

Implémenter le design physique d'une application

Module 477

Cours 2

Bienvenue à la session 2019

Hans W. Schwendimann

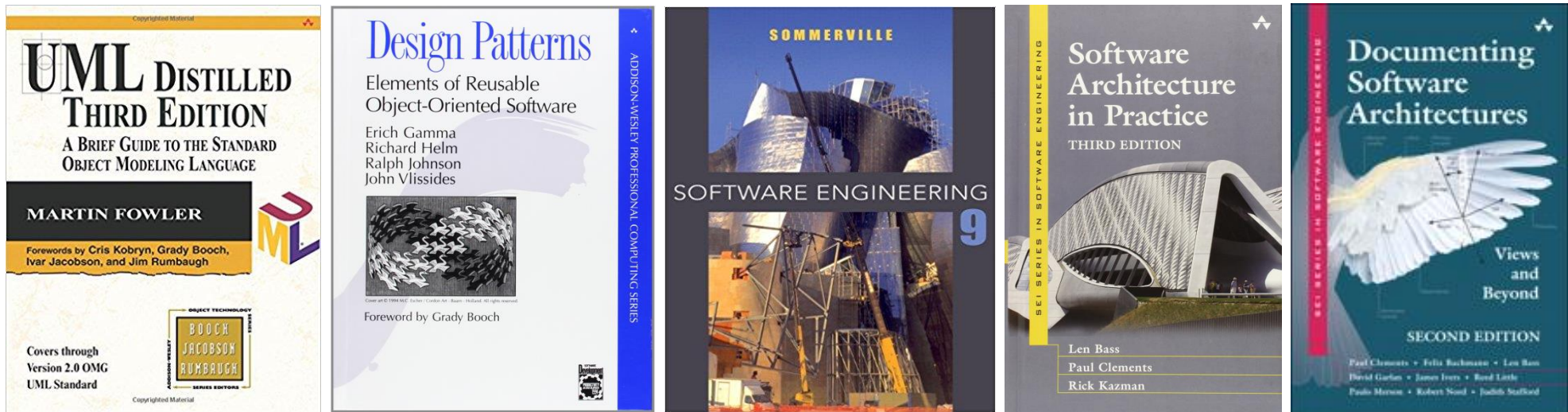
hws@idec.ch

12 périodes (4 soirées) de cours (salle 3)

Horaires : 18h30-21h15 – Pause : 20h05-20h25

Fin de la session : le mardi 26 mars 2019

Bibliographie et références



Références :

- [The Unified Modeling Language Diagrams](#)
- [UML 2 - De l'apprentissage à la pratique](#)
- [Practical UML: A Hands-On Introduction for Developers](#)
- https://fr.wikipedia.org/wiki/Architecture_logicielle

Déroulement du module

Planning

Date	Sujet
Mardi 05 mars 2019	- Introduction aux Test unitaires (TDD) - Framework
Mardi 12 mars 2019	- Introduction aux «SOLID Principles» - Etude de cas (suite)
Mardi 19 mars 2019	- Etude de cas (suite)
Mardi 26 mars 2019	- Etude de cas (suite et fin)



Thème : SOLID Principles

Mardi 12 mars 2019

Planning

- Présentation
 - Présentation et objectifs
 - Rappels cours 1
- Introduction aux SOLID Principles
 - Single responsibility principle
 - Open/closed principle
 - Liskov substitution principle
 - Interface segregation principle
 - Dependency inversion principle (DIP)
- Etude de cas (continuation)
 - Les entités du domaine
- ✓ Pause 20h05 – 20h25
- Etude de cas (continuation)
 - Les entités du domaine
- Devoirs et feed-back de la soirée

Introduction au design physique d'une application

Module 477– Implémenter le design physique d'une application

Objectif :

- ✓ Développer les artefacts du design orienté objet d'une application à partir de l'analyse jusqu'à son implémentation physique



SOLID Principales

Introduction aux SOLID Principles

Introduction aux SOLID Principles

Introduction aux SOLID Principles

«Le développement de logiciel n'est pas un jeu de Jenga»



SOLID

Software development is not a Jenga game.

Introduction aux SOLID Principles

Introduction aux SOLID Principles

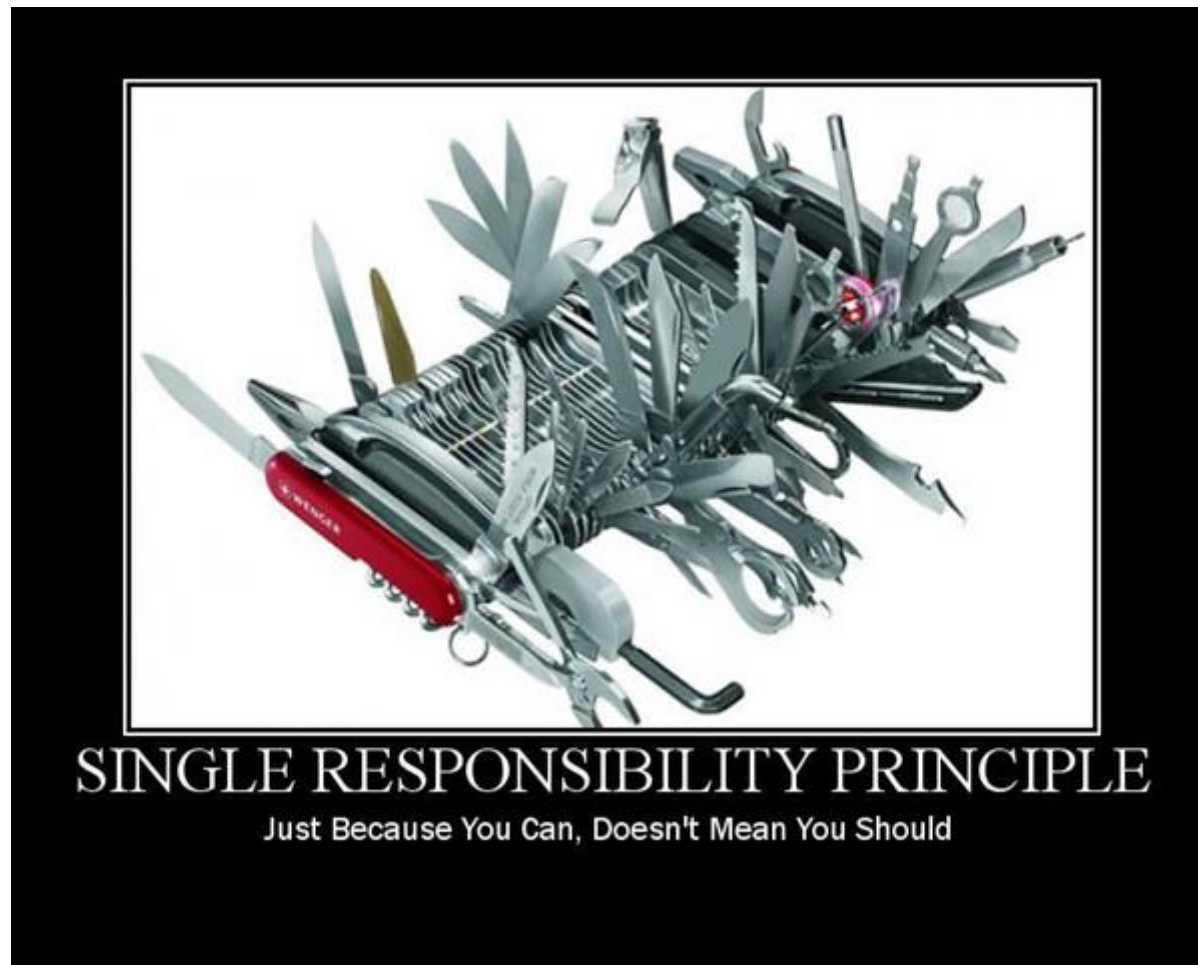
«Le développement de logiciel n'est pas un jeu de Jenga»

- Au début de la phase de la conception, le système semble beau, élégant, pur...
- ...mais ensuite, pendant le développement, on a des hacks, des implémentations vites faites et négligentes pour respecter les délais.
- Le logiciel commence à pourrir!
- Pour s'assurer de maintenir la qualité de la conception, on doit se baser sur des principes SOLID!
 - Introduits par Uncle Bob (Robert C. Martin)
- Quels sont ces principes?

