IDEC école supérieure d'informatique

Implémenter le design physique d'une application

Module 477

Cours 1

Bienvenue à la session 2019

Hans W. Schwendimann

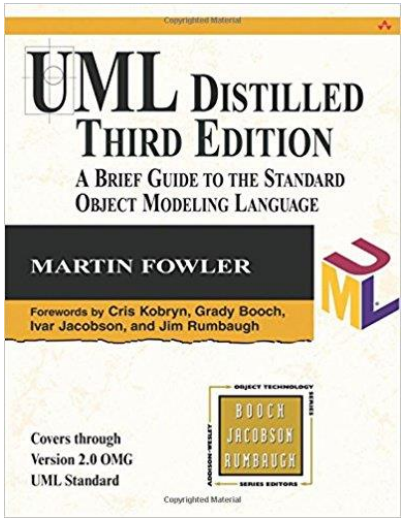
hws@idec.ch

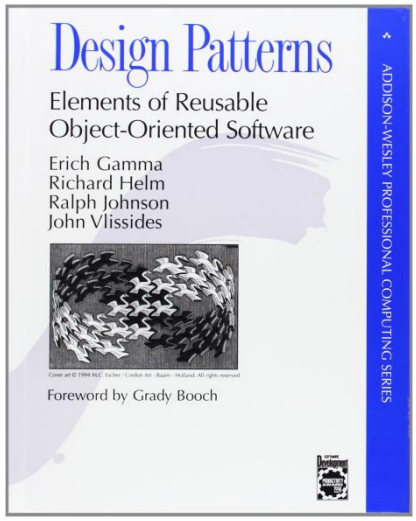
12 périodes (4 soirées) de cours (salle 3)

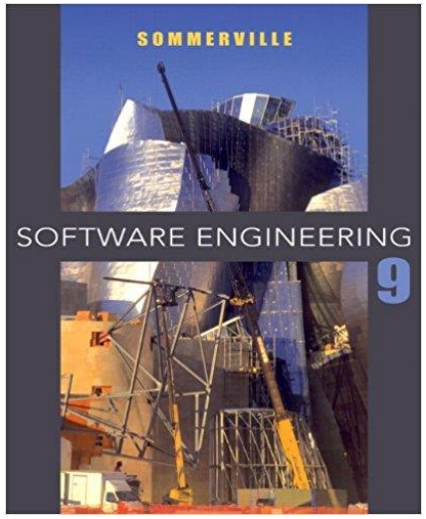
Horaires : 18h30-21h15 – Pause : 20h05-20h25

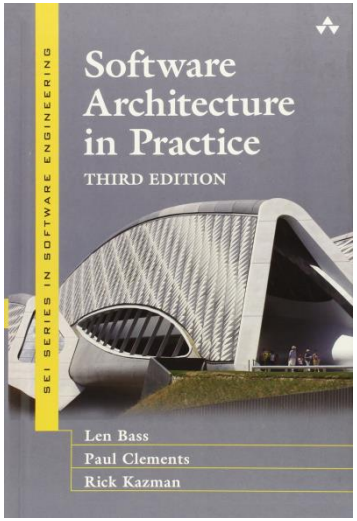
Fin de la session : le mardi 26 mars 2019

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	- Etude de cas (suite)
Mardi 19 mars 2019	- Etude de cas (suite)
Mardi 26 mars 2019	- Etude de cas (suite et fin)

Thème : Introduction aux Test unitaires (TDD)

Mardi 05 mars 2019

Planning

- Présentation
  - Présentation et objectifs
  - Correction test module 476
- Introduction au TDD
  - Qu'est ce que le TDD?
  - Principales étapes du TDD
  - Avantages
  - Inconvénients
- Mise en pratique du TDD
  - Créer un calculateur simple

✓ Pause 20h05 – 20h25

- Framework
  - Introduction
- Etude de cas
  - Initialisation
- Devoirs et feed-back de la soirée

