

Cas d'utilisation : bonne & mauvaise utilisation !

février 06

Résumé

Les cas d'utilisation sont à la fois le concept le plus simple d'UML, le moins technique, mais aussi souvent le plus mal utilisé !

Cette présentation vise à faire une synthèse des mauvaises et des bonnes pratiques autour de la mise en œuvre des cas d'utilisation, en s'appuyant sur des exemples ainsi que sur des avis d'experts reconnus (Jacobson, Cockburn, etc.).

Table des matières

| | |
|--|-----------|
| 1. LES CAS D'UTILISATION DANS UML ET UP | 4 |
| 1.1 Les cas d'utilisation dans UML..... | 4 |
| 1.2 Les cas d'utilisation dans UP | 5 |
| 2. LES MAUVAISES PRATIQUES LES PLUS COURANTES | 5 |
| 2.1 Des cas d'utilisation trop petits !..... | 6 |
| 2.2 Sur-utilisation du diagramme de cas d'utilisation. | 6 |
| 2.3 La décomposition fonctionnelle avec la relation « include » | 7 |
| 2.4 Autres erreurs fréquentes | 8 |
| 2.4.1 Confondre cas d'utilisation et processus métier | 8 |
| 2.4.2 Identifier de « mauvais » acteurs..... | 8 |
| 3. LES RECOMMANDATIONS DES EXPERTS | 9 |
| 3.1 A tout seigneur, tout honneur : I. Jacobson | 9 |
| 3.2 Un UseCaseMaster : A. Cockburn | 9 |
| 3.3 C. Larman | 11 |
| 4. CONCLUSION | 12 |

Liste des Figures

| | | |
|-------------|---|----|
| Figure 1 : | Les bases du diagramme de cas d'utilisation d'UML 2.0..... | 4 |
| Figure 2 : | Les concepts avancés du diagramme de cas d'utilisation..... | 4 |
| Figure 3 : | Rôles et « artifacts » de la discipline Requirements | 5 |
| Figure 4 : | Le use case comme ensemble de scénarios | 6 |
| Figure 5 : | Exemple de sur-utilisation du diagramme de use cases..... | 7 |
| Figure 6 : | Le cadre ref comme alternative à la décomposition de use cases..... | 7 |
| Figure 7 : | Exemple de confusion avec le niveau « métier »..... | 8 |
| Figure 8 : | Distinction claire entre niveau « métier » et niveau « système »..... | 8 |
| Figure 9 : | Conseils généraux de description textuelle de use cases | 10 |
| Figure 10 : | Conseils de C. Larman pour tester la validité des use cases | 11 |
| Figure 11 : | Du diagramme de séquence système à la conception | 11 |

1. Les cas d'utilisation dans UML et UP

1.1 Les cas d'utilisation dans UML

Le diagramme de cas d'utilisation est l'un des treize types de diagrammes d'UML 2.0. Il montre les interactions fonctionnelles entre les acteurs et le système à l'étude.

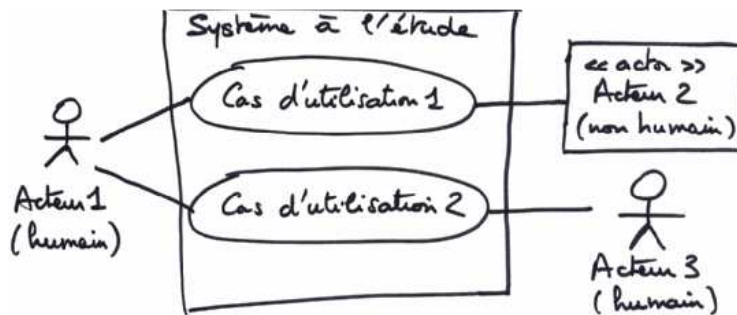


Figure 1 : Les bases du diagramme de cas d'utilisation d'UML 2.0

Précisons les définitions des concepts de cas d'utilisation (*use case*) et acteur (*actor*) :

- **Acteur** : rôle joué par un utilisateur humain ou un autre système qui interagit directement avec le système étudié. Un acteur participe à au moins un cas d'utilisation.
- **Cas d'utilisation** : ensemble de séquences d'actions réalisées par le système produisant un résultat observable intéressant pour un acteur particulier. Egalement : collection de scénarios reliés par un objectif utilisateur commun (un scénario est une séquence d'étapes décrivant une interaction entre un acteur et le système).