SOLID – Single Responsibility Principle

Introduction aux SOLID Principles

«Parce que tu peux ne signifie pas que tu dois»



SOLID – Single Responsibility Principle

Introduction aux SOLID Principles

«Parce que tu peux ne signifie pas que tu dois»

- Une classe doit avoir une seule *responsabilité*!
- ...mais qu'est-ce que ça veut dire?
- Une classe ne devrait avoir qu'une seule raison de changer
- Est-il possible de prévoir les changements d'une classe?

```

public class Book
{
    public string getTitle()
    {
        return "A Great Book";
    }

    public string getAuthor()
    {
        return "John Doe";
    }

    public string getCurrentPage()
    {
        return "current page content";
    }

    public void printCurrentPage()
    {
        // print page...
    }
}
    
```



```

static void Main(string[] args)
{
    Book book = new Book();

    IPrinter printer = new HtmlPrinter();

    printer.printPage( book.getCurrentPage() );
}
    
```

```

public interface IPrinter
{
    void printPage(Page page);
}

public class PlainTextPrinter : IPrinter
{
    public void printPage(Page page)
    {
        // print a TextPage
    }
}

public class HtmlPrinter : IPrinter
{
    public void printPage(Page page)
    {
        // print a HtmlPage
    }
}
    
```

SOLID – Open / Closed Principle

Introduction aux SOLID Principles

«La chirurgie à cœur ouvert n'est pas nécessaire pour mettre un manteau»



SOLID – Open / Closed Principle

Introduction aux SOLID Principles

«La chirurgie à cœur ouvert n'est pas nécessaire pour mettre un manteau»

- Les entités logicielles doivent être ouvertes aux extensions, mais fermées aux modifications.
- Pensez toujours : « qu'est-ce qui va se passer si l'entité change? »

```
public class Rectangle
{
    public double Width { get; set; }
    public double Height { get; set; }
}

public class AreaCalculator
{
    public double Area(object[] shapes)
    {
        double area = 0;
        foreach (var shape in shapes)
        {
            if (shape is Rectangle)
            {
                Rectangle rectangle = (Rectangle)shape;
                area += rectangle.Width * rectangle.Height;
            }
            else
            {
                Circle circle = (Circle)shape;
                area += circle.Radius * circle.Radius * Math.PI;
            }
        }

        return area;
    }
}
```

```
public class AreaCalculator
{
    public double Area(Shape[] shapes)
    {
        double area = 0;
        foreach (var shape in shapes)
        {
            area += shape.Area();
        }

        return area;
    }
}
```

```
public abstract class Shape
{
    public abstract double Area();
}

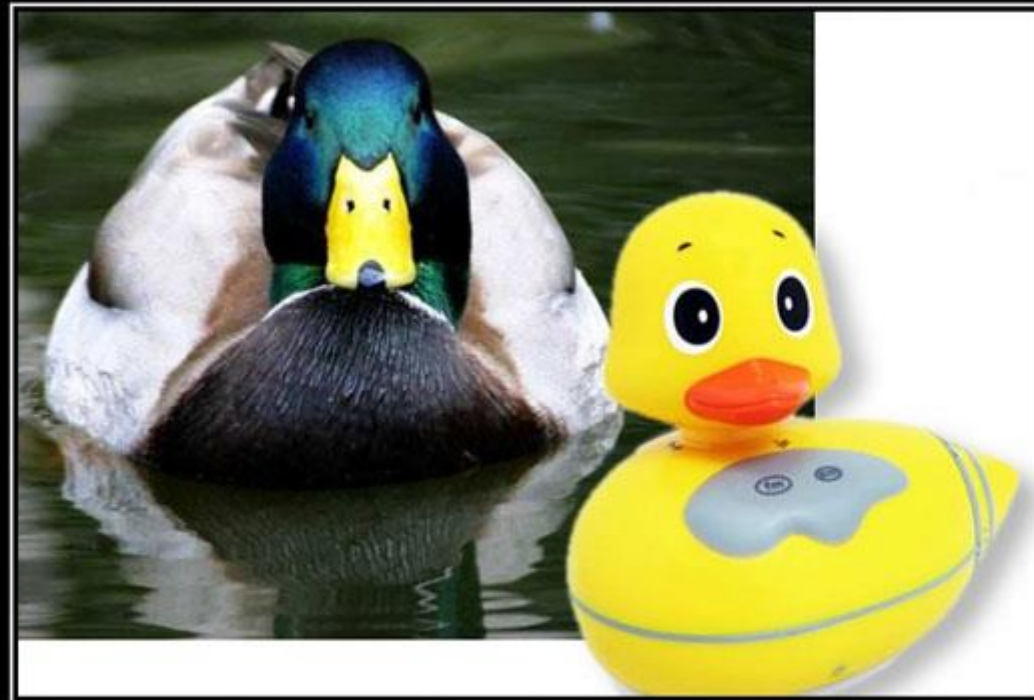
public class Rectangle : Shape
{
    public double Width { get; set; }
    public double Height { get; set; }
    public override double Area()
    {
        return Width * Height;
    }
}

public class Circle : Shape
{
    public double Radius { get; set; }
    public override double Area()
    {
        return Radius * Radius * Math.PI;
    }
}
```

SOLID – Liskov Substitution Principle

Introduction aux SOLID Principles

«Si cela ressemble à un canard, sonne comme un canard, mais a besoin de piles, vous avez probablement la mauvaise abstraction»



LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

SOLID – Liskov Substitution Principle

Introduction aux SOLID Principles

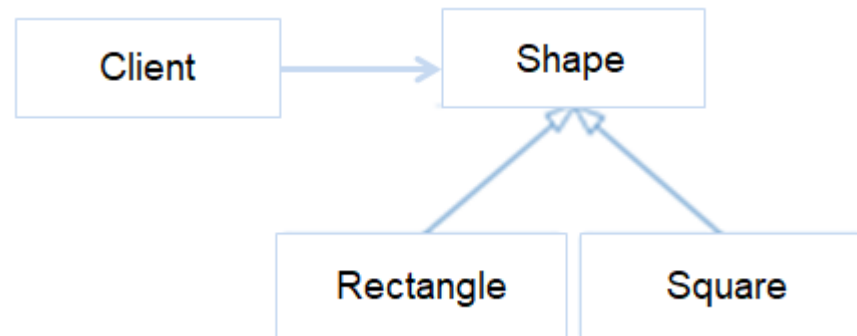
«Si cela ressemble à un canard, sonne comme un canard, mais a besoin de piles, vous avez probablement la mauvaise abstraction»