Introduction au 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

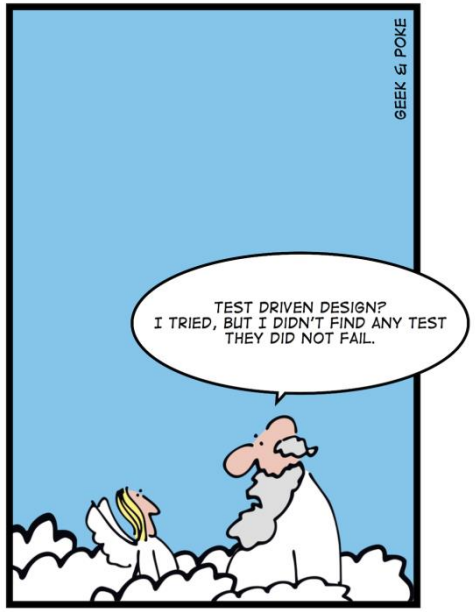
Introduction aux Test unitaires (TDD)

Introduction au TDD

Qu'est ce que TDD

Introduction au TDD

- Le TDD est une technique de développement qui est dirigée par les tests unitaires
  - les tests unitaires sont d'abord développés
  - ce qui permet de couvrir une fonctionnalité après une autre
  - permet de développer les fonctionnalités en respectant un cahier des charges

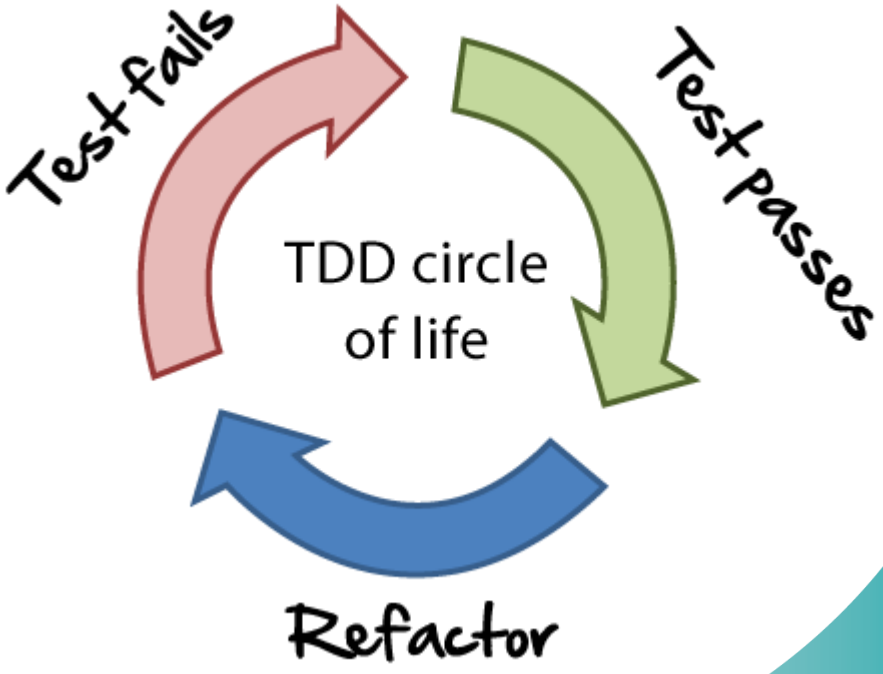


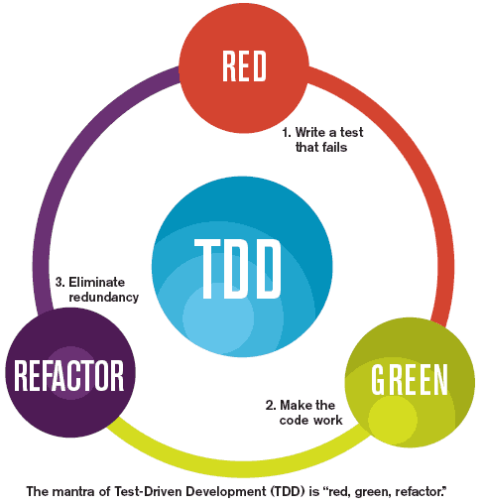
Principales étapes du TDD

Introduction au TDD

Le cycle du TDD se déroule en 5 étapes :

