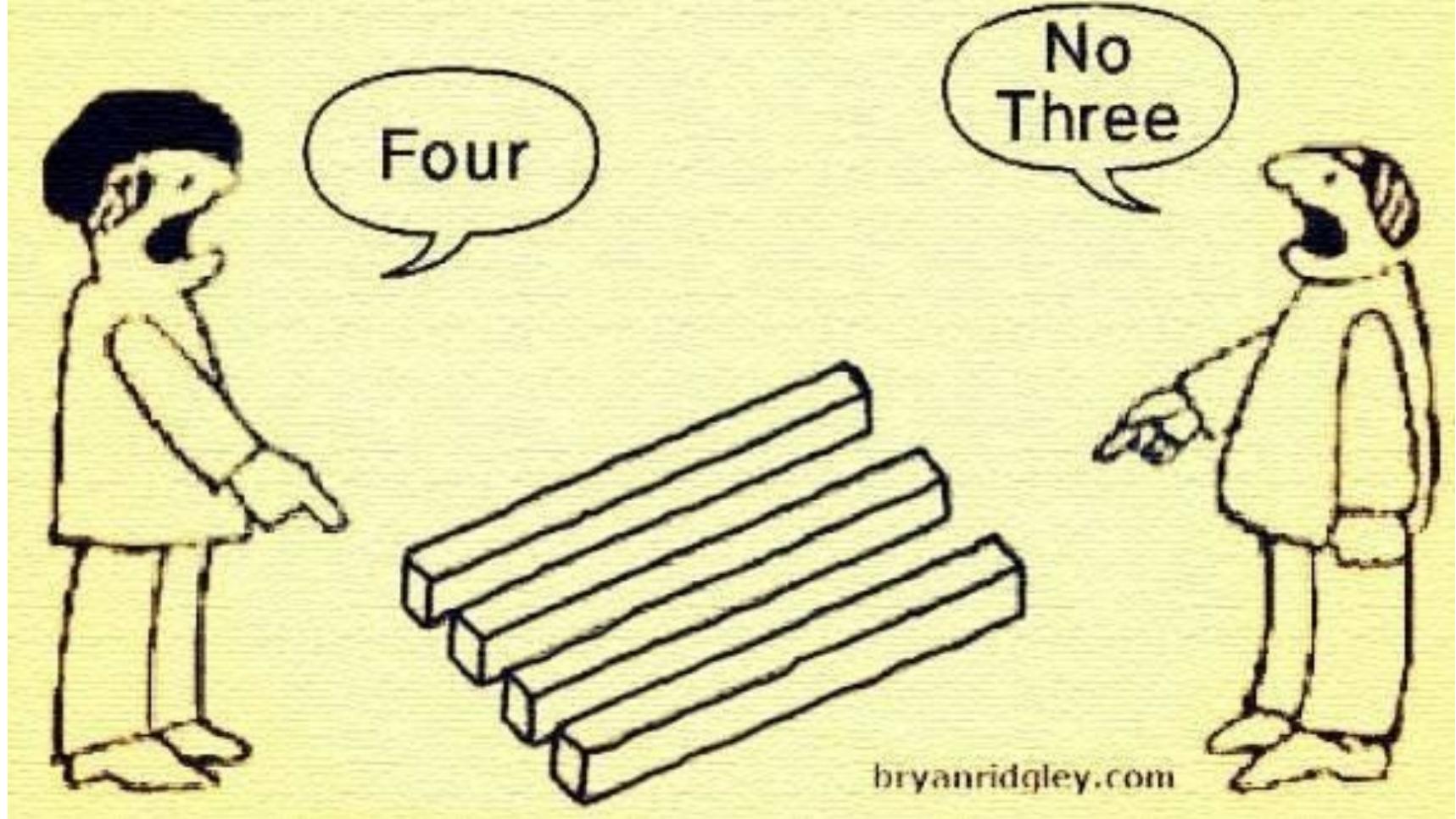


Reality can be so complex that equally valid observations from differing perspectives can appear to be contradictory.





TRIPOS CONCENTRATIONS  
• Systèmes d'information  
• Logiciels industriels  
• Systèmes interactifs

PERSPECTIVES D'EMPLOI

Grâce à leur formation multidisciplinaire, les diplômés en génie logiciel de l'ETS sont habilités à travailler dans :  
• conception et développement de produits interactifs  
• contrôle et assurance qualité des systèmes  
• gestion et coordination de projets  
• intégration et coordination des technologies  
• maintenance et amélioration  
• simulateurs

# LOG410 – Analyse de besoins et spécifications

## Cours #8

### Gestion des changements, CMMI-DEV

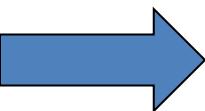
Enseignants: Alain Dion, ing. et Yves Durocher, ing., M. ing.

# Retour sur le dernier cours

- Processus d'affaires
- Gestion de produit
- Gestion de la portée du projet
- Gestion du client

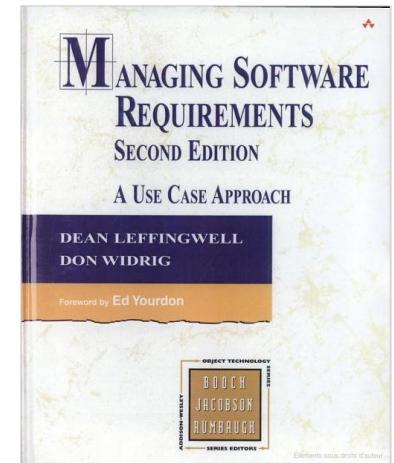


# Plan du cours



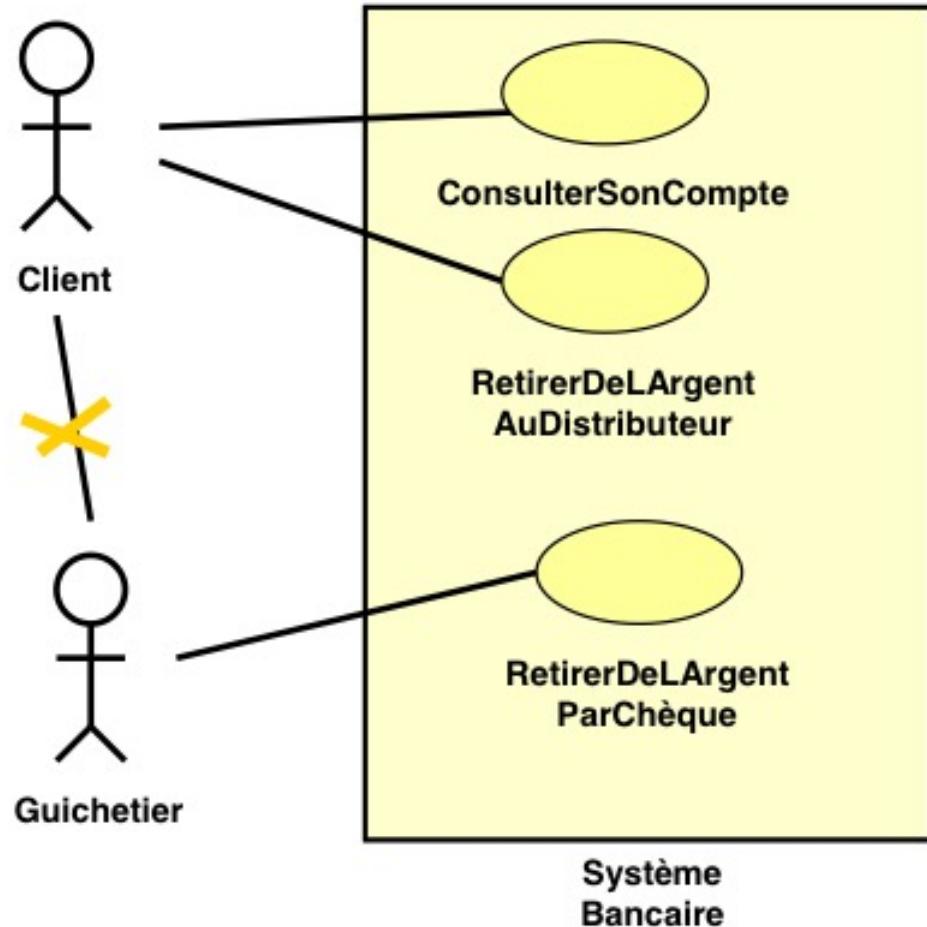
- Clarifier les cas d'utilisation (CU)
- Exercice – Étiquettes intelligentes
- Précisions concernant les exigences
- Des CU aux cas de tests
- Gestion des changements
- Modèle CMMI-DEV

Chapitre 21



# Lien entre acteurs

- Ne doit pas être ajouté!
- Est extérieur au système.
- UML se concentre:
  - ❖ Sur la description du système
  - ❖ L'interaction système - extérieur

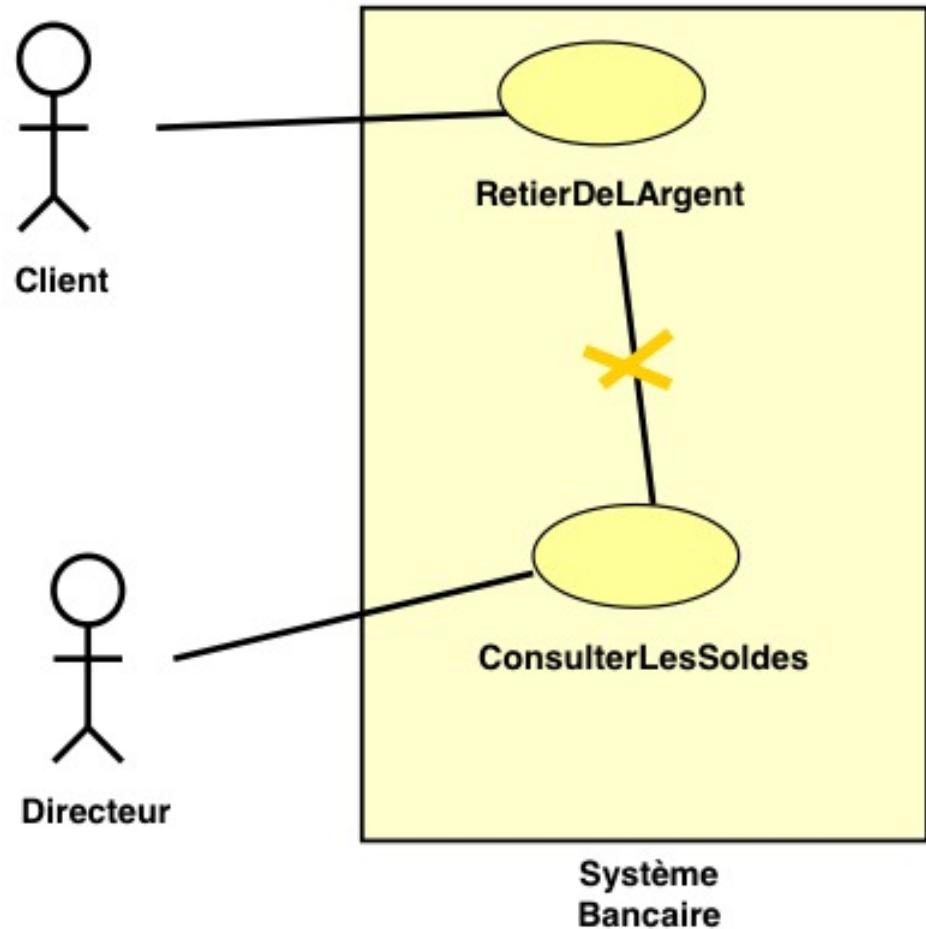


Référence: Page 30

Présentation « Le langage UML : Les cas d'utilisation »,  
Lydie du Bousquet,  
[Lydie.du-bousquet@imag.fr](mailto:Lydie.du-bousquet@imag.fr)

# Lien entre les CU

- Ne doit pas être ajouté!
- Aucun lien entre les CU.
- UML se concentre:
  - ❖ Sur la description du système
  - ❖ L'interaction système - extérieur