- Les fonctions qui utilisent des références à des classes de base doivent pouvoir utiliser des objets de classes dérivées sans le savoir
- Toutes les méthodes qui prennent un paramètre de type « Shape », peuvent accepter un argument de type « Rectangle ».

```

[TestMethod]
public void TwentyFourfor4x6Rectangleand9for3x3Square()
{
    var shapes = new List<Shape>{
        new Rectangle{Height=4,Width=6},
        new Square{Sides=3}
    };
    var areas = new List<int>();

    foreach (Shape shape in shapes)
    {
        areas.Add(shape.Area());
    }
    Assert.AreEqual(24, areas[0]);
    Assert.AreEqual(9, areas[1]);
}
    
```



```

public abstract class Shape
{
    public abstract int Area();
}
    
```

```

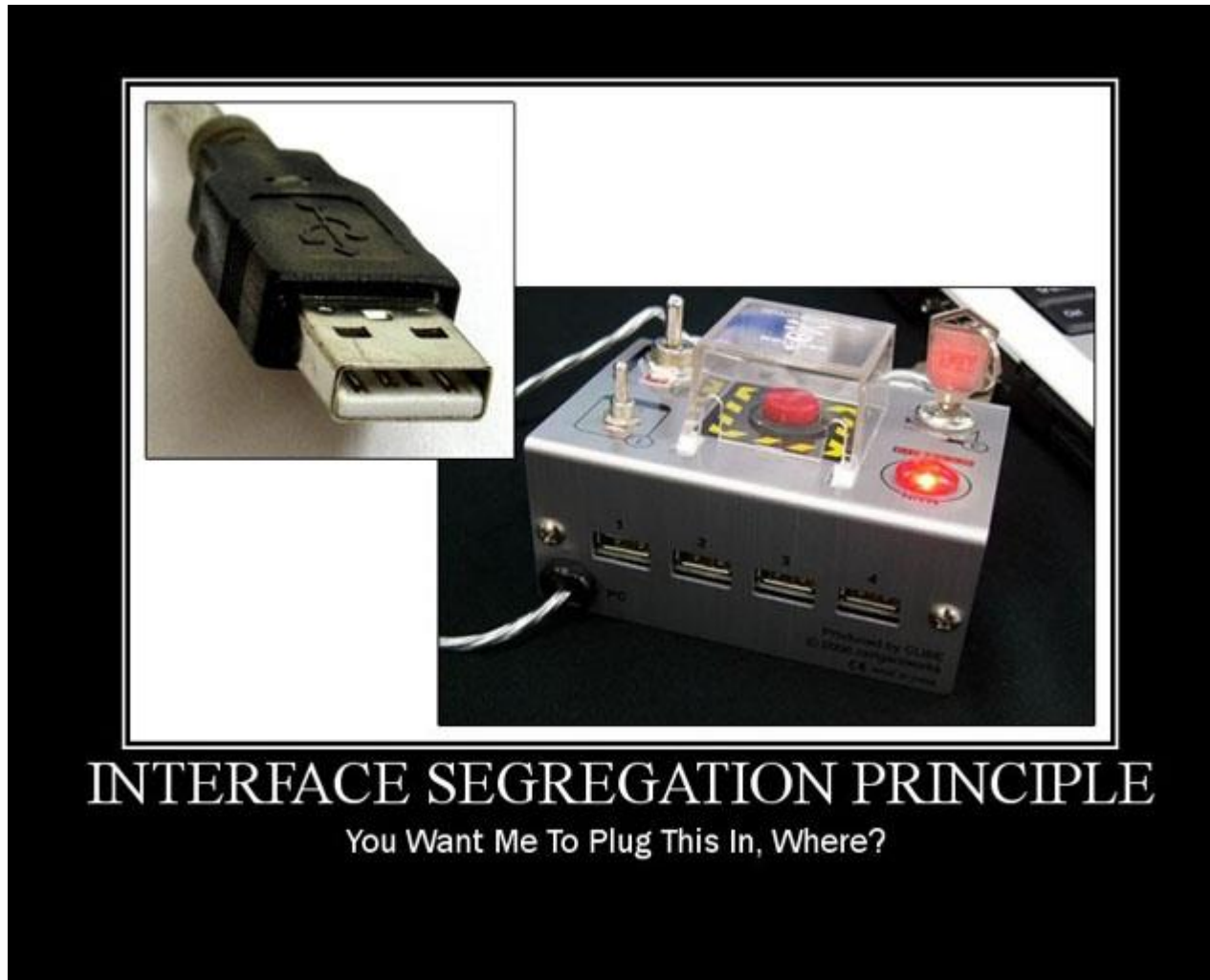
public class Square : Shape
{
    public int Sides;
    public override int Area()
    {
        return Sides * Sides;
    }
}

public class Rectangle : Shape
{
    public int Height { get; set; }
    public int Width { get; set; }
    public override int Area()
    {
        return Height * Width;
    }
}
    
```

SOLID – Interface Segregation Principle

Introduction aux SOLID Principles

«Tu veux que je mette ça où ?»

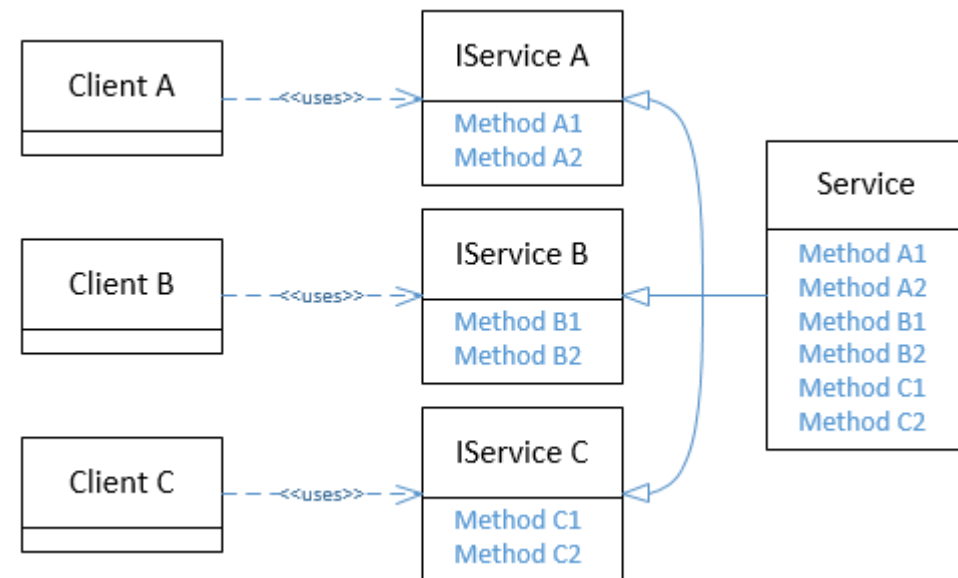
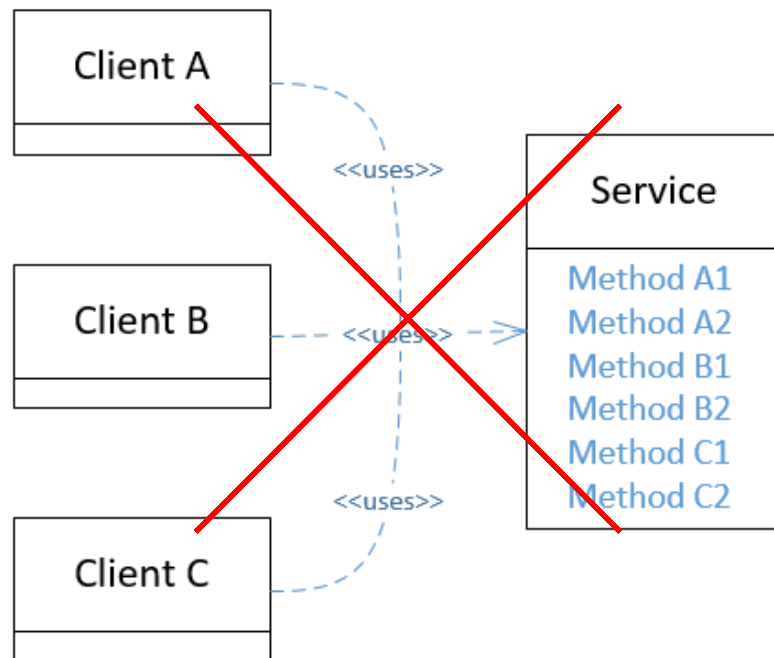


SOLID – Interface Segregation Principle

Introduction aux SOLID Principles

«Tu veux que je mette ça où ?»

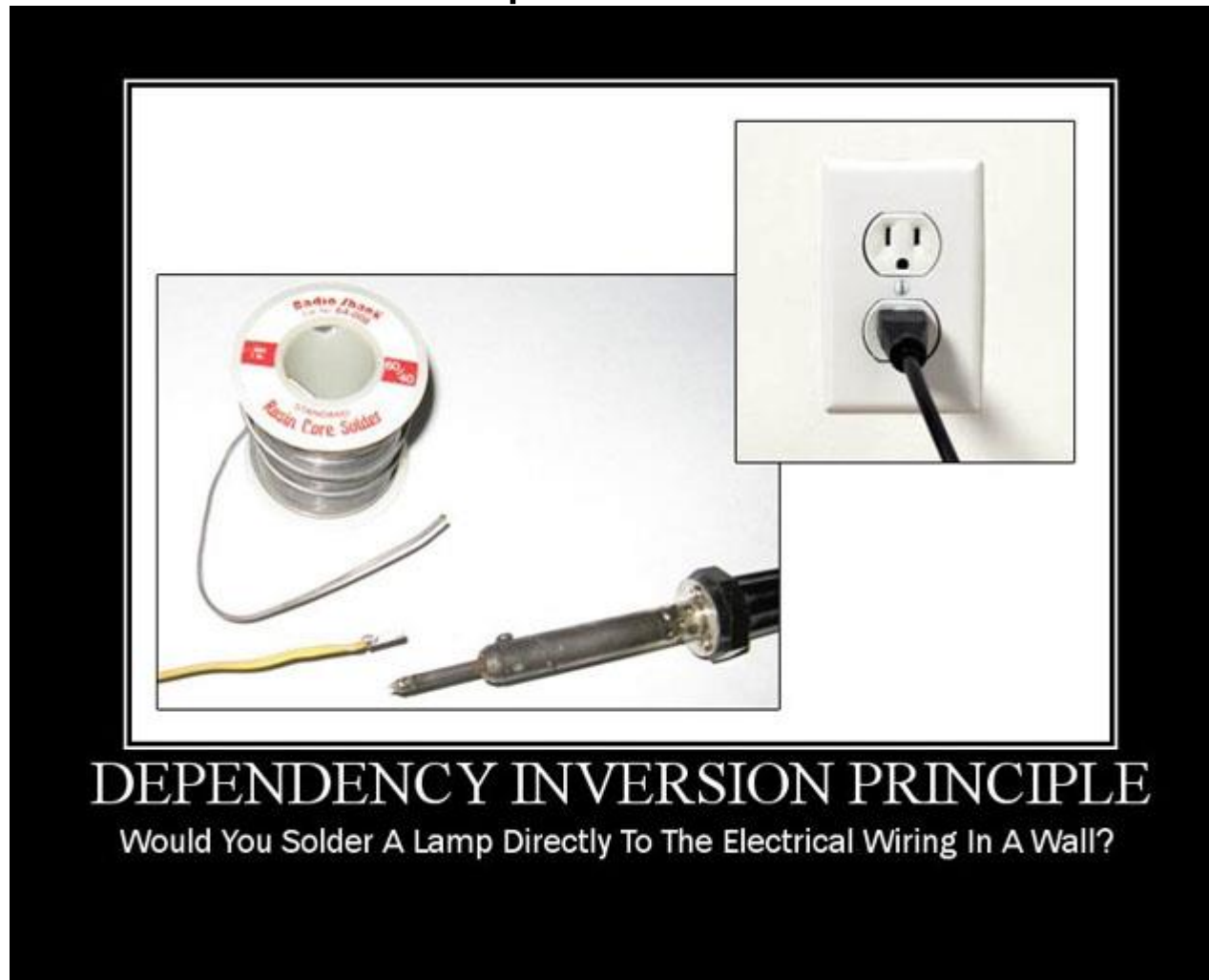
- Les clients ne devraient pas être forcés de dépendre de méthodes qu'ils n'utilisent pas...
- De nombreuses interfaces spécifiques aux clients valent mieux qu'une seule interface...



