.

# How to get started.


LONG, HARD
JOURNEY.

# Rule 1. Follow the style guide

# Rule 1. Follow the style guide

## What is it?

**Coding** conventions are a set of **guidelines** for a specific programming language that recommend programming **style**, practices, and methods for each aspect of a program written in that language.

**Coding** conventions are only applicable to the human maintainers and peer reviewers of a software project.

Wikipedia

# Rule 1. Follow the style guide

1. Read the guide carefully
2. Learn the basics by heart
3. Look up corner cases
4. Apply the rules religiously

Result: your programs will be better than those written by the majority of university graduates.

# Rule 1. Follow the style guide

Many resources out there.

***Examples***

1. Java: Google Java style guide
   https://google.github.io/styleguide/javaguide.html
2. C#: Microsoft coding conventions
   https://docs.microsoft.com/en-us/dotnet/csharp/.../coding-conventions
3. PHP: PSR-2
   http://www.php-fig.org/psr/psr-2/

# Rule 2: Create descriptive names

# Rule 2: Create descriptive names

1. Class and type names should be nouns.
2. Methods names should contain a verb.
3. Function names should describe what the function returns.
4. Use long descriptive names - help you and colleagues understand what the code does.
5. Give accurate names - Did you mean highestPrice, rather than bestPrice?
6. Give specific names - Should it be getBestPrice, rather than getBest?

# Rule 3: Comment and document

**Example**

This is bad:

```
protected $d; // elapsed time in days
```

This is good:

```
protected $elapsedTimeInDays;
```

# Rule 3: Comment and document

**Example**

This is good:

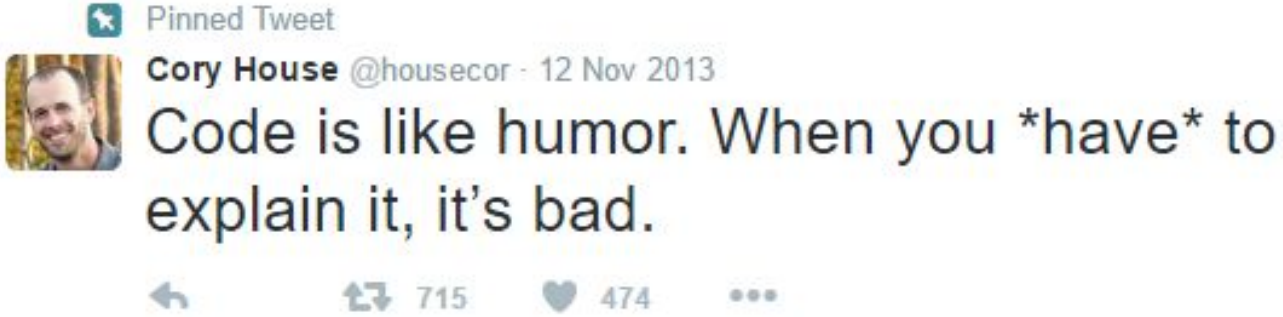
```
class Product {
  private $price;

  public function increasePrice($dollarsToAddToPrice) {
    $this->price += $dollarsToAddToPrice;
  }
}
```

# Rule 3: Comment and document

# Rule 3: Comment and document

**Don’t comment, let code be you comment**

1. Code should explain itself
2. If code is readable you don’t need comments
3. Comments do not make up your code & may contain lies



# Rule 3: Comment and document

**Example**

This is bad:

```
// Check to see if the employee is eligible for full benefits
if ($employee->flags && self::HOURLY_FLAG && $employee->age > 65)
```

This is good:

```
if ($employee->isEligibleForFullBenefits())
```

# Rule 3: Comment and document

**So when do you write a comment?**

1. Explain your intention in comments

For example:

```
// if we sort the array here the logic becomes
Simpler in calculatePayment() method
```

# Rule 3: Comment and document

**So when do you write a comment?**

2. Warn of consequences in comments

For example:

```
// this script will take a very long time to run
```

# Rule 3: Comment and document

**So when do you write a comment?**

3. Emphasize important points in comments

For example:

```
// The trim function is very important, in
most cases the username has a trailing space
```

# Rule 4: Don't repeat yourself

# Rule 4: Don't repeat yourself

**Also known as DRY**

1. Never copy-and-paste code in the same project.
2. Abstract the common parts into a routine or class, with appropriate parameters.

# Rule 5: The smaller the better

# Rule 5: The smaller the better

**Split your Code into Short, Focused Units**

1. A function should only do one thing
2. No nested control structure
3. Less arguments are better (3 or less if possible)

For example:

```
Circle makeCircle(Point center, double radius);
```

Is better than

```
Circle makeCircle(double x, double y, double radius);
```

# Rule 5: The smaller the better

**Split your Code into Short, Focused Units**

3. No side effects - Functions do what the name suggests and nothing else.
4. Avoid output arguments - If returning something is not enough then your function is probably doing more than one thing.

For example:

```
email.addSignature();
```

Is better than

```
addSignature(email);
```

# Rule 5: The smaller the better

**Split your Code into Short, Focused Units**

5. Error Handling is one thing - Throwing exceptions is better than returning different codes dependent on errors.

# Rule 6: Don't overdesign

# Rule 6: Don't overdesign

1. Keep your design focused on today's needs.
2. Your code can be general to accommodate future evolution, but only if that doesn't make it more complex.
3. You can't guess what tomorrow will bring.
4. When the code's structure no longer fits the task, refactoring it to a more appropriate design.

# Rule 7: Other code smells and heuristics

# Rule 7: Other code smells and heuristics

There are a lot more that you can do to identify and avoid bad code. Here is a list of some code smells and anti-patterns to avoid.

