

L3 MIAGE

Conception Orientée Objet

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



Dominique Ribouchon – Septembre 2017₁

© D. Ribouchon

S1

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

S1

2

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**

§1

3

- Communication via l'ENT/**Jalon (mail unice.fr)**

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

§1

5

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

§1

6

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

§1

7

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** 8

§1

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

© D. Ribouchon

§1

9

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

© D. Ribouchon

§1

10

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

§1

11

Tâches d'ingénierie logicielle présentées "en V"

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)

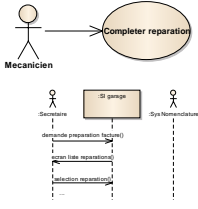

§1

12

§1

Spécification des exigences – Requirements

Le "quoi externe"



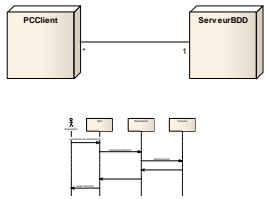

- Définit le logiciel, dans une vision **externe**, i.e. vision de son **exécution pour ses utilisateurs**
- Principalement:
 - les fonctionnalités
 - les IHM
 - ...
- C'est ici que se joue principalement la **qualité externe**

13

§1

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**

14

© D. Ribouchon

7

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:
 - ...

§1

- Pratique 3: Cohérence de l'ensemble: réponse aux exigences

Les bases de la méthode

Mise en œuvre pratique



- Objectif du projet: fournir un logiciel gérant des rectangles, appelé **logiciel "Rectangles manager"**, en 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



**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 ...

§1

20

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
```

§1

21

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

§1

22

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

§1

23

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

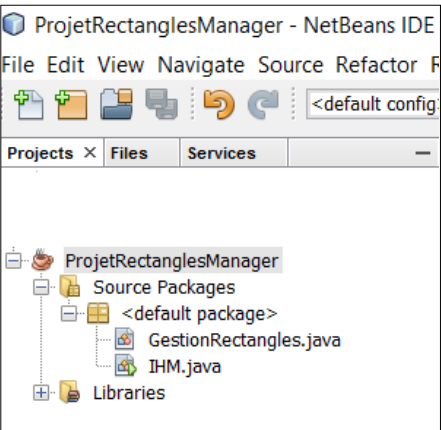
§1

26

Approche suivie par l'architecte logiciel DR

Codage (1/3)

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



§1

27

```
1 import java.util.Scanner;
2 public class IHM {
3     public static void main(String[] args) {
4         //Variables locales à la méthode
5         int largeur;
6         int longueur;
7         int surface;
8         int idPremierRectangle;
9         int idDeuxiemeRectangle;
10        Scanner saisie = new Scanner(System.in);
11
12        //Création du premier rectangle
13        System.out.println("Donner la largeur du premier rectangle: ");
14        largeur = saisie.nextInt();
15        System.out.println("Donner la longueur du premier rectangle: ");
16        longueur = saisie.nextInt();
17        idPremierRectangle = GestionRectangles.creer(largeur, longueur);
18
19        //Création du deuxieme rectangle
20        System.out.println("Donner la largeur du deuxieme rectangle: ");
21        largeur = saisie.nextInt();
22        System.out.println("Donner la longueur du deuxieme rectangle: ");
23        longueur = saisie.nextInt();
24        idDeuxiemeRectangle = GestionRectangles.creer(largeur, longueur);
25        surface = GestionRectangles.getSurface(idPremierRectangle);
26
27        //Affichage des 2 surfaces
28        System.out.println("La surface du premier rectangle est: " + surface);
29        surface = GestionRectangles.getSurface(idDeuxiemeRectangle);
30        System.out.println("La surface du deuxième rectangle est: " + surface);
31
32        //Modification de la longueur du premier rectangle et affichage de la nouvelle surface
33        System.out.println("Modifier la longueur du premier rectangle: ");
34        longueur = saisie.nextInt();
35        GestionRectangles.setLongueur(idPremierRectangle, longueur);
36        surface = GestionRectangles.getSurface(idPremierRectangle);
37        System.out.println("La surface du premier rectangle est: " + surface);
38    }
39 }
```

§1

§1

```
public class GestionRectangles {
    static private class Rectangle {
        private int numero;
        private int largeur;
        private int longueur;
    }

    private final static int NB_MAX_RECTANGLES = 2;
    private static Rectangle[] lesRectangles = new Rectangle[NB_MAX_RECTANGLES];
    private static int nbRectangles = 0;

    public static int creer (int largeur, int longueur){
        lesRectangles[nbRectangles] = new Rectangle();
        lesRectangles[nbRectangles].largeur = largeur;
        lesRectangles[nbRectangles].longueur = longueur;
        nbRectangles++;
        return nbRectangles - 1;
    }

    public static int getLargeur(int numRectangle){
        return lesRectangles[numRectangle].largeur;
    }

    public static int getLongueur(int numRectangle){
        return lesRectangles[numRectangle].longueur;
    }

    public static int getSurface(int numRectangle){
        return lesRectangles[numRectangle].largeur * lesRectangles[numRectangle].longueur;
    }

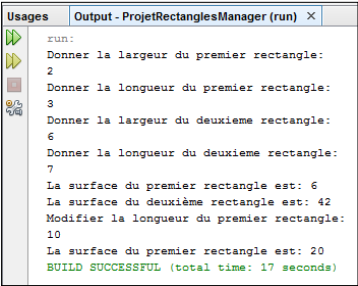
    public static void setLongueur(int numRectangle, int longueur){
        lesRectangles[numRectangle].longueur = longueur;
    }
}
```

§1

Approche suivie par l'architecte logiciel DR

Exécution des tests de qualification

- Exécution des 4 scénarios – copie écran scénario 1:



