Mise en oeuvre d'un processus Agile : développement d'un outil de gestion collaborative de modèles de conception

Ludovic Bouteloup

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MEMOIRE

présenté par Ludovic BOUTELOUP

en vue d'obtenir

LE DIPLÔME D'INGENIEUR C.N.A.M.

en INFORMATIQUE

Mise en œuvre d'un processus Agile : développement d'un outil de gestion collaborative de modèles de conception

Soutenu le 15 Décembre 2010

JURY

Président : M. Eric Gressier-Soudan

Membres : M. Jean-Pierre Giraudin
M. André Plisson
M. Mathias Voisin-Fradin
Mme. Dominique Rieu
Mme Sophie Dupuy-Chessa

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MEMOIRE

présenté par Ludovic BOUTELOUP

en vue d'obtenir

LE DIPLÔME D'INGENIEUR C.N.A.M.

en INFORMATIQUE

Mise en œuvre d'un processus Agile : développement d'un outil de gestion collaborative de modèles de conception

Soutenu le 15 Décembre 2010

Les travaux relatifs à ce mémoire ont été effectués au sein de l'équipe SIGMA (Systèmes d'Information – inGénierie et Modélisation Adaptables) du LIG (Laboratoire d'Informatique de Grenoble) sous la direction de Dominique RIEU.

Remerciements

Je tiens à remercier tout d'abord M. Jean-Pierre GIRAUDIN, responsable de la filière informatique au CNAM de Grenoble pour l'attribution de ce sujet pour lequel je porte un grand intérêt.

Mes remerciements vont également aux membres du jury, pour avoir accepté de consacrer du temps à la lecture de ce mémoire et me permettre de leur présenter le résultat de ce travail.

Je remercie toute l'équipe SIGMA du Laboratoire d'informatique de Grenoble pour sa convivialité, son sérieux et son investissement dans mes recherches.

Mes remerciements se portent aussi vers Marcela MAFTOUL et David RIDEAU, impliqués dans un projet en Technologies de l'Information pour Grenoble Universités, pour leur retour d'expérience, leur aide et conseils.

Je remercie M. Pascal BOURGEOIS, stagiaire en cycle ingénieur CNAM au sein de l'équipe SIGMA, Mme Sophie DUPUY-CHESSA et Mme Dominique RIEU, toutes deux enseignantes et membres de l'équipe SIGMA du Laboratoire d'Informatique de Grenoble pour leur collaboration sur ce projet, leur présence, leur réactivité et leur implication.

Je remercie encore tout particulièrement Mme Dominique RIEU, enseignante et membre de l'équipe SIGMA du Laboratoire d'Informatique de Grenoble pour s'être investie en tant que tutrice, pour le partage de ses connaissances et pour sa disponibilité tout au long de ces neuf mois.

Enfin, pour leurs encouragements, leur soutien et surtout leur patience, je remercie ma famille, mes proches et toutes les personnes que j'ai pu impliquer dans la réalisation de ce travail.

Table des matières

Table des figures

Table des tableaux

Liste des abréviations

AGAP : Atelier de Gestion et d'Application de Patrons.

AUP : Agile Unified Process.

DSI : Direction des Systèmes d'Information.

COPEN : COmmunities of Patterns Environment.

FDD : *Feature* Driven Development.

HTML : Hyper Text Markup Language.

JSON : JavaScript Object Notation.

LIG : Laboratoire d'Informatique de Grenoble.

MVC : Modèle-Vue-Contrôleur.

RAD : Rapid Application Development.

RUP : Rational Unified Process.

SIGMA : Systèmes d'Information – inGénierie et Modélisation Adaptables.

SSII : Société de Services en Ingénierie Informatique.

UML : Unified Modeling Language.

XP : eXtreme Programming.

XUP : eXtreme Unified Process.

SGC : Système de Gestion de Contenu.

SVN : SubVersioN.

Introduction

Introduction

L'informatique prend une place de plus en plus importante dans la société, il est désormais incontournable de faire appel à une société de services, un éditeur de logiciel, ou même souvent d'avoir son propre centre de compétences au sein de la société. Les SSII (Société de Services en Ingénierie Informatique) sont de plus en plus présentes et les entreprises sont de moins en moins frileuses au lancement de nouveaux projets. Mais les méthodes de développement n'évoluent pas aussi vite que le développement de ces produits lui-même. Nous arrivons très souvent à des projets insatisfaisants pour le client, ne correspondant pas aux attentes et exigences, parfois même n'aboutissant pas !

Il est important de trouver une façon de penser, une façon de travailler qui corresponde au besoin actuel. Ce besoin est de fournir rapidement le produit et qu'il corresponde exactement à ce que souhaite le métier. C'est pour faire face à cette forte demande et ce manque de réussite des projets que de nouvelles méthodes de developpement voient le jour.

D'après Seligman, une méthode est un ensemble de moyens d'investigation constitués par *« a way of thinking, a way of modeling, a way of working and a way of supporting »*. Cela signifie que toute méthode est définie par une approche ou un paradigme, comporte des modèles pour définir le produit, propose des processus ou démarches et doit être supportée par des outils logiciels.

Une méthode de gestion de projets permet d'éviter le risque de livrer un produit de mauvaise qualité. Comme le dit Tom Gilb en 1998 dans son œuvre Principles of Software Engineering Management : *« Si vous n'éliminez pas activement les risques de votre projet ce sont eux qui vous élimineront ».*
Il y a eu une évolution dans l'appréhension d'un projet informatique depuis les années 1960. Tout d'abord l'approche est simple : une décomposition d'un projet en sous-tâches qui enchaînées les unes après les autres aboutissent au projet global. Le fait d'effectuer les tâches sans pouvoir revenir dessus pose des problèmes, des risques d'échecs menacent. La démarche se retourne donc vers un parallélisme entre les phases de développement et de vérification afin de pouvoir contrôler au fur et à mesure l'adéquation entre le résultat demandé et le résultat obtenu. Cette vision n'est toujours pas suffisante car, même si elle permet de diminuer les risques de dérive, elle ne permet pas de revenir sur une tâche déjà réalisée si celle–ci ne convient plus. C'est pour cela qu'une vision en spirale a ensuite émergée, elle permet de travailler sur chacune des tâches avec la possibilité de remaniement afin d'étayer le travail déjà réalisé si besoin, de plus une modélisation objet associable à cette pratique facilite la conception et la réutilisation de l'existant. Se succédèrent donc des approches cartésiennes pour la décomposition en sous-tâches successives, puis systémiques pour la modélisation des données et enfin objet pour accentuer cette modélisation et fournir plus de détails. Malgré cette évolution, les besoins et l'implémentation ont toujours trop souvent un résultat qui ne satisfait pas le client. C'est pour cela qu'une vision centrée sur le client et sa description des fonctionnalités voit le jour après les années 1990. Cette philosophie appelée Agile permet de mettre le client et les utilisateurs au centre de l'équipe et donc de répondre réellement au besoin. L'approche en spirale est aussi revue pour devenir itérative et pouvoir fournir à chaque cycle un lot de fonctionnalités opérationnelles pour le client. Cette phase a aussi pour but d'avoir un retour des utilisateurs afin de pouvoir réadapter la fonctionnalité à l'itération suivante et convenir totalement aux attentes [1].

Après la production d'une synthèse « Etude et synthèse sur les méthodes Agile » faisant un état de l'art sur les méthodes Agile que ce mémoire reprend vis-à-vis de la gestion de projets, ce stage a permis d'implémenter une organisation de travail Agile, le choix s'est porté sur un projet de développement d'un outil de gestion de modèles réutilisables et documentés sous la forme de

patrons. Un sujet intéressant pour de nombreuses raisons. Le savoir et le savoir-faire sont toujours grandissants et encore trop mal exploités de nos jours, de nombreux secteurs comme les systèmes d'information aimeraient améliorer la performance des processus d'élaboration et la qualité des produits. C'est pour cela qu'apparaît une nouvelle forme d'ingénierie : la réutilisation de composants. Cette application aura pour but d'aborder la réutilisation à travers des catalogues de patrons dont la principale caractéristique est la constitution d'une base de savoirs et de savoir-faire suivant un formalisme permettant son utilisation dans différents contextes [2].

Un environnement permettant la communication et la collaboration pour la création de patrons « design for reuse », mais aussi la réutilisation de ces connaissances modélisées, la description de la mise en pratique et un retour d'expérience « design by reuse » est donc nécessaire [3].

L'environnement visé doit permettre à des communautés de partager, maintenir des savoirs et savoir-faire sous la forme de patrons collaboratifs. Longtemps les patrons (design patterns, patrons d'architecture) ont été perçus comme des connaissances stabilisées. A présent, l'approche patron est également perçue comme un moyen structurant et efficace de partage de connaissances non-stabilisées. Il faudra se concentrer sur la communication et la collaboration, mais parler de qualité de communication et de collaboration pour ce genre d'outils informatiques nécessite de mettre en place un espace communautaire avec tous ses rôles, droits et fonctionnalités.

La réalisation de ce projet, par la création et l'utilisation de patrons ou même la collaboration qui peut être présente autour pour trouver une modélisation simple, générique et efficace est un aspect très complémentaire à la philosophie Agile car tous deux basés sur la communication et le partage.

Une première partie de ce mémoire présentera un état de l'art au niveau des trois points essentiels à l'implémentation de cette application : les méthodes de gestion de projets Agile, les patrons collaboratifs ainsi que les espaces communautaires afin d'avoir un recul critique et judicieux sur les technologies à utiliser. La deuxième étape de ce travail est logiquement la réalisation du projet, elle présentera la solution de manière concrète (à l'aide de diagrammes), la méthode mise en œuvre pour développer l'outil ainsi que les livrables. Enfin un bilan et les perspectives d'avenir d'utilisation, de reprise et d'évolution viendront clore ce mémoire.

Etat de l’art

1. Etat de l'art

1.1. Gestion de projets : une intéressante évolution

1.1.1. Cycles de vie : un historique

Le cycle de vie d'un logiciel désigne toutes les étapes du développement, de sa conception à sa livraison. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés. Des modèles de cycle de vie ont été mis au point définissant les étapes du développement ainsi que les documents à produire permettant de les valider.

Pour arriver à des méthodes de gestion de projets en cycle de vie itératif, une évolution logique s'est faite sur l'importance des risques d'échec. Un historique de l'évolution des cycles de vie est disponible en figure 1.

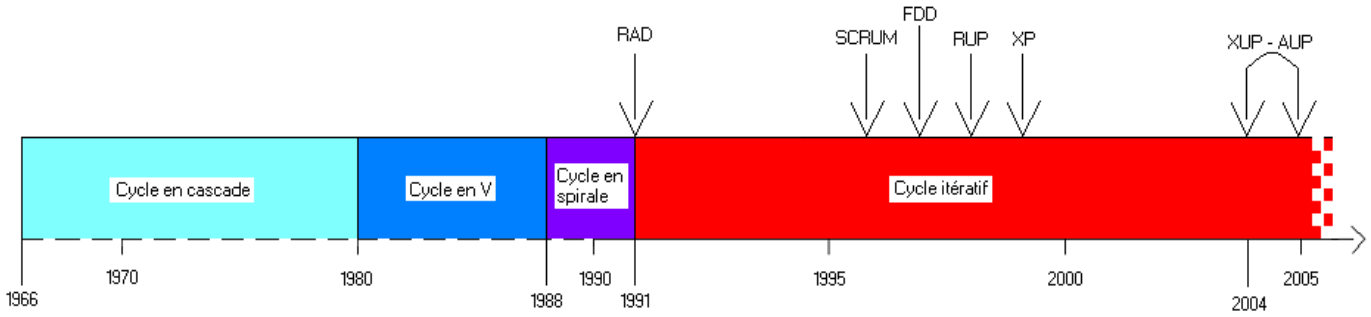


Figure 1 : Evolution des cycles de vie et leurs méthodes

Depuis le cycle en cascade où chaque phase se réalisait uniquement si la précédente était terminée, pour arriver à un cycle d'itérations où toutes les phases sont retouchables, il y a eu des solutions de mise en évidence des tests dans le développement par un cycle en V puis un cycle en spirale permettant d'approfondir à chaque version successive le travail réalisé.

Cycle en Cascade

La première approche consistait à développer et à valider, les modèles en cascade, cycle mis au point en 1966 et décrit figure 2. C'est une vision simple avec des dates bien définies pour chaque étape et on ne pouvait passer à la suivante qu'après avoir jugé la précédente satisfaisante. L'inconvénient majeur du modèle de cycle de vie en cascade est que la vérification du bon fonctionnement du système est réalisée trop tardivement : lors de la phase d'intégration, ou pire, lors de la mise en production [4].

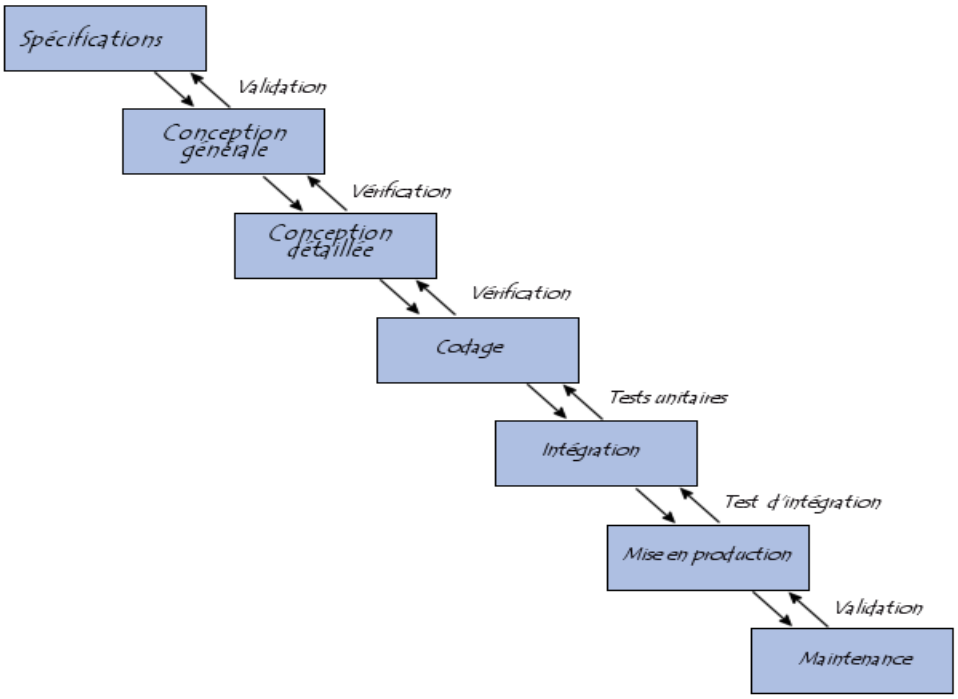


Figure 2 : Modèle de cycle de vie en cascade [5]

Cycle en V

Par la suite, d'autres modèles ont vu le jour comme le cycle en V (figure 3), un standard de l'industrie depuis les années 1980 et certainement le plus utilisé. Il part du principe que les procédures de vérification de la conformité du logiciel aux spécifications doivent être élaborées dès les phases de conception. En effet, il y a une réelle mise en valeur des tests lors du développement du logiciel. Malgré cette relation supplémentaire, la vérification du bon fonctionnement du système reste trop tardive [4].

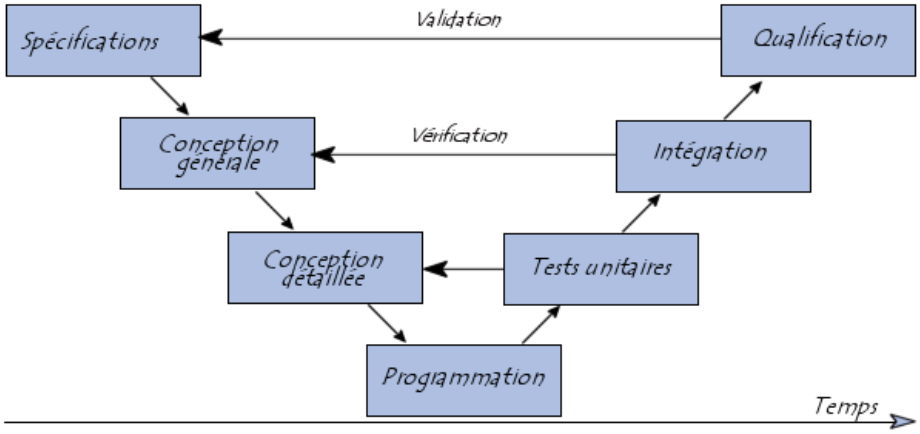


Figure 3 : Modèle de cycle de vie en V [5]

Cycle en Spirale

En 1988, Barry Boehm commence à parler de cycles différemment, il parle de versions successives en définissant un modèle en Spirale décrit en figure 4Figure 4. Cette approche consiste à affiner l'analyse préliminaire au cours de chaque cycle. Le développement reprend les différentes étapes du cycle en V, mais par l'implémentation de versions successives, le cycle recommence en proposant un produit de plus en plus complet. Ce procédé permet de mettre l'accent sur l'activité d'analyse

des risques trop peu visible sur les modèles précédents, ainsi que l'intérêt d'élaborer des prototypes [4].

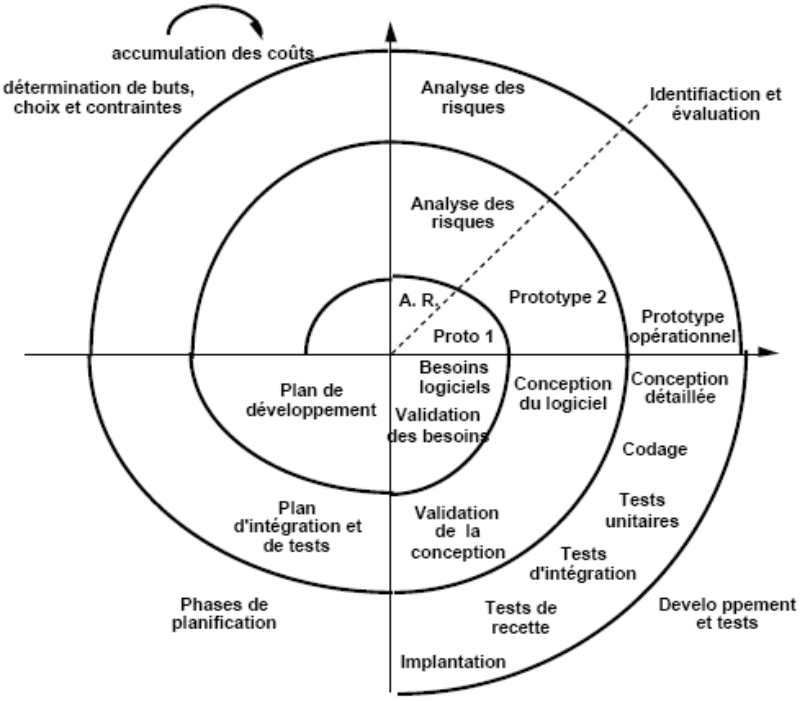


Figure 4 : Modèle de cycle de vie en spirale [6]

Cycle Itératif Agile

Le nombre trop élevé de projets échoués a mené à l'évolution des méthodes de conduite de projet qui ne correspondaient plus au besoin des entreprises. La figure 5 montre la progression des succès de projets réalisés depuis 1995.

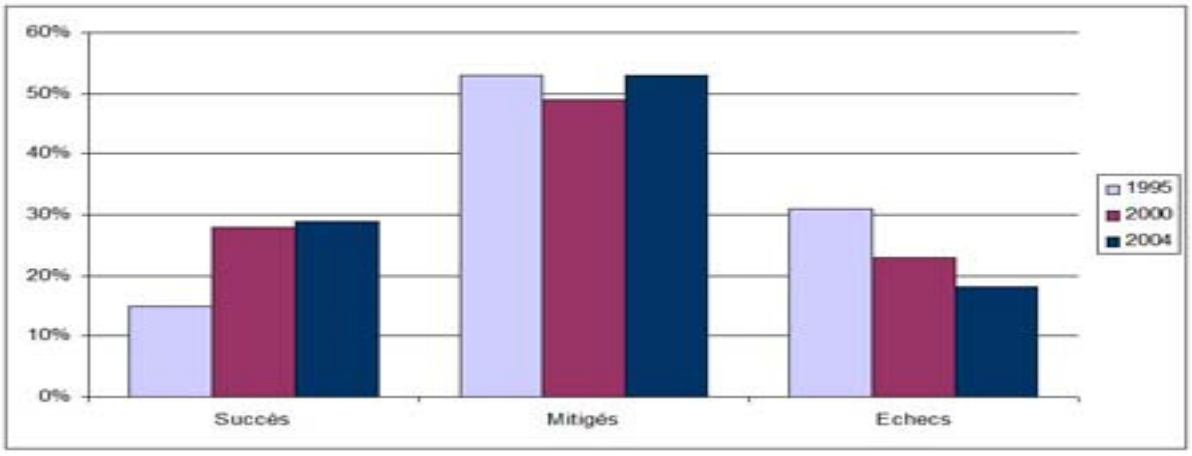


Figure 5 : Pourcentage de réussite des projets informatiques [8]

Cette figure illustre que depuis la venue des méthodes Agile, le nombre d'échecs en termes de projets informatiques a pratiquement été divisé par deux en dix ans (1995-2004). Ce n'est pas tout, on peut voir aussi que le taux de projets mitigés reste inchangé, autrement dit la baisse du taux d'échecs se traduit directement par des réussites de projets informatiques.

C'est à partir des années 1990, avec l'approche Agile, que le contrôle du bon fonctionnement arrive tôt dans le projet. C'est d'abord par l'approche itérative des phases de développement et par la

livraison des lots issus de celle-ci que les erreurs détectées ne deviennent pas des échecs en fin de projet mais simplement des améliorations à apporter lors d'une prochaine itération [7]. Voici en figure 6 un modèle générique de la vision Agile.

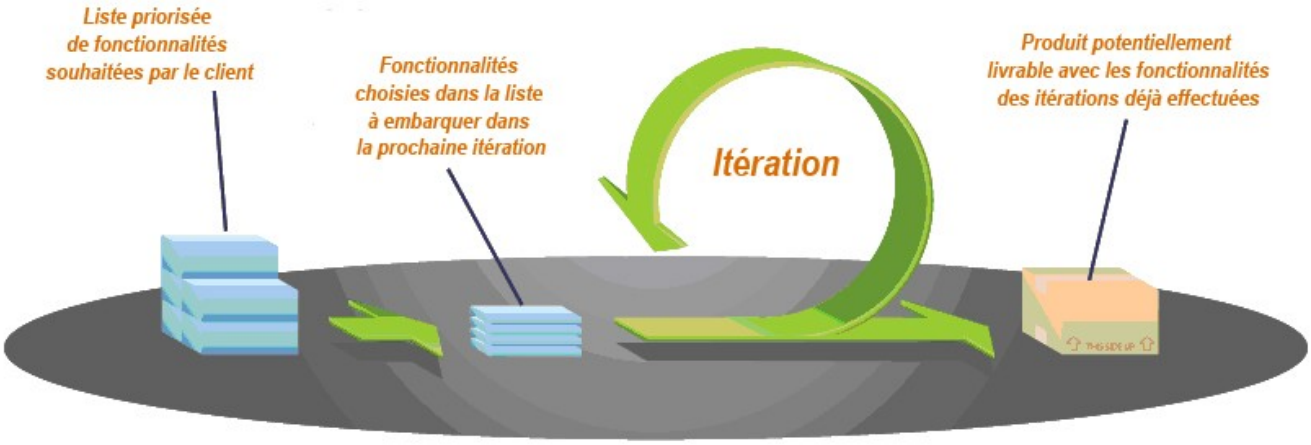


Figure 6 : Vue globale de cycle de vie Agile [1]

Ce cycle de vie permet de livrer de façon successive les fonctionnalités souhaitées par le client. Chaque itération permet de fournir un produit livrable, utilisable, comprenant les fonctionnalités des itérations effectuées

1.1.2. Une nouvelle vision projet : Manifeste Agile

Depuis les années 1990, une nouvelle façon de penser « développement logiciel » émerge. Mais c'est en novembre 2001, lors d'un rassemblement de dix-sept personnes telles que Ken Schwaber, Kent Beck et Jeff Sutherland reconnus pour leurs apports respectifs au développement d'applications informatiques sous la forme de plusieurs méthodes (eXtreme Programming, Scrum etc.) qu'est ressorti le Manifeste de Développement Logiciel Agile formalisant ce mouvement. C'est dans le cadre relaxant d'une station de ski des montagnes de l'Utah entre discussions, ski et bons repas que la vision Agile apparut sous forme de consensus. Ce sont finalement quatre valeurs qui sont ressorties de cette rencontre ainsi que douze principes associés permettant le développement Agile de logiciels. La première valeur est l'aspect humain et communicatif avec la mise en avant de l'interaction entre les personnes plutôt que les processus et les outils. Pour deuxième valeur : la livraison d'un produit opérationnel est la priorité vis-à-vis d'une documentation parfois excessive. La troisième valeur est la collaboration avec le client, elle est devenue primordial et ne doit pas se restreindre à une simple négociation de contrat. Une dernière valeur Agile est la réactivité face au changement qui est préférée au suivi d'une conception définitive qui parfois ne mène pas à la demande réelle du client [9].

Sur la base de ces quatre dimensions d'un projet, voici les douze principes déclinés aussi appelés bonnes pratiques ou « Best practices » définissant le développent dit Agile [9] :

1. Notre première priorité est de satisfaire le client en livrant tôt et régulièrement des logiciels utiles.

2. Le changement est accepté, même tardivement dans le développement. Les processus agiles exploitent le changement comme avantage compétitif pour le client.

3. Livrer fréquemment une application fonctionnelle, toutes les deux semaines à deux mois, avec une tendance pour la période la plus courte.

4. Les experts métier et les développeurs doivent collaborer quotidiennement au projet.

5. Bâtissez le projet autour de personnes motivées. Donnez-leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail.

6. La méthode la plus efficace pour transmettre l'information est une conversation en face à face.

7. Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet.

8. Les processus agiles promeuvent un rythme de développement soutenable. Commanditaires, développeurs et utilisateurs devraient pouvoir maintenir le rythme indéfiniment.

9. Une attention continue à l'excellence technique et à la qualité de la conception améliore l'agilité.

10. La simplicité (l'art de maximiser la quantité de travail à ne pas faire) est essentielle.

11. Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto organisent.

12. À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens.

Comme l'a souligné une personne en commentaire d'un article sur l'Agile publié par Jean-Claude Grosjean, un treizième principe pourrait être cohérent avec cette philosophie de cycle itératif [10] :

13. Eviter absolument les régressions entre les livraisons d'applications fonctionnelles.

Ce treizième principe rappelle la métaphore du cycle PDCA (Plan, Do, Check, Act/Adjust) d'amélioration de la qualité : la roue de Deming.

La gestion de projets Agile requiert de mettre le client au centre du projet, d'avoir une équipe soudée autour de lui, qui s'autogère et qui a pour valeur première la communication en face à face. Cette équipe doit fournir des livraisons régulières. La communication orale ainsi que la livraison de lots réguliers peuvent entraîner des pertes d'information ou de la régression, ces principes demandent de l'attention et de la rigueur.

Depuis Janvier 2002, un regroupement de professionnels s'est formé afin de créer une alliance Agile qui a pour objectif de défendre le manifeste et de promouvoir les méthodes de développement qui s'appuient sur ses principes. Actuellement plus de quatre mille deux cent soixante membres à travers le monde organisent des événements tels que des séminaires de certification Agile ou du « coaching ». Ce phénomène est de plus en plus présent de nos jours. Il encourage à faire évoluer les visions sur le modèle Agile. Une majorité des membres de cette alliance sont des sociétés de « consulting » qui répercutent ensuite leur savoir-faire sur les autres sociétés [11]. De plus en plus, les SSII comme OPEN (sponsor pour l'Agile tour 2008 de Grenoble) ou les grands groupes innovateurs comme Yahoo se mettent à « l'Agile » [12]. Ce mouvement permet une avancée dans les méthodes de développement qui se basent de plus en plus sur cette nouvelle philosophie.

1.1.3. Des méthodes de développement Agile

Un certain nombre de méthodes respectent les critères du manifeste Agile. Ce sont d'ailleurs elles qui ont poussé à publier ce consensus. Certaines sont mises en pratiques et ont prouvé leur efficacité.

Voici les méthodes les plus répandues :

RAD (Rapid Application Development) est mise au point par James Martin en 1991. Même si le cycle « en spirale » permettait de revenir sur chaque phase, elle est la première méthode de développement de logiciels où le cycle de développement est en rupture fondamentale avec celui des méthodes antérieures dites « en cascade ». C'est principalement la vision en cycle itératif et à la construction d'un produit partiel mais livrable à chaque itération que **RAD** fut innovante [13].

SCRUM est créée par Ken Schwaber et Jeff Sutherland en 1996. Le terme SCRUM est emprunté au rugby et signifie mêlée, Il permet de souligner que cette méthode met en avant l'aspect équipe soudée qui cherche à atteindre un but comme c'est le cas en rugby pour avancer le ballon lors d'une mêlée. Le principe de base de Scrum est d'impliquer le client comme membre de l'équipe, de le focaliser sur un ensemble de fonctionnalités à réaliser dans des itérations de durée fixe de deux à quatre semaines et de fournir un produit partiel fonctionnel à chaque itération. Par contre, la méthode Scrum ne propose aucune technique d'ingénierie du logiciel comme des normes ou pratiques de développement. Il est nécessaire de lui adjoindre une méthode complémentaire [14].

FDD (*Feature* Driven Development) est proposée en 1997 par Jeff De Luca avec l'influence de Peter Coad et son approche de modélisation objet. C'est lors d'un projet ayant échoué à deux reprises dans le domaine de la restructuration bancaire à Melbourne en Australie que Jeff De Luca a pu faire ses preuves en utilisant une approche minimaliste de cinq étapes nommée FDD. Cette méthode est plus exactement un guide qu'un processus normatif qui respecte elle aussi un cycle itératif avec des livraisons régulières [15].

RUP (Rational Unified Process) proposée en 1998 par Rational Software (IBM) est l'une des plus célèbres implémentations de la méthode UP (Unified Process). Même s'il en existe d'autres comme XUP (Extreme Unified Process) qui est une instanciation hybride intégrant UP avec Extreme Programming, ou AUP (Agile Unified Process) mettant plus l'accent sur le contexte du projet que sur la théorie, RUP permet plutôt de donner un cadre au développement logiciel afin d'allier philosophie Agile et modélisation répondant aux exigences fondamentales préconisées par les créateurs d'UML (Unified Modeling Language) [16].

XP (eXtreme Programming) a été mise en place par Kent Beck, en collaboration avec Ward Cunningham et Ron Jeffries sur un projet pour la compagnie Chrysler dans les années 1990. C'est en 1999 qu'est officialisée la méthode XP avec la sortie du livre « eXtreme Programming explained » [17]. Elle recherche l'efficacité maximale en concentrant l'effort de travail sur l'objectif de développer le bon logiciel et de ne pas s'égarer. La démarche est légère, pragmatique, disciplinée, empirique et adaptative [18]. Celle-ci correspond très bien à la couche orientée implémentation complémentaire à SCRUM car la philosophie est exactement la même [17].

1.1.4. Agilité : Mise en pratique des méthodes

Il est intéressant de comparer ces méthodes sur différents aspects tels que les phases de leur cycle de vie et leur niveau d'abstraction. D'autre part, le niveau d'exigence avec les acteurs, leur rôle ainsi que les artefacts rentrent en compte. Le quotidien des projets peut énormément varier suivant la méthode utilisée, cela demande encore de bien cerner la méthode à adopter sur un projet. Un dernier point qui reste tout aussi important est la palette d'outils disponibles pour mener à bien chaque phase de développement. Certains outils sont chers, d'autres complexes, certaines méthodes sont riches en logiciels d'autres moins. Mais toutes les méthodes n'ont pas besoin de logiciels spécifiques, il est parfois possible de les mettre en place avec un nombre d'outils minimaliste.
La comparaison de quatre méthodes pertinentes pour ce sujet et fréquemment utilisées **FDD**, **RUP**, **XP** et **SCRUM** va permettre de mieux situer le périmètre Agile existant, et de pouvoir effectuer une critique et un choix judicieux de méthode appartenant à cette philosophie. Ces méthodes sont étudiées ci-dessous suivant quatre dimensions :

- Cycle de vie.
- Acteurs, rôles et artefacts.
- Quotidiens.
- Outils.

Des cycles de vie Agile

Cette vision « Manifeste Agile » est respectée par les méthodes dites Agile. Elles l'appliquent toutes, en ajoutant leurs spécificités. C'est maintenant que l'on peut commencer à constater des différences entres méthodes. Certaines vont être généralistes et vont donner une trame d'avancement sur le projet plutôt que des normalisations comme **SCRUM** et **XP**, d'autres vont cibler la modélisation orientée objet en complétant le langage UML comme FDD ou RUP.

Tout d'abord la méthode la plus abstraite est SCRUM. Elle ajoute une trame sans imposer de technologie de développement. La façon de procéder à l'intérieur de l'itération n'est pas imposée. Comme on peut voir sur la vue globale en figure 7, une « mêlée » quotidienne, c'est-à-dire une petite réunion d'équipe effectuée tous les jours a été ajoutée et certains termes renommés par rapport à la vue globale Agile (figure 6).

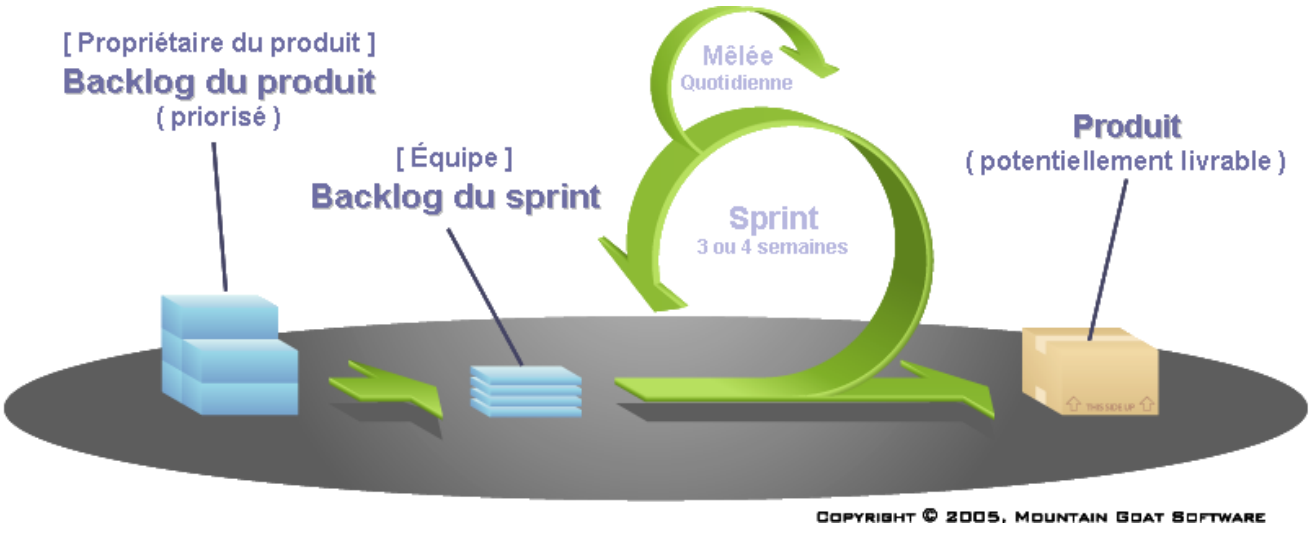


Figure 7 : Vue globale SCRUM [14]

Avec une abstraction légèrement moins élevée que celle de **SCRUM** au niveau de l'organisation de l'équipe, la méthode **XP** se base vraiment sur le cycle Agile précisé en figure 6. Elle permet de garder une liberté au niveau du cycle itératif et sur le choix des technologies mais, en échange, elle est très intéressante au sein de l'itération : elle apporte des contraintes, des bonnes pratiques sur la façon de travailler en se basant sur des valeurs comme la simplicité, la communication, les retours d'information, le courage et le respect [18].

Ces deux méthodes à forte abstraction n'ont pas une vision technique des fonctionnalités à apporter mais plutôt fonctionnelle. Les « use cases » habituels sont remplacés par des « *user stories* » qui détaillent les fonctionnalités sans utiliser de mots-clés techniques afin d'être vraiment en phase avec le client et les utilisateurs. La première pousse à l'extrême le management en privilégiant la communication et l'esprit d'équipe et la seconde l'intégration avec des techniques de développement bien spécifiques telles que la programmation en binôme, l'appropriation collective du code avec du remaniement par les autres membres de l'équipe si besoin. Une manière efficace est donc d'appliquer la méthodologie **SCRUM** et de mettre en place **XP** au sein de l'itération comme le montre la figure 8. En effet le résultat du mélange des pratiques de « management » **SCRUM** avec les pratiques d'intégration **XP** apporte une méthode bien cadrée dans tous les domaines et phases d'un projet [19].

