L3 MIAGE

Conception Orientée Objet

Une démarche de conception orientée objet (et au-delà …)

Dominique Ribouchon – Septembre 2017

§1

Objectifs du cours

- Maîtriser une **démarche** pour développer un logiciel de **qualité**:
  - Qualité **externe**: répondre aux **besoins utilisateur**
  - Qualité **interne**: maintenabilité et **évolutivité**

- Maîtriser les principes fondamentaux de la **conception logicielle**, applicable à toute technologie, en particulier objet:
  - **Séparation des préoccupations** en unités de code
  - **Limitation des dépendances**
  - **Cohérence** de l'ensemble, pour répondre aux besoins

§1

Organisation du cours – approche pédagogique

- Articulé autour d'une étude de cas, ce cours suit une approche pédagogique dirigée par la pratique:
  - **Cours/TD/TP mélangés,**
  - Articulés autour d'une **étude de cas**: projet de développement d'un logiciel pilotant un ascenseur
  - Etude cas réalisée en **équipes**
- Techniques pédagogiques ciblant:
  - La **responsabilisation** et **l'autonomie**,
  - La **capacité de concentration**
  - et basée sur le **respect mutuel**
- Communication via l'ENT**/Jalon (mail unice.fr)**

§1

3

Modalité de Contrôle de Connaissances (MCC)

- **Deux examens**
  - Examen prérequis (QCM): 10%
  - Examen final (QCM): 90%

§1

4

Plan détaillé du cours (1/3)

- Objectifs du projet de développement
  - Les **tâches d'ingénierie logicielle** – zoom sur la spécification des **exigences**, la **conception** et **codage**
  - La **gestion de projet itérative** et incrémentale – Mise en place du projet ascenseur

- **Sprint 1**: Réaliser le début du scénario "Appeler l'ascenseur"
  - Spécifier les exigences: affiner et adapter au périmètre
  - Concevoir: réfléchir à la structuration en unités de codes (en classes java)
  - Coder
  - Tester

Plan détaillé du cours (2/3)

- **Sprint 2**: Refactoring du code pour améliorer sa maintenabilité et évolutivité – **bonnes pratiques de conception fondamentales - UML**
  - Concevoir: **séparer les préoccupations en classes** et **limiter les dépendances** – Formalisation en **UML**
  - Coder
  - Tester
- **Sprint 3:** Elargir le périmètre fonctionnel au scénario complet - **communication entre classes**
  - Exigences: compléter le scénario
  - Concevoir:
    - compléter la structuration en classes
    - représenter la **communication entre classes**
  - Coder
  - Tester

Plan détaillé du cours (3/3)

- **Sprint 4**: Refactoring du code – **architecturer en packages**
  - Concevoir: structurer le logiciel en **couches**
  - Coder
  - Tester
- **Sprint 5**: Refactoring du code – affiner la communication entre couches - modes **requête/notification**
  - Concevoir: concevoir la communication entre couches
  - Coder
  - Tester
- **Sprint 6**: Refactoring du code – conception et programmation **OO**
  - Concevoir: structurer en **classes instanciables** et liées par des relations **d'association**
  - Coder
  - Tester

Plan du cours

- Objectifs du projet de développement et méthode
  - Les **tâches d'ingénierie logicielle** – zoom sur la spécification des **exigences**, la **conception** et **codage**
  - La **gestion de projet itérative** - Mise en place du projet ascenseur
- **Sprint 1**: Réaliser le début du scénario "Appeler l'ascenseur – la **séparation des préoccupations** en unités de codes ("classes")
- **Sprint 2**: Refactoring du code pour améliorer sa maintenabilité et évolutivité – **bonnes pratiques de conception fondamentales - UML**
- **Sprint 3:** Elargir le périmètre fonctionnel au scénario complet - **communication entre classes**
- **Sprint 4**: Refactoring du code – **architecturer en packages**
- **Sprint 5**: Refactoring du code – affiner la communication entre couches - modes **requête/notification**
- **Sprint 6**: Refactoring du code – conception et programmation **OO**