SOLID – Dependency Inversion Principle

Introduction aux SOLID Principles

«Souderiez-vous une lampe directement sur le câblage électrique dans le mur ?»



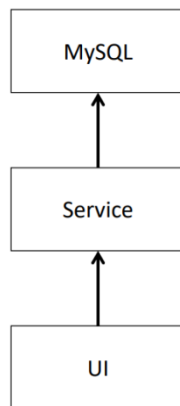
SOLID – Dependency Inversion Principle

Introduction aux SOLID Principles

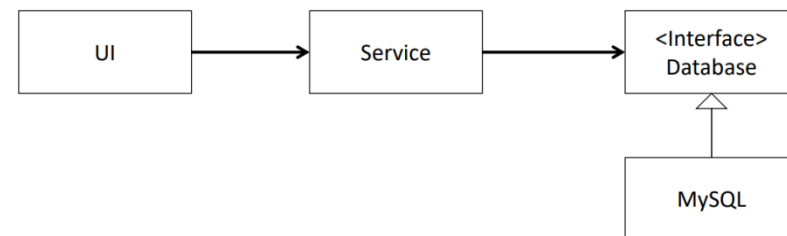
«Souderiez-vous une lampe directement sur le câblage électrique dans le mur ?»

- Dépend des abstractions. Ne dépend pas de classes concrètes...
- Le **Saint Graal** de l'architecture logicielle!
- Si on veut changer la base de données, combien de classes doit-on changer?
- Si on dépend de bibliothèques logicielles, notre système est enfermé dans un langage spécifique. (DIP violé)

DIP violé?



DIP garanti!

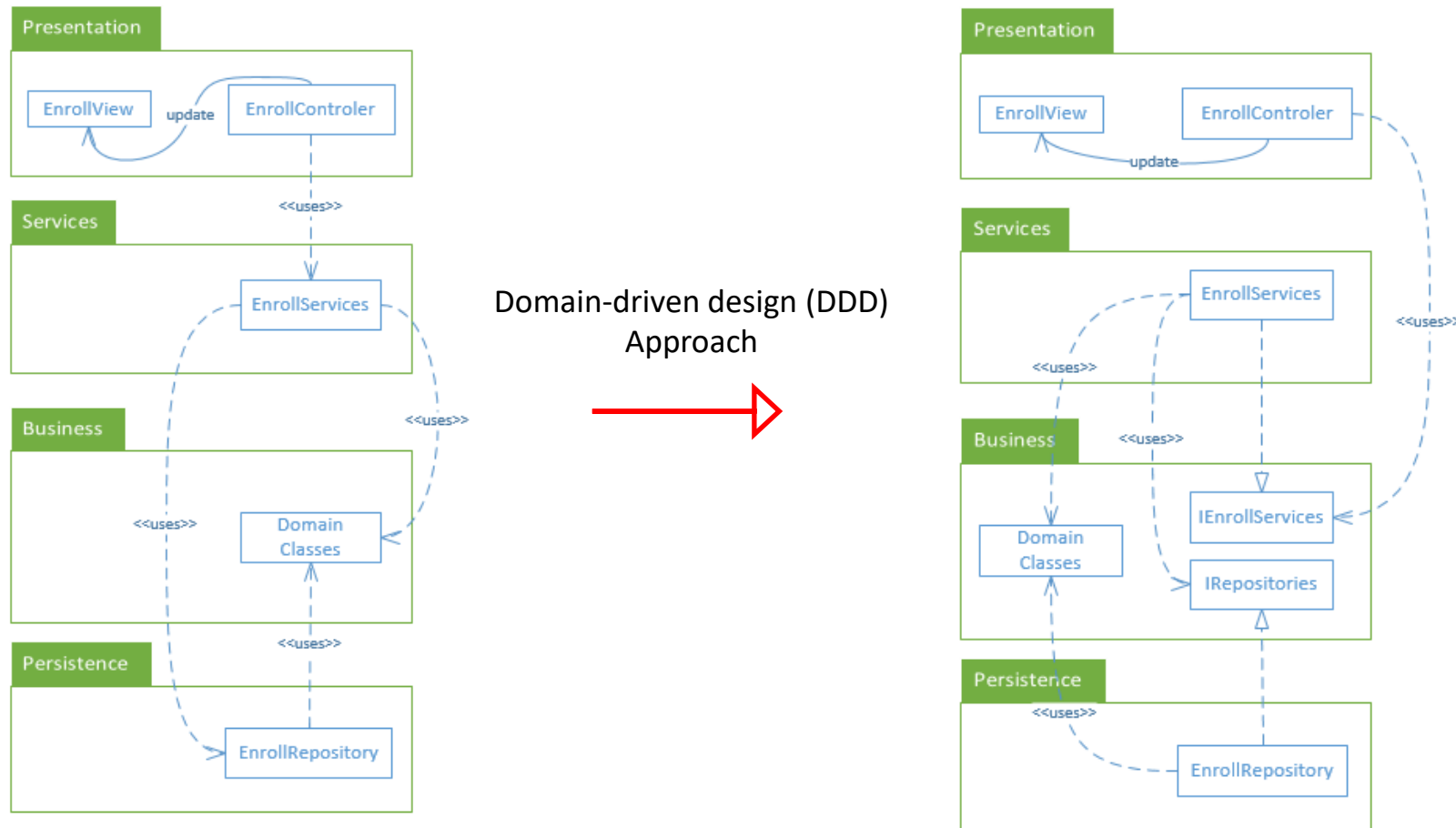


SOLID – Dependency Inversion Principle

Introduction aux SOLID Principles

«Souderiez-vous une lampe directement sur le câblage électrique dans le mur ?»

- Dépend des abstractions. Ne dépend pas de classes concrètes...
- Le **Saint Graal** de l'architecture logicielle!