1. l'écriture du premier test qui couvre une fonctionnalité
2. la vérification de l'échec du test
   (car la fonctionnalité n'est pas encore implémentée)
3. l'écriture du code pour implémenter la fonctionnalité
   (et la vérification de la réussite du test)
4. vérification de la réussite du test
5. re-factorisation le code et vérification qu'il n'y ait pas de régression



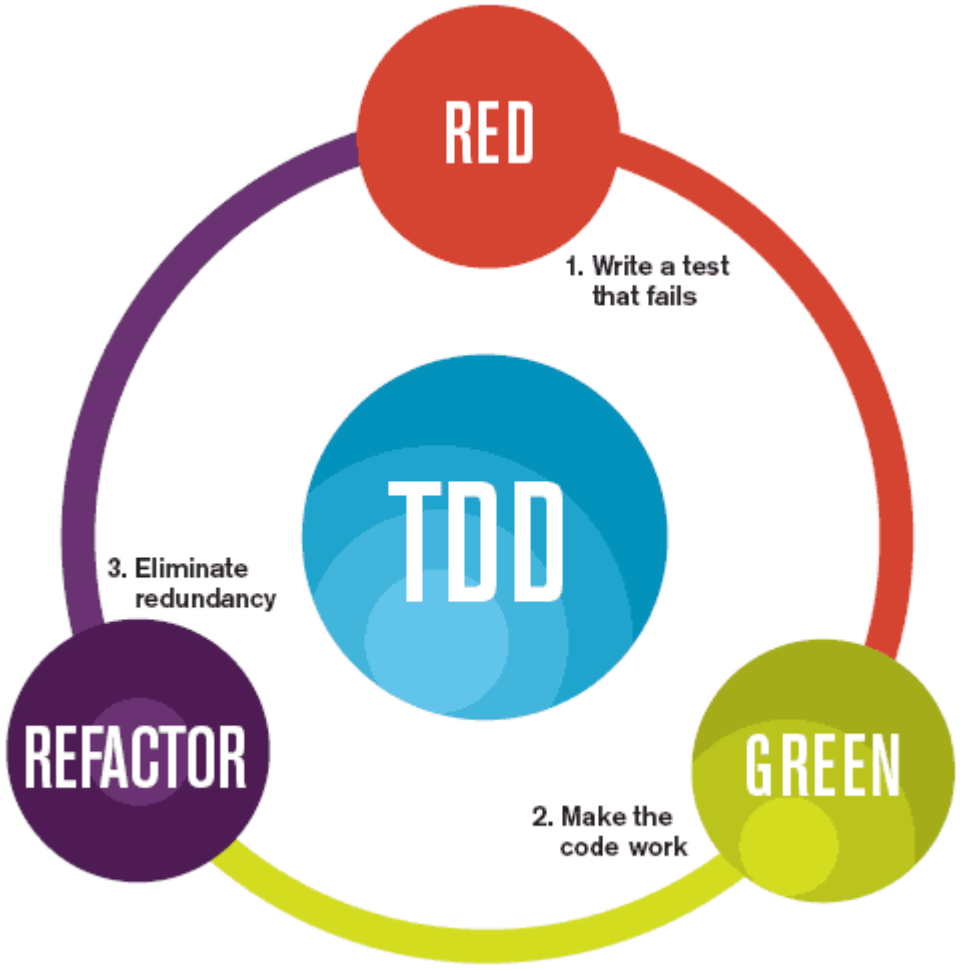


The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Principales étapes du TDD

