QUALITÉ

R5.A.08

Ludovic Pradel

DÉTAILS PRATIQUES

- R5.A.08 semaine 48-04
- CM : 1
- TD : 6
- TP : 6
- Évaluation : contrôle de connaissance + note de TP (projet sur SAE)

Ressource R5.A.08 Qualité de développement

Informatique > Développement > Qualité de développement

Descriptif détaillé

Objectif

L'objectif de cette ressource est de renforcer les capacités de qualité de développement. Cette ressource permet de choisir et d'implémenter des architectures adaptées aux besoins en anticipant les résultats de diverses métriques.

Savoirs de référence étudiés

– Caractéristiques de qualité (par ex. : robustesse, maintenabilité, portabilité, extensibilité…)
– Techniques d'inspections (par ex. : revue de code, walkthrough…)
– Documentation (par ex. : manuels utilisateurs, formations…)

Prolongements suggérés

– Développement dirigé par les tests
– Développement dirigé par les comportements

Indications de mise en œuvre

Cette ressource est largement identique à la ressource R5.D.07 et peut être mutualisée.

Robustesse | Documentation | Sécurité | Revue de code

Cursus S5 parcours A

Heures totales (30h)	**15h** TD et **15h** TP
programme national	9h TD et 9h TP
adaptation locale SAÉ	2h TD et 2h TP
adaptation locale non fléchée	4h TD et 4h TP
Exemple de contribution aux SAÉ	
S5.A.01 Dév. avancé	2h TD et 2h TP

Coefficients de pondération

UE	*Parcours*	Coeff.
UE 5.1	*parcours A*	7%
UE 5.2	*parcours A*	5%

Compétence 2

Analyser et optimiser des applications

AC 1 Anticiper les résultats de diverses métriques (temps d'exécution, occupation mémoire…)

Compétence 1

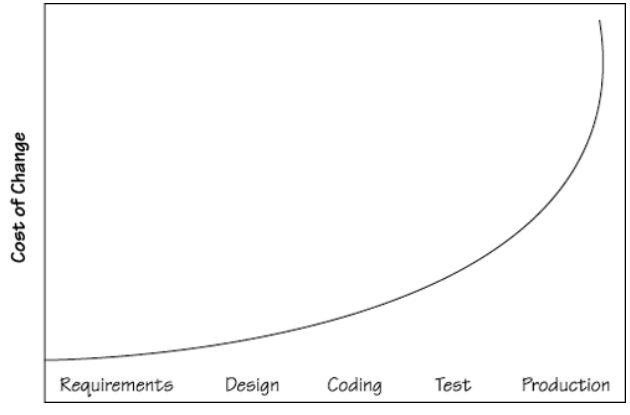
Adapter des applications sur un ensemble de supports (embarqué, web, mobile, IoT…)

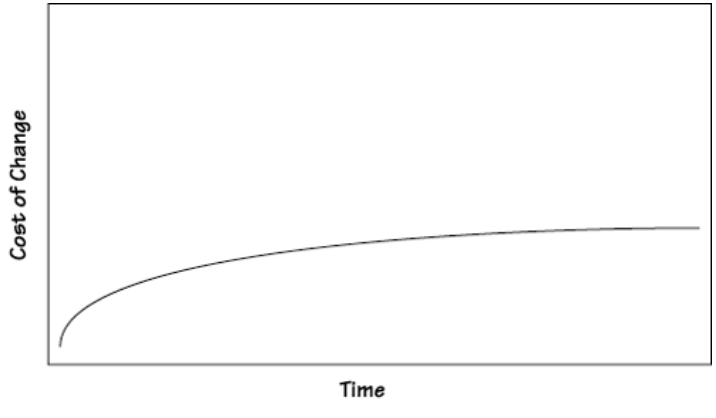
AC 1 Choisir et implémenter les architectures adaptées

C'EST QUOI LA QUALITÉ ?

Un logiciel est de bonne qualité si le coût d'ajout d'une fonctionnalité est constant dans le temps.

C'EST QUOI LA QUALITÉ ?