1. Dead code
2. Large classes
3. God object - an object that knows too much or does too much.
4. Multiple languages in one file
5. Framework core modifications
6. Magic numbers - replace with const or var

# Rule 7: Other code smells and heuristics

7. Long if conditions - replace with function
8. Call super’s overwritten methods
9. Circular dependency
10. Circular references
11. Sequential coupling
12. Hard-coding
13. Too much inheritance - composition is better than inheritance

# Quality automation.


THE MOST POWERFUL
TOOL WE HAVE AS
DEVELOPERS IS
AUTOMATION.
- Scott Hanselman

# Unit testing

# Unit Testing

## What is it?

**Unit testing** is a software **testing** method by which individual units of source code, sets of one or more computer program modules together with associated control data, [...], are tested to determine whether they are fit for use.

Wikipedia

# Unit Testing

The complexity of modern software makes it expensive and difficult to continually manually test.

Unit tests allow:

1. A more productive approach is to accompany every small part of your code with tests that verify its correct function.
2. This approach simplifies debugging by allowing you to catch errors early, close to their source.
3. Unit testing also allows you to refactor the code with confidence.

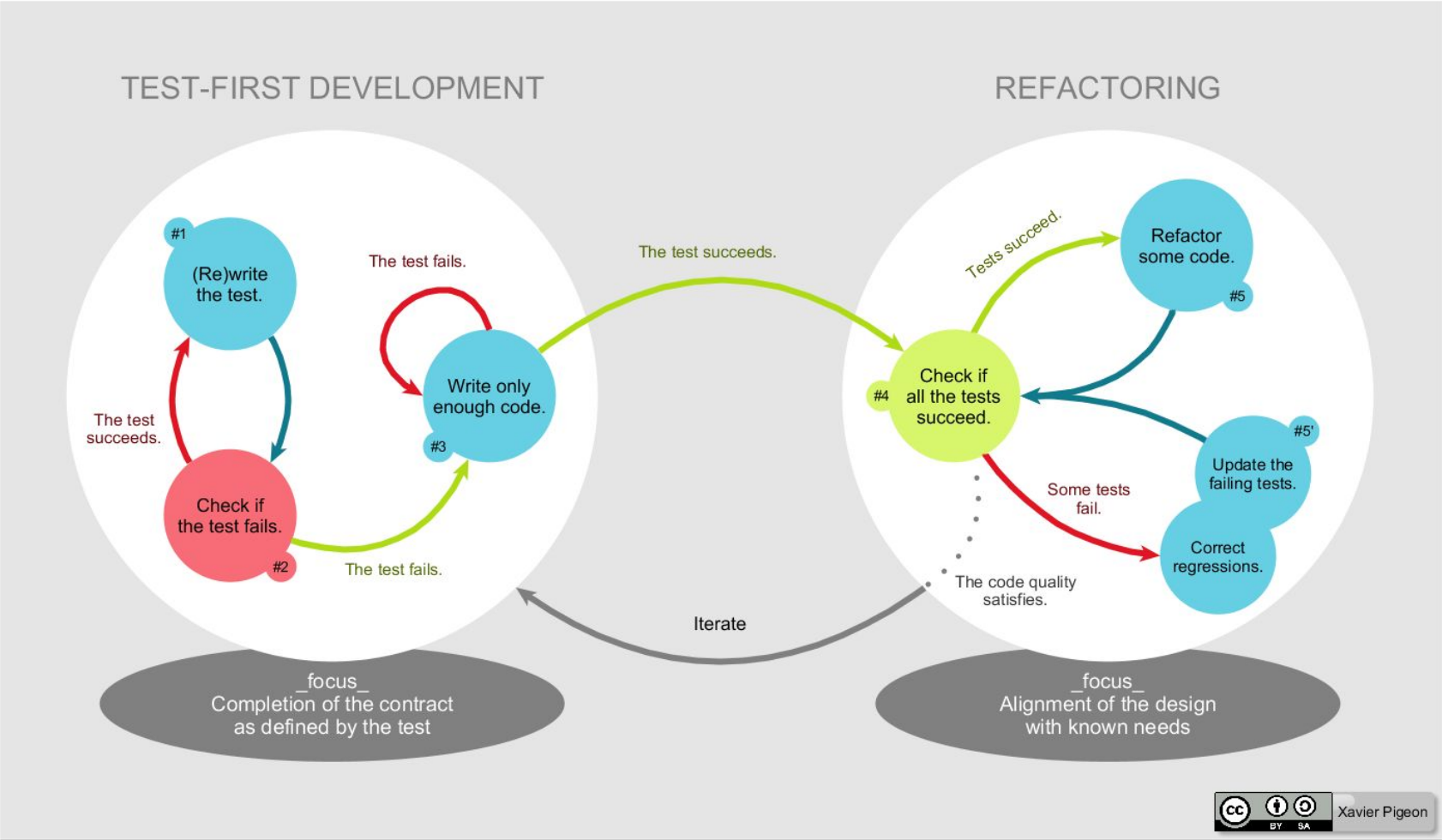
# Unit Testing

Many resources out there.

