



# CSC4102 : Spécification et préparation des tests de validation

Denis Conan, avec Paul Gibson

Janvier 2025





# Sommaire

1. Motivations et objectifs de la séance
2. Spécification
3. Préparation des tests de validation
4. Mise en pratique en TP (2h) + HP (3h)

# 1 Motivations et objectifs de la séance

- 1.1 Pourquoi spécifier ?
- 1.2 Pourquoi préparer des tests ?
- 1.3 Contenu — Différentes vues d'un système informatique
- 1.4 Contenu — Quels sont les objectifs de la séance ?

# 1.1 Pourquoi spécifier ?

- Satisfaction d'un besoin est construite à partir 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 (QUOI) par l'énoncé d'une solution (COMMENT)

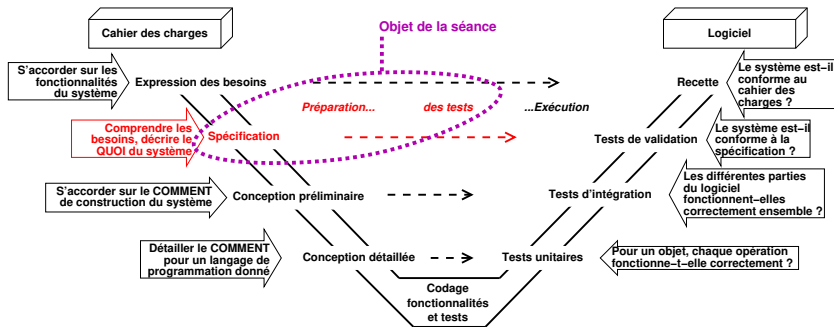
Leslie Lamport, ACM A.M. Turing Award, 2013 :  
*Programming Should Be More Than Coding* [Lamport, 2015]



- *Some misconceptions*
  - *We can't specify mathematically [at least semi-formally] what the user wants*
- *Some relevant claims*
  - *Not knowing what a program should do means we have to think even harder, which means that a spec is even more important*
  - *A specification of what the code does should say everything that anyone needs to know to use the code*
  - *If you don't start with a specification, every piece of the code you write is a patch [or at least some pieces]*

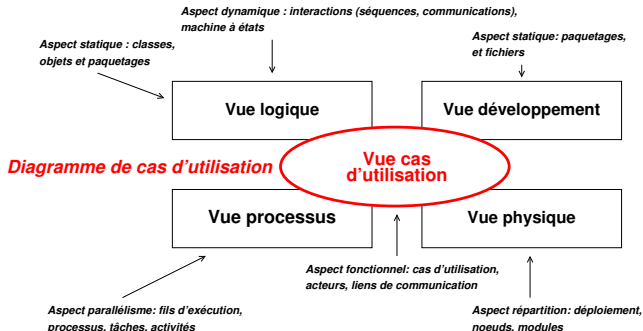
## 1.2 Pourquoi préparer des tests ?

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



## 1.3 Contenu — Différentes vues d'un système informatique

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



## 1.4 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

- 2.1 Modélisation des fonctionnalités dans un diagramme de cas d'utilisation
- 2.2 Modélisation des préconditions et postconditions d'un cas d'utilisation
- 2.3 Éléments de méthodologie



## 2.1 Modélisation des fonctionnalités dans un diagramme 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

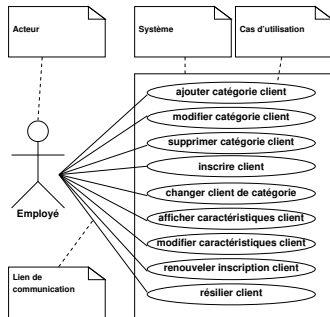
- Les fonctionnalités sont modélisées par des cas d'utilisation

- 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 ?