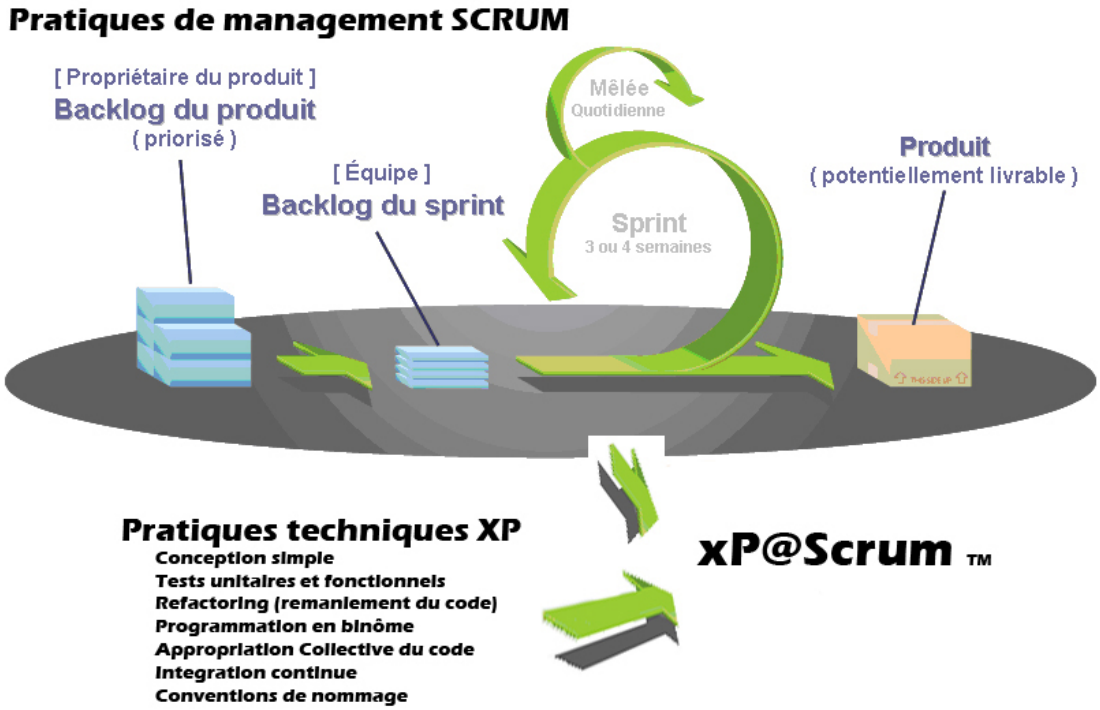


Figure 8 : Vue globale SCRUM et XP [19]

De plus, un point essentiel de l'Agile peu mis en valeur dans les figures précédentes est le principe 2 du manifeste explicité précédemment : le changement est accepté, même tardivement dans le développement. Les processus agiles exploitent le changement comme avantage compétitif pour le client. Voici un schéma plus détaillé en figure 9 issu du modèle **XP** mettant en valeur ce principe avec cet ajout de scénario utilisateur qui ressort de l'itération pour revenir dans les fonctionnalités à développer :

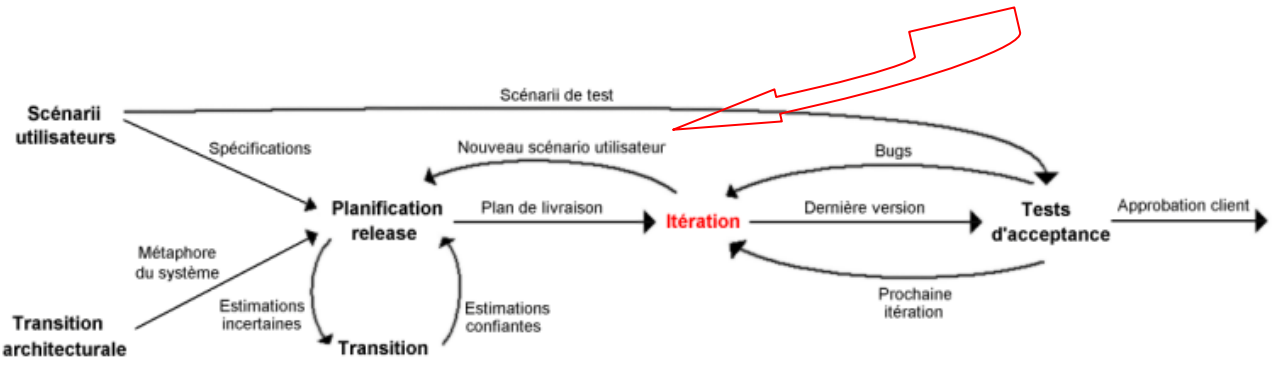


Figure 9 : Modèle de cycle de vie XP [20]

Cette adaptation au changement est illustré sur cette figure par une flèche « Nouveau scénario utilisateur » permettant de planifier le scénario d'un nouveau besoin découvert lors d'une itération.

Dans un genre beaucoup moins abstrait les méthodes FDD et RUP ne se focalisent pas sur cette souplesse de management mais associent la philosophie Agile à la modélisation objet UML utilisée en génie logiciel. UML est un langage très abouti, utilisant des cas d'utilisation à un niveau technique détaillé et normé. C'est comme cela que le niveau d'abstraction des spécifications de fonctionnalités mais aussi le choix des technologies se concrétisent. Le plus complexe à mettre en œuvre et à planifier mais le moins concret des deux est RUP. Il remanie en quelque sorte le modèle Agile au point de vue itération en donnant des possibilités itératives tout au long des phases du cycle de vie. La modélisation en figure 10 illustre bien cette possibilité [21].

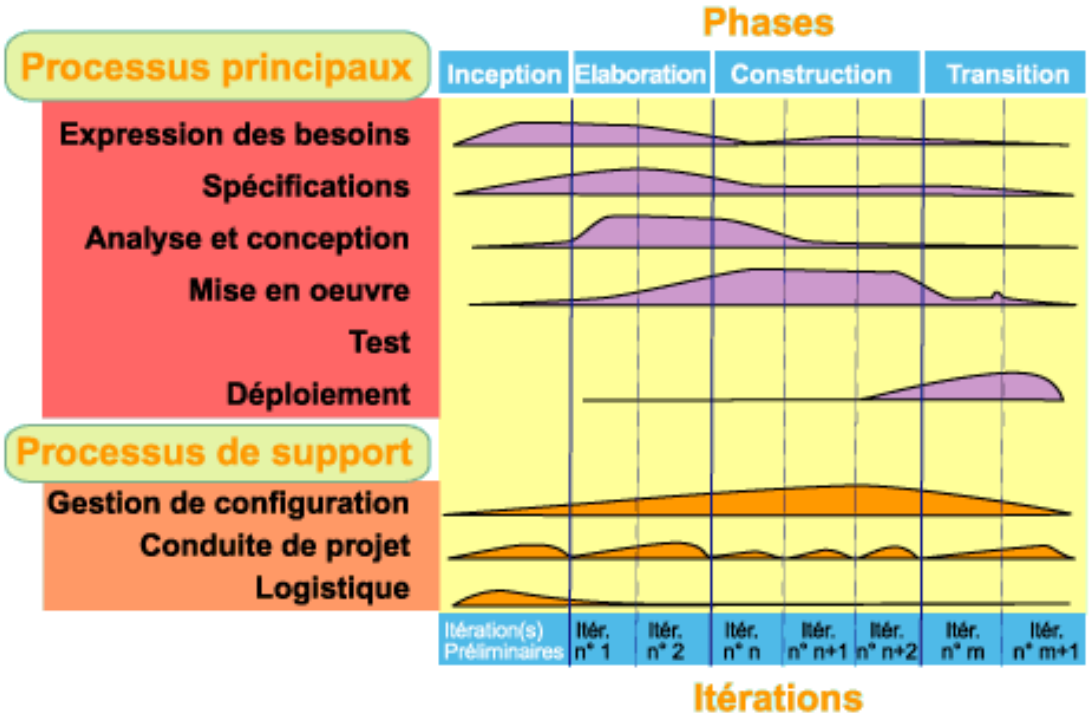


Figure 10 : Phases du cycle de vie RUP [21]

Comme **RUP**, **FDD** ne laisse pas beaucoup de liberté de conception car également basé sur UML. Sa complexité est moins grande que celle de **RUP** mais il a une particularité qui est celle d'élaborer un modèle global de classes UML avant de spécifier les fonctionnalités à embarquer. Ce modèle global est une ébauche qui sera bien sûr améliorée tout au long des itérations [22]. Le cycle de vie d'un projet **FDD** est disponible en figure 11.

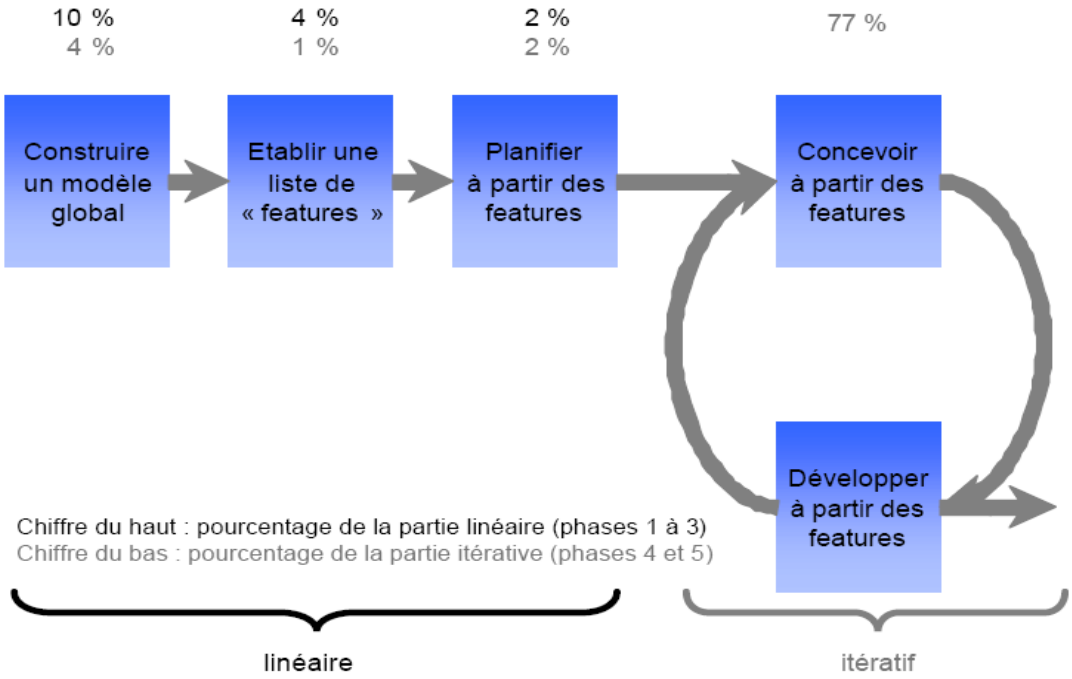


Figure 11 : Phases du cycle de vie FDD [23]

Il est intéressant de remarquer la répartition du temps entre les phases, même si la majeure partie du travail est très logiquement dans le cycle itératif, et qu'une étape de gestion des fonctionnalités le précède, tout de même 14% du projet est dédié à la construction initiale d'un modèle global normé.

Cette première étape technique qui consiste à élaborer un diagramme de classes UML est-elle en adéquation avec la philosophie Agile ?
La première valeur Agile est de privilégier l'interaction entre les personnes plutôt que les processus et les outils. Agile vise aussi à développer en allant au plus simple, KISS (Keep It Simple, Stupid), faire simple, stupide. Il est d'ailleurs possible d'illustrer cette vision avec une citation d'Antoine de Saint-Exupéry : *« Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher ».* L'anticipation sur la création d'un modèle UML avant même d'avoir listé les fonctionnalités à développer ne suit pas cette approche, il faudrait d'abord définir ce qu'il faut développer avant de commencer la conception. L'utilisation d'UML ne favorise pas non plus la vision Agile. Il y a donc un paradoxe à vouloir rigidifier en les codifiant des pratiques par nature destinées à la souplesse. Même s'il existe la variante Agile **AUP** de RUP, il est cohérent de penser que **FDD** et **RUP** ne sont pas des méthodes dites Agile visant à la simplicité dans l'organisation du projet, dans l'utilisation des outils et dans les phases du cycle de vie. Ces méthodes à la frontière de l'agilité sont présentées comparées et critiquées comme des méthodes Agile dans ce mémoire car elles ont leur rôle sur l'avancée et l'image de cette philosophie, au niveau de la réflexion, de la méthodologie et du périmètre Agile.

Même si l'avantage commun est l'itération, chacune de ces méthodes à sa particularité : c'est la situation, le contexte du projet qui décidera laquelle est la plus adaptée. **SCRUM** ne permet pas autant de rigueur que **RUP** dans les phases d'implémentation. Cette méthode correspond à des projets avec un nombre de personnes limité afin de favoriser la communication. **RUP** n'attache pas autant d'importance au management organisationnel mais détaille beaucoup plus la partie technique, ce qui permet de travailler avec une équipe plus importante. **FDD** est une version moins extrême que **RUP**, les pratiques d'implémentation ne sont pas autant développées et peuvent donc être plus facilement mises en place. Dans la même idée que **FDD**, **XP** pousse moins loin la gestion organisationnelle mais ne néglige pas la partie implémentation, ce qui permet de gérer tous les aspects du projet. Mais une méthode à retenir est la combinaison de **XP** à **SCRUM.** Les pratiques

d'implémentation ajoutées à une gestion d'organisation très complète permettent un produit de qualité car chaque phase du projet est contrôlée. De plus, ces deux méthodes laissent une souplesse, une ouverture sur les choix d'implémentation permettant d'être mis en place sur de nombreux projets, ce que ne procurent pas **FDD** ou **RUP**.

Chaque méthode a son intérêt une fois placé dans un contexte. En effet il est préférable d'utiliser une méthode plutôt qu'une autre une fois que l'environnement de travail est défini, mais aussi que la taille et la capacité de l'équipe sont précisées, que les délais de livraison sont fixés ou même que les contraintes clients sont exposées.

Trop souvent les membres et leur rôle ou les artefacts rentrant en compte tout au long d'un projet sont mal gérés. La manière de gérer un projet doit être bien définie pour qu'il aboutisse sur un succès. L'annexe C décrit de manière humoristique le manque d'organisation qu'il peut y avoir sur une structure mal définie menant à l'échec d'un projet.

Pour la réussite d'un projet Agile, il faut que les rôles et les artefacts soient bien définis et compris de tous.

Des acteurs et leurs rôles

XP a une vision simple des acteurs d'un projet [7].

1. **Client :** équivalent au rôle de directeur de produit ; il peut s'agir du client présent au sein de l'équipe ou d'un membre de l'équipe jouant ce rôle. Il doit rester disponible pour orienter les développeurs sur les fonctionnalités et leur priorité en exprimant les besoins de l'application sous forme de scénarios. Il a le rôle d'acceptation ou de rejet des résultats.
2. **Manager :** garant de l'application ; il s'assure que le processus est bien suivi, organise et planifie les réunions, enregistre les résultats, et protège son équipe des perturbations extérieures.
3. **Développeur :** Il estime les délais, écrit le logiciel en fonction des tests unitaires et s'occupe des tâches d'ingénierie logicielle.
4. **Testeur :** implémente et exécute les tests fonctionnels en fonction des scénarios et s'assure de la bonne compréhension des résultats de ses tests.

SCRUM réorganise et complète cette liste de rôles en ajoutant des acteurs permettant d'apporter un point de vue supplémentaire [19] :

1. **Product Owner :** assimilé au Client dans XP.
2. **Scrum Master :** assimilé au Manager dans XP.
3. **Team Members :** correspond au Développeur et Testeur dans XP, c'est une équipe multidisciplinaire constituée généralement de cinq à dix personnes permettant de répondre aux exigences de réalisation des fonctionnalités du projet. Sa particularité est qu'elle s'autogère et organise elle-même son travail.
4. **Stakeholders :** ce sont toutes les personnes curieuses, intéressées par le développement du projet. Ils ne font pas partie de l'équipe, c'est peut-être quelqu'un qui a déjà traité des cas similaires ou tout simplement un membre de l'équipe du bureau d'à côté qui se tient au courant de ce qui se développe dans son entreprise. Ils peuvent assister aux discussions et donner leur opinion.
5. **Users :** futurs utilisateurs des fonctionnalités à développer. Il est important d'avoir leur point de vue avant, pendant et en fin de développement.

FDD et **RUP** ont une vision plus technique, moins orientée autour du client et des utilisateurs même si cet aspect est pris en compte.

Voici les rôles des acteurs d'un projet **FDD** [22] **:**

1. **Expert du domaine :** utilisateurs, client ou sponsors. C'est un regroupement des personnes maîtrisant le domaine impacté par le projet.
2. **Chef de projet :** gère et maintient le projet comme un système pour qu'il fonctionne au mieux.
3. **Architecte en Chef :** responsable du modèle global du projet. Il permet aussi de faciliter la modélisation en organisant des sessions collaboratives afin d'étendre les compétences dans l'équipe.
4. **Directeur de développement :** en charge de la gestion des ressources, il doit les assister dans leur avancement et résoudre au quotidien les conflits au sein de l'équipe.
5. **Programmeur en Chef :** très expérimenté, il participe à l'analyse, la modélisation, au cycle de développement des fonctionnalités.
6. **Propriétaire de classes :** membre de l'équipe de développement, il doit participer à l'analyse mais surtout coder, tester et documenter les fonctionnalités prises en charge.

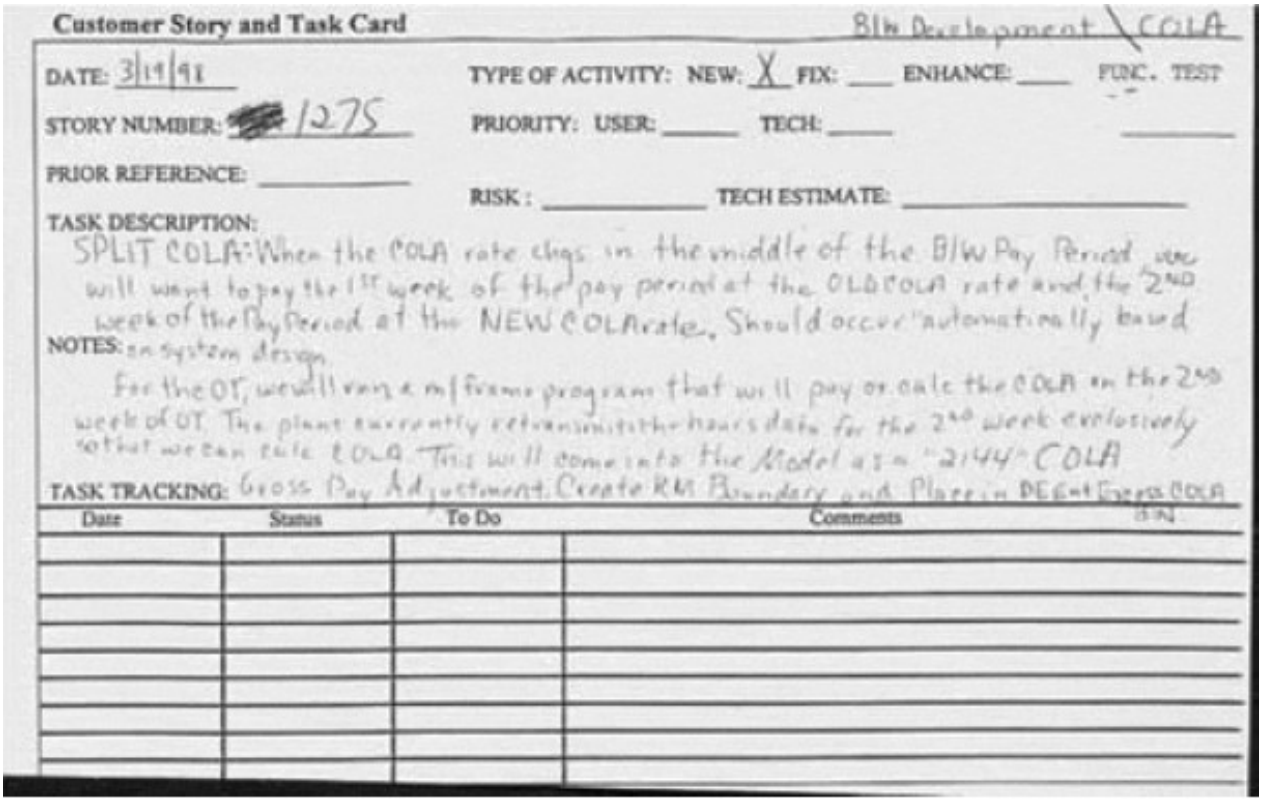
Avec en soutien :

7. **Responsable des releases** (versions du logiciel) **:** assiste le chef de projet et le directeur de développent sur les tâches administratives comme la planification des réunions ou l'enregistrement des résultats. Il prend en quelque sorte le rôle de Scrum Master dans SCRUM ou de manager dans XP.
8. **GURU du langage de programmation :** propose des solutions techniques en cas de besoin.
9. **Ingénieur de Build :** aide à la compilation et au déploiement du logiciel.
10. **Administrateur Système :** présent pour la configuration système.

RUP [21] détaille beaucoup plus les rôles et les acteurs du projet que **FDD**. Il ne spécifie pas un acteur qui code, teste et documente. Il propose par exemple pour les tests, un testeur d'intégration, un testeur de performances et un testeur système. Pour la conception la même vision de découpage est proposée avec la séparation de la conception générale, de la conception de base de données, d'interface utilisateur, de tests et la conception métier. Suivant le besoin, l'affinage des rôles peut amener jusqu'à 27 acteurs distincts. Cette vision devient complexe à mettre en œuvre et à gérer mais peut être utile sur un projet de grande ampleur.

Des artefacts

En plus de la définition de rôles, un certain nombre d'artefacts entrent en jeu tout au long du projet. Pour **XP** le principal artefact est l'application elle-même, développée au plus simple et auto-documentée par l'intermédiaire des commentaires directement dans le code source. Les autres documents vraiment importants sont les fiches scénarios descriptives du comportement de chaque fonctionnalité appelées « *user stories* » illustrées en figure 12.


Customer Story and Task Card

B/W Development / COLA

DATE: 3/19/98

TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST

STORY NUMBER: 1275

PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___

RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:

SPLIT COLA: When the COLA rate chgs in the middle of the B/W Pay Period we will want to pay the 1st week of the pay period at the OLD COLA rate and the 2nd week of the Pay Period at the NEW COLA rate. Should occur automatically based on system design.

NOTES:

For the OT, we will run a mainframe program that will pay or calc the COLA on the 2nd week of OT. The plant currently retransmits the hours data for the 2nd week exclusively so that we can calc COLA. This will come into the Model as a "2144" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DEV w/ Split COLA

Date	Status	To Do	Comments	BN

Figure 12 : Exemple de user Story [7]

Leur format est court afin de pousser les rédacteurs à être le plus synthétique possible. Les « *user stories* » sont utilisées à la fois pour la rédaction des tests et pour le recettage. Elles contiennent tous les champs nécessaires à la compréhension de la fonctionnalité à développer rédigés de manière textuelle. Elles sont aussi évaluées en fonction de leurs priorités et de leurs risques [23].

SCRUM est plus complète à ce niveau là. Elle détaille la gestion organisationnelle XP ; cinq artefacts entrent en jeu [25] :

1. **Product Backlog** (Backlog du produit) qui est une liste priorisée de fonctionnalités souhaitées par le Product Owner. Chaque fonctionnalité est décrite, comme pour XP. Elle correspond à une *« user story ».*
2. **Product Backlog Burndown** montre l'avancement des fonctionnalités traitées du Product Backlog sous forme de graphique.
3. ***Sprint* Backlog :** fonctionnalités du Product Backlog affectées à un *sprint* (itération).
4. ***Sprint* Backlog Burndown** montre l'avancement des fonctionnalités traitées du *sprint* backlog sous forme de graphique.
5. **Impediment List :** liste des fonctionnalités n'ayant pu être traitées par manque d'information ou d'outils. Le Scrum Master doit organiser son équipe afin de résoudre cet obstacle rapidement.

Un sixième artefact peut être la somme de tous les rapports et statistiques utiles à l'avancement du projet ou d'un prochain *sprint* tels qu'un historique des « bugs » ou des versions du projet [25].

Comme pour les acteurs, les artefacts **FDD** et **RUP** sont plutôt orientés sur la modélisation UML, la conception et la réalisation. Mise à part la liste de fonctionnalités explicitées par le client expert du domaine, les artefacts sont centrés sur le modèle de cas d'utilisation, les spécifications, la conception (classes, paquetages etc.) [22]. **RUP** offre la possibilité d'être très complet en termes de documentation et livrables. Il classe donc ses artefacts selon cinq ensembles d'informations qui sont les exigences (vision métier), la conception (architecture, vision logicielle), l'implémentation, le déploiement et la gestion (rentabilité, planning) [21].

Des activités quotidiennes

Chaque acteur et artefact a son rôle dans le cycle de vie de chacune des méthodes. Les processus sont spécifiques et impactent le quotidien de l'équipe. Le plus démonstratif de la philosophie Agile est **XP@Scrum** (modèle en figure 8 de la partie 1.1.2). Il permet de traiter avec un grand niveau d'exigence les critères du manifeste Agile sans ajout superflu.

Une fois le projet lancé, une succession d'étapes est enchaînée jusqu'à la livraison du produit complet comprenant tous les livrables. La première étape est la réalisation du Product Backlog afin d'avoir le périmètre du projet. Ensuite une réunion de planification de *sprint* est nécessaire afin de lister les fonctionnalités à inclure dans celui-ci. Cette réunion est planifiée par le Scrum Master, toute l'équipe y participe pour apporter son point de vue, chiffrer et améliorer la communication. L'itération est la phase suivante : le *sprint* peut commencer. Chaque journée commence par un Daily Scrum ou mêlée quotidienne. Cette petite réunion consiste pour chacun des membres de l'équipe à donner trois informations : ce que l'on a fait depuis le dernier Daily Scrum, ce que l'on va faire jusqu'au prochain et si des problèmes ont été rencontrés. Si ce n'est pas fait durant la journée, le s*print* backlog est mis à jour ainsi que le *sprint* backlog Burndown. S'il n'y a aucun point bloquant, chaque personne s'associe dans le *sprint* backlog, il précise la tâche qu'il développera jusqu'au prochain Daily Scrum ou s'il continue son activité en cours si celle-ci n'est pas terminée [25].

XP impacte fortement cette phase de développement, des contraintes sur la façon de travailler sont à mettre en œuvre.

1. Les développements sont à faire en binôme, pratique appelée « pair programming ». Le premier tient le clavier et écrit le code. Le second est là pour assister, aider et suggérer de nouvelles possibilités ou soulever certains problèmes. Cette solution permet une revue de code en continu et privilégie la communication et le retour d'information. Les binômes permutent fréquemment.
2. L'équipe doit avoir un espace de travail commun y compris avec le client, un bureau commun afin de faciliter la communication orale face à face et le travail en binôme.
3. L'intégration continue est aussi une pratique à adopter, l'équipe entretient une version testée et opérationnelle du système qui contient les toutes dernières modifications. Cette pratique assure également un retour rapide d'information.
4. Les tests unitaires doivent êtres rédigés avant le développement. Les tests fonctionnels doivent être rédigés par une autre personne que celle qui développe. Ces pratiques permettent de vérifier la cohérence entre les rédactions des tests et le développent, donc d'être sûr que la *« user story »* est claire.
5. Conception incrémentale, simple : il ne faut implémenter que ce qui doit être implémenté, pas d'anticipation sur les futures fonctionnalités. La documentation aussi doit être simple, minimale pour une vision évolutive.
6. Code partagé : chacun des membres peut apporter des modifications dans tous les développements. L'amélioration continue ou le « refactoring » du code est une priorité.
7. Une convention de nommage et des normes sont à respecter à tous les niveaux, que ce soit pour la documentation ou le code [7].

Une fois la journée passée, le prochain Daily Scrum a lieu. Soit tout va bien et le processus de développement continue, soit il y a un problème soulevé par un des membres qu'il faut résoudre. Différents types de problèmes peuvent ressortir : ce peut être le Product Owner qui signale qu'un développement ne correspond pas à ses attentes ; il faut le remanier. Dans ce cas là, cette fonctionnalité passe dans la liste des développements du *sprint* suivant. Un autre genre de point à résoudre est le développement d'une fonctionnalité qui pose problème. Une discussion au sein de l'équipe, des conseils ou même une aide supplémentaire sur le développement pour une durée bien

définie, « timeboxée » d'une heure par exemple peut débloquer la situation. Si ce soutien n'est pas suffisant, la fonctionnalité ira dans la liste des fonctionnalités n'ayant pu être traitées, et ce sera au Scrum Master de trouver rapidement une façon d'avoir la solution, de ressortir cette tâche de la liste, par une aide externe à l'équipe par exemple. Ce cycle de réalisation continue jusqu'à la fin du *sprint* (deux à quatre semaines). Puis une fois terminée, chaque fonctionnalité est présentée au Product Owner, aux Stakeholders et à un représentant des utilisateurs lors d'une démonstration planifiée à l'avance par le Scrum Master. Grâce à l'intégration continue, le produit doit pouvoir être livré et être utilisable par le client. Tous les points remontés sont listés dans le Backlog du *sprint* suivant ainsi que les fonctionnalités n'ayant pu être traitées dans ce *sprint* s'il y en a. Une rétrospective du *sprint* achevé est organisée pour améliorer la gestion du prochain et se situer dans l'avancement du projet. Chacun donne son avis, la communication est mise en avant et chaque point de vue compte. Le Backlog du produit peut maintenant être à son tour mis à jour pour être prêt pour la prochaine itération jusqu'à la dernière qui signifiera la fin du projet et la livraison finale du produit comprenant tous les livrables [25].

Un schéma synthétique récapitulatif est disponible ci-dessous en figure 13.

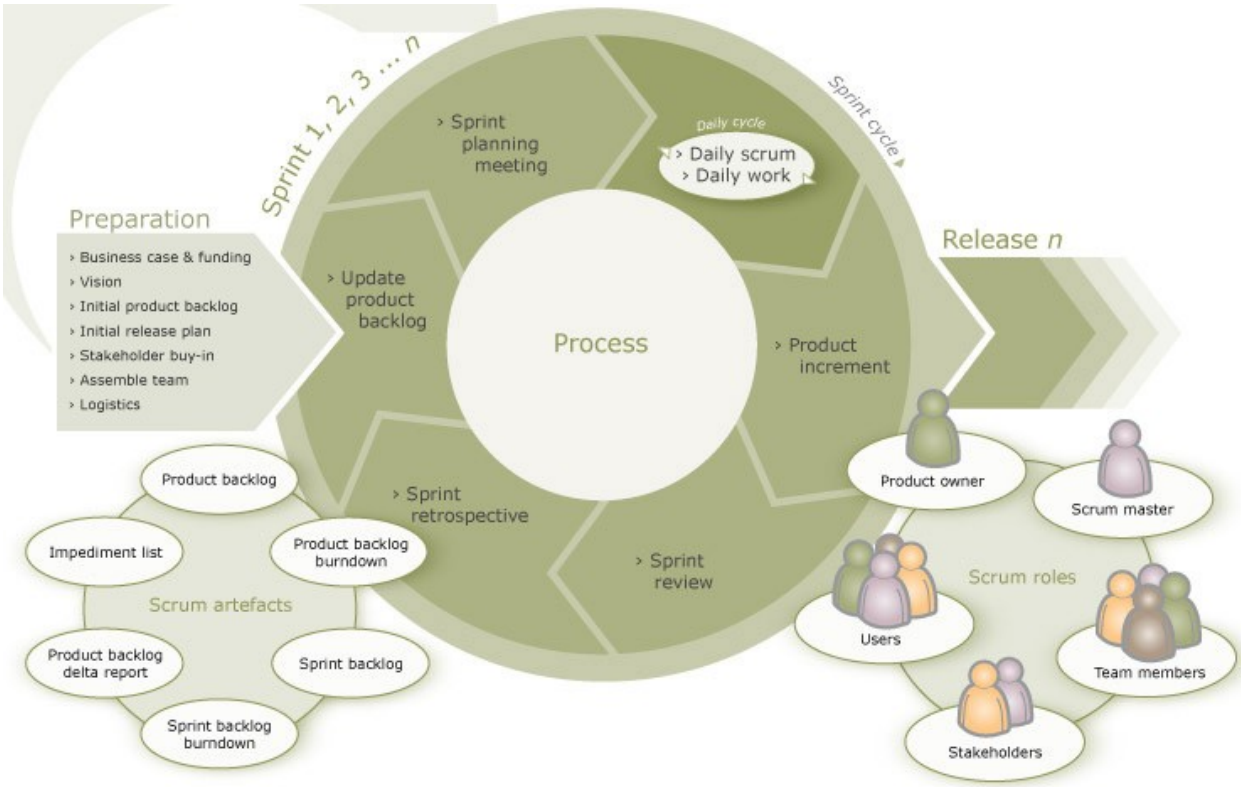


Figure 13 : Processus de développement SCRUM [25]

Même si la philosophie et les idées générales sont les mêmes, les processus **FDD** et **RUP** sont plus stricts, la communication est moins mise en avant, la réunion quotidienne du matin n'est pas pratiquée. Les processus sont eux centrés sur l'architecture et la modélisation UML.

FDD se déroule en cinq étapes, une première qui consiste à donner une vision globale, cerner le périmètre du projet afin de pouvoir créer un modèle global UML. Ce modèle est ensuite découpé en domaines pour que chacun soit revu et discuté. Les acteurs concernés sont les experts du domaine, les architectes en chef ainsi que les programmeurs en chef. Tout est regroupé à la fin de cette phase pour passer à la suivante qui est l'élaboration de la liste de domaines à traiter. Chaque domaine ressorti se découpe en activités métier qui donnent elles-mêmes naissance à des fonctionnalités spécifiées sous la forme <action> <résultat> <objet> : par exemple, calculer le nombre d'employés d'une société. Ce sont toujours les mêmes acteurs en charge de cette deuxième phase. La troisième

partie du processus consiste à prioriser, planifier le développement et affecter les fonctionnalités. Les acteurs de cette étape sont le chef de projet, le directeur de développement et les programmeurs en chef à qui seront affectées les fonctionnalités sous forme de classes. C'est maintenant que la phase itérative arrive. Elle se découpe en deux parties. La première concerne les programmeurs en chef et les futurs propriétaires de classes pour concevoir les diagrammes de séquences, raffiner le modèle global mais aussi écrire le prologue des classes et méthodes. La seconde est de passer à la réalisation, ce sont les propriétaires de classes qui sont chargés de l'implémentation, de l'inspection du code, des tests unitaires et de la génération des livrables. Cette itération est renouvelée tant qu'il reste des fonctionnalités [22].

RUP quant à lui, est beaucoup plus complexe. Il ne propose pas de modèle global UML en début de projet mais il propose une itération de chaque phase permettant de remettre en question et de rediscuter chaque fonctionnalité. Chaque phase met une priorité sur un processus, les premières sur les besoins, les suivantes sur la conception, ensuite la réalisation puis les tests. Cette vision permet de facilement revenir sur des fonctionnalités à améliorer, de gérer plus facilement les risques et les exigences. **RUP** est très structuré pour une méthode Agile. Il offre donc la possibilité de gestion de grosses équipes, mais un défaut important est le manque d'une fabrication rapide d'un prototype. Le fait d'itérer chaque phase, notamment la conception, ne permet pas d'avoir un résultat démontrable rapide du produit qui peut être primordial lors d'un appel d'offre qui demande une forte réactivité. Cette méthode s'adapte plutôt à un projet en interne ou le demandeur et le réalisateur sont confondus [21].

Des outils

Le choix de la méthode Agile à adopter est directement orienté par le niveau d'abstraction et de liberté que nous voulons sur le projet. Néanmoins, la disponibilité et le choix des outils restent une part importante dans le déroulement de celui-ci. C'est au quotidien que ces logiciels seront utilisés. Il est donc primordial qu'ils soient intuitifs, simples et rapides d'utilisation au quotidien mais respectant les contraintes de la méthode et les possibilités budgétaires de l'entreprise qui sont souvent restreintes.

XP ne requiert pas de logiciel spécifique. Les fonctionnalités, leur description et les membres affectés peuvent être gérés sous Excel. La mise en place d'une gestion de configuration commune est bien sûr nécessaire mais celle-ci est en générale déjà possible au sein de l'entreprise.

SCRUM, un peu plus contraignant que **XP** sur les pratiques de management, offre plusieurs logiciels de gestion de projets. Il y a trois possibilités, la première est l'outillage minimum, un fichier Excel s'adaptant exactement au projet peut être créé. Par contre la création des courbes et autres calculs ou automatismes nécessitent des connaissances afin de personnaliser par des fonctions le fichier Excel. En complément de ce fichier, un récapitulatif de l'avancement et de l'affectation des tâches à l'aide de « post-its », sur un mur du bureau de l'équipe comme le montre la figure 14 permet une vision et une communication rapide, simple et efficace. Cette pratique est très souvent utilisée mais il faut bien sûr être rigoureux et ne pas délaisser l'un par rapport à l'autre et bien mettre les deux suivis à jour. Cette pratique est peu coûteuse et permet vraiment d'adapter le fichier au contexte du projet et de l'entreprise.

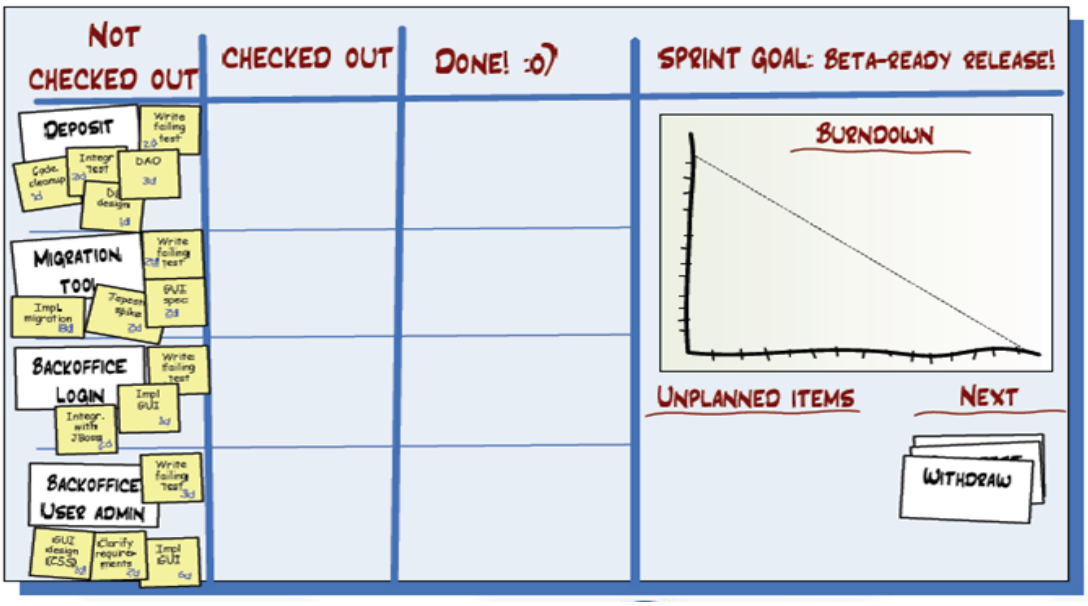


Figure 14 : Sprint backlog présenté sous forme de « post-its » [26]

Une deuxième vision est possible mais plus onéreuse, c'est l'utilisation d'un logiciel commercial comme ScrumWorks, complimenté par Sun Microsystems pour son intuitivité [29]. Cet outil de gestion **SCRUM** est plus complet, la description des *sprints* et des *user stories* est plus détaillée que dans IceScrum. La version pro est d'environ 2000 € pour une équipe de dix personnes sur un an [28]. Cette solution est intéressante pour un projet complexe et de longue durée.