```
run:
Donner la largeur du premier rectangle:
2
Donner la longueur du premier rectangle:
3
Donner la largeur du deuxieme rectangle:
6
Donner la longueur du deuxieme rectangle:
7
La surface du premier rectangle est: 6
La surface du deuxième rectangle est: 42
Modifier la longueur du premier rectangle:
10
La surface du premier rectangle est: 20
BUILD SUCCESSFUL (total time: 17 seconds)
```

- Rapport de tests de qualification:

Tests passés le 10/11/16, par DR

Scénario 1: OK

Scénario 2: OK

Scénario 3: OK

Scénario 4: OK

30

Approche suivie par l'architecte logiciel DR

Préparation revue

- Démo: exécuter les scénarios 1 et 2
- Présentation de la conception: présenter le slide suivant

§1

31

Revue du projet "Rectangles manager"

Choix de conception (1/3)

- Le logiciel "Rectangles manager" est constitué de 2 unités de codes: 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

§1

32

Revue du projet "Rectangles manager"

Choix de conception (2/3)

- Vérification de l'application des bonnes pratiques:
 - 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
 - **Eviter la redondance de données**
 - Pratique 3: Cohérence de l'ensemble: réponse aux exigences

§1

33

Revue du projet "Rectangles manager"

Choix de conception (3/3)

- Vérification de l'évolutivité – impact des évolutions suivantes:
 - Garantir que la largeur est toujours inférieure à la longueur
 - Réaliser une IHM graphique, avec un écran permettant de créer les rectangles et de les représenter graphiquement
 - Donner la possibilité de créer jusqu'à 100 rectangles

§1

34

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** ³⁵

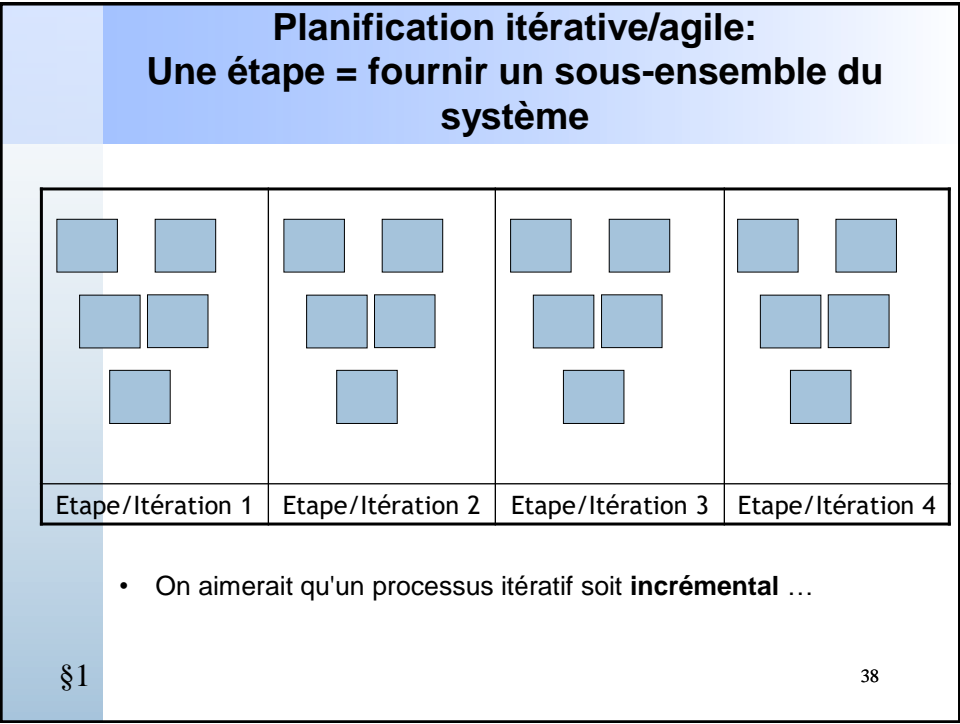
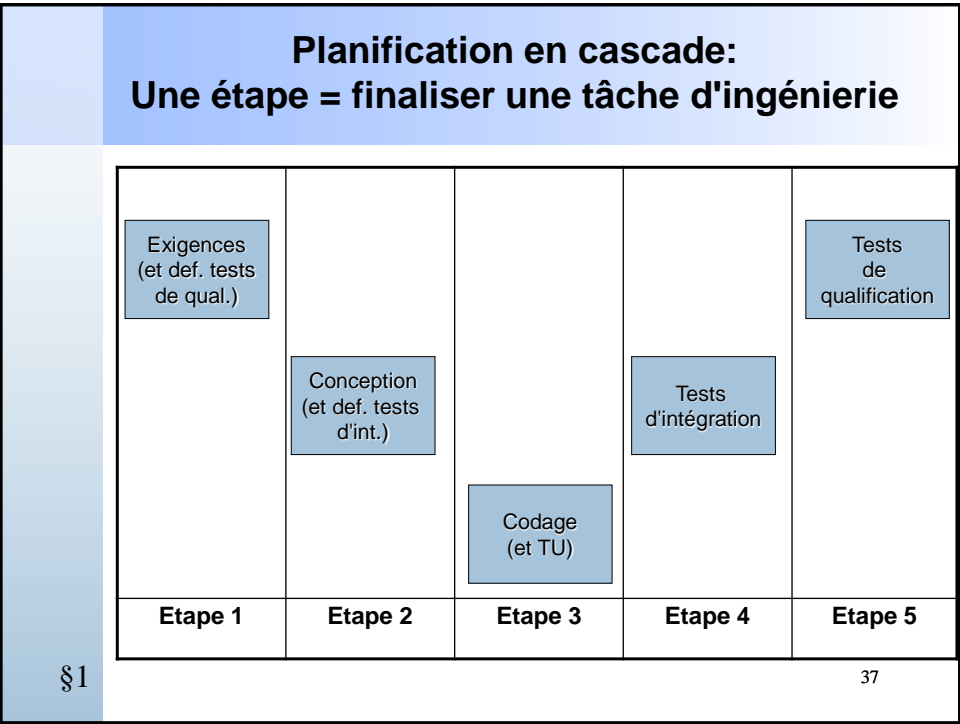
§1

La gestion de projet

- L'activité de gestion de projet adresse en particulier:
 - la **planification**: découper le projet en étapes, fixer des objectifs à chaque étape
 - pour chaque étape, et en fonction des objectifs fixés, définition des moyens:
 - les tâches d'ingénierie effectuées
 - les ressources affectées, en particulier, les ressources humaines
- Deux grands types de planification:
 - planification dite "**en cascade**"
 - planification dite "**itérative**", "**agile**"

§1

36



Méthode de gestion de projet Scrum

- La base de Scrum est l'**empirisme**: les connaissances proviennent de l'expérience =>
 - approche **itérative et incrémentale**
 - pilotée par la recherche de **valeur** et la levée des risques