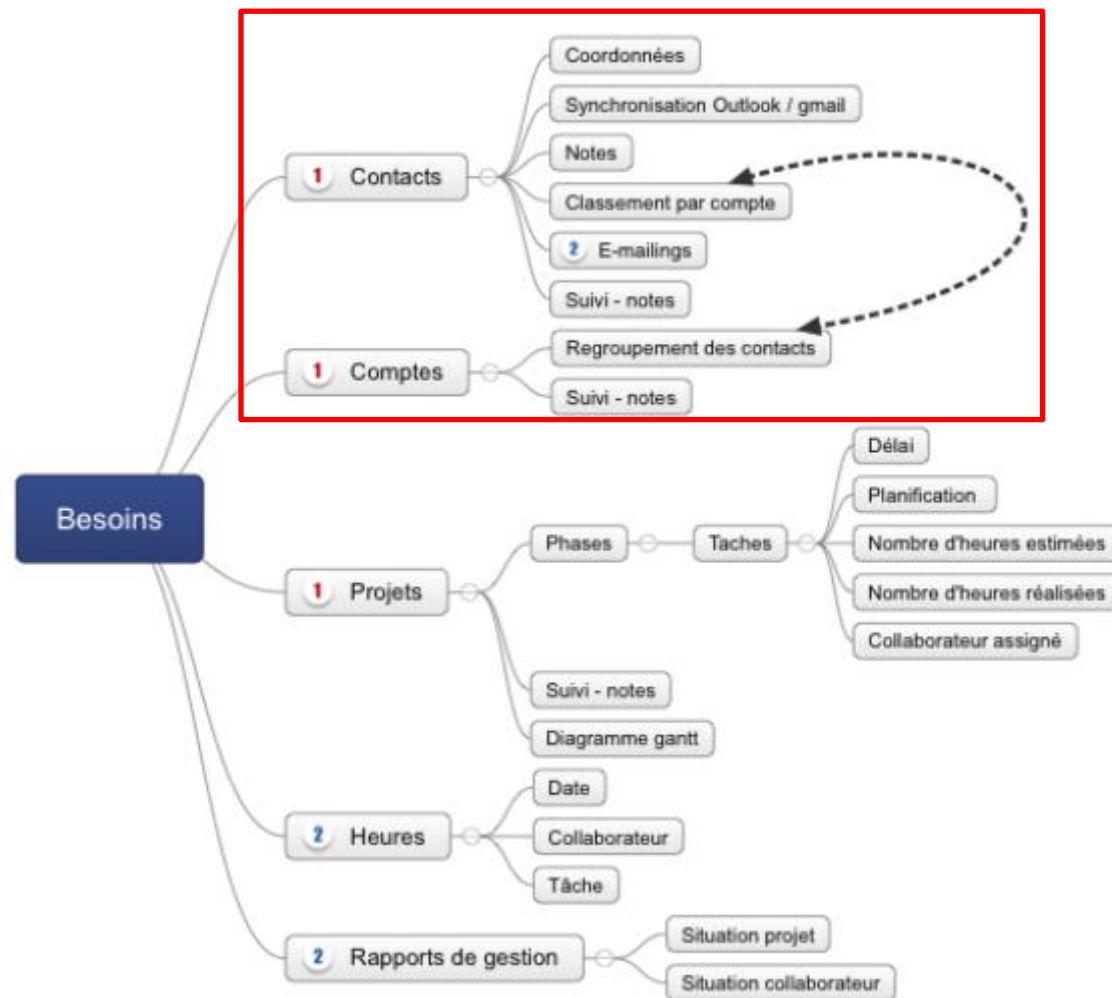
Introduction aux Test unitaires (TDD)