Objectifs d'un projet de développement logiciel

- Fournir un système de qualité, au moindre coût et dans les délais les plus courts: allier **qualité** et **productivité**
- Deux niveaux de qualité du système:
  - La réponse aux **besoins utilisateur**:
    - qualité externe
    - => vision de l'utilisateur
  - La maintenabilité, l'évolutivité, i.e. **"l'agilité du système":**
    - qualité interne,
    - => vision du développeur
  - La **réponse au besoin** est prépondérante
  - La qualité interne, rejoint, à **long terme**, l'objectif de productivité

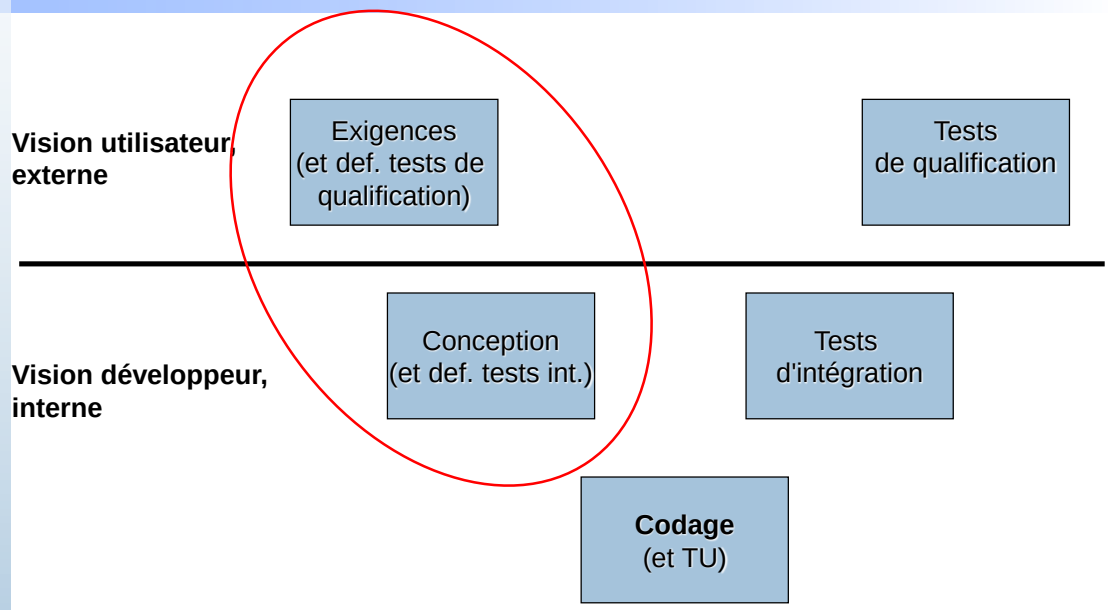
Une des clés: travailler avec un peu de méthode ….

- Ne pas se poser toutes les questions à la fois => distinguer différentes **tâches d'ingénierie logicielle** (spécification des exigences, conception, codage …)
- **Communiquer** avec d'autres personnes avant de faire des choix importants … et avant de coder

Les tâches d'ingénierie logicielle classiques

- Les activités de **définition** du logiciel, typiquement:
  - **Spécification des exigences**:le "**quoi externe**"
    (incluant la définition des tests de qualification)
  - **Conception**: le "**comment interne**"
    (incluant la définition des tests d'intégration)
- L'activité de codage
  (incluant la définition et l'exécution des tests unitaires)
- Les activités d'exécution des tests, typiquement:
  - Tests d'intégration
  - Tests de qualification

Tâches d'ingénierie logicielle présentées "en V"

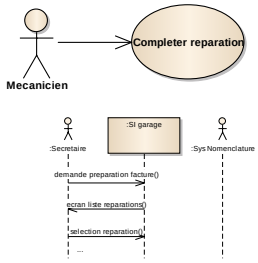


§1

12

Spécification des exigences – Requirements

Le "quoi externe"



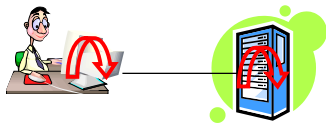
- Définit le logiciel, dans une vision **externe**, i.e. vision de son **exécution pour ses utilisateurs**
- Principalement:
  - les fonctionnalités
  - les IHM
  - …
- C'est ici que se joue principalement la **qualité externe**

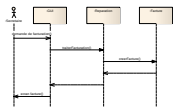
§1

13

Conception – Design

Le "comment interne"