RUP et **FDD** possèdent eux aussi leurs solutions logicielles payantes. Tout d'abord **RUP**, créé par Rational Software, filiale d'IBM possède son outil IBM Rational Method Composer permettant la gestion des processus et des bonnes pratiques de **RUP**. Son coût peut varier suivant les accords et contrats que l'entreprise possède avec IBM. Son point fort est la possibilité d'intégration ou de migration vers d'autres produits Rational comme Rational Software Architect permettant la modélisation UML.

Pour **FDD**, une possibilité est le logiciel commercial, xProcess d'Ivis. Très complet, il permet de gérer efficacement un projet. Son défaut du coût d'achat est maintenant éliminé car il vient de rejoindre le côté des logiciels libres et s'appelle désormais OpenxProcess. La société veut augmenter sa communauté, son nombre d'adhérents. Elle propose pour ses utilisateurs des formations et certifications afin de se servir efficacement d'OpenxProcess. Même si le coût d'achat devient nul, étant très complet et ouvert à plusieurs méthodes de gestion de projets, sa complexité n'est pas négligeable, il est peut être judicieux de prévoir un coût de formation aux utilisateurs [33].

Ce qui amène à une dernière solution : le logiciel libre. Cette catégorie d'outils est assez riche, elle est disponible pour l'ensemble des méthodes.

FDD offre plusieurs types de solutions avec différents niveaux de complexité. Le plus simple d'utilisation et gratuit est FDD Tools qui fournit une petite partie de la méthodologie **FDD**. Il permet de créer les fonctionnalités, de les planifier mais pas de les prendre d'une liste pour les insérer dans une itération ce qui est dommage car cette phase itérative est l'intérêt de cette méthode [31].

Pour être plus complet dans l'utilisation de la méthode **FDD** et toujours en logiciel libre, FDD Project Management Application permet un niveau de détail plus élaboré que FDD Tools comme l'association des fonctionnalités de la liste globale à une itération. Nous retrouvons aussi une

planification plus exhaustive que dans FDD Tools avec la possibilité de rentrer les dates, la planification des phases pour chacune des fonctionnalités. La visualisation de rapports sous forme HTML ou PDF est aussi incluse, ce qui permet de bien situer et documenter l'avancement du projet [32].

Un dernier exemple celui-ci pour la méthode **SCRUM** est IceScrum développé par des étudiants encadrés par une équipe de développeurs. Cet applicatif est open-sources, régulièrement mis à jour et son utilisation très intuitive nous fait oublier quelques petits bugs sans importance. Il correspond très bien au processus **SCRUM** comme l'illustre la figure 15 [27].

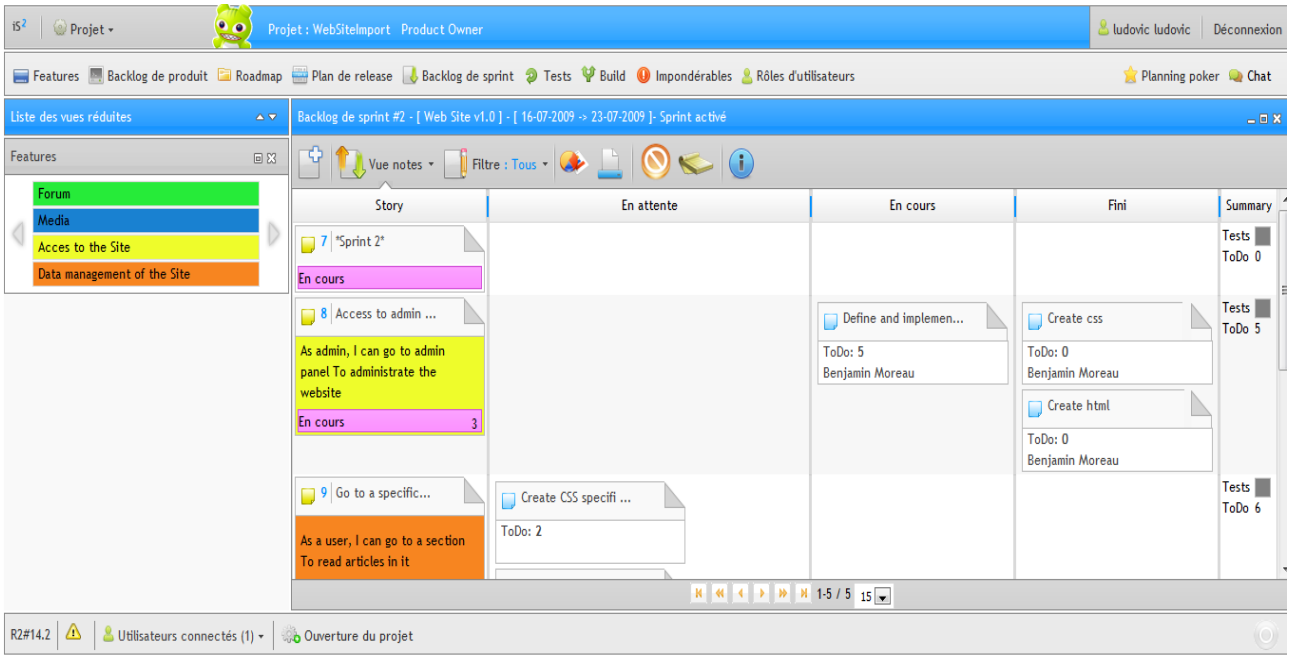


Figure 15 : Exemple IceScrum [27]

Il propose la création de « *feature*s » puis des *user stories* associées. L'organisation itérative est gérée sous forme de *sprints* embarquant les *user stories* prioritaires (figure 15). La gestion des tests et des impondérables est aussi proposée. Cet outil complet est intuitif, ergonomique et libre.

Pour chacune de ces méthodes, des possibilités peu coûteuses et souvent complètes sont offertes, il est donc possible de mettre en place une méthode de gestion de projets Agile avec peu d'investissement budgétaire. L'investissement à faire et de mettre en place la disponibilité des outils aux membres de l'équipe, qu'ils soient intuitifs et surtout leur programmer une formation d'une durée plus ou moins longue afin qu'ils soient opérationnels et que leur seule préoccupation soit le projet uniquement.

Une synthèse en tableau 1 compare la pertinence des méthodes Agile présentées : la fusion XP@SCRUM, FDD ainsi que RUP.

Tableau 1 : Comparatif des méthodes de gestion de projets Agile pertinentes

	XP@SCRUM	FDD	RUP
Cycle de vie	**Livraison itérative de lots opérationnels.**	**Livraison itérative de lots opérationnels.**	**Livraison itérative de lots opérationnels.**
Acteurs, rôles et artefacts	**Communication orale au sein d'une petite équipe mettant le client au centre du projet. La rédaction de spécifications se fait au fil des sprints. Un ensemble de bonnes pratiques permet un travail de qualité.**	**Communication orale au sein d'une petite équipe mettant le client au centre du projet. Une première version UML est spécifiée en début de projet afin d'avoir une base de travail. La rédaction de spécifications se fait au fil des sprints.**	**Nécessité d'une grosse équipe pour développer un projet de grande envergure durant une longue période.**
Activités quotidiennes	**Communication et entraide.**	**Communication et entraide**	**Communication et entraide difficile au sein d'une grosse équipe.**
Outils	**Intuitif, libre et simple à mettre en place.**	**Assez intuitif, libre et simple à mettre en place.**	**Coûteux et complexe.**
Vision Agile	**Très respectée.**	**La modélisation UML en début de projet ne fait pas partie des principes Agile.**	**Très structuré pour une méthode Agile.**

Très bon Bon Prudence

L'intérêt de ce mémoire est de mettre en place une gestion de projets Agile, au sein d'une petite équipe de quatre personnes. L'objectif est de privilégier la communication et le travail en équipe. Les méthodes présentées ne sont pas toutes les plus adéquates au projet. **FDD** se base sur une modélisation UML ce qui s'éloigne de l'objectif principal de cette gestion par la communication. **RUP** ne correspond pas non plus aux critères car c'est une méthode qui s'applique dans le cadre d'une grosse équipe. Il reste donc deux méthodes très orientées Agile, **XP** et **SCRUM**. Etant donné que ces deux méthodes sont complémentaires, la solution la plus évidente est la gestion Agile **XP@SCRUM**. La mise en place de cette méthode se fera par un outil simple, intuitif, libre et la visualisation à l'aide de post-its. C'est donc grâce à un tableau de post-its et du Logiciel open-source le plus intuitif, IceScrum que la gestion Agile **XP@SCRUM** est ici la plus cohérente.

1.2. Patrons collaboratifs

Après un état des lieux sur les possibilités de gestion de projets Agile et les moyens nécessaires pour leur mise en œuvre, il faut maintenant se pencher sur le cœur de l'outil de gestion de patrons collaboratifs. Ce stage étant effectué au sein de l'équipe SIGMA du LIG, équipe spécialisée en modélisation adaptable, il n'est pas difficile de se renseigner sur les possibilités actuelles en termes de modélisation sous forme de patrons. Le travail de recherche sur les technologies existantes liées à l'ingénierie de « patrons » ici réalisé a principalement été basé sur la forte expertise et connaissance de l'équipe.

Le besoin actuel pour de nombreux secteurs comme l'ingénierie des systèmes d'information est d'augmenter la performance de leurs processus et d'améliorer la qualité de leurs produits. C'est de là qu'apparaît une nouvelle forme d'ingénierie : la conception par composants définis comme une solution testée et acceptée pour résoudre un problème fréquemment rencontré.
Pour intégrer la réutilisation dans tout le processus de développement d'applications de natures et de technologies diverses, une grande variété de modèles réutilisables a d'ores et déjà été proposée comme les objets métiers, les frameworks ou les patrons. Les patrons représentant le cœur de l'outil de gestion souhaité, voient le jour, principalement depuis des travaux orientés architecture de C. Alexander dans les années 1970, puis formalisés pour la conception de logiciels en 1995 dans le livre du « Gang of Four » (GoF) de E. Gamma, R. Helm, R. Johnson et J. Vlissides. Différents types font leur apparition notamment dans les années 1990 comme les patrons de conception (design patterns) du Gof, les patrons d'analyse de P. Coad ou les patrons processus de S. W. Ambler [34] [35] [36] [37].

1.2.1. Philosophie Patron

De manière générale, un patron décrit un **problème** fréquemment rencontré dans un **contexte** particulier ainsi qu'une **solution** générale et consensuelle pour résoudre ce problème [2]. Voici les principaux aspects d'un patron :

- Un patron constitue une base de savoir-faire pour résoudre un problème récurrent dans un contexte applicatif ou technologique particulier. Les patrons existent à tous les niveaux du cycle de vie d'une application : ainsi, il existe des patrons d'analyse, de conception et d'implantation [2].

- Un patron peut être un patron produit qui capitalise des spécifications ou des implantations d'un but à atteindre ou un patron processus, qui capitalise des spécifications ou des implantations d'une démarche à suivre pour atteindre le résultat [2].

- Les patrons sont généralement organisés en catalogues de patrons. Un catalogue définit une collection de patrons avec des règles permettant de les combiner. Chaque catalogue de patrons utilise son propre formalisme de représentation constitué de différentes rubriques [2].

Les solutions proposées par les patrons sont exprimées sous la forme de spécifications semi-formelles de niveau conceptuel qu'il s'agit d'adapter. L'utilisation d'un patron se fait par imitation du patron : duplication de la solution, puis adaptation de cette solution au contexte. Dans le cadre de la conception d'une application, il s'agit ensuite d'intégrer le résultat d'une ou de plusieurs imitations de patrons afin de disposer d'un modèle cohérent du système [2].

Il nécessite deux principaux acteurs : le premier que l'on peut appeler l'ingénieur de patrons qui a pour but de créer et initialiser le patron, et le second que l'on nommera ingénieur d'applications qui a pour but d'imiter le patron, de l'adapter à un contexte qui lui est propre et apporter son retour d'expérience [2].

Des nombreux travaux pour structurer les patrons apparaissent grâce à certaines pointures du domaine comme E. Gamma qui définit un standard de référence en termes de patrons produit suivant un formalisme comportant 14 rubriques. S.W. Ambler quant à lui, documente ses patrons processus en 8 rubriques simples et compréhensibles. Mais ces langages ont leurs faiblesses, des descriptions en langue naturelle manquant de structure, la réutilisabilité n'est pas évidente et les relations inter-patrons quasi inexistantes [34].

Une illustration très répandue des patrons est l'architecture MVC (Modèle-Vue-Contrôleur), utilisée pour cadrer une structure d'interfaces d'applications. Cette architecture organise l'application en un modèle (modèle de données), une vue (présentation, interface utilisateur) et un contrôleur (logique de contrôle, gestion des événements, synchronisation), chacun ayant un rôle précis dans l'interface. Voici un schéma en figure 16 illustrant la solution MVC puis en figure 17 un schéma de cas d'utilisation Ash.MVC proposé par Ashwini Kumar Rath [38] [39].

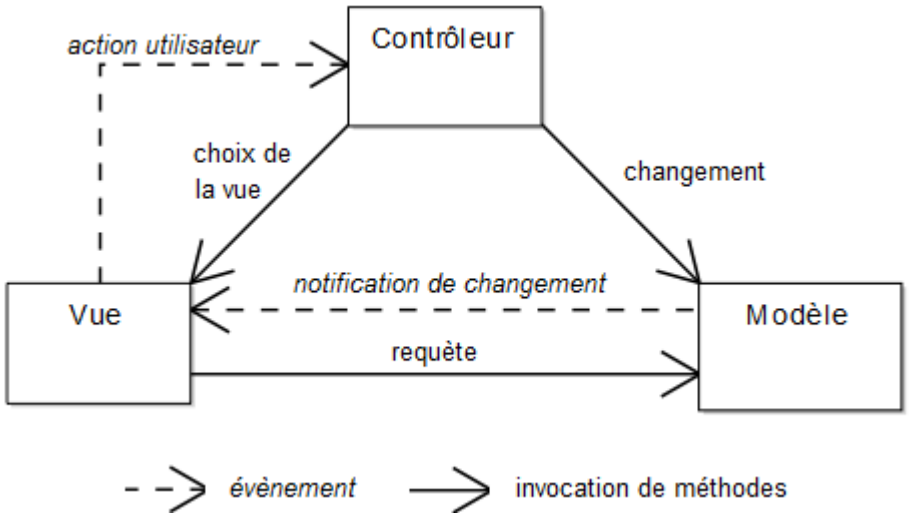


Figure 16 : Architecture MVC (Solution) [38]

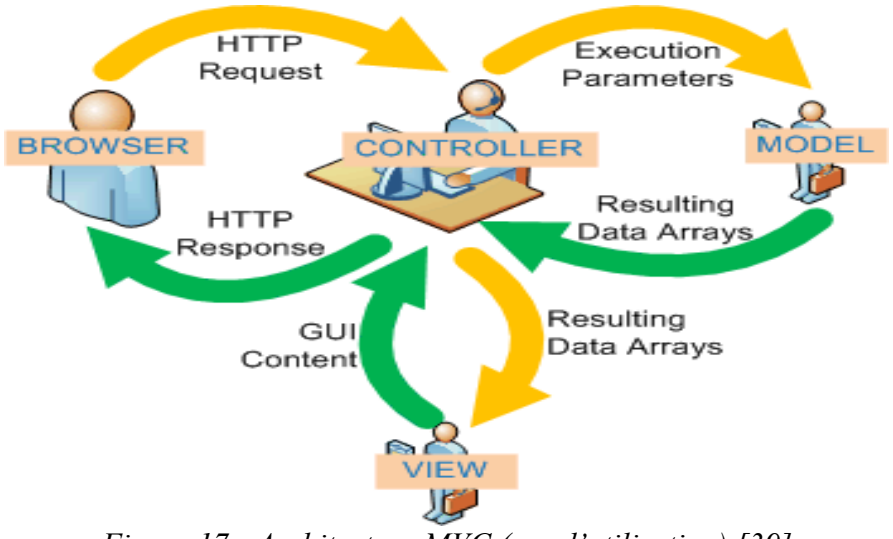


Figure 17 : Architecture MVC (cas d'utilisation) [39]

Pour représenter un patron, de nombreux formalismes ont été proposés. Nous entendons par formalisme la structure adoptée par le concepteur de patrons pour représenter des patrons. Chaque formalisme s'articule autour d'un triplet {problème, solution, contexte} mais deux classes se distinguent : les formalismes narratifs tels que les patrons d'Alexander, et les formalismes structurés pour les patrons d'analyse, de conception, ou de processus. Par opposition aux formalismes narratifs par nature peu structurés et informels, le second type de formalisme offre une meilleure représentation des patrons.

Ces formalismes structurés, composés chacun d'un ensemble de rubriques qui leur est propre, sont globalement équivalents dans la mesure où ils expriment tous le triplet {problème, solution, contexte}. Ils diffèrent cependant par le nombre de rubriques proposées et le degré de détail de celles-ci. Les patrons existants à l'heure actuelle sont de plus en plus nombreux et de plus en plus variés [34].

Le formalisme P-SIGMA proposé par l'équipe SIGMA du LIG de Grenoble plus complet que les précédents en termes de rubriques est très intéressant : il permet d'uniformiser la représentation des différents types de patrons, d'améliorer l'interface de sélection d'un patron ou d'un ensemble de patrons. Il offre aussi la possibilité de préciser leur utilisation et d'exprimer les relations inter-patrons [34] [35].

Ce formalisme est très bien conçu, il permet de modéliser une majorité de cas avec le minimum de champs afin d'être le plus précis possible. Voici la définition de chacune des rubriques en tableau 2 afin d'illustrer le contenu d'un patron utilisant ce formalisme.

Tableau 2 : Contenu d'un patron avec P-Sigma [34]

Interface	
Identifiant	Définit le couple problème/solution à partir duquel le patron pourra être référencé. Constitue la clé principale de communication entre le concepteur de patrons et les utilisateurs. *Champ :* ▪ Un champ textuel *Exemple :* ▪ Architecture MVC
Classification	Définit la fonction du patron par un ensemble de mots-clés du domaine (termes du domaine). Donne intuitivement la classification du domaine. *Champs :* ▪ Un champ textuel ▪ Eventuellement un champ formel (expression logique de mots-clés du domaine) *Exemple :* ▪ Architecture, méthode de conception
Contexte	Décrit la pré-condition pour l'application du patron. Peut être obtenu en appliquant la solution modèle d'un ou de plusieurs patrons ; les noms des patrons correspondants constituent alors le champ formel. *Champs :* ▪ Un champ textuel ▪ Eventuellement un champ formel : {Patron} *Exemple :* ▪ Au sein d'une application logicielle

Problème	Définit le problème résolu par le patron. *Champ :* ▪ Un champ textuel *Exemple :* ▪ Organisation de l'Interface Homme-Machine
Force	Définit les apports induits par l'application du patron. *Champs :* ▪ Un champ textuel ▪ Eventuellement un champ formel : expression logique des critères de qualité associées à une technologie *Exemple :* ▪ Permet une interaction au sein même de l'Interface Homme-Machine
Réalisation	
Solution Démarche	Indique la solution du problème en termes de processus à suivre. Un diagramme d'activités permet éventuellement de représenter la démarche sous la forme : [garde] Patron. *Champs :* ▪ Un champ textuel ▪ Eventuellement un champ formel de type diagramme d'activités *Exemple :* ▪ Diagramme d'activité décrivant la mise en place de l'architecture
Solution Modèle	Décrit la solution en termes de produits attendus après l'application du patron. *Champs :* ▪ Un champ textuel ▪ Un champ de type diagramme de classes ▪ Eventuellement un champ de type {diagramme de séquence} *Exemple :* ▪ Diagramme de séquence précisant les dialogues entre les trois tiers composant l'Interface Homme-Machine (en Image)
Cas d'application	Décrit des exemples d'imitation de la Solution Modèle, optionnelle mais fortement conseillée pour faciliter la compréhension de la solution du patron. *Champs :* ▪ Un champ textuel ▪ Un champ de type diagramme de classes ▪ Eventuellement un champ de type {diagramme de séquence} *Exemple :* ▪ Mise en place d'une application logicielle
Conséquence	Présente les limites et les bénéfices de l'application de la solution. Peut inclure un nouvel ensemble de problèmes faisant apparaître la nécessité d'appliquer de nouveaux patrons. *Champ :* ▪ Un champ textuel *Exemple :* ▪ Le dialogue au sein de l'Interface Home-Machine est normé, il faut le respecter

Relation	
Utilise	Si un patron P1 utilise un patron P2, alors : ▪ La solution démarche de P1 doit être exprimée en utilisant le patron P2. ▪ La classification de P2 peut être enrichie par rapport à celle de P1 : de nouveaux mot clés sont éventuellement ajoutés dans la classification de P2. ▪ Le contexte de P2 peut être enrichi par rapport à celui de P1.
Requiert	Si un patron P1 requiert un patron P2, alors : ▪ L'application de P2 doit être un pré-requis à l'application de P1. ▪ P2 doit apparaître dans le contexte de P1.
Alternative	Un patron P1 est une alternative d'un patron P2 si les deux patrons se différencient par leur force qui justifie deux solutions différentes au même problème : ▪ P1 et P2 ont la même classification, le même contexte et le même problème. ▪ Seule la rubrique « Force » des deux patrons est différente.
Raffine	Si un patron P1 raffine un patron P2, alors : ▪ Le problème de P1 doit être une spécialisation de celui de P2. ▪ La classification de P1 peut être enrichie par rapport à celle de P2. ▪ La force de P1 peut être enrichie par rapport à celle de P2. ▪ Le contexte de P1 peut être enrichi par rapport à celui de P2.

Pour permettre une uniformisation inter-patrons, une cohérence relationnelle entre patrons et une logique, il est indispensable de regrouper les patrons sous forme de systèmes. C'est là que la notion de catalogue de patrons fait son apparition. Il permet une interaction entre eux selon un certain nombre de principes, de règles communes, notamment du formalisme [34].

En complément de ce langage, un aspect reste à associer au patron : la collaboration. Faire interagir plusieurs personnes autour d'un patron, rendre ce patron collaboratif permettrait de le voir évoluer plus rapidement et plus efficacement.

Un tel patron collaboratif, nécessaire pour notre projet, doit respecter deux critères :

- Il doit être discuté, complété en collaboration au sein d'un groupe et validé une fois abouti.
- Des cas d'utilisations, des retours d'expérience doivent venir l'enrichir, le valoriser ou peut être même le remettre en question.

1.2.2. Outils existants

La vision du travail à base de patrons est de plus en plus recherchée et de nouveaux environnements de travail voient le jour. Il reste difficile actuellement de trouver des solutions permettant la gestion de patrons collaboratifs.

Des prototypes existent comme Pattern Tool issu de travaux de l'université d'Utrecht en 1996 ou par la suite FACE en 1997. Ils ne permettent pas la traçabilité ou la sélection de patrons à l'intérieur

de catalogues. Ils se concentrent essentiellement sur la représentation objet et l'instanciation des solutions.

Des outils commerciaux sont disponibles comme Framework Studio fonctionnant uniquement avec le modeleur Rational Rose laissant peu d'initiatives à l'utilisateur ou Objecteering proposant une dizaine de patrons des vingt-trois de Gamma ne permet pas d'ajouter des rubriques directement. Il propose simplement de les lier en tant que page HTML. Outre leur aspect commercial, ils sont souvent figés, il n'est pas possible d'ajouter des patrons autres que les patrons prédéfinis et la combinaison des imitations de patrons n'est pas prise en compte [36].

Mais d'autres outils existent, notamment AGAP, un atelier créé par l'équipe SIGMA du LIG de Grenoble donc bien maîtrisé par l'équipe. Il offre la possibilité d'avoir des catalogues de patrons, de spécifier pour chaque catalogue son formalisme (par exemple P-Sigma), la création de patrons et leur définition suivant le formalisme précisé par le catalogue ainsi que leur imitation [2]. Les critères principaux d'un patron sont ici respectés.

Cet atelier est structuré en trois composants :

- Un « Formalisme » permettant de définir un langage et ses rubriques pour un catalogue ou système de patrons.
- Un « Système de patrons » composé de patrons conformes au formalisme qui lui est associé.
- Un « Système d'information » qui permet l'instanciation de patrons par imitation.

Il est aussi caractérisé par deux rôles pour deux acteurs :

- Un ingénieur de patrons qui crée les systèmes, les formalismes et les patrons.
- Un ingénieur d'applications qui visualise les systèmes de patrons et imite ses patrons.

Cette modélisation respecte la définition d'un patron, permet de les regrouper à l'intérieur d'un catalogue (Système-Patrons) en les structurant grâce à un langage commun. En se plaçant d'un point de vue patron collaboratif cette solution est moins cohérente. En effet, les deux acteurs ont chacun leur propre rôle sans aucune collaboration possible. Il manque une possibilité de dialogue entre les ingénieurs de patrons et les ingénieurs d'applications. L'aspect collaboratif n'est pas du tout présent, ce qui freine fortement l'évolution du patron et son cycle de vie. Il ne faudrait pas se restreindre à deux acteurs possédants leur propre rôle mais proposer un vrai espace collaboratif où les membres pourraient participer dans chacune des phases du cycle d'un patron.

Les outils existants ne possèdent pas tous la maturité attendue et ne correspondent pas au besoin. Un outil par contre tire son épingle du jeu : l'atelier AGAP conçu par l'équipe SIGMA. Il possède une base très intéressante en termes de modélisation de patrons mais il est nécessaire de mettre en valeur un aspect collaboratif par exemple en lui associant une gestion communautaire. Il servira donc de base de discussion pour concevoir un nouvel outil.

1.3. Espace communautaire personnalisé

La communication autour d'un patron n'est pas évidente, il est nécessaire d'apporter certaines fonctionnalités pour le rendre vivant et le faire évoluer. Ajouter de la collaboration à un patron entraîne d'identifier chaque participant, c'est-à-dire le besoin d'un espace collaboratif avec identification.

Voici les fonctionnalités importantes de l'espace collaboratif souhaité :

- Une gestion par pseudo permettant différents droits et accès sur l'outil.
- Un forum de discussion.
- Un espace de commentaires (exemple : sur un patron pour une pertinence plus directe et plus forte que les messages du forum).

Le développement d'une solution propriétaire capable de fournir ce genre de fonctionnalités, avec de la souplesse et de l'évolution continue n'est pas envisageable dans le cadre de ce projet. Ajouter une couche communautaire à un atelier tel qu'AGAP serait compliqué et avec des normes propres à l'application. Elle ne permettrait pas une évolution continue d'autant plus que le turnover autour de l'outil serait élevé. Pour un gain en temps et en perspective d'évolution, il faut trouver un outil proposant l'espace collaboratif souhaité avec des normes standard. Le choix se porte sur un type de solution particulier : un système de gestion de contenu.

1.3.1. Philosophie des Systèmes de Gestion de Contenu

Un Système de Gestion de Contenu (SGC) plus connus sous le nom anglophone Content Management Systems (CMS) est un logiciel destiné à la conception, à l'administration et à la mise à jour dynamique de sites web ou d'applications multimédias [40].

La solution n'est pas forcément de développer des modules collaboratifs autour du catalogue et de ses patrons mais au contraire, mettre en place un système de patrons au sein d'un espace communautaire adéquat à ce genre de communication. Pour ce faire, différents SGC peuvent correspondre.

En effet un SGC permet, grâce à une gestion de droits, à plusieurs individus de travailler sur un même document (que l'on pourrait assimiler à un patron) appartenant à une communauté (que l'on pourrait assimiler à un catalogue), le contenu est structuré sous forme de documents, blogs ou forums de discussions. Plus techniquement, l'utilisation de workflows ou de versionning est parfois possible.

Un SGC permet une certaine personnalisation ou configuration par interface web, ce qui est intéressant si l'administrateur n'a pas de compétences techniques car il n'as pas besoin d'aller apporter ses modifications directement dans le code.

Pour ce qui est du code ou de la structure de l'outil, il y a une séparation entre contenu et présentation. Le contenu est enregistré dans une base de données alors que la présentation est définie par des gabarits et des feuilles de styles.

Voici les caractéristiques communes aux SGC [40] :

- Utilisation d'interfaces web.
- Séparation entre contenu et présentation.
- Édition de page simplifiée.
- Multiples méthodes de rangement de l'information.
- Gestion des droits.
- Gestion de versions successives et/ou concourantes.
- Travail collaboratif.
- Commentaires devenant eux-mêmes sources d'information.
- Amélioration qualitative continue.
- Rentabilité en termes de coût de développement.

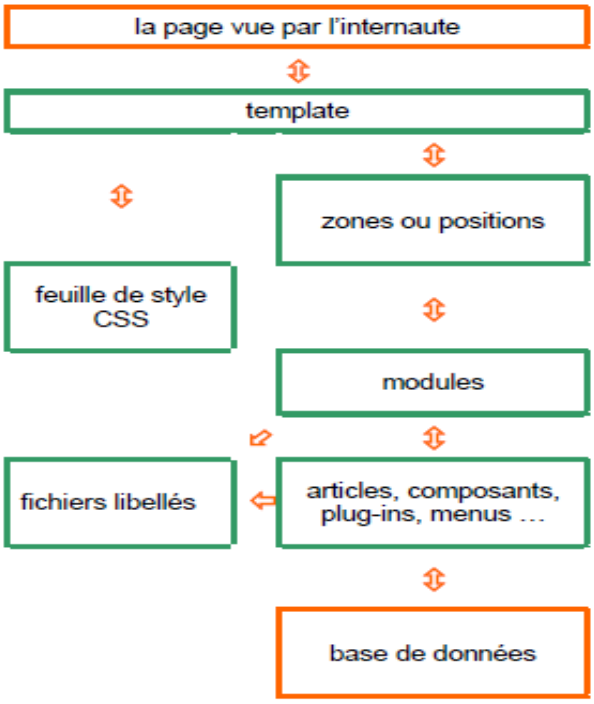
Un SGC, grâce à tous ces points, est un socle incontournable pour la réalisation de l'outil dans notre projet. La puissance de ses fonctionnalités, leur généricité et leurs standards offrent une marge de manœuvre afin de développer un produit qui corresponde le plus aux exigences clientes en partant de fondations existantes, propres et structurées. Pour un gain en temps et qualité, il est nécessaire de partir de ce socle normé, sur une technologie évolutive, manipulable et adaptée aux possibilités de l'équipe.

1.3.2. Outils existants

Certains SGC se rapprochent plus spécifiquement du besoin en termes d'espace collaboratif par leur structure fiable, leur facilité d'implémentation ou leur orientation communautaire [41].
Trois SGC (Joomla, Drupal et Alfresco) ciblés pour leur pertinence vis-à-vis des attentes collaboratives de ce projet sont présentés ci-dessous :

JOOMLA

Le premier à être retenu est le plus simple d'implémentation, Joomla, un SGC libre qui signifie « tous ensemble », développé par une équipe internationale dont voici un petit schéma simplifié (figure 18) expliquant l'architecture pour la génération d'une page [42].



La page qui apparaît sur le navigateur s'appuie sur un template.

Le template définit différentes zones dans la page autour du corps de la page.

Le template s'appuie sur des feuilles de style pour l'affichage du texte sur les différentes pages du site.

Dans chaque zone, on trouve différents modules.

Selon les cas, les modules de la page font appel à un composant, un plug-in, un menu, un article.

La valeur des différents libellés est stockée dans des fichiers.

Les composants, plug-ins, menus, articles vont chercher leurs données dans la base de données.

Figure 18 : Architecture simplifiée de Joomla [42]

Pour personnaliser cet outil et se rapprocher du besoin client, de nombreux plugins existent et peuvent être intégrés comme le très populaire open-source « Community Builder » qui ajoute différents champs aux groupes d'utilisateurs ou au profil comme un compte « Facebook ». Il permet aussi d'autres workflows plus complets ou l'ajout d'images. Pour la customisation, d'autres modules peuvent être facilement créés comme des formulaires spécifiques pour un patron par exemple [43] [44].

DRUPAL

Un deuxième outil intéressant qui s'oriente principalement sur l'aspect communautaire est Drupal. C'est un SGC gérant l'information sous forme de nœuds que l'on rattache à un module forum, commentaire, livre collaboratif, etc.

Développé initialement par Dries Buytaert à partir des années 2000, Drupal fut principalement reconnu et récompensé ces trois dernières années aux États-Unis, premier dans la catégorie du meilleur système de gestion de contenu en accès libre, deuxième meilleur système de gestion de contenu en accès libre orienté réseaux sociaux, derrière Word Press ou deuxième meilleur système de gestion de contenu en accès libre et en PHP, juste derrière Joomla !

Drupal est moins intuitif que Joomla mais il peut être utilisé à quatre niveaux différents suivant la connaissance du développeur :

1. Tel quel : une fois celui-ci installé et paramétré, il est utilisable pour créer du contenu structuré et commentable par des utilisateurs qui peuvent s'enregistrer sur le site. Les menus du site ont alors un aspect standard.

2. Personnalisation simple : il est ensuite possible de personnaliser l'emplacement d'affichage, ou l'affichage lui-même, de composants visuels standards (date et heure, derniers commentaires, nombre de connectés, etc.), ainsi que l'apparence graphique du site.

3. Extension par ajouts externes : ajout, paramétrage et personnalisation de modules optionnels n'appartenant pas au noyau. À ce stade et au suivant, il n'est pas rare que le développeur du site écrive aussi un thème de présentation qui lui soit propre.

4. Extension par développement interne : écriture de nouveaux modules, qu'il est souvent efficace (mais nullement obligatoire) de présenter ensuite à la communauté afin que celle-ci puisse participer à leur évolution.

Drupal utilise une base de données comprenant typiquement 60 à 300 tables selon les modules activés et une hiérarchie de fonctions toutes substituables permettant au développeur d'application expérimenté de réécrire la seule partie qu'il désire modifier, et uniquement au niveau d'abstraction auquel il s'intéresse, sans toucher au reste. Drupal comporte environ 4000 API, mais le site api.drupal.org permet de les retrouver en accès direct par une partie quelconque du contenu de leur nom. Dans la pratique, un module simple peut fort bien n'en utiliser qu'une dizaine, voire moins [45].

La figure 19 présente la solution Drupal modélisée sous une architecture MVC (cf. chapitre 1.2.1).

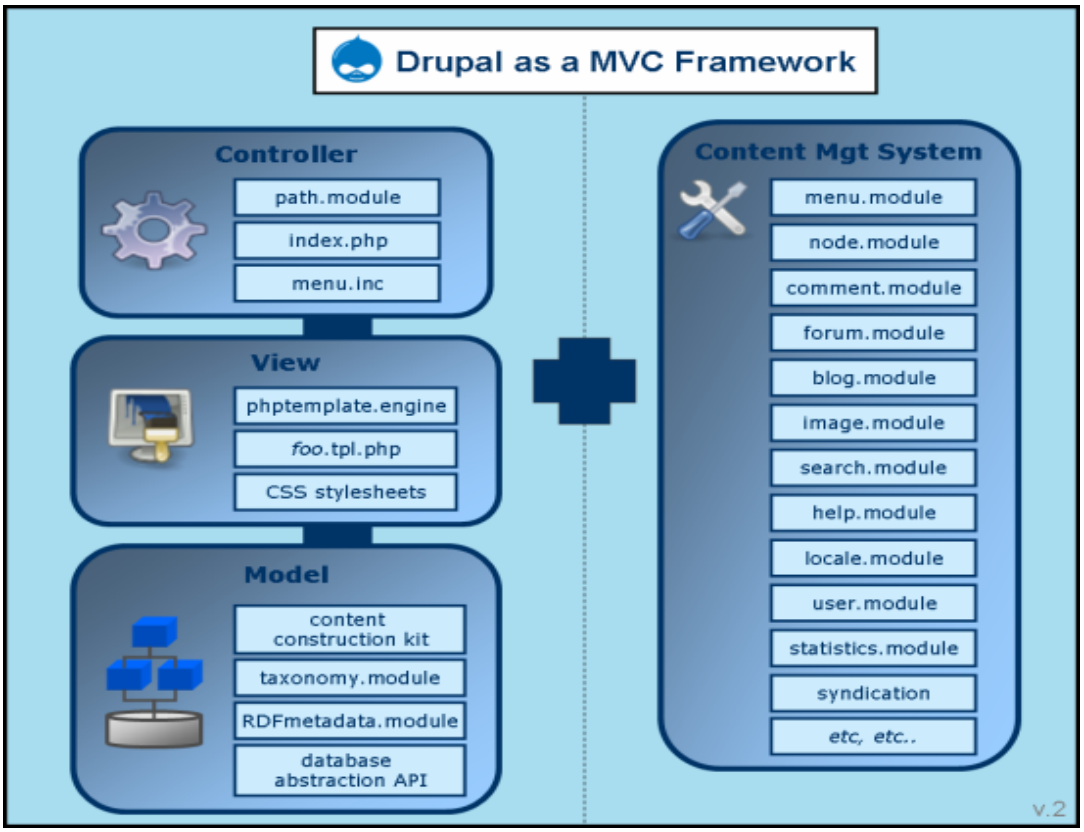


Figure 19 : Drupal sous architecture MVC [46]

Drupal possède une architecture, faisant appel a différents services disponibles tel qu'un forum et un traitement des données sous forme de nœud. La gestion de contenu sous forme de nœuds est très intéressante pour la maintenance de l'outil, elle ne nécessite pas les compétences d'un développeur, elle demande simplement une maîtrise fonctionnelle de l'outil.

ALFRESCO

Une dernière alternative est le SGC Alfresco, orienté entreprise. Il est plus puissant en termes de possibilités de personnalisation mais aussi plus complexe que ses concurrents.

Alfresco, fondé en 2005 est à la base le résultat des travaux d'une équipe qui provient massivement de Documentum, leader historique du marché, mais aussi d'Interwoven pour la partie gestion de contenu web. Le co-fondateur d'Alfresco est aussi le co-fondateur de Documentum et vient s'ajouter à la liste des entrepreneurs ayant décidé de bâtir une nouvelle société articulée sur le logiciel libre.
La mission que se donne Alfresco est : « ouvrir le monde de la Gestion de Contenu afin d'augmenter les innovations grâce à la participation de la communauté et au libre accès au code source, et viser à fournir une application complète à moindre coût, et avec plus d'agilité ». Il faut observer deux facettes importantes de ce SGC : la première est la création intuitive en seulement quelques clics de souris d'un site de partage de documents (incluant forum, wiki, etc.). La seconde et sa complexité de développement. Ciblant les professionnels, la personnalisation ou le développement Alfresco n'est pas à la portée de n'importe quel utilisateur.

