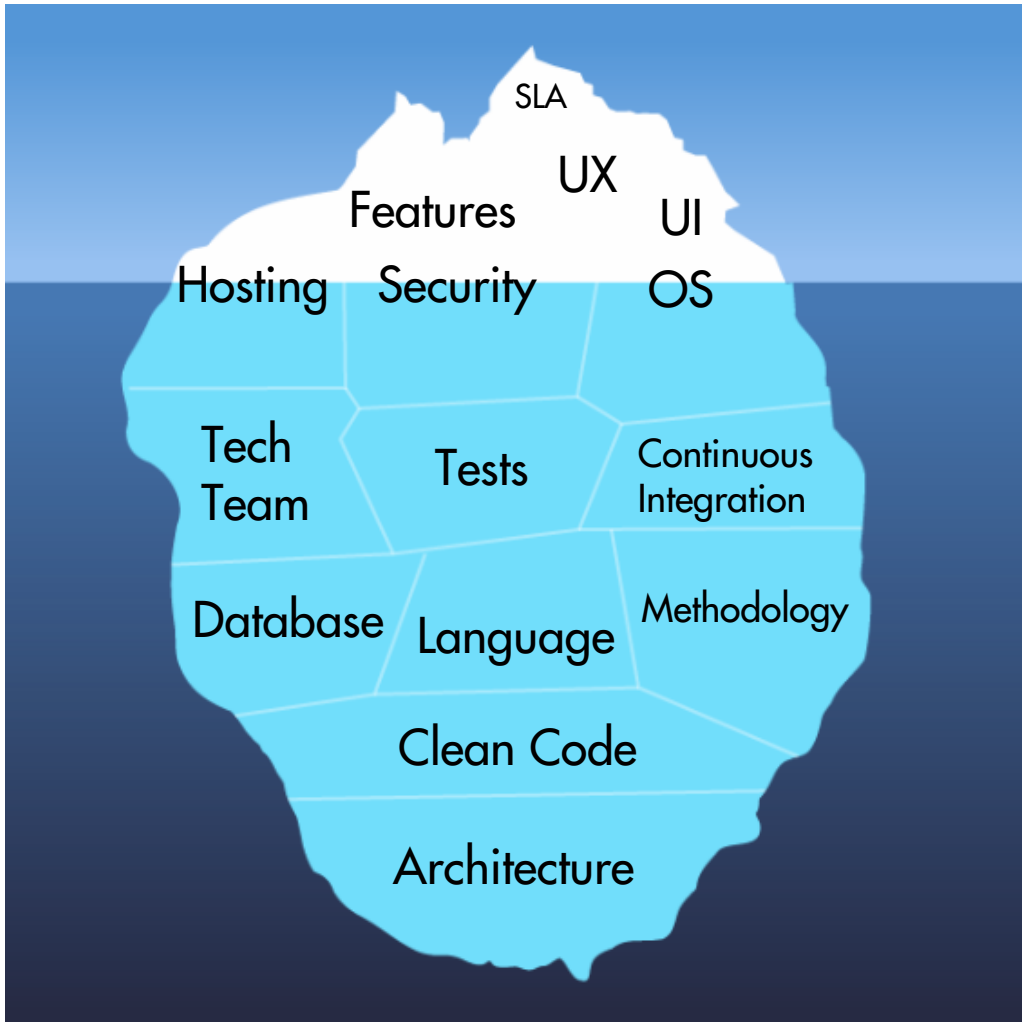
Software craftmanship

Emmanuel CHAFFRAIX

30/04/2020

Why are you here ?

Tech is the hidden part of the iceberg



What we will discuss

Code

Clean code, good practices, tests

Methodologies

Agility, programming

Software Architecture

Desktop application, cloud

Who am I ?

EPITA 2009 – GISTR

2010 – 2013: IT Consultant in CIB

2013 – 2017: Knowledge manager

2017 – Present: CTO & Co-founder at Crafty

Clean code, what is it ?

30/04/2020

What is your pain ?

Bugs ?

Coding style ?

Working with others ?

Developer life...

Never thanked, only complaints !

By code :

- Compilation failed
- Seg fault
- Stack overflow

By client :

- ASAP
- Bad UX
- Not working

...is awesome

We create product

We discover new trades

We learn !

Sotfware Craftmanship Manifesto

Not only working software,
but also **well-crafted software**
Not only responding to change,
but also **steadily adding value**
Not only individuals and interactions,
but also **a community of professionals**
Not only customer collaboration,
but also **productive partnerships**

Source: http://manifesto.softwarecraftsmanship.org/

```
public class ShippingController extends Controller {
    private CartService cartService ;
    public void process(ShippingForm form) throws Exception , SQLException {
        CartBean cartBean = cartService.getCart(form.getCartId());
        BigDecimal cost = null ;
        if(cartBean.getTotal() <= 100) {
            cost = new BigDecimal(4.99) ;
            if(form.getOption() == 1) {// expedited
                for(Item i : cartBean.getItems()) {
                    if(i.getCat() == 'B') {
                        cost.add(new BigDecimal(2.99));
                    }
                    if(i.getCat() == 'O') {// other by weight
                        cost.add(new BigDecimal(i. getWeight() / 1000).multiply(new BigDecimal(1.99)));
                    }
                }
            }
            if(form.getOption() == 2) {// priority
                for(Item i : cartBean.getItems()) {
                    if(i.getCat() == 'B') {
                        cost.add(new BigDecimal(4.99));
                    }
                    if(i.getCat() == 'O') {// other by weight
                        cost.add(new BigDecimal(i. getWeight() / 1000).multiply(new BigDecimal(2.99)));
                    }
                }
            }
        }
        form.setCost(cost);
    }
}
```

Magic numbers

Logic, but number used

Really close

B / O ?

KG ?