- Les trois piliers de son implémentation: la transparence, l'inspection et l'adaptation
- Le cadre est léger, n'imposant que peu de choses:
 - Les releases et les **sprints** (i.e. itérations)
 - **Une équipe avec trois rôles** – l'importance des gens
 - Un backlog contenant le travail à faire

§1

39

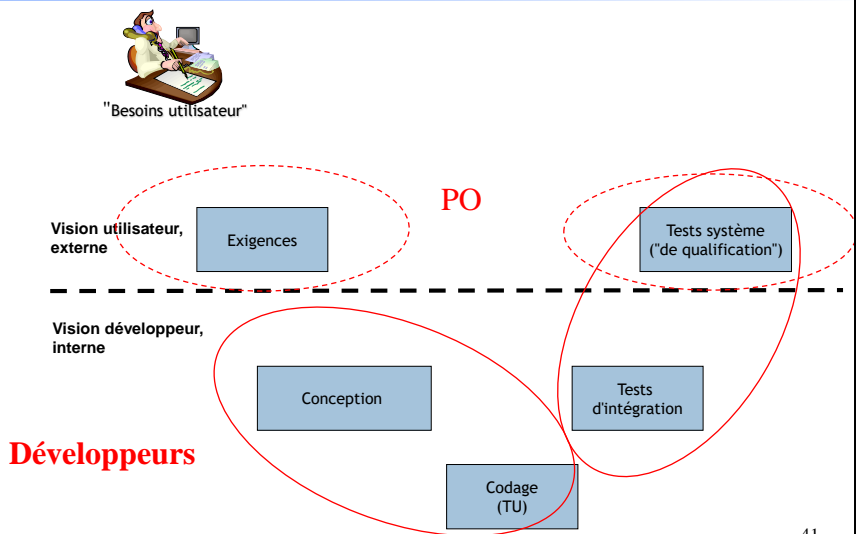
Une équipe avec trois rôles – L'importance des gens

- Une **équipe** Scrum est constituée des personnes qui contribuent à produire un résultat à chaque sprint et comporte 3 rôles:
 - Développeur
 - Product Owner (PO)
 - ScrumMaster (SM) – un des développeurs
- Les parties prenantes, extérieures à l'équipe, typiquement:
 - Utilisateurs, clients, marketing ... (MOA)
 - Experts

§1

40

tâches d'ingénierie et rôles



Mise en place des équipes Scrum

- Equipes de 4 personnes:
 - 4 développeurs
 - 1 PO (D. Ribouchon)
- 1 expert architecte logiciel (D. Ribouchon)

Plan de développement

- Le projet est divisé en deux phases/releases:
 - Phase/Release 1: Fournir le logiciel ascenseur
 - réalisant un scénario simple du CU "Appeler l'ascenseur",
 - "relativement" maintenable et évolutif,
 - dans une approche OO simple
 - Phase/Release 2: Fournir le logiciel ascenseur
 - réalisant un scénario simple du CU "Appeler l'ascenseur",
 - "très" maintenable et évolutif
 - dans une approche OO avancée

§1

43

Projet Ascenseur – Plan des sprints de la phase 1 (1/2)

- **Sprint 1:**
 - Réaliser le début du scénario "Appeler l'ascenseur – la **séparation des préoccupations** en unités de codes ("classes")
 - Début-fin: 21/09 -29/09
- **Sprint 2:**
 - Objectif: Refactoring du code pour améliorer sa maintenabilité et évolutivité – **bonnes pratiques de conception fondamentales** – formaliser en **UML**
 - Début-fin: à définir
- **Sprint 3:**
 - Objectif: Elargir le périmètre fonctionnel au scénario complet - **communication entre classes**
 - Début-fin: à définir

§1

44

Projet Ascenseur – Plan des sprints de la phase 1 (2/2)

- **Sprint 4:**
 - Objectif: Refactoring du code – **architecturer en packages**
 - Début-fin: à définir
- **Sprint 5:**
 - Objectif: Refactoring du code – affiner la communication entre couches - modes **requête/notification**
 - Début-fin: à définir
- **Sprint 6:**
 - Objectif: Refactoring du code – conception et programmation **OO**
 - Fin souhaitée: 07/01

§1

45

Plan du cours

- Objectifs du projet de développement et méthode