La figure 20 montre l'architecture d'Alfresco.

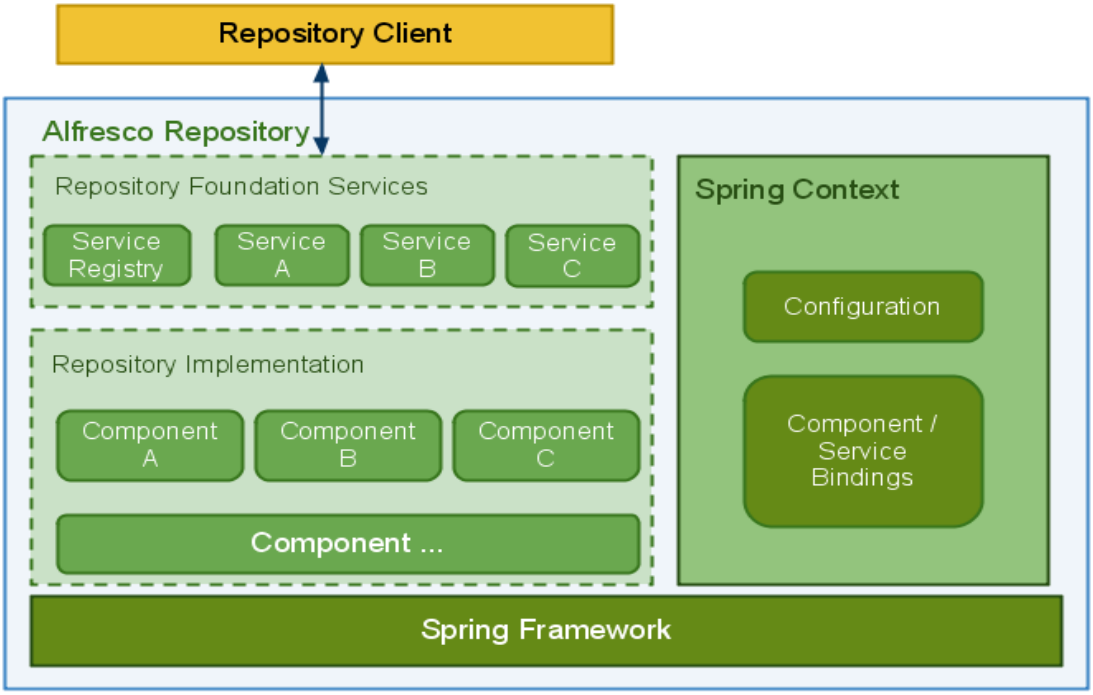


Figure 20 : Architecture Alfresco [54]

Gérant comme Drupal, l'information sous forme de nœuds, mais utilisant une architecture plus complexe, il exige un minimum d'expertise dans le domaine [47]. En effet, Alfresco utilise les composants open source les plus récents pour construire sa solution en programmation Java:

- **Spring** : Framework libre pour construire et définir l'infrastructure d'une application java. La création et la mise en relation d'objets se fait par l'intermédiaire d'un fichier de configuration décrivant les objets [48].

- **Hibernate** : Framework open-source gérant la persistance des objets en base de données relationnelle [49].

- **MySQL** : Système de gestion de base de données (SGBD) libre [50].

- **Lucene** : Moteur de recherche libre écrit en Java qui permet d'indexer et de rechercher du texte [51].

- **MyFaces** : Framework libre qui implémente les Java Server Faces (JSF) afin de développer des applications Web. En se basant sur la notion de composants, ils sont riches et manipulables grâce à une API et permettent une liaison simple client-serveur [52].

- **Programmation orientée aspect** : Paradigme de programmation qui permet de séparer les considérations techniques (aspect en anglais) des descriptions métier dans une application [53].

Depuis Alfresco 3.0, est inclus Alfresco Share, une solution collaborative de gestion de contenu rapide à déployer. Voici quelques fonctionnalités proposées par Alfresco Share [46] :

- une bibliothèque de documents (avec transfert de contenu en nombre, vignettes, visionneuse Flash, métadonnées, balises, sélections multiples et flux RSS) facilite l'utilisation d'outils de gestion de contenu.

- une fonction de recherche permet à l'utilisateur de rechercher aisément du contenu.

- des équipes virtuelles composées de membres internes et externes peuvent être créées pour des projets et des communautés.

- des flux d'activités informent les utilisateurs des nouveautés ou des évolutions inhérentes à un projet donné, en leur dévoilant l'auteur, la nature, la date et l'emplacement du contenu ajouté, modifié ou annoté ainsi que l'identité des nouveaux membres de l'équipe et les dates phares du calendrier.

- un tableau de bord personnalisé, assorti d'une interface interactive sophistiquée, permet aux utilisateurs de réaliser des configurations de sites sur mesure, suivant l'importance d'un rôle ou d'un projet.

- la création de contenu à l'aide d'un outil librement choisi : wiki, blog, Microsoft Office.

- un environnement privilégiant la rapidité de développement applicatif, utilisant un langage de script allégé et des composants réutilisables en évitant .Net et Java.

- une architecture à n niveaux, offrant une réelle évolutivité sans investissement matériel ou logiciel conséquent, capable d'accueillir davantage d'utilisateurs sur les ressources matérielles existantes.

La figure 21 nous montre la solution Share communiquant avec Alfresco.

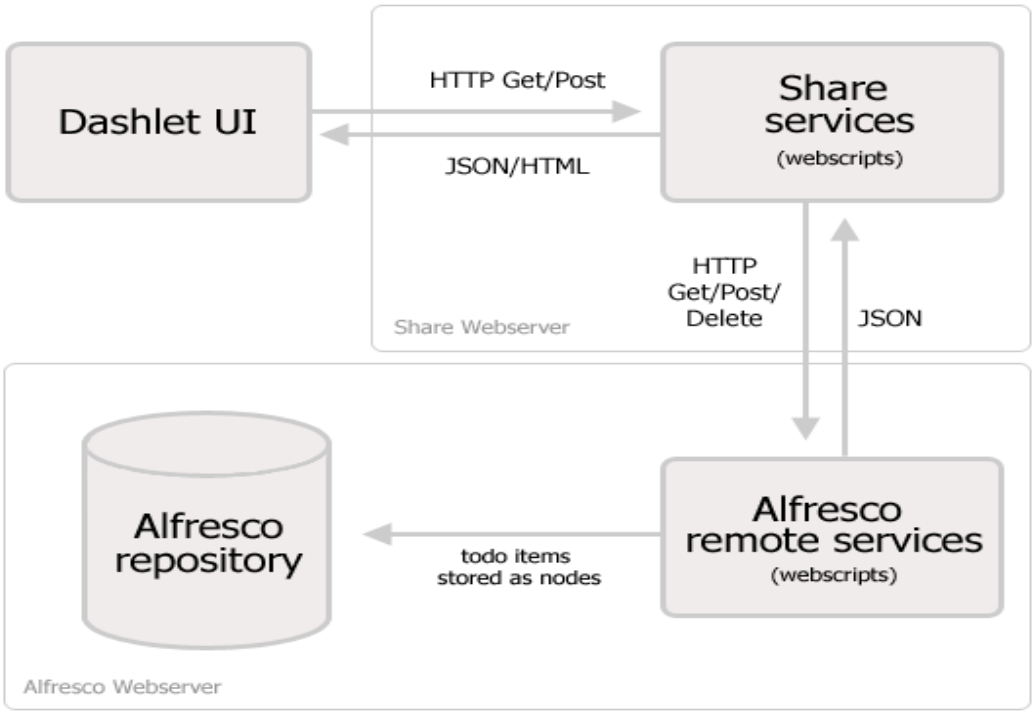


Figure 21 : Architecture Alfresco Share [55]

La solution Share communiquant avec Alfresco facilite réellement la vision communautaire en ne proposant que des services (wiki, forum, etc.) et l'affichage.

Chaque SGC a son point fort. Après avoir restreint le périmètre à trois outils, il faut identifier la solution la plus intéressante dans notre contexte. Le tableau 3 est une synthèse comparative de ces trois outils basé sur cinq critères essentiels.

Tableau 3 : Comparatif des SGC ciblés

	Accès	Technologie	Environnement	Complexité	Evolutivité
Joomla	**Libre**	**PHP**	**Partage communautaire**	**Intuitif**	**Nombreux modules existants**
Drupal	**Libre**	**PHP**	**Très orienté communauté**	**Complexe mais API très riche**	**Flexible par son API et sa structure de données sous forme de nœuds**
Alfresco	**Libre**	**JAVA**	**Partage communautaire**	**Très bien normé mais très complexe**	**Flexible par sa structure de données sous forme de nœuds et sa large possibilité de personnalisation tout en étant solide grâce à une structure normée**

Très bon Bon Prudence

Ces trois SGC proposent des solutions très proches des besoins collaboratifs de l'outil à développer. Mais chacun a son avantage et sa force, Joomla est simple et intuitif, Drupal l'est moins mais l'aspect communautaire recherché est ici privilégié et facilité par une API très riche. Alfresco quant à lui, se base sur une technologie solide pour correspondre aux plus grosses structures et aux besoins professionnels, son socle est plus complexe à interpréter et à personnaliser mais il reste plus structuré et évolutif que les deux précédents.

Alfresco utilise des composants standards normés et connus des entreprises. Il a toute la puissance, la reconnaissance et la sécurité qu'exige une structure de grande envergure. La DSI (Direction des Systèmes d'Information) des universités du PRES (Pôle de Recherche et d'Enseignement Supérieur) de Grenoble met en place Alfresco Share pour optimiser le travail collaboratif au sein de l'université. C'est donc Alfresco combiné avec Share qui est la solution choisie pour développer une gestion de patrons collaborative.

Le développement de l'outil de gestion de modèles de conception se basera donc vis-à-vis du besoin et de son contexte sur ce triplet très pertinent {XP@SCRUM, AGAP, ALFRESCO}. Ce triplet s'harmonise extrêmement bien, l'état d'esprit collaboratif ressort dans chacune des solutions.

Solution Copen

(COmmunities of Patterns Environment)

2. Solution COPEN (COmmunities of Patterns ENvironment)

Après un état de l'art mettant en évidence que les produits existants ne correspondent que partiellement au besoin, nous avons défini un nouvel outil basé sur le triplet {XP@SCRUM, AGAP, ALFRESCO} :
COPEN (COmmunities of Patterns ENvironment). Développé lors de ce mémoire en utilisant la fusion de deux méthodes Agile, XP et SCRUM, COPEN ajoute des possibilités de collaboration autour de la gestion de patrons que pouvait proposer l'atelier AGAP. En effet, grâce au CMS Alfresco offrant une structure de développement solide et un espace collaboratif déjà implémenté, la modélisation sous forme de patrons devient nettement plus vivante et évolutive.

Le développent en méthode Agile ne signifie pas livrer un travail uniquement spécifié par des *user stories*. Au fil du projet, il peut être intéressant de modéliser l'outil sous forme de diagrammes. Non seulement cela peut aider un développeur dans sa tâche, mais cela peut aussi permettre à tout moment de reprendre une méthode de gestion de projets plus classique et basée sur des modèles. Cette approche est prise en compte dans ce mémoire.

Tout d'abord une analyse et une conception de la solution COPEN est présentée afin de définir concrètement l'outil. Une seconde partie précise la stratégie mise en place permettant son développement en méthode Agile. Puis une dernière partie apportera un bilan et les perspectives de l'outil.

2.1. Analyse et conception

En tout premier lieu, avant de parler d'implémentation de COPEN dans le SGC Alfresco, trois visions sont importantes afin de comprendre la conception de l'outil.

- La première vision est statique sous forme de classes dont l'objectif est essentiellement de mettre en évidence les principaux concepts manipulés et leurs propriétés.
- La seconde est celle en cas d'utilisation dont l'objectif est d'identifier les acteurs et les fonctionnalités auxquelles ils ont accès.
- La dernière, dynamique, présente le cycle de vie des principaux objets du système : les communautés et les patrons.

2.1.1. Vision statique

Figure 22Figure 22, un diagramme de classes modélise COPEN, les membres, les communautés, les patrons et les collaborations.

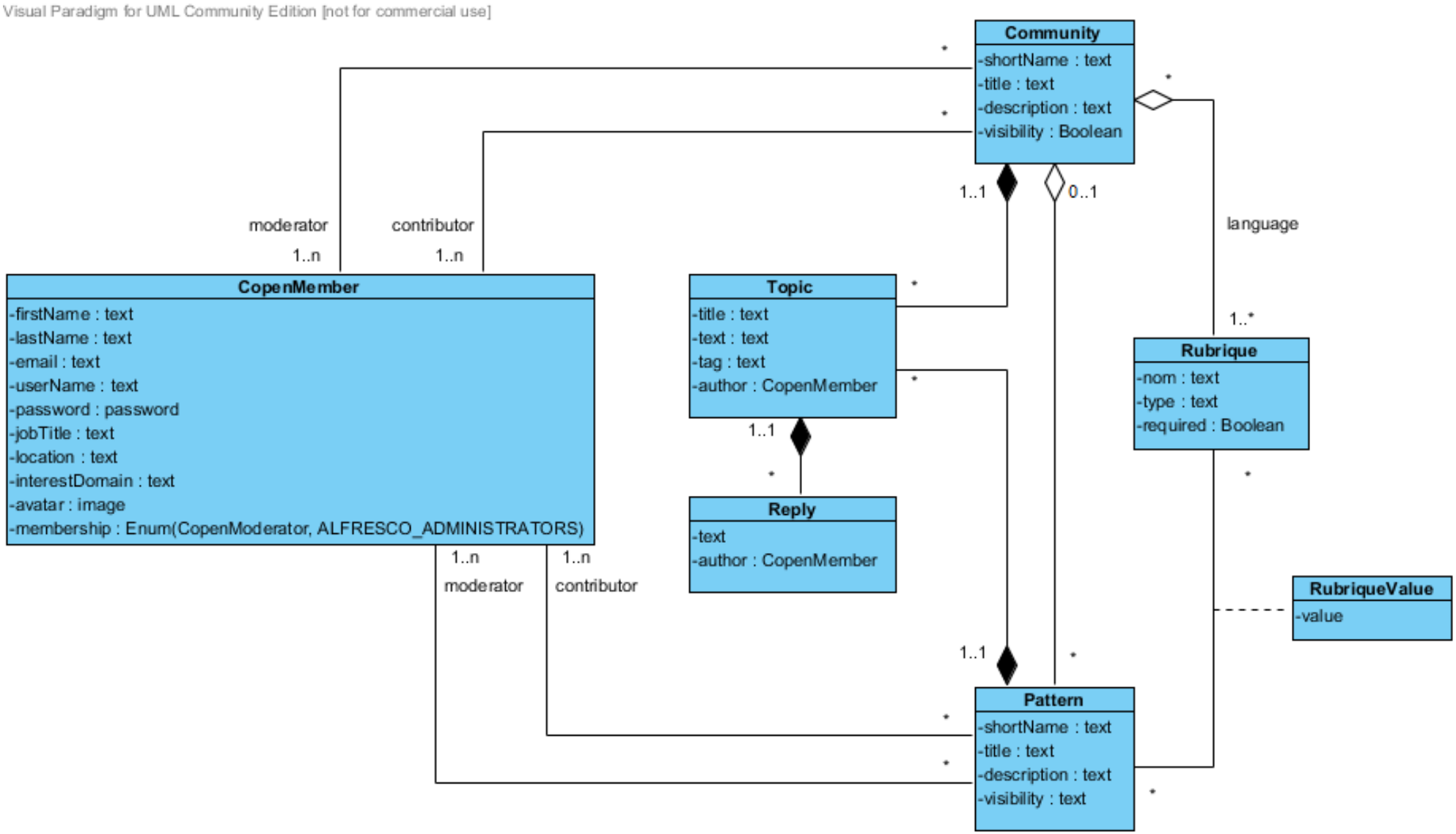


Figure 22 : Diagramme de classes COPEN

Ces classes ont chacune leur intérêt :

- COPENMember permet de définir les membres de l'outil avec leurs identifiants et informations personnelles.
- Community est une classe caractérisée par un titre et une description.
- Pattern est une classe caractérisée par un titre et une description.
- Les classes Rubrique et RubriqueValue permettent de définir un langage composé de Rubriques ainsi que son instanciation grâce à des valeurs pour chacune des rubriques.
- Les classes Topic et Reply permettent un fil de discussions au niveau des communautés et des patrons.

Quatre relations sont importantes à souligner :

- Tout d'abord la relation de rôles « communautaires » entre le membre COPEN et la communauté ou le patron.
- L'aspect collaboratif est présent au sein d'une communauté mais aussi d'un patron grâce au sujet de discussion « Topic » et aux objets « Reply » permettant échange et discussion.
- L'association de rubriques à une communauté lui définissant son propre langage.
- Le rattachement de patrons à une communauté avec pour chaque patron, une instanciation de chacune des rubriques de la communauté.

Pour illustrer cette vision par classes et apporter des détails supplémentaires sur l'interaction de chacune d'entre elles, la vision en cas d'utilisation est très pertinente.

2.1.2. Cas d'utilisation

Cette vision est présentée ci-dessus sous la forme de plusieurs diagrammes. Tout d'abord une présentation des membres puis une vue de chacun d'entre eux face aux fonctionnalités auxquelles ils ont accès sous la forme de cas d'utilisation.

2.1.2.1. Membres

Une présentation des membres COPEN en tant qu'acteurs interagissant avec le système est décrite en figure 23.

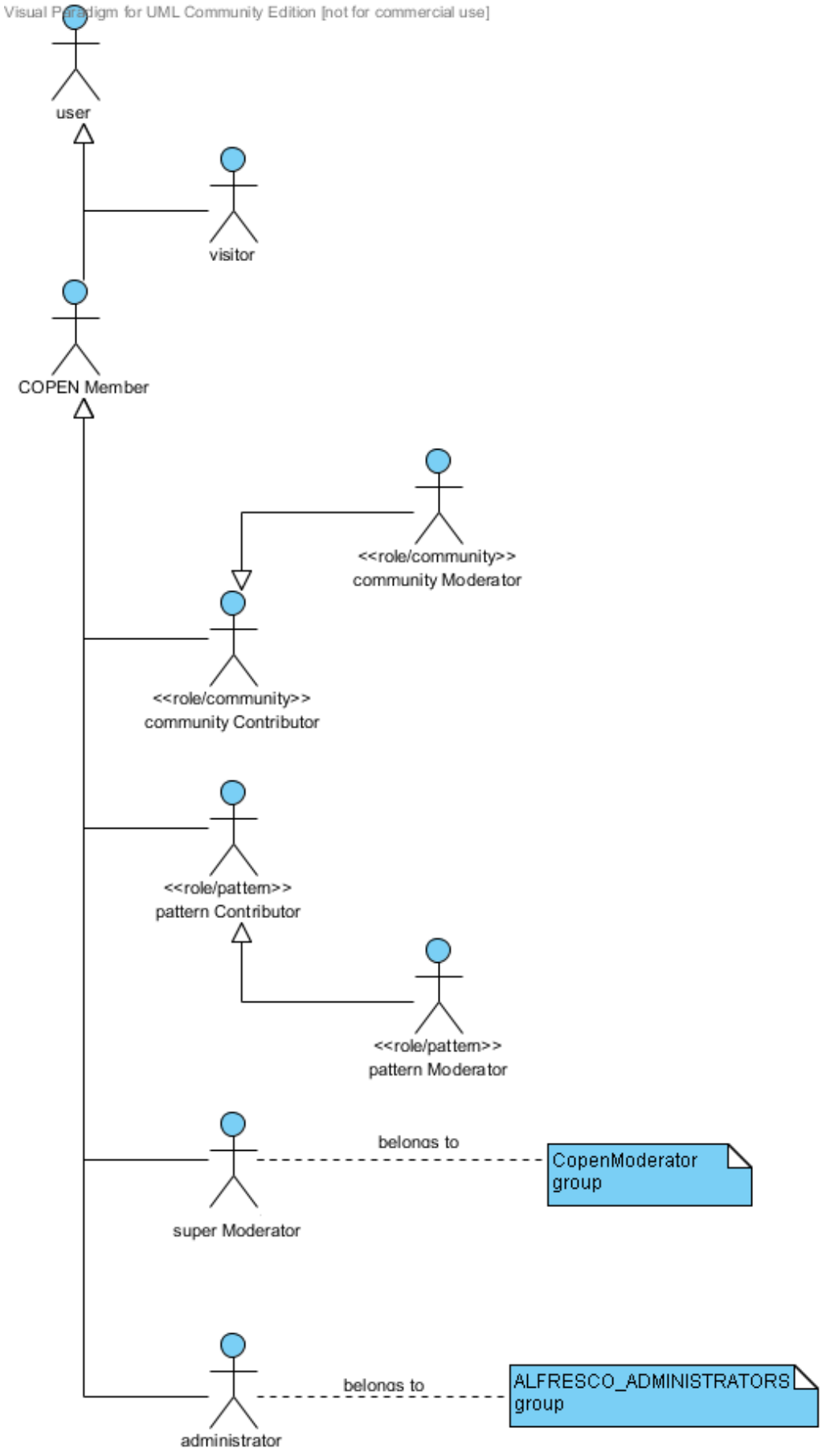


Figure 23 : Membres COPEN

Utilisateurs

Différents type d'utilisateurs existent afin de pouvoir accéder à COPEN avec des restrictions de droits :

- Visitor : il possède les droits les plus restreints permettant simplement la visualisation de communautés et patrons publics. Ce visiteur n'est pas un adhérent de COPEN.

- Membre COPEN : il a une possibilité de création de communautés et patrons, de collaboration avec d'autres membres via un forum ou de modération suivant le rôle qui lui est attribué (Contributor ou Moderator).

- Membre superModerator : il possède le statut « moderator » sur toutes les communautés et patrons existants afin de pouvoir contrôler l'ensemble des informations disponibles sur l'outil.

- Membre administrator : il possède un espace de gestion de groupes et de membres afin de les rechercher, visualiser, créer, éditer, affecter à des rôles ou de les supprimer.

Rôles

Chaque membre COPEN a des droits suivant son profil et son rôle :

Deux rôles affectent les communautés :

- Le rôle contributor qui permet de collaborer en donnant son avis sur la communauté ou de créer des patrons au sein de celle-ci.

- Le rôle manager ajoute des droits d'invitation de membres COPEN mais aussi de publication et d'édition de descriptions de la communauté.

Deux rôles affectent les patrons :

- Le rôle contributor qui permet de collaborer en donnant son avis sur le patron.

- Le rôle manager ajoute des droits d'invitation de membres COPEN mais aussi de publication et d'édition du patron. L'édition est plus riche que pour une communauté, ici l'ensemble des rubriques caractérisant le langage sont modifiables.

2.1.2.2. Cas d'utilisation pour les membres « COPEN Member »

Figure 24, une vue globale spécifie les possibilités offertes à l'utilisateur mais plus particulièrement au membre COPEN sur les membres, communautés et patrons.

Vue globale

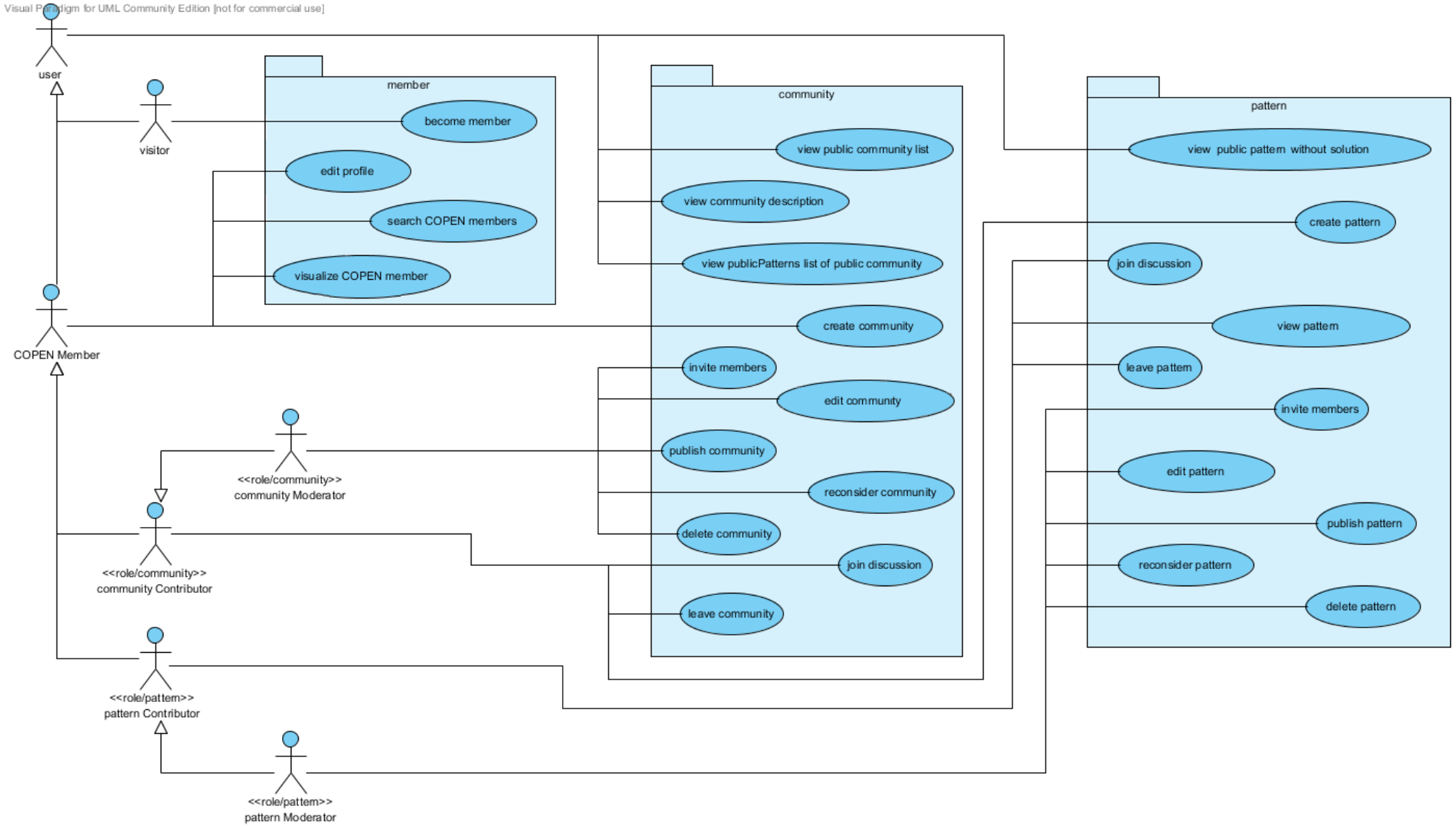


Figure 24 : Diagramme global des cas d'utilisation pour le membre « Copen Member »

Il faut retenir de cette vue d'ensemble les différences de point de vue entre chaque membre et rôle COPEN. Un visiteur ne peut visualiser que les données publiées, un membre COPEN « Contributor » est là pour la partie collaborative, donner son avis et faire vivre la communauté et ses patrons. Un membre « Moderator », lui, possède plus de responsabilités : il édite, modélise les données, invite et prend en compte les avis et suggestions des autres membres.

Voici un découpage en trois vues permettant de comprendre les cas d'utilisation permettant la gestion des trois principales classes : membres COPEN, communautés et patrons.

Vue membre

La figure 25 ainsi que sa description permet de comprendre les cas d'utilisation sur les membres

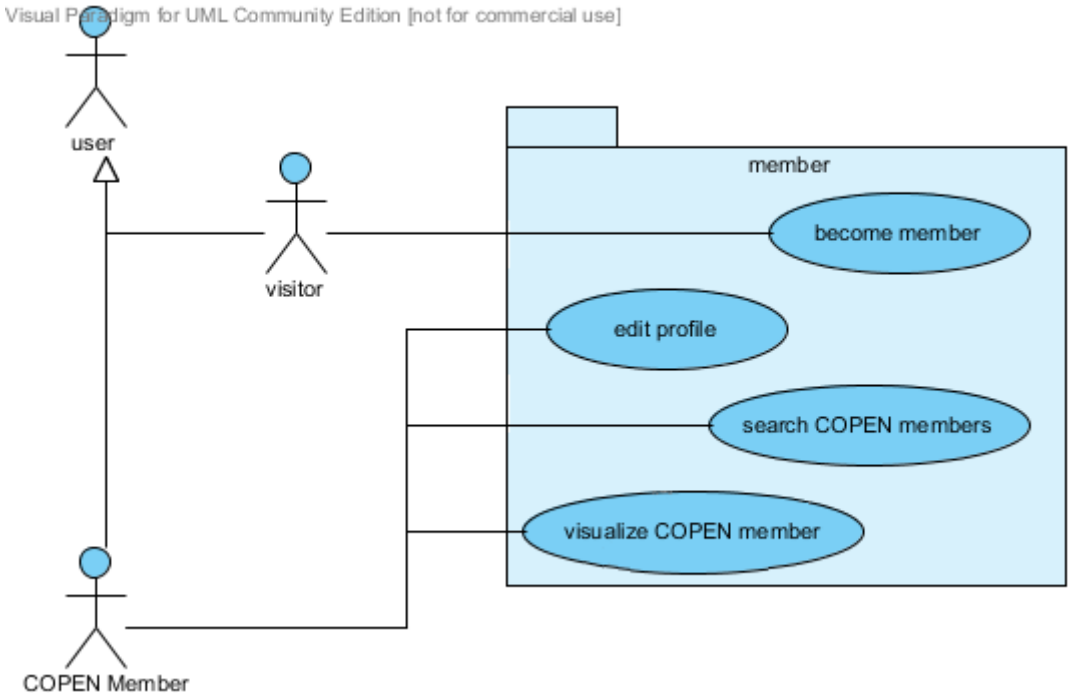


Figure 25 : Diagramme de cas d'utilisation : vue membre

become member
Demander, en tant que visiteur, la création de compte en envoyant un mail avec les informations nécessaires (First Name, Last Name, Email address, User Name, Password, Job Title, Location, Interest domain) à l'adresse électronique précisée sur la page d'accueil.

edit profile
Accéder aux données personnelles de son profil et modifier les champs qui le caractérise (Email address, Password, Job Title, Location, Interest domain, avatar).

search copen members
Rechercher des membres COPEN en précisant une partie de leur nom.

visualize copen member
Visualiser un membre COPEN : profil, les communautés et patrons publics ou privés que mes droits permettent de visualiser.

Grâce à l'aspect communautaire apporté par le socle Alfresco, chacun à la possibilité de connaître les caractéristiques des membres avec lesquels il collabore. Cela permet aussi de montrer les intérêts de chacun pour une communauté ou un patron.

Vue communauté

La figure 26 ainsi que sa description permet de comprendre les cas d'utilisation sur les communautés.

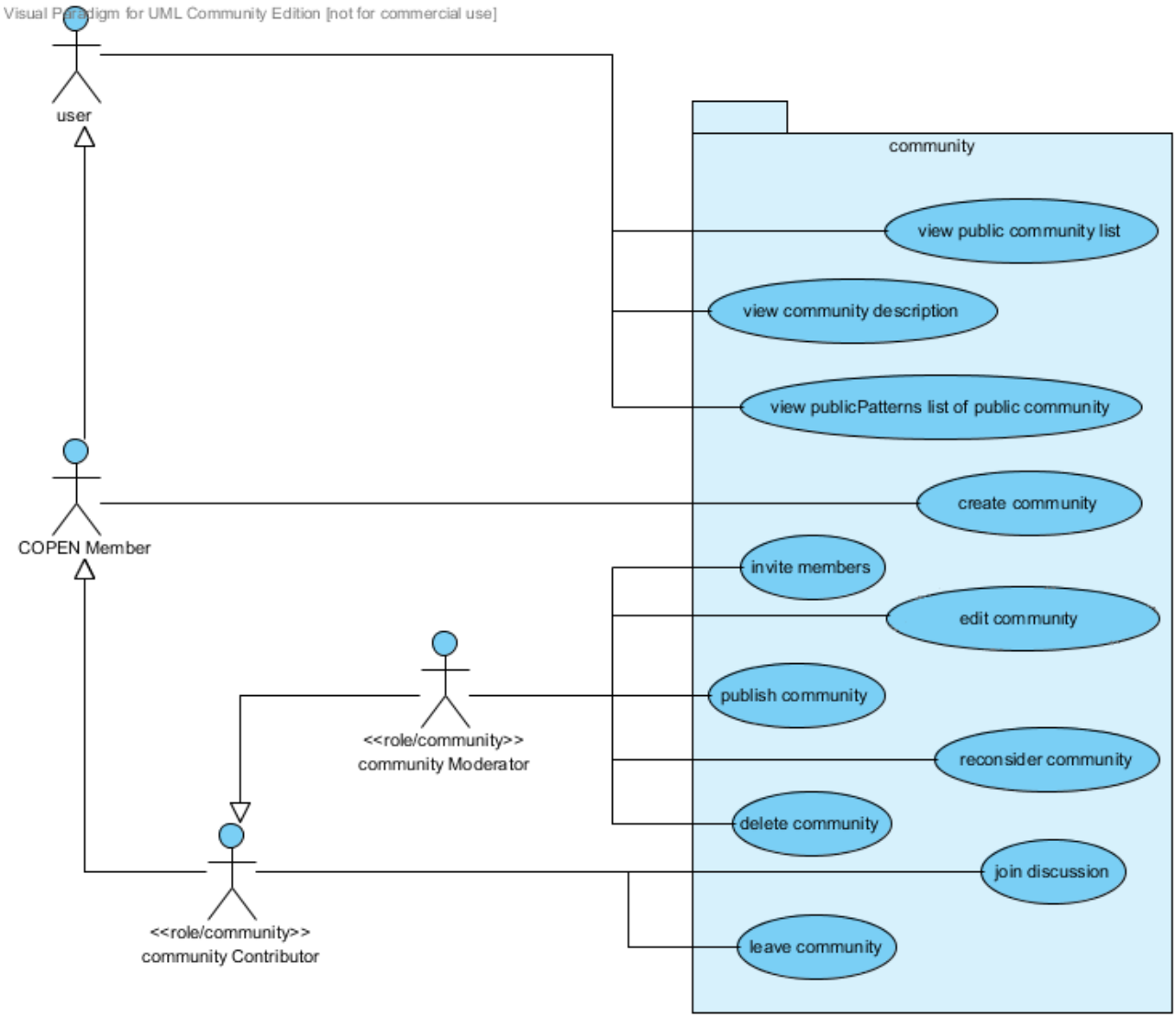


Figure 26 : Diagramme de cas d'utilisation : vue communauté

view public community list
Visualiser la liste de toutes les communautés publiques.

view community description
Visualiser la description d'une communauté.

view public patterns list of public community
Visualiser la liste de tous les patrons publics accessibles au sein d'une communauté publique.

create community
Accéder à la page de création de communauté. Spécifier le titre, la description et le langage (basé sur P-Sigma) qui sera utilisé pour tous les patrons qui lui appartiendront.

edit community
Editer la description renseignée lors de la création de la communauté.

join discussion (for community)
En tant que membre (collaborator ou moderator) de la communauté, il est possible de visualiser le forum de discussion puis participer en créant des sujets de discussions et en collaborant sur d'autres déjà créés.

leave community
Quitter la communauté, c'est-à-dire annuler son adhésion à celle-ci.

invite members
A partir de la communauté souhaitée, pour une personne non-inscrite, préciser son nom, prénom et adresse électronique puis l'ajouter à l'espace d'invitation. Choisir un rôle à cette personne sur la communauté (contributor ou moderator) et lancer l'invitation. Celle-ci lui sera envoyée par mail avec un lien direct et ses identifiants pour accéder à l'outil. L'invitation peut aussi se faire en sélectionnant un membre COPEN (aucun compte ne sera recréé).

publish community
Une fois que la communauté est correctement créée et décrite, il est possible de la publier, c'est-à-dire la rendre visible de tous.

reconsider community
Après retours d'expérience ou autres événements, il peut être intéressant de reconsidérer une communauté, la privatiser à nouveau afin d'apporter les modifications nécessaires.

delete community
Supprimer une communauté n'ayant plus aucun intérêt.

Une communauté vie, elle est créée, discutée, animée, complétée ou simplement visualisée. La collaboration lui permet de ne pas devenir obsolète mais de grandir et se perfectionner avec le temps. Plus la collaboration est forte au sein de cette communauté, plus la création de patrons (essence d'une communauté) sera importante et contribuera à son avenir. C'est la collection de patrons qui caractérise une communauté et qui enrichit les discussions.

Vue patron

La figure 27 ainsi que sa description permet de comprendre l'importance des cas d'utilisation sur les patrons.