UML ajoute à ces concepts simples des relations possibles entre cas d'utilisation et acteurs, dont nous parlerons d'ailleurs dans la suite de l'article, parce qu'elles sont justement souvent l'occasion d'erreurs ou de mauvaises pratiques !

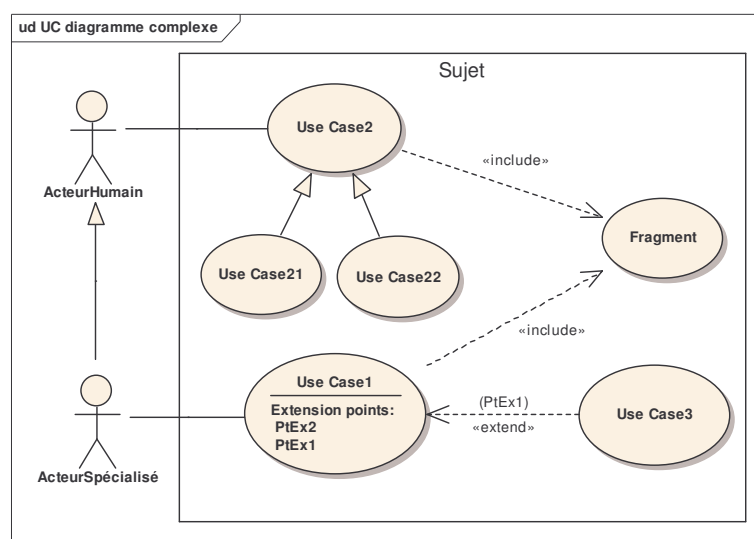


Figure 2 : Les concepts avancés du diagramme de cas d'utilisation

1.2 Les cas d'utilisation dans UP

Le Processus Unifié (en anglais *Unified Process* = *UP*) est sous-tendu par un ensemble de « bonnes pratiques », dont l'une des plus importantes est la gestion des exigences. En bref, il s'agit d'adopter une démarche itérative et incrémentale, sans pour autant tolérer le laisser-aller.

Dans ce cadre, UP place le modèle des cas d'utilisation comme élément central de la discipline *Requirements* (Gestion des exigences). Ce modèle ne contient pas seulement le ou les diagrammes de cas d'utilisation, mais surtout l'ensemble des cas d'utilisation rédigés et éventuellement complétés par des représentations dynamiques (diagrammes de séquence ou d'activité). La figure 3, inspirée du RUP™ de Rational, montre comment les *artifacts* de la discipline *Requirements* sont centrés autour du concept de *Use Case*.