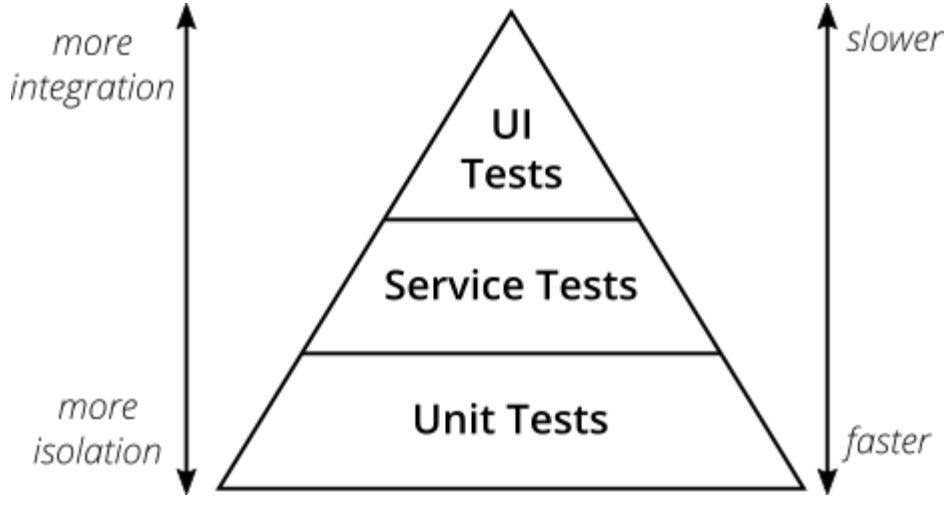


4 RÈGLES DU DESIGN SIMPLE

- *Tests Pass*
- *Express Intent*
- *No Duplication*
- *Small*

Extreme Programming, Kent Beck

TYPES DE TESTS



ACCEPTANCE TESTS

Dans une approche agile :

- Ecrits conjointement
- Basés sur des exemples
- Disponibles dès le début
- Deviennent la spécification
- Lisibles par les personnes du métier
- S'executent rapidement

CRITÈRES D'ACCEPTATION

Formalisme simple d'écriture :

> ***Given*** *some initial context,*
> ***When*** *an event occurs,*
> ***Then*** *ensure some outcomes.*
>
> *Dan North - BDD*

POURQUOI AUTOMATISER ?