- Définit le logiciel, dans une vision **interne**, i.e. vision de sa matérialisation sous la forme de **fichiers de code source** et exécutables **pour les développeurs** (et l'équipe d'exploitation)
- Il s'agit de définir tous les choix structurants concernant l'implémentation
- C'est ici que se joue la **qualité interne**, avec l'application de **bonnes pratiques de conception**

§1

14

Bonnes pratiques de conception du code source
Les fondamentaux

- Pratique 1- Séparation des préoccupations en unités de code:
  - Chaque unité de code possède une responsabilité claire, reflétée par son nom
  - La responsabilité d'une unité de code n'est pas partagée avec une autre unité (i.e. pas de duplication de code)
  - Les données et les traitements associés sont définis dans la même unité, qui est responsable de l'intégrité de ses données
  - …
- Pratique 2- Limitation des dépendances:
  - …
- Pratique 3: Cohérence de l'ensemble: réponse aux exigences

§1

15

Les bases de la méthode
Mise en œuvre pratique

- Objectif du projet: fournir un logiciel gérant des rectangles, appelé **logiciel "Rectangles manager",** an alliant:
  - Qualité (externe et interne)
  - Productivité (court terme et long terme)

- Moyens – Méthode :
  - Gestion de projet : construire les équipes projet, constituées de compétences complémentaires
  - réaliser les différentes tâches d'ingénierie logicielle
    - Spécifier les exigences
    - Concevoir
    - Coder
    - Tester

§1

16

Une donnée d'entrée au projet: les besoins généraux utilisateur

"Besoins utilisateur"

Vision utilisateur, externe

Exigences (et def. tests de qualification)

Tests de qualification

Vision développeur, interne

Conception (et def. tests int.)

Tests d'intégration

Codage (et TU)

§1

17

Besoins utilisateur pour le logiciel "Rectangles manager"

- L'utilisateur souhaite un logiciel lui permettant de créer 2 rectangles:
  - Saisir leurs largeurs et longueurs respectives
  - Afficher les surfaces des deux rectangles créés
  - Modifier la longueur du premier rectangle
  - Afficher la nouvelle surface du premier rectangle
- Pour la version actuelle, l'utilisateur n'attache pas d'importance à la qualité de l'IHM (une saisie et affichage console peut convenir)
- Par contre, le logiciel devra pouvoir évoluer facilement vers une IHM graphique, ainsi que sur la gestion d'un nombre de rectangles variable

§1

18

Spécifications des exigences – Exigences fonctionnelles

- Le logiciel n'offre qu'une seule fonctionnalité, i.e. un seul "cas d'utilisation": Manipuler deux rectangles

- Un seul scénario est réalisé dans cette version du logiciel:
  - L'utilisateur lance le logiciel
  - Le logiciel l'invite à saisir la largeur, puis la longueur du premier rectangle
  - Le logiciel l'invite à saisir la largeur, puis la longueur du deuxième rectangle
  - Le logiciel affiche les surfaces des deux rectangles créés
  - Le logiciel invite l'utilisateur à modifier la longueur du premier rectangle
  - Le logiciel affiche la nouvelle surface du premier rectangle

§1

Spécifications des exigences – Exigences non fonctionnelles

- Aucune exigence non fonctionnelle concernant l'utilisation du logiciel n'est à prendre en compte dans cette version:
  - Utilisabilité: pas d'exigences sur l'ergonomie. Une saisie et affichage console peut convenir
  - Robustesse: pas d'exigence à ce niveau, en particulier le logiciel peut "*planter*" en cas de saisie erronée
  - Performances: aucune exigence au niveau des temps de réponse
- Par contre, le logiciel doit être maintenable et évolutif (IHM graphique, la gestion d'un nombre de rectangles variable …

Spécifications des exigences – Description détaillées des interfaces

- Description de l'IHM:
  - Les saisies et affichages se font au travers de la console utilisateur
  - Affichage et saisie console pour le cas d'utilisation "Manipuler deux rectangles":

```
Donner la largeur du premier rectangle:
2
Donner la longueur du premier rectangle:
4
Donner la largeur du deuxieme rectangle:
2
Donner la longueur du deuxieme rectangle:
3
La surface du premier rectangle est: 8
La surface du deuxième rectangle est: 6
Modifier la longueur du premier rectangle:
6
La surface du premier rectangle est: 12
```