Introduction au TDD

- Conçu par Philippe Kruchten
- Décrit l'architec de plusieurs vues simultané



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

4+1 architecturals views

Avantages

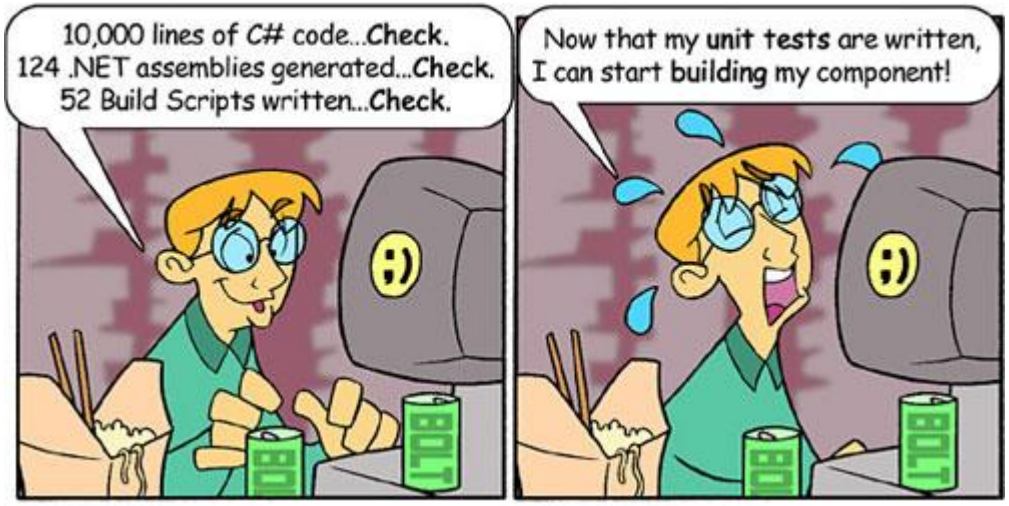
Introduction au TDD

- Teste exhaustivement le code produit
- Force à penser la fonctionnalité avant le code
- Force à penser l'interface avant l'implémentation
- Document le code par les tests
- Se concentre sur le code utile (une fonctionnalité codée est une fonctionnalité à maintenir)
- Produit itérativement du code (emergent design)

Inconvénients

Introduction au TDD

- Difficile à pratiquer et à maitriser
- Génère plus de tests que de code
- Difficulté à tester le GUI, DB, réseaux
  (Solution → fake, mock)
- Oblige à maintenir les tests



Introduction aux Test unitaires (TDD)

Mise en pratique du TDD

Créer un calculateur simple

Mise en pratique du TDD

Fonctionnalités :

1. créer une calculatrice string simple avec une méthode int Add(string s).
2. la méthode peut prendre 0, 1 ou 2 nombres, et retournera leur somme (pour une chaîne vide, elle retournera 0) par exemple "" ou "1" ou "1,2".
3. autoriser la méthode Add à gérer un nombre inconnu de nombres.
4. autoriser la méthode Add à gérer les retours à la ligne entre les nombres (au lieu de virgules).
5. l'entrée suivante est ok : "1\n2,3" (sera égal à 6).
6. Supporte différents délimiteurs.
7. Pour changer un délimiteur, le début de la chaîne contiendra une ligne distincte qui ressemble à ceci : "// [délimiteur] \n [nombres ...]" par exemple "//; \n1; 2" devrait retourner trois où le délimiteur par default est ' ;'.
8. La première ligne est facultative. Tous les scénarios existants doivent toujours être pris en charge.
9. Appeler la méthode Add() avec un nombre négatif lèvera une exception "négatifs non autorisés" - et le négatif qui a été passé. S'il y a plusieurs négatifs, montrez-les tous dans le message d'exception.
10. les nombres supérieurs à 1000 doivent être ignorés, ce qui ajoute 2 + 1001 = 2.
11. les délimiteurs peuvent être de n'importe quelle longueur avec le format suivant "// [délimiteur] n" par exemple : "// [-] n1-2-3" devrait retourner 6.
12. autoriser plusieurs délimiteurs comme ceci : "// [delim1] [delim2] n" par exemple "// [-] [n1-2.