***Examples***

1. Java - JUnit: http://junit.org/junit5/
2. C# - XUnit: https://xunit.github.io/
3. PHP: PHPUnit: https://phpunit.de/

# Test driven development

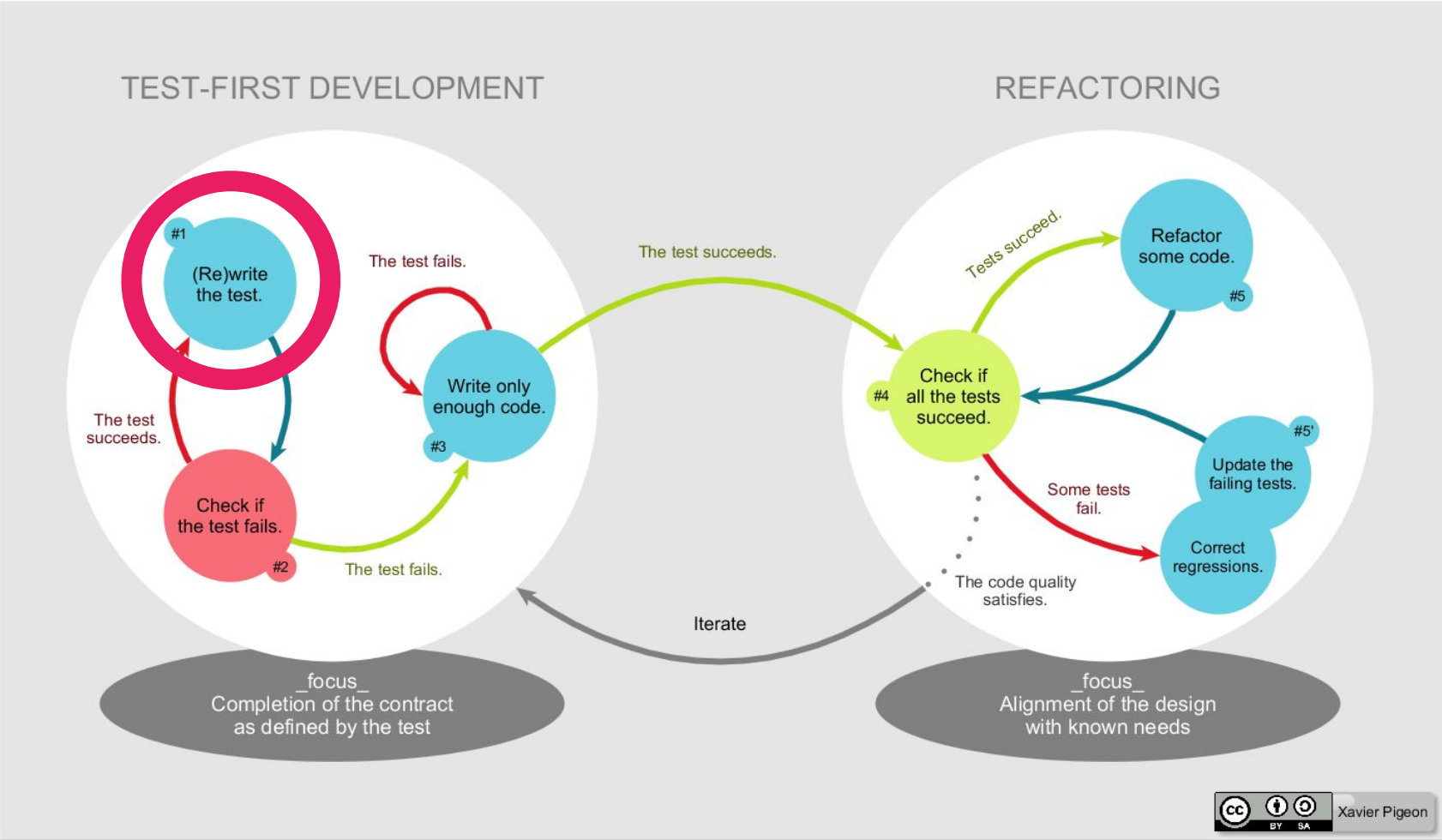

TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
The test succeeds.
Tests succeed.
Refactor some code.
#5
#4
Check if all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
_focus_
Completion of the contract as defined by the test
_focus_
Alignment of the design with known needs
Xavier Pigeon

# Test driven development

**1. Add a test - Each new feature begins with writing a test**

Writing unit tests before the code is written makes the you focus on the requirements before writing the code, an important value.

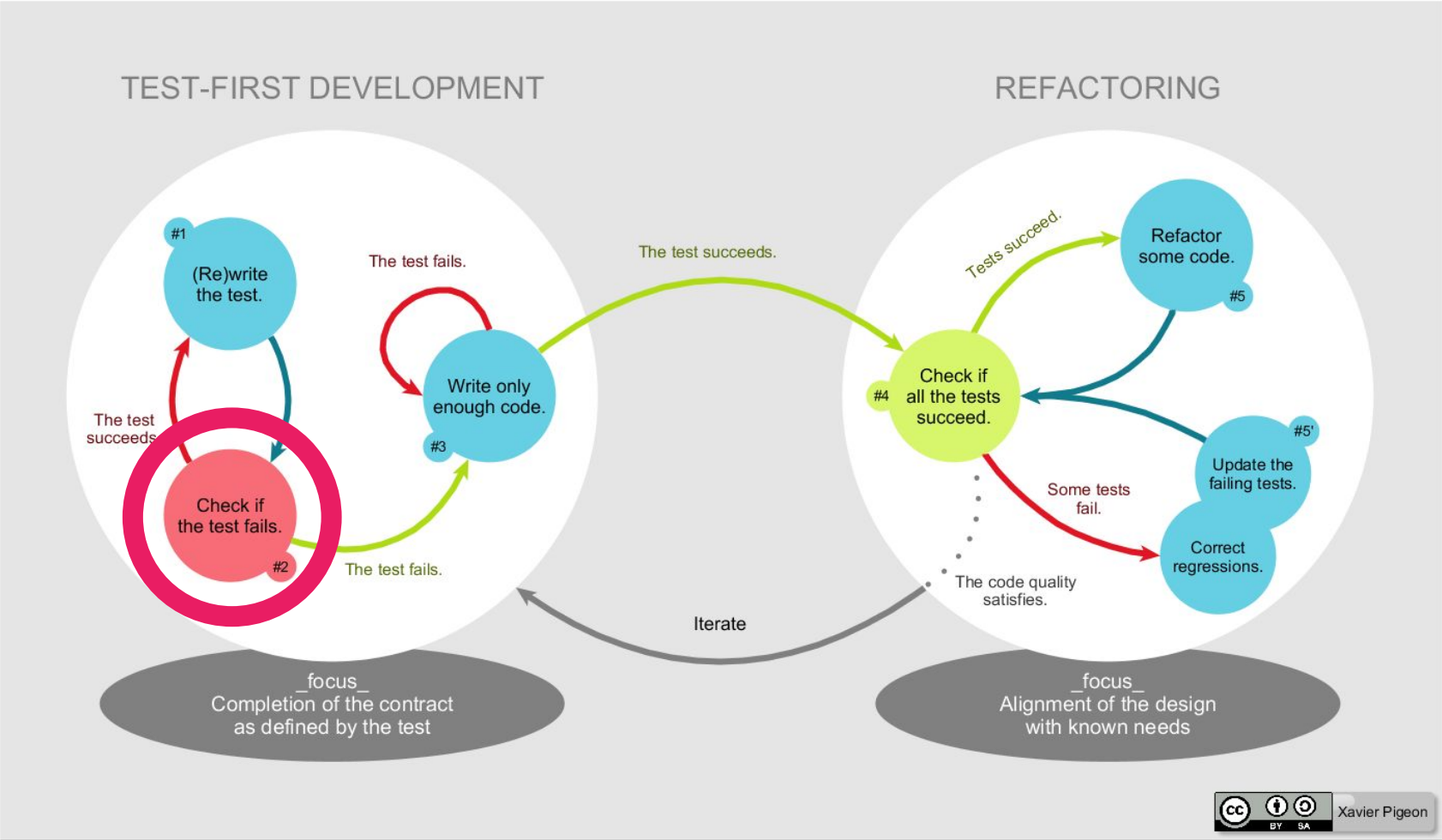
1. Write a test that defines a function or improvements of a function, which should be very clear and short.
2. Clearly understand the feature's specification and requirements.
3. Create use cases and user stories to cover the requirements and exception conditions.
4. Write the tests.


TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
The test succeeds.
Tests succeed.
Refactor some code.
#5
#4
Check if all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
_focus_
Completion of the contract as defined by the test
_focus_
Alignment of the design with known needs
Xavier Pigeon

# Test driven development

**2. Run all tests and see if the new test fails**

1. This validates that the tests are working correctly.
2. It shows that the new test does not pass without requiring new code because the required behavior already exists, and it rules out the possibility that the new test is flawed and will always pass.
3. The new test should fail for the expected reason. This step increases the developer's confidence in the new test.

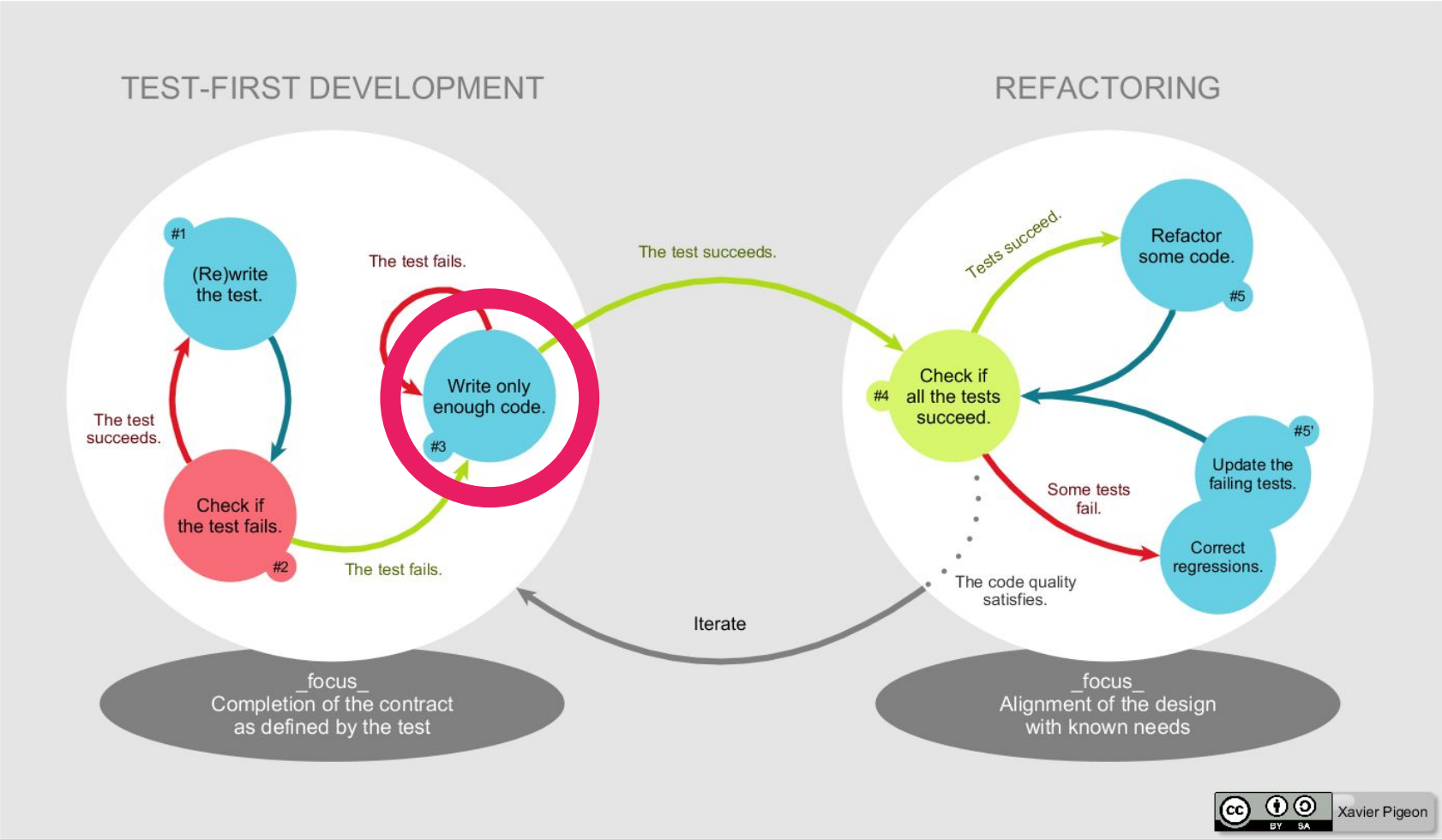

TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
The test succeeds.
Tests succeed.
Refactor some code.
#5
#4
Check if all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
_focus_
Completion of the contract as defined by the test
_focus_
Alignment of the design with known needs
Xavier Pigeon

# Test driven development

**3. Write the code**

You must not write code that is beyond the functionality that the test checks.