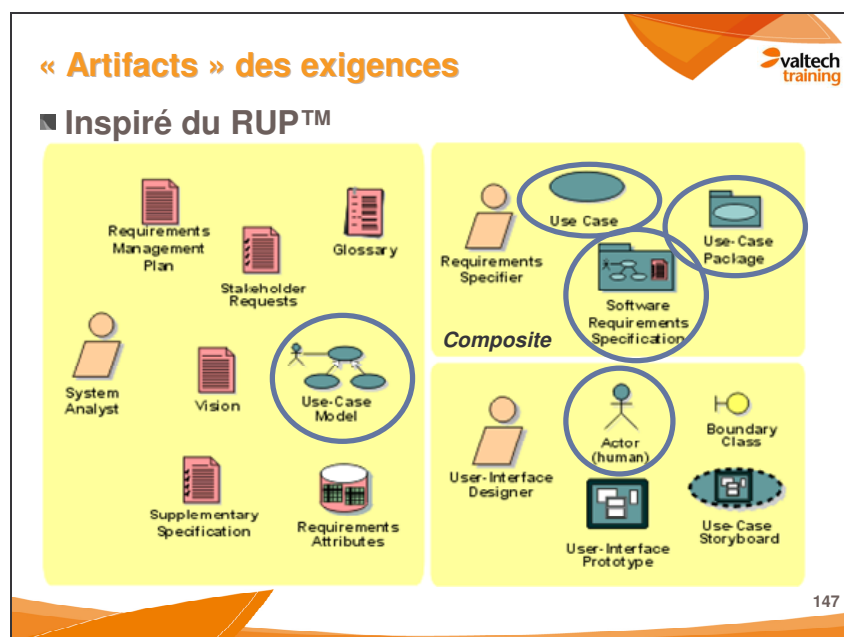


Figure 3 : Rôles et « artifacts » de la discipline Requirements

2. Les mauvaises pratiques les plus courantes

J'ai découvert les cas d'utilisation en 1994, dans un article de James Rumbaugh intitulé « *Getting started: Using use cases to capture requirements* » (JOOP 09/94). Fervent adepte de la méthode OMT depuis deux ans, je sentais bien qu'il manquait quelque chose pour dialoguer efficacement avec des utilisateurs et des clients non informaticiens. Personne n'envisageait sérieusement à l'époque d'utiliser directement les diagrammes de classes pour discuter d'un cahier des charges !

Pourtant, malgré l'apparente simplicité des concepts de cas d'utilisation et d'acteur, j'ai été souvent confronté dans ma pratique de consultant et de formateur à plusieurs écueils qui ont mis en danger de nombreux projets :

- La mauvaise appréciation de la taille d'un cas d'utilisation, amenant à une multiplication de « petits » cas d'utilisation ;
- L'importance exagérée accordée à la forme graphique du diagramme de cas d'utilisation, et aux relations d'utilisation et d'extension entre use cases ;
- L'utilisation erronée des cas d'utilisation comme substitut à une décomposition fonctionnelle.

Nous allons détailler les mauvaises pratiques les plus fréquentes dans les paragraphes suivants.

2.1 Des cas d'utilisation trop petits !

De nombreux auteurs reconnus, tels que Berard¹, Firesmith² et Fowler³, ont rapidement pointé l'existence de risques importants de mauvais emploi des cas d'utilisation. Ces risques sont principalement liés à la difficulté de savoir à quel niveau d'abstraction se placer, et quel système modéliser. Il est important de rappeler qu'un cas d'utilisation est une collection de scénarios, chaque scénario étant lui-même une séquence d'étapes : un cas d'utilisation n'est donc ni un seul scénario, ni encore moins une seule interaction acteur-système ! C'est dans cet esprit que la recommandation de Jacobson (pas plus de vingt cas d'utilisation !) prend tout son sens.

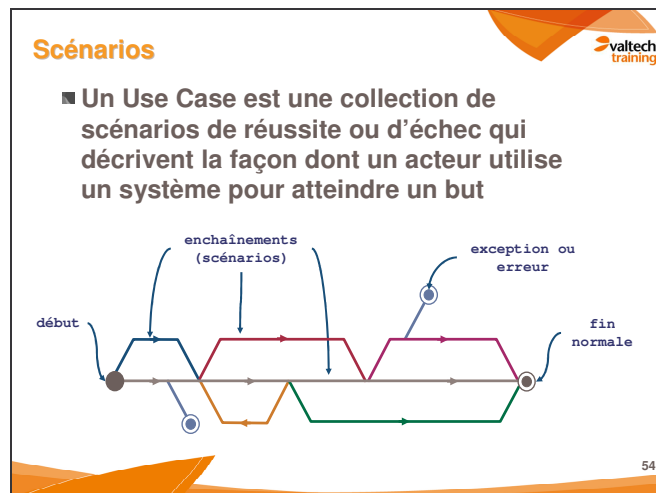


Figure 4 : Le use case comme ensemble de scénarios

2.2 Sur-utilisation du diagramme de cas d'utilisation.

L'autre travers, hélas fort répandu, consiste à penser qu'il suffit de dessiner un diagramme de *use cases* en ajoutant de préférence tout un tas de relations ésotériques entre les cas d'utilisation pour faire une bonne analyse des besoins...