Créer un calculateur simple

Mise en pratique du TDD

Itération 1

Exigence 2a : Pour une chaîne vide, la méthode retourne 0.

Itération 2

Exigence 2b : Si un seul nombre, la méthode retourne le nombre.

Itération 3

Exigence 2c : La méthode peut prendre 0, 1 ou 2 nombres séparés par une virgule (,).

Itération 4

Exigence 3 : La méthode peut prendre plusieurs nombres.

Itération 5

Refactorisation

Itération 6

Exigences 4-5 : La méthode autorise le retour à la ligne comme séparateur.

Itération 7

Exigence 6 : Plusieurs séparateurs consécutifs lèvent une exception.

Créer un calculateur simple (suite)

Mise en pratique du TDD

Itération 8

Exigences 6-7 : La méthode autorise différents séparateurs personnalisés.

Itération 9

Exigence 8 : La première ligne est facultative.

Voir tests antérieurs...

Itération 10

Exigence 9 : Appeler la méthode avec un nombre négatif lèvera une exception

Itération 11

Exigence 9b : Le message de l'exception des nombres négatifs doit contenir le ou les nombres négatifs passés en paramètre.

Il reste à faire comme devoir...

Implémentez les exigences 10, 11,12 manquantes ainsi que le client console utilisant la classe SimpleCalculator...

Introduction aux Test unitaires (TDD)