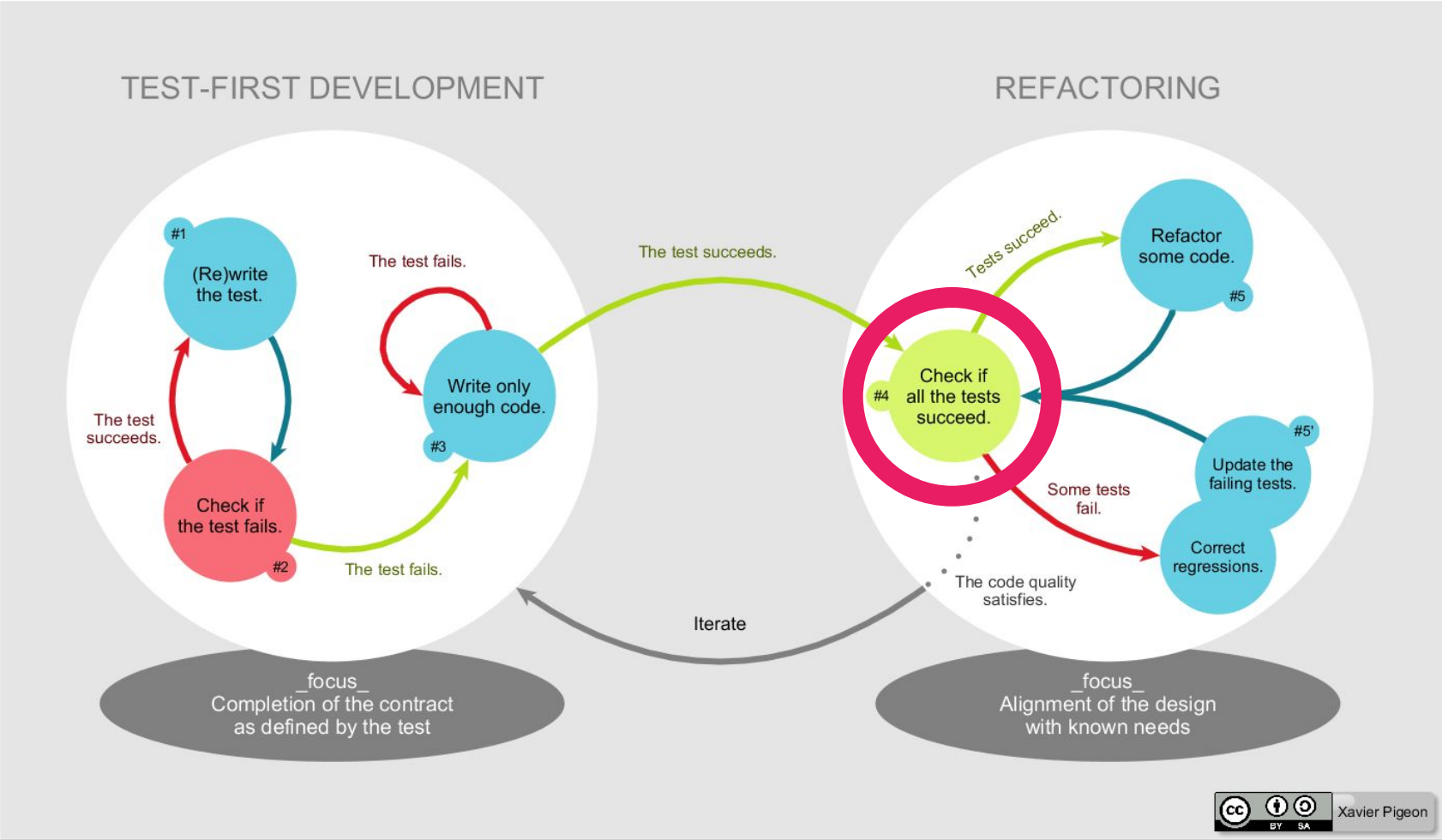
1. Write code that causes the tests to pass.
2. The new code written at this stage is not perfect and may, for example, pass the test in an inelegant way. That is acceptable because it will be improved in Step 5.
3. At this point, the only purpose of the written code is to pass the tests.


TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
The test succeeds.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
Tests succeed.
Refactor some code.
#5
#4
Check if all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
_focus_
Completion of the contract as defined by the test
_focus_
Alignment of the design with known needs
Xavier Pigeon

# Test driven development

**4. Run tests**

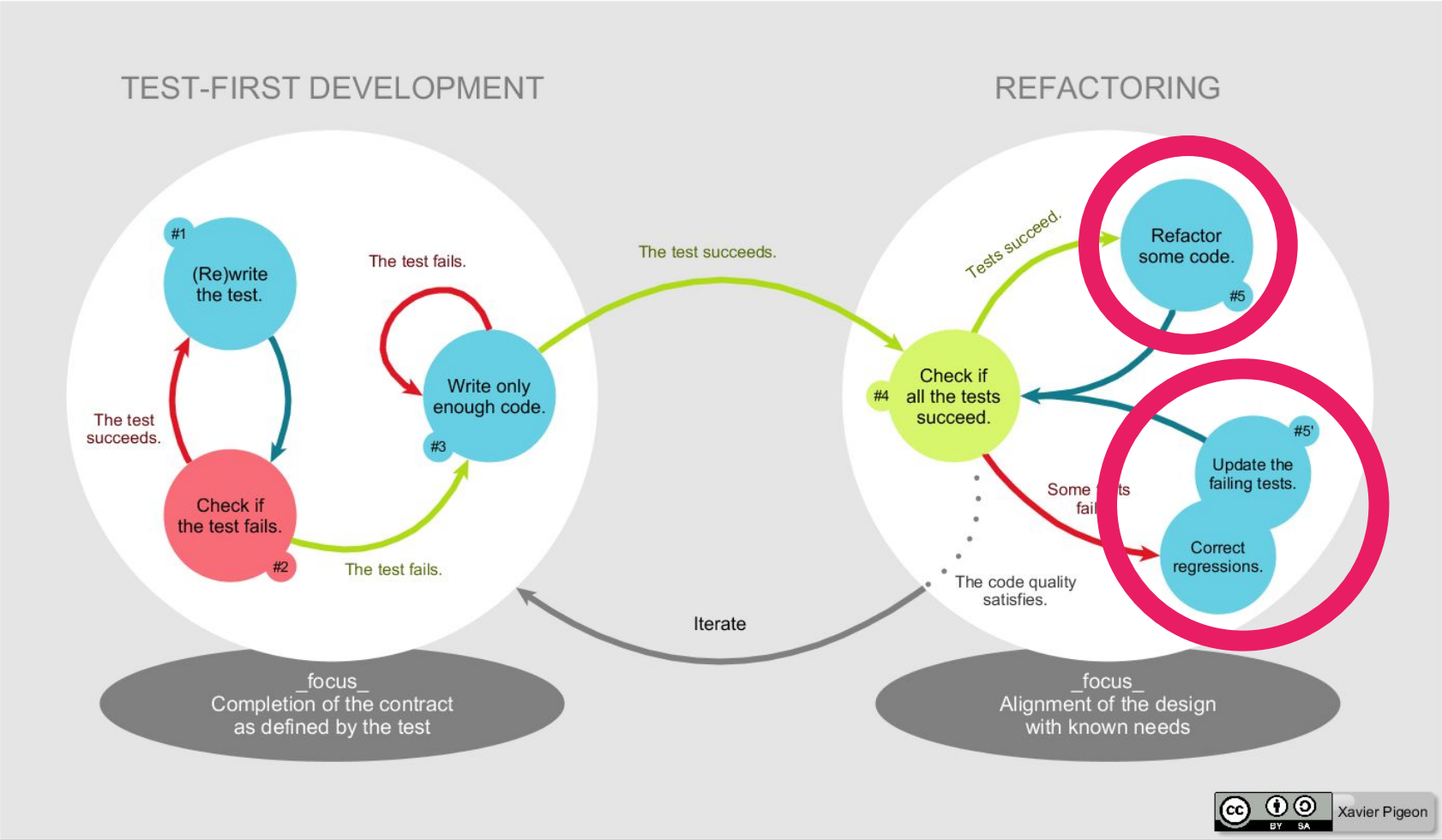
1. If all test cases now pass, the programmer can be confident that the new code meets the test requirements.
2. The new code also does not break or degrade any existing features since past tests should still pass.
3. If other any tests are now failing, the new code must be adjusted until they pass.


TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
The test succeeds.
Tests succeed.
Refactor some code.
#5
#4
Check if all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
_focus_
Completion of the contract as defined by the test
_focus_
Alignment of the design with known needs
Xavier Pigeon