Etude de cas AcmeSystem

Besoins

Etude de cas AcmeSystem

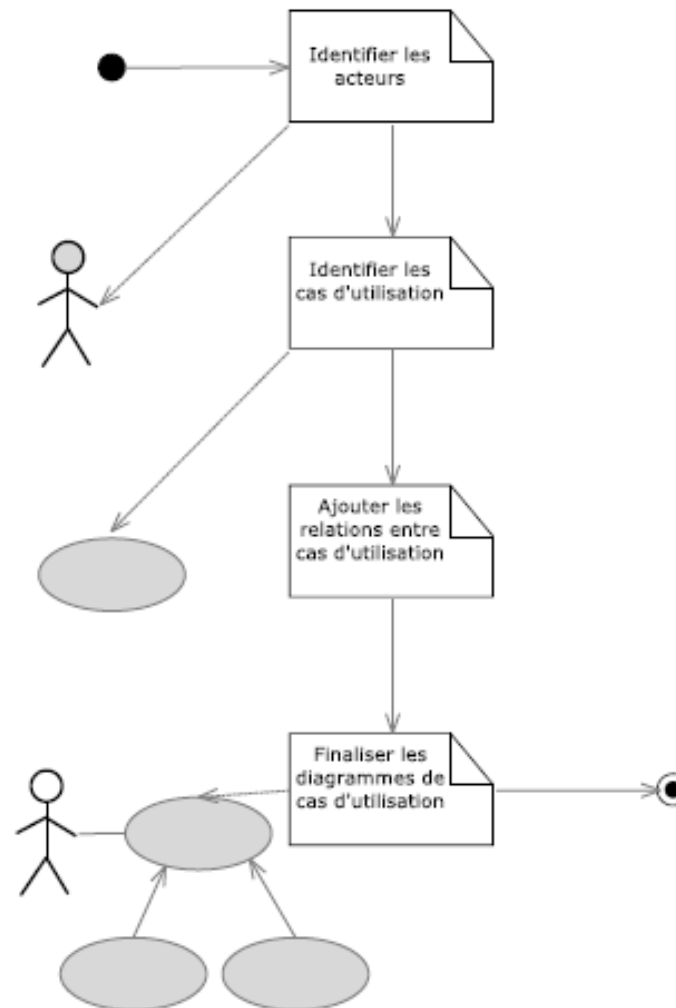
Schéma heuristique des besoins



Besoins

Etude de cas AcmeSystem

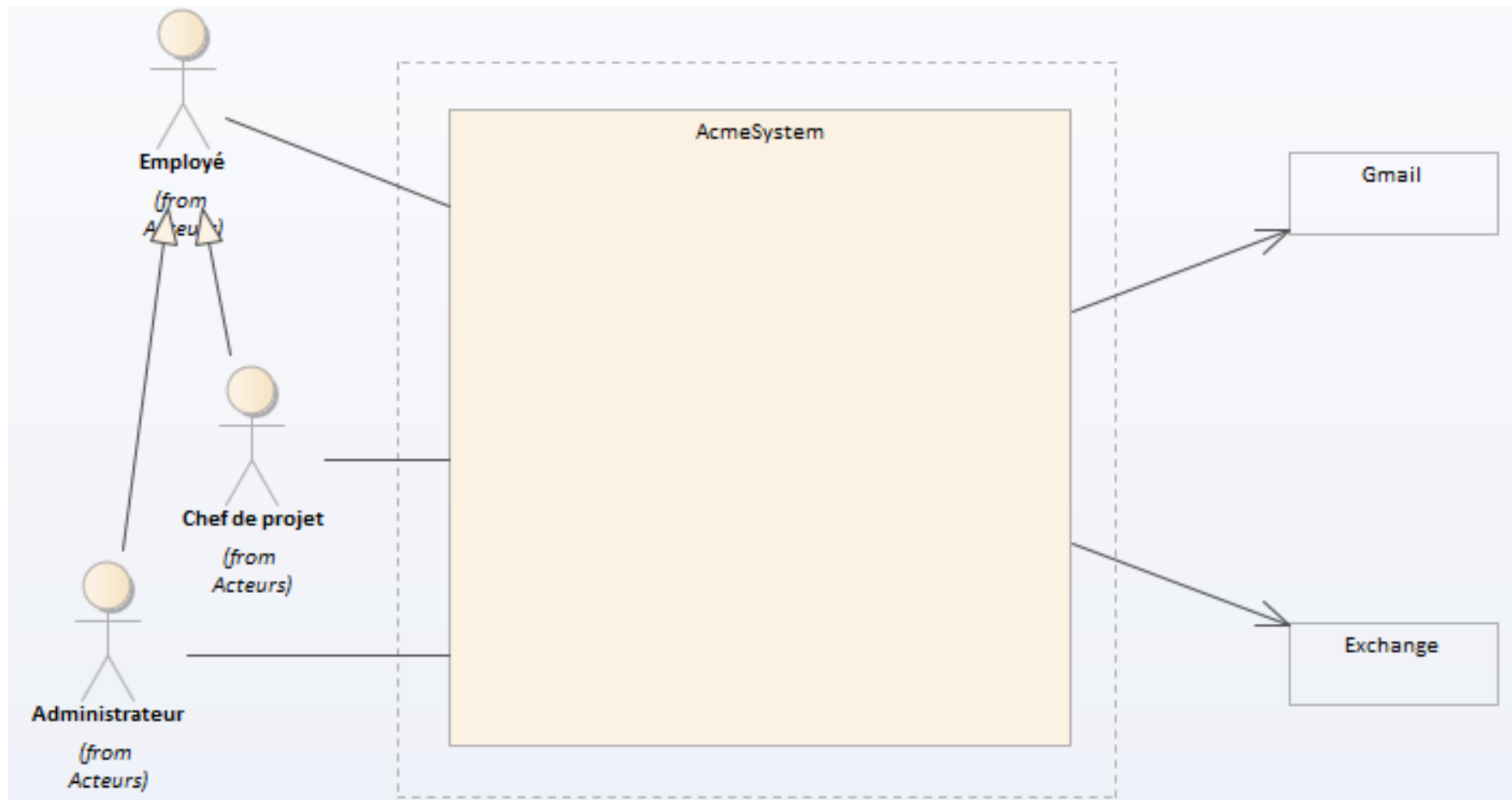
Identification des Uses Cases



Besoins

Etude de cas AcmeSystem

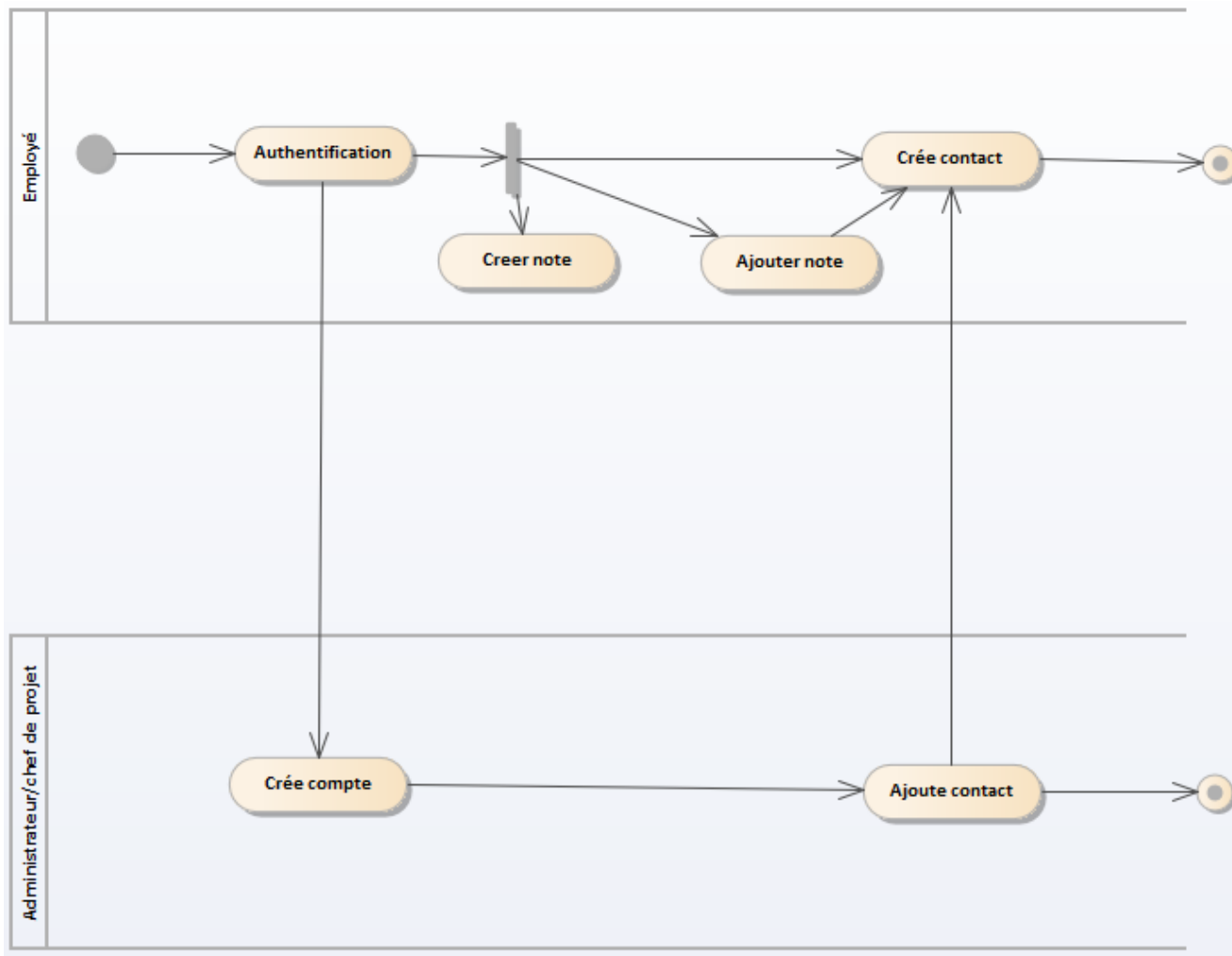
Diagramme de contexte et identification des acteurs