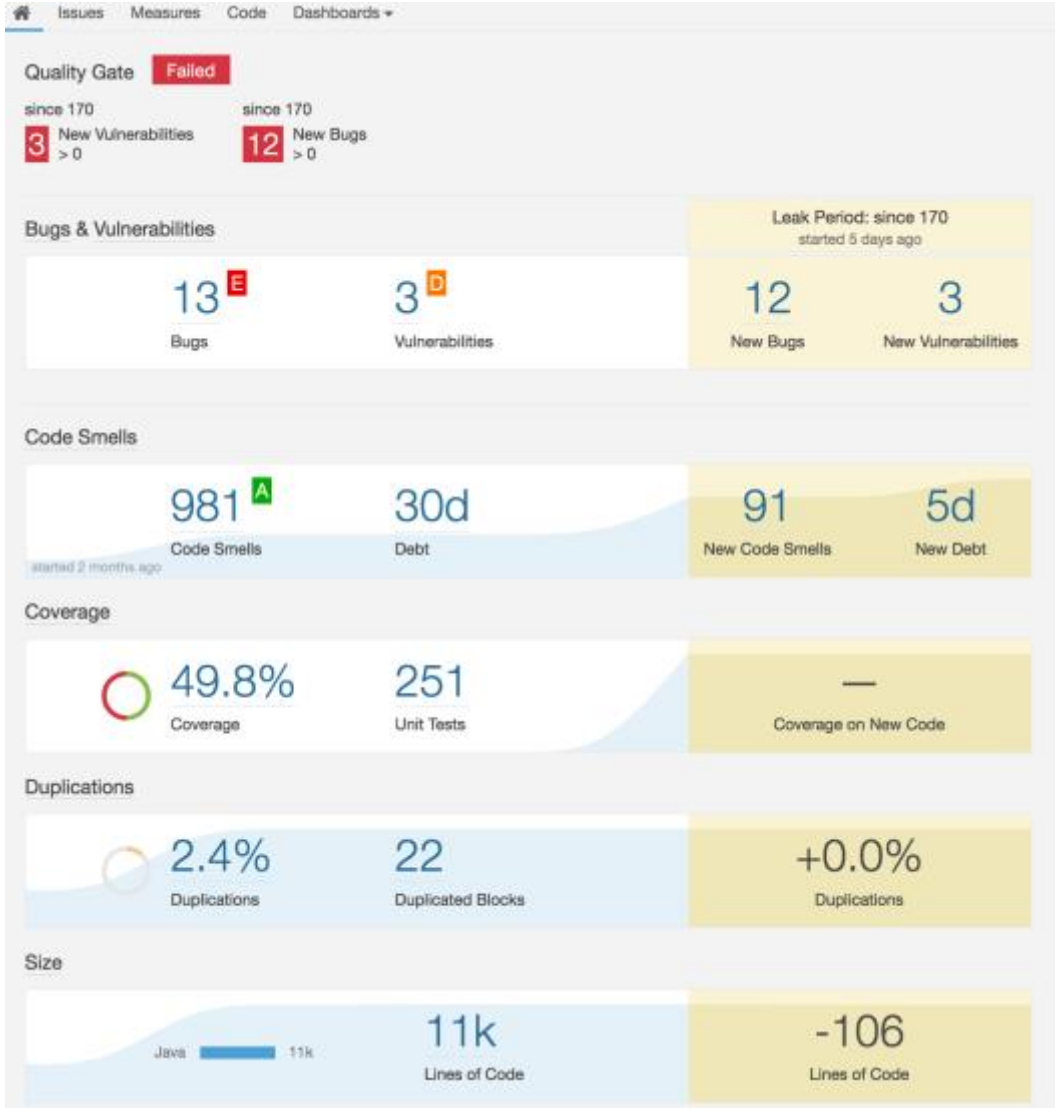
What is the currency ? € / $ / £

Technical debt impacts

Code readability

New features development

Bug fixing



Be KISS

Keep It Simple, Stupid

leboncoin

Be DRY

Don’t repeat yourself

```
for(Item i : cartBean.getItems()) {
    if(i.getCat() == 'B') {
        cost.add(new BigDecimal(2.99));
    }
    if(i.getCat() == 'O') {// other by weight
        cost.add(new BigDecimal(i. getWeight() / 1000).multiply(new BigDecimal(1.99)));
    }
}
```

Be focused

YAGNI : You Ain’t Gonna Need It

DIE

Duplication Is Evil

Copy / Paste is easy, but hard to bug fix

```
if (i % 2 == 0) {
  if (i % 1 == 0) {
    doTheJob();
  }
}
```

```
foo.Bar();
if (foo != null) {
  foo.Fizz();
}
```

```
public class ShippingController extends Controller {
    private CartService cartService ;
    public void process(ShippingForm form) throws Exception , SQLException {
        CartBean cartBean = cartService.getCart(form.getCartId());
        BigDecimal cost = null ;
        if(cartBean.getTotal() <= 100) {
            cost = new BigDecimal(4.99) ;
            if(form.getOption() == 1) {// expedited
                for(Item i : cartBean.getItems()) {
                    if(i.getCat() == 'B') {
                        cost.add(new BigDecimal(2.99));
                    }
                    if(i.getCat() == 'O') {// other by weight
                        cost.add(new BigDecimal(i. getWeight() / 1000).multiply(new BigDecimal(1.99)));
                    }
                }
            }
            if(form.getOption() == 2) {// priority
                for(Item i : cartBean.getItems()) {
                    if(i.getCat() == 'B') {
                        cost.add(new BigDecimal(4.99));
                    }
                    if(i.getCat() == 'O') {// other by weight
                        cost.add(new BigDecimal(i. getWeight() / 1000).multiply(new BigDecimal(2.99)));
                    }
                }
            }
        }
        form.setCost(cost);
    }
}
```

```
public Amount calculateShippingCost(Cart cart, ShippingOption chosenOption)
{
    if (cart.getTotalPrice().lessThan(THRESHOLD_FOR_FREE_SHIPPING)) {
        Amount shippingCost = getCostPerShipment(cart.getAllProductCategories());
        for (Item item : cart.getItems());
        {
            if (chosenOption == EXPEDITED) {
                shippingCost = shippingCost.add(ExpeditedShipping.getCostPerItem(item.getProduct()));
            }

            if (chosenOption == PRIORITY) {
                shippingCost = shippingCost.add(PriorityShipping.getCostPerItem(item.getProduct()));
            }
        }
        return shippingCost;
    }
    return ZERO_DOLLAR;
}
```

Simple rules

Bugs are everywhere : one operation per line helps you to identify the location

```
list.Add(line.WordToString());
```

Become :

```
var words = line.WordToString();
list.Add(words);
```

Simple rules

Do not ask twice for the same thing, it improves your system perfs

```
Foo(lines.Where(l => l.section == null).ToArray());
Bar(lines.Where(l => l.section == null).ToArray(), lines);
```

Become :

```
var linesWithSection = lines.Where(l => l.section == null).ToArray()
Foo(linesWithSection);
Bar(linesWithSection, lines);
```

Simple rules

Everytime, ask you the question :

Should I be able to understand my code in 6 months ? One year ?