- **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**

§1

46

Objectifs du sprint 1

- **Objectif:** Réaliser le début du scénario "Appeler l'ascenseur" – Prise en compte de la pression du bouton d'appel
- **Contraintes:** Nous ne disposons ni du HW ni du dispositif de communication au travers des registres
- **Début-Fin:** A définir
- **Revue de sprint:** A définir
 - Parties prenantes: "MOA", expert architecte logiciel
 - Le PO anime la réunion
 - Les développeurs:
 - Font une démo
 - Expliquent la structuration de leur code
 - Recherchent des pistes d'amélioration de la maintenabilité et de l'évolutivité du code avec l'architecte logiciel

§1

Réalisation du Sprint 1

- Spécifier les exigences: compléter et adapter au périmètre
- Définir les tests de qualification
- Concevoir: réfléchir à la structuration en unités de codes (en classes java)
- Coder
- Exécuter les tests de qualification

§1

48

Sprint 1 – Exigences

Composition physique du système ascenseur

- Pour la phase 1, le système ascenseur est constitué des éléments physiques suivants:
 - 1 cabine composée de 2 éléments mécaniques :
 - 1 porte, appelée porte de cabine, dont le numéro est 4
 - un entraînement, dont le numéro est 0
 - 4 étages (incluant le RDC, de numéro 0), chacun composé de 2 éléments mécaniques:
 - d'un bouton, jouant le rôle de bouton d'appel, dont le numéro est le numéro de l'étage
 - d'une porte, appelée porte d'étage, dont le numéro est le numéro de l'étage

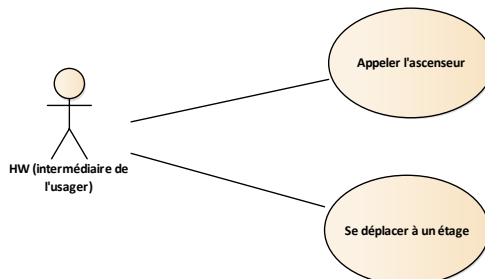
§1

49

Sprint 1 – Exigences

Comprendre les exigences fonctionnelles sur le logiciel ascenseur

- Dans les exigences fournies, sont utilisés à la fois le **langage naturel** et le formalisme graphique **UML**



§1

50

UML: formalisme de modélisation (1/2)

- UML (Unified Modeling Language) est un **langage de modélisation** graphique adapté à la description de logiciels
- Standard OMG depuis 1997, il est le plus utilisé dans l'ingénierie logicielle aujourd'hui
- Issus des méthodes Booch (Grady Booch), OMT (James Rumbaugh) et OOSE (Ivar Jacobson)
- Intègre les concepts objets
- Dernière version:
 - 2.5 (juin 2015): <http://www.omg.org/spec/UML/>

§1

51

UML: formalisme de modélisation (2/2)

- Il définit:
 - un ensemble de concepts, des **éléments de modèle**, possédant une sémantique claire (e.g. "Classe")
 - une "boîte à outils" de formalismes de modélisation graphique: les **diagrammes** UML représentant graphiquement les éléments de modèle (e.g. "Diagramme de classes")