# Test driven development

**5. Refactor code**

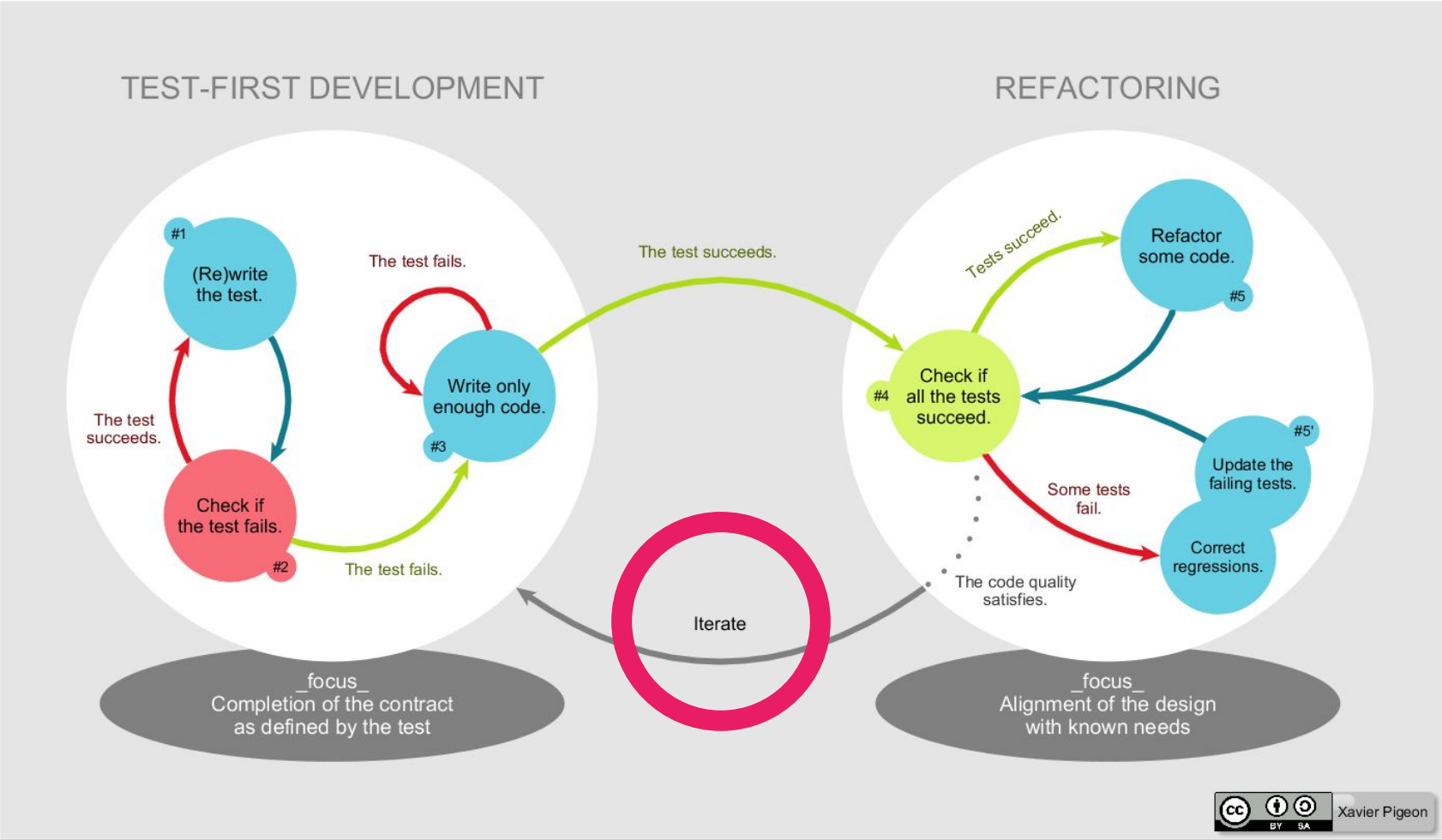
1. The growing code base must be cleaned up regularly.
2. New code can be moved from where it was convenient for passing a test to where it more logically belongs. Duplication must be removed.
3. Clean the code following the rules discussed in the presentation.
4. Inheritance hierarchies may be rearranged to be more logical and helpful, and perhaps to benefit from recognized design patterns.
5. By continually re-running the test cases throughout each refactoring phase, the developer can be confident that process is not altering any existing functionality.


TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
The test succeeds.
Tests succeed.
Refactor some code.
#5
#4
Check if all the tests succeed.
#5'
Update the failing tests.
Correct regressions.
The code quality satisfies.
Iterate
_focus_
Completion of the contract as defined by the test
_focus_
Alignment of the design with known needs
Xavier Pigeon

# Test driven development

**6. Repeat**

1. Starting with another new test, the cycle is then repeated to push forward the functionality.
2. The size of the steps should always be small, with as few as 1 to 10 edits between each test run.
3. If new code does not rapidly satisfy a new test, or other tests fail unexpectedly, the programmer should undo or revert in preference to excessive debugging.


TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
The test succeeds.
Tests succeed.
Refactor some code.
#5
#4
Check if all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
_focus_
Completion of the contract as defined by the test
_focus_
Alignment of the design with known needs
Xavier Pigeon

# Test automation tools

# Automated code review

**They provide automated capability to show health of an application and highlight coding issues.**

Tools such as:

- SonarQube (free): https://www.sonarqube.org/
- CodeClimate: https://codeclimate.com/
- Codacy: https://www.codacy.com/

CODE CLIMATE

bryan@codeclimate.com

Phpmyadmin | Feed | Code | Issues | Branches | Trends | Settings | Search by name

D

# Commenting/FileCommentSniff.php

Updated more than 3 months ago.

154 Complexity | 148 Duplication | 768 Lines | 11 Methods
14.0 Complexity / M | 0 Churn | 415 Lines of Code | 38 LOC / Method

List View | Source View

- All Issues 4
- Complexity 3
- Duplication 1

## High total complexity (complexity = 154)

## Complex method processTags (complexity = 44)

```
328     protected function processTags($commentStart, $commentEnd)
329     {
330         $docBlock    = (get_class($this) === 'PMAStandard_Sniffs_Commenting_FileCommentSniff') ? 'file'
331         $foundTags   = $this->commentParser->getTagOrders();
332         $orderIndex  = 0;
```

View more

## Identical code found in two :class_method nodes (mass*2 = 296)

- PMAStandard/Sniffs/Commenting/FileCommentSniff.php:561...586

```
561     protected function processPackage($errorPos)
562     {
563         $package = $this->commentParser->getPackage();
564         if ($package !== null) {
565             $content = $package->getContent();
566             if ($content !== '') {
```

# IDE - Integrated development environment

**The craftsman's essential tool**

1. Integrate all these tools we talked about directly into your IDE. Automated code review, build tools for TDD, etc.
2. Get to know all the features.
3. Try to do it all without using the mouse.

# APM - Application performance management

**What is it?**

APM is the monitoring and management of performance and availability of software applications. APM tries to detect and diagnose complex application performance problems to maintain an expected level of service.

Tools such as:

- New Relic: https://newrelic.com/
- Sentry: https://sentry.io/welcome/

New Relic APM

Alerts New | Tools | Help | NewRelic Administration | 3

Applications | Service maps | Key transactions

APPS
RPM

TIME PICKER
Last 60 minutes ending now

MONITORING

- Overview
- Service maps
- App map
- Transactions
- Databases
- External services
- Ruby VMs

EVENTS

- **Error analytics**
- Errors
- Violations
- Deployments
- Thread profiler

REPORTS

- SLA
- Availability
- Capacity
- Scalability
- Web transactions

Search keys and attributes...