Et là, les auteurs d'UML, quelle que soit l'admiration qu'on leur porte par ailleurs, en prennent une bonne part de responsabilité ! Ne trouve-t-on pas déjà dans l'article de J. Rumbaugh précité une discussion sur la façon de combiner les cas d'utilisation ? Jacobson décrivait initialement deux façons de relier des cas d'utilisation : *extends* et *uses*, alors que Rumbaugh proposait d'unifier les deux en : *adds*. Cette distinction a pourtant perduré dans UML jusqu'à la version 1.1 incluse, mais sous la forme incompréhensible de stéréotypes de généralisations. Je ne compte plus les centaines de messages sur les mailing-lists spécialisées qu'ont suscités ces stéréotypes ! C'est bien simple : la sémantique de la relation *extends* était si compliquée à comprendre que nombre d'experts recommandaient de ne pas s'en servir, ou alors l'expliquaient à leur façon ...

Rosenberg a ainsi décrit dans son livre⁴ dix façons d'échouer : l'une d'entre elles consiste à passer du temps à s'interroger pour savoir si telle ou telle relation entre cas d'utilisation est un « *extends* » ou un « *uses* » ! Et puis, coup de théâtre dans UML 1.3, les généralisations stéréotypées deviennent de simples dépendances (flèche pointillée) et les mots-clés se transforment subtilement en « *extend* » et « *include* ». La généralisation entre cas d'utilisation devient la troisième relation

¹ *Be Careful with Use Cases*, E. Berard, 1996

² *Use Cases: the Pros and Cons*, D. Firesmith, 1997, Technical Journal.

³ *Use and Abuse Cases*, M. Fowler, 04/1998, Distributed Computing.

⁴ *Use Case Driven Object Modeling with UML*, D. Rosenberg, 1999, Addison-Wesley.

possible. Effectivement, c'est une amélioration du méta-modèle UML, mais la focalisation autour de ce débat de spécialistes a fait croire à tort aux néophytes que se trouvait là le réel point-clé au sujet des cas d'utilisation.

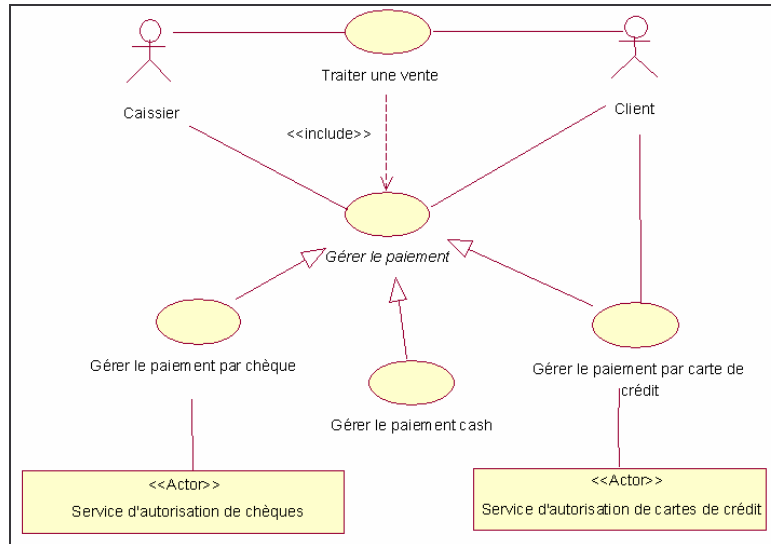


Figure 5 : Exemple de sur-utilisation du diagramme de use cases

2.3 La décomposition fonctionnelle avec la relation « include ».

Lors de mes premières missions de conseil sur l'introduction d'UML et d'UP, j'ai rencontré des analystes qui avaient une longue pratique des méthodes d'analyse structurée (SADT, SA, ou SA/RT) et qui ont rapidement amalgamé les cas d'utilisation à des fonctions SADT (ou des processus SA). Ils détournèrent ainsi les *use cases* pour continuer tout simplement à faire ce qu'ils avaient l'habitude de faire et ne voulaient absolument pas abandonner, à savoir une analyse fonctionnelle structurée, tout en prétendant être conformes à UML ou à UP ! En outre, leurs cas d'utilisation n'avaient pas le point de vue de l'utilisateur, mais celui de l'application étudiée.

