Chapitre 4

Partie1: Spécification et préparation des tests de validation

Enseignant: Ahlem Baccouche
ahlem.baccouche@ihec.ucar.tn

Plan Chapitre 4- Partie1

1. Motivations et objectifs de la séance
2. Spécification
3. Préparation des tests de validation
4. Mise en pratique en TD

1 Motivations et objectifs de la séance

1. Contexte — Qui demande une spécification ?
2. Intérêt — Pourquoi spécifier ?
3. Intérêt — Pourquoi préparer des tests ?
4. Contenu — Différentes vues d'un système informatique
5. Contenu — Quels sont les objectifs de la séance ?

1.1 Contexte — Qui demande une spécification ?

- *Toute satisfaction d'un besoin est construite à partir de la représentation, de la formalisation de ce besoin*
  - *Plus cette formalisation est riche, plus la solution est riche et adaptée*
- Éviter de « *remplacer l'énoncé d'un problème par l'énoncé d'une solution* »
  + Une analyse raisonnablement poussée élargit l'éventail des solutions

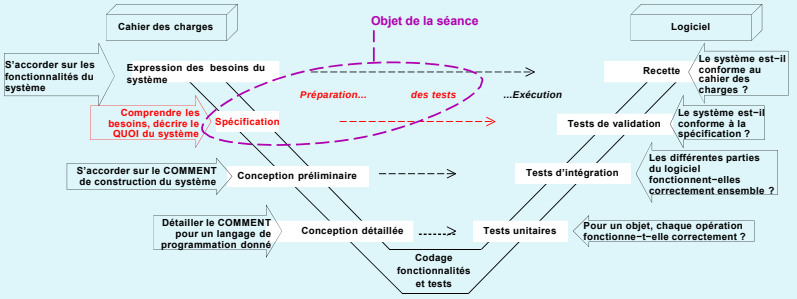
1.2 Intérêt — Pourquoi spécifier ?

Leslie Lamport, ACM A.M. Turing Award, 2013 :
Programming Should Be More Than Coding [Lamport, 2015]

- *Some misconceptions*
  - *Notation mathématique non possible [ou au moins semi-formelle]*
    Pour représenter le besoin de l'utilisateur
- Remarques pertinentes
  - *Ne pas savoir ce qu'un programme doit faire signifie que nous devons réfléchir encore plus, ce qui signifie qu'une spécification est encore plus importante.*
  - *Une spécification de ce que fait le code devrait indiquer tout ce que quiconque doit savoir pour utiliser le code*
    - Spécification = le « quoi »
  - *Si vous ne commencez pas par une spécification, chaque morceau de code que vous écrivez est une parcelle [ou au moins quelques morceaux]*

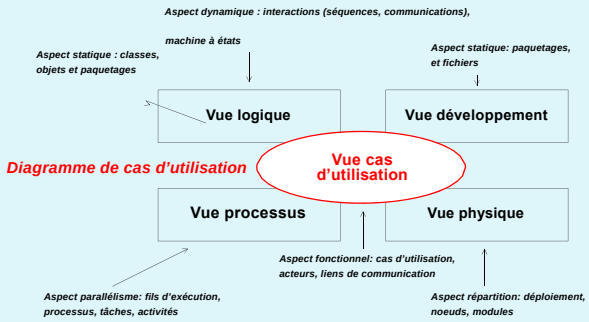
1.3 Intérêt — Pourquoi préparer des tests ?

- Chaque sprint commence par une analyse/spécification des fonctionnalités
- Chaque sprint se termine par la validation des fonctionnalités



1.4 Contenu — Différentes vues d'un système informatique

- Par exemple, les « 4+1 vues » [Kruchten, 1995]
  - L'utilisateur final est au centre



1.5 Contenu — Quels sont les objectifs de la séance ?

- Spécification
  - Définir et expliquer le rôle de la spécification
  - Modéliser les fonctionnalités dans un diagramme de cas d'utilisation
  - Formaliser leurs préconditions et postconditions
- Préparation des tests de validation
  - Définir et expliquer le rôle de la préparation des tests de validation
  - Formaliser les scénarios des tests de validation dans des tables de décision

2 Spécification

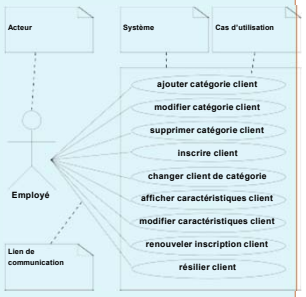
1. Modélisation des fonctionnalités dans un diagramme de cas d'utilisation
2. Modélisation des préconditions et postconditions d'un cas d'utilisation
3 Éléments de méthodologie