Back to groupings list

## Error class

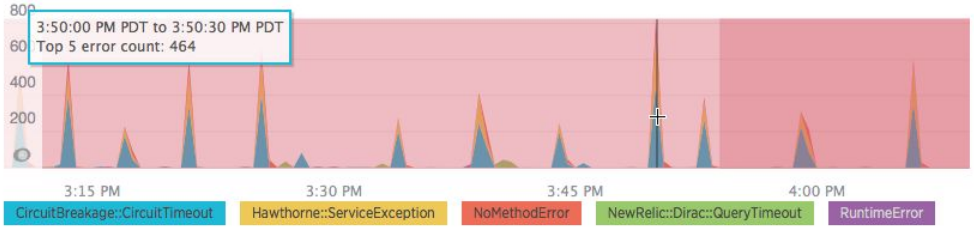
| FILTER PAGE BY | Error count |
|---|---|
| CircuitBreakage::CircuitTime | 4043 |
| Hawthorne::ServiceExceptio | 1756 |
| NoMethodError | 790 |
| NewRelic::Dirac::QueryTime | 182 |
| RuntimeError | 59 |
| TypeError | 38 |
| ActiveRecord::RecordNotSave | 33 |
| ActionView::Template::Error | 29 |
| NewRelic::Dirac::QueryError | 29 |
| ActionController::UnknownHtt | 28 |
| Net::OpenTimeout | 19 |
| Rack::Timeout::RequestTimeo | 13 |
| Net::HTTP::Persistent::Error | 9 |
| Timeout::Error | 4 |
| SamlController::SamlAdapter:: | 4 |
| ZeroDivisionError | 3 |
| SamlController::SamlAdapter:: | 3 |

## Error rate

for all errors


## Top 5 errors

by error class




View query | View in Insights

**Error traces** | Error frequency by error class

| Count | Transaction name and error class | Error message | First Occurrence | Last Occurrence |
|---|---|---|---|---|
| 1667 | ChartData::MetricChartsController#error_rate<br>CircuitBreakage::CircuitTimeout | execution expired | 22:10 | 23:11 |
| 1578 | CurrentStatusController#status_bar<br>Hawthorne::ServiceException | Hawthorne::ServiceException | 22:10 | 23:11 |

# Peer code review

# Peer Code Review

Code review is systematic examination (sometimes referred to as peer review) of computer source code. It is intended to find mistakes overlooked in software development, improving the overall quality of software. Reviews are done in various forms such as pair programming, informal walkthroughs, and formal inspections.[1]

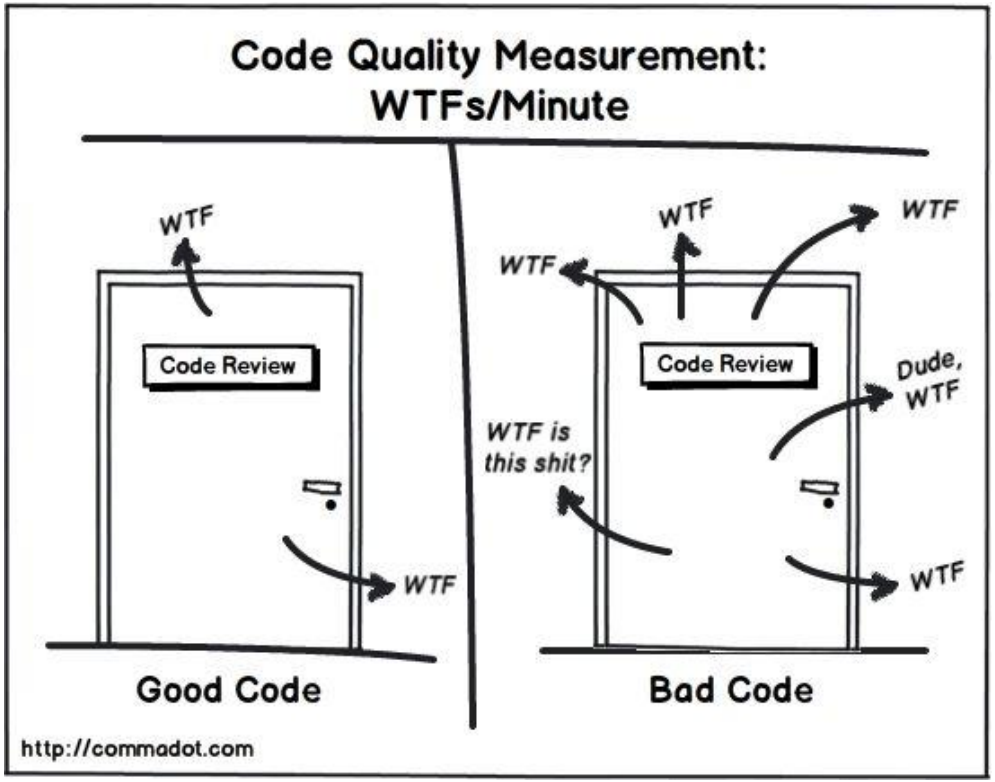
Wikipedia

# HUMAN INTELLIGENCE.

# Conclusion

# How clean should it be?

## Less code-smells



# The boy scout rule.

"Always check [code] in cleaner than when you checked it out."

- Robert C. Martin

# Conclusion

Books on art don’t promise to make you an artist.

“Continuous effort - not strength or intelligence - is the key to unlocking our potential.”

-Winston Churchill

# Books

**Clean Code - A Handbook of Agile Software Craftsmanship**

*by Robert C. Martin*

**Code Complete 2**

*by Steve McConnell*

**Craftman**

*by Robert C. Martin*


QUESTIONS ?

# Sites

## References

- http://www.informit.com/articles/article.aspx?p=2223710
- https://www.butterfly.com.au/blog/website-development/clean-high-quality-code-a-guide-on-how-to-become-a-better-programmer
- https://confluence.sakaiproject.org/display/BOOT/Best+Practices+for+High+Quality+Code