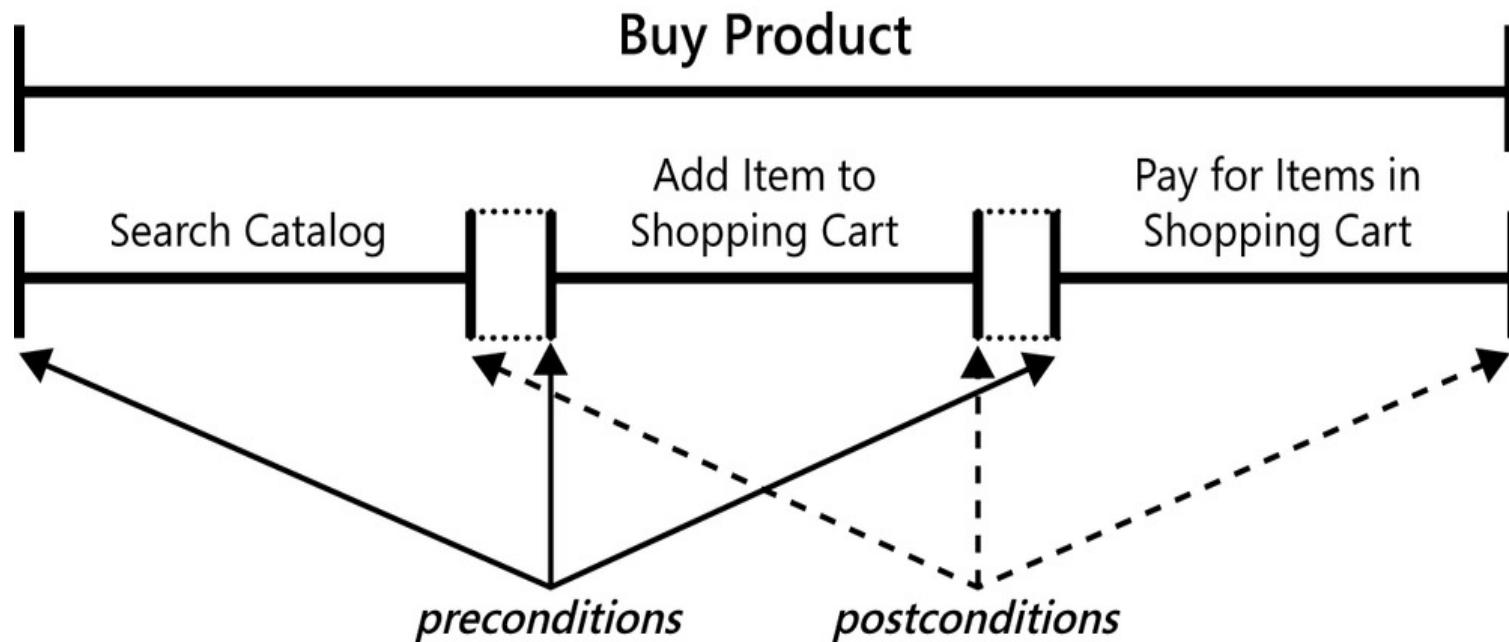


Référence: Page 31

Présentation « Le langage UML : Les cas d'utilisation »,  
Lydie du Bousquet,  
[Lydie.du-bousquet@imag.fr](mailto:Lydie.du-bousquet@imag.fr)

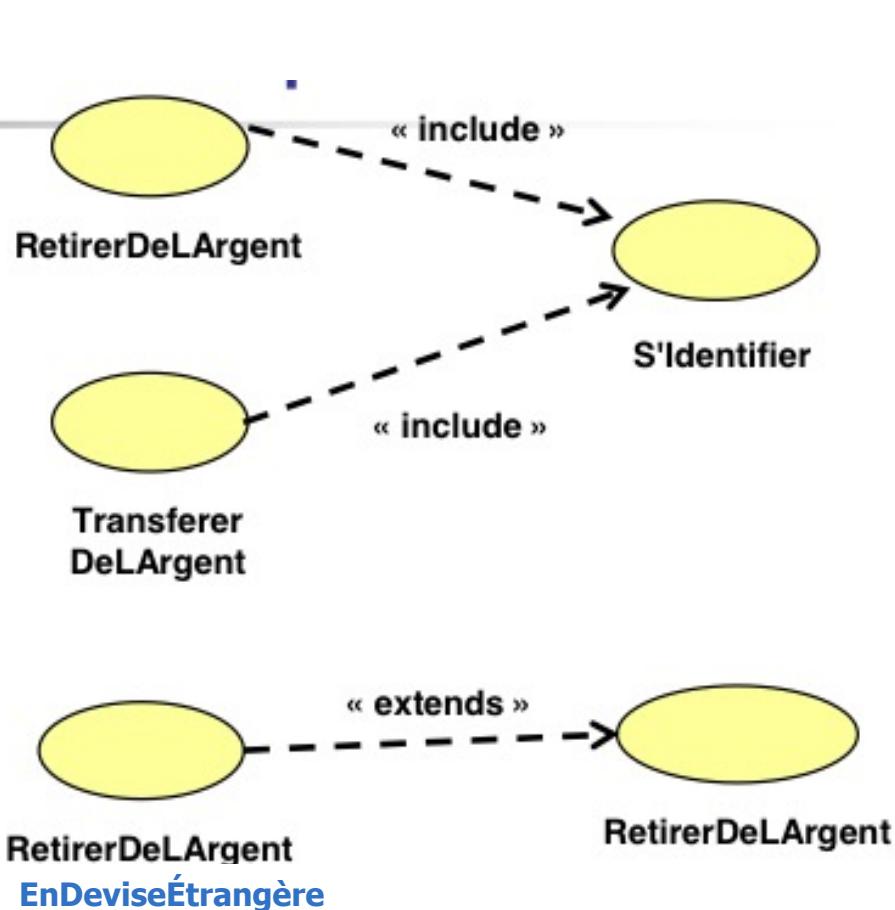
# Chaînage des conditions des CU

- Prévoir la possibilité de grouper des CU ensemble afin d'avoir une vue « macroscopique ».
  - ❖ Prenez garde à vos pré conditions et post conditions!

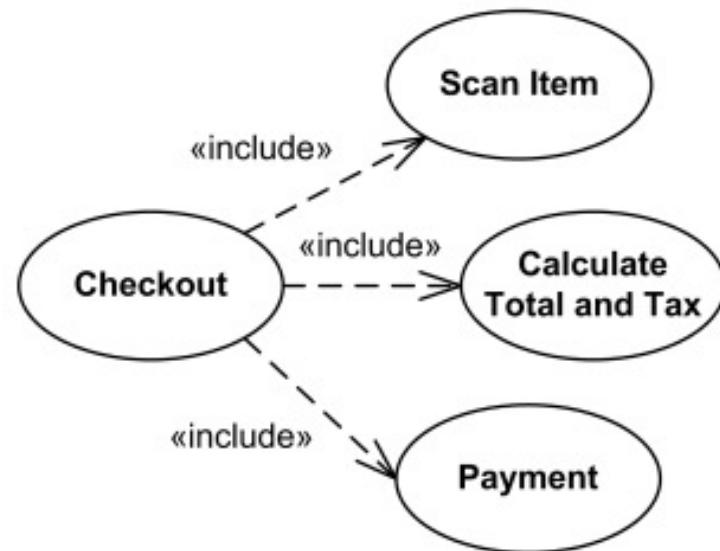


Extrait de « Software requirements » 3<sup>ième</sup> édition , K. Wiegers, page 156

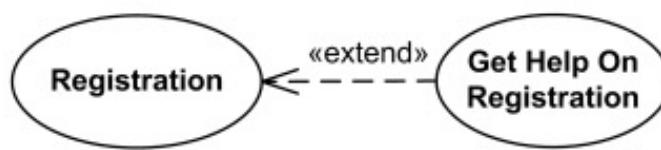
# Relations entre CU (inclusion, extension)



Référence: Page 46,  
Présentation « Le langage UML : Les cas d'utilisation »,  
Lydie du Bousquet, Lydie.du-bousquet@imag.fr



*Checkout use case includes several use cases - Scan Item, Calculate Total and Tax, and Payment*

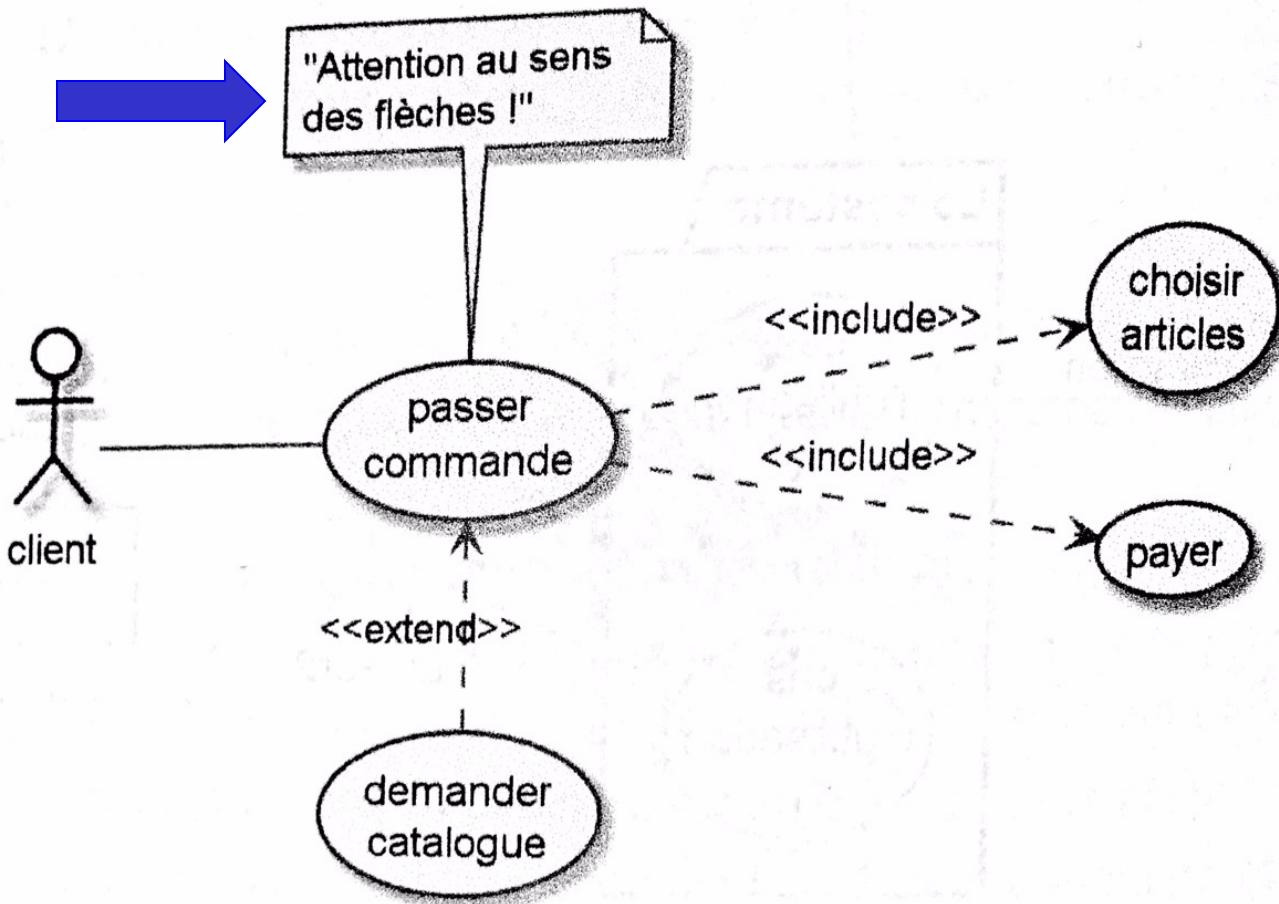


*Registration use case is complete and meaningful on its own.  
It could be extended with optional Get Help On Registration use case.*

<http://www.uml-diagrams.org>

# Autre exemple...

## Exemple



**FIGURE 11.3** Un exemple de relations d'inclusion et d'extension entre cas

Tiré du livre « *Analyse des besoins pour le développement logiciel* », Jacques Longchamp, Dunod, 2015

Département de génie logiciel et des TI

# CU – Mise en garde

- **Software requirements analysis: Five use case traps to avoid, by Karl E. Wiegers, Ph.D.**
  - ❖ Use cases have become a popular requirements development technique. Focusing on users and their goals, rather than on product features, improves the chances of developing a software package that truly meets customer needs. However, a mystique has grown up around use cases, and **many organizations struggle to use them successfully**. Here are several traps to avoid as your requirements analysts begin to apply the use case technique.

<http://searchsoftwarequality.techtarget.com/tip/Software-requirements-analysis-Five-use-case-traps-to-avoid?src=itke%20stub>

# CU – Mise en garde (suite)

## ○ Voici les failles à prévenir:

- ❖ Trap #1: Use cases that users don't understand
- ❖ Trap #2: Too many use cases
- ❖ Trap #3: Overly complex use cases
- ❖ Trap #4: Describing specific user interface elements and actions
- ❖ Trap #5: Not using other requirement models

<http://searchsoftwarequality.techtarget.com/tip/Software-requirements-analysis-Five-use-case-traps-to-avoid?src=itke%20stub>

# CU – Mise en garde (suite)

## ⦿ Trap #1: Use cases that users don't understand

- ❖ Use cases are a way to represent user requirements, which describe **what the user needs to be able to do with the product**. Use cases **should focus on tasks a user needs to accomplish with the help of the system**, so they should relate to the user's business processes. Your users should be able to read and review use cases to find possible problems, such as missing alternative flows or incorrectly handled exceptions.