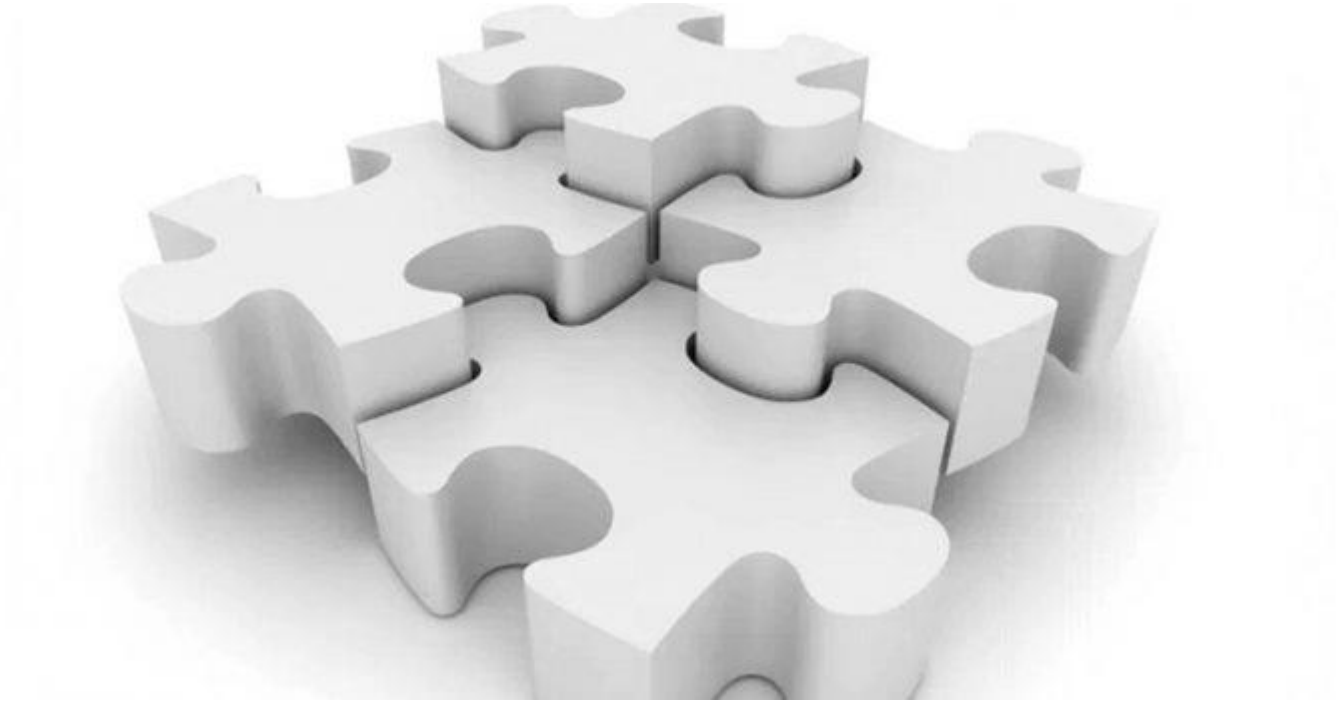
Framework

Introduction

Framework

Un **Framework** est un espace de travail modulaire

- C'est un ensemble de bibliothèques et de conventions permettant le développement rapide d'applications
- Il fournit suffisamment de briques logicielles et impose suffisamment de rigueur pour pouvoir produire une application aboutie et facile à maintenir



Types de Framework

Framework

On trouve différents types de **Framework** :

- **Framework** d'infrastructure système : pour développer des systèmes d'exploitation, des interfaces graphiques, des outils de communication. (exemple : Framework .Net, Eclipse, NetBeans, Struts, Cocoa)
- **Framework** d'intégration intergicielle : pour fédérer des applications hétérogènes. Pour mettre à dispositions différentes technologies sous la forme d'une interface unique. (exemple : Ampoliros avec ses interfaces RPC, SOAP, XML)
- **Framework** d'entreprise : pour développer des applications spécifiques au secteur d'activité de l'entreprise.
- **Framework** orienté Système de gestion de contenu