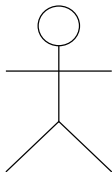
- Un modèle de cas d'utilisation se définit par :

- 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. »

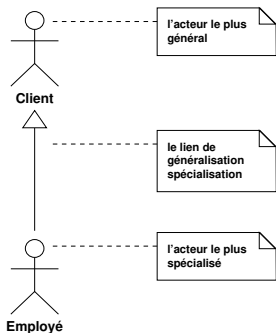


**Employé**

**<< acteur >>  
Employé**

## 2.1.3 Relation de généralisation spécialisation entre acteurs

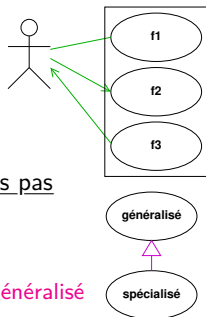
- La **généralisation spécialisation** est appelée héritage en programmation (**EST-UN**)
  - « La médiathèque met à la disposition des clients des ordinateurs pour qu'ils consultent le catalogue, leurs emprunts, et puissent mettre à jour leur adresse. »



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

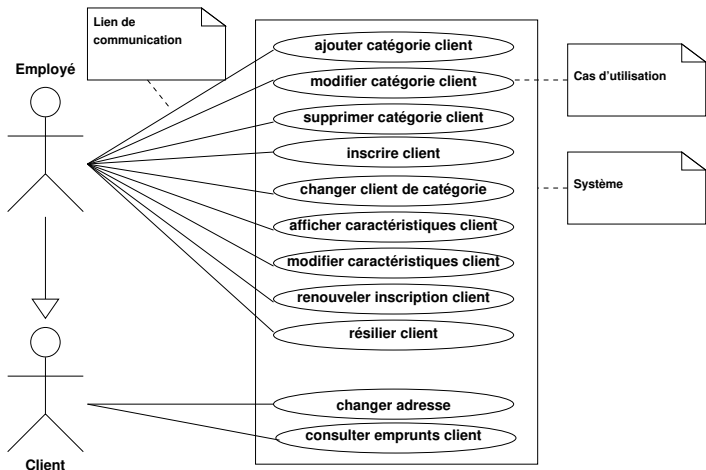
## 2.1.4 Lien de communication et relation entre cas d'utilisation

- 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 que nous n'utiliserons pas

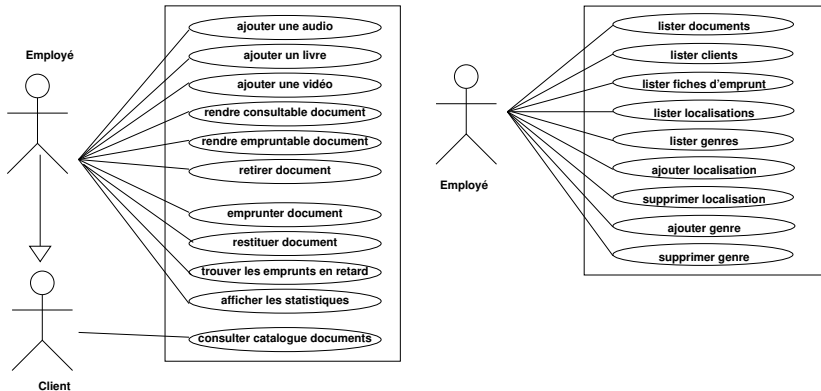


- Généralisation spécialisation de cas d'utilisation
  - Un cas d'utilisation est un type spécial d'un autre
    - Le cas d'utilisation spécialisé diffère quelque peu du généralisé
- Inclusion (<<includes>>) :
  - un cas d'utilisation inclut une séquence d'actions qui représente un autre cas d'utilisation : le premier « inclut » le second cas 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 postconditions d'un cas d'utilisation

- 2.2.1 Précondition et postcondition
- 2.2.2 Précondition et postcondition
  - Exemple simple
- 2.2.3 Précondition et postcondition
  - Exemple plus complexe
- 2.2.4 Pré-/post-condition avec généralisation spécialisation de cas d'utilisation \*
- 2.2.5 Élection des 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 precondition n'est pas satisfaite
    - Pour nous, precondition non satisfaite  $\implies$  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 vraie 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
    - $\wedge$  code du document bien formé ( $\neq \text{null} \wedge \neq \text{vide}$ )
    - $\wedge$  informations sur le document Audio bien formées ( $\neq \text{null} \wedge \neq \text{vide}$ )
    - $\wedge$  document avec le code donné n'existe pas
  - Postcondition
    - document Audio avec le code donné existe

---

0.  $\wedge$  (ET),  $\vee$  (OR), et  $\neg$  (négation)