2.1 Modélisation des fonctionnalités dans un dia- gramme de cas d'utilisation

2.1.1 Diagrammes de cas d'utilisation
2.1.2 Acteur
2.1.3 Relation de généralisation spécialisation entre acteurs
2.1.4 Lien de communication et relation entre cas d'utilisation
2.1.5 Médiathèque, cas d'utilisation pour la gestion des clients
2.1.6 Médiathèque, autres cas d'utilisation

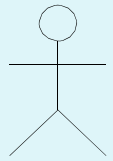
2.1.1 Diagrammes de cas d'utilisation

- Les fonctionnalités sont modélisées par des cas d'utilisation
- Un diagramme de cas d'utilisation définit :
  - Le système
  - Les acteurs
  - Les cas d'utilisation (fonctionnalités)
  - Les liens entre acteurs et cas d'utilisation
    - Quel acteur accède à quel cas d'utilisation ?
- Un modèle de cas d'utilisation se définit par :
  - Des diagrammes de cas d'utilisation
  - Une description textuelle des scénarios d'utilisation
    - Dans les études de cas que nous donnons, les cas d'utilisation sont généralement assez simples pour éviter une description détaillée dans des scénarios



2.1.2 Acteur

- Un acteur représente une personne, un périphérique ou un autre système qui joue un rôle (interagit) avec le système
  - « Les clients s'inscrivent auprès d'un employé de la médiathèque, et empruntent et rendent un document par l'intermédiaire d'un employé de la médiathèque. »

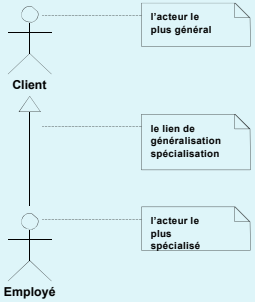


Employé

<< acteur >>
Employé

2.1.3 Relation de généralisation spécialisation entre acteurs

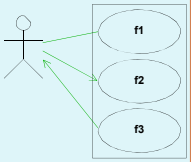
- La généralisation spécialisation est appelée héritage en programmation (EST-UN)
  - « La médiathèque met à la disposition des clients des ordinateurs pour qu'ils consultent le catalogue, leurs emprunts, et puissent mettre à jour leur adresse. »



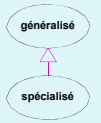
- Rôle de la généralisation spécialisation = factoriser des fonctionnalités
  - « Tout ce que le client peut faire, l'employé peut le faire »

2.1.4 Lien de communication et relation entre cas d'utilisation

- Un cas d'utilisation modélise une suite d'interactions entre un acteur et le système
  - Le plus souvent pour nous, une seule interaction
  - Possibilité d'orienter un lien pour préciser si c'est une requête (de l'acteur) ou une notification (du système)

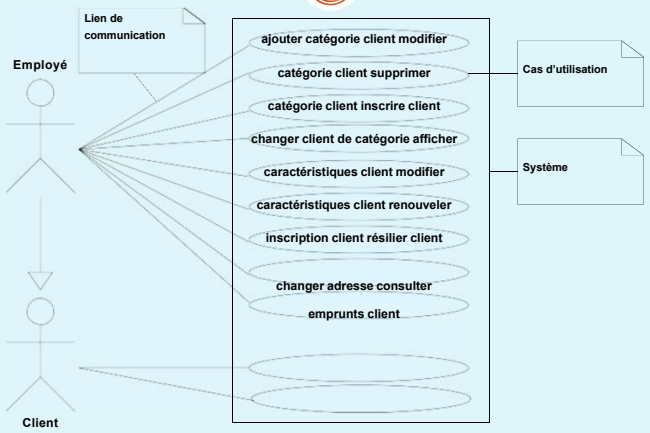


- UML propose les trois concepts suivants
  - Généralisation spécialisation de cas d'utilisation
    - Un cas d'utilisation est un type spécial d'un autre
      - Le cas d'utilisation spécialisé diffère quelque peu du généralisé

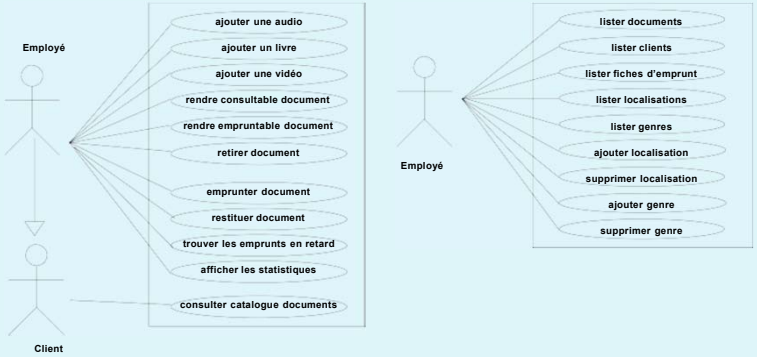


  - Inclusion (<<includes>>) :
    - un cas d'utilisation inclut une séquence d'actions qui représente un autre cas d'utilisation : le premier « inclut » le second d'utilisation
  - (controversé) Extension (<<extends>>) :
    - Un cas d'utilisation peut réutiliser un cas d'utilisation complet, mais la réutilisation est *optionnelle* et dépend des conditions d'exécution