La décomposition fonctionnelle, très en vogue dans les années 80, a montré ses limites dans la pratique, en particulier sur les gros projets. Il ne s'agit donc pas de pérenniser, avec la décomposition des cas d'utilisation, les problèmes connus que les méthodes orientées objet visent justement à éviter. D'ailleurs, l'ajout par UML 2.0 de la possibilité de faire référence à un diagramme de séquence dans un autre diagramme de séquence grâce au cadre *ref* permet de résoudre le problème de la décomposition graphique d'interactions complexes sans faire appel à la décomposition de *use cases*.

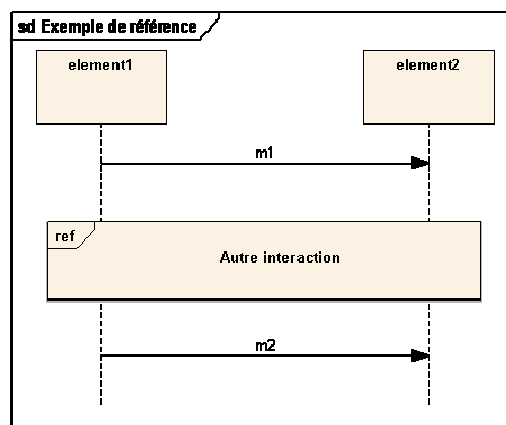


Figure 6 : Le cadre ref comme alternative à la décomposition de use cases

2.4 Autres erreurs fréquentes ...

2.4.1 Confondre cas d'utilisation et processus métier

Symptômes : les événements de début et de fin du cas d'utilisation ne sont pas discernables par le système, on décrit dans le détail les interactions entre les acteurs eux-mêmes. La figure 4 en donne une illustration courante.

Titre : Enregistrer l'emprunt de documents

Acteurs : Bibliothécaire (principal), Client

Description : Ce Use Case commence lorsqu'un client arrive au comptoir de prêt pour emprunter des livres et des vidéos. Il les présente au bibliothécaire qui enregistre l'emprunt de chaque document après avoir vérifié l'identité de l'emprunteur. Le système édite un bulletin de prêt et le client part alors avec les documents empruntés.

Figure 7 : Exemple de confusion avec le niveau « métier »

La description d'un use case doit se concentrer sur les interactions entre les acteurs et le système, pas sur les interactions entre les acteurs. On peut corriger l'exemple précédent de la façon suivante.



Les bonnes descriptions sont claires !

- **Titre :** Enregistrer l'emprunt de documents
- **Acteurs :** Bibliothécaire (principal), Client
- **Description :**
 - *Ce Use Case (niveau métier) commence lorsque le client arrive au comptoir de prêt pour emprunter des livres et des vidéos.*
 - *Ce Use Case (niveau système) commence lorsque le Bibliothécaire fournit l'identifiant de l'emprunteur.*
 - ...
 - *Ce Use Case (niveau système) se termine lorsque le bulletin de prêt (contenant les dates de retour, etc.) est imprimé par le système.*
 - *Ce Use Case (niveau métier) se termine lorsque le Bibliothécaire donne le bulletin et les ressources prêtées au Client.*

53

Figure 8 : Distinction claire entre niveau « métier » et niveau « système »

2.4.2 Identifier de « mauvais » acteurs

Symptômes : l'acteur est en fait interne au système (sous-système ou composant interne), est un simple dispositif d'entrée-sortie (il ne va pas « profiter » de l'exécution du *use case* ...), est une entité physique concrète (et non pas un rôle ou un profil), il n'interagit pas directement avec le système, etc.

3. Les recommandations des experts

3.1 A tout seigneur, tout honneur : I. Jacobson

Dans un article passionnant intitulé : *Use Cases – Yesterday, Today, and Tomorrow*⁵, I. Jacobson a fait un point circonstancié sur l'utilisation des use cases et leur avenir.