<http://searchsoftwarequality.techtarget.com/tip/Software-requirements-analysis-Five-use-case-traps-to-avoid?src=itke%20stub>

# CU – Mise en garde (suite)

## ⦿ Trap #2: Too many use cases

❖ When analysts are generating scores or hundreds of use cases, something's wrong. This normally means that the use cases are written at too low an abstraction level. **Each use case should be general enough to cover several related scenarios on a common theme.** Some of these will be success scenarios, and others represent conditions in which the use case might not succeed -- exceptions. If you are caught in a use case explosion, try moving up the abstraction level to group together similar use cases, treating them as alternative flows of a single, more abstract use case.

<http://searchsoftwarequality.techtarget.com/tip/Software-requirements-analysis-Five-use-case-traps-to-avoid?src=itke%20stub>

# CU – Mise en garde (suite)

## ① Trap #3: Overly complex use cases

- ❖ The general guideline is that **the number of steps in the normal flow of a use case should not exceed approximately a dozen.** I once read a use case with nearly 50 steps in the normal flow. The problem was that the "normal flow" included many branching possibilities and error conditions that could arise, along with how to handle them. That is, the normal flow actually included alternative flows and exceptions. **A better strategy is to pick a simple, default, success path through the use case and call that the normal flow.** Then write separate alternative flows to cover variations on this and exception flows to describe the error conditions. This gives you a use case with multiple small packages of information, which is much easier to understand and manage than a huge use case that tries to handle every possibility in a single flow description.

<http://searchsoftwarequality.techtarget.com/tip/Software-requirements-analysis-Five-use-case-traps-to-avoid?src=itke%20stub>

# CU – Mise en garde (suite)

## ① Trap #4: Describing specific user interface elements and actions

- ❖ Write "essential" use cases that describe the interactions between the user and the system at an abstract level, **without incorporating user interface specifics**. The use case description should not include a screen design, although simple user interface prototypes can be valuable to facilitate the use case exploration. I don't even like to hear terminology in the use case that alludes to specific user interface controls. Saying "User clicks on OK" implies a GUI interface using a mouse and buttons. But what if it would make more sense to use a touch screen or speech recognition interface? Imposing premature design constraints in the use case can lead to a suboptimal design, unless you're adding a new capability to an existing application where the screens already exist.

<http://searchsoftwarequality.techtarget.com/tip/Software-requirements-analysis-Five-use-case-traps-to-avoid?src=itke%20stub>

# CU – Mise en garde (suite)

## ⦿ Trap #5: Not using other requirement models

- ❖ Analysts who begin employing use cases sometimes seem to forget everything else they know about requirements specification. Use cases are a great aid for exploring requirements for interactive systems, kiosks and Web sites. However, **they do not work as well for event-driven real-time systems, data warehouses or batch processes.**
- ❖ **Avoid the temptation to force fit all of your functional requirements into use cases.** Supplement the use case descriptions with a detailed list of functional requirements, non-functional requirements, graphical analysis models, prototypes, a data dictionary and other representations of requirements information. Use cases are valuable in many situations, but add them to your analyst toolkit instead of replacing your current tools with them.

<http://searchsoftwarequality.techtarget.com/tip/Software-requirements-analysis-Five-use-case-traps-to-avoid?src=itke%20stub>

# + Use Case Pitfalls!

Disponible sur Moodle!

## Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases

Susan Lilly  
SRA International, Inc.  
4300 Fair Lakes Ct.  
Fairfax, VA 22033  
[susan\\_lilly@sra.com](mailto:susan_lilly@sra.com)  
703-227-5103

### Abstract

*One of the beauties of use cases is their accessible, informal format. Use cases are easy to write, and the graphical notation is trivial. Because of their simplicity, use cases are not intimidating, even for teams that have little experience with formal requirements specification and management. However, the simplicity can be deceptive; writing good use cases takes some skill and practice. Many groups writing use cases for the first time run into similar kinds of problems. This paper presents the author's "Top Ten" list of use case pitfalls and problems, based on observations from a number of real projects. The paper outlines the symptoms of the problems, and recommends pragmatic cures for each. Examples are provided to illustrate the problems and their solutions.*

### Introduction

Over the past few years, we have seen a number of projects make their first attempts at developing use cases. These projects have used use cases in a number of ways: as the entire system requirements specification, as part of the system requirements, as an analysis technique to elicit user requirements that were subsequently specified in other forms (e.g., traditional "shall's"), and as software subsystem-level requirements. The project teams that developed the use cases have included developers and/or analysts; in some cases the project teams have included customers or end users as well.

Although the project teams had little trouble getting started with use cases, many of them encountered similar problems in applying them on a larger scale. These problems include undefined or inconsistent system boundary, use case model complexity, use case specification length and granularity, and use cases that are hard to understand or never complete. These have been grouped and summarized here as a "Top Ten" list of use case pitfalls and problems, which may be encountered by inexperienced practitioners.

A sample problem is used to provide simple examples throughout this paper. *The Baseball Ticket Order System* is a computer system that is to be deployed to simplify customer sales for baseball games. Customers may view the season schedule and reserve tickets at kiosks placed in convenient locations, such as malls. Alternately, customers may call an 800 number and a phone clerk will reserve tickets for them. The customer may pay by credit card, or may pay at the time the tickets are picked up at the stadium on the day of the game.

# Plan du cours

- Clarifier les cas d'utilisation (CU)
- Exercice – Étiquettes intelligentes
- Précisions concernant les exigences
- Des CU aux cas de tests
- Gestion des changements
- Modèle CMMI-DEV



# Étiquette RFID (Radio-Frequency ID)

- Connaissez-vous cette technologie?
- Quel est le principe de fonctionnement?

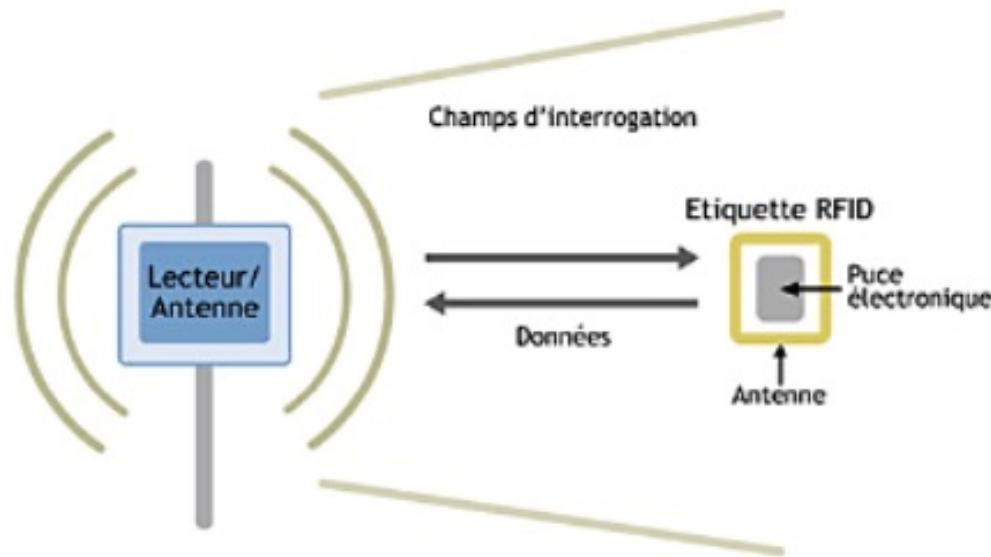


Figure 22 - Principe de fonctionnement d'une étiquette RFID

- Exemple d'application: Thermopatch

# Exercice – Étiquettes intelligentes



- Mise en application CU avec inclusion / extension (ref. cours #8)
- Mise en contexte:
  - Nous implantons un système d'étiquettes intelligentes dans une bibliothèque en remplacement de celles à code à barre.
  - Le préposé au comptoir de prêt devra pouvoir lire l'ensemble d'une pile de livres afin d'accélérer le service (lors du prêt ou le retour).
  - Les techniciens, à l'aide d'un lecteur mobile, devront pouvoir lire les différents livres d'un chariot afin qu'une route puisse être établie afin de replacer chacun des ouvrages au bon endroit!
  - Afin de retracer un livre, le technicien peut utiliser le nouveau cherche livre, grâce à son immense antenne, il permet de localiser un livre perdu dans la bibliothèque.
  - Des lecteurs mobiles seront aussi prêtés aux utilisateurs. Ceux-ci permettront de retracer facilement le livre recherché ainsi que de consulter une biographie de l'auteur.



# Exercice – Étiquettes intelligentes

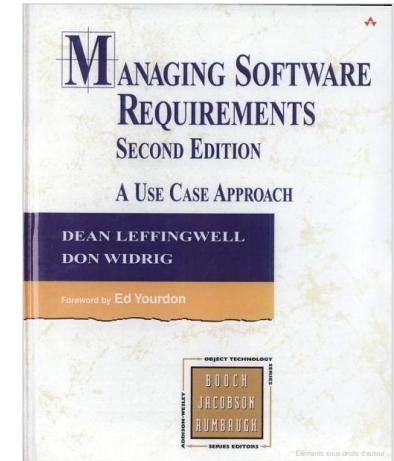
- Durée 30 minutes!
- En équipe de cinq vous devez:
  - ❖ Dessiner un diagramme des cas d'utilisation
    - Inclure tous les acteurs nécessaires!
    - Ajouter les cas d'utilisation avec inclusion et extension!



# Plan du cours

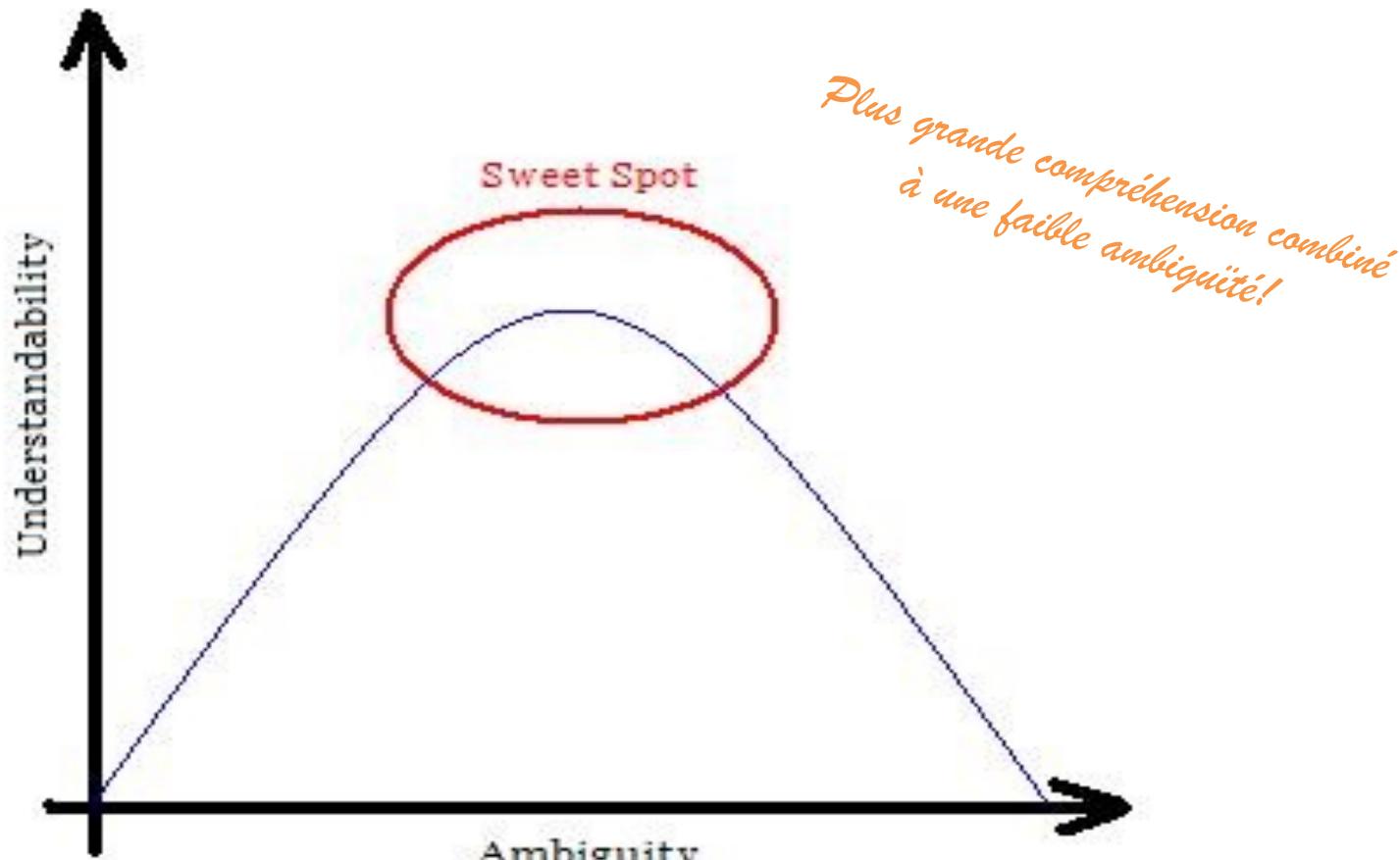
- Clarifier les cas d'utilisation (CU)
- Exercice – Étiquettes intelligentes
- Précisions concernant les exigences
- Des CU aux cas de tests
- Gestion des changements
- Modèle CMMI-DEV