So :

- Use explicit names
- Read your code
- Code phrases

Simple rules

Rules in EPITA's coding style

Old book but still relevant : *Code Complete, Steve McConnel*

Another Example : Balanced Brackets

```
static void Main(String[] args)
{
    int T = Convert.ToInt32(Console.ReadLine());
    for (int a0 = 0; a0 < T; a0++)
    {
        string s = Console.ReadLine();
        bool isBalanced = true;
        Stack<char> chars = new Stack<char>();
        for (int i = 0; i < s.Length; i++)
        {
            var c = s[i];
            if (c == '(' || c == '{' || c == '[') chars.Push(c);
            else if (chars.Count > 0)
            {
                var t = chars.Peek();
                if ((c == ')' && t == '(') || (c == '}' && t == '{') || (c == ']' && t == '[')) chars.Pop();
                else { isBalanced = false; break; }
            }
            else { isBalanced = false; break; }
        }
        Console.WriteLine(isBalanced && chars.Count == 0 ? "YES" : "NO");
    }
}
```

Another Example refactored

```
private static readonly char[] opening = { '(', '[', '{' };
private static readonly char[] closing = { ')', ']', '}' };

public static void Main(string[] args)
{
    int step = ConsoleAdapter.ReadInt();
    while (step > 0)
    {
        step--;
        var line = ConsoleAdapter.ReadLine();
        var result = Check(line);
        ConsoleAdapter.Write(result);
    }
}
```

Another Example refactored

```
public static bool Check(string s)
{
    Stack<char> last = null;
    var isValid = true;
    if (s != null)
    {
        last = new Stack<char>(s.Length / 2);
        for (var i = 0; isValid && (i < s.Length); i++)
        {
            var current = s[i];
            if (opening.Contains(current)) {
                last.Push(current);
            } else if (closing.Contains(current)) {
                int position = closing.Select((c, j) => new { c, j }).First(x => x.c == current).j;
                isValid = (last.Count > 0) && (last.Pop() == opening[position]);
            }
            else { isValid = false; }
        }
    }
    return isValid && (last == null || last.Count == 0);
}
```

OO Principles

Encapsulation

Inheritance

Polymorphism

- Overloading
- Templates / Generics
- Subtypings

S.O.L.I.D

SRP : Single responsibility principle

OCP : Open/closed principle

LSP : Liskov substitution principle

ISP : Interface segregation principle

DIP : Dependency inversion principle

Single Responsibility Principle

A class should have only one reason to change.

```
public class Person {
    public string Name { get; set; }
    public string Email { get; set; }

    public Person(string name, string email)
    {
        Name = name;
        Email = email;
        ValidateEmail();
    }

    private void ValidateEmail() {
        // throw if not an email
    }
}
```

```
public class Person
{
    public string Name { get; set; }
    public Email Email { get; set; }
}

public class Email {
    public string Address { get; private set; }
    public Email(string address) {
        Address = address;
        ValidateEmail();
    }
    private void ValidateEmail() {
        // throw if not an email
    }
}
```

Single Responsibility Principle

A class should have only one reason to change.

```
public class EmailSender {
  public void SendEmail(string customerID,
    string emailNotificationType) {
    //STEP1: load customer details
    //STEP2: get email content
    //STEP3: send email (using SmtpClient class)
  }

  public string GetEmailContent(Customer customer,
    string emailNotificationType) {
    // Build the email notification content
   }
}
```

```
public class CustomerRespository {
  public Customer GetCustomer(string id) {
    // logic load customer from Database
  }
}
public class EmailContentBuilder {
  public string getEmailContent(
    Customer customerDetails) {
    // logic to build email content
  }
}
public class EmailSender {
 public void SendEmail(string emailaddress,
    string subject, string bodycontent) {
    //logic to send email out
  }
}
```

Open/Close Principle

Software entitites should be open for extension, but closed for modification

```
public class Greeter {
    String formality;

    public String Greet() {
        if (this.formality == "formal") {
            return "Good evening, sir.";
        } else if (this.formality == "casual") {
            return "Sup bro?";
        } else if (this.formality == "intimate") {
            return "Hello Darling!";
        } else {
            return "Hello.";
        }
    }

    public void SetFormality(string formality) {
        this.formality = formality;
    }
}
```

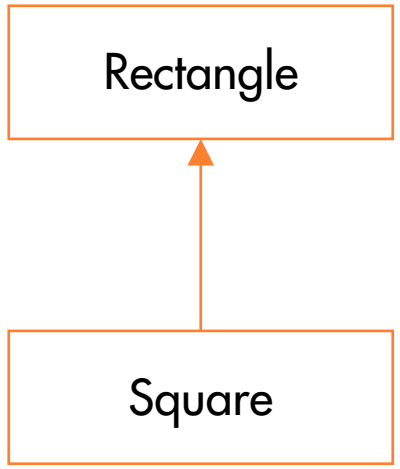
```
public class Greeter {
    private Personality personality;

    public Greeter(Personality personality) {
        this.personality = personality;
    }

    public String greet() {
        return this.personality.greet();
    }
}
```

Liskov Substitution Principle

If S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program



Liskov Substitution Principle

If S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program

```
abstract class Apartment {
    int squareFootage;
    int numberOfBedrooms;
}
public class PenthouseSuite extends Apartment {
    public PenthouseSuite() {
        this.numberOfBedrooms = 4;
    }
}
public class Studio extends Apartment {
    public Studio() {
        this.numberOfBedrooms = 0;
    }
}
```

```
public class UnitUpgrader {
    public void upgrade(Apartment apartment) {
       if (apartment.getClass() != Studio.class)
            apartment.numberOfBedrooms += 1;
    }
}
```

Liskov Substitution Principle

If S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program

```
abstract class Apartment {
    int squareFootage;
    int numberOfBedrooms;
}
public class PenthouseSuite extends Apartment {
    public PenthouseSuite() {
        this.numberOfBedrooms = 4;
    }
}
public class Studio extends Apartment {
    public Studio() {
        this.numberOfBedrooms = 0;
    }
}
```

```
public class UnitUpgrader {
    public void upgrade(Apartment apartment) {
        apartment.numberOfBedrooms += 1;
    }

    public void upgrade(Studio apartment) {
    }
}
```