Besoins

Etude de cas AcmeSystem

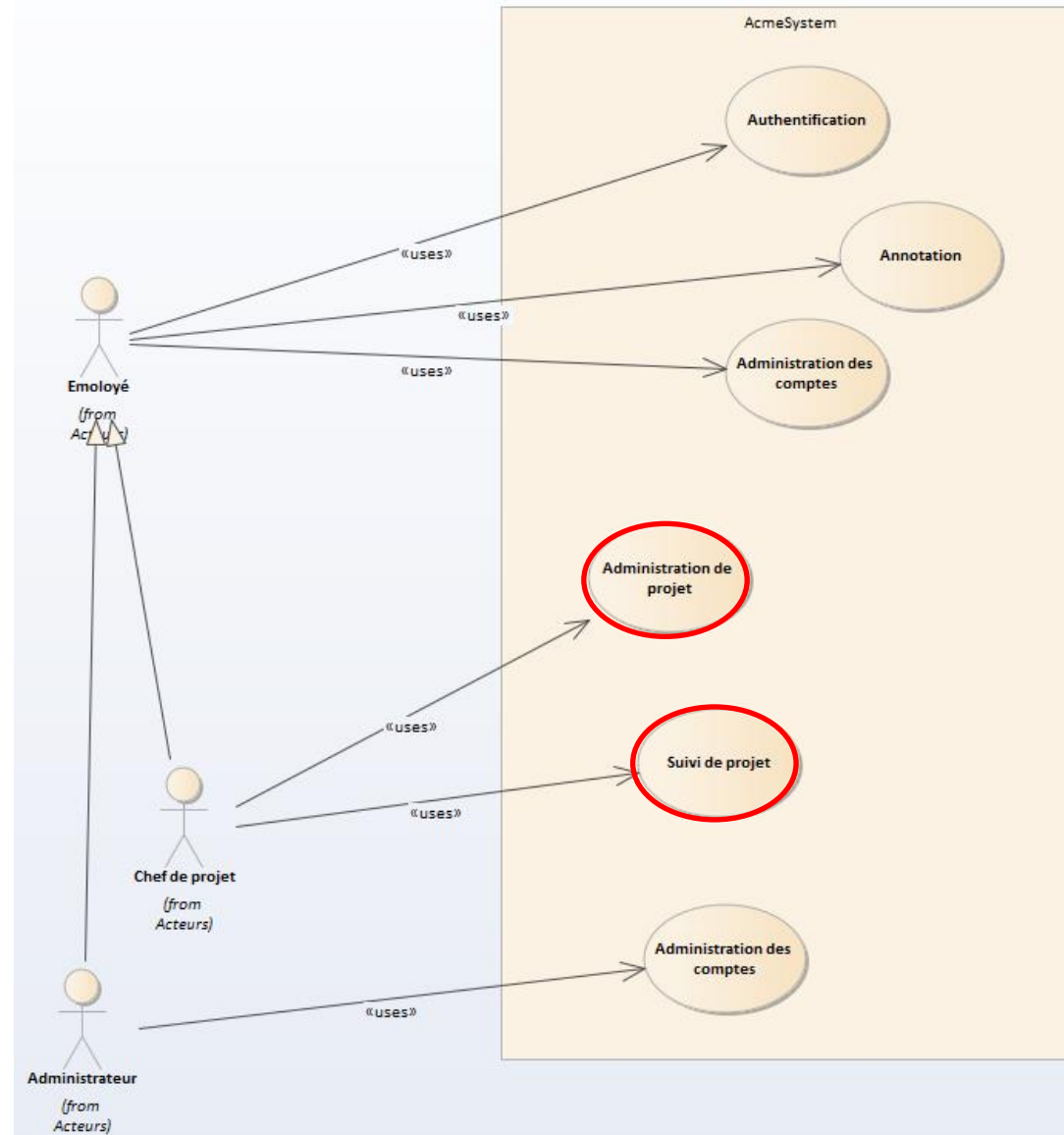
Diagramme d'activité de au niveau



Besoins

Etude de cas AcmeSystem

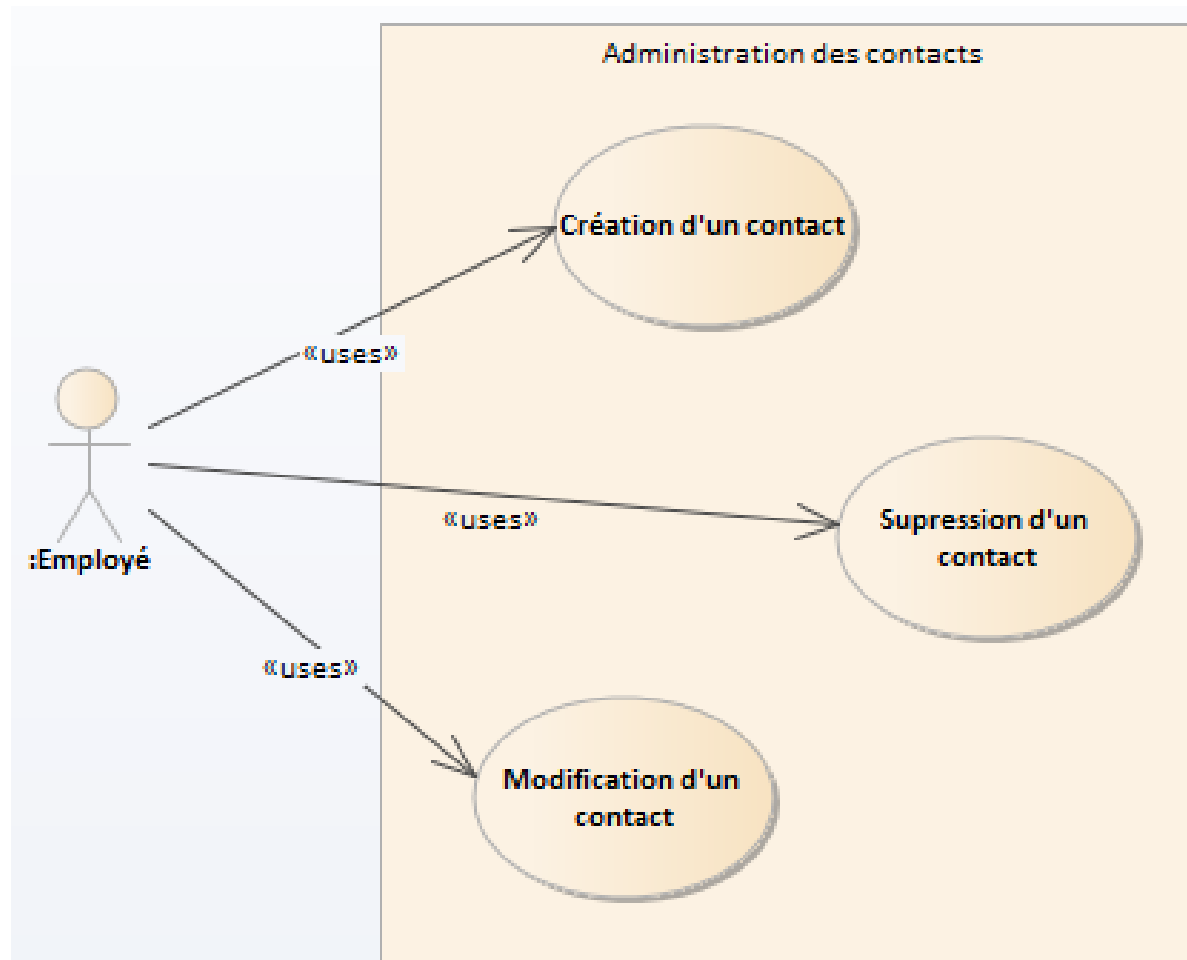
Diagramme de cas d'utilisation général



Besoins

Etude de cas AcmeSystem

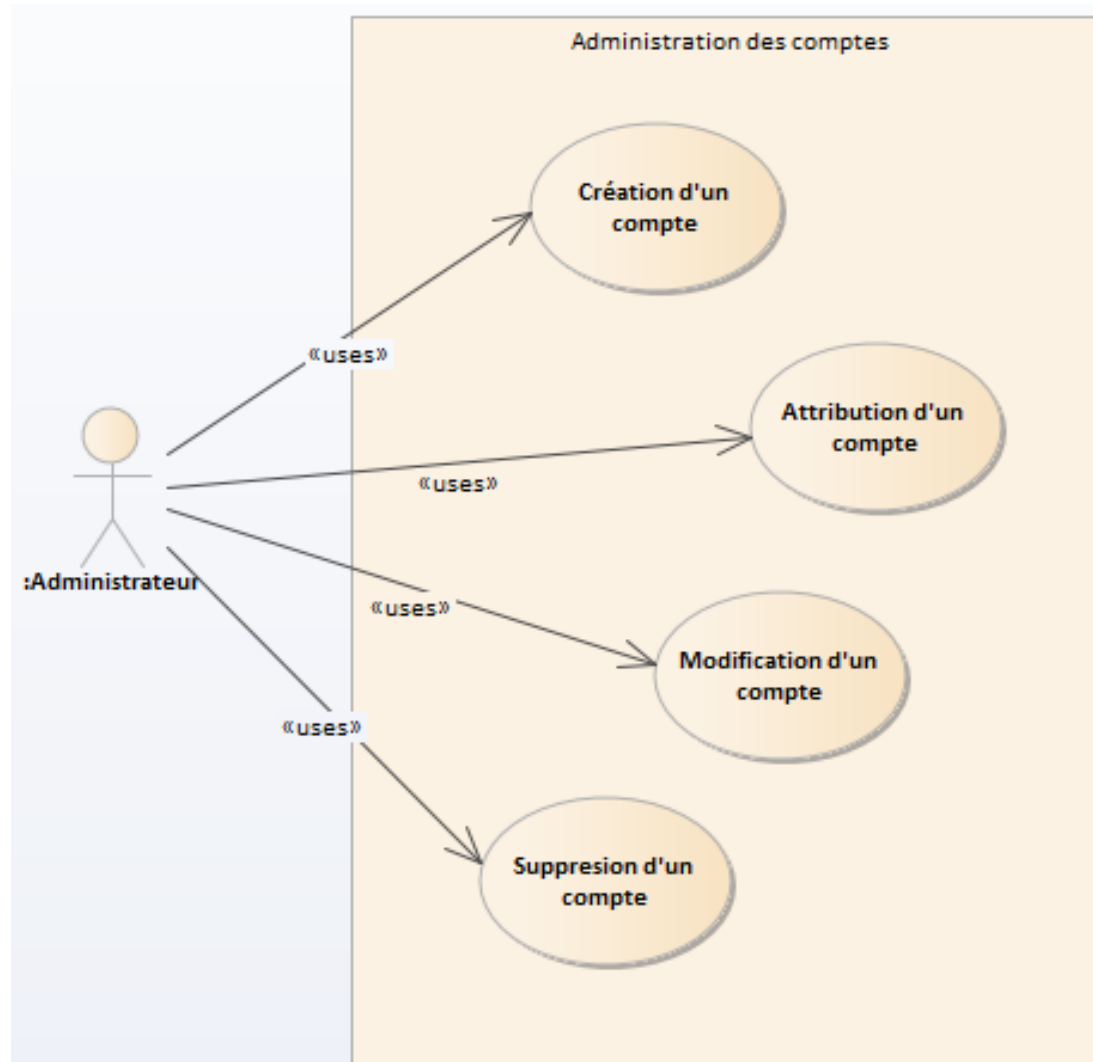
Cas d'utilisation Administration des contacts