Chapitre 23-25



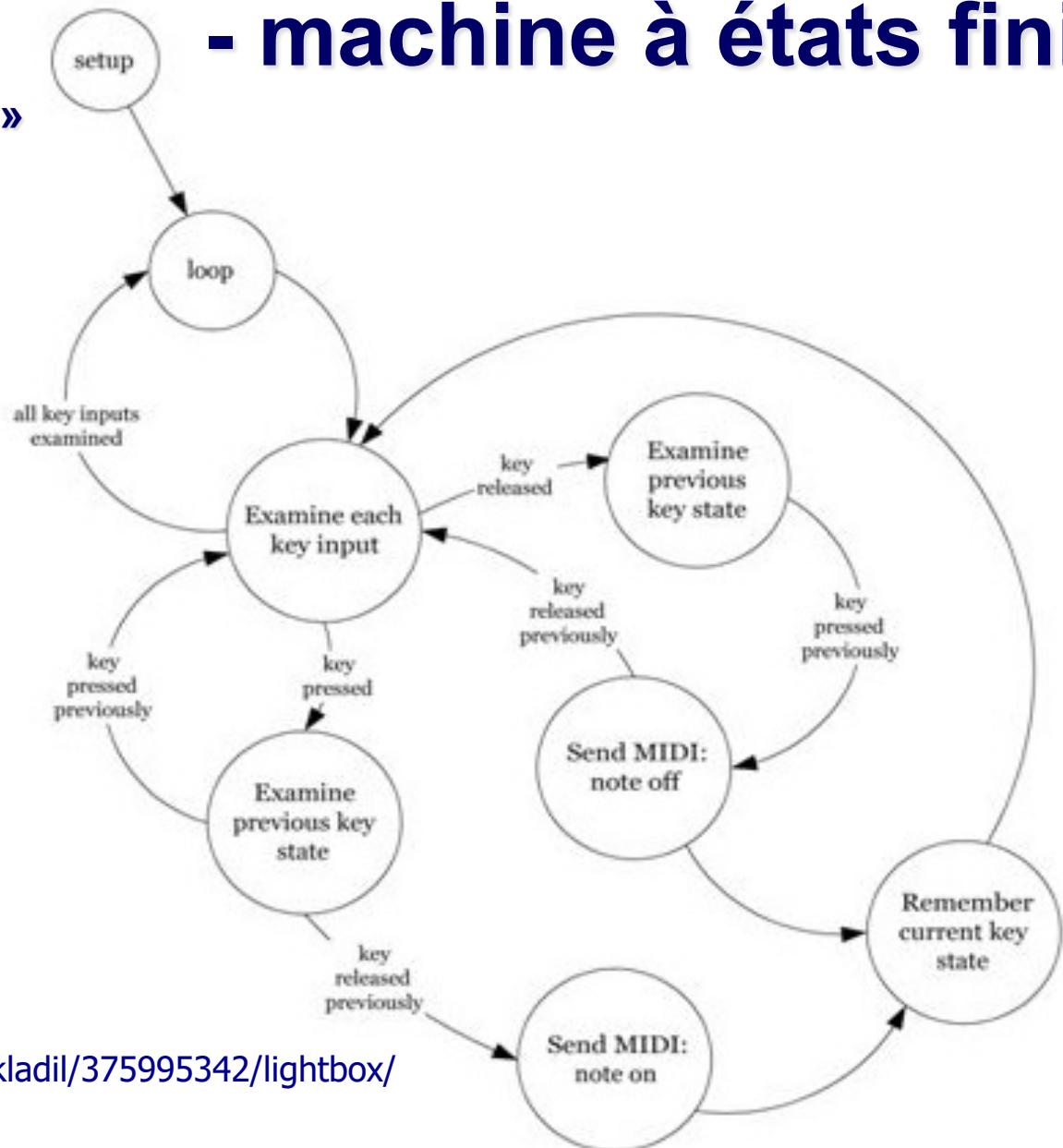
# Ambiguïté et spécificité

- Qu'est-ce que le « sweet spot »?



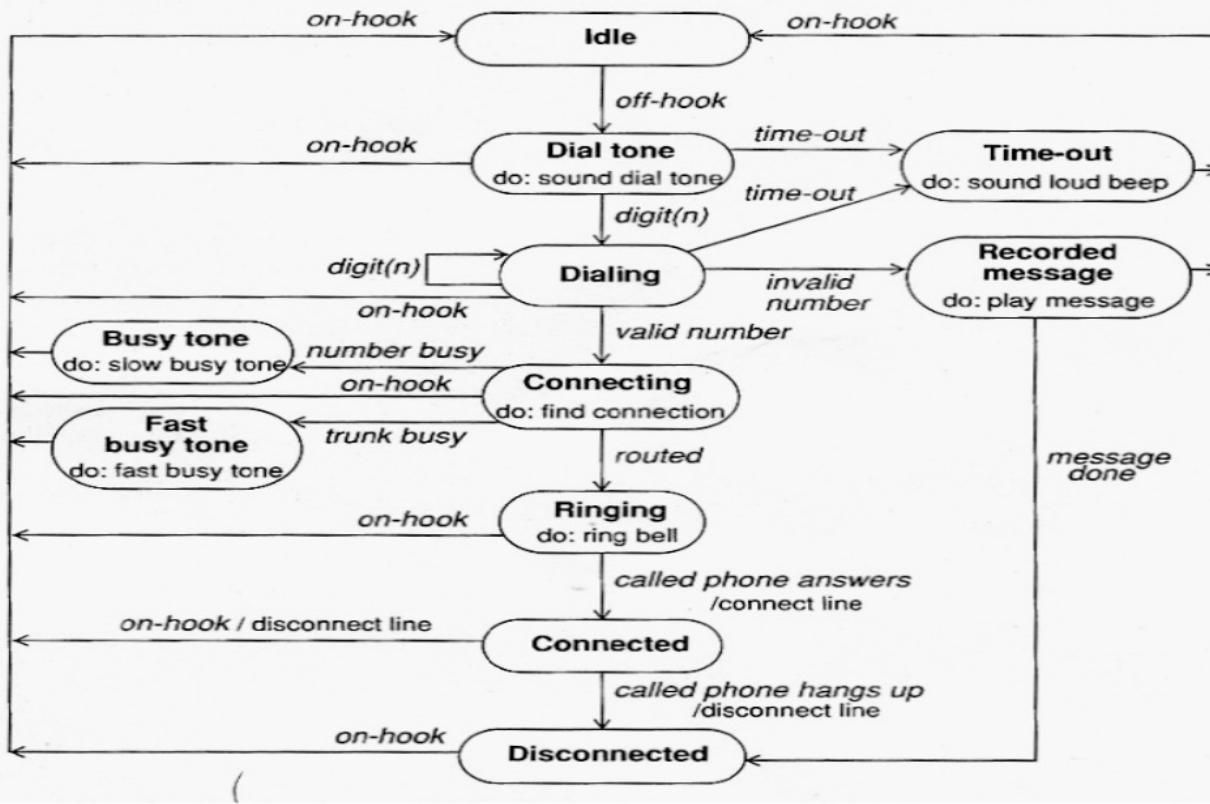
# Diagramme - machine à états finis

« Finite state machines »



<https://www.flickr.com/photos/emkladil/375995342/lightbox/>

# Finite State Machine for Phone



States: Idle, Dial tone, ....

Inputs: off-hook, on-hook, digit, ...

Outputs: sound dial tone, loud beep, play message,....

# Des CU à l'implantation

- Implantation directe de la conception au code
  - ❖ Merveilleux lorsque possible!
- Sinon, difficulté de déterminer comment et où sont implantées les exigences!
  - ❖ Ça devient un problème d'orthogonalité!

# Problème d'orthogonalité

- Dans certains cas, il y a peu de corrélation entre les exigences et la conception et l'implémentation.
- La forme de nos exigences et la forme de la conception et de l'implantation sont différentes!
- Il n'y a pas de façon facile de suivre ou de retracer de l'exigence à la conception et au code!

Requirement:  
“100,000 trades  
an hour”

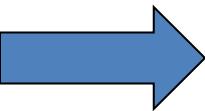


Code



# Plan du cours

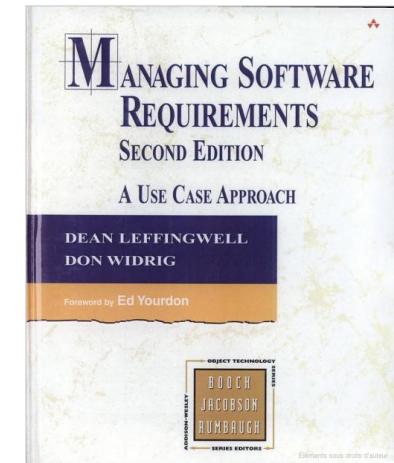
- Clarifier les cas d'utilisation (CU)
- Exercice – Machine à recycler
- Précisions concernant les exigences



Des CU aux cas de tests

- Gestion des changements
- Modèle CMMI-DEV

Chapitre 26



# Vérification vs validation

- Verification is an attempt to ensure that the product is built correctly, in the sense that the output products of an activity meet the specifications imposed on them in previous activities.
- Validation is an attempt to ensure that the right product is built, that is, the product fulfills its specific intended purpose.

Guide SWEBOK, Chapitre 11 Software Quality, Section 2.2

# Système vérifié vs validé

- Un **système vérifié** rencontre les exigences et les spécifications (point de vue des designers)
- Un **système validé** rencontre la définition des exigences (point de vue du client)

# Validation du système

- Revue de la définition des exigences
- Prototypage
- Test d'acceptation : fait par le client (ou supervisé par le client) afin de s'assurer que le système est conforme à ses exigences (ou à sa satisfaction)

# Vérification du système

- Revues techniques, inspections et audits
- Traçabilité
- Gestion de la configuration et des changements
- Tests

# Tests

- **Module, composant ou test unitaire** : vérifie les fonctions des composants dans un environnement contrôlé.
- **Test d'intégration** : vérifie si les composants fonctionnent ensemble dans un système intégré.
- **Test fonctionnel** : vérifie la conformité des fonctionnalités du système avec les exigences fonctionnelles spécifiées.

# Plus de tests

- ◉ **Test de performance** : vérifie les performances du système intégré afin de s'assurer qu'elles correspondent bien aux exigences (ex: temps réponse). Doit être fait dans le même environnement du client
- ◉ **Test d'installation** : le système est déployé dans l'environnement du client et nous devons nous assurer que tout fonctionne bien.