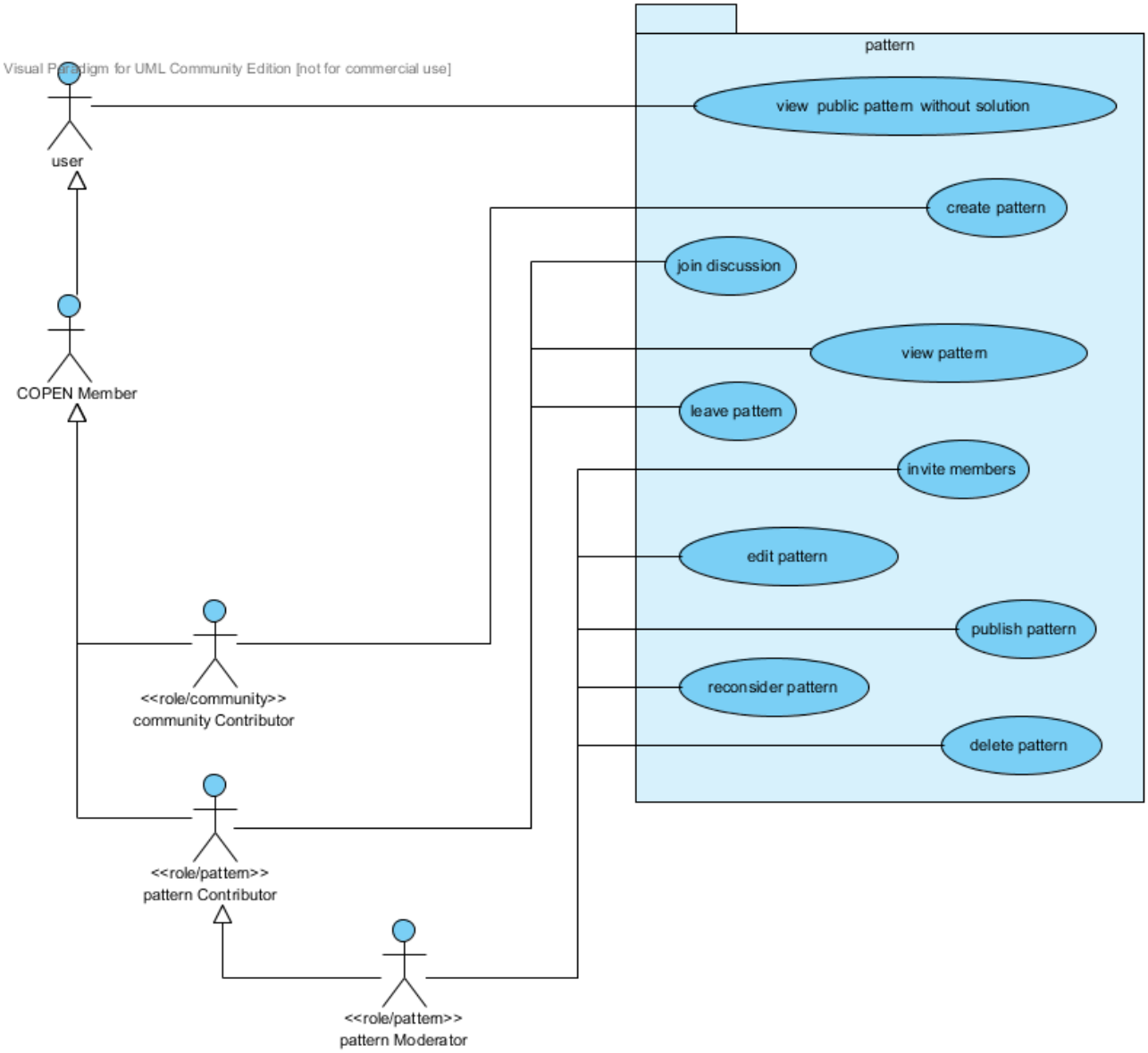


Figure 27 : Diagramme de cas d'utilisation : vue patron

Un patron COPEN propose une modélisation d'un produit ou d'un processus, pour ce faire, chacune des rubriques d'une communauté est ici instanciée (cas d'utilisation « edit pattern»). C'est pour cela qu'au sein d'un patron la collaboration est non seulement intéressante mais devient primordiale. Il est incontournable pour la conception d'un patron d'avoir des avis extérieurs, de les discuter, de retravailler ce patron et de le faire évoluer au fil du temps.

create pattern
A partir d'une communauté, accéder à la page de création de patron (en tant que membre de la communauté). Spécifier le titre, la description. Les rubriques du langage de la communauté le caractériseront.

view public pattern without solution
Accéder à une communauté publique puis un patron public sous celle-ci. En tant qu'utilisateur ne possédant pas les droits de visualisation de solution (exemple : visitor), la description du patron est affichée sans solution (un message de droits insuffisants s'affiche).

join discussion (for pattern)
En tant que membre (collaborator ou moderator) du patron, il est possible de visualiser le forum de discussion de celui-ci puis participer en créant des sujets de discussions et en collaborant sur d'autres déjà créés.

view pattern
Accéder à une communauté puis un patron sous celle-ci. Possédant les droits de visualisation de solution (membre du patron), la description du patron est affichée avec sa solution. Cette description est caractérisée par les rubriques du langage de la communauté à laquelle il appartient.

leave pattern
Quitter le patron, c'est-à-dire annuler son adhésion à celui-ci.

invite members
A partir du patron souhaité, pour une personne non-inscrite, préciser son nom, prénom et adresse électronique puis l'ajouter à l'espace d'invitation. Choisir un rôle à cette personne sur le patron (contributor, moderator) et lancer l'invitation. Celle-ci lui sera envoyée par mail avec un lien direct et ses identifiants pour accéder à l'outil. L'invitation peut aussi se faire en sélectionnant un membre COPEN (aucun compte ne sera recréé).

edit pattern
Accéder à une communauté puis un patron sous celle-ci. Possédant les droits de moderator sur le patron, il est possible d'éditer les rubriques de celui-ci ainsi que la description de l'espace patron.

publish pattern
Une fois que le patron est correctement créé et décrit, il est possible de publier le patron, c'est-à-dire le rendre visible de tous.

reconsider pattern
Après retours d'expérience ou autres événements, il peut être intéressant de reconsidérer un patron, le privatiser à nouveau afin d'apporter les modifications nécessaires.

delete pattern
Supprimer un patron n'ayant plus aucun intérêt.

Pour que les patrons COPEN soit cohérents, il faut aussi que ce soit un outil crédible et que les membres soient sérieux. Pour que les communautés s'agrandissent et que les patrons soient significatifs et durent dans le temps, il faut une modération globale, une « super-Modération » qui veille à ce que tout se déroule dans de bonnes conditions. C'est dans ce sens qu'un membre COPEN avec le profil « super Moderator » vient apporter sa collaboration.

2.1.2.3. Cas d'utilisation pour les membres « super Moderator »

Un diagramme en figure 28 souligne l'accès aux informations et aux cas d'utilisations que possède le membre « super Moderator ».

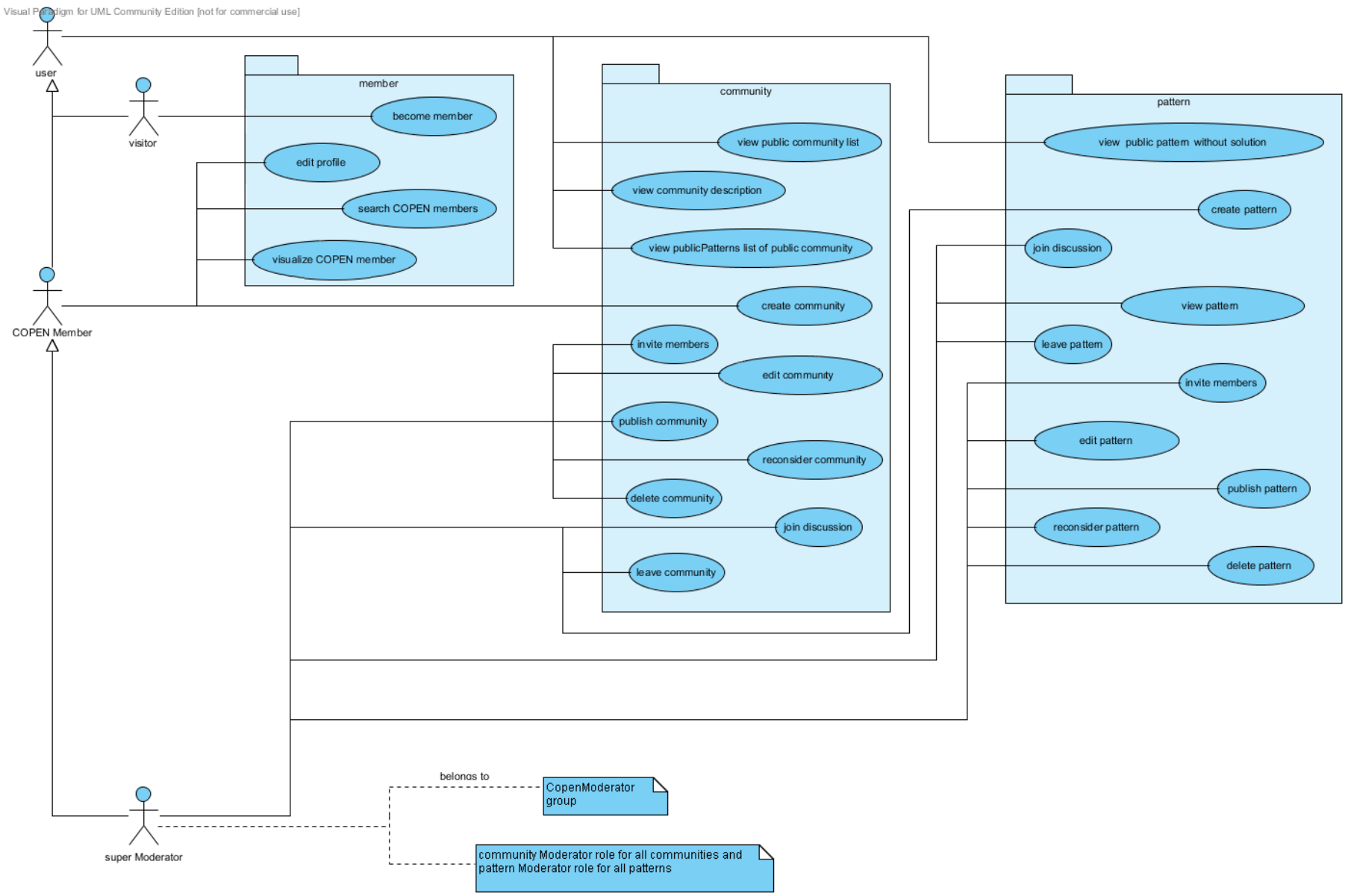


Figure 28 : Diagramme des cas d'utilisation pour le membre « super Moderator »

Les cas d'utilisation sont les mêmes que ceux du « COPEN Member ».

Une particularité caractérise tout de même le « super Moderator » :
Le rôle de « moderator » lui est affecté pour chaque communauté ainsi que pour chaque patron. En effet son rôle n'est pas seulement d'utiliser l'outil mais de veiller à son bon fonctionnement. Afin d'accéder à tous les contenus et de contrôler toutes les activités, chaque création de communauté ou de patron implique de lui affecter le rôle de modérateur directement. C'est principalement de cette manière que le « super Moderator » apporte sa collaboration et fait vivre l'outil.

2.1.2.4. Cas d'utilisation pour les membres « administrator »

L'administrateur s'occupe de la gestion des membres COPEN. Un diagramme en figure 29 souligne l'accès aux informations et aux cas d'utilisations que possède le membre « administrator ».

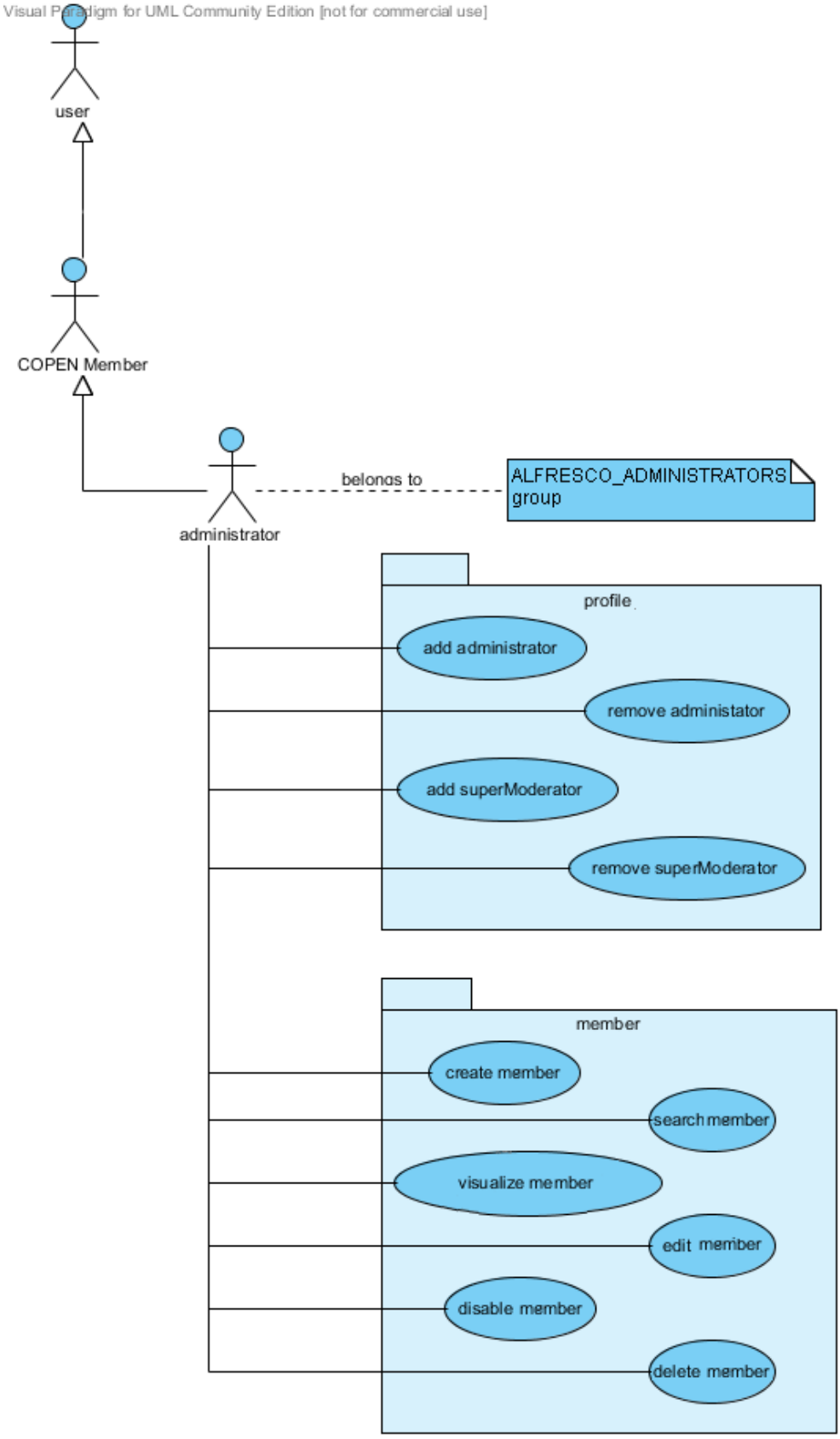


Figure 29 : Diagramme des cas d'utilisation pour le membre « administrator »

add administrator
Ajouter le rôle « administrator » à un membre.

remove administrator
Supprimer le rôle « administrator » à un membre.

add super Moderator
Ajouter le rôle « super Moderator » à un membre.

remove super Moderator
Supprimer le rôle « super Moderator » à un membre.

create member
Lorsqu'il est nécessaire de créer un utilisateur, un espace réservé à l'administrateur lui permet de remplir les champs nécessaires (First Name, Last Name, Email address, User Name, Password, Job Title, Location, Interest domain) et de valider la création.

search member
Rechercher un utilisateur en précisant une partie de son nom permettant ainsi de retrouver un membre avec un résultat de recherche plus détaillé que pour un autre membre COPEN classique.

visualize member
Afficher un utilisateur permet à l'administrateur de visualiser les caractéristiques de celui-ci avec plus de précisions et d'informations (activation du compte) qu'à partir d'un membre COPEN classique.

edit member
L'administrateur après avoir visualisé les caractéristiques du profil d'un utilisateur, peut choisir d'éditer celles-ci afin de modifier la donnée erronée.

disable member
Choisir l'option « disable » pour un utilisateur permet de désactiver son compte jusqu'à ce que l'on change cette valeur à « enable ».

delete member
La suppression d'un membre COPEN est disponible si celui-ci n'est plus utile ou du moins plus souhaité.

Ce membre est essentiel à l'évolution de l'outil, c'est lui qui crée les membres COPEN qui gère leur profil, qui suspend leur compte ou même qui les supprime. Il a sa part, tout comme le « super Moderator » dans le bon fonctionnement de COPEN.

La collaboration est omniprésente au sein de cet environnement. Il y a d'un côté une collaboration entre les membres pour que l'outil vive dans de bonnes conditions mais aussi une collaboration à l'intérieur des communautés créées afin de modéliser processus et produits. Ces communautés ainsi que les patrons qui les caractérisent ont un cycle de vie centré sur cette collaboration. C'est le fruit de celle-ci qui va permettre de publier une communauté ou un patron abouti. Dans un cas inverse, cette collaboration peut aussi permettre de privatiser des communautés ou patrons devenus obsolètes afin de les rediscuter.

2.1.3. Cycles de vie

Le cycle de vie d'une communauté ou d'un patron doit leur permettre une constante évolution. Chaque état doit laisser place à la discussion et à la reconsidération de la communauté ou du patron. Leur représentation ci-dessous détaille cet aspect.

2.1.3.1. Cycle de vie d'une communauté

La figure 30 illustre le cycle de vie d'une communauté.

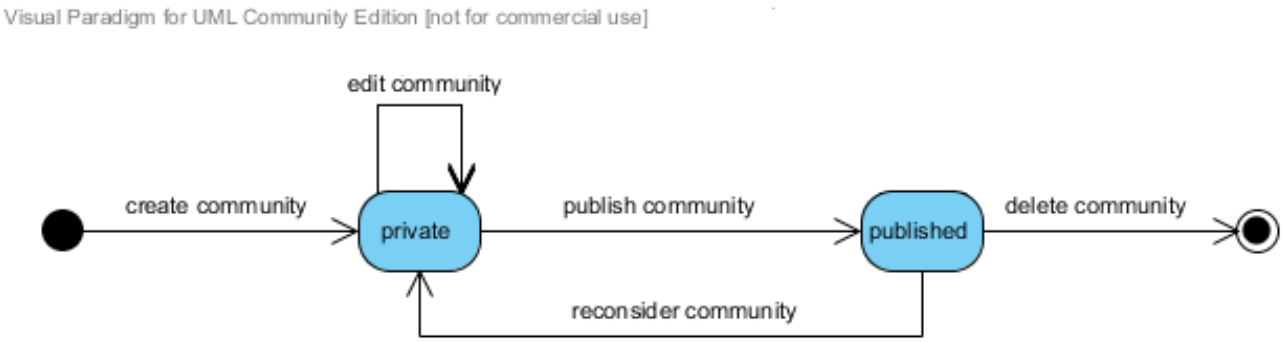


Figure 30 : Cycle de vie d'une communauté

Lorsqu'une communauté existe, deux états peuvent la caractériser :

- Un premier état « private » permet de la faire évoluer, la compléter dans un cadre fermé, en invitant seulement quelques membres essentiels. L'édition de la description d'une communauté avant publication caractérise principalement l'intérêt de cet état.

- Un second état « published » permet tout autant la discussion mais n'a pas pour but de modifier ses informations. La communauté est considérée aboutie. Si la collaboration de membres amène à un besoin de modification, elle devra alors redevenir privée pour être modifiée.

Cet aspect collaboratif permet de visualiser des communautés publiques abouties et approuvées par un groupement de membres ayant échangé leurs avis.

Ce principe de collaboration est aussi présent dans le cycle de vie d'un patron afin d'être cohérent et fiable.

2.1.3.2. Cycle de vie d'un patron

La figure 31 illustre le cycle de vie d'un patron.

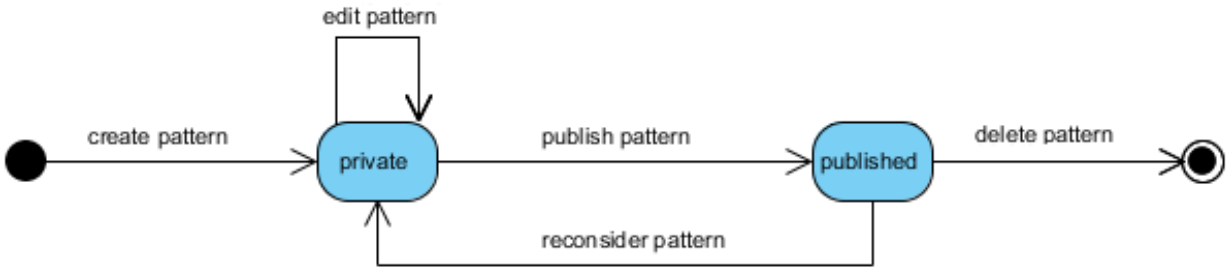


Figure 31 : Cycle de vie d'un patron

Lorsqu'un patron existe, deux états peuvent la caractériser :

- Un premier état « private » permet de le faire évoluer, le compléter dans un cadre fermé, en invitant seulement quelques membres essentiels. L'édition de la description du patron avant publication caractérise principalement l'intérêt de cet état. En effet le langage d'une communauté (constitué de rubriques) que le patron pourra instancier permet à celui-ci de modéliser le processus ou le produit souhaité.

- Un second état « published » permet tout autant la discussion mais n'a pas pour but de modifier ses informations. Le patron est considéré abouti. Si la collaboration de membres amène à un besoin de modification, il devra alors redevenir privé pour être modifié.

Cet aspect collaboratif permet de visualiser des patrons publics aboutis et approuvés par un groupement de membres ayant échangé leurs avis.

2.1.3.3. Contraintes entre cycles de vie

Le fait de rattacher des patrons à une communauté (relation de composition) crée des contraintes entre les deux cycles de vie :

- Un patron peut seulement être créé sous une communauté.

- La suppression d'une communauté n'entraîne pas la suppression des patrons qu'elle possède mais restreint leur accès uniquement par le profil de leurs membres.

- L'adhésion d'un membre COPEN à un patron ne lui ajoute pas l'adhésion à la communauté à laquelle le patron appartient.

Les communautés permettent de regrouper des patrons à leur création, mais la suppression des communautés ne les supprime pas pour autant, il a été décidé de garder les patrons, chaque modérateur de patron supprimera son patron s'il trouve celui-ci obsolète. De plus, certaines libertés sont gardées :

- Une communauté peut être publique sans pour autant posséder de patrons.

- Une communauté peut être publique tout en possédant des patrons privés.

- Une communauté peut être privée tout en possédant des patrons publics.

- Le changement de statut d'une communauté n'impacte en rien le statut des patrons qu'elle possède.

- Le changement de statut d'un patron n'impacte en rien le statut de la communauté à laquelle il appartient.

Cette liberté dans les statuts permet de garder des patrons visibles même s'il est nécessaire de revoir la description de la communauté à laquelle ils appartiennent.

COPEN est un outil de modélisation par patrons. Mais pour lui permettre d'avoir des patrons crédibles, fiables et à jour, la solution est la communication, l'échange et le dialogue. C'est pour cela que COPEN, comme les précédents diagrammes l'ont démontré, dispose d'une solution collaborative au cœur de son espace communautaire et de ses patrons. Chaque phase du cycle de vie propose des cas d'utilisations basés sur de la coopération, cette approche permet d'avoir des patrons les plus vivants possible, ce qui manquait dans les outils précédents.

Cet environnement communautaire est tout simplement l'adaptation à la modélisation par patrons du SGC Alfresco.

2.2. Architecture de développement

2.2.1. Implantation COPEN dans un Framework Alfresco

La base de développement de notre outil communautaire est le Framework Alfresco Community version 3.2 r2 contenant les applications web Alfresco et Share comme présenté dans les Outils existants (figure 20 et figure 21) de la partie « Espace communautaire personnalisé » de l'Etat de l'art.

L'environnement que propose ce Framework est complet et très proche de celui recherché pour un environnement de patrons collaboratifs. La modélisation objet proposée par le repository Alfresco est suffisante pour modéliser les objets COPEN (définis en figure 22). Il n'est donc pas nécessaire de mettre en place un environnement de développement Java. C'est au niveau de la couche services qu'il faut personnaliser ces deux applications. La première, Alfresco nécessite principalement une personnalisation définissant la relation entre les nœuds communautés et les nœuds patrons. La seconde, Share, nécessite beaucoup plus de modifications. Son rôle est de permettre à un client d'accéder aux données Alfresco au sein d'un espace collaboratif. C'est celle-ci qu'il faudra adapter pour correspondre à l'environnement de patrons spécifié dans les sections 2.1 ci-dessus. Share est donc le cœur de l'outil à développer, la personnalisation de cette application se nomme COPEN.

Une illustration de l'architecture composée des Applications Alfresco et COPEN est disponible en figure 32.

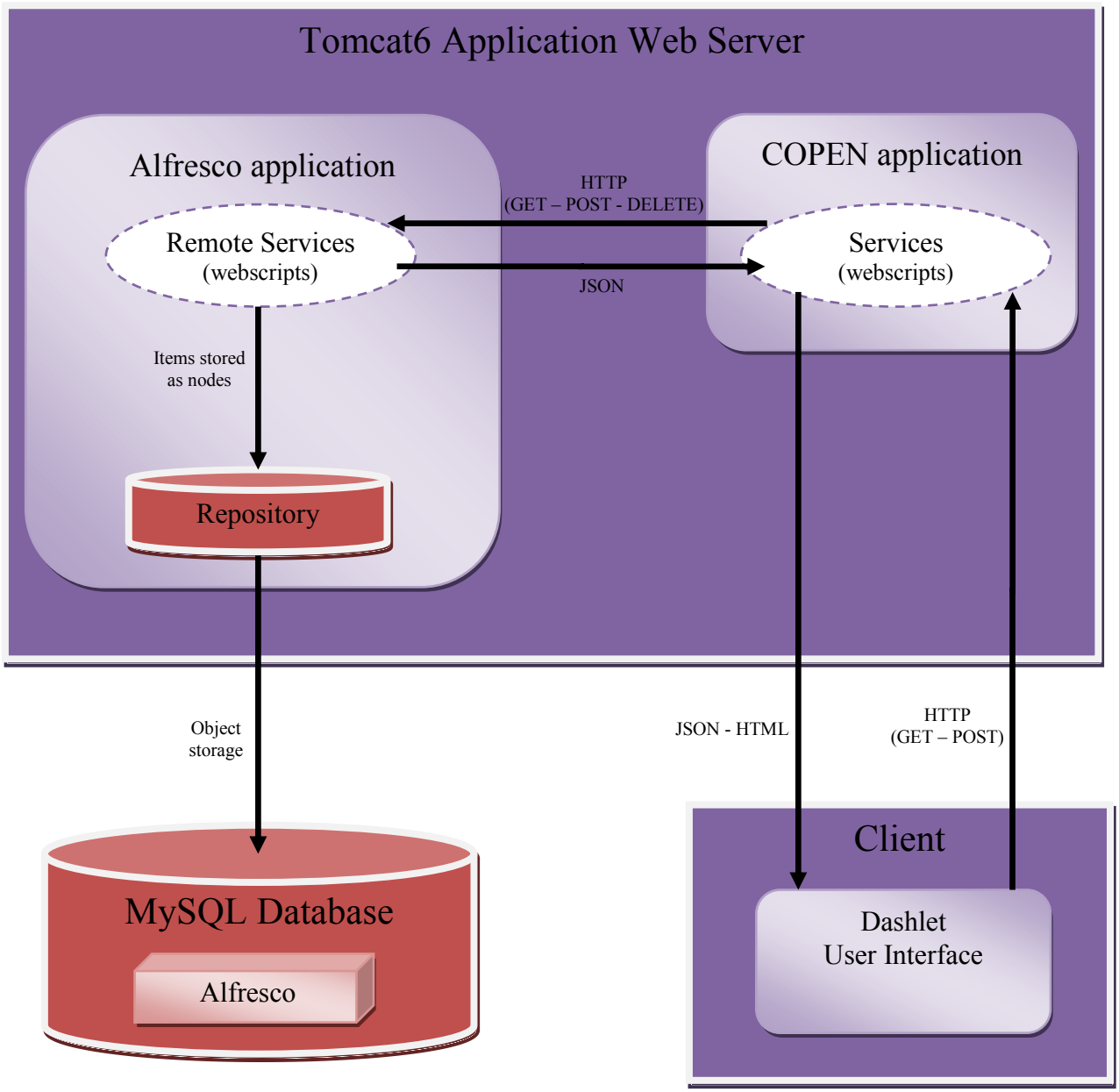


Figure 32 : Architecture des applications Alfresco et COPEN

Scénario d'utilisation :

1. Un client envoie une demande d'affichage via une requête http à l'application COPEN
2. COPEN, grâce au service approprié, contrôle la demande et envoie une correspondance à l'application Alfresco.
3. Alfresco, grâce au service approprié accède à son repository qui gère les objets et leur stockage en base de données.
4. Il renvoie ensuite la réponse en format JSON (JavaScript Object Notation) à l'application COPEN.
5. COPEN renvoie à son tour le résultat au client en format JSON ou HTML (Hyper Text Markup Language) grâce au service approprié.

L'architecture proposée permet une facilité dans le développement de l'outil. Le fait de simplement personnaliser les services sans avoir à enrichir le modèle objet Alfresco est important pour une évolution future, important pour le développement de nouvelles fonctionnalités simplement en ajoutant des services mais aussi pour mettre à jour la version d'Alfresco Community. Résolution de

bugs, nouvelles fonctionnalités, enrichissement du modèle objet, toutes ces mises à jour peuvent être intégrées à l'architecture COPEN sans demander un gros effort d'intégration de versions.

COPEN axe sa modélisation par patron sur de la collaboration grâce au fait qu'il soit basé sur une application collaborative Alfresco Share. La modélisation des objets COPEN est possible sans ajouter de classes Java.

2.2.2. Implantation d'un service COPEN

Chacun des cas d'utilisations nécessite le développement de services. Il est nécessaire d'implanter un service au niveau de l'application COPEN mais parfois, comme pour l'édition de patrons qui est une personnalisation COPEN, un service au niveau Alfresco doit aussi être implanté.

Un service est développé sous forme de webscripts : ensembles structurés de fichiers permettant de séparer le modèle, la vue et le contrôle conformément au modèle MVC. L'avantage du développement de services par webscripts est notamment :

- Le traitement de données plus important effectué côté client permettant de limiter les échanges avec le serveur.
- L'implantation est directement prise en compte, il n'est pas nécessaire de redéployer l'application pour que les modifications soient prisent en compte.

Un exemple de webscript situé au niveau de l'application COPEN est présenté ci-dessous. Il permet l'édition des rubriques de langage pour un patron. Tout d'abord voici l'arborescence d'un webscript en figure 33.

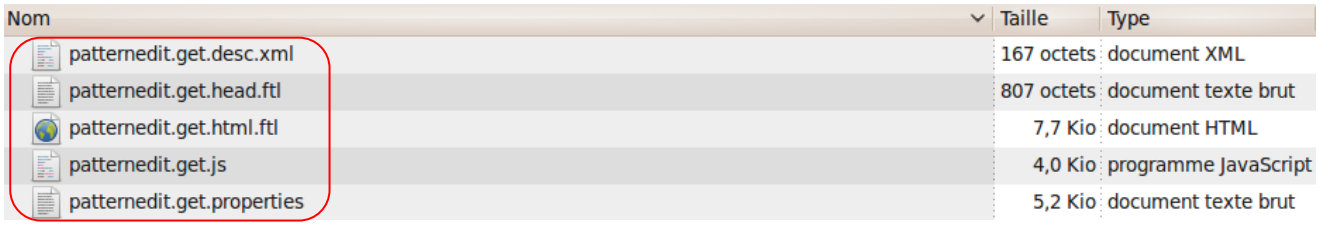


Figure 33 : Arborescence d'un webscript

Cette arborescence est structurée en plusieurs parties (une description, un contrôleur, une vue, un modèle) importés grâce à un fichier « head ».

Un webscript possède tout d'abord une description xml (figure 34) : patternedit.get.desc.xml

```
<webscript>
  <shortname>Edit a pattern</shortname>
  <description>Edit pattern form</description>
  <url>/components/Copen/pattern/patternedit</url>
</webscript>
```

Figure 34 : Description d'un webscript

Cela lui permet d'être identifié et utilisé. Son utilisation commence par la partie contrôle en JavaScript (figure 35) : patternedit.get.js.

```
//COPEN: Récuperation du contenu de chaque attribut
var attribute = "";
for(var i=0; i<attributes.length;i++)
{
        attributesValues[i] = new Array();
        for(var j=0; j<attributes[i].length;j++)
        {
                if(attributes[i][j].indexOf('-') != -1){
                        attribute = attributes[i][j].substring(0,attributes[i][j].indexOf('-'));
                        result = connector.call("/api/path/content/workspace/SpacesStore/Sites/"+url+"/contentPattern/"+attribute);
                        if (result.status == 200){
                                attributesValues[i][j]=
                                        connector.get("/api/path/content/workspace/SpacesStore/Sites/"+
                                                url+
                                                "/contentPattern/"+attribute);
                        }
                        else{
                                attributesValues[i][j]="";
                        }
                }
                else{
                        attributesValues[i][j]="";
                }
        }

}
```

Figure 35 : Partie Contrôle d'un webscript

Celle-ci permet de récupérer des objets et de contrôler leur valeur, ici celles des rubriques du langage. Cette partie accède à l'API Alfresco afin d'obtenir les objets et informations nécessaires à l'accès à la vue (figure 36) : patternedit.get.html.ftl

```
<!-- COPEN: affichage de l'attribut solution
                        uniquement pour les membres de la communauté ou du patron-->
<#if attribute?contains("Solution")
                        && isMember != "200" && isMemberOfCommunity != "200">
        <div class="detail-list-item ">
        <div class="toolbar flat-button theme-bg-2" align="center">
           <span ><h2>${msg("language.attribut.${currentAttribute}")}</h2></span>
        </div>
        <p>${msg("text.solution")}</p>
  </div>
<#else>
        <!-- COPEN: Affichage des attributs de type text -->
        <#if currentType == "text">
          <div class="detail-list-item ">
                <div class="toolbar flat-button theme-bg-2" align="center">
                   <span >
                        <h2>${msg("language.attribut.${currentAttribute}")}</h2>
                   </span>
                </div>
                <p>${attributesValues[i][j]}</p>
          </div>
        </#if>
        <!-- COPEN: Affichage des attributs de type text+img -->
        <#if currentType == "img" || currentType == "text+img">
           <div class="detail-list-item ">
                <div class="toolbar flat-button theme-bg-2" align="center">
                   <span>
                        <h2>${msg("language.attribut.${currentAttribute}")}</h2>
                   </span>
                </div>
                <#if currentType == "text+img">
                        <p>${attributesValues[i][j]}</p>
                </#if>
                <p><a href="${url.context}/proxy/alfresco/api/path/content/
}/contentPattern/${currentAttribute}img" target="_blank">
                        <img class="photoimg"
                             src="${url.context}/proxy/alfresco/api/path/content/
}/contentPattern/${currentAttribute}img"
                             alt=""
                             height="500%"
                             width="100%"
                        />
                </a></p>
           </div>
```

Figure 36 : Partie Vue d'un webscript

La vue permet d'afficher les informations de la façon souhaitée comme par exemple les rubriques de langage du patron suivant leur type. Elle utilise pour cela un moteur de modèle Freemarker contenant du Java et JavaScript pour le traitement des données (figure 35) et du langage HTML ou du JSON pour leur affichage dans un fichier de type « ftl » (figure 36Figure 36). Une

externalisation des labels de la vue est aussi disponible (figure 37Figure 37) : patternedit.get.properties.

```
label.majorVersion=Major Version
label.comments=Comments

message.uploading=Document is being uploaded...
message.success=Document successfully uploaded
message.failure=Document could not be uploaded
message.failure.413=Quota Exceeded

############## language ######################
language.type.text=Text
language.type.img=Image
language.type.text+img=Text & Image
language.type.keywords=Key words
language.type.links=Pattern Relation
language.categorie.general= General Informations
language.categorie.realisation=Realisation Informations
language.categorie.relation=Relations :
language.attribut.classification=Classification :
language.attribut.context=Context :
language.attribut.problem=Problem :
language.attribut.strength=Strength :
language.attribut.procedureSolution=Procedure Solution :
language.attribut.modelSolution=Model of the Solution :
language.attribut.applicationExamples=Application Examples :
language.attribut.applicationConsequences=Application Consequences :
language.attribut.use=Use :
language.attribut.refine=Refine :
language.attribut.need=Need :
language.attribut.alternative=Alternative :
```

Figure 37 : Externalisation des labels d'une vue d'un webscript

Tous les labels de la page, comme les rubriques, sont répertoriés ici, cela permet de changer une valeur sans entrer réellement dans le code source mais permet aussi une future internationalisation. La vue importe un modèle pour son affichage (figure 38) : patternedit.get.head.ftl.

```
<#include "../../component.head.inc">
<!-- Pattern edit -->
<@script type="text/javascript" src="${page.url.context}/components/Copen/pattern/patternedit.js"></@script>
<@link rel="stylesheet" type="text/css" href="${page.url.context}/components/documentlibrary/toolbar.css" />

<@link rel="stylesheet" type="text/css" href="${page.url.context}/components/upload/html-upload.css" />
<@script type="text/javascript" src="${page.url.context}/components/upload/html-upload.js"></@script>
<@script type="text/javascript" src="${page.url.context}/components/upload/flash-upload.js"></@script>
<@link rel="stylesheet" type="text/css" href="${page.url.context}/components/upload/flash-upload.css" />
<@script type="text/javascript" src="${page.url.context}/components/upload/file-upload.js"></@script>
```

Figure 38 : Partie Import d'un webscript

On remarque l'import patternedit.js, celui-ci permet d'instancier des boutons ou des éditeurs spécifiés dans notre vue. Il permet aussi en règle générale d'éviter les échanges client-serveur grâce à des technologies comme AJAX (Asynchronous Javascript and XML) disponibles dans une bibliothèque YUI (Yahoo User Interface). Cette bibliothèque offre de nombreux modules tels que des « rich editors ». Un exemple est la possibilité d'activer ou désactiver les validations de boutons suivant le remplissage des champs d'un « rich editor » et ce, sans échange avec le serveur. Un modèle importé est décrit en figure 39 : patternedit.js.

```
if(typeAttr == "text" || typeAttr == "text+img" || typeAttr == "keywords"){
        editors[i] =
        this.widgets.editor = new Alfresco.util.RichEditor(
                                            Alfresco.constants.HTML_EDITOR,
                                            this.id + '-' + attr[i]
                                            , this.options.editorConfig);
         this.widgets.editor.render();
         editors[i] = this.widgets.editor;
         // Add validation to the rich text editor
         this.widgets.validateOnZero = 0;
         var keyUpIdentifier =
                (Alfresco.constants.HTML_EDITOR === 'YAHOO.widget.SimpleEditor') ? 'editorKeyUp' : 'onKeyUp';
         editors[i].subscribe(keyUpIdentifier, function (e)
           {
                this.widgets.validateOnZero++;
                YAHOO.lang.later(1000, this, this.validateAfterEditorChange());
           }, this, true);
}

if(typeAttr == "img" || typeAttr == "text+img"){
        // FileUpload button
        this.widgets.FileUploadButton =
                        Alfresco.util.createYUIButton(this, "FileUpload-button", this.onFormFileUploadButtonClick,
         {
            disabled: false,
            value: "create"
         });
         this.widgets.upload = Alfresco.util.createYUIButton(this, "button-upload-"+attr[i]+"img", this.onUpload);
}
```

Figure 39 : Partie Modèle d'un webscript

Il montre l'instanciation des éditeurs de texte et des boutons de téléchargement d'image lors de l'édition de patrons.