- Formalisme très générique :
 - Pour tout type de système logiciel (ou pas ...)
 - Les mêmes éléments de modèle et diagrammes sont utilisés dans différentes activités de définition

=> formalisme très (trop?) abstrait

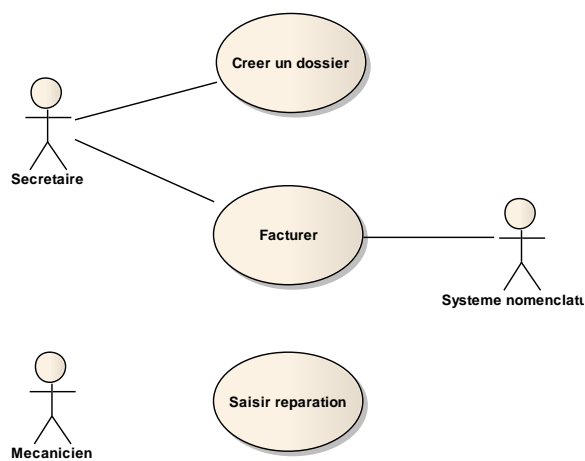
§1

52

Les 13 diagrammes

- de cas d'utilisation (use case diagram)
 - de séquence (sequence diagram)
 - de machine d'états (state machine diagram)
 - de classes (class diagram)
 - d'objets (object diagram)
 - de package (package diagram)
 - de déploiement (deployment diagram)
 - de composants (component diagram)
 - de structure composite (composite structure diagram)
 - de communication (communication diagram)
 - de timing (timing diagram)
 - d'activité (activity diagram)
 - de vue d'ensemble des interactions (interaction overview diagram)
- §1 53

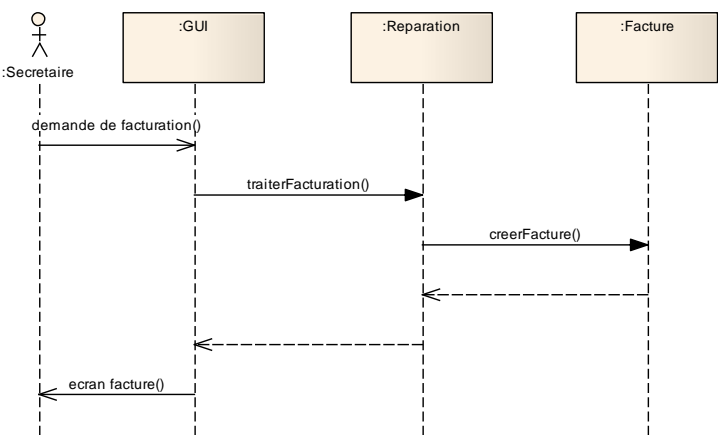
Diagramme de cas d'utilisation



§1

54

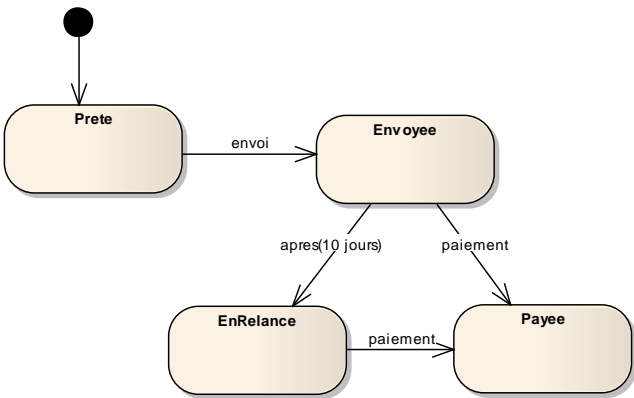
Diagramme de séquence



§1

55

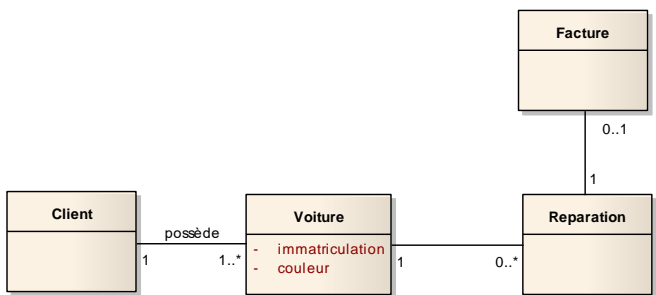
Diagramme de machine d'états



§1

56

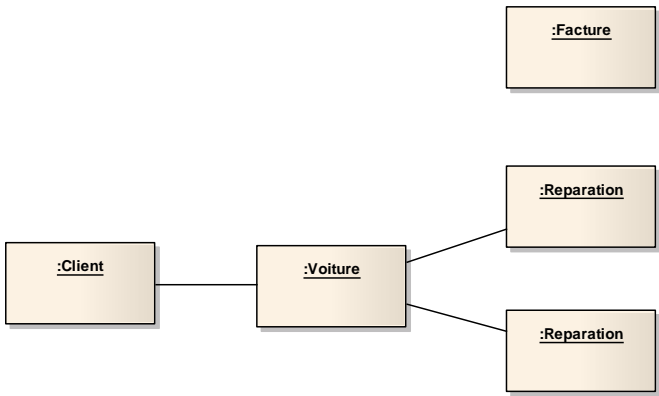
Diagramme de classes



§1

57

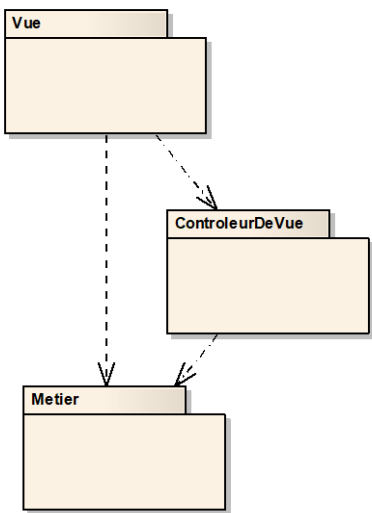
Diagramme d'objets



§1

58

Diagramme de package



§1

59

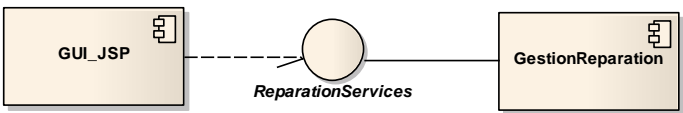
Diagramme de déploiement



§1

60

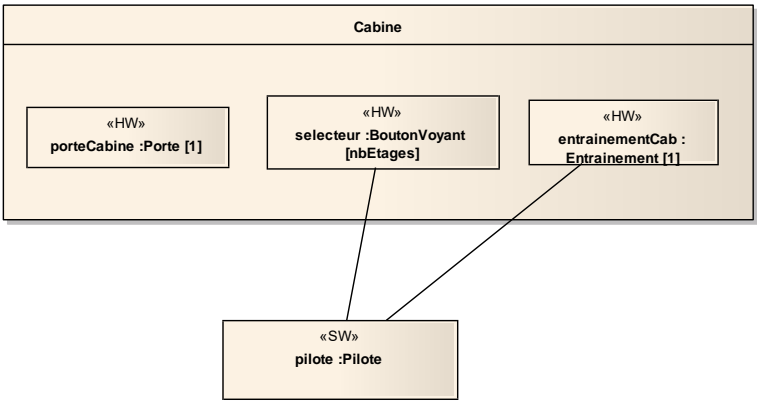
Diagramme de composants



§1

61

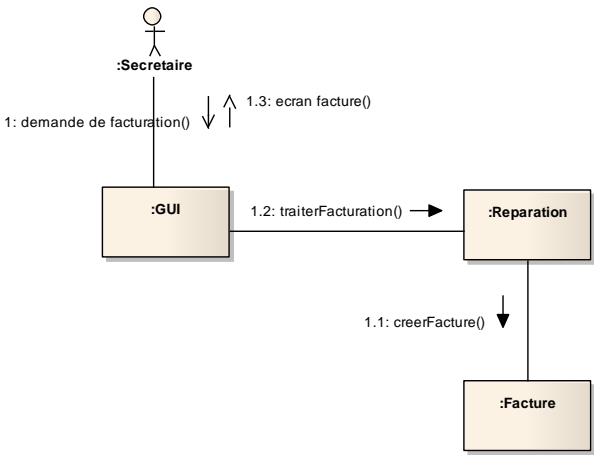
Diagramme de structure composite



§1

62

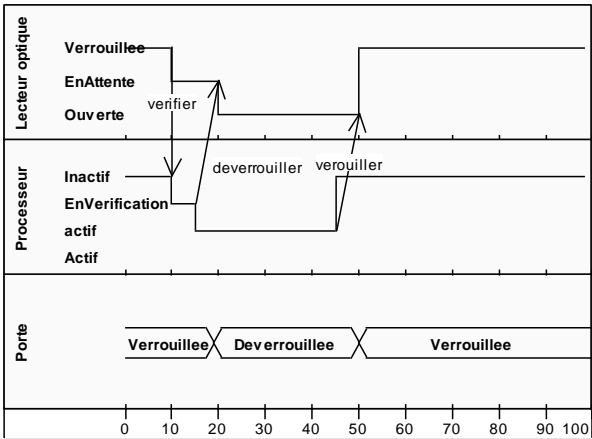
Diagramme de communication



§1

63

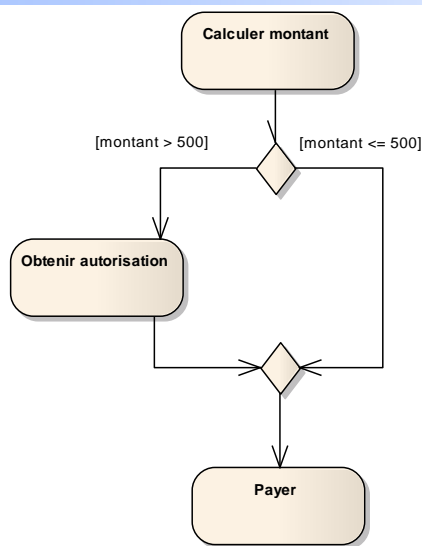
Diagramme de timing



§1

64

Diagramme d'activité

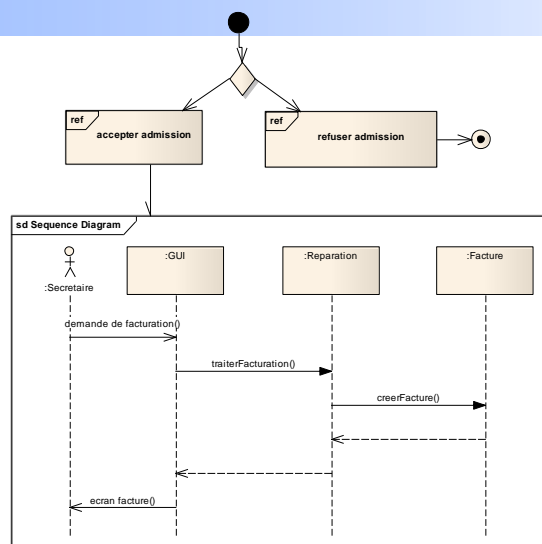


§1

65

65

Diagramme de vue d'ensemble des interactions

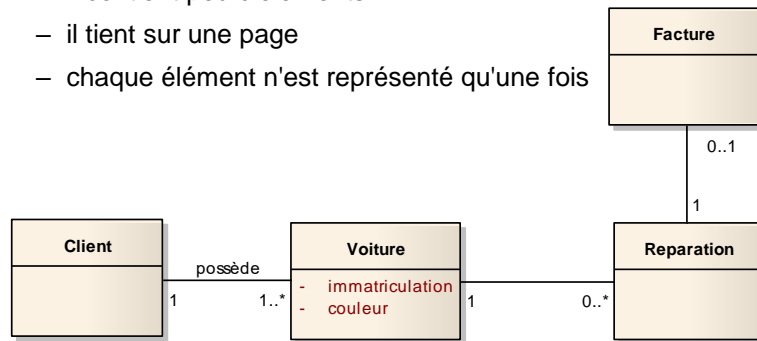


§1

66

Bonne pratique applicable à tout diagramme

- Pour être lisible, un diagramme doit être simple:
 - il contient peu d'éléments
 - il tient sur une page
 - chaque élément n'est représenté qu'une fois

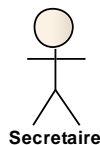


§1

67

UML et les exigences - La notion UML d'acteur (1/2)

- **Acteur** = "Classeur contenant les entités **extérieures** à un **sujet** et qui interagissent directement avec lui. Il caractérise un rôle joué par un utilisateur ou un ensemble d'utilisateurs en rapport avec le sujet."

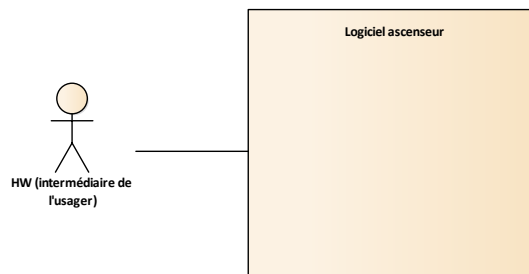


§1

68