2.1.5 Médiathèque, cas d'utilisation pour la gestion des clients



2.1.6 Médiathèque, autres cas d'utilisation



2.2 Modélisation des préconditions et postcondi- tions d'un cas d'utilisation

2.2.1 Précondition et postcondition

- Description d'un cas d'utilisation = ce que font l'acteur et le système
  - Pour nous, de manière « préformattée »
    - Données en entrée et valeur retournée
      - + Conditions d'exécution (préconditions et postconditions)
- Précondition : conditions qui doivent être vraies lorsque le cas d'utilisation est appelé
  - Exprimée sur les données en entrée et l'état du système avant l'appel
  - Selon la spécification UML, le comportement n'est pas défini lorsque la précondition n'est pas satisfaite
    - Pour nous, précondition non satisfaite =⇒ message d'erreur à l'acteur
- Postcondition : conditions qui doivent être vraies lorsque le cas d'utilisation est terminé, si les préconditions étaient satisfaites
  - Exprimée sur les données en entrée, l'état du système avant et après l'appel, et la valeur de retour
    - Pour nous, postcondition... ou message d'erreur

2.2.2 Précondition et postcondition — Exemple simple

- Cas d'utilisation « Ajouter un document Audio » de l'étude de cas Médiathèque
  - Précondition
    - ∧ code du document bien formé (¬ null ∧ ¬ vide)
      ∧ informations sur le document Audio bien formées (¬ null ∧ ¬ vide)
      ∧ document avec le code donné n'existe pas
  - Postcondition
    - document Audio avec le code donné existe

0. ∧ (ET), ∨ (OR), et ¬ (négation)

2.2.3 Précondition et postcondition — Exemple plus complexe

- Cas d'utilisation « Emprunter un document » de l'étude de cas Médiathèque
  - Précondition
    - ∧ nom et prénom du client bien formés (¬ null ∧ ¬ vide)
    - ∧ code du document bien formé (¬ null ∧ ¬ vide)
    - ∧ client existe ∧ client sans emprunt en retard ∧ client pas au maximum de ses emprunts
    - ∧ document existe ∧ document empruntable ∧ document non emprunté
  - Postcondition
    - ∧ +1 sur le nombre d'emprunts en cours du client
    - ∧ document emprunté

0. ∧ (ET), ∨ (OR), et ¬ (négation)

2.2.5 Élection des cas d'utilisation pour le sprint 1

- Constat = beaucoup de cas d'utilisation (≥ quantité de travail acceptable pour le sprint 1)
  - Pour chaque cas d'utilisation, pré-/post-conditions, conception, programmation, etc.
- Solution = discuter (avec le client) du niveau de priorité des cas d'utilisation
  - Par analogie avec la RFC 2119 de l'IETF définissant trois niveaux [Bradner, 1997]
    - must *(required, shall)*, should *(recommended), and* may *(optional)*
- Pour nous, trois niveaux de priorité : Haute, Moyenne, basse
  - Raisonnement de bon sens, à avoir avec le client
    - Pour retirer une entité du système, elle doit y être
    - Pour lister les entités d'un type donné, elles doivent y être
    - Lister les entités peut être intéressant pour tester
    - Est-il possible d'avoir le respect des règles de gestion choisies sans retirer ni lister certaines entités ?
  - Donc, sans doute : ajouter = Haute, lister = Moyenne, et retirer = basse

2.3 Éléments de méthodologie

- Identifier les acteurs qui utilisent/exécutent des fonctionnalités
  - Si cela s'avère pertinent, organiser les acteurs par relation de généralisation spécialisation
- Pour chaque acteur, rechercher les cas d'utilisation du système
- Dans notre étude de cas (seulement)
  - Il n'est pas nécessaire d'utiliser de généralisation spécialisation de cas d'utilisation
  - Il n'est pas nécessaire d'utiliser « include » ou « extends »
- Choisir les cas d'utilisation du Sprint
- Pour chaque cas d'utilisation non trivial
  - Déterminer les données en entrée et la valeur de retour
  - Formuler la précondition et la postcondition