De manière purement quantitative, COPEN a nécessité :

- de créer dans COPEN environ vingt webscripts comprenant chacun en moyenne 400 lignes de code.
- de retoucher dans COPEN une trentaine de webscripts existants.
- De créer un webscript dans Alfresco et dans réadapter quatre autres.

Il est intéressant de souligner que le choix Alfresco correspond de près au besoin. COPEN s'adapte sans avoir à apporter de personnalisation dans le cœur de l'outil Alfresco même au niveau des webscripts.

Il est important de remarquer que l'architecture mise en place permet une mise à jour facile de l'application COPEN que ce soit par du développement des services uniquement ou de l'intégration d'une nouvelle version.

2.3. Environnement de gestion de projets Agile

Si la collaboration est au cœur de l'outil COPEN, elle est aussi au cœur de la gestion de projets choisie pour le développer. C'est en toute cohérence que la méthode Agile SCRUM fusionnée aux « best practices » d'XP est choisie pour développer cet environnement COPEN.

2.3.1. Mise en place de la méthode XP@SCRUM

L'équipe est assez restreinte, composée de quatre personnes :

- Deux membres prendront le rôle de « product owner ».
- Une personne (moi) est présente à la fois en tant que « scrum master » et « développeur ».
- Une dernière personne participera au développement à temps plein afin de répondre au sujet de son mémoire CNAM qui est l'import/export de patrons via COPEN.

L'utilisation d'un outil de gestion de projets est essentielle au bon déroulement du projet. C'est la solution IceScrum spécifiée en figure 15 de l'état de l'art qui est choisie. Elle permet de créer un compte pour chaque membre de l'équipe avec le rôle qui lui correspond. L'avantage est que chacun met à jour les tâches qu'il s'est affecté durant le daily scrum.

Pour se faire, l'ensemble de l'équipe doit savoir interpréter et utiliser le logiciel. Une formation est donc effectuée pour chacun des membres par le « scrum master ». En plus d'une présentation générale, le client est formé sur sa tâche de « product owner » et le développeur sur la sienne afin de pouvoir être autonome sur le logiciel.

La familiarisation avec l'outil permet d'accéder à la tâche suivante qui est la définition des fonctionnalités caractérisant l'outil à développer. C'est lors d'une réunion réunissant tous les membres (product owner, scrum master et développeur) qu'est déterminé le périmètre de la solution à développer. Toute l'équipe connaissant IceScrum, c'est à l'aide de cette solution qu'est identifiée chaque fonctionnalité sous forme de « *feature* » Agile.

Les sections suivantes illustrent l'utilisation de la méthode XP@SCRUM pour notre projet de développement de l'outil COPEN.

2.3.2. Définition des fonctionnalités de COPEN

La figure 40 présente le périmètre de COPEN sous forme de « *features* » IceScrum.

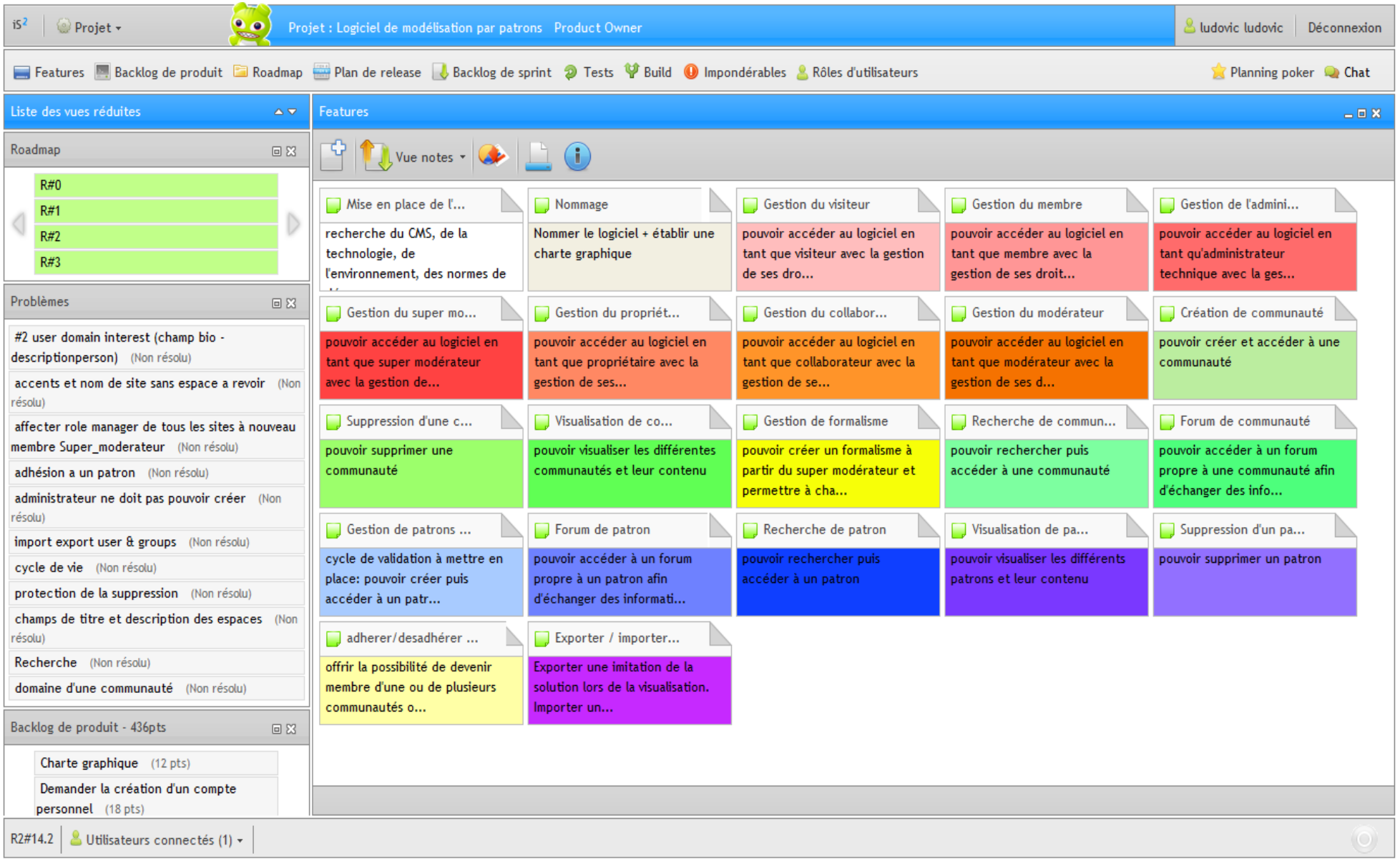


Figure 40 : Liste des fonctionnalités caractérisant COPEN

Vingt deux *features* définissent COPEN. Elles possèdent une description textuelle, un ordre d'affichage suivant leur priorité mais aussi une couleur.

Cinq déclinaisons de couleur sont visibles :

1. Blanc : *Features* essentielles au lancement du développement.
2. Rouge : *Features* permettant la gestion des membres COPEN.
3. Vert : *Features* permettant la gestion des communautés.
4. Jaune : *Features* liées aux communautés et aux patrons.
5. Bleu : *Features* permettant la gestion des patrons.

Chacune des *features* sera décomposée en *user stories* qui seront embarquées lors des différents *sprints*. Ces *user stories* héritent de la couleur affectée aux *features*. Leur nombre étant plus élevé, c'est important à ce niveau là de pouvoir cerner l'appartenance de chaque story à une *feature*. Une gestion de projets claire et soignée est essentielle au démarrage d'un projet et l'est d'autant plus au fil des itérations.

2.3.3. Organisation en s*prints*

Il est important d'avoir une cohérence dans les itérations, chaque *sprint* doit être structuré de la même façon pour un travail en équipe efficace.

2.3.3.1. Définition d'un *sprint*

Il demande des pré-requis :

- Le *sprint* précédent est terminé et livré.
- Les prochaines *user stories* les plus prioritaires sont spécifiées, estimées en heures, validées par le « product owner » et assez nombreuses pour couvrir le *sprint*.

Il possède des « technical » stories obligatoires à embarquer :

- Spécifier assez de *user stories* pour préparer le *sprint* prochain.
- Inviter les membres pour les prochaines réunions (démonstration, revue, rétrospective) et leur envoyer un bilan à la fin de chacune.
- Faire une démonstration au reste de l'équipe (product owner, futurs utilisateurs ou autre « stackholder ») du lot livré en fin de *sprint*.
- Faire une revue et une rétrospective en fin de *sprint*.

D'une durée de trois semaines, un *sprint* permet d'embarquer un certain nombre de *user stories* spécifiées et prêtes à être développées.

2.3.3.2. Définition d'une *user story* COPEN

Une « *user story* » possède plusieurs caractéristiques telles qu'une description textuelle et une priorité. Il a été choisi pour COPEN de lui associer encore quatre tâches chiffrées en heures :

- La rédaction des tests
- Le codage et les tests unitaires
- Le passage des tests
- La rédaction de la documentation

Ces tâches permettent de diriger le développement et de ne négliger aucune phase d'implantation de la *user story*.

Chacune des *features* est détaillée par un ensemble de *user stories*. La *feature* « Gestion de patrons » (figure 41) est présentée à titre d'exemple afin de comprendre la décomposition en *user stories*.

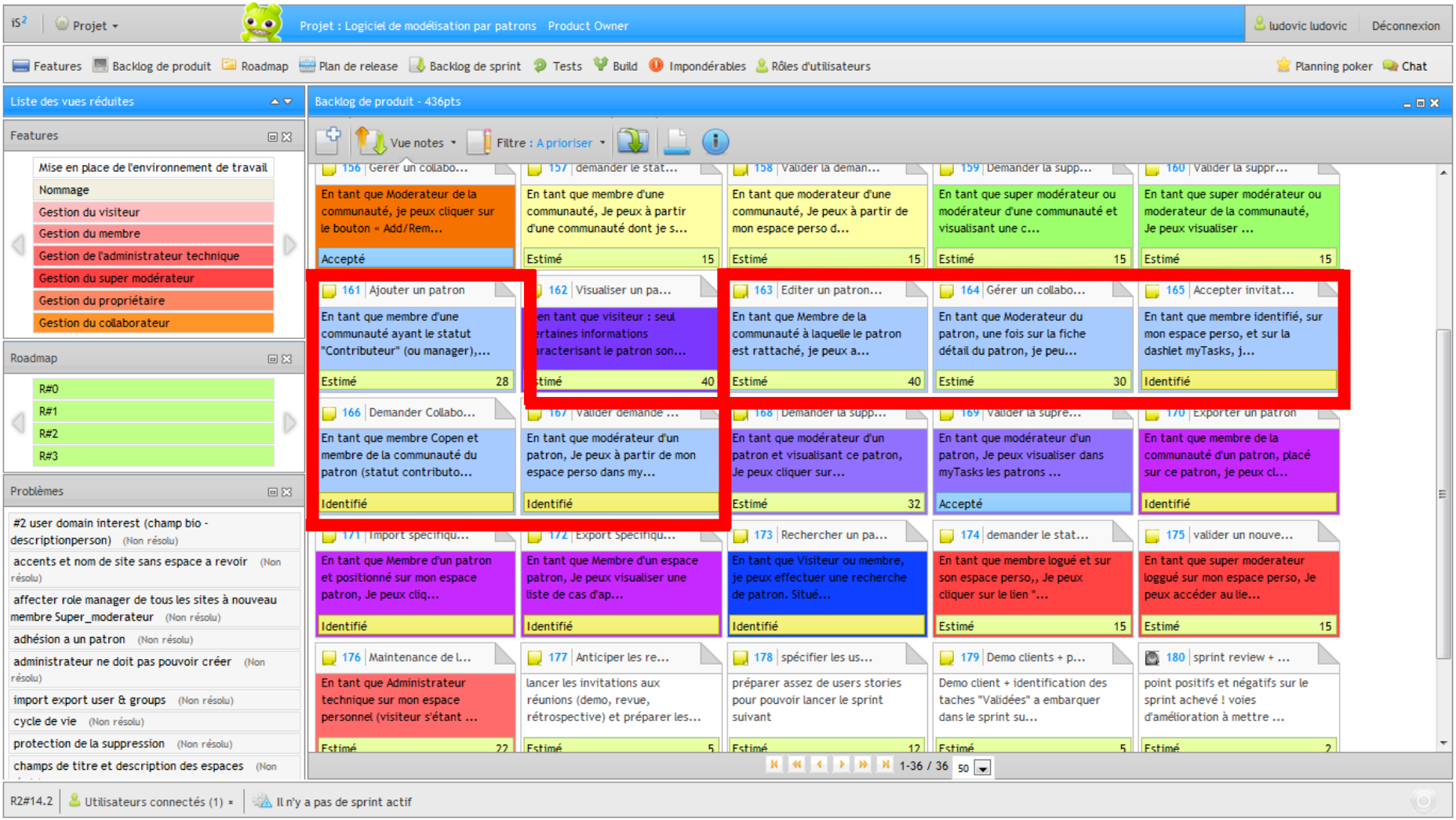


Figure 41 : user stories associées à la feature « gestion de patrons »

Chaque story est ensuite embarquée dans un *sprint* suivant sa priorité. La figure 42 présente la story « Editer un patron collaboratif » embarquée au sein du *sprint* S#8.

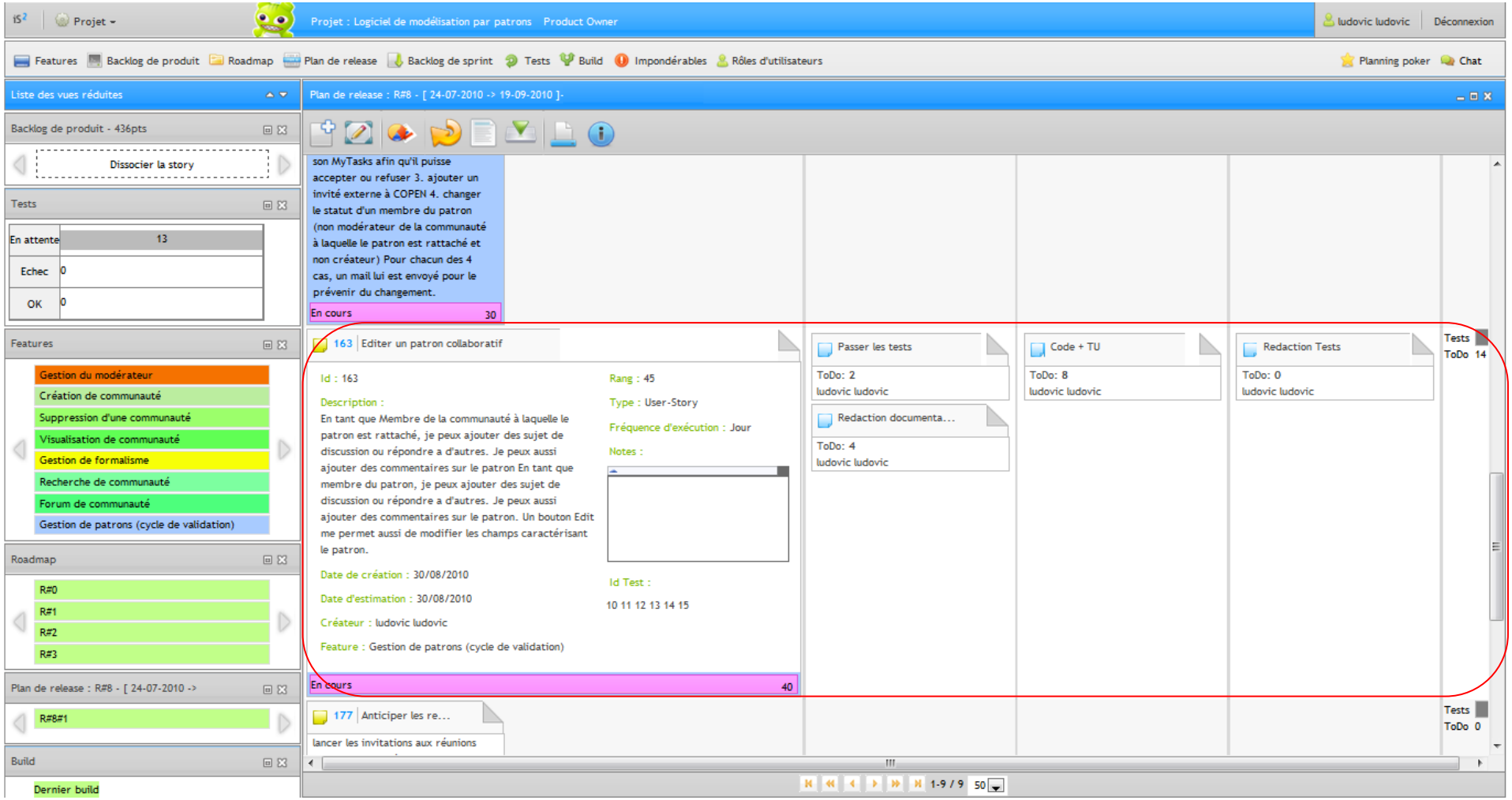


Figure 42 : user story « Editer un patron collaboratif »

Cette story contient une description textuelle, l'association à une *feature*, un rang de priorité, quatre tâches (avec l'estimation en heure du reste à faire) ainsi que des tests associés.
Une fois le développement de la story effectué, ces tests sont passés afin de vérifier que le développement corresponde exactement aux exigences du client. Une illustration d'un cas de test pour la story « Editer un patron collaboratif » est disponible ci-dessous en figure 43.

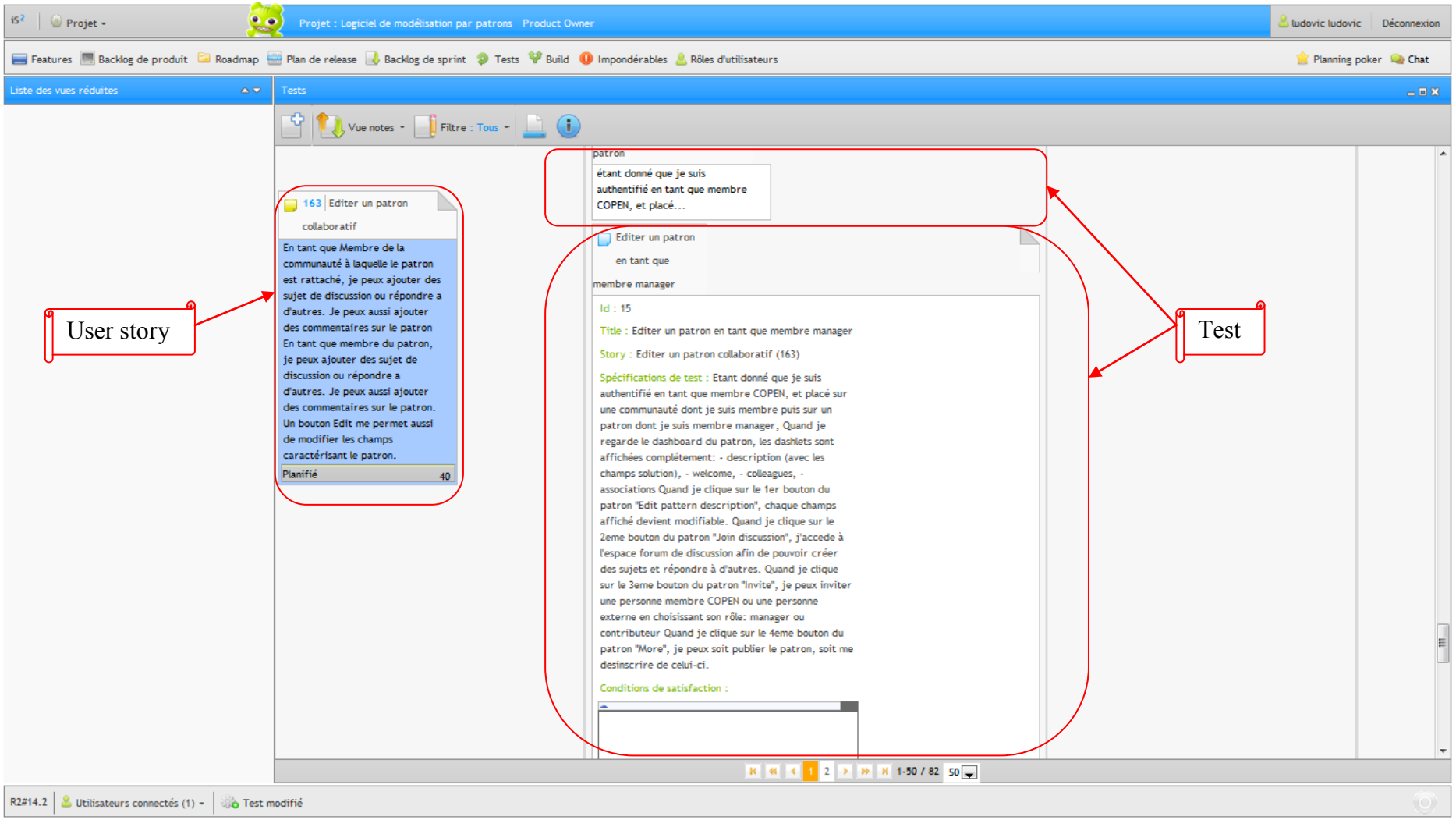


Figure 43 : Test « Editer un patron en tant que membre manager »

C'est une fois que sont validés l'ensemble des tests des *user stories* d'une feature que celle-ci est considérée comme terminée.

Ces *user stories* sont traitées lors d'un *sprint* une fois définies, structurées, validées et chiffrées. Un *sprint* ne contient qu'un nombre limité de *user stories*, un calcul spécifique est donc à l'origine du nombre de *user stories* embarquées.

2.3.3.3. Capacité d'un *sprint* en *user stories*

Deux chiffres impactent la composition d'un *sprint* :

- Le nombre total de jours de travail sur les 3 semaines que compte le *sprint*.
  *(1 développeur à 100% + 1 développeur à 75%) * nb jours travaillés = nb jours total*

- Le coefficient d'efficacité : il est estimé qu'un développeur dans une journée de travail n'est disponible qu'à 80% de son temps sur la *user story* qu'il s'est affecté.
  *7 heures d'une journée * 80% = 5,6 heures de travail effectif sur COPEN*

La multiplication du nombre de jours total de travail au nombre d'heures effectives permet de connaître le nombre d'heures caractérisant le *sprint*. Il suffit ensuite d'embarquer les *user stories* les plus prioritaires jusqu'à couverture de ce nombre d'heures.

Chaque *sprint* permet d'embarquer un nombre de *user stories* significatif permettant une évolution notable des fonctionnalités de l'outil. IceScrum permet de visualiser la feuille de route du projet (roadmap) afin de retrouver ce qui a été développé au fur et à mesure des *sprints*.

2.3.3.4. Roadmap

Voici en figure 44 la roadmap complète du projet COPEN, celle-ci sera vue en détail par la suite.

Figure 44 : Roadmap

Celle-ci est constituée de neuf sections représentant les neuf *sprints* de trois semaines effectués lors de ce projet. Chacun d'entre eux a un objectif et celui-ci est démontré en fin de *sprint*. Pour illustrer l'avancement du projet et de ses *sprints* une autre visualisation du projet a été mise en place : la visualisation sous forme de post-its. Une photo du bureau de l'équipe en fin de projet disponible en figure 45 montre la gestion du projet mise en place.

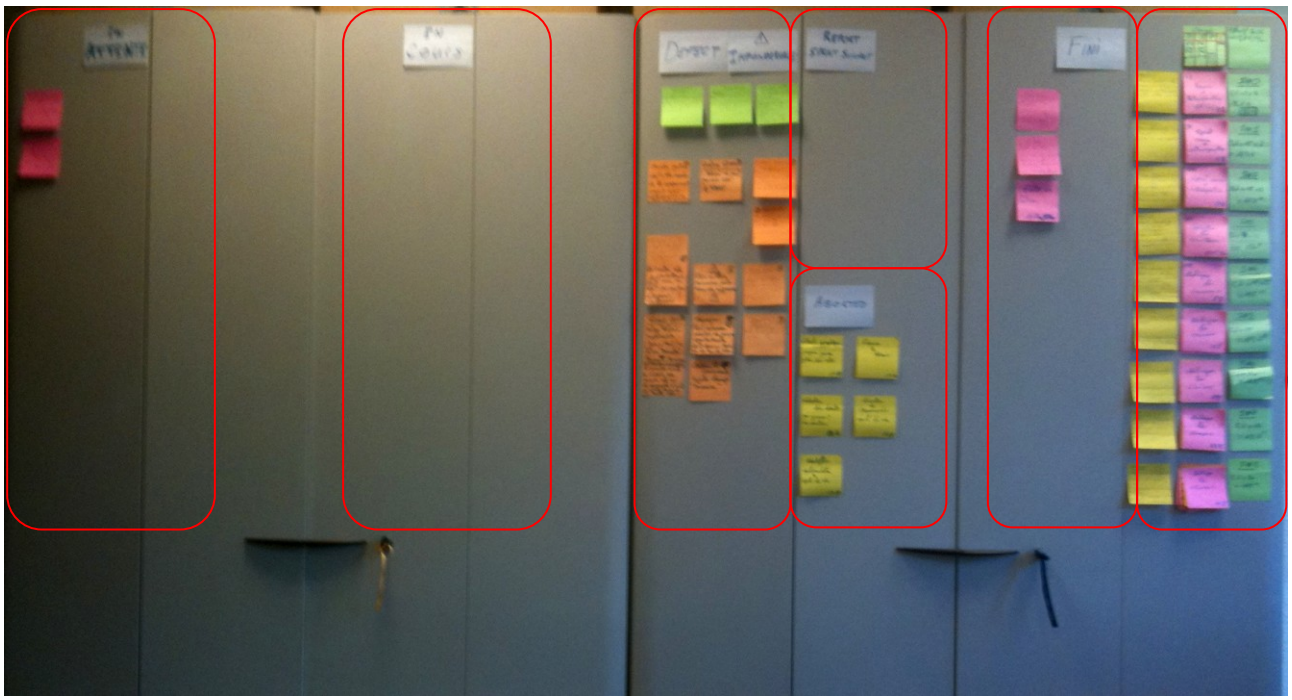


Figure 45 : Tableau de post-its en fin de projet

Cette modélisation du projet permet d'apporter une motivation supplémentaire et soude chaque jour un peu plus l'équipe avec l'avancement du développement. Six parties caractérisent le projet. De gauche à droite il y a les tâches de « *user stories* » en attente puis celles en cours. Ensuite les impondérables à traiter dès que possible, les tâches reportées au *sprint* suivant et en dessous les tâches abandonnées devenues obsolètes. Les deux dernières colonnes représentent tout ce qui est fini : les tâches terminées du *sprint* en cours puis les *sprints* achevés. Chacun s'investit donc dans le projet au quotidien en déplaçant la tâche qu'il s'est affecté suivant son avancement. La communication est favorisée ainsi que l'esprit d'équipe.

La communication, l'avis de chacun et l'investissement personnel nourrissent la gestion du projet. Chaque *sprint* a permis une progression et une amélioration continue dans l'organisation du projet.

2.3.4. Déroulement des *sprints*

Certaines fonctionnalités sont essentielles pour le développement des suivantes. Les *sprints* doivent donc intégrer ces priorités et ordonner les fonctionnalités.

Voici les quatre parties caractérisant l'ordre de développement durant les neuf *sprints* du projet :

- *Sprints* S#0 et S#1 : Mise en place de l'environnement et formation des développeurs
- *Sprints* S#2 et S#3 : Gestion de l'espace personnel
- *Sprints* S#4 et S#5 : Gestion des communautés
- *Sprints* S#6, S#7 et S#8 : Gestion des patrons et des impondérables urgents

Chacune apporte des fonctionnalités, des retours d'expérience et des bonnes pratiques à mettre en œuvre lors du *sprint* suivant.

Une description, un résultat ainsi qu'un bilan permet d'analyser chaque partie. La mise en place de l'environnement en est le premier exemple car tout le développement COPEN est fondé sur celui-ci.

2.3.4.1. Mise en place de l'environnement et formation

Les deux premiers *sprints* S#0 et S#1 ayant permis la mise en place de l'environnement sont décrits en figure 46.

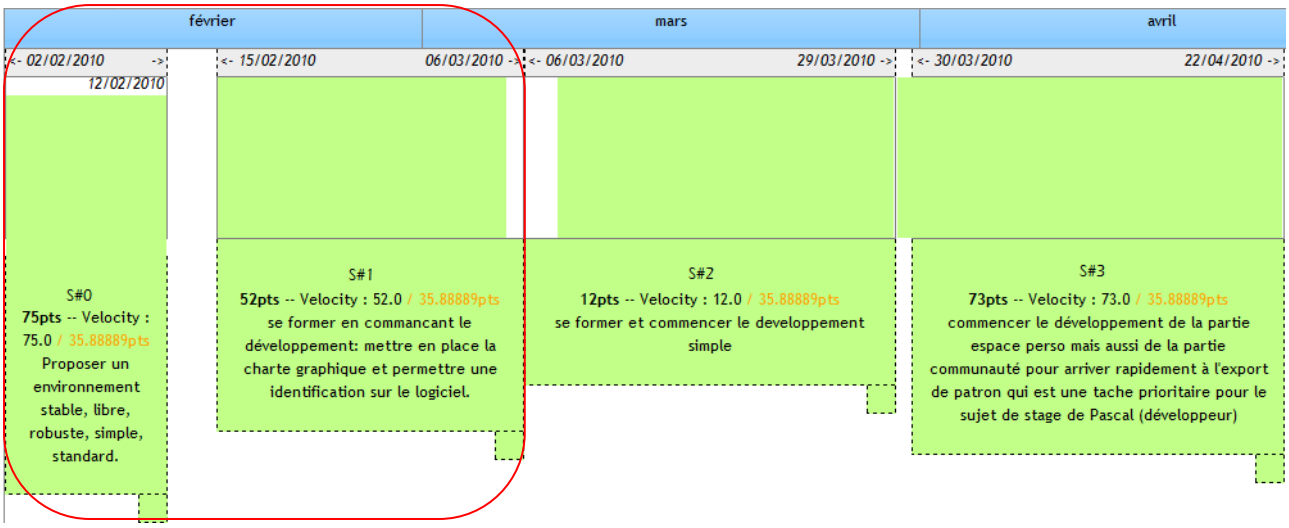


Figure 46 : Caractéristiques des sprints S#0 et S#1

1) Description

Préparer un environnement de développement et monter en compétences sont deux tâches réparties entre les deux développeurs. Chacun progresse dans sa tâche tout en formant le deuxième développeur sur son avancement. Cela permet un travail collaboratif mais aussi d'avoir son avis sur des points bloquants.

Installer un environnement ou commencer le développement à titre de formation implique de comprendre la structure des fichiers et comment l'aborder (cf. annexe A). Les méthodes de déploiements sont multiples, il n'est pas évident de trouver laquelle utiliser afin de correspondre au besoin en développement. Il est également difficile de s'appuyer sur de la documentation, quasi inexistante vu qu'Alfresco Share n'en n'est qu'à sa première version. La recherche d'aide au développement fut une orientation choisie lors de ce *sprint*. Peu de personnes travaillant sur Alfresco, il est difficile de trouver de l'aide. Cependant la persévérance dans les recherches a porté ses fruits. Grenoble Université a lancé un projet de développement d'une application de partage d'informations basée sur Alfresco. Une relation a donc été construite afin d'échanger et partager des connaissances. Il est par la même occasion très intéressant de comprendre la structure de développement choisie par cette équipe afin de finaliser la mise en place de celle du projet COPEN.

Ce *sprint* permet donc d'établir une relation de collaboration avec une autre équipe de développement Alfresco, de mettre en place un environnement de travail efficace ainsi que de se former sur la technologie Alfresco.

2) Résultat

Un environnement de projets Eclipse de type Maven configuré sous CVS est illustré par la figure 47.

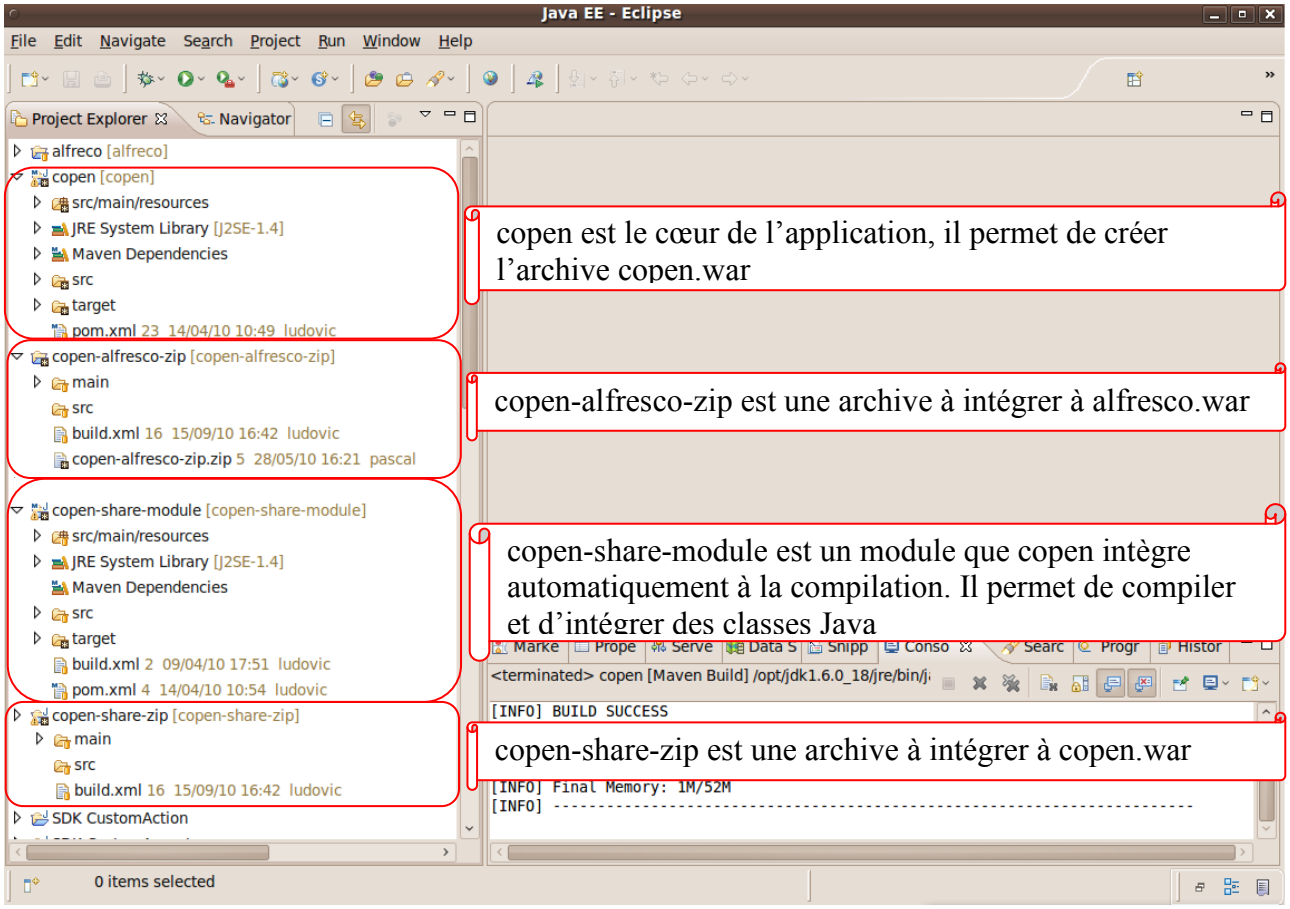


Figure 47 : Environnement Eclipse

Le développent s'effectue grâce à Eclipse avec 4 projets synchronisés sous un Subversion (SVN). Afin de correspondre à la structure Alfresco et Alfresco Share, on intègre un plugin de déploiement Maven à nos projets qui les compile suivant un archétype spécifié (modèle général d'Alfresco et Alfresco Share) puis en intégrant le nouveau développement afin de donner l'archive COPEN-1.0. Un guide d'installation d'environnement est disponible en annexe B.

Un outil de gestion de projets Redmine, résumé en figure 48 permet un espace de partage commun aux membres de l'équipe.

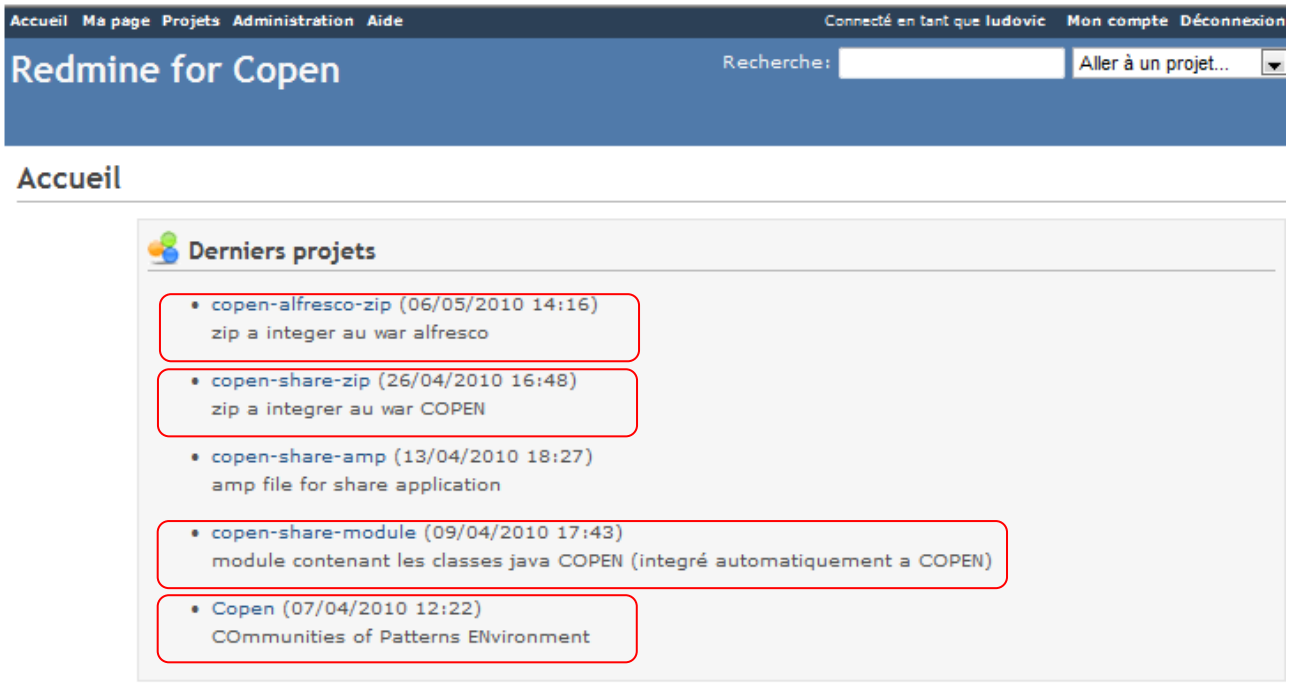


Figure 48 : Application de gestion de projets web Redmine

Un outil de gestion de projets web Redmine est installé. Il permet l'accès au SVN de chaque projet Eclipse ainsi qu'à des espaces de partage de documents facilitant le développement collaboratif. C'est donc sur cet espace commun qu'est effectuée la livraison du lot de chaque *sprint*.