UML et les exigences - La notion UML d'acteur (2/2)

- Dans le cas des exigences du logiciel ascenseur:
 - Le sujet (i.e. le système considéré) est le logiciel ascenseur
 - L'unique acteur est le HW

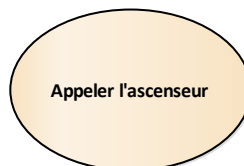


§1

69

UML et les exigences - La notion UML de cas d'utilisation

- **Cas d'utilisation** = "Unité cohérente de fonctionnalité (i.e. service de qualité) fournie par un système qui se manifeste par des séquences de messages échangés entre le système et un ou plusieurs acteurs, ainsi que par des actions accomplies par le système."



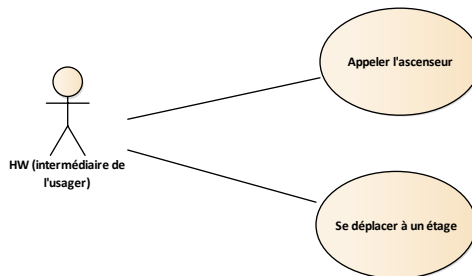
- Un cas d'utilisation raconte une "histoire" entre un acteur et le système
- Description orientée tests

§1

70

UML et les exigences - Diagramme de cas d'utilisation

- Un diagramme de cas d'utilisation représente:
 - les cas d'utilisation du logiciel
 - les acteurs associés, c'est-à-dire le ou les acteurs communiquant avec le logiciel pendant le déroulement de l'histoire



§1

71

UML et les exigences - Raconter l'histoire

- L'histoire est racontée de deux façons:
 - De façon exhaustive, en langage naturel:

- Le CU commence quand l'utilisateur appelle l'ascenseur en utilisant le bouton d'appel. Le HW envoie au logiciel un message "pression" avec pour paramètre le numéro du bouton d'appel de l'étage. [exception HS]
- Si la cabine est arrêtée, portes fermées :
 - o Si la cabine est arrêtée à un autre étage, le logiciel fait se déplacer la cabine à l'étage de l'appel de la façon suivante :
 - Le logiciel envoie au HW un message "monter", si la cabine se tient en-dessous, ou bien un message "descendre", si la cabine se tient au-dessus. Dans les 2 cas, le message contient le numéro de l'entraînement de la cabine.
 - A chaque nouvel étage, le HW envoie un message "nouvelEtage" au logiciel.
 - Si le service est pour cet étage, le logiciel fait s'arrêter la cabine en envoyant un message "arreter" au HW.
 - o Le logiciel fait s'ouvrir les portes (porte d'étage et porte de cabine) en envoyant au HW des messages "ouvrir".
 - o L'utilisateur peut entrer dans l'ascenseur. Le CU est terminé.
- Sinon l'appel n'est pas pris en compte. Le logiciel ne fait rien.

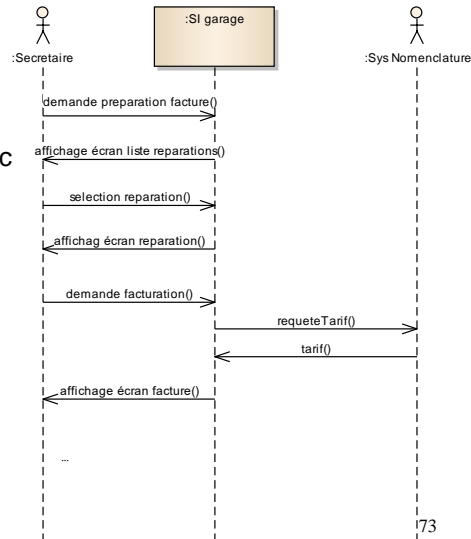
- De façon sélective, un **scénario** possible, en UML, sous la forme d'un diagramme de séquence

§1

72

La notion UML de scénario de cas d'utilisation

- Scénario = un chemin particulier dans un CU