3 Préparation des tests de validation

1. Définition et rôle des tests de validation
2. Autre exemple de table de décision
3 Éléments de méthodologie

3.1 Définition et rôle des tests de validation

- Tests de validation = vérifications des fonctionnalités de la spécification[1]
  - Ce sont donc des tests fonctionnels
  - À partir de la spécification (quoi) et non à partir de la réalisation (comment)
  - Il est possible de les préparer dès maintenant pour les exécuter avant la livraison
- Préparation = écrire des jeux de tests
  - Tests des cas normaux
    Tests hors limites : les erreurs sont correctement signalées, voire récupérées
    Tests aux limites : considérer les bornes des domaines de valeur des entrées
  - Pour nous, écriture sous la forme de tables de décisions
    - Les sorties sont, à tout moment, uniquement définies par les entrées ainsi que l'état du système
      Donc, à partir des préconditions et postconditions

1. À ne pas confondre avec la vérification de la correction (en anglais, *soundness*)

3.1.1 Table de décisions

- Cas d'utilisation « Ajouter un document Audio »
  - Précondition = code du document bien formé (¬ null ∧ ¬ vide) ∧ informations sur le document Audio bien formées (¬ null ∧ ¬ vide) ∧ document avec le code donné n'existe pas
  - Postcondition = document Audio avec le code donné existe

		1	2	3	4
Précondition	Code du document bien formé (non null et non vide)	F	T	T	T
	Informations sur l'audio bien formées (non null et non vide)		F	T	T
	Document avec ce code inexistant			F	T
Postcondition	Ajout effectué	F	F	F	T
	Nombre de jeux de tests	2	$n \times 2$	1	1

n : est le nombre de champs autre que le code du document : tous des chaînes de caractères

3.2 Autre exemple de table de décision

- Cas d'utilisation « Emprunter » de l'étude de cas Médiathèque
  - Précondition = client existe ∧ client sans emprunt en retard ∧ client pas au maximum de ses emprunts ∧ document existe ∧ document empruntable ∧ document non emprunté
  - Postcondition = +1 sur le nombre d'emprunt en cours du client ∧ document emprunté ∧ +1 sur le nombre d'emprunts du document

Version simplifiée : précondition sur les données en entrée « bien formées » ignorées

			1	2	3	4	5	6	7
Précondition	Client	inscrit	F	T	T				T
	Emprunts du client	sans retard		F	T				T
		nb < max			F				T
	Document	existant				F	T	T	T
		empruntable					F	T	T
		disponible						F	T
Postcondition	Emprunt accepté		F	F	F	F	F	F	T
	Nombre de jeux de tests		1	1	3^2	1	1	1	3^2

2. Pour les différentes catégories de clients

3.3 Éléments de méthodologie

- Choisir les tests (c'est une difficulté inhérente au développement logiciel)
  - Trop de tests à préparer, programmer et exécuter pour la couverture complète de tous les états possibles du système
- Notre démarche dans le module
  - Sélectionner les cas d'utilisation
    - ∧ Haute priorité établie avec le client par quelques règles de bon sens
    - ∧ Précondition et postcondition non triviales
    - ∧ Avec règle de gestion importante[3] et complexe pour le système
  - Écrire les tables de décisions de ces cas d'utilisation
    - Pensez à calculer le nombre minimum de jeux de tests
      - Fonction des domaines de valeurs des données en entrée et en sortie

3. Ne pas confondre « urgence » et « importance »

4 Mise en pratique en TD

- À partir du cahier des charges
  - Diagramme de cas d'utilisation, puis priorités des cas d'utilisation
  - Pré-/post-conditions des cas d'utilisation de Haute priorité
  - Table de décision des tests de validation des cas d'utilisation
- **Rendu de la séance en TD : Dépôt Git + doc. PDF sur classroom**
- Verifies bien que le fichier PDF est généré!
- Compléments « Pour aller plus loin » (Tests logiciels préparation des tests de validation)
- Préparation de la prochaine séance :
  - Prérequis « Éléments de cours et exercice sur le diagramme de classes »

Références I

Bradner, S. (1997).
RFC 2119—Key words for use in RFCs to Indicate Requirement Levels.
Best current practice, network working group, IETF.

Clave, A. (2016).
UML au service de l'analyse des métiers (Business Analysis).
ENI Éditions.

Kruchten, P. (1995).
The 4+1 View Model of Architecture.
IEEE Software, 12(6) :42–50.

Lamport, L. (2015).
Programming Should Be More Than Coding.
Stanford EE Computer Systems Colloquium.

Conan, D. and Gibson, P. (2022)
Introduction au Génie logiciel pour applications orientées objet