Interface Segregation Principle

Many client-specific interfaces are better than one general-purpose interface

```
interface IAmAUser
{
    void Login(string user, string password);
    void Logout();
    bool IsConnected();
    bool IsAdmin();
    Right[] GetRights();
}
```

```
interface IHaveASession
{
    void Login(string user, string password);
    void Logout();
    bool IsConnected();
}
interface IHaveRights
{
    bool IsAdmin();
    Right[] GetRights();
}
```

Dependency Injection Principle

One should "depend upon abstractions, not concretions."

```
public class FileLogger {
    public void Info(string message) {
        // Write message into a file
    }
}

public class Application {
    private FileLogger logger;

    public Application() {
        this.logger = new FileLogger();
    }

    public void Run() {
        logger.Info("Running");
    }
}
```

```
public class FileLogger : ILogger {
    public void Info(string message) {
        // Write message into a file
    }
}

public class Application {
    private ILogger logger;

    public Application(ILogger logger) {
        this.logger = logger();
    }

    public void Run() {
        logger.Info("Running");
    }
}
```

Be Agile

30/04/2020

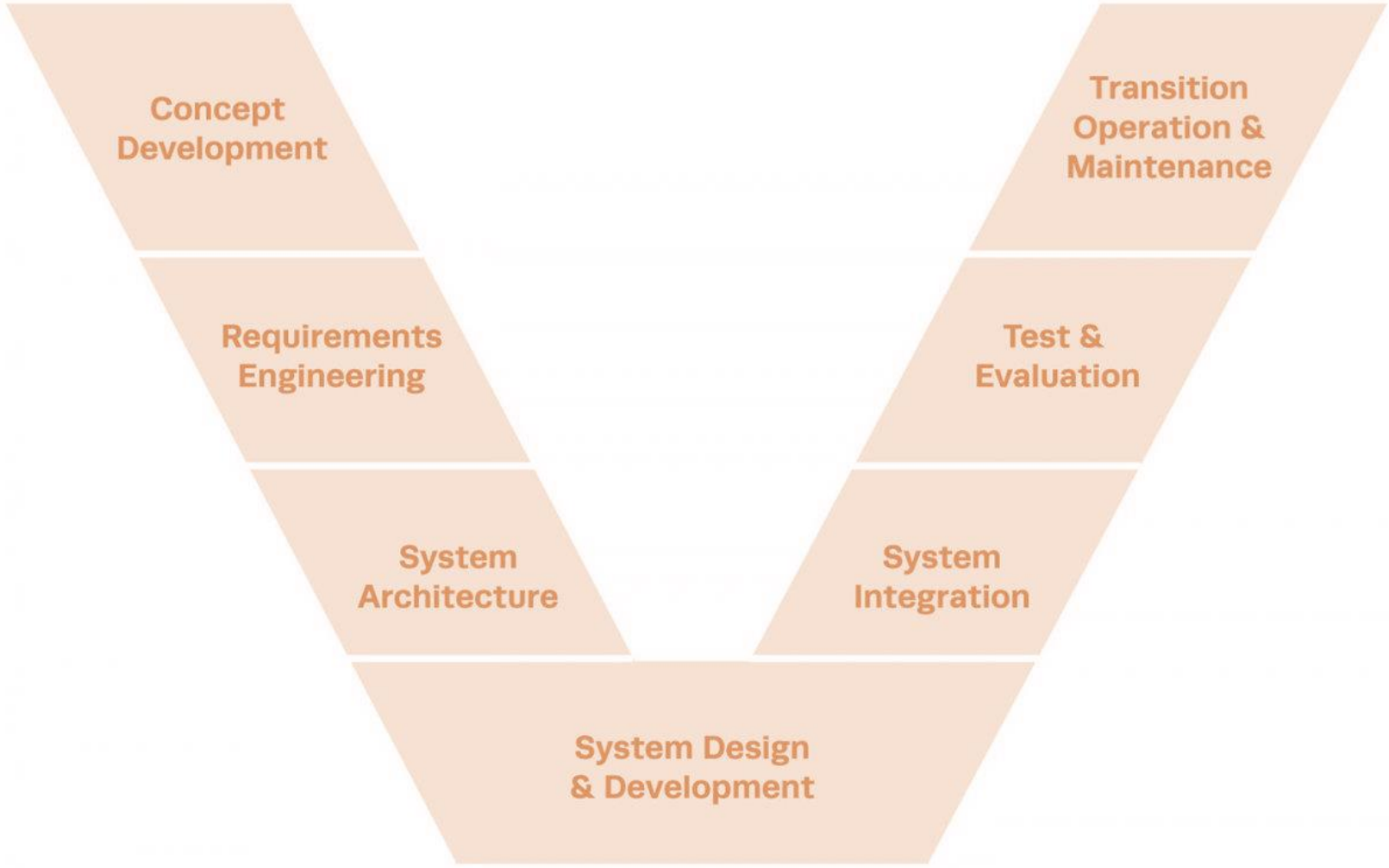
V Cycle

Functional level

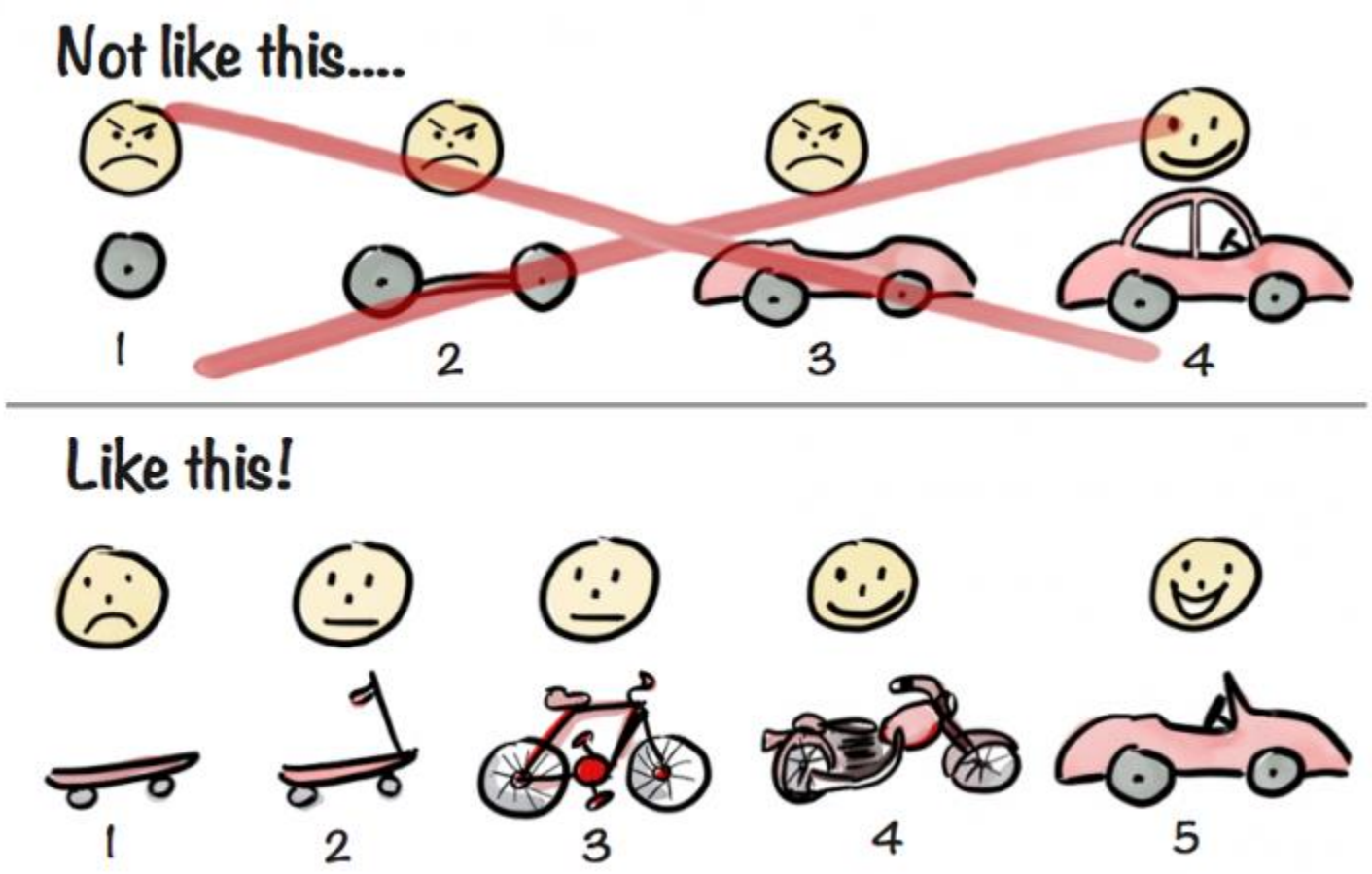
System level

Subsytem level