Besoins

Etude de cas AcmeSystem

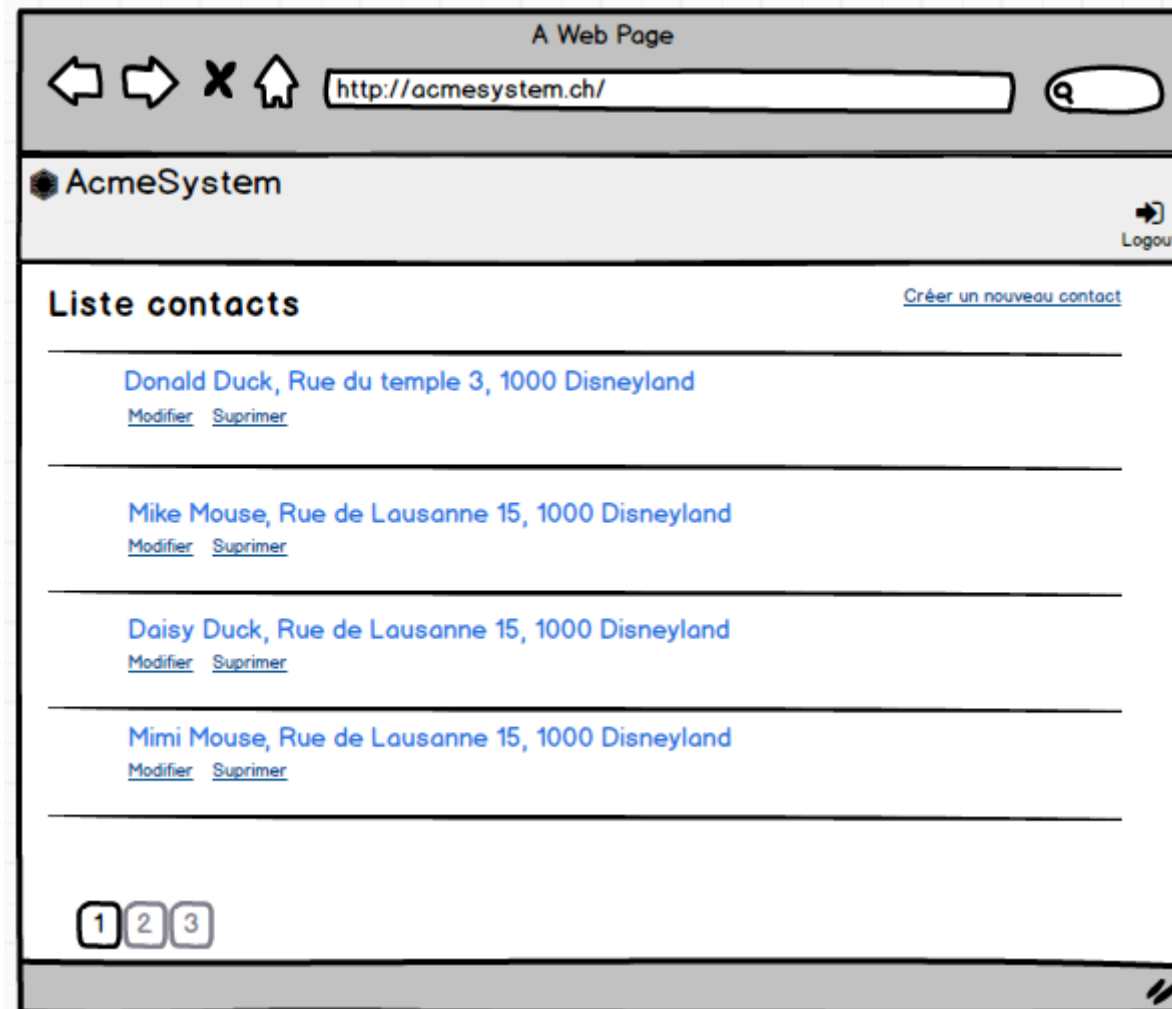
Cas d'utilisation Administration des comptes



Besoins

Etude de cas AcmeSystem

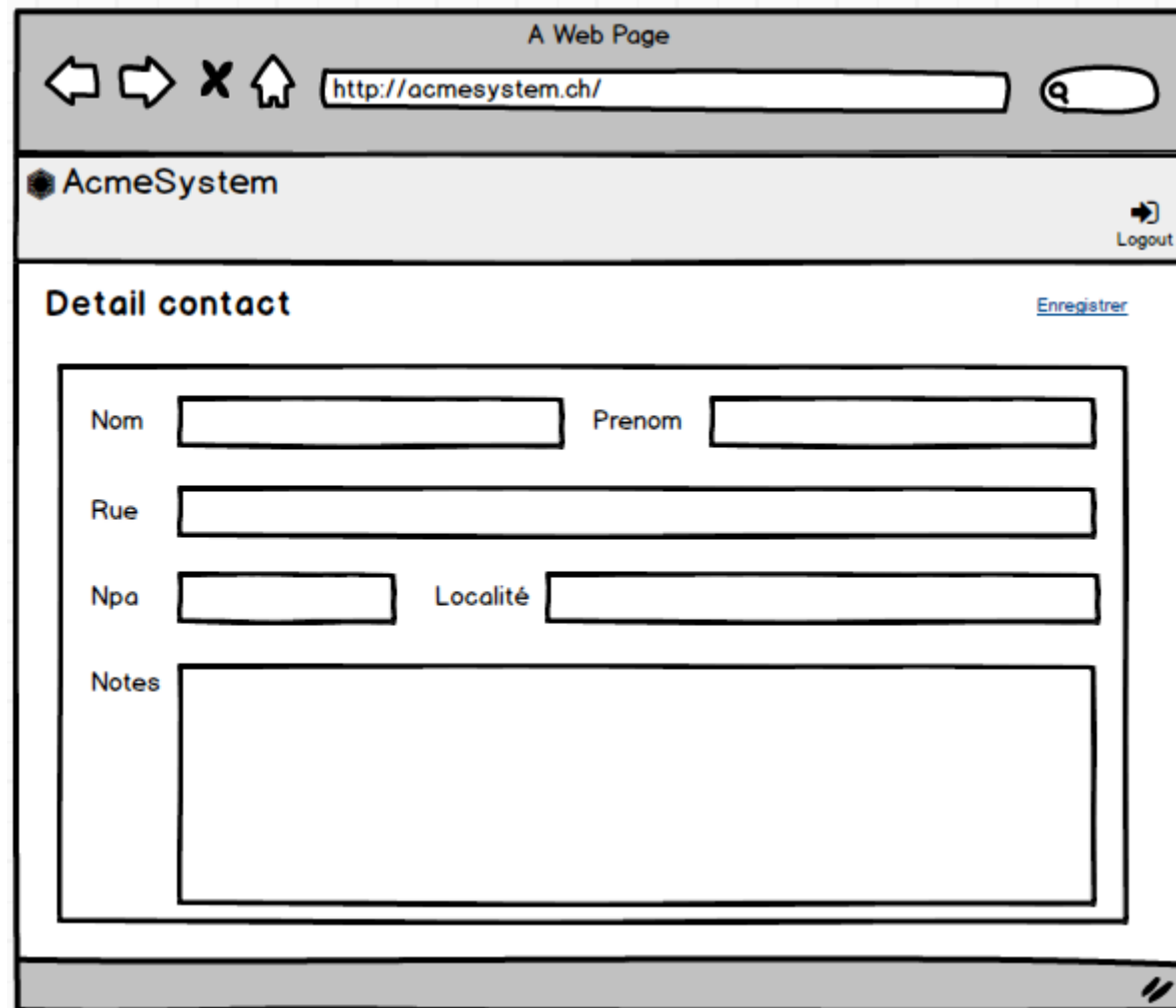
Mockup – Liste de contacts



Besoins


Etude de cas AcmeSystem

Mockup – Détail contact



A Web Page

[←](#)
[→](#)
[X](#)
[🏠](#)

 AcmeSystem
 [Logout](#)

Detail contact [Enregistrer](#)

Nom Prenom

Rue

Npa Localité

Notes

Besoins

Etude de cas AcmeSystem

Cas d'utilisation Administration des comptes

	Description
Acteur principal	Utilisateur
Objectif	L'utilisateur effectue son login dans le système.
Pré-conditions	Les données personnelles permettant l'authentification existent dans le système.
Déclencheur	L'utilisateur ouvre la page de login du système.
Scénario nominal	<p>Le système affiche un formulaire de login avec les champs nom d'utilisateur et mot de passe. :</p> <p>L'utilisateur remplit les valeurs du formulaire et sélectionne « soumettre » une fois terminé.</p> <p>Le système valide les données.</p> <p>Le système permet à l'utilisateur de rentrer dans le système.</p>
Extensions	<p>Le login échoue si :</p> <p>Le couple de valeurs nom d'utilisateur + mot de passe ne sont pas trouvés dans le système</p> <p>L'utilisateur n'est plus actif</p>

UC1 Authentification

Besoins

Etude de cas AcmeSystem

Cas d'utilisation Administration des comptes

	Description
Acteur principal	Administrateur
Objectif	L'administrateur doit pouvoir rajouter un contact à un compte.
Pré-conditions	Doit être authentifié. Doit être administrateur.
Déclencheur	L'administrateur choisit la configuration des comptes.
Scénario nominal	Le système affiche la liste des comptes. L'administrateur choisit le compte qui l'intéresse. Le système affiche les données du compte qui contiennent la liste des contacts qui y sont engagés et qui n'y sont pas. L'utilisateur choisit un ou plusieurs de ces contacts et valide. Le système valide les données. Le système enregistre les modifications. Le système répond à l'administrateur en lui présentant le résultat.
Extensions	Le compte ou le contact n'existe pas préalablement et doit être créé.

attribution d'un contact à un comte