## 2.2.3 Précondition et postcondition — Exemple plus complexe

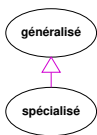
- Cas d'utilisation « Emprunter un document » de l'étude de cas Médiathèque
  - Précondition
    - $\wedge$  nom et prénom du client bien formés ( $\neq \text{null} \wedge \neq \text{vide}$ )
    - $\wedge$  code du document bien formé ( $\neq \text{null} \wedge \neq \text{vide}$ )
    - $\wedge$  client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

---

0.  $\wedge$  (ET),  $\vee$  (OR), et  $\neg$  (négation)

## 2.2.4 Pré-/post-condition avec généralisation spécialisation de cas d'utilisation \*

- Généralisation spécialisation = remplacer le généralisé par le spécialisé
- On peut demander moins à l'acteur, qui est ici le « client »
  - c.-à-d., précondition du spécialisé = précondition du généralisé **affaiblie**
  - c.-à-d., **précondition du généralisé  $\implies$  précondition du spécialisé**
- P.ex., cas d'utilisation généralisé avec précondition  $pre_g \triangleq arg \in \{1, 2, 3\}$   
et cas d'utilisation spécialisé avec précondition  $pre_s \triangleq arg \in \{1, 2, 3\} \vee arg \in \{0, 4, 5\}$   
Puisque  $\{1, 2, 3\} \subseteq \{0, 1, 2, 3, 4, 5\}$ , nous avons ( $pre_g \implies pre_s$ )
- Le cas d'utilisation, qui est ici le « fournisseur », peut fournir plus/être plus précis dans sa réponse/la valeur retournée
  - c.-à-d., postcondition du spécialisé = postcondition du généralisé **renforcée**
  - c.-à-d., **postcondition du généralisé  $\longleftarrow$  postcondition du spécialisé**
- P.ex., cas d'utilisation généralisé avec postcondition  $post_g \triangleq vretour \in \mathbb{N}$   
et cas d'utilisation spécialisé avec postcondition  $post_s \triangleq vretour \in \mathbb{N} \wedge vretour \geq 0$   
Puisque  $\mathbb{N} \supseteq \mathbb{N}^*$ , nous avons ( $post_g \longleftarrow post_s$ )



## 2.2.5 Élection des cas d'utilisation

- Constat = beaucoup de cas d'utilisation ( $\geq$  quantité de travail acceptable)
  - 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 de tester 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
  - 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 de priorité haute
- Pour chaque cas d'utilisation de priorité haute et non trivial
  - Déterminer les données en entrée et la valeur de retour
  - Formuler la précondition et la postcondition

## 3 Préparation des tests de validation

- 3.1 Définition et rôle des tests de validation
- 3.2 Autre exemple de table de décision
- 3.3 Éléments de méthodologie

## 3.1 Définition et rôle des tests de validation

- Tests de validation = vérifications des fonctionnalités de la spécification<sup>1</sup>
  - 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 ( $\neq \text{null} \wedge \neq \text{vide}$ )  $\wedge$  informations sur le document Audio bien formées ( $\neq \text{null} \wedge \neq \text{vide}$ )  $\wedge$  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 =  $\text{client existe} \wedge \text{client sans emprunt en retard} \wedge \text{client pas au maximum de ces emprunts} \wedge \text{document existe} \wedge \text{document empruntable} \wedge \text{document non emprunté}$
  - Postcondition =  $+1 \text{ sur le nombre d'emprunt en cours du client} \wedge \text{document emprunté} \wedge +1 \text{ sur le nombre d'emprunts du document}$

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 <sup>2</sup>	1	1	1	3 <sup>2</sup>	

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<sup>3</sup> 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 TP (2h) + HP (3h)

- À partir du cahier des charges (lu avant ce cours)
  - 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 HP : diagramme(s) PlantUML + `readme.md`
  - Vérifiez bien que les images `svg` et `png` sont à jour !
- Suivi 1 : spécification et 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.

Kruchten, P. (1995).

The 4+1 View Model of Architecture.

*IEEE Software*, 12(6) :42–50.

Lampert, L. (2015).

Programming Should Be More Than Coding.

Stanford EE Computer Systems Colloquium.