feedback continu

non-régression permanente

plus fiable qu'un test manuel

permet aux testeurs d'utiliser au mieux leurs compétences

AUGMENTE LA CONFIANCE

TESTS UNITAIRES

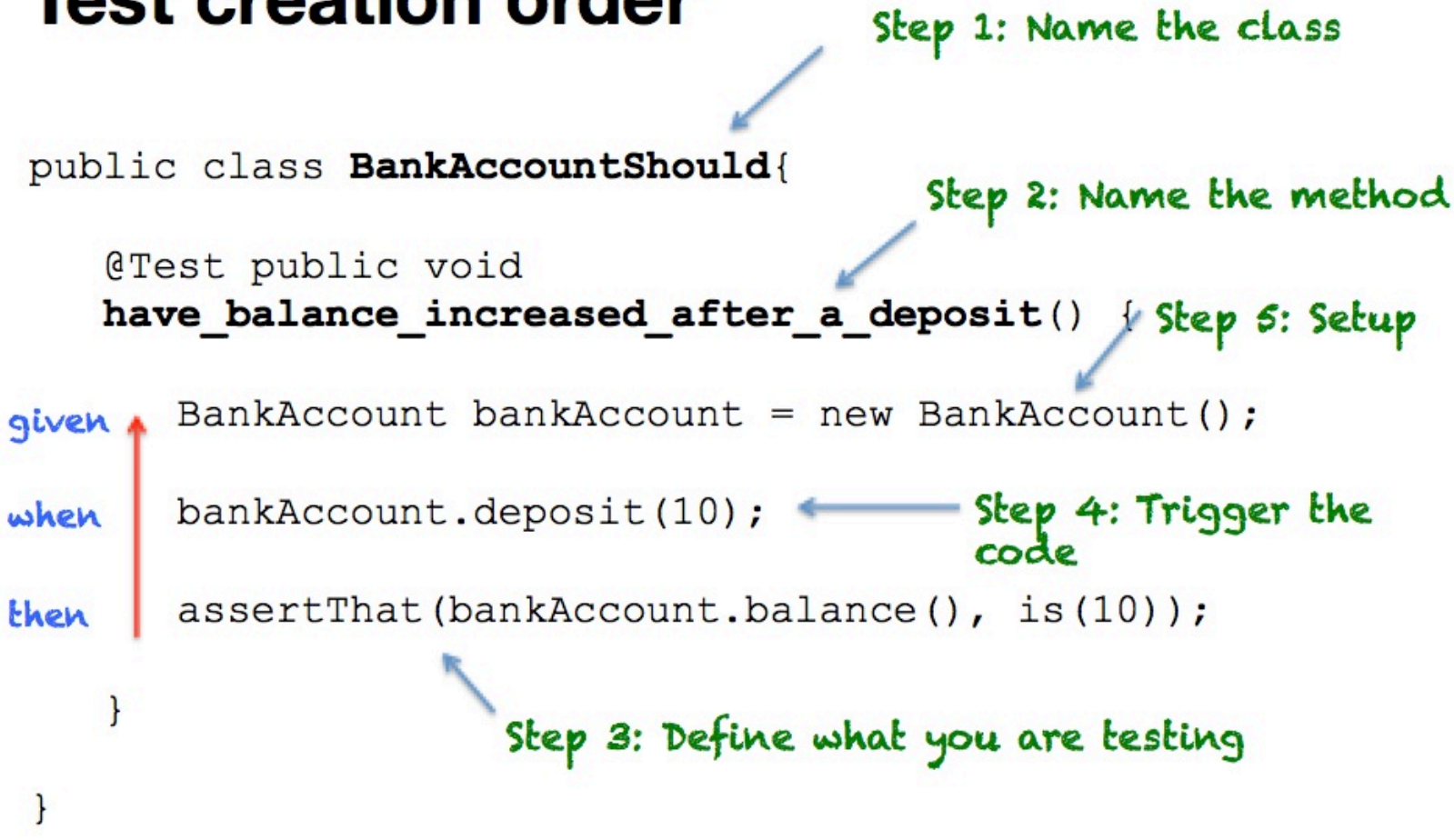
DÉFINITION

un test unitaire est un test qui vérifie un comportement d'un morceau de code isolé du reste.

QUALITÉS D'UN TEST UNITAIRE

- Fast
- Isolated
- Repeatable
- Self-validating
- Timely

Test creation order



LES TESTS APRÈS LE CODE.

LES TESTS APRÈS LE CODE.

- 1. Réflexion sur le design.
- 2. Écriture du code en suivant le design.
- 3. Écriture des tests.

(FAUX) PROBLÈMES

C'est bon ça marche, pas besoin de test.

Je suis à la bourre, et j'ai du fonctionnel plus important. J'écrirai les tests plus tard.

Les tests c'est pour ceux qui savent pas coder.

(GROS) PROBLÈMES

- on peut avoir écrit du code qui n'est pas testable.
- on ne voit jamais les tests échouer
- souvent, les tests sont écrits en étudiant le code.

LES TESTS AVANT LE CODE.

LES TESTS AVANT LE CODE.

1. Réfléchir à ce que je veux faire.
2. Écrire les tests correspondants.
3. Écrire le code qui les fera passer.

L'INTÉRÊT

- au moins, on est sur que les tests seront écrits
- le code sera forcemment testable