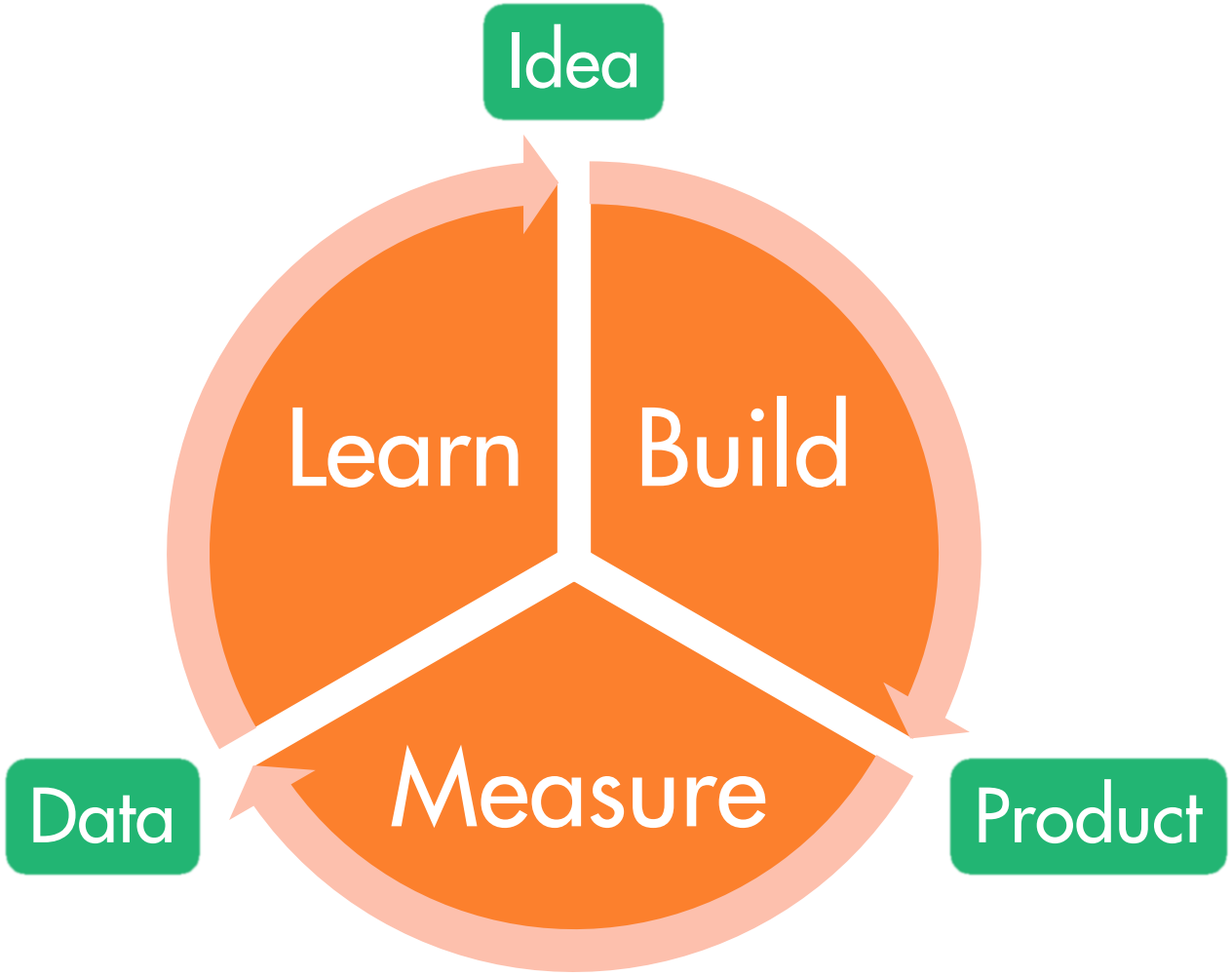
Component level



Agile



Agile



SCRUM Methodology

Product Owner

Scrum master

Developer team

Backlog

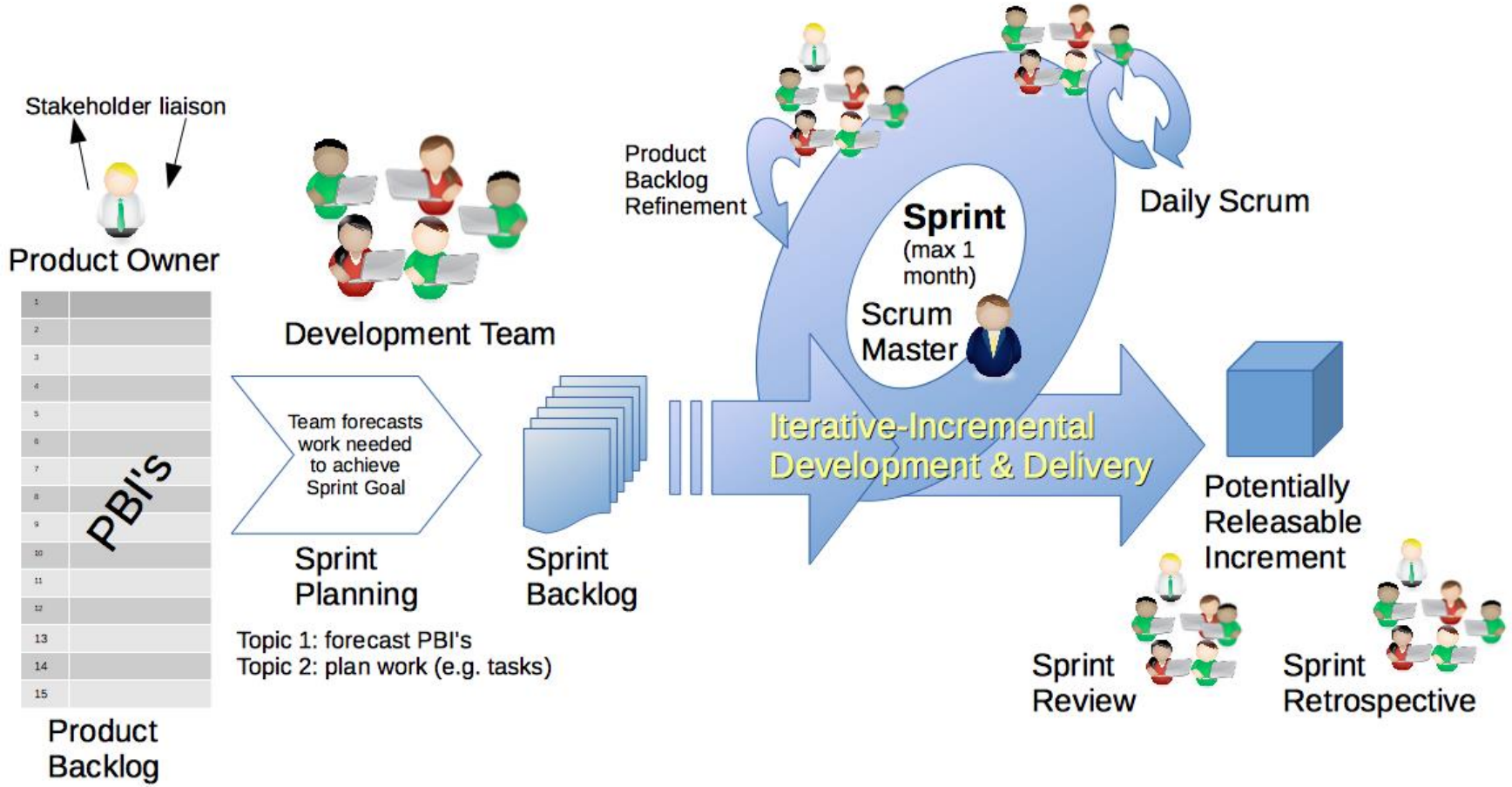
SCRUM Workflow

Sprint Meeting Planning

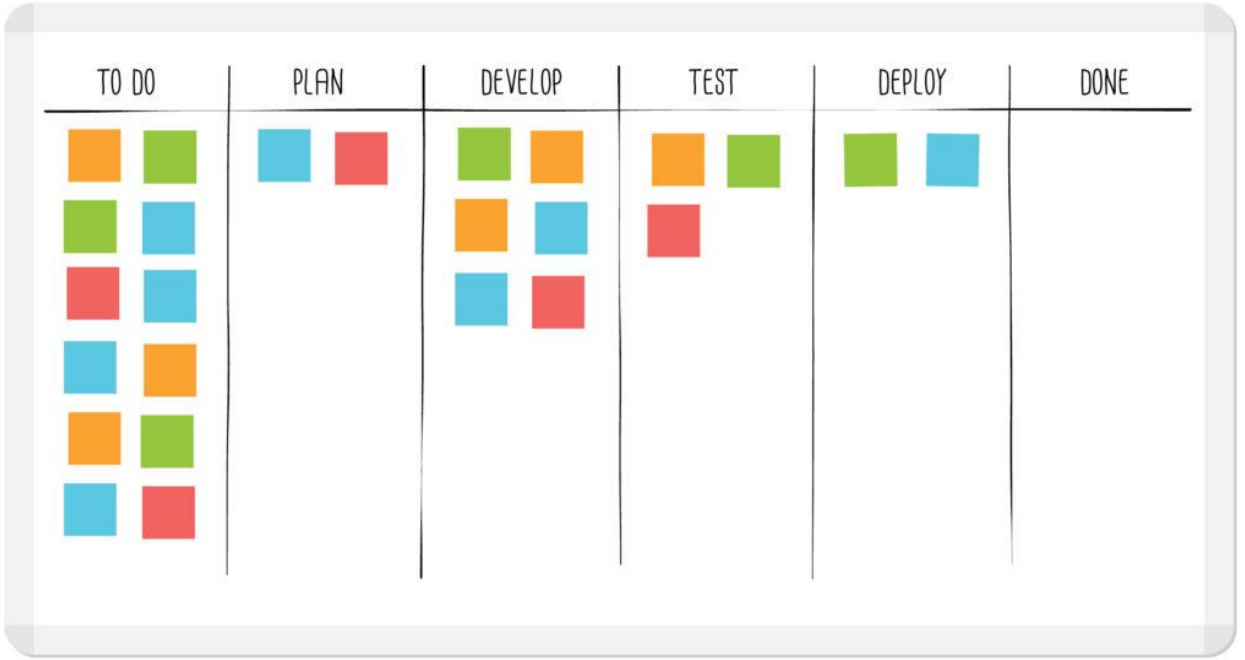
Daily Scrum Meeting

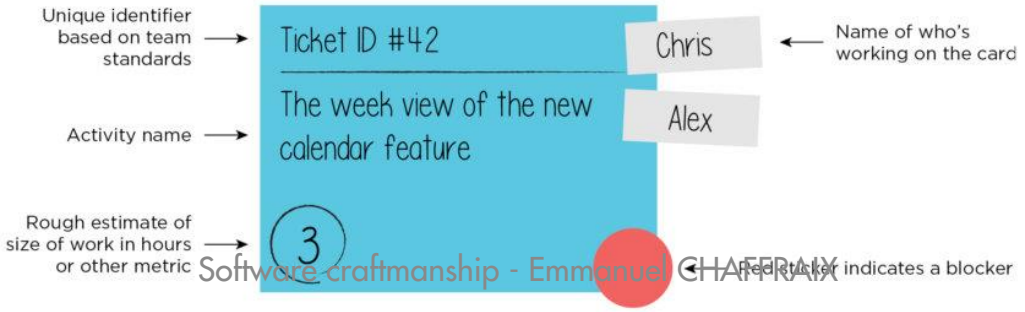
Sprint Retrospective

SCRUM Diagram



Kanban


TO DO
PLAN
DEVELOP
TEST
DEPLOY
DONE


Unique identifier based on team standards
Ticket ID #42
Chris
Name of who's working on the card
Activity name
The week view of the new calendar feature
Alex
Rough estimate of size of work in hours or other metric
3
Red sticker indicates a blocker

Think tests