La version COPEN du premier lot n'est autre qu'Alfresco Share déployé grâce à l'environnement tout juste configuré. Une illustration en figure 49 présente le résultat obtenu.

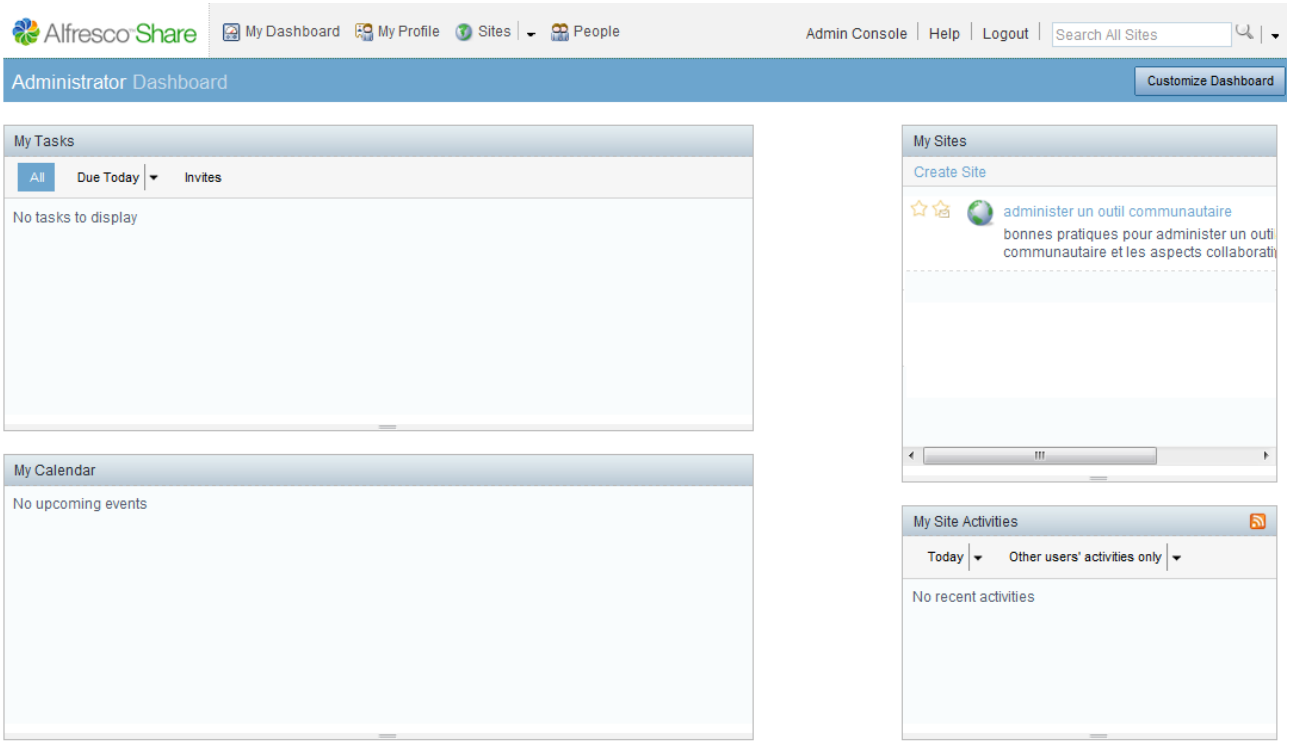


Figure 49 : Alfresco Share

L'environnement de développement créé permet de déployer COPEN. Aucun développement n'est fait à ce jour, c'est donc Alfresco Share classique qui est déployé sur le serveur. Un guide est décrit

en annexe B afin de déployer COPEN tel que l'illustre la figure 32 (Architecture des applications Alfresco et COPEN).

3) Bilan

Les améliorations à prendre en compte dans les *sprints* suivants sont :

1. Se limiter au périmètre de l'outil et épurer Alfresco Share qui contient beaucoup de fonctionnalités inutiles.
2. Communiquer la connaissance au sein de l'équipe au fur et à mesure de la formation.
3. Garder en tête l'importance du daily scrum qui reste la base de la communication SCRUM.
4. Prendre du temps pour se former, s'entraider et monter en compétence.
5. Se limiter à travailler sur une seule tâche à la fois pour être plus efficace.

Nous notons les pratiques Agile suivantes :

1. Recherche de personnes pouvant aider à débloquer une situation par le Scrum Master.
2. Soutien entre développeurs sur des tâches difficiles.
3. Livraison d'un premier lot.

2.3.4.2. Gestion de l'espace personnel

Les *sprints* S#2 et S#3 ayant permis la gestion de l'espace personnel sont illustrés en figure 50.

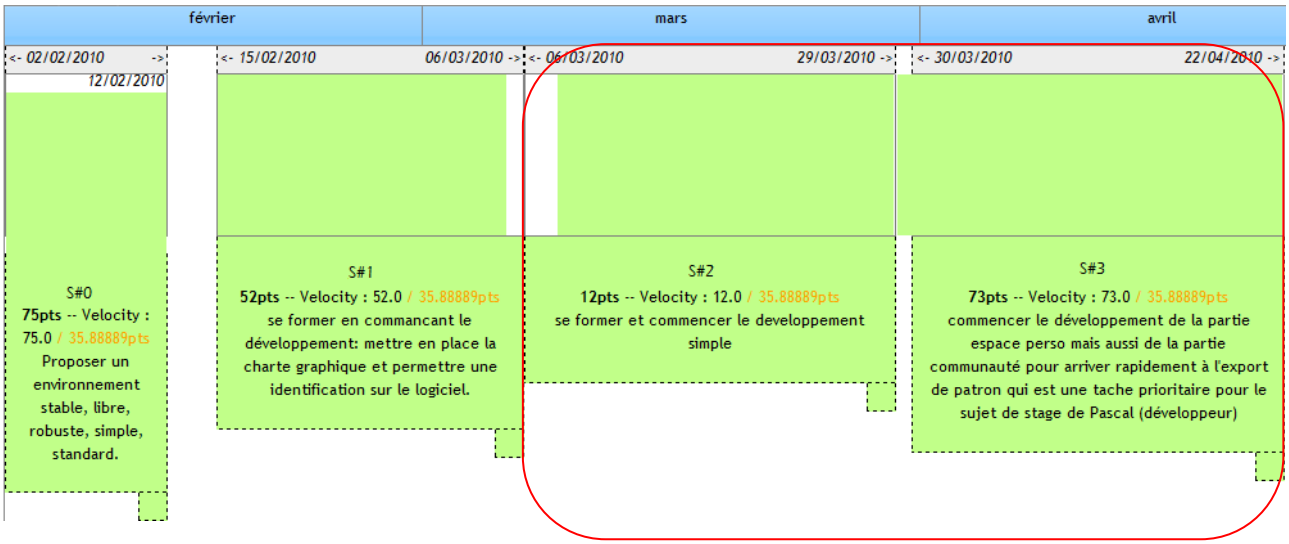


Figure 50 : Caractéristiques des sprints S#2 et S#3

1) Description

Maintenant que le déploiement est maîtrisé et structuré, les *sprints* S#2 et S#3 permettent de se concentrer sur le développement. La cible de ces *sprints* est l'accès à un espace personnel suivant les droits du membre après authentification. Le temps de formation durant les deux *sprints* précédents fut court, c'est donc au fil de l'eau que la prise de compétence Alfresco se fait. L'avancement n'est pas rapide et certains critères de *user stories* sont complexes à mettre en œuvre. Etant donné que le développement est ralenti par la formation continue des développeurs et que l'un d'entre eux doit attendre l'import/export de patrons COPEN, il a été décidé de suspendre quelques caractéristiques complexes facultatives au fonctionnement de COPEN. Celles-ci sont identifiées et placer dans la liste des impondérables du projet pour être traitées ultérieurement. Cette stratégie

permet à la fin de ces deux *sprints* d'accéder à l'espace personnel de chacun des membres à partir d'une page d'accueil en tenant compte de leurs droits.

2) Résultat

La page d'accueil est décrite en figure 51.

Figure 51 : Page d'accueil COPEN

La page d'accueil COPEN est caractérisée par une description ainsi qu'un espace d'authentification permettant l'accès à un espace personnel exposé en figure 52.

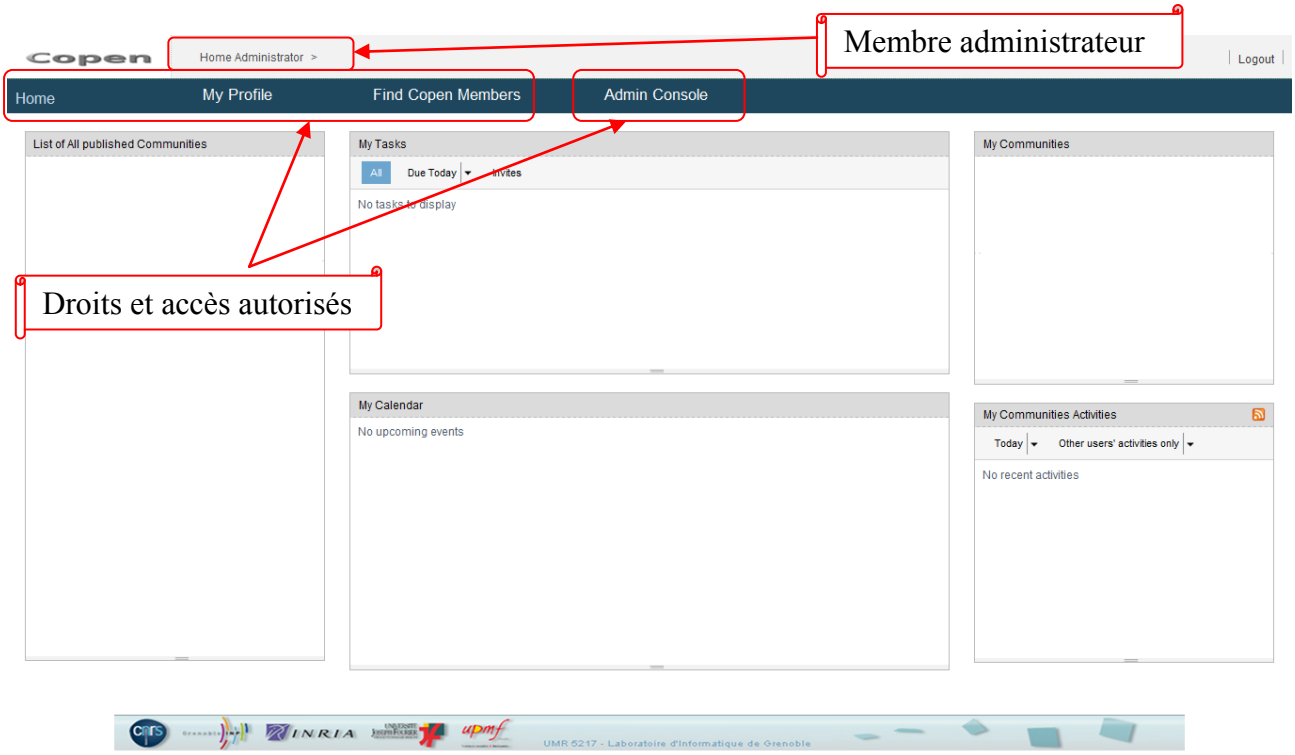


Figure 52 : Espace personnel COPEN

Cet espace personnel correspond à celui de l'administrateur, en plus des accès aux profils, il possède particulièrement le menu « Admin Console » permettant d'accéder aux fonctionnalités d'administration des membres COPEN.

3) Bilan

Les améliorations à prendre en compte dans les *sprints* suivants sont :

1. Prioriser les fonctionnalités, se focaliser sur les plus importantes et les répartir.
2. L'import/export de patrons doit être une des priorités.
3. Ne pas rester bloqué sur un développement, chercher de l'aide.
4. Trouver des solutions ou personnes ayant des compétences techniques pouvant répondre à nos besoins.
5. Utiliser l'espace commun mis en place avec les développeurs de Grenoble Université afin de pouvoir leur poser des questions et échanger.
6. Présence importante du client au daily scrum afin qu'il reste au centre du projet et qu'il interagisse.

Nous notons les pratiques Agile suivantes :

1. Recherche de solutions pouvant aider à débloquer une situation par le Scrum Master.
2. Travail en binôme sur des tâches difficiles.
3. Adaptation et priorisation du besoin face au contexte et aux contraintes (suspension de tâches).
4. Conventions de nommage COPEN.
5. Appropriation collective du code. Le développement n'est pas individuel mais appartient à une équipe soudée.

2.3.4.3. Gestion des communautés

Les *sprints* S#4 et S#5 ayant permis la gestion des communautés sont décris en figure 53.

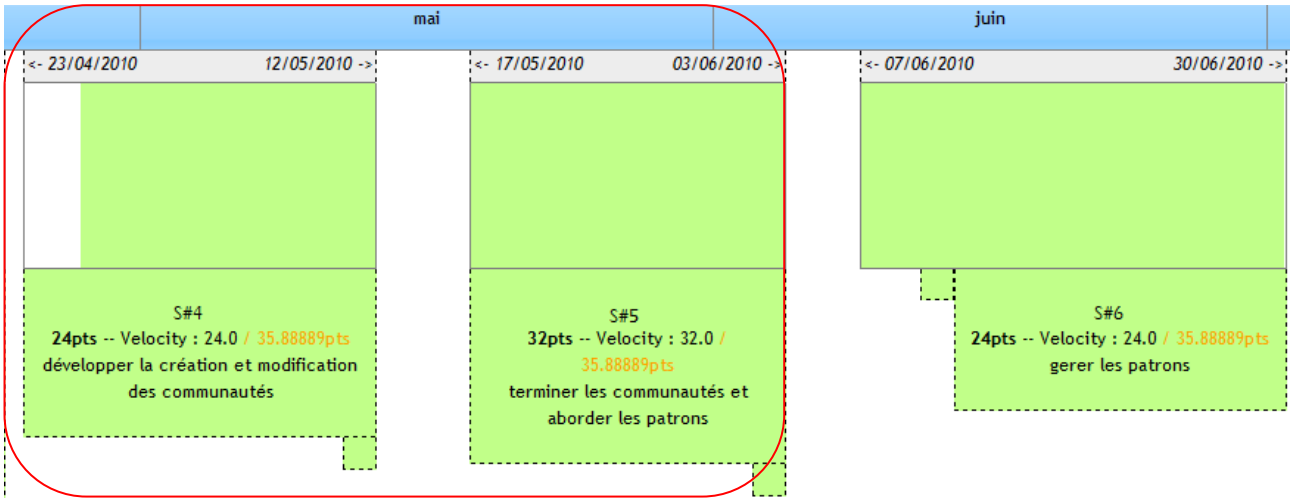


Figure 53 : Caractéristiques des sprints S#4 et S#5

1) Description

L'authentification permet de se connecter et d'accéder aux espaces personnels correspondants. La fonctionnalité suivante est la création de communauté. Ce sont donc les *user stories* de création de communautés qui sont embarquées lors de ces *sprints*. Il s'agit maintenant que chaque membre accède aux différentes communautés, et ce, avec la mise en place d'un cycle de vie cohérent. Chacun doit pouvoir jouer son rôle, collaborer et participer à l'évolution des communautés dont il est membre. Le chiffrage de ces *user stories* ne couvrent pas toutes les heures prévues pour ce *sprint*. Deux axes importants s'offrent : l'association dynamique d'un langage à la création de communautés ou la création de patrons. La priorité reste pour un des développeurs COPEN, la création de patrons afin d'arriver à son sujet de mémoire CNAM : l'import/export de patrons COPEN. Il est donc choisi en priorité de commencer la création de patrons au sein de communautés et de laisser de côté la mise en place du langage pour le moment. L'équipe prend de l'assurance au fil des *sprints* et avance à chaque fois plus efficacement. C'est dans cette optique-là qu'une nouvelle forme de retour et de bilan du *sprint* passé a été mise en place : une rétrospective en fin de *sprint* définissant les points positifs et les points négatifs les plus marquants apporte une approche motivante à l'amélioration continue au sein de l'équipe.

2) Résultat

L'espace communautaire est maintenant accessible comme le montre la figure 54.

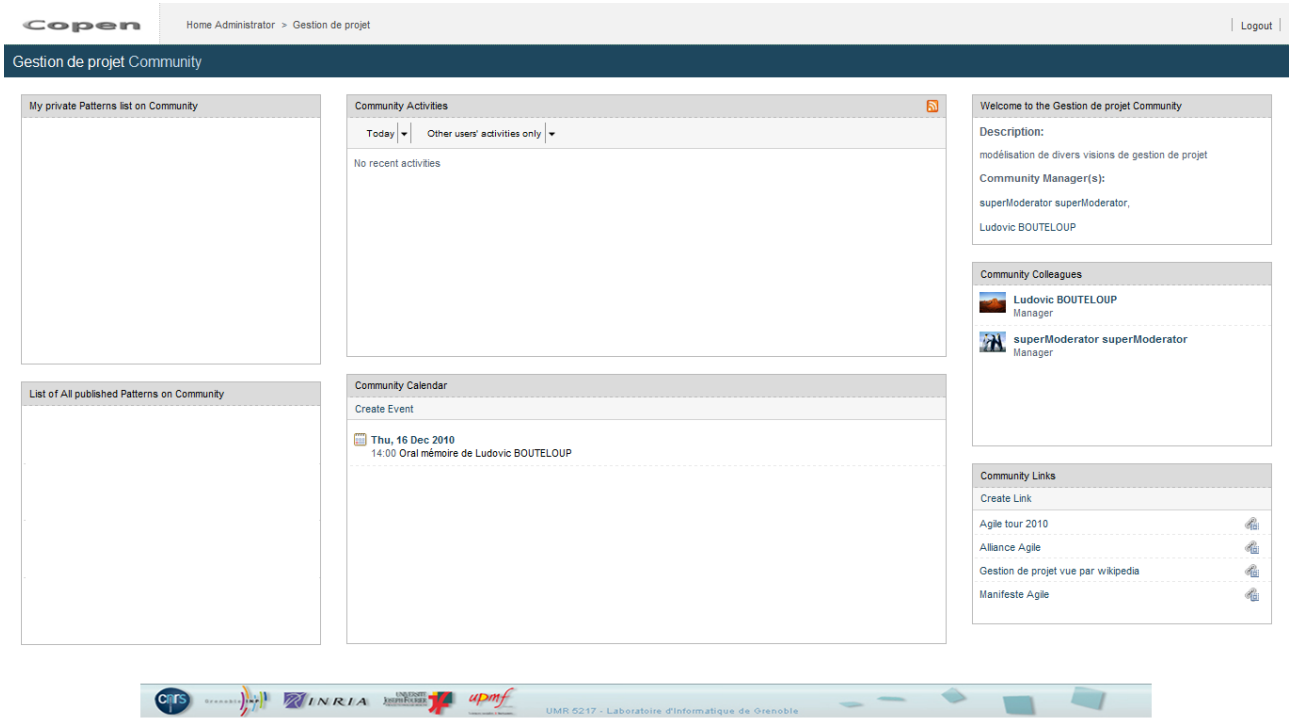


Figure 54 : Espace communautaire COPEN

La création de communautés est disponible. L'espace d'une communauté présenté illustre la description de celle-ci.

A gauche la liste des futurs patrons sera accessible, en haut les patrons privés, en bas les patrons publics.

Au milieu, deux espaces permettent de spécifier les activités sur la communauté. En haut les activités impactant directement la communauté comme une réponse sur le forum. En bas, l'annonce d'événements intéressants sous forme de calendrier.

La partie de droite décrit tout en haut la communauté. Au milieu, ce sont ses membres affichés sous forme de liste et en bas des liens intéressants à l'épanouissement de celle-ci.

3) Bilan

Rétrospective des *sprints* effectués :
+ : Vision plus claire du travail à réaliser en fin de *sprint*.
+ : Satisfaction globale de l'équipe, meilleure maîtrise d'Alfresco.
- : Panique : une période d'avancement tête baissée dans le développement sans prendre de recul sur le peu de quantité de travail produit. Il aurait fallu réagir plus tôt.
- : Ne pas sous-estimer l'importance du traçage des points bloquants pour la future correction.

Les améliorations à prendre en compte dans les *sprints* suivants sont :

1. Alfresco Share doit être personnalisé, tout doit devenir COPEN (messages d'erreurs etc.).
2. Se contenter de faire ce que l'on sait faire, il faut avancer dans les fonctionnalités !
3. Lister les points bloquants pour la suite du projet.
4. Se concentrer sur le cœur du logiciel : les patrons.
5. Ne pas négliger le projet en ne lui apportant que du développement, tous les livrables sont importants.

Nous notons les pratiques Agile suivantes :

1. Avancement des *user stories* revu régulièrement pour s'adapter aux priorités.
2. Intégration continue.
3. Relecture de code et Refactoring.

2.3.4.4. Gestion des patrons et des impondérables urgents

Les *sprints* S#6, S#7 et S#8 utiles au développement de la gestion de patrons et des impondérables urgents sont énumérés en figure 55.

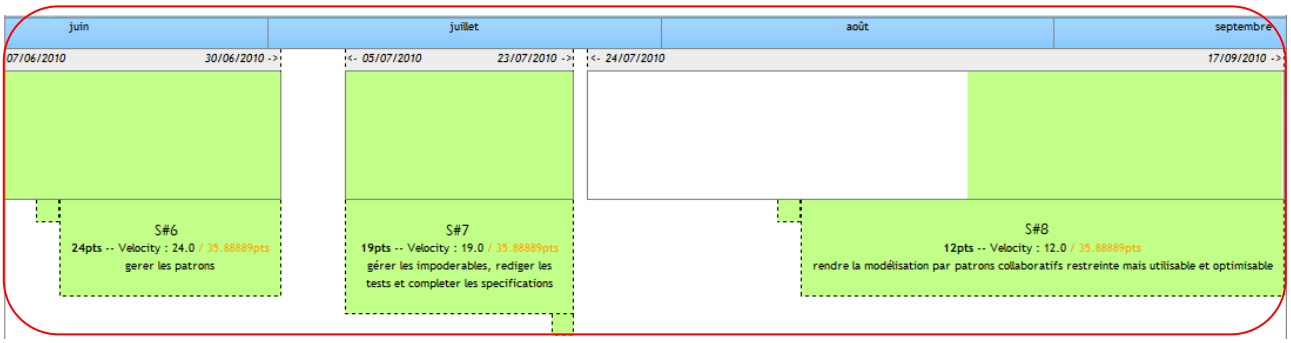


Figure 55 : Caractéristiques des sprints S#6, S#7 et S#8

1) Description

Ces trois *sprints* sont les derniers du projet. Aller à l'essentiel est à ce moment là encore plus significatif. Le *sprint* S#6 va permettre la mise en place du cycle de vie du patron et l'édition des rubriques (figé, sans langage de communautés pour le moment). Il est primordial d'accéder rapidement à la modélisation par patrons afin que l'import/export soit accessible au développeur pour clore son mémoire. Le *sprint* S#7 permet quant à lui de gérer les points importants non traités lors de la rapide avancée vers les patrons. Les fonctionnalités comme l'association d'un langage propre à une communauté ou le tri dans les données des différents espaces à afficher (personnel, communauté et patron) sont embarquées. La concrétisation des fonctionnalités de l'outil permet aussi d'ajouter une tâche de documentation afin de mettre à jour les spécifications existantes. C'est lors de ce *sprint* S#7 qu'un serveur distant a été fourni, une tâche de déploiement et tests sur ce serveur fut intéressante et permit de trouver des bugs d'interprétation par les différents navigateurs web. Ensuite le *sprint* S#8, dernier *sprint* du projet, a permis de clôturer le développement logiciel en repassant les tests et en traitant les petits détails importants à la mise en production de l'outil.

2) Résultat

Le premier point essentiel caractérisant l'outil est le choix du langage propre à une communauté, donc à tous les patrons la caractérisant (cf. figure 56).

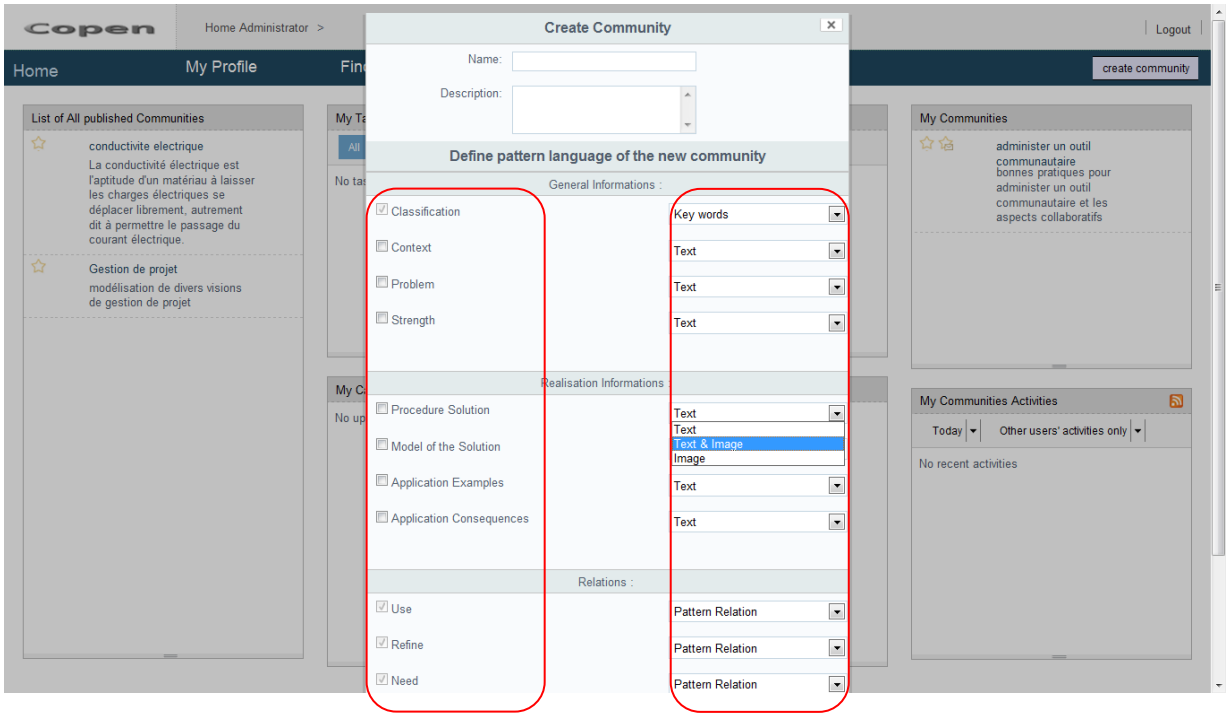


Figure 56 : Espace communautaire avec spécification de son propre langage

On remarque que la définition du langage est possible grâce à la sélection des attributs en les cochant et grâce au choix de leur type lors de la création de la communauté. Le langage de référence basé sur P-Sigma est stocké dans un nœud de type contenu Alfresco. Un développeur peut aller très facilement modifier ce contenu pour ajouter ou enlever des champs. Dynamiquement, COPEN instanciera le nouveau langage.

La figure 57 présente un patron basé sur ce langage commun.

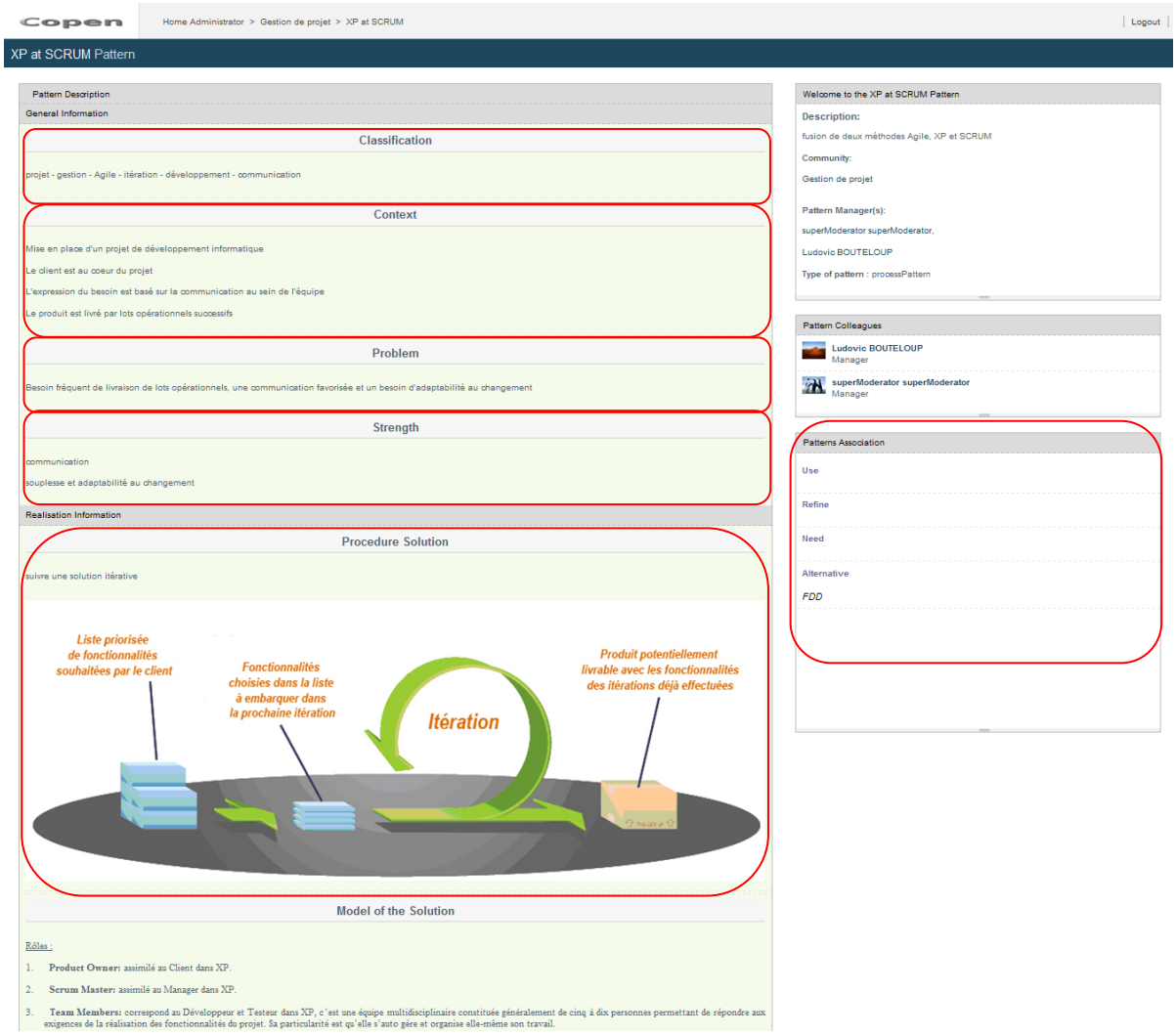


Figure 57 : Espace patron édité avec le langage défini par la communauté

Une description centrale ainsi qu'un espace en bas à droite pour les relations inter-patrons correspondent aux rubriques du langage suivant leur type.

La description centrale affiche toutes les rubriques (mises à part les relations) suivant le type associé. Les 3 premières rubriques encadrées sont de type texte et la quatrième, est de type texte+image, ce qui permet de télécharger une image ainsi que de la décrire.

L'encadré en bas à droite apporte la partie relation du langage. Il est possible de spécifier les patrons apportant une relation ; ils sont affichés sous forme de lien pour une navigation par relation.

La figure 57 représente un patron reprenant les principes de la méthode Agile XP@SCRUM à l'aide des rubriques du langage défini pour la communauté à laquelle il appartient.

3) Bilan

Rétrospective des *sprints* effectués :
+ : Travail en équipe privilégié et efficace en S#6.
+ : Motivation, persévérance et réflexion.
+ : Avancée dans le langage communautaire (développé de façon personnalisable pour l'administrateur).

- : Rien ne sert d'avancer le développement trop vite si les tests ou les artefacts comme la documentation ne suivent point.
- : Aide demandée trop tardivement.
- : Documentation presque oubliée en S#6.

Les améliorations à prendre en compte dans les *sprints* suivants sont :

1. Développer au plus simple pour répondre au besoin (« keep it simple, stupid »).
2. Continuer à mettre à jour la documentation et les points bloquants au fil de l'eau afin de gagner du temps.
3. Demander rapidement de l'aide lorsque l'on bloque sur un problème.

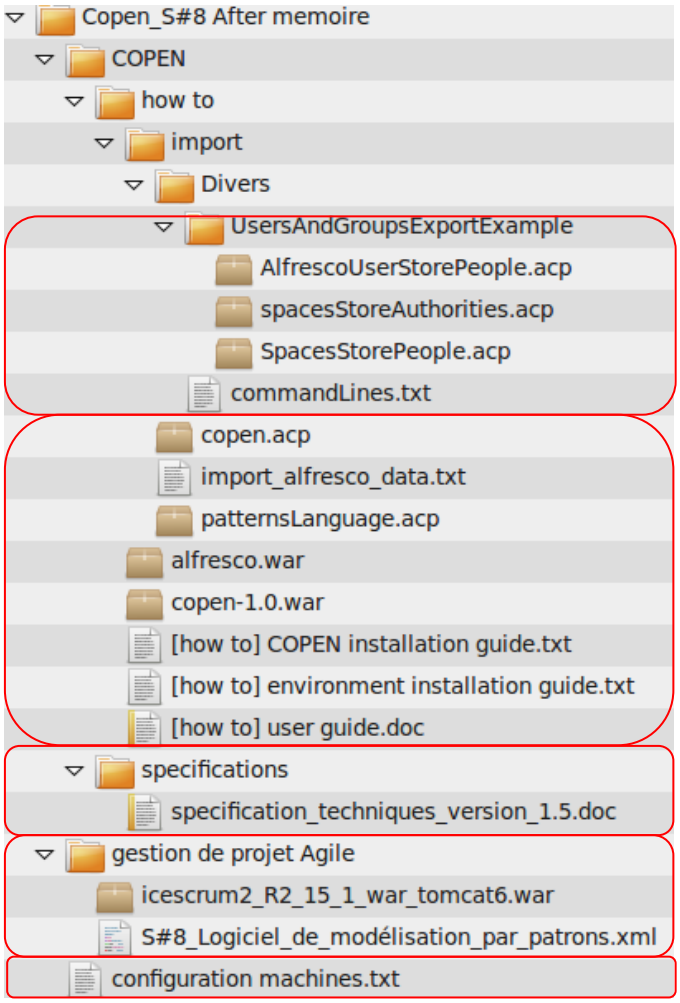
Nous notons les pratiques Agile suivantes :

1. Aller à l'essentiel, proposer une conception simple.
2. Travail en binôme.
3. Passage de tests approfondi.

Cette méthode a permis, pour chacun de ces *sprints*, de livrer des lots opérationnels, contenant des fonctionnalités utilisables par le client.

2.3.5. Livraisons COPEN

La structure de livraison de chaque lot est organisée clairement. La figure 58 présente l'arborescence de livraisons.



Aide : Lignes de commandes utiles pour l'import export des données Alfresco

Fichiers utiles à l'installation de l'environnement de développement ainsi qu'au déploiement de COPEN

Spécifications techniques

Sauvegarde de la gestion du projet

Inventaire des machines disponibles

Figure 58 : Structure des livraisons des sprints

Une arborescence telle que celle-là permet la livraison non seulement de l'outil COPEN mais aussi de tout le nécessaire pour le réinstaller, le comprendre et le faire évoluer. C'est dans cette optique de reprise et d'évolution que COPEN est développé.

2.4. Bilan et perspectives

L'ensemble de ces neuf *sprints* a permis, au sein d'un espace de travail structuré et automatisé grâce à Eclipse et son plugin Maven, de développer l'essentiel des fonctionnalités COPEN.

La première étape de formation et de préparation de l'environnement est la plus « laborieuse », il n'est pas évident de s'auto-former sur Alfresco et sur la façon de le développer et encore moins de trouver une aide efficace.

La première étape du développement est de créer la base communautaire avec sa gestion des membres. Elle permet d'affecter des droits aux membres COPEN : de l'administration pour la gestion des membres ou de la super-modération pour le contrôle des contenus qu'ils publient. Cette base correspond au premier développement effectué, parfois difficile mais au final, une première satisfaction.

Ensuite la gestion de communautés permet à des membres COPEN de créer des communautés en leur associant un langage, de les faire vivre à l'aide de forums, de les actualiser en les éditant et de les publier. Chaque membre peut posséder un rôle dans la collaboration que permet une communauté. Le premier est le rôle de contributeur afin de donner son avis et de participer à son évolution. Le second est modérateur, afin d'éditer la description et contrôler le cycle de vie d'une communauté. Tout deux permettent aux membres COPEN de créer des patrons pour chacune d'entre elles. La personnalisation COPEN prend réellement forme à partir de ce moment là. L'amélioration continue, l'adaptation et la simplification des tâches ont permis d'avancer le développent sans prendre de retard. Quelques points comme la rédaction des tests ou de la documentation restent tout de même à améliorer.

La dernière partie concerne les patrons. Ce sont des espaces collaboratifs permettant de modéliser un processus ou un produit suivant un langage propre à la communauté à laquelle ils appartiennent. De même que pour les communautés, les rôles de contributeur et modérateur peuvent être affectés aux membres COPEN afin qu'ils participent à l'évolution de celui-ci. Forums, édition des rubriques du langage et publication caractérisent cette évolution. Cette partie « gestion de patrons » est la plus importante en termes de charge de travail fourni. Le cycle en *sprints* devient rodé et la compétence des développeurs plus grande, l'évolution du projet est cohérente, la qualité et la capacité en charge de travail augmente de *sprints* en *sprints*.