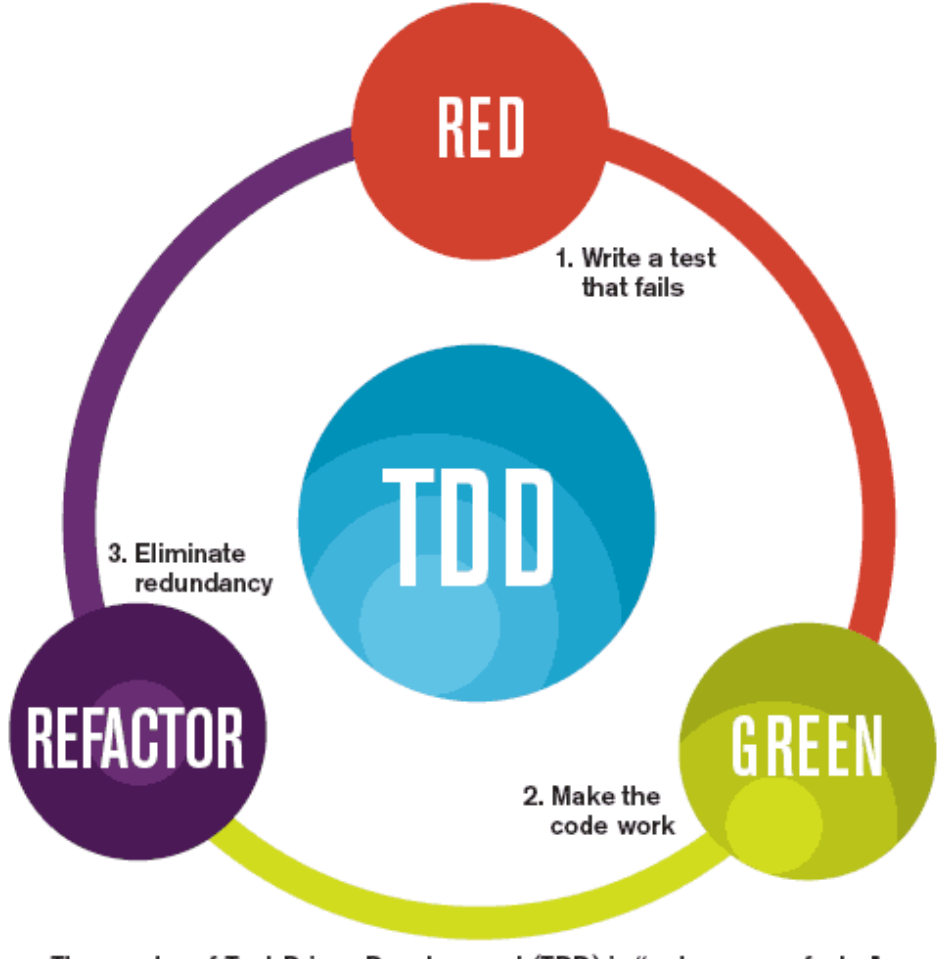
PROBLÈMES

- les tests sont rouges pendant longtemps
- ça impose un big design upfront
- erreur et oubli obligent à presque tout refaire
- le passage du rouge au vert n'est pas toujours net

TDD

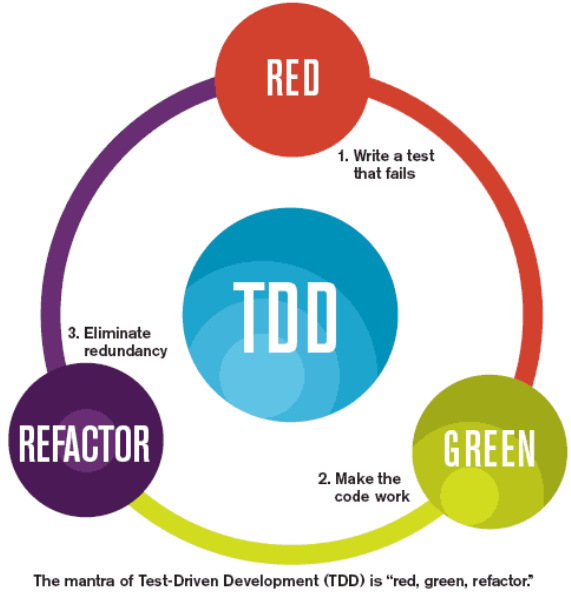
TEST DRIVEN DESIGN ?

l'objectif n'est pas les tests, c'est le design !
la batterie de tests n'est qu'un effet de bord en cadeau



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

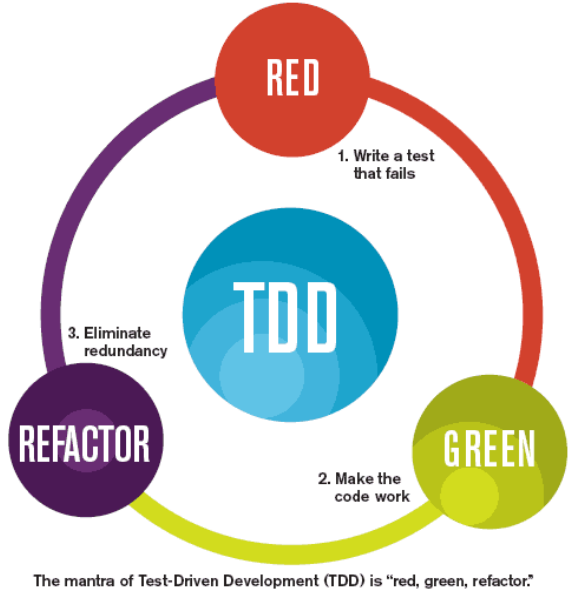
RED



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Quel est le test à écrire pour faire la plus petite étape ?