Why do we test ?

Tests helps to understand requirements

Tests protects future developments

Tests are a fall protection

We are human afterall

How do we test ?

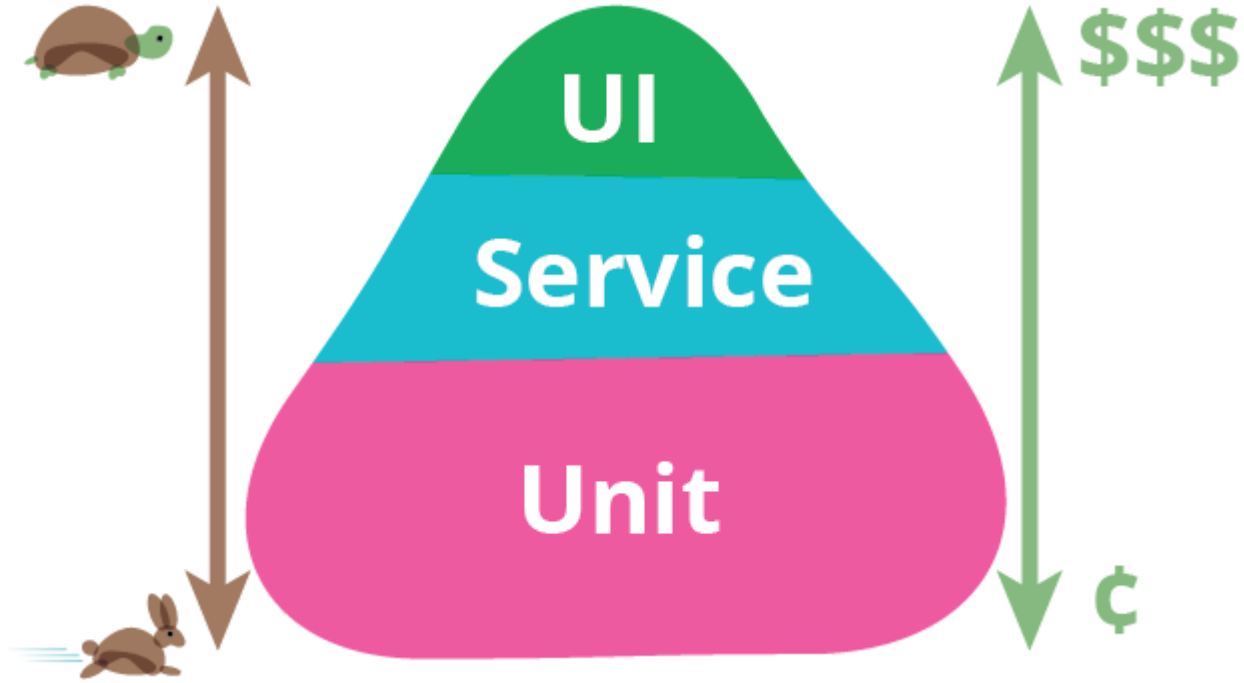
We create code to test our code

Unit Test : Focus on ONE function usage, mock dependencies

Integration Test : Crosses the boundary between components

Behaviour Test : An example of the user using the system

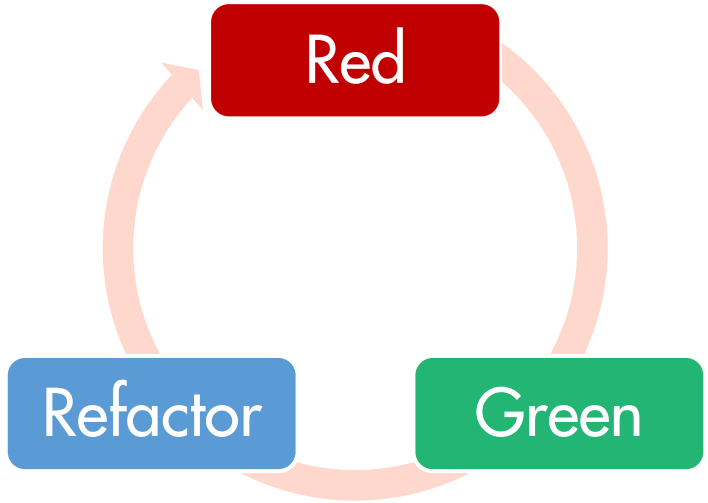
The test pyramid



TDD : Test Driven Development

No code without a test

1. Write a tests which fails
2. Write the code which fulfil the test
3. Refactor source code (feature and test !)



What is a good unit test ?