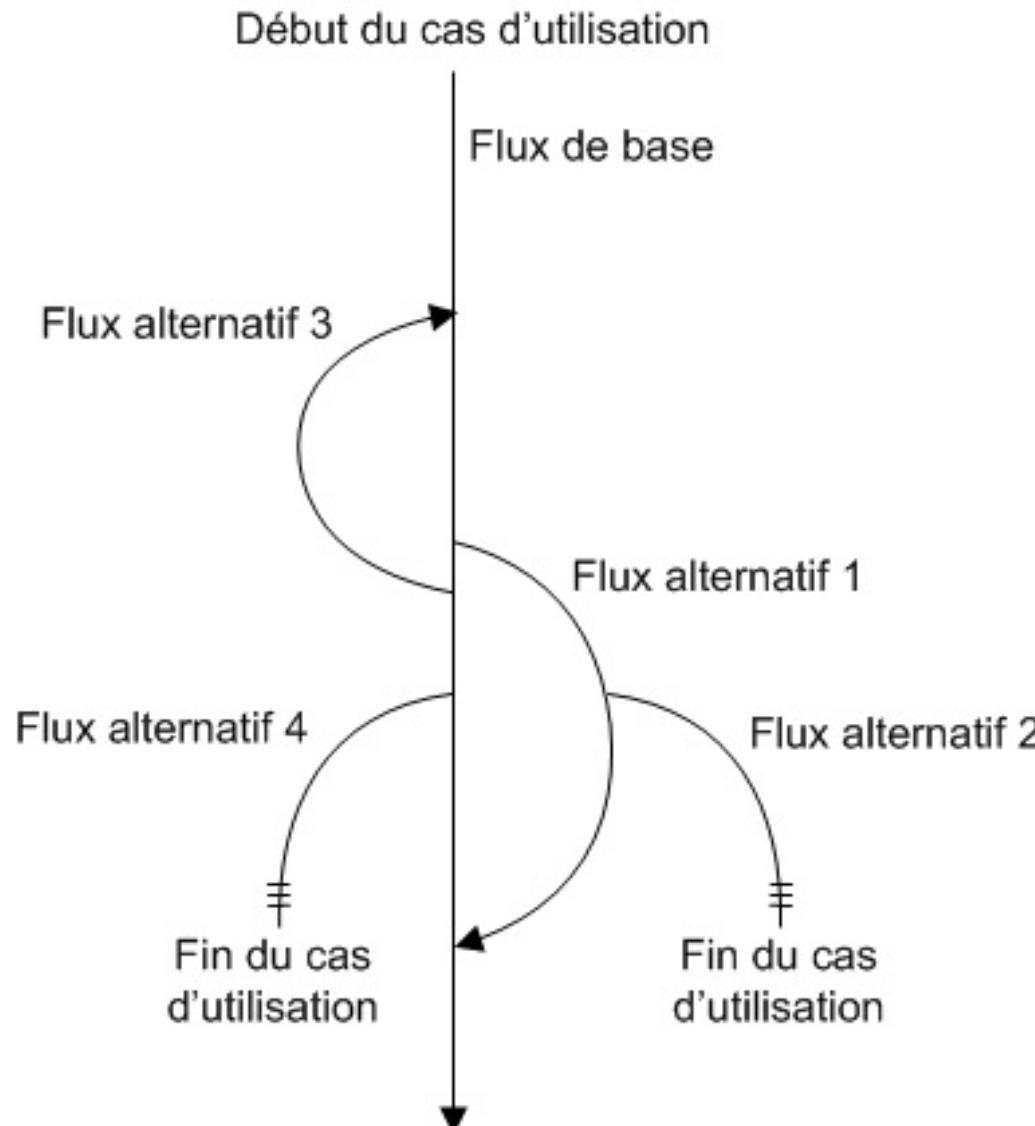
# Des cas d'utilisation aux cas de test

1. Identifier les scénarios du cas d'utilisation
2. Pour chaque scénario, identifier un ou des cas de test
3. Pour chaque cas de test, identifier les conditions qui vont causer son exécution
4. Compléter le cas de test en ajoutant des valeurs aux données

# Étape 1: Qu'est-ce qu'un scénario?

- Un scénario, ou une instance d'un cas d'utilisation, est une exécution d'un cas d'utilisation dans laquelle un utilisateur particulier exécute le cas d'utilisation de manière spécifique.

# Étape 1: Diagramme de flux du cas d'utilisation



# Étape 1: Matrice des scénarios

- Voir le tableau 26-1 à la page 311 du manuel du cours

Scenario Number	Originating Flow	Alternate Flow	Next Alternate	Next Alternate
1	Basic flow			
2	Basic flow	Alternate flow 1		
3	Basic flow	Alternate flow 1	Alternate flow 2	
4	Basic flow	Alternate flow 3		
5	Basic flow	Alternate flow 3	Alternate flow 1	
6	Basic flow	Alternate flow 3	Alternate flow 1	Alternate flow 2
7	Basic flow	Alternate flow 4		
8	Basic flow	Alternate flow 3	Alternate flow 4	

Diagram illustrating the flowchart for Figure 26-2:

```
graph TD; Start((Start use case)) -- Basic flow --> A(( )); A --> B(( )); B --> C(( )); C --> D(( )); D --> E(( )); E --> F(( )); F --> G(( )); G --> H(( )); H --> I(( )); I --> J(( )); J --> K(( )); K --> L(( )); L --> M(( )); M --> N(( )); N --> O(( )); O --> P(( )); P --> Q(( )); Q --> R(( )); R --> S(( )); S --> T(( )); T --> U(( )); U --> V(( )); V --> W(( )); W --> X(( )); X --> Y(( )); Y --> Z(( )); Z --> End1((End use case)); Z --> End2((End use case)); Z --> End3((End use case)); Z --> End4((End use case));
```

The diagram shows a vertical flow from top to bottom, starting at "Start use case" and ending at four "End use case" points. The "Basic flow" is a straight vertical line. Four "Alternate flows" branch off from the basic flow: "Alternate flow 1" branches to the second node; "Alternate flow 2" branches to the fourth node; "Alternate flow 3" branches to the eighth node; and "Alternate flow 4" branches to the fifth node. From each alternate flow node, the flow continues vertically down to the final "End use case" point.

# Étape 2: Qu'est-ce qu'un cas de test?

- Un cas de test est un ensemble d'entrées, de conditions d'exécution et de résultats attendus développés pour un objectif particulier dans le but de vérifier un chemin spécifique du logiciel ou pour vérifier la rencontre d'une exigence particulière.

# Étape 2: Identifier les cas de test

- Pour chaque scénario
  - ❖ Ajouter une ligne dans la matrice des cas de test
  - ❖ Identifier les données concernées par l'exécution spécifique du scénario
    - Données entrées par l'utilisateur
    - Données fournies par le système
  - ❖ Identifier le résultat attendu
  - ❖ Matrice des cas de test
    - Voir le tableau 26-2 et 26-3 de la page 312 et 313 du manuel du cours

# Étape 3: Identifier les conditions de test

- Un cas de test peut seulement être exécuté selon un ensemble de conditions uniques qui ne se retrouvent pas dans un autre cas de test.
- Pour les trouver, il faut se baser sur ce qui déclenche les flux alternatifs du cas d'utilisation.
- Dans un cas de test, une condition peut être:
  - ❖ Valide ou vraie (V)
  - ❖ Invalide (I) ou fausse (F)
  - ❖ Non applicable (N/A)
- Matrice des cas de test avec l'identification des conditions
  - ❖ Voir le tableau 26-5 à la page 315 du manuel du cours

# Étape 4: Ajouter des valeurs aux cas de test

- Des valeurs réelles doivent être ajoutées dans le but de pouvoir exécuter un cas de test.
- Il faut tester les valeurs à l'intérieur des plages acceptables, les minimums et les maximums, mais aussi des données invalides.
- Matrice des cas de test avec la valeur des données
  - ❖ Voir le tableau 26-6 à la page 316 du manuel du cours

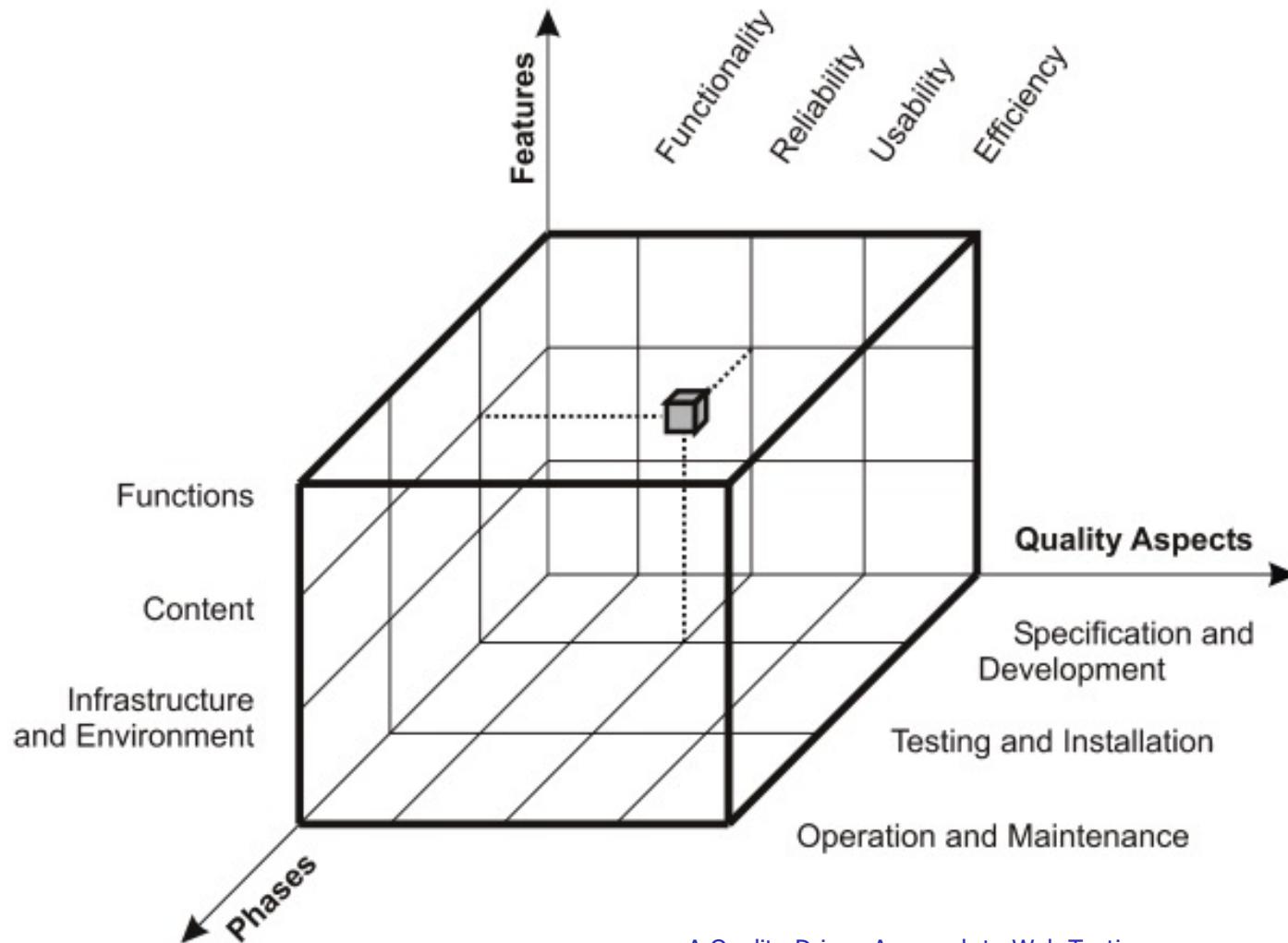
# Un des tableaux du chapitre #26!

Table 26–6 Control Light Test Cases with Data Values

Test Case ID	Scenario	Description	Condition: Button Pressed < Timer Period	Condition: Button Pressed > Timer Period	Condition: Button Released After Being Held	Condition	Expected Result
1	1	Basic flow: Resident releases button before timer period ends	< 1 sec. in .1-sec. intervals	I	N/A	Light on	Light goes off
2	1	Basic flow: Resident releases button before timer period ends	< 1 sec. in .1-sec. intervals	I	N/A	Light off	Light goes on
3	2	Alternate flow: Resident continuously presses button for longer than timer period	I	1–60 sec.	N/A	N/A	Light level goes up and down continuou
4	3	Resident releases switch after continuously pressing button	I	I	V	N/A	Light stays at last brightness

Images extraites du chapitre 26 du livre « Managing software requirements », 2<sup>nd</sup> edition, D. Leffingwell

# Schéma des essais d'un site Web



A Quality-Driven Approach to Web Testing  
Rudolf Ramler, Edgar Weippl, Mario Winterer, Wieland Schwinger,  
and Josef Altmann  
<http://www.scch.at>

# Plan du cours

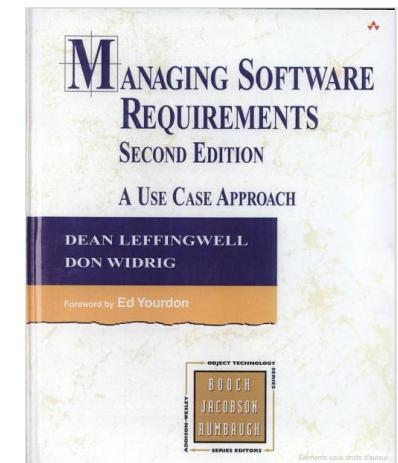
- ◉ Clarifier les cas d'utilisation (CU)
- ◉ Exercice – Machine à recycler
- ◉ Précisions concernant les exigences
- ◉ Des CU aux cas de tests