Concevoir le logiciel

- Avant de coder, réfléchir à la structuration en unités de code en appliquant les bonnes pratiques de séparation en préoccupations:
  - Chaque unité de code possède une responsabilité claire, reflétée par son nom
  - La responsabilité d'une unité de code n'est pas partagée avec une autre unité (i.e. pas de duplication de code)
  - Les données et les traitements associés sont définis dans la même unité, qui est responsable de l'intégrité de ses données

Coder et tester

- Coder en Java
  - Aide sur la classe main:

```
import java.util.Scanner;
public class ??? {
   public static void main(String[] args) {
      …
      Scanner saisie = new Scanner(System.in);
      System.out.println("Donner la largeur du premier rectangle: ");
      largeur = saisie.nextInt();
```

- Tester

Approche suivie par l'architecte logiciel DR
Définition des tests de qualification (1/2)

- Définition de 4 scénarios de tests

- Scénario 1 – 2 rectangles totalement différents
  - Saisir largeur 2 et longueur 3
  - Saisir largeur 6 et longueur 7
  - Vérifier que le logiciel affiche des surfaces 6 et 42
  - Saisir longueur 10
  - Vérifier que le logiciel affiche la surface 20

- Scénario 2 – les 2 rectangles des exigences – IHM
  - Saisir largeur 2 et longueur 4
  - Saisir largeur 2 et longueur 3
  - Vérifier que le logiciel affiche des surfaces 8 et 6
  - Saisir longueur 6
  - Vérifier que le logiciel affiche la surface 12

§1

24

Approche suivie par l'architecte logiciel DR
Définition des tests de qualification (2/2)

- Scénario 3 – 2 rectangles identiques
  - Saisir largeur 3 et longueur 5
  - Saisir largeur 3 et longueur 5
  - Vérifier que le logiciel affiche des surfaces 15 et 15
  - Saisir longueur 6
  - Vérifier que le logiciel affiche la surface 18
- Scénario 4 – 2 rectangles grandes dimension
  - Saisir largeur 30 et longueur 50
  - Saisir largeur 100 et longueur 200
  - Vérifier que le logiciel affiche des surfaces 1500 et 20000
  - Saisir longueur 100
  - Vérifier que le logiciel affiche la surface 3000
- Notons que les tests s'effectuent par saisie et affichage console, comme demandé dans les exigences. Aucun code de test n'est nécessaire

§1

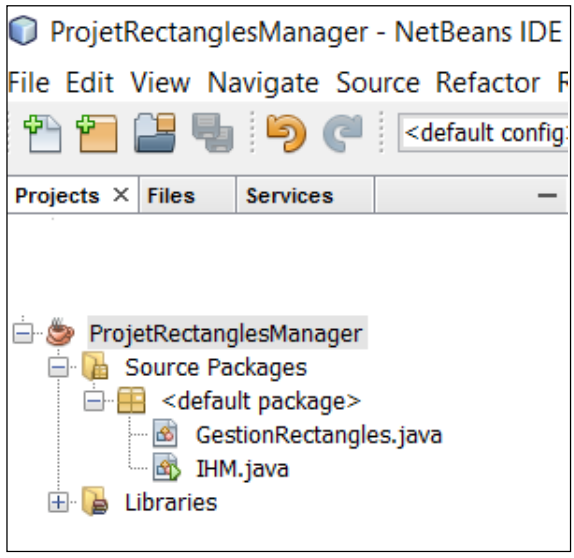
25

Approche suivie par l'architecte logiciel DR Conception du logiciel

- Contraintes de conception du code:
  - Le code est réalisé en Java
  - Adopter une structuration simple à 2 classes
  - Seules des notions acquises dans le cours POO au 10/11/16 seront utilisées
- Le logiciel "Rectangles manager" est constitué de 2 unités de code: classe "IHM" et classe "GestionRectangles"
  - Responsabilité de la classe IHM: Interfaçage avec l'utilisateur, au travers d'affichage et de saisie console
  - Responsabilité de la classe "GestionRectangles": gérer les informations des rectangles et garantir leur intégrité

Approche suivie par l'architecte logiciel DR Codage (1/3)

- L'IDE utilisé est NetBeans
- Le projet NetBeans est appelé " ProjetRectanglesManager"



§1

27