Easy to understand, clear when it fails

Determinist

Focused

AAA : Arrange / Act / Assert

BDD : Behaviour Driven Development

A simple test :

Given …….. When ……. Then …..

Given *a user with an account*
When *he tries to create a new account with existing account credentials*
Then *it must be logged in with the existing account*

BDD : Behaviour Driven Development

A feature contains tests

```
Feature: Serve coffee
  Coffee should not be served until paid for
  Coffee should not be served until
         the button has been pressed
  If there is no coffee left then money should be refunded

Scenario: Buy last coffee
  Given there are 1 coffees left in the machine
        And I have deposited 1$
  When I press the coffee button
  Then I should be served a coffee
```

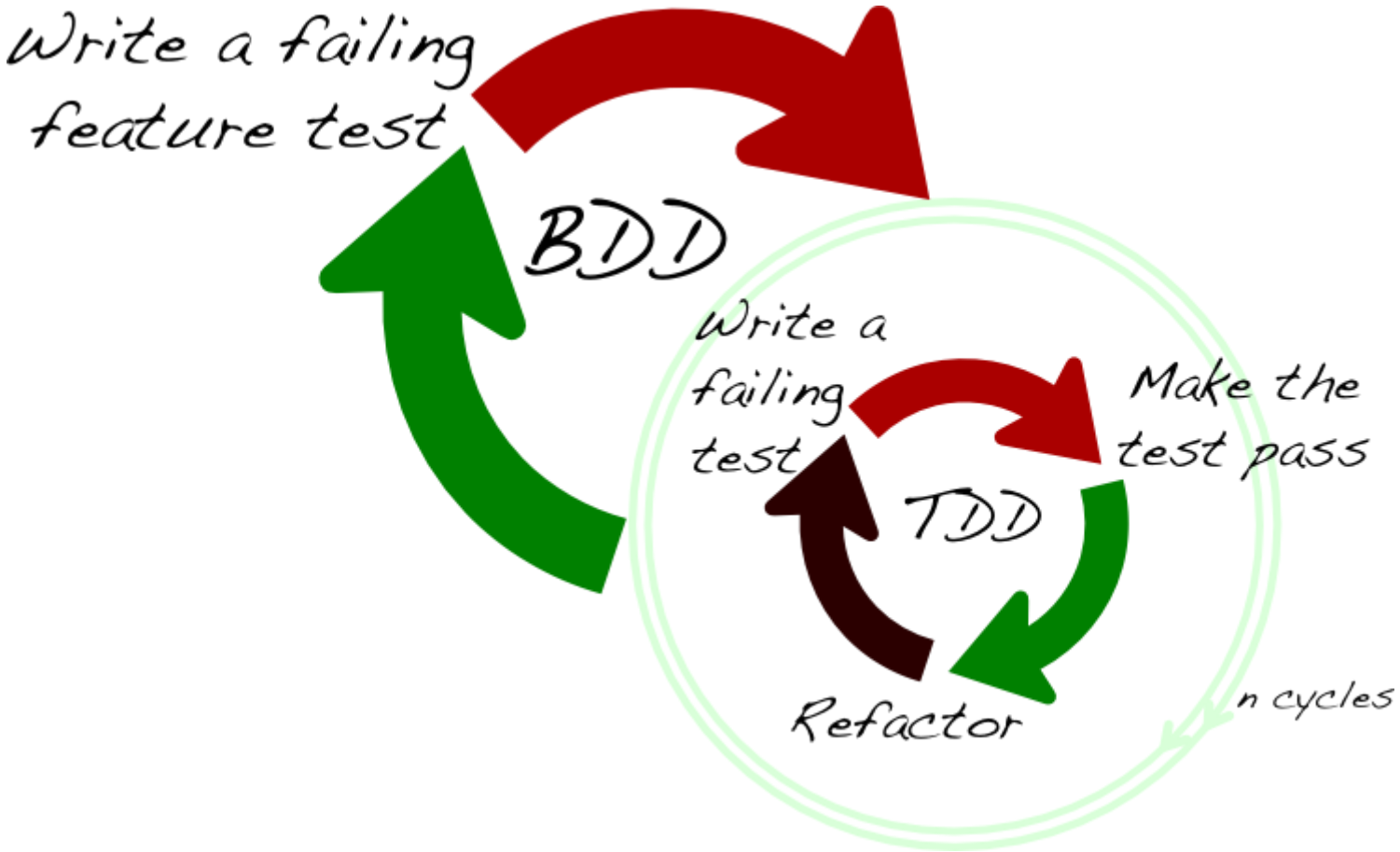
```
public class ServeCoffeeSteps
{
  [Given(@"there are (.*) coffees left in the machine")]
  public void GivenThereAreCoffeesLeftInTheMachine(int p0) {
    ScenarioContext.Current.Pending();
  }

  [Given(@"I have deposited (.*)\$")]
  public void GivenIHaveDeposited(int p0) {
    ScenarioContext.Current.Pending();
  }

  [When(@"I press the coffee button")]
  public void WhenIPressTheCoffeeButton() {
    ScenarioContext.Current.Pending();
  }

  [Then(@"I should be served a coffee")]
  public void ThenIShouldBeServedACoffee() {
    ScenarioContext.Current.Pending();
  }
}
```

The double loop



Test coverage

```
public class ClassToCover
{
    public static bool IsPalindrome(string word)
    {
        if (word == null)
        {
            return false;
        }
        return string.Join("", word.Reverse()) == word;
    }
}
```

```
[TestFixture]
public class CoveringClass
{
    [Test]
    public void Given_Kayak_Then_It_Is_A_Palindrome()
    {
        var result = ClassToCover.IsPalindrome("kayak");
        Assert.That(result, Is.True);
    }
}
```

A 100 % coverage is not a bug-free code