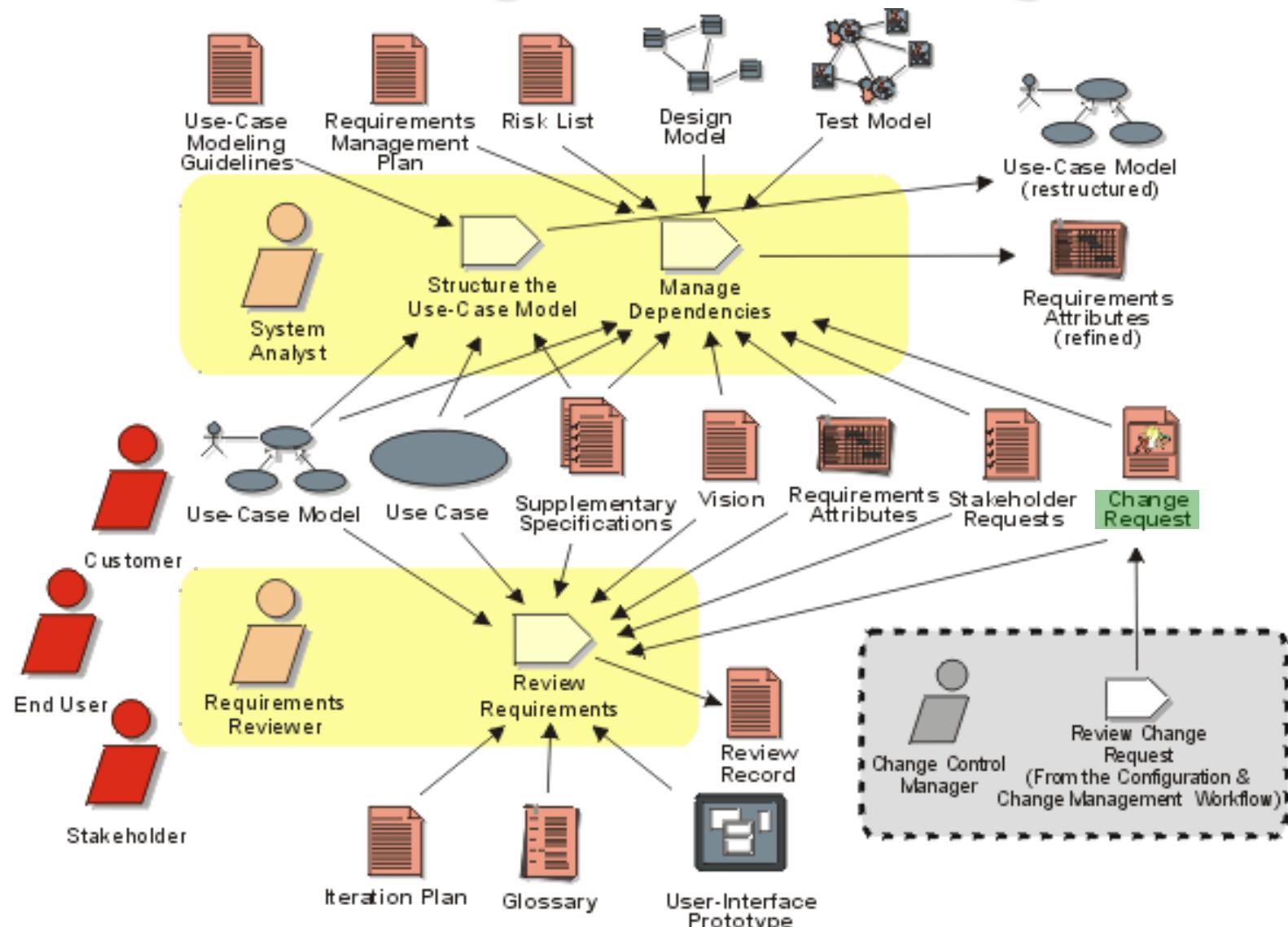
Gestion des changements

- ◉ Modèle CMMI-DEV

Chapitre 28



# Gérer les changements d'exigences



# Facteurs externes du changement

## Facteurs externes dont nous avons peu ou pas le contrôle

Un changement survient au problème que l'on tente de solutionner avec le nouveau système.

L'utilisateur change d'idées ou la perception de ce que le système doit faire.

Environnement externe a changé, créant ainsi de nouvelles contraintes et/ou de nouvelles opportunités.

Un insidieux facteur externe est la venu du système lui-même .

# Facteurs internes du changement

## Facteurs internes

Nous avons oublié de demander aux bonnes personnes, les bonnes questions ou bon moment durant l'effort de cueillette des exigences.

Nous avons faillis à créer un processus pratique afin d'aider à gérer les changements aux exigences.

L'itération des exigences à la conception engendre de nouvelles exigences.

# Fuite des exigences

## Exemples

Requêtes des clients transmises directement aux développeurs.

Erreurs qui ont été effectuées, livrées et qui doivent être supportées.

Caractéristiques du matériels qui n'a pas été incluse ou qui ne fonctionne pas.

Œufs de Pâques livrés par les développeurs.

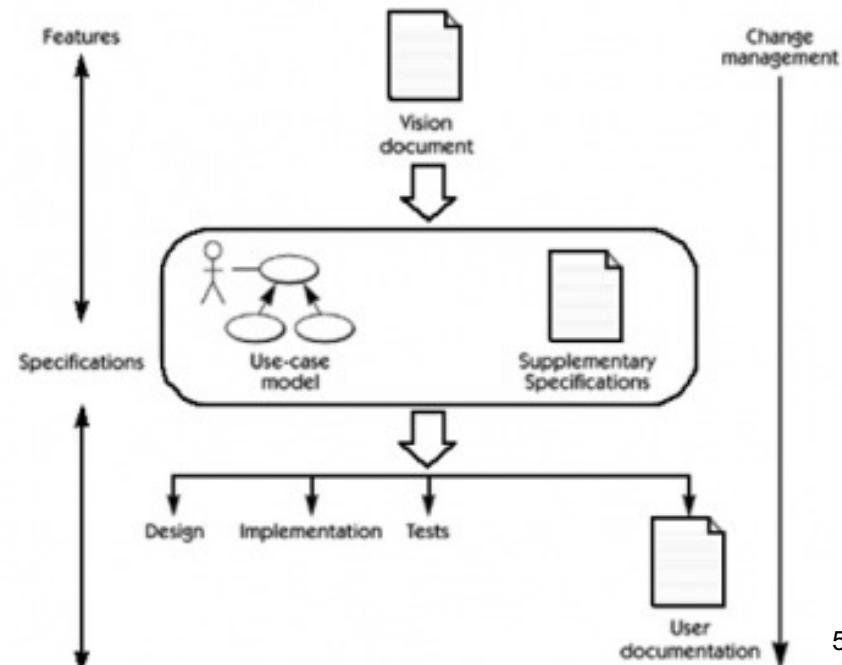
...

*Représente jusqu'à 50% de  
l'effort de la portée totale du projet!*

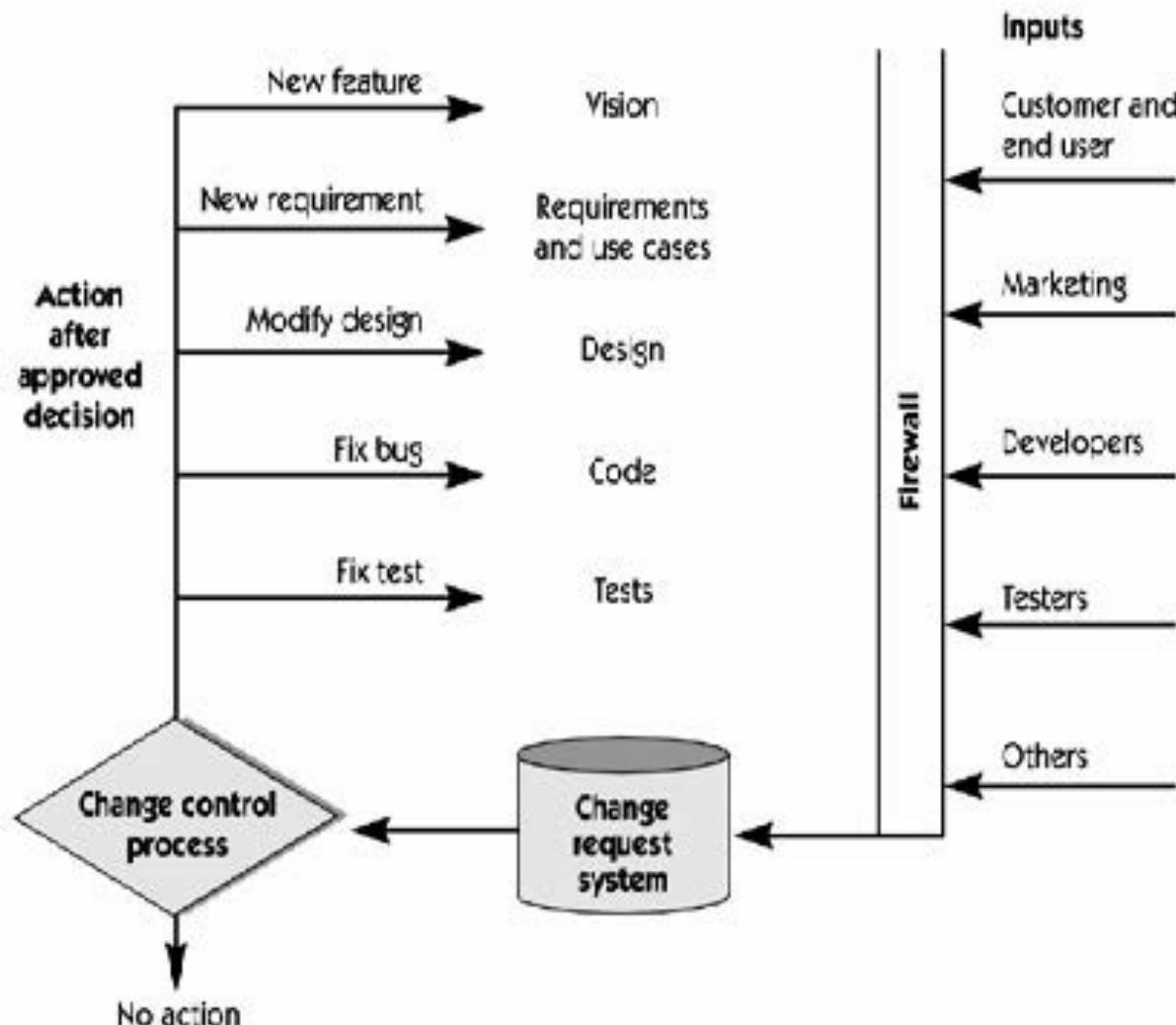
# Processus afin de gérer les changements

Afin de gérer efficacement les changements, le processus inclut les étapes suivantes:

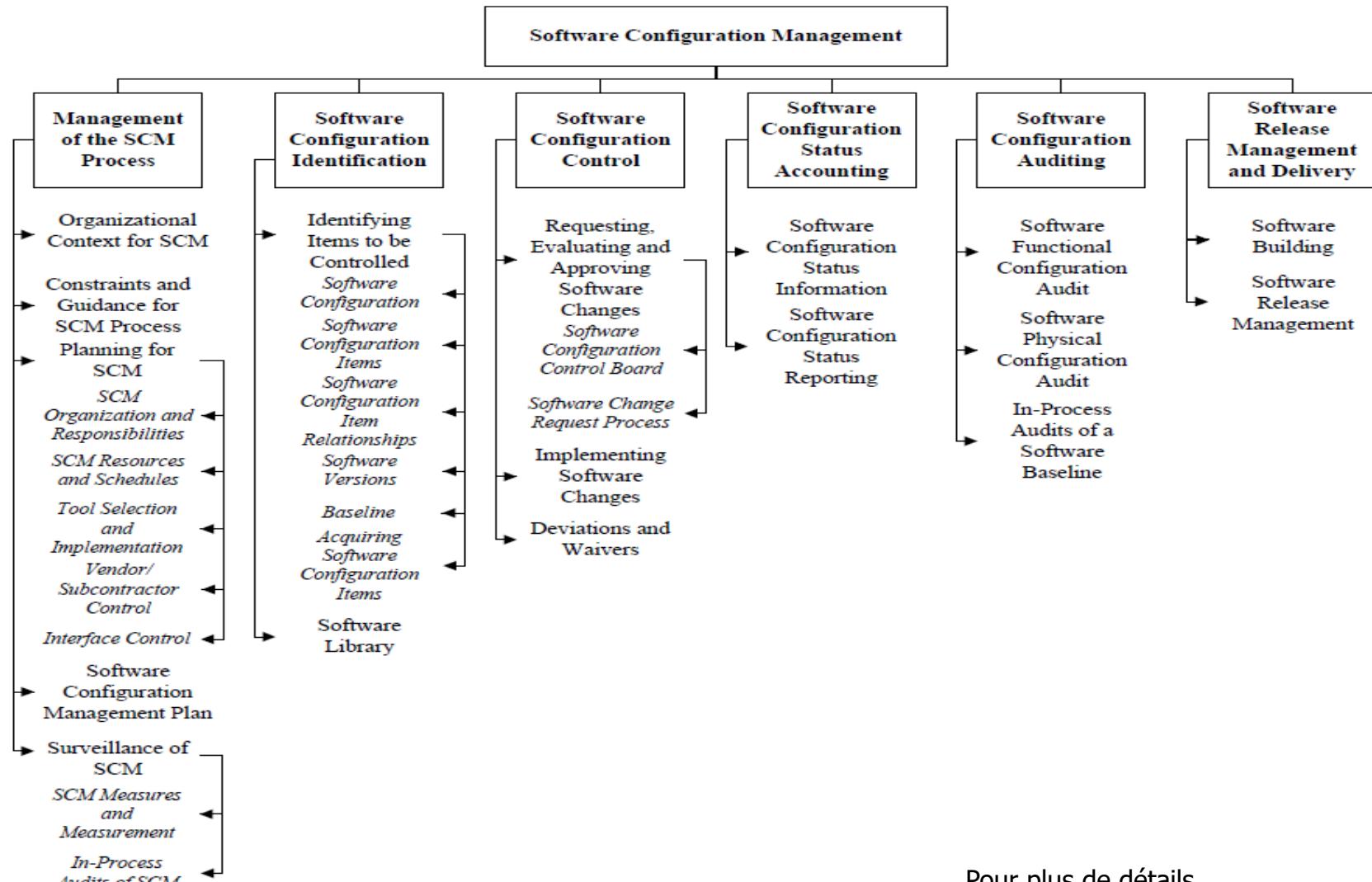
1. Reconnaître que les changements sont inévitables, et les prévoir!
2. Établir la base de référence des exigences.
3. Établir un canal unique pour contrôler le changement.
4. Utiliser un système de contrôle afin de capturer les changements.
5. Gérer le changement hiérarchiquement.