Une des avantages de ces **Framework** est la réutilisation de leur code…

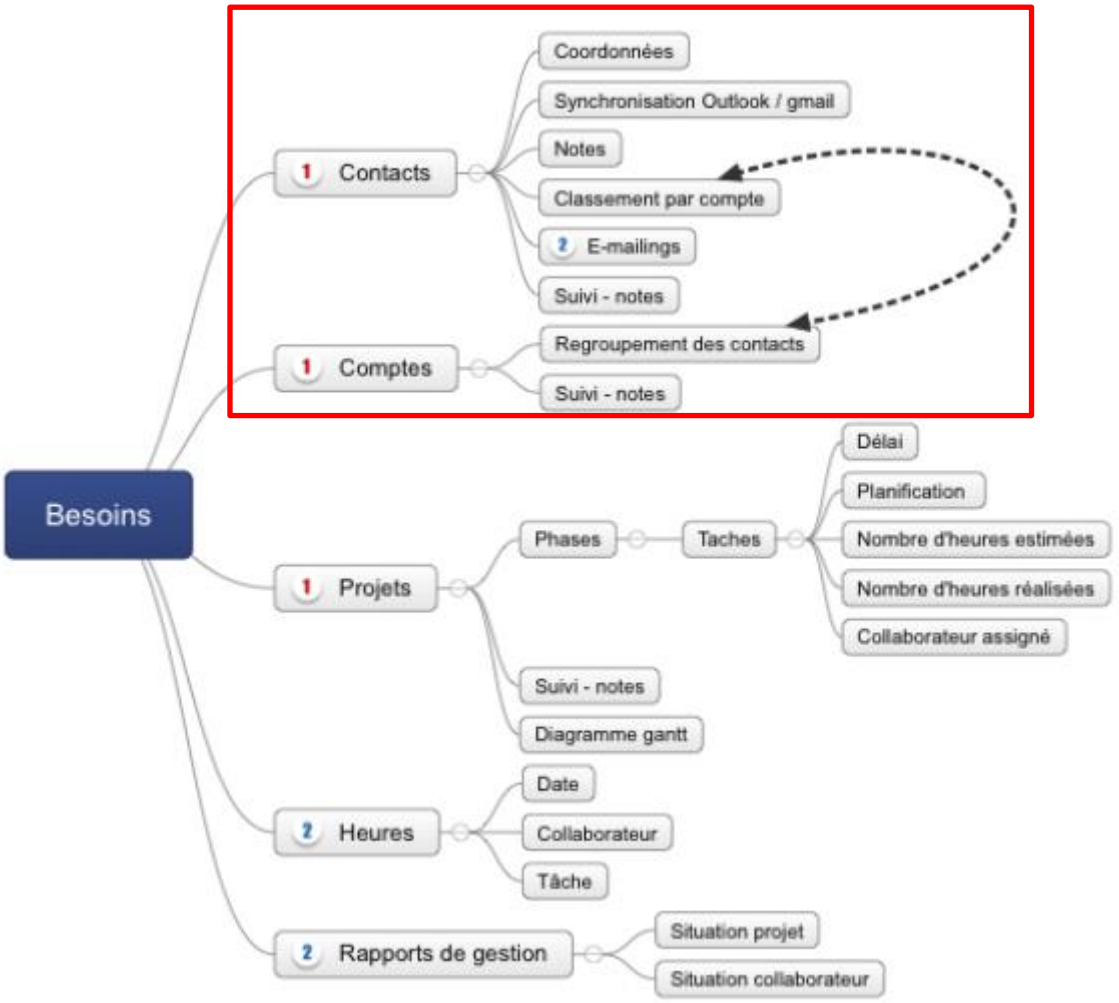

Introduction aux Test unitaires (TDD)

Etude de cas AcmeSystem

Besoins

Etude de cas AcmeSystem

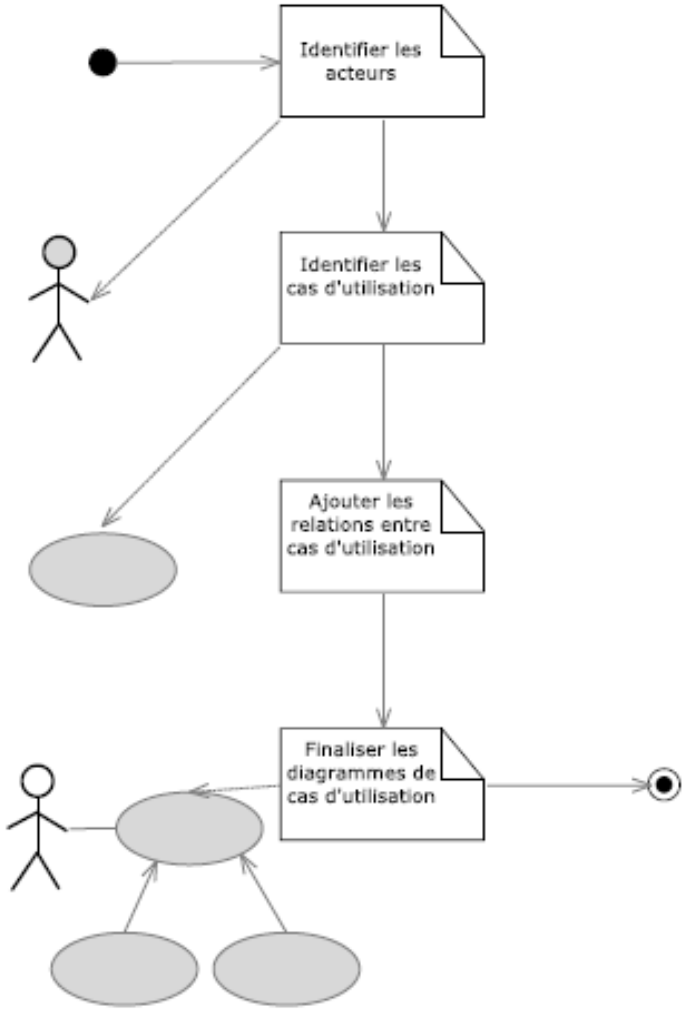
Schéma heuristique des besoins



Besoins

Etude de cas AcmeSystem

Identification des Uses Cases



Besoins

Etude de cas AcmeSystem

Diagramme de contexte et identification des acteurs