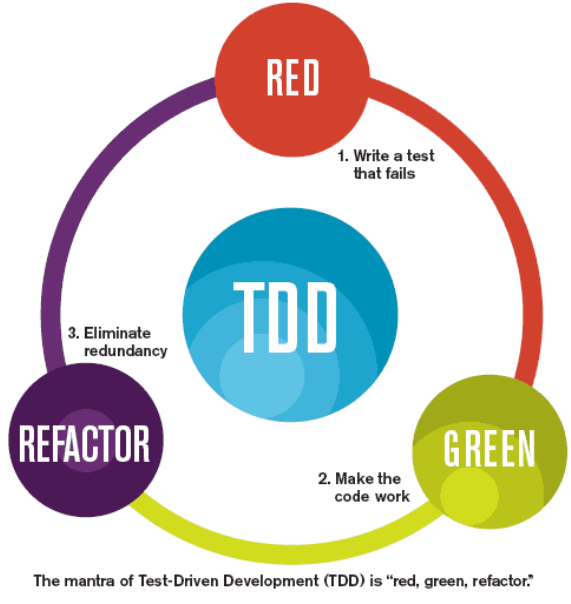
GREEN



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Quel est le plus petit truc à faire pour faire passer le test ?

REFACTOR



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

On nettoie

On fait apparaître des concepts métiers

LEGACY-CODE

- existant ?
- de mauvaise qualité ?
- vieux ?
- on ne sait pas ce qu'il fait ?
- plein de dépendances ?
- Du code sans tests ?

LA SEULE VÉRITÉ EST LE CODE

Les specs ne sont pas à jour et pas fiables

NE RIEN CHANGER AU FONCTIONNEMENT

Un bug suffisamment vieux devient une feature

Toujours demander au PO

POURQUOI REFACTORER ?

Refactor: not because you know the abstraction, but because you want to find it.

Martin Fowler

POURQUOI REFACTORER ?

Le code fonctionne très bien, ne pas refactorer pour le principe.
C'est couteux et risqué.

Pour comprendre le fonctionnement ?

Pour ajouter une feature ?

Pour rembourser la dette ?

PRINCIPAUX REFACTORING

- Rename
- Extract
- Inline
- Move
- Safe delete

VOS OUTILS

- Gestion de version
- Temps (timebox)
- Bloc-notes
- vos IDE

QUAND REFACTORER ?

- 4 règles de design simple
- Objects Calisthenics
- Principes SOLID

4 RÈGLES DU DESIGN SIMPLE

- *Tests Pass*
- *Express Intent*
- *No Duplication*
- *Small*

Extreme Programming, Kent Beck

OBJECT CALISTHENICS

La perfection est atteinte, non pas lorsqu'il n'y a plus rien à ajouter, mais lorsqu'il n'y a plus rien à retirer.

Antoine de Saint-Exupéry

OBJECT CALISTHENICS

- Only One Level Of Indentation Per Method
- Don’t Use The ELSE Keyword
- Wrap All Primitives And Strings
- First Class Collections
- One Dot Per Line
- Don’t Abbreviate
- Keep All Entities Small
- No Classes With More Than Two Instance Variables
- No Getters/Setters/Properties

ONLY ONE LEVEL OF INDENTATION PER METHOD

- Permet de s'assurer qu'une méthode ne fait qu'une seule chose.
- Réduit la taille de vos méthodes, qui seront plus facilement réutilisables.

DON’T USE THE ELSE KEYWORD

- Préférez une ligne d'execution principale avec quelques cas spéciaux.
- Pour les cas complexes, utilisez le polymorphisme.
- Utilisation du pattern Null Object pour exprimer un résultat sans valeur.

WRAP ALL PRIMITIVES AND STRINGS

- Pas de types primitifs en arguments de méthode.
- Les fonctions ne peuvent pas retourner de types primitifs.
- Pour chaque type primitif nécessaire, créez une nouvelle classe.
- C'est ok d'avoir des types primitifs en membre privé d'une classe.