# Change Request Flow



# Exigences logicielles et gestion de la configuration



Pour plus de détails,  
référez-vous au Guide  
SWEBOk, Version 2004

# Exigences logicielles et gestion de la configuration

## 2.1.2. Software configuration item

Software items with potential to become SCIs include plans, **specifications and design documentation**, testing materials, software tools, source and executable code, code libraries, data and data dictionaries, and documentation for installation, maintenance, operations, and software use.

Pour plus de détails,  
référez-vous au Guide  
SWEBOK, Version 2004

# Exigences logicielles et gestion de la configuration

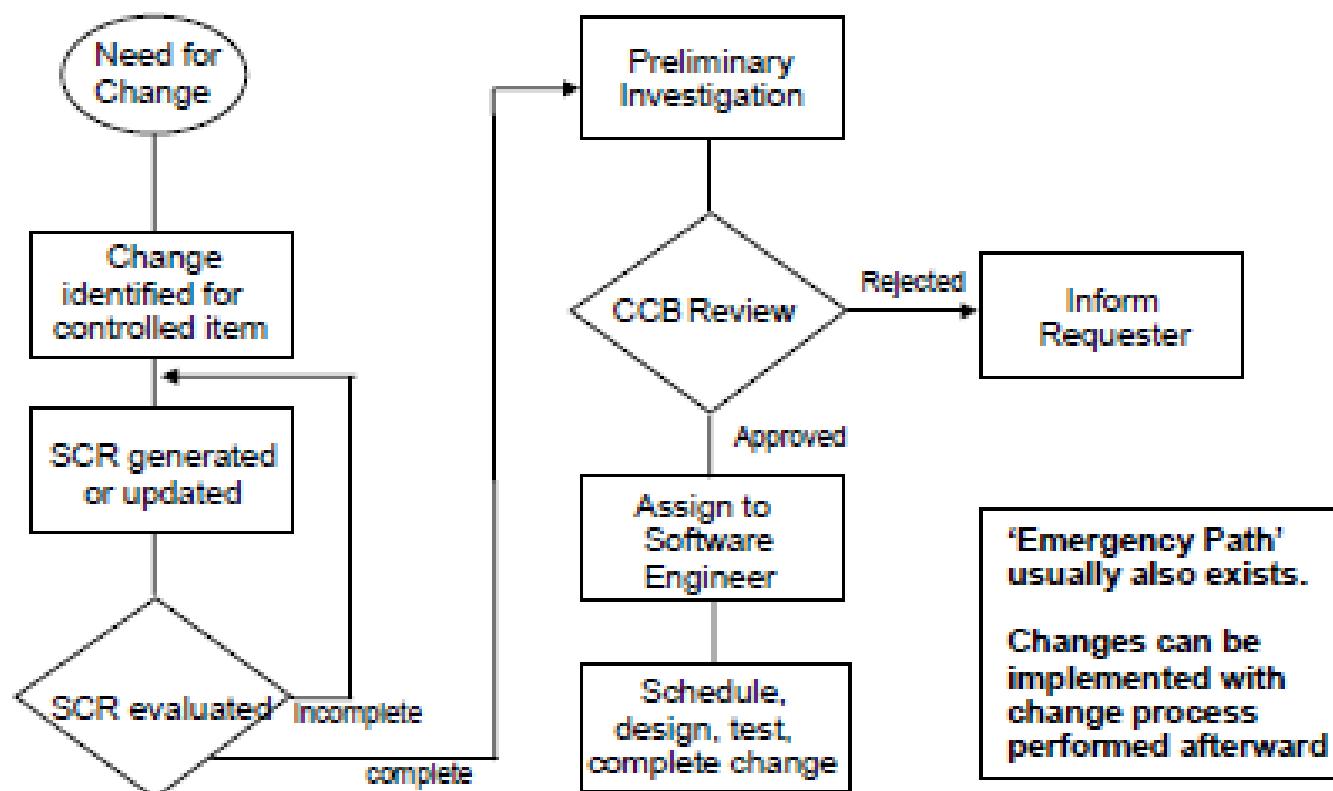
## 2.1.5. Baseline

Commonly used baselines are the functional, allocated, developmental, and product baselines (see, for example, [Ber92]).

**The functional baseline corresponds to the reviewed system requirements.** **The allocated baseline corresponds to the reviewed software requirements specification and software interface requirements specification.** **The developmental baseline represents the evolving software configuration at selected times during the software life cycle.** Change authority for this baseline typically rests primarily with the development organization, but may be shared with other organizations (for example, SCM or Test). The product baseline corresponds to the completed software product delivered for system integration.

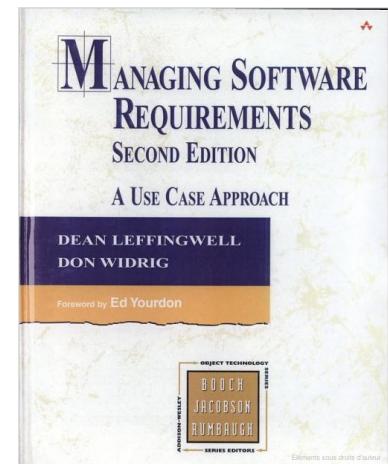
# Exigences logicielles et gestion de la configuration

## 3. Software Configuration Control



# Plan du cours

- Clarifier les cas d'utilisation (CU)
- Exercice – Machine à recycler
- Précisions concernant les exigences
- Des CU aux cas de tests
- Gestion des changements
- Modèle CMMI-DEV

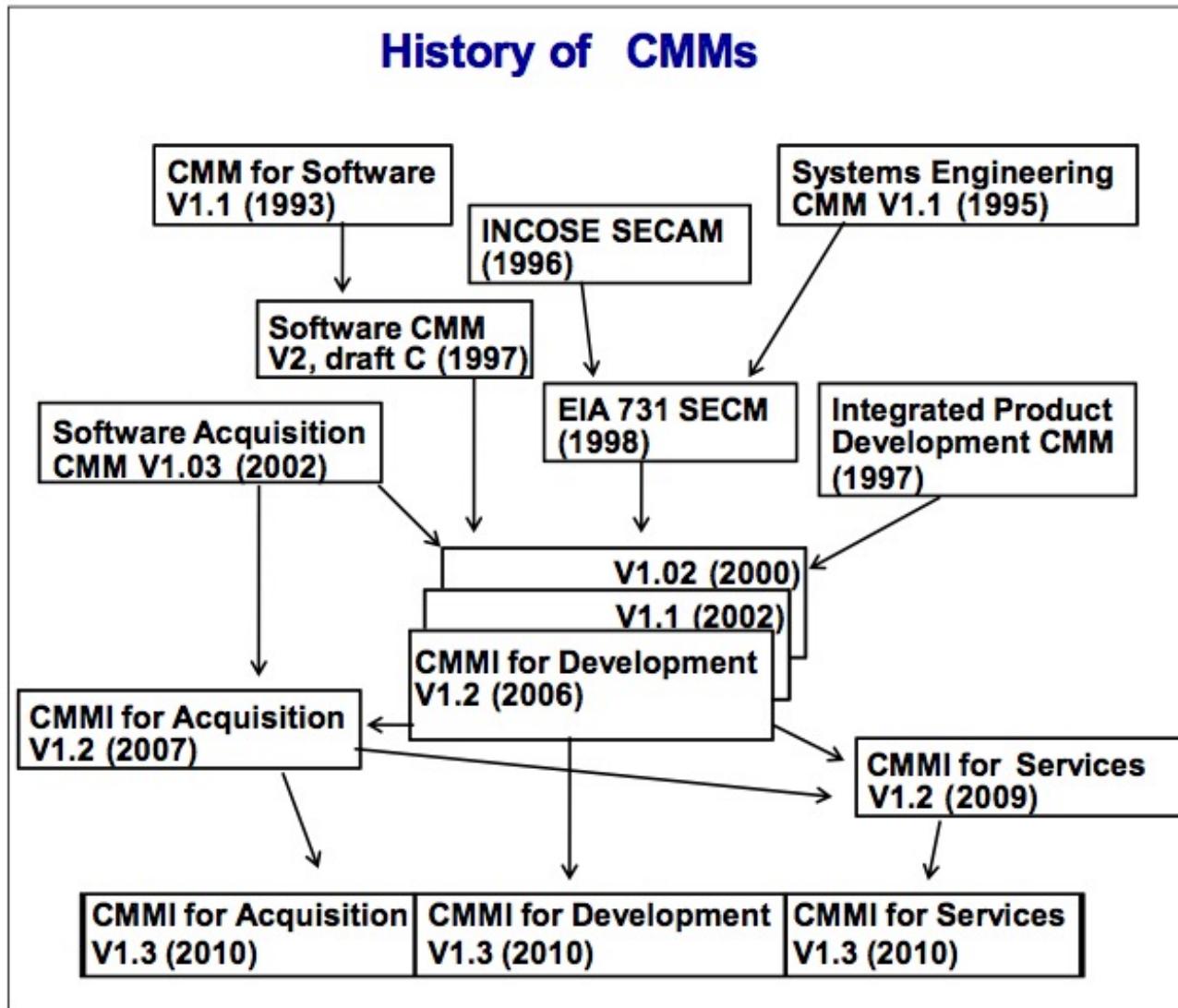


Annexe - F

# Qu'est-ce que CMMI-DEV?

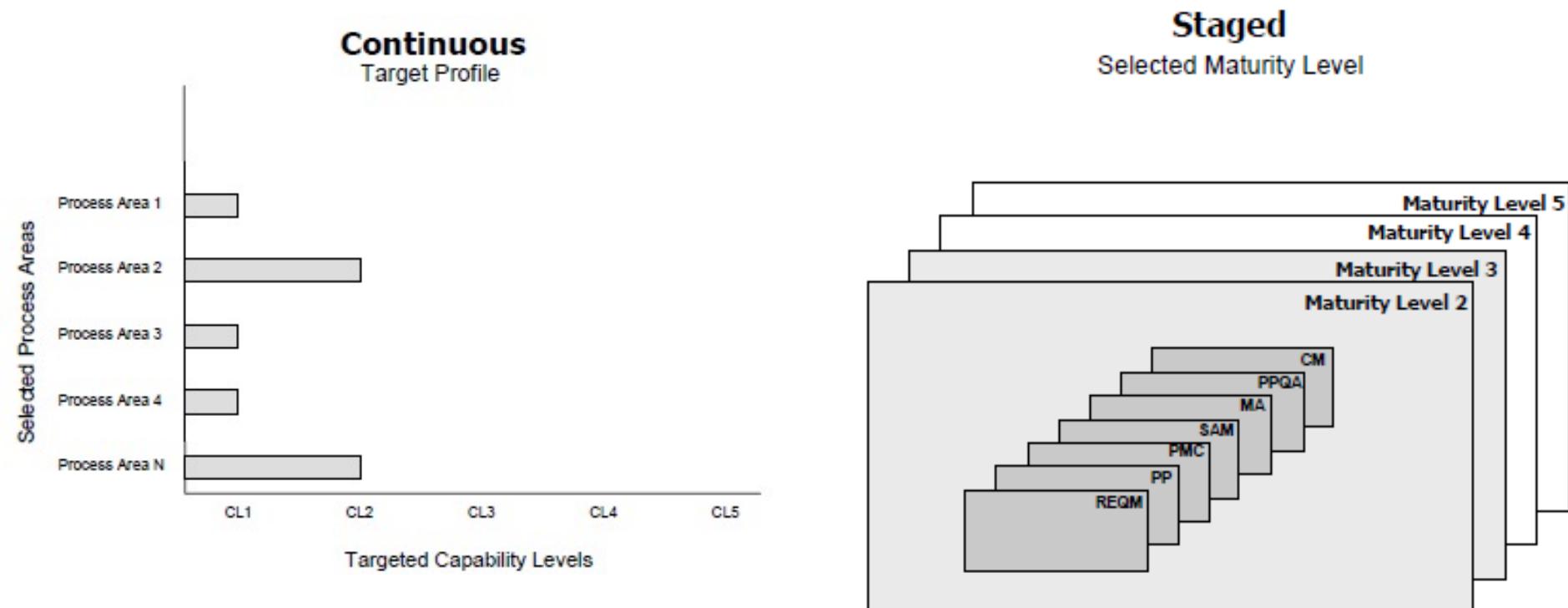
- Capability Maturity Model Integration for Development
- CMM versus CMMI
- Collection des meilleures pratiques qui aide les entreprises à améliorer leurs processus.
- Possède 22 « process area »