- Un scénario est généralement décrit avec un **diagramme de séquence**



§1

73

Sprint 1 – Exigences Compléter les exigences (1/2)

- Définir un scénario significatif du cas d'utilisation "Appeler l'ascenseur":
 - Nom du scénario: Scénario normal – appel de la cabine, arrêtée à un autre étage
 - Préconditions: la cabine est arrêtée, portes fermées, à l'étage 1 – l'utilisateur est à l'étage 3
 - Début du scénario: Suite à la pression du bouton d'appel par l'utilisateur, le HW envoie un message "pression" au logiciel, avec pour paramètre le numéro du bouton d'appel de l'étage 3.

§1

74

Sprint 1 – Exigences

Compléter les exigences (2/2)

- Décrire ce scénario avec un diagramme de séquence



§1

75

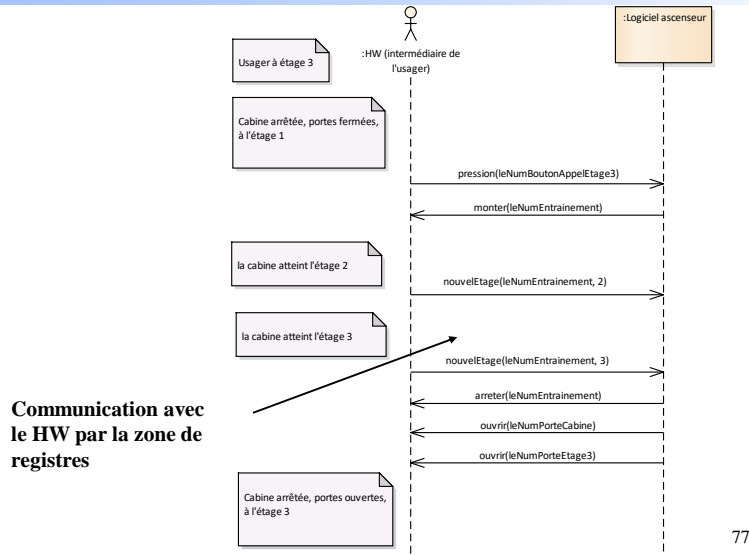
Page blanche

§1

76

Sprint 1 – Exigences

Définir le périmètre à développer dans le sprint (2/2)



Sprint 1 – Exigences

Définir le périmètre à développer dans le sprint (1/2)

- Périmètre réalisé dans le sprint 1:
 - Cas d'utilisation: "Appeler l'ascenseur"
 - Scénario: " Scénario normal – appel de la cabine, arrêtée à un autre étage"
 - Partie du scénario: pression du bouton d'appel et montée de la cabine

§1

78

Sprint 1 – Exigences

Définir le périmètre à développer dans le sprint (2/2)



§1

79

Sprint 1 – Exigences

Adapter le périmètre aux contraintes (1/2)

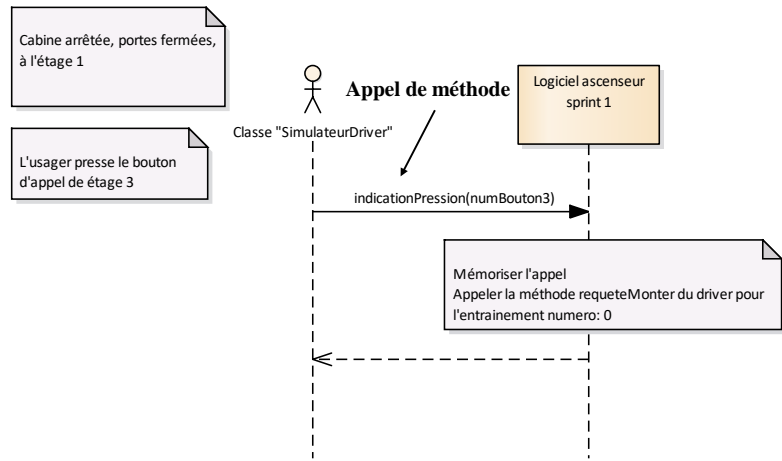
- Prise en compte de la contrainte "HW et dispositif de communication non disponibles"
- Stratégie adoptée:
 - Exécuter le logiciel sur un ordinateur de bureau standard (PC, Mac ...)
 - Mise en place d'une classe "SimulateurDriver", en dehors du logiciel ascenseur:
 - porteuse du main
 - responsable la simulation du driver pour l'envoi des messages provenant du HW: appel de méthodes sur le logiciel ascenseur

§1

80

Sprint 1 – Exigences

Adapter le périmètre aux contraintes (2/2)



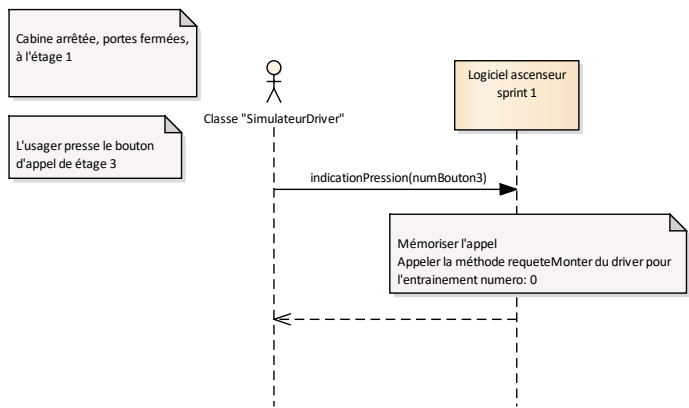
§1

81

Sprint 1 - Définition des tests de qualification (1/5)



- Quel est le problème ?



§1

82

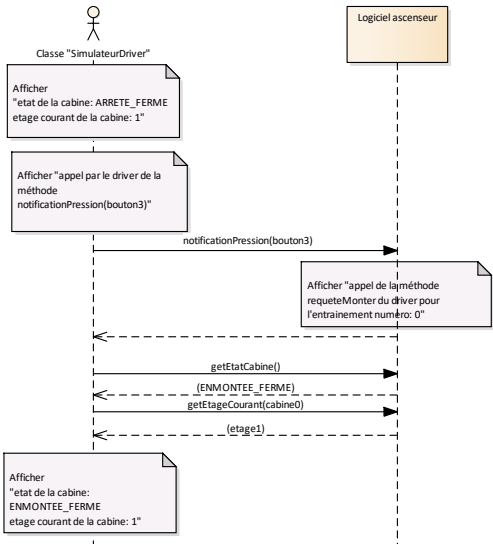
Sprint 1 - Définition des tests de qualification (2/5)

- Stratégie adoptée:
 - La classe SimulateurDriver initialise le logiciel: amène le logiciel dans l'état pré-requis scénario
 - L'appel des méthodes sur le driver est remplacé par des affichage console

§1

83

Sprint 1 - Définition des tests de qualification (3/5)



§1

84

Sprint 1 - Définition des tests de qualification (4/5)