FIRST CLASS COLLECTIONS

- Pas de collections en arguments de méthode.
- Créez une nouvelle classe pour chaque collection.
- Un classe qui contient une collection ne peut contenir aucune autre variable membre.

ONE DOT PER LINE (-> IN PHP)

- Ne pas enchainer les appels de méthodes.
- Dog.Body.Tail.Wag() => Dog.expressHappiness()
- Demander à un objet de faire une action, plutôt que de lui demander sa représentation interne.

DON’T ABBREVIATE

- une abbréviation est souvent déroutante.

KEEP ALL ENTITIES SMALL

- 10 fichiers par package,
- 50 lignes par classe,
- 5 lignes par méthode,
- 2 arguments par méthode,

NO CLASSES WITH MORE THAN TWO INSTANCE VARIABLES

- Lié à la règle : Wrap All Primitives And Strings,
- Pourquoi 2 ?
- Choix arbitraire permettant de découpler vos classes au maximum.

NO GETTERS/SETTERS/PROPERTIES

- Autrement connue sous le nom de : Tell, don't ask.
- Aligné avec les idées de langage orienté objet (messaging)

SOLID

Ces principes ne sont pas des règles à suivre impérativement.

Fiez-vous à votre jugement pour savoir quand les suivre et quand les enfreindre

SOLID

- S - Single-responsiblity principle
- O - Open-closed principle
- L - Liskov substitution principle
- I - Interface segregation principle
- D - Dependency Inversion Principle

S - SINGLE-RESPONSIBLITY PRINCIPLE

A class should have one and only one reason to change, meaning that a class should have only one job.

S - SINGLE-RESPONSIBLITY PRINCIPLE

```
namespace Fohjin.Solid
{
    public class OrderProcessor
    {
        public void Process(Order order)
        {
            if (order.IsValid && Save(order))
            {
                SendConfirmationMessage(order);
            }
        }
        private static bool Save(Order order)
        {
            using (SqlConnection connection = new SqlConnection("something"))
            {
                // Save the order to the database
            }
            return true;
        }
        private static void SendConfirmationMessage(Order order)
        {
            // Send an e-mail to the client
            var name = order.Customer.Name;
            var email = order.Customer.Email;
        }
    }
}
```

S - SINGLE-RESPONSIBLITY PRINCIPLE

```
namespace Fohjin.Solid
{
    public class OrderProcessor {
        public void Process(Order order) {
            if (order.IsValid && new Repository().Save(order))
            {
                new MailSender().SendConfirmationMessage(order);
            }
        }
    }
    public class MailSender {
        public void SendConfirmationMessage(Order order) {
            // Send an e-mail to the client
            var name = order.Customer.Name;
            var email = order.Customer.Email;
        }
    }
    public class Repository {
        public bool Save(Order order) {
            using (SqlConnection connection = new SqlConnection("something"))
            {
                // Save the order to the database
            }
            return true;
        }
    }
}
```

D - DEPENDENCY INVERSION PRINCIPLE

one should "depend upon abstractions, [not] concretions."

A. High level modules should not depend upon low level modules. Both should depend upon abstractions.
B. Abstractions should not depend upon details. Details should depend upon abstractions.

D - DEPENDENCY INVERSION PRINCIPLE

```
namespace Fohjin.Solid
{
    public class OrderProcessor {
        public void Process(Order order) {
            if (order.IsValid && new Repository().Save(order))
            {
                new MailSender().SendConfirmationMessage(order);
            }
        }
    }
    public class MailSender {
        public void SendConfirmationMessage(Order order) {
            // Send an e-mail to the client
            var name = order.Customer.Name;
            var email = order.Customer.Email;
        }
    }
    public class Repository {
        public bool Save(Order order) {
            using (SqlConnection connection = new SqlConnection("something"))
            {
                // Save the order to the database
            }
            return true;
        }
    }
}
```

D - DEPENDENCY INVERSION PRINCIPLE

```
namespace Fohjin.Solid
{
    public class OrderProcessor
    {
        private readonly IRepository _repository;
        private readonly IMailSender _mailSender;

        public OrderProcessor(IRepository repository, IMailSender mailSender)
        {
            _repository = repository;
            _mailSender = mailSender;
        }

        public void Process(Order order)
        {
            if (order.IsValid && _repository.Save(order))
            {
                _mailSender.SendConfirmationMessage(order);
            }
        }
    }
}
```

O - OPEN-CLOSED PRINCIPLE

Objects or entities should be open for extension, but closed for modification.