# Évolution de CMM



# Modèle de maturité - CMMI

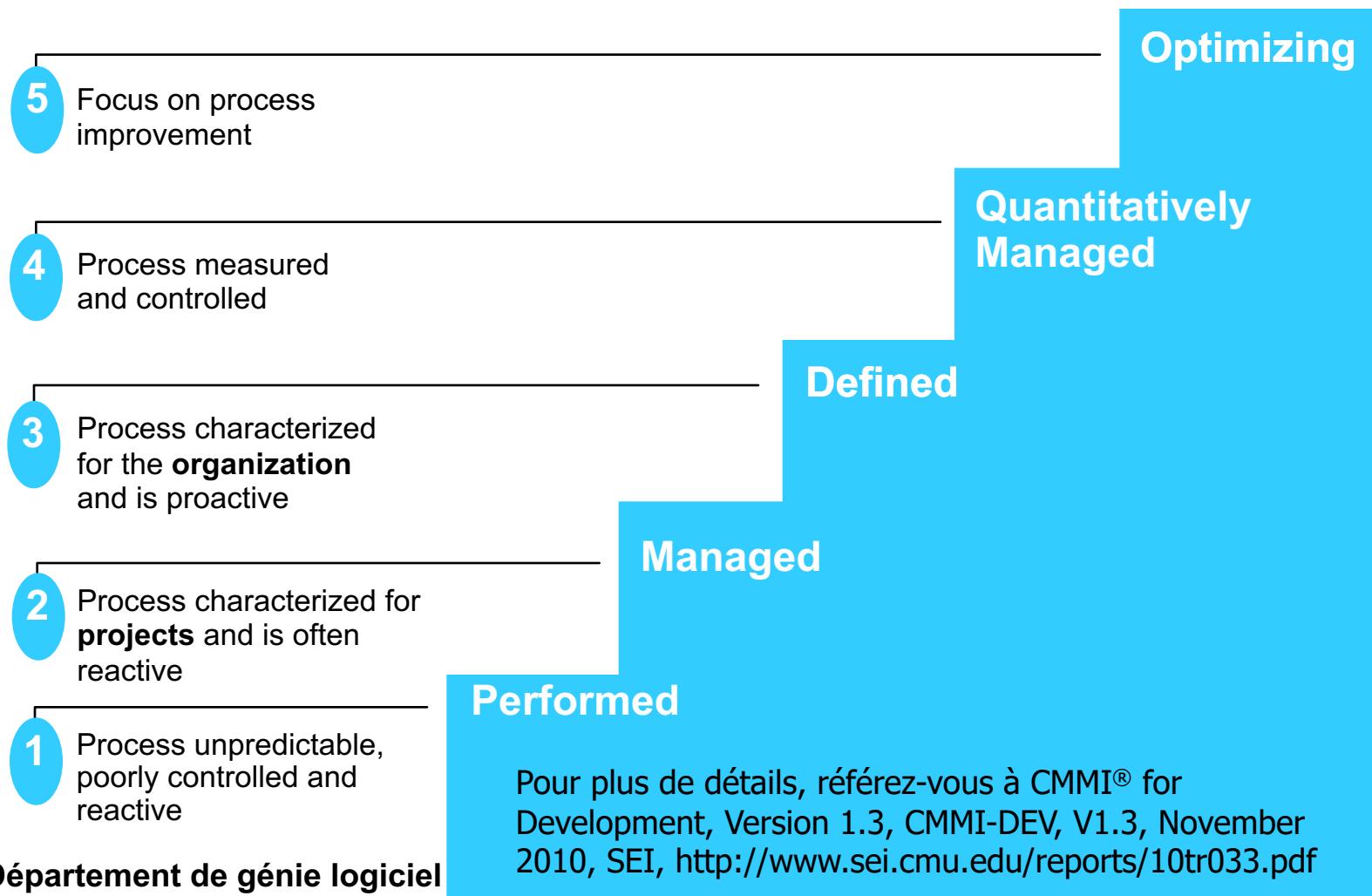
## Capability Maturity Model Integration (CMMI)



Pour plus de détails, référez-vous à CMMI® for Development, Version 1.3, CMMI-DEV, V1.3, November 2010, SEI,  
<http://www.sei.cmu.edu/reports/10tr033.pdf>

# Modèle de maturité – CMMI (suite)

## Staged



# Exigences logicielles et amélioration des processus

Name	Abbr	ML	CL1	CL2	CL3	CL4	CL5
Requirements Management	REQM	2					
Project Planning	PP	2					
Project Monitoring and Control	PMC	2					
Supplier Agreement Management	SAM	2					
Measurement and Analysis	MA	2					
Process and Product Quality Assurance	PPQA	2					
Configuration Management	CM	2					
Requirements Development	RD	3					
Technical Solution	TS	3					
Product Integration	PI	3					
Verification	VER	3					
Validation	VAL	3					
Organizational Process Focus	OPF	3					
Organizational Process Definition +IPPD	OPD +IPPD	3					
Organizational Training	OT	3					
Integrated Project Management +IPPD	IPM +IPPD	3					
Risk Management	RSKM	3					
Decision Analysis and Resolution	DAR	3					
Organizational Process Performance	OPP	4					
Quantitative Project Management	QPM	4					
Organizational Innovation and Deployment	OID	5					
Causal Analysis and Resolution	CAR	5					

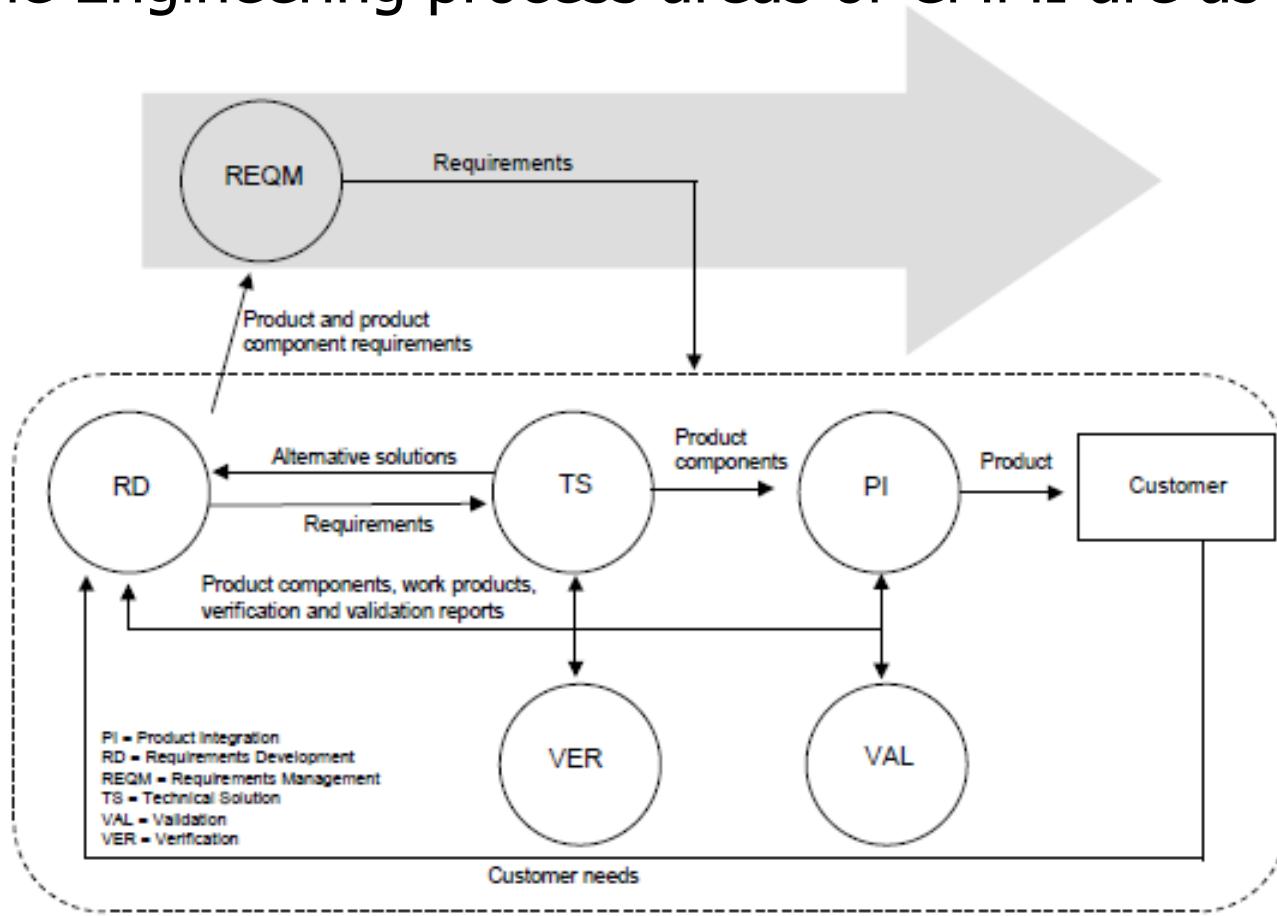
Pour plus de détails, référez-vous à CMMI® for Development, Version 1.3, CMMI-DEV, V1.3, November 2010, SEI, <http://www.sei.cmu.edu/reports/10tr033.pdf>

Département de génie

LOG410 Été 2021 — Alain Dion, ing. et Yves Durocher, ing., M.

# Le processus - CMMI

The Engineering process areas of CMMI are as follows:



Pour plus de détails, référez-vous à CMMI® for Development, Version 1.3, CMMI-DEV, V1.3, November 2010, SEI,  
<http://www.sei.cmu.edu/reports/10tr033.pdf>

# Exigences logicielles et amélioration des processus

## REQUIREMENTS MANAGEMENT

---

An Engineering Process Area at Maturity Level 2

### Purpose

---

The purpose of Requirements Management (REQM) is to manage the requirements of the project's products and product components and to identify inconsistencies between those requirements and the project's plans and work products.

Pour plus de détails, référez-vous à CMMI® for Development, Version 1.3, CMMI-DEV, V1.3, November 2010, SEI,  
<http://www.sei.cmu.edu/reports/10tr033.pdf>

# Exigences logicielles et amélioration des processus

## Specific goals and practices for REQM

Specific goal	Specific practice
SG 1 Manage Requirements	SP 1.1 Obtain an Understanding of Requirements
	SP 1.2 Obtain Commitment to Requirements
	SP 1.3 Manage Requirements Changes
	SP 1.4 Maintain Bidirectional Traceability of Requirements
	SP 1.5 Identify Inconsistencies Between Project Work and Requirements

Pour plus de détails, référez-vous à CMMI® for Development, Version 1.3, CMMI-DEV, V1.3, November 2010, SEI,  
<http://www.sei.cmu.edu/reports/10tr033.pdf>

# Exigences logicielles et amélioration des processus

## REQUIREMENTS DEVELOPMENT

---

An Engineering Process Area at Maturity Level 3

### Purpose

---

The purpose of Requirements Development (RD) is to produce and analyze customer, product, and product component requirements.

Plus de détails, référez-vous à CMMI® for Development, Version 1.3, CMMI-DEV, V1.3, November 2010, SEI,  
<http://www.sei.cmu.edu/reports/10tr033.pdf>

# Exigences logicielles et amélioration des processus

## Specific goals and practices for RD

Specific goal	Specific practice
SG 1 Develop Customer Requirements	SP 1.1 Elicit Needs
	SP 1.2 Develop the Customer Requirements
SG 2 Develop Product Requirements	SP 2.1 Establish Product and Product Component Requirements
	SP 2.2 Allocate Product Component Requirements
	SP 2.3 Identify Interface Requirements
SG 3 Analyze and Validate Requirements	SP 3.1 Establish Operational Concepts and Scenarios
	SP 3.2 Establish a Definition of Required Functionality
	SP 3.3 Analyze Requirements
	SP 3.4 Analyze Requirements to Achieve Balance
	SP 3.5 Validate Requirements

Pour plus de détails,  
référez-vous à CMMI® for  
Development, Version 1.3,  
CMMI-DEV, V1.3, November  
2010, SEI,  
<http://www.sei.cmu.edu/reports/10tr033.pdf>