```
package test_logiciel_ascenseur;
import logiciel_ascenseur.*;
public class SimulateurDriver {

    public static void main(String[] args) {
        // Initialisation: Ascenseur a 4 etages, dont le RDC
        ????.creerBouton(0,0);
        ...

        //Execution du scenario par simulation des drivers
        System.out.println("...
        ????.notificationPression(3);

        //Affichage état cabine - vérification post-conditions
        System.out.println("etat ...

    }
}
```

§1

85

Sprint 1 - Définition des tests de qualification (5/5)

- Sortie console:

```
run:
etat de la cabine: ARRETE_FERME
etage courant de la cabine: 1

appel par le driver de la méthode notificationPression(bouton3)
appel de la méthode requeteMonter du driver pour l'entrainement numero: 0

etat de la cabine: ENMONTEE_FERME
etage courant de la cabine: 1
```

§1

86

Sprint 1 – A vous de jouer ...

- Spécifier les exigences: compléter et adapter au périmètre
- Définir les tests de qualification
- Concevoir: réfléchir à la structuration du logiciel ascenseur en unités de codes (en classes java)
- Coder
- Exécuter les tests de qualification

§1

87

Sprint 1 – Concevoir Rappels des objectifs

- Concevoir le logiciel ascenseur réalisé dans le sprint 1:
 - De qualité:
 - Répondant aux exigences (qualité externe)
 - Maintenable et évolutif (qualité interne)
 - En respectant les délais:
 - Début-Fin: 17/11-23/11
 - Revue de sprint: 24/11
 - Démo
 - Explication de la structuration du code
 - Recherche de pistes d'amélioration de la maintenabilité et de l'évolutivité du code avec l'architecte logiciel

§1

88

Sprint 1 – Concevoir

Contraintes de conception

- Le code est réalisé en Java
- Seules des notions acquises dans le cours POO au 10/11/16 seront utilisées

§1

89

Sprint 1 – Concevoir

Bonne pratiques à utiliser (1/2)

- Structurer 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

§1

90

Sprint 1 – Concevoir Bonne pratiques à utiliser (2/2)

- Concevoir un code homogène: "dupliquer" une bonne façon de coder
=> choisir un **pattern de conception** de nos classes, en accord avec nos objectifs, nos contraintes et nos bonnes pratiques



§1

91