Je vous propose d'en retirer quelques idées fortes :

- Les *use cases* sont à la base une idée simple et évidente, ils marchent bien avec les objets et la manière de penser « objet » ;
- Les *use cases* ne sont pas juste une technique pour gérer les exigences, ils permettent de relier toutes les activités à l'intérieur d'un projet ;
- J'ai vu de bons modèles de *use cases* pour des systèmes commerciaux avec seulement cinq *use cases*, et d'autres avec jusqu'à quarante *use cases*. La limite de vingt que je préconise ne prend pas en compte les *use cases* généralisés ou les fragments d'inclusion/extension.
- N'essayez pas de trop formaliser les *use cases* avec des diagrammes complexes comme les diagrammes d'activité et d'états : le modèle de *use cases* est destiné avant tout à communiquer avec les clients et les utilisateurs. Le texte structuré fait très bien l'affaire.
- Rappelez-vous que les *use cases* d'extension servent un objectif très particulier : ajouter un comportement à un *use case* existant sans le modifier. Un problème potentiel consiste à créer de profondes hiérarchies de dépendances d'extension : essayez de ne pas étendre une extension. D'autre part, préférez un *use case* d'extension à un scénario alternatif uniquement s'il est complètement distinct du *use case* étendu : le *use case* de base doit être complet par lui-même et ne pas nécessiter l'extension.
- L'inclusion est simplement un mécanisme permettant de factoriser des flots d'événements communs entre plusieurs descriptions de *use cases*. Ne vous en servez pas pour faire de la décomposition fonctionnelle.
- Les fragments inclus ou d'extension devraient être traités comme ils le méritent : comme moins importants que les « vrais » *use cases*. On devrait même les représenter graphiquement avec un symbole différent.
- Ajoutez des stéréotypes aux *use cases* pour les classer.
- Faites des scénarios des citoyens de première classe : un scénario est une instance de *use case*. Un bon scénario est un bon cas de test.

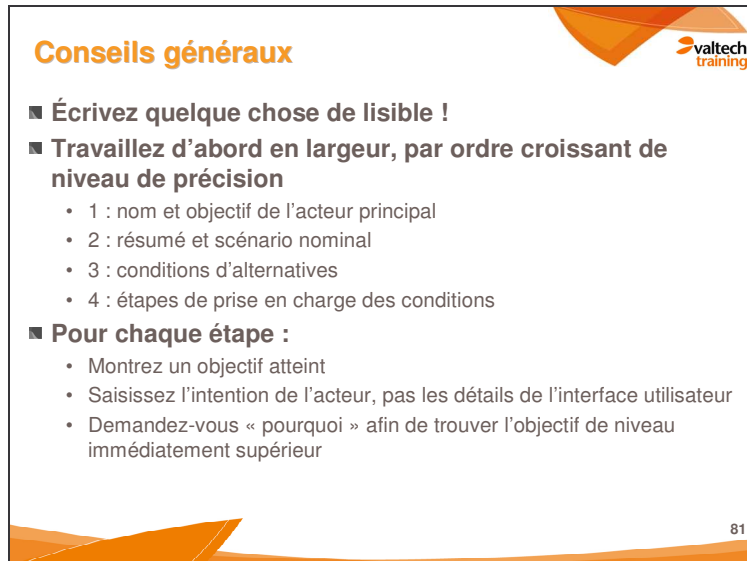
Enfin, pour élargir la portée de son article, I. Jacobson évoque le parallèle entre la relation d'extension entre *use cases* et l'AOP (*Aspect-Oriented Programming*) : ajouter du comportement à un système existant sans le modifier. Il a d'ailleurs publié depuis chez Addison-Wesley un livre entier sur le sujet, intitulé : *Aspect-oriented software development with use cases*.