Ces trois aspects permettent l'utilisation de l'outil suivant le besoin collaboratif qui était exprimé. Certaines fonctionnalités développées de façon minimalistes afin de répondre au besoin de la manière la plus simple peuvent être revues. Quelques améliorations sont possibles afin d'optimiser son utilisation :

- La possibilité d'inviter directement un membre à un patron sans qu'il soit membre de la communauté est intéressante mais la navigation peut être améliorée pour devenir plus intuitive.

- L'import/export des membres sous format xml afin de pouvoir les sauvegarder et les installer sur une autre plateforme est à prévoir. Mais aussi l'import/export des communautés et patrons. Il serait intéressant d'importer ou d'exporter des patrons ou des cas d'utilisation via COPEN.

- L'inscription des membres se fait actuellement par envoi de mail à l'administrateur qui les inscrit un par un. L'amélioration serait d'offrir à chaque visiteur la possibilité de s'inscrire directement sur COPEN. Alfresco restreint ces droits actuellement, une version future les ouvrira certainement.

- Lorsqu'un rôle de « superModerator » est attribué à un membre COPEN, celui-ci devient modérateur de toutes les communautés et de tous les patrons prochainement créés, mais l'ajout du droit de modérateur aux communautés et patrons déjà existants n'est pas pris en compte.

- L'ajout d'un workflow lors de l'inscription ou invitation d'un membre, de la suppression de communautés ou de patrons serait une procédure intéressante pour la cohérence du logiciel. Il permettrait d'accepter, refuser ou discuter chacune des actions.

- Le moteur de recherche que propose Alfresco est caché car il a besoin d'être personnalisé aux communautés et aux patrons. Pour aider à la navigation lorsque beaucoup de communautés ou patrons sont créés, la personnalisation de ce moteur peut être intéressante.

COPEN est donc un environnement de gestion de patrons collaboratifs répondant globalement au besoin et prêt à l'utilisation même si des améliorations pourraient le rendre plus complet.

Conclusion

Conclusion

La gestion de projets est très complexe. La mise en place d'une gestion de projets Agile pour le développement d'un environnement de patrons collaboratifs m'a permis une bonne expérience dans ce domaine. Tout d'abord par la découverte de trois solutions : XP@SCRUM pour la méthode de gestion de projet, l'atelier AGAP pour comprendre la modélisation par patrons et Alfresco pour un socle collaboratif. L'utilisation de ces trois solutions pour en former une quatrième, COPEN, a permis d'aborder toutes les phases importantes d'un projet. Il y a là pour moi une montée en compétence, de la spécification fonctionnelle et technique à une mise en place d'un environnement de travail, du développement, des tests, de la livraison continue et bien entendu tout cela guidé par une gestion de projets XP@SCRUM. Cette expérience en gestion de projets a permis d'avoir un recul plus critique sur l'importance du contexte du projet et de son déroulement mais aussi sur celle des pratiques Agiles.

L'initialisation du projet est très importante. Le contexte en est l'illustre exemple. Le choix des membres de l'équipe, le nombre de développeurs, les compétences de chacun, leur sérieux et la motivation qu'ils dégagent ne sont pas à négliger. Ces critères permettent de connaitre la durée de formation nécessaire à chacun et la manière de les appréhender tout au long du projet. Le développeur choisi à temps plein sur ce projet ne possédait que très peu de bases en développement d'applications, l'importance de ce détail sous-estimé a fait perdre du temps en début de projet.

Le contexte c'est aussi le client et son domaine de compétences. Le client fait partie de l'équipe de recherche SIGMA spécialisée en modélisation adaptable, en d'autres termes le client est très familier avec la représentation de l'information sous forme de schémas normés tels que les propose UML. L'approche Agile, tout particulièrement la méthode XP@SCRUM propose paradoxalement aux habitudes de l'équipe une définition textuelle du projet. Au lieu de proposer cette description textuelle censée être plus proche du client mais finalement ici rébarbative, il aurait été plus judicieux de proposer une méthode définissant des modèles. Certains points fonctionnels décrits textuellement sont venus à plusieurs reprises en sujet de réunion, comme les droits des membres ou les cycles de vie des patrons. A contrario, il a été remarqué que la communication autour des rubriques du langage à développer était favorisée car le client pouvait s'appuyer sur le modèle de l'atelier AGAP pour s'exprimer. Dans le contexte de cette équipe de recherche, une modélisation rapide en début de projet aurait permis une base de travail et de communication plus efficace. La méthode FDD à la frontière de l'Agile correspond justement au besoin, elle base sa pratique Agile sur une modélisation UML simple spécifiée au lancement du projet. Elle aurait pu être choisie pour améliorer la communication et permettre de gagner du temps et de l'efficacité dans la compréhension des besoins.

Le développement itératif permettant la livraison régulière de lots fonctionnels est quant à lui très intéressant. La démonstration client en fin de *sprint* permet de manipuler l'outil, d'échanger avec le client, d'avoir son retour sur les fonctionnalités et s'il le faut, anticiper un changement dans le *sprint* suivant. Ce cycle itératif permet cette souplesse afin de livrer le plus tôt possible un résultat opérationnel pour le client. La description des *user stories* spécifiées dans les premiers *sprints* du projet demandait de mettre en place des mécanismes complexes et longs à développer. Il a été possible en manipulant l'outil lors d'une démonstration de revoir et adapter les *user stories* pour qu'elles répondent au besoin de la manière la plus simple possible. Cela a permis de ne pas bloquer sur du développement et d'arriver le plus vite possible à l'essentiel du besoin client.

XP@SCRUM, très orientée sur la communication entre membres de l'équipe, cache tout de même des pratiques essentielles à la réussite du projet. La première concerne les *user stories*, même si l'on ne modélise pas de façon normée l'outil dans les spécifications, la gestion des *user stories* doit être bien structurée, ordonnée afin de toujours avoir une rédaction claire et non ambigüe.

La deuxième pratique vise la documentation, les spécifications fonctionnelles et techniques. Elles doivent être rédigées au fil des développements pour ne pas perdre de temps en fin de *sprint*.

Une troisième pratique importante est la relecture de code. Le fait de travailler sur des descriptions textuelles entraîne de fournir une attention particulière au code. Une relecture rapide permet souvent de clarifier le développement, le travail en binôme permet lui aussi de travailler de manière propre simple et commentée.

La dernière pratique est l'importance du rôle de chacun. Comme l'a prouvé celui du « scrum master », c'est grâce à la responsabilité à trouver des solutions aux problèmes que la relation avec Grenoble Université est née et a permis de solutionner quelques points bloquants.

C'est pour cela que chacun des *sprints* proposés par cette gestion de projets Agile permet de faire un bilan, d'en tirer des axes d'amélioration et de mettre en œuvre une amélioration continue de la stratégie appliquée. Lors du développement de COPEN, la mise en place en cours de projet d'une rétrospective pointant les points négatifs mais aussi les points positifs a permis un autre recul sur le *sprint* réalisé et une communication encore plus présente. Ce dispositif a aidé à l'amélioration continue et à l'utilisation la plus régulière possible des bonnes pratiques comme le très intéressant travail en binôme. Malgré des tâches prévues à cet effet, certaines pratiques n'ont pas réussi à être améliorées comme souhaitées. Le manque de temps, l'envie de livrer un résultat a fait apparaître la mauvaise tendance à négliger la rédaction des tests et de la documentation au fil de l'eau, même si l'effort fourni lors des derniers *sprints* a comblé ce point négatif. C'est donc avec l'amélioration continue et la prise d'expérience que le déroulement d'un projet gagne en efficacité.

C'est sur un ensemble de bonnes pratiques, comme la communication, l'amélioration continue, l'adaptation au changement et au contexte mais aussi le respect de normes ou le travail en binôme que neuf *sprints* de trois semaines ont été livrés.

Cet outil est prêt à affronter l'avenir. Basé sur Alfresco il respecte des normes qui le garderont fiable, cohérent et compréhensible de tout développeur. Il est conçu pour évoluer et cela même en cas de fort « turnover ». De plus, tous les éléments nécessaires ont été livrés. Les spécifications fonctionnelles et techniques contenant une modélisation UML de COPEN sont fournies permettant l'évolution de l'outil à l'aide de méthodes de gestion de projets plus classiques. Un guide d'installation de l'environnement de développement pour l'aspect évolutif est disponible. Un guide de déploiement de l'application sur un serveur et un guide utilisateur sont aussi présents afin d'utiliser l'outil dans de bonnes conditions.

La version COPEN livrée constitue un bon noyau. Il a été conçu de façon à correspondre au besoin de l'équipe dans un temps imparti. Il ne va pas se limiter à être le résultat d'un mémoire, ses fonctionnalités réduites sont déjà intéressantes pour l'équipe SIGMA. C'est lors d'un nouveau projet en collaboration avec d'autres équipes que COPEN va être utilisé prochainement. Chaque équipe, voire chaque collaborateur possédera ses propres savoirs et compétences sur le projet. La modélisation du travail de chaque membre sur COPEN permettra de mettre en commun l'avancement de chacun et proposera une collaboration autour de celui-ci. COPEN sera utilisé plus largement, lors de divers projets de différentes tailles : un exemple est l'intérêt d'une équipe IHM (Interface Homme Machine) pour COPEN afin de créer leurs modèles et cas d'utilisation, voire même dans une future version : les importer ou exporter. L'utilisation de COPEN est envisagée, mais son évolution l'est tout autant ! Un développeur (à temps partiel) se chargera d'améliorer cet outil au fil des utilisations. La pratique au quotidien permettra de cibler les axes d'amélioration les plus pertinents.

Annexes

Annexes

A. Architecture des fichiers COPEN-1.0

1) Structure & Arborescence

Afin de pouvoir publier une page supplémentaire à Copen (ici il s'agit de réaliser la page d'accueil du site) il est important de suivre l'architecture préconisée par Alfreco :

En premier lieu il est nécessaire de se positionner au niveau du répertoire racine :

***WEB_RACINE* =.../tomcat/webapps/copen/WEB-INF/classes/alfresco**

Remarque : dans tous les cas le nommage des fichiers est très important.

Voici donc comment est structurée l'arborescence de COPEN-1.0 sous ***WEB_RACINE*** :

- /sites-data/pages : Fichier de description xml : le nom du fichier correspond au nom appelé dans le navigateur.

Ex : http://*AdresseDuServeur:8080*/copen-1.0/page/Nom_du_fichier.

Ce fichier fait référence à un template-instance.

- /templates : ficher « .ftl » correspondant à la description html d'une page sous format freemarker. Il est entièrement composé de tag « <@région … /> » faisant référence à des fichiers « .xml » de régions permettant d'affecter un composant à une zone de la page Web.

- /sites-data/components : Fichier « .xml » de description de la région permettant le lien avec le(s) fichier(s) de composants rattachés. Le tag <url> permet notamment de localiser le fichier composant rattaché à partir de l'arborescence WEB_RACINE/site-webscripts.

- /sites-data/templates-instance : permet de renseigner la localisation du template à partir de l'arborescence **WEB_RACINE***/templates* et ce via le tag « template-type »

- /site-webscripts : répertoire recensant tous les composants webscripts.

  Chacun d'eux est défini par :

  - un fichier de description (.xml)
  - un fichier de sortie (.ftl, JSON suivant le format de sortie désiré). Un fichier head peut aussi être associé permettant de faire des imports d'autres scripts ou de feuilles de styles par exemple)

Optionnellement :

- un fichier JavaScript ou .jsp pour le côté contrôleur MVC

Sous Alfresco, la structure pour développer des webscripts est la même, Voici l'arborescence WEB_RACINE d' Alfresco :

WEB_RACINE : alfresco\WEB-INF\classes\alfresco\templates\webscripts\org\alfresco\

2) Exemple de développement : Accéder au logiciel

Pour notre cas, la page de login initial d'Alfresco Share a été modifiée afin d'avoir le rendu de celle du site COPEN Home.

Chemin: WEB_RACINE/templates/org/alfresco/global/slingshot-login.ftl)

Ceci permet :

- d'accéder à COPEN via l'url http://<nomduserveur><port>/copen

- permettre le retour à cette page apres logout de l'espace perso.

Dans ce fichier, 6 régions ont été définies :

- **topCopen** d'étendue « global » : permet de donner un nom au site et d'assiocier notamment un css.
- **headerCopen** d'étendue « global » : bandeau LIG et logo COPEN
- **navCopen** d'étendue « global » : barre des menu (un js est associé pour offrir un renderer)
- **mainCopen** d'étendue « global (peut être mis en page) : texte de la page d'accueil situé dans la partie gauche (Remarque : les contenu de ces textes sont dans le backend alfresco sous CompanyHome/Data_dictionary/CopenAccueil avec droit de lecture pour les visiteurs).
- **SidebarCopen** détendue « page » : composé de l'encart contenant l'espace login/authentification à COPEN mais aussi deux zones de textes (possibilités de renvoi sur contenu « news », « calendrier » par exemple).
- **Footer** d'étendue « global » contenant le bandeau lig bas.

Chaque région fait référence aux composants/webscripts situés dans WEB_RACINE/site-webscripts/Copen/...

B. Livrables de COPEN-1.0

1) Guide d'installation de l'environnement de développement COPEN-1.0

[how to] environment installation guide.txt

```
/*********************************************************************
****** Installation de l'environnement de développement COPEN *******
*********************************************************************/

1. Telecharger eclipse j2ee galiléo:
http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads
/release/galileo/SR2/eclipse-jee-galileo-SR2-linux-gtk.tar.gz

2. Télécharger puis installer les plugins eclipse necessaires au déploiement
de COPEN:
par eclipse: http://subclipse.tigris.org/update_1.4.x

3. Télécharger puis installer dans le repertoire plugin d'eclipse, les
plugins eclipse necessaires au déploiement de COPEN:
dezipper  GEF-ALL 3.5.2 => http://www.eclipse.org/gef/downloads/
puis si besoin unpack en utilisant la commande: unpack200 nom_du_.pack
nouveau_nom_sans_.pack

4. Télécharger puis installer les plugins eclipse necessaires au déploiement
de COPEN:
par eclipse: http://m2eclipse.sonatype.org/sites/m2e

5. Pointer dans windows preferences d'éclipse vers jdk au lieu de jvm
(telecharger un jdk)
dans eclipse.ini =>
-vm
/opt/jdk1.6.0_18/bin

6. Installer maven et obtenir les fichiers dans le root/.m2 =>sudo apt-get
install maven2

7. Utiliser le bon fichier de configuration pour les projets maven:
/etc/maven2/settings.xml et le copier dans root/.m2

8. Placer l'archetype share dans eclipse preferences=>
http://maven.alfresco.com/nexus/content/repositories/releases/archetype-
catalog.xml

9. Synchroniser avec le SVN de la machine: /var/svn/
Il y a 5 projets sont à récuperer en gestion de conf sous Eclipse:
File - new - other - SVN>Checkout Projects from SVN et spécifier ce chemin:
http://<MON_SERVEUR_SVN>/svn/copen/<NOM_DU_PROJET>
copen                    => partie principale de COPEN
copen-share-module       => dépendance intégré automatiquement à copen lors
du maven install de copen
copen-alfresco-zip       => zip qui sera à intégrer au war alfresco
copen-share-zip          => zip qui sera à integrer au war copen

10. Développer des fonctionnalités dans COPEN
copen:
     /src/main/resources/ correspond au copen-1.0/WEB-INF/
     /taget/ correspond au résultat de compilation maven
copen-share-module:
```

/src/java correspond à l'emplacement du développement des classes java
/src/main/resources/ correspond au données du .jar qui sera placé dans copen-1.0/WEB-INF/lib à la compilation
/taget/ correspond au résultat de compilation maven
copen-alfresco-zip
/main correspond au répertoire racine alfresco d'alfresco.war
copen-share-zip
/main correspond au répertoire racine copen-1.0 de copen-1.0.war

11. Compiler Copen
1. Faire un maven clean des 2 projet Maven: copen & copen-share-module
2. Faire une compilation Ant des 2 projets: copen-alfresco-zip & copen-share-zip afin d'obtenir les 2 archives .zip à intéger
3. Faire une compilation Ant du projet: copen-share-module afin de générer les .class des classes java ajoutées si il y a lieu.
4. Faire un maven install du projet copen-share-module afin de générer la structure à integrer à COPEN
5. Faire un maven install du projet copen afin de générer le copen-1.0.war de COPEN en intégrant automatiquement dans celui-ci copen-share-module grâce à la dépendance
6. Integrer copen-share-zip dans copen-1.0.war
7. Integrer copen-alfresco-zip dans alfresco.war

12. Suivre la doc d'installation "[how to] COPEN installation guide" afin de déployer COPEN sur le serveur d'application.
Rmq1: Si COPEN est délà installé et que vous souhaitez garder ses données, suivre simplement les étapes 7,8,9.

2) Guide de déploiement COPEN-1.0

[how to] COPEN installation guide.txt

```
/**********************************
****** Installation de COPEN *******
**********************************/

1. Se rendre sur le site web Alfresco:
http://wiki.alfresco.com/wiki/Community_Edition_file_list_32r2

2. Télécharger Alfresco-Community-3.2r2-Linux-x86-Install ainsi que sa doc
d'installation "simple install instructions for Linux"

3. Suivre les étapes 1 à 15 de la doc d'installation
Rmq1: Si l'installation se fait à distance, il faut utiliser une interface
X11
(configuration de son poste très bien expliqué
ici=>http://www.commentcamarche.net/faq/6884-serveur-x-sous-windows)
Rmq2: Si la base alfresco n'a pas été crée, il faut executer le script
/opt/Alfresco/extras/databases/mysql/db_setup.sql et supprimer le contenu de
alf_data.

4. Dans opt/Alfresco, mettre à jour les .sh en fonction du chemin du serveur
d'application tomcat souhaité

5. Dans le serveur d'application tomcat souhaité
<monChemin>/tomcat6/webapps/, copier les archives fournies: copen-1.0.war et
alfresco.war

6. Vérifier la présence ou placer la librairie de connection MySql dans le
serveur tomcat: tomcat/lib/mysql-connector-java-5.1.7-bin.jar

7. Configurer les propriétés de base de données et de mails d'alfresco.war
dans WEB-INF/classes/alfresco-global.properties
exemple BD:
#
# Sample database connection properties
#-------------
db.name=alfresco
db.username=alfresco
db.password=alfresco
db.host=sigmacomp
db.port=3306

#
# External locations
#-------------
ooo.exe=/usr/lib/openoffice/program/soffice
ooo.user=<%ShortInstallDir%>/alf_data/oouser
img.root=/usr/local
swf.exe=/opt/Alfresco/bin/pdf2swf

#
# Initial amysql-connector-java-5.1.7-bin.jardmin password
#-------------
alfresco_user_store.adminpassword=209c6174da490caeb422f3fa5a7ae634

#
```

```
# MySQL connection
#-------------
db.driver=org.gjt.mm.mysql.Driver
db.url=jdbc:mysql://${db.host}:${db.port}/${db.name}
hibernate.dialect=org.hibernate.dialect.MySQLInnoDBDialect

8. Ajouter s'il n'est pas déja présent, le port des services RMI
d'alfresco.war dans WEB-INF/classes/alfresco-global.properties
(pas toujours pris en compte dans alfresco-shared.properties)
#
# Properties shared between the Alfresco server
# and its remote clients (e.g.: the virtualization server).
#
# Ports used by Alfresco
#
# Note:  These ports are also used by the virtualization server
#        (hence, they're in a seperate file that can be copied easily).

# Remote RMI services
alfresco.rmi.services.port=50500
alfresco.rmi.services.host=sigmacomp

9. Modifier les "localhost" présents dans les fichiers properties
d'alfresco.war afin de spécifier la bonne addresse.
Rmq1: les fichiers properties se trouvent dans /WEB-INF/classes etc..
Rmq2: une verification ulterieure peut etre faite grâce à la commande: find
tomcat/webapps/ -name "*.properties" -exec grep -Hn "localhost" {} \;

10. Créer deux liens symboliques pour la conversion de fichiers:
     Rechercher le converter grace à la commande: sudo find / -name convert
-print
     Créer un lien symbolique de /usr/local/bin/convert vers ce converter:
sudo ln -s /usr/bin/convert /usr/local/bin/convert
     Rechercher le pdf2swf grace à la commande: sudo find / -name pdf2swf -
print
     Créer un lien symbolique de /opt/Alfresco/bin/pdf2swf vers ce
converter: sudo ln -s /usr/bin/pdf2swf /opt/Alfresco/bin/pdf2swf

11. Démarrer le serveur tomcat à l'aide du script /opt/Alfresco/tomcat.sh
start et vérifier que les 2 applications sont déployées sur le serveur
Rmq1: Il faut toujours démarrer le serveur avec ce script et non avec celui
de tomcat (ou le configurer)!
Rmq2: En cas d'erreur de démarrage, vérifier les étapes 4 et 7 qui sont
souvent la source du problème

12. Eteindre le serveur tomcat à l'aide du script /opt/Alfresco/tomcat.sh
stop

13. Importer (en suivant import/import_alfresco_data) les .acp dans alfresco
puis redemarrer

14. Créer un utilisateur sous alfresco visitor/visitor (avec les droits les
plus restreints)

15. Créer un groupe strictement nommé "CopenModerator" avec un user (sauf
visitor) en dessous qui sera super moderateur.

16. Personnaliser à COPEN le contenu d'envoi d'invitation par mail dans
Company Home > Data Dictionary > Email Templates > invite  :
<#assign inviterPersonRef=args["inviterPersonRef"]/>
<#assign inviterPerson=companyhome.nodeByReference[inviterPersonRef]/>
```

```
<#assign inviteePersonRef=args["inviteePersonRef"]/>
<#assign inviteePerson=companyhome.nodeByReference[inviteePersonRef]/>

Hello ${inviteePerson.properties["cm:firstName"]},

You have been invited by ${inviterPerson.properties["cm:firstName"]}
${inviterPerson.properties["cm:lastName"]} to join the '${args["siteName"]}'
space.

Your role in the space will be ${args["inviteeSiteRole"]}.

To accept this invitation click the link below.

${args["acceptLink"]}

<#if args["inviteeGenPassword"]?exists>
and enter the following information:

Username: ${args["inviteeUserName"]}
Password: ${args["inviteeGenPassword"]}

We strongly advise you to change your password when you log in for the first
time.

</#if>
If you do not want to join the space then click here:

${args["rejectLink"]}

Regards,

COPEN Team
COmmunities of Patterns ENvironment
```

C. Gestion de projets anarchique

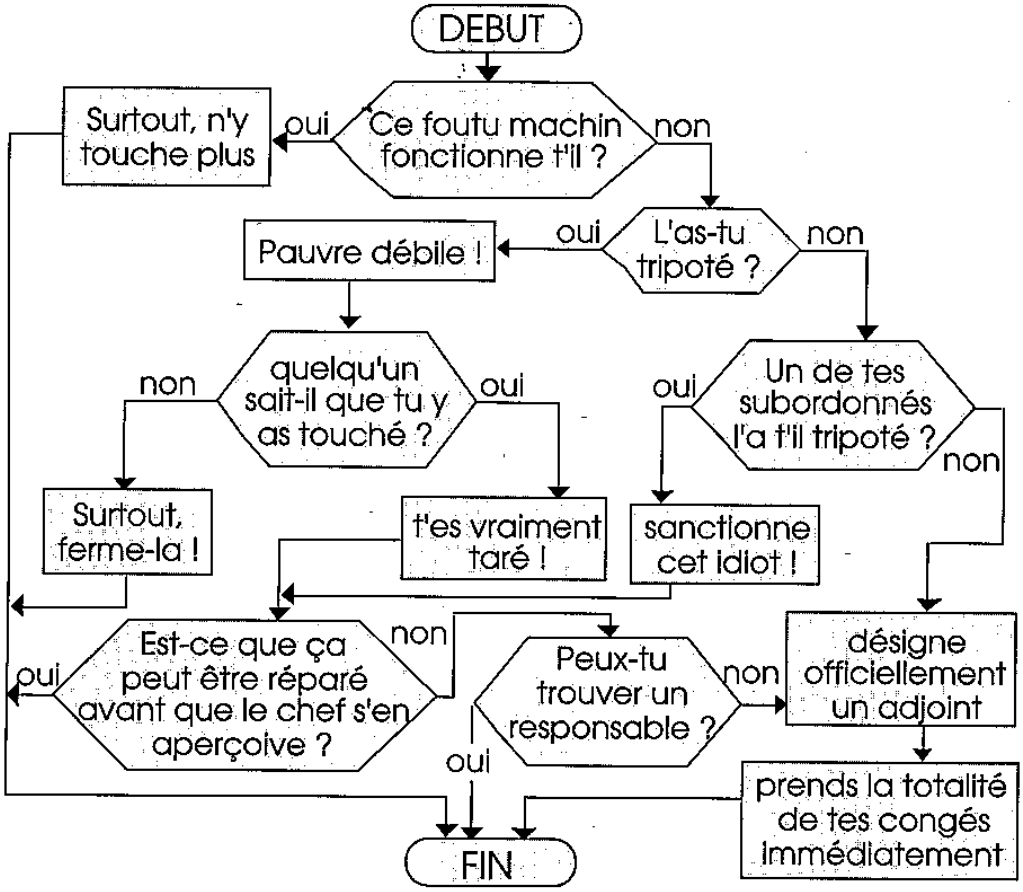


Figure 59 : Comment dégager sa responsabilité de chef de projet [24]

Un projet doit être structuré, chaque membre de l'équipe doit avoir son rôle et sa part de responsabilité dans un projet. L'investissement personnel et la communication quotidienne contribue à la réussite collective d'un projet et permet d'éviter des problèmes cachés qui de toute façon ressortiront tôt ou tard.

Il n'est bien sûr souhaité à personne de mener un projet comme l'illustre la figure 59 !

Glossaire

Backlog du produit :
liste priorisée de fonctionnalités souhaitées par le client.

Catalogue de patrons :
collection de patrons avec des règles permettant de les combiner. Chaque catalogue de patrons utilise son propre formalisme de représentation constitué de différentes rubriques.

Daily Scrum :
(mêlée quotidienne) réunion quotidienne de 15 minutes effectuée le matin.

***Feature* :**
fonctionnalité.

Imitation du patron :
duplication de la solution, puis adaptation de cette solution au contexte.

Impediment List :
liste des fonctionnalités n'ayant pu être traitées par manque d'information ou d'outils.

Mêlée quotidienne :
réunion quotidienne de 15 minutes effectuée le matin.

Patron :
base de savoir-faire pour résoudre un problème récurrent dans un contexte applicatif ou technologique particulier. Les patrons existent à tous les niveaux du cycle de vie d'une application : ainsi, il existe des patrons d'analyse, de conception et d'implantation.

Patron processus :
patron capitalisant des spécifications ou des implantations d'une démarche à suivre pour atteindre le résultat.

patrons produit :
patron capitalisant des spécifications ou des implantations d'un but à atteindre.

Product Backlog (Backlog du produit) **:**
liste priorisée de fonctionnalités souhaitées par le client.

Product Backlog Burndown :
Illustration de l'avancement des fonctionnalités traitées du Product Backlog sous forme de graphique.

Recettage :
phases de développement d'un projet au cours de laquelle les différents acteurs se rencontrent afin de vérifier que le produit est conforme aux attentes formulées.

***Sprint* :**
itération durant généralement entre deux et quatre semaines.

***Sprint* Backlog :**
fonctionnalités du Product Backlog affectées à un *sprint*.

***Sprint* Backlog Burndown :**
Illustration de l'avancement des fonctionnalités traitées du *sprint* backlog sous forme de graphique.

***User Story* :**
fiche scénario descriptive du comportement d'une fonctionnalité.

Workflow :
modélisation et gestion automatisée de l'ensemble des tâches à accomplir lors de la réalisation d'un processus métier.

XP@SCRUM :
fusion de deux méthodes Agile, les principes XP sont intégrés à SCRUM.

Bibliographie

[1] L. Bouteloup, Etude et synthèse sur les méthodes Agile, CNAM Synthèse TEST, Grenoble, 2009.

[2] A. Conte, I Hassine, J-P. Giraudin, D. Rieu, AGAP : un Atelier de Gestion et d'Application de Patrons, Inforsid'01, Genève, 2001.

[3] V. Pujalte, P Ramadour, Corine Cauvet, Recherche de composants réutilisables : une approche centrée sur l'assistance à l'utilisateur, Inforsid'04, Biarritz 2004.

[4] Audibert, L., Cours UML. [En ligne]. http://laurent-audibert.developpez.com/Cours-UML/html/Cours-UML005.html, Septembre 2010.

[5] Comment ça marche. Cycle de vie. [En ligne]. http://www.commentcamarche.net/contents/genie-logiciel/cycle-de-vie.php3, Septembre 2010.

[6] Lompré, N., Conception du cycle de vie. [En ligne]. http://web.univ-pau.fr/~lompre/conception/conception.htm, Septembre 2010.

[7] Beck, K., 1999. Extreme Programming Explained : Embrace Change. Addison-Wesley Professional, Longman, Publishing Co., Inc Boston, Massachusetts. 190p.

[8] Richand, N., Agilité. [En ligne]. http://www.richand.info/blog/?p=11, Septembre 2010.

[9] Agile manifesto. [En ligne]. http://agilemanifesto.org/, Septembre 2010.

[10] Qualitystreet. [En ligne]. http://www.qualitystreet.fr/2009/02/27/agile-manifesto-12-principes-a-ne-pas-perdre-de-vue/, Septembre 2010.

[11] Agile alliance. [En ligne]. http://www.agilealliance.org/, Septembre 2010.

[12] Agile tour. [En ligne]. http://www.agiletour.com/, Septembre 2010.

[13] Rapid Application Development, Wikipedia. [En ligne]. http://en.wikipedia.org/wiki/Rapid_application_development, Septembre 2010.

[14] Wikipedia. SCRUM. [En ligne]. http://fr.wikipedia.org/wiki/Scrum, Septembre 2010.

[15] Wikipedia. Feature Driven Development, Wikipedia. [En ligne]. http://en.wikipedia.org/wiki/Feature_Driven_Development, Septembre 2010.

[16] Wikipedia. Unified Process. [En ligne]. http://fr.wikipedia.org/wiki/Unified_Process, Septembre 2010.

[17] Wikipedia. Extreme Programming, Wikipedia [En ligne]. http://en.wikipedia.org/wiki/Extreme_Programming, Septembre 2010.

[18] Chenu, E., eXtreme Programming. [En ligne]. http://manu40k.free.fr/eXtremeProgramming2emeEdition.pdf, Septembre 2010.

[19] Schweber, K., Scrum with XP. [En ligne]. http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/scrum/ScrumXP.pdf, Septembre 2010.

[20] Wikipedia. Extreme Progrramming, Wikipedia. [En ligne]. http://fr.wikipedia.org/wiki/Extreme_programming, Septembre 2010.

[21] Kruchten, P., 2000. Introduction au Rational Unified Process. Addison Wesley, Traduction Edition Eyrolles, Paris. 257p

[22] Feature Driven Development. [En ligne]. http://www.step-10.com/SoftwareProcess/FeatureDrivenDevelopment/, Septembre 2010.

[23] Cnrs. Gestion de projet. [En ligne]. http://www.dsi.cnrs.fr/methodes/gestion-projet/methodologie/BI-methodes-agiles.pdf, Septembre 2010.

[24] Claviez, J., 2002, 4eme édition. Diriger un projet informatique. JCI inc. Québec-livres, Canada. 317p

[25] Scrum for team system. [En ligne]. http://www.scrumforteamsystem.com/ProcessGuidanceOld/v2/ProcessGuidance.aspx, Septembre 2010.

[26] Justaddwater. Touch Screen. [En ligne]. http://justaddwater.dk/2008/03/14/where-to-find-a-giant-affordable-touch-screen/, Septembre 2010.

[27] IceScrum. [En ligne]. http://www.icescrum.org/, Septembre 2010.

[28] Agiliste.fr. Outils de gestion de projet. [En ligne]. http://www.agiliste.fr/Home/outils-gestion-projet, Septembre 2010.

[29] Danube. Scrumworks. [En ligne]. http://www.danube.com/scrumworks, Septembre 2010.

[30] IBM. Forum. [En ligne]. http://www.ibm.com/developerworks/forums/thread.jspa?threadID=165333, Septembre 2010.

[31] FDDTools. [En ligne]. http://fddtools.sourceforge.net/, Septembre 2010.

[32] FDDPMA. [En ligne]. http://www.fddpma.net/fddpma/projectGroupWorkplaceView.jsf, Septembre 2010.

[33] XProcess. [En ligne]. http://www.openxprocess.com/xprocess/, Septembre 2010.

[34] A. Conte, M Fredj, J-P. Giraudin, D. Rieu, P-Sigma : un formalisme pour une représentation unifiée de patrons, Inforsid'01, Genève, 2001.

[35] A. Conte, I Hassine, J-P. Giraudin, D. Rieu, Un environnement et un formalisme pour la définition, la gestion et l'application de patrons, Revue des sciences et technologies de l'information, article vol 6/2, pp.9-39, 2001.

[36] A. Conte, I Hassine, J-P. Giraudin, D. Rieu, AGAP : un Atelier de Gestion et d'Application de Patrons, Inforsid'01, Genève, 2001.

[37] Patron de conception, Wikipedia. [En ligne].
http://fr.wikipedia.org/wiki/Patron_de_conception, Septembre 2010.

[38] Modèle-vue-contrôleur, Wikipedia. [En ligne].
http://fr.wikipedia.org/wiki/Modèle-Vue-Contrôleur, Septembre 2010.

[39] ash.MVC 1.0, Softpedia. [En ligne].
http://webscripts.softpedia.com/script/Development-Scripts-js/ash-MVC-32366.html, Septembre 2010.

[40] Système de gestion de contenu, Wikipedia. [En ligne].
http://fr.wikipedia.org/wiki/Système_de_gestion_de_contenu, Septembre 2010.

[41] Liste de systèmes de gestion de contenu, Wikipedia. [En ligne].
http://fr.wikipedia.org/wiki/Liste_de_systèmes_de_gestion_de_contenu, Septembre 2010.

[42] I. Gautreau, D. Lagaert, Communauté Joomla, Joomla pour les nuls version 1.5, Creative Commons, pp 7-34, 2010.

[43] Joomla popular extensions. [En ligne].http://extensions.joomla.org/extensions/popular, Septembre 2010.

[44] Joomla [En ligne]. http://extensions.joomla.org, Septembre 2010.

[45] Drupal, Wikipedia. [En ligne]. http://fr.wikipedia.org/wiki/Drupal, Septembre 2010.

[46] Drupal as a MVC Framework, archivemati.[En ligne].
http://archivemati.ca/2006/01/21/drupal-as-a-mvc-framework/, Septembre 2010.

[47] Alfresco.[En ligne]. https : //www.alfresco.com/fr, Septembre 2010.

[48] Spring, Wikipedia. [En ligne]. http://fr.wikipedia.org/wiki/Spring_framework, Septembre 2010.

[49] Hibernate, Wikipedia. [En ligne]. http://fr.wikipedia.org/wiki/Hibernate, Septembre 2010.

[50] MySQL, Wikipedia. [En ligne]. http://fr.wikipedia.org/wiki/MySQL, Septembre 2010.

[51] Lucene, Wikipedia. [En ligne]. http://fr.wikipedia.org/wiki/Lucene, Septembre 2010.

[52] MyFaces, Developpez.com. [En ligne].
http://schmitt.developpez.com/tutoriel/java/jsf/introduction/, Septembre 2010.

[53] Programmation orientée aspect, Wikipedia. [En ligne].
http://fr.wikipedia.org/wiki/Programmation_orientée_aspect, Septembre 2010.

[54] Ixxus release a Todo-list dashlet for the Alfresco Share application, Ixxus.[En ligne].
http://www.ixxus.com/blog/2009/04/dashlet-todo-list, Septembre 2010.

[55] Alfresco Repository Architecture, Wiki Alfresco.[En ligne].
http://wiki.alfresco.com/wiki/Alfresco_Repository_Architecture, Septembre 2010.

MEMOIRE D'INGENIEUR C.N.A.M. en INFORMATIQUE

Mise en œuvre d'un processus Agile : développement d'un outil de gestion collaborative de modèles de conception

Ludovic BOUTELOUP **Grenoble, le 15 Décembre 2010**

Résumé

Depuis les années 1960, le besoin des entreprises en termes de projet informatique change, le taux de réussite des projets est très mauvais, les méthodes de gestion ne sont plus adaptées et doivent évoluer à leur tour. Ce changement a abouti, depuis les années 1990, à une philosophie appelée Agile, elle a pour but de mettre le client en avant et de lui livrer des parties opérationnelles du projet régulièrement pour qu'il puisse juger de la validité de celles-ci et donc supprimer les risques d'échec en fin de projet. Le premier objectif de ce mémoire est de mettre en place une méthode de gestion de projets Agile dans le cadre du développement d'un logiciel de recherche. En effet, l'équipe SIGMA du LIG possède un outil de gestion de modèles de conception n'arrivant pas à répondre au besoin collaboratif souhaité par l'équipe. Le second objectif est donc le développement d'un outil basé sur un espace communautaire permettant aux membres de modéliser produits ou processus sous forme de patrons collaboratifs. Un outil axé sur la collaboration est le cas idéal pour mettre en place un projet lui aussi très orienté communication et collaboration. Une équipe de quatre membres, une collaboration extérieure, une gestion de projets Agile et neuf itérations permettent de voir naître COPEN : COmmunities of Patterns ENvironment.

Mots-clés : gestion de projets, méthode de développement Agile, collaboration, patron, COPEN.

Abstract

Since 1960, the requirements in terms of information technology have changed for the companies. The projects success rates are low and the management processes have to be adapted. These changes have led since 1990 to a new philosophy called Agile. Its main goal is to make Business people and developers work together daily throughout the project to deliver operational releases to customer regularly. That enables all the people involved to check that it works fine. This method makes the failure probability drop significantly. The first objective of this report is to set up a method of Agile project management through the development of a software of research. Indeed, the SIGMA team of the LIG has a management tool of design models failing to answer the collaborative need wished by the team. The second objective thus is to develop a tool based on a community space which allows the members to model products or processes in the form of collaborative patterns. A tool focused on the collaboration is the ideal case to set up a project also oriented communication and collaboration. A team of four members, an external collaboration, an Agile project management and nine iterations allow seeing the birth of COPEN, Communities of Patterns Environment.

Keywords: project management, Agile development method, collaboration, pattern, COPEN.