O - OPEN-CLOSED PRINCIPLE

```
namespace Fohjin.Solid
{
    public class OrderProcessor {
        public void Process(Order order) {
            if (order.IsValid && new Repository().Save(order))
            {
                new MailSender().SendConfirmationMessage(order);
            }
        }
    }
    public class MailSender {
        public void SendConfirmationMessage(Order order) {
            // Send an e-mail to the client
            var name = order.Customer.Name;
            var email = order.Customer.Email;
        }
    }
    public class Repository {
        public bool Save(Order order) {
            using (SqlConnection connection = new SqlConnection("something"))
            {
                // Save the order to the database
            }
            return true;
        }
    }
}
```

O - OPEN-CLOSED PRINCIPLE

```
namespace Fohjin.Solid
{
    public class OrderProcessor
    {
        private readonly IRepository _repository;
        private readonly IMailSender _mailSender;

        public OrderProcessor(IRepository repository, IMailSender mailSender)
        {
            _repository = repository;
            _mailSender = mailSender;
        }

        public void Process(Order order)
        {
            if (order.IsValid && _repository.Save(order))
            {
                _mailSender.SendConfirmationMessage(order);
            }
        }
    }
}
```

L - LISKOV SUBSTITUTION PRINCIPLE

Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.

Barbara Liskov & Jeannette Wing

objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.

L - LISKOV SUBSTITUTION PRINCIPLE

```
namespace Fohjin.Solid
{
    public class Order {
        private readonly List<Item> _items = new List<Item>();
        public bool IsValid { get { return CheckIsValid(); } }

        private bool CheckIsValid() {
            var isValid = true;
            _items.ForEach(item => {
               if (!item.IsInStock) isValid = false;
            });
            return isValid;
        }
    }
    public class PriorityOrder : Order
    {
        private readonly List<Item> _items = new List<Item>();
        public new bool IsValid { get { return AreItemsInStock(); } }

        private bool AreItemsInStock()
        {
            _items.ForEach(item => {
                if (!item.IsInStock) throw new Exception("No items in stock");
            });
            return true;
        }
    }
}
```

L - LISKOV SUBSTITUTION PRINCIPLE

```
namespace Fohjin.Solid
{
    public class Rectangle
    {
        public double Width { get; set; }
        public double Height { get; set; }
    }
    public class Square : Rectangle
    {
        public new double Width { get; set; }
        public new double Height
        {
            get { return Width; }
            set { Width = value; }
        }
    }
    public class SomeOtherClassMakingAssumptions
    {
        public void SurfaceTest(Rectangle rectangle)
        {
            rectangle.Width = 4;
            rectangle.Height = 2;
            Assert.That(rectangle.Width * rectangle.Height, Is.EqualTo(8));
        }
    }
}
```

I - INTERFACE SEGREGATION PRINCIPLE

Clients should not be forced to depend upon interfaces that they do not use.

BDD

OBJECTIFS

- meilleure adéquation avec les besoins du client
- faire intervenir le client tôt dans le processus de test

COMMENT FAIRE ?

#1 : DES FEATURES

```
Feature: Is it Friday yet?
  Everybody wants to know when it's Friday

  Scenario: Sunday isn't Friday
    Given today is Sunday
    When I ask whether it's Friday yet
    Then I should be told "Nope"
```

COMMENT FAIRE ?

#2 : UNE GRAMMAIRE

```
public class StepDefinitions {
    private String today;
    private String actualAnswer;

    @Given("today is Sunday")
    public void today_is_Sunday() {
        today = "Sunday";
    }

    @When("I ask whether it's Friday yet")
    public void i_ask_whether_it_s_Friday_yet() {
        actualAnswer = IsItFriday.isItFriday(today);
    }

    @Then("I should be told {string}")
    public void i_should_be_told(String expectedAnswer) {
        assertThat(actualAnswer).isEqualTo(expectedAnswer);
    }
}
```