3.2 Un UseCaseMaster : A. Cockburn

Une des forces du livre d'Alistair Cockburn, intitulé *Rédiger des cas d'utilisation efficaces* est de formaliser les notions de portée, d'acteur et d'objectifs pour détailler tous les types possibles de cas d'utilisation et leur pertinence par rapport à *votre* problème. On peut donc le remercier pour son travail de sappe, parfaitement à contre-courant, visant à redonner toute son importance à la description textuelle des cas d'utilisation en comparaison des pauvres artifices graphiques du diagramme de *use cases*. La principale valeur ajoutée de l'auteur consiste en la quantité de

⁵ Cet article est disponible sur le web, par exemple sur www.therationaledge.com, ou sur le site de Jacobson : http://www.jaczone.com/papers/use_cases-2002-11-26.pdf.

conseils clairs et pragmatiques qu'il fournit pour *écrire* des cas d'utilisation pertinents, utiles, lisibles, et au bon niveau d'abstraction. Un exemple en est donné sur la figure suivante :



Conseils généraux

- **Écrivez quelque chose de lisible !**
- **Travaillez d'abord en largeur, par ordre croissant de niveau de précision**
 - 1 : nom et objectif de l'acteur principal
 - 2 : résumé et scénario nominal
 - 3 : conditions d'alternatives
 - 4 : étapes de prise en charge des conditions
- **Pour chaque étape :**
 - Montrez un objectif atteint
 - Saisissez l'intention de l'acteur, pas les détails de l'interface utilisateur
 - Demandez-vous « pourquoi » afin de trouver l'objectif de niveau immédiatement supérieur

Figure 9 : Conseils généraux de description textuelle de use cases

Citons quelques autres conseils ou rappels très utiles (et détaillés dans le livre ...) :

- Un cas d'utilisation est un essai en prose
- Veillez à la lisibilité du cas d'utilisation
- Une seule forme de phrase
- Identifiez le bon niveau d'objectif
- Laissez de côté l'interface utilisateur
- Les intervenants ont besoin de garanties
- Travaillez d'abord en largeur
- Les acteurs jouent des rôles
- ...


Enfin, dans la partie II de l'ouvrage, Cockburn aborde ce qu'il appelle des sujets récurrents :

- Quand en avons-nous fini ?
- Gérer de nombreux cas d'utilisation
- Cas d'utilisation CRUD et paramétrés
- Modélisation des processus métier
- Les exigences manquantes
- Les cas d'utilisation dans le processus global

Bref, une véritable mine de conseils, d'exemples, et de questions à se poser pour réussir son modèle de cas d'utilisation et leur description ...

3.3 C. Larman

Dans la dernière version de son best-seller, intitulé *UML 2 et les design patterns*, Craig Larman propose un moyen pragmatique de tester la validité d'un use case.



Tests pour la validité des UC

- **Test du patron ! ☺**
 - Votre patron vous demande : « À quoi avez-vous passé la journée ? »
- **Test PME**
 - PME = Processus Métier Élémentaire
 - Tâche effectuée par une personne en un lieu et un temps donnés, en réponse à un événement, et qui ajoute une valeur commerciale mesurable et laisse les données dans un état cohérent
- **Test de la taille**
 - Un cas d'utilisation est très rarement constitué d'une seule action ou d'une seule étape

56

Figure 10 : Conseils de C. Larman pour tester la validité des use cases

C. Larman est également un ardent promoteur du diagramme de séquence utilisé pour représenter le scénario nominal d'un cas d'utilisation : « *A system sequence diagram is a fast and easily created artifact that illustrates input and output events related to the system under discussion.* ». Ce diagramme de séquence système lui sert à identifier les « opérations système » que le concepteur devra concevoir et implémenter, assurant ainsi une forte traçabilité entre les exigences fonctionnelles et le code réalisé.

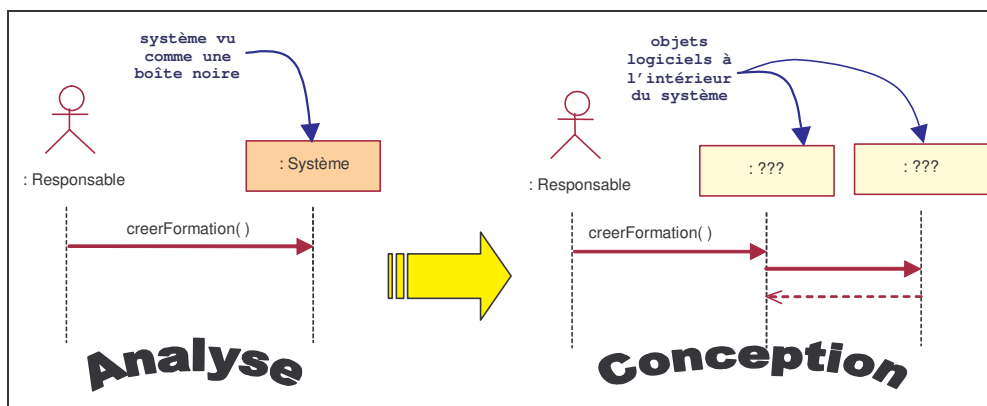


Figure 11 : Du diagramme de séquence système à la conception

4. Conclusion

Dans *UML 2 par la pratique*, je fais une compilation de recommandations simples sur la bonne utilisation des *use cases*, résumée ci-après.

- Un seul acteur principal par cas d'utilisation. Nous appelons acteur *principal* celui pour qui le cas d'utilisation produit un résultat observable. Par opposition, nous qualifions d'acteurs *secondaires* les autres participants du cas d'utilisation.
- Dans la mesure du possible, disposez les acteurs principaux à gauche des cas d'utilisation, et les acteurs secondaires à droite.
- Utilisez la forme graphique du *stick man* pour les acteurs humains, et la représentation rectangulaire avec le mot-clé <<actor>> pour les systèmes connectés.
- Éliminez les acteurs « physiques » au profit des acteurs « logiques » : l'acteur est celui qui bénéficie de l'utilisation du système.
- Répertoriez en tant qu'acteurs uniquement les entités externes (et pas des composants internes au système étudié) qui interagissent directement avec le système.
- Ne confondez pas rôle et entité concrète. Une même entité externe concrète peut jouer successivement différents rôles par rapport au système étudié, et être modélisée par plusieurs acteurs. Réciproquement, le même rôle peut être tenu par plusieurs entités externes concrètes, qui seront alors modélisées par le même acteur.
- Utilisez la relation d'inclusion entre cas d'utilisation pour éviter de devoir décrire plusieurs fois le même enchaînement, en factorisant ce comportement commun dans un cas d'utilisation supplémentaire inclus.
- N'abusez pas des relations entre cas d'utilisation (surtout extension et généralisation) : elles peuvent rendre les diagrammes de cas d'utilisation difficiles à décrypter pour les experts métier qui sont censés les valider.
- Limitez à 20 le nombre de vos cas d'utilisation de base (en dehors des cas inclus, spécialisés, ou des extensions). Avec cette limite arbitraire, on reste synthétique et on ne tombe pas dans le piège de la granularité trop fine des cas d'utilisation. Un cas d'utilisation ne doit donc pas se réduire à une seule séquence, et encore moins à une simple action.
- Le diagramme de cas d'utilisation n'est qu'une sorte de table des matières graphique des besoins logiciels. N'y passez pas trop de temps et d'énergie, mais concentrez-vous plutôt sur la description textuelle détaillée des cas d'utilisation !

Quelques références bibliographiques pour en savoir plus :

- I. Jacobson, *Aspect-oriented software development with use cases*, 2005, Addison-Wesley
- P. Roques, *UML 2 par la pratique*, 4^e éd., 2005, Eyrolles.
- C. Larman, *UML 2 et les Design Patterns*, 2005, Campus Press.
- A. Cockburn, *Rédiger des cas d'utilisation efficaces*, 2002, Eyrolles.
- G. Overgaard, K. Palmkvist, *Use Cases - Patterns and Blueprints*, 2004, Addison Wesley
- E. Yourdon, *Managing Software Reqs - A Use Case Approach*, 2003, Addison Wesley
- D. Kulak, *Use Cases: Requirements in context*, 2003, Addison-Wesley
- K. Bittner, I. Spence, *Use Case Modeling*, 2003, Addison-Wesley
- G. Schneider, J. Winters, *Applying Use Cases*, 2001, Addison-Wesley
- D. Rosenberg, *Use Case Driven Object Modeling with UML*, 1999, Addison-Wesley