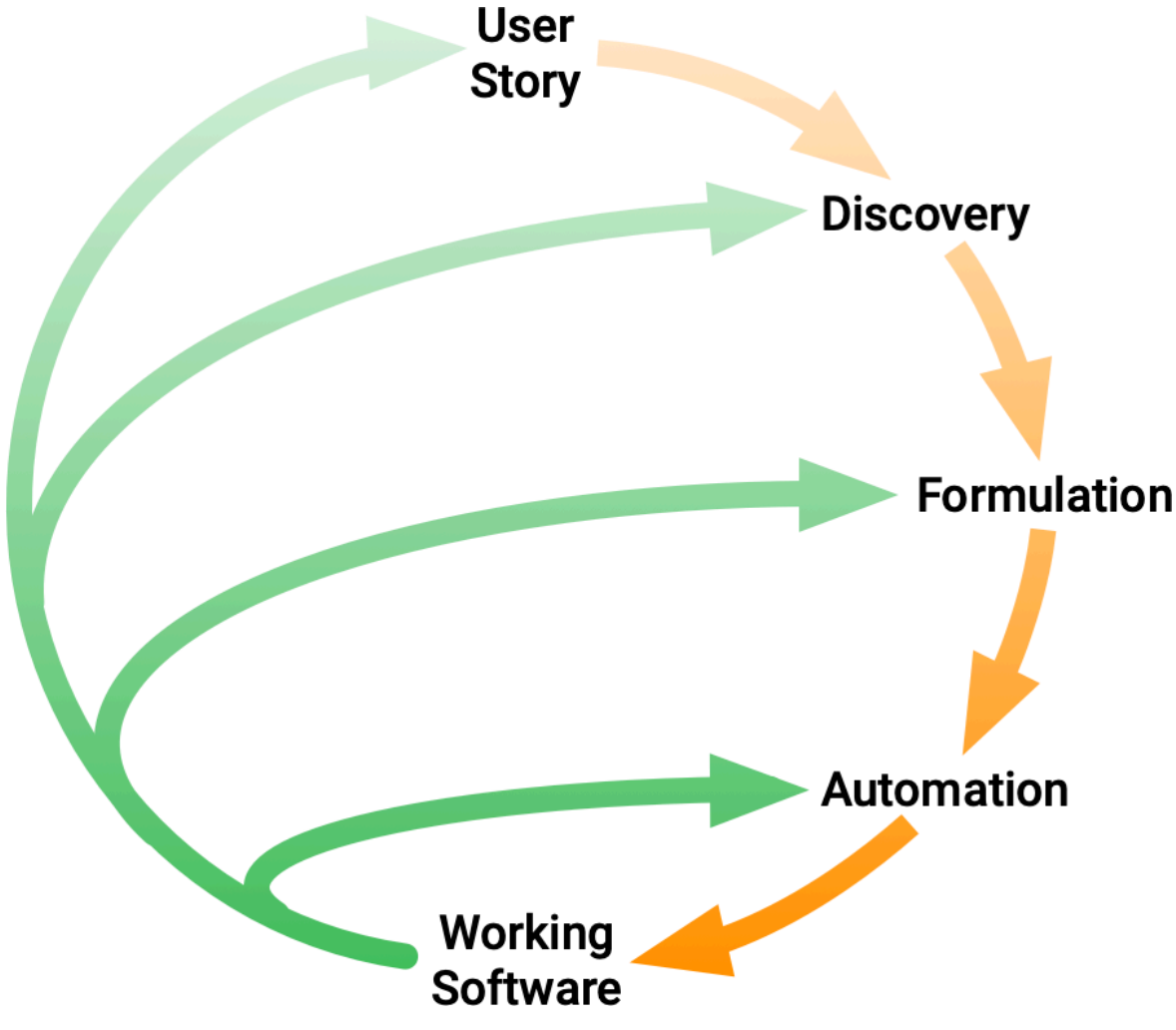
COMMENT FAIRE ?

#3 : DU CODE DE PROD

```
class IsItFriday {
    static String isItFriday(String today) {
        return null;
    }
}
```

BDD ?

Une manière d'écrire des tests



GHERKIN ?

Un langage permettant d'écrire des scénarios

```
Scenario: Discount of 20% for 4 different books
Given A basket
When I add a book of volume 1 to basket
    And I add a book of volume 2 to basket
    And I add a book of volume 2 to basket
    And I add a book of volume 3 to basket
    And I add a book of volume 3 to basket
Then The basket price is 25.60 euros
```

CUCUMBER ?

Une librairie disponible dans plusieurs langages de programmation