Standardisation des processus de développement et de maintenance du logiciel

Éric Cisternas

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

VERSAILLES

MEMOIRE

Présenté en vue d'obtenir

Le DIPLOME d'INGENIEUR CNAM

SPECIALITE : INFORMATIQUE

OPTION : Audit des Systèmes d'Information

Par

CISTERNAS Eric

Standardisation des processus de développement et de maintenance du logiciel

Soutenu le 11 Janvier 2013

JURY

PRESIDENT :

MEMBRES :

Résumé

Afin d'accroitre la qualité logicielle il est nécessaire de respecter des normes, bonnes pratiques et référentiels du marché, tels qu'ISO 12207 pour les processus de développement, ISO 14764 pour la maintenance logiciel et le processus issu d'ISO 12207 pour la vérification et la validation ainsi que CMMI pour les bonnes pratiques.

Appliquer ces standards et bonnes pratiques en les adaptant à la taille de l'entreprise et à la fabrication d'un Progiciel de Gestion Intégré a constitué un défi de taille dans la mesure où ces changements affectent l'ensemble des services de développement logiciel et une grande partie de l'entreprise.

L'ensemble de ces projets a constitué un véritable défi car l'adoption de ces standards adaptés à la taille de l'entreprise dans le but de construire un progiciel de gestion intégré, induit de multiples changements dans la plupart des départements de l'entreprise.

L'amélioration des processus de fabrication et de maintenance a contribué à l'accroissement de la qualité logiciel du produit fabriqué.

Summary

Improving software quality is done by following standards, best practices references of the market.

Those standards are ISO 12207 for software lifecycle processes, ISO 14764 for software maintenance, ISO 12207 derivative process for control and validation and CMMI for process improvement.

This project was a challenge because applying these standards and best practices, and adapting them to the size of the company in order to build Enterprise Resource Planning software implies multiple changes in most of the departments of the company.

Making these changes helped improving software quality for the product.

Remerciements

Je tiens à remercier M. Jean-Luc Brison, Président Directeur Général de la société Gestimum de m'avoir donné la chance de réaliser l'ensemble de ces projets.

Je remercie M. Emmanuel Triballier qui a contribué de façon active à ces changements, en donnant des avis pertinents et de m'avoir soutenu dans l'ensemble des projets réalisés.

Je remercie Mr Keryvel qui m'a aidé à la rédaction de ce mémoire et qui m'a apporté de précieux conseils.

Je remercie Mr Gaquère pour sa relecture attentive et son soutien lors du processus de création de ce mémoire.

Je remercie ma famille, mes enfants et mon épouse qui ont su se montrer patients pendant tout le temps consacré à étudier au CNAM et à préparer ce mémoire.

Sommaire

Table des illustrations

1 Refonte des processus de développement chez un éditeur de logiciels.

Le projet décrit dans ce mémoire constitue la refonte des processus de développement logiciels. Il a été réalisé de septembre 2010 à décembre 2011 au sein de la société Gestimum, issue de la société EBP Informatique (Editeur de logiciels de gestion). Gestimum développe, maintient et commercialise le Progiciel de Gestion Intégré nommé GESTIMUM, développé en Pascal Objet avec l'IDE (Integrated Developemnt Environment) Delphi.

La société Gestimum comporte 5 départements qui sont Services, Développement, Fidélisation, Administration et Commercial, dont deux sont concernés par cette refonte:

- Le département Services
    - Ce département se subdivise en 2 services comptant en tout 3 personnes qui ont pour missions :

1. L'Assistance aux utilisateurs
    - Assister les utilisateurs dans les difficultés qu'ils peuvent rencontrer quand une action particulière ne donne pas les résultats escomptés. Cette difficulté peut survenir lors de l'utilisation de la solution ou en cas de méconnaissance fonctionnelle par exemple.
2. Réalisation des demandes spécifiques
    - Réaliser des documents commerciaux spécifiques en utilisant les logos et chartes graphiques des clients et en extrayant des données spécifiques de la base de données GESTIMUM afin de s'approcher au maximum des documents commerciaux utilisés par le client.

      La création de factures avec le logo du client, des calculs spécifiques de conditionnement (compter le nombre d'éléments par

unité de conditionnement par exemple) et les faire apparaître sur un document de vente (une facture ou un devis).

- Le département Développement logiciel, composé de 2 services soit 4 personnes au total.

  1. Le développement de nouvelles fonctionnalités

     - Développer les demandes d'évolution exprimées, qui sont soit de nouvelles fonctionnalités, soit une modification d'une fonctionnalité existante.

  2. La correction d'anomalies

     - Corriger un comportement non attendu, un dysfonctionnement constaté empêchant l'utilisation prévue de la solution développée.

C'est dans le cadre d'un travail d'équipe regroupant sept personnes que ces transformations ont pu être réalisées.

A mon arrivée au sein de la société Gestimum en septembre 2010 en tant que Directeur du Développement et des Services, j'ai pu constater qu'en raison d'un développement rapide de ses activités, le développement logiciel n'avait pas la qualité escomptée par les clients et n'avait pas de ligne directrice claire comme une feuille de route du logiciel par exemple.

En adoptant des nouveaux processus de développement nous pouvons espérer améliorer la qualité logiciel dans son ensemble et par voie de conséquence diminuer les coûts liés à sa maintenance.

1.1 Etat du développement logiciel

J'ai pu constater que le flux d'informations en provenance des clients et autres partenaires de la société était géré avec des outils peu adaptés au partage de l'information et à l'ouverture du système d'information vers l'extérieur.

Le flux d'informations en provenance des clients indiquait soit un dysfonctionnement (demande de maintenance) soit une demande d'amélioration ou d'évolution. Deux outils distincts traitaient ces informations : le PGI (Progiciel de Gestion Integré) GESTIMUM (module actions) et un autre outil nommé Devteam (chargé des projets de développement).

L'information se trouvait saisie à deux endroits simultanément et par deux personnes différentes ce qui engendrait des erreurs de copie et nuisait à son homogénéité.

Un grand nombre d'anomalies constatées par les clients n'était pas corrigées dans les versions qui sortaient.

Il n'y avait pas de priorités cohérentes dans les choix effectués pour la correction des anomalies, ces choix se faisant par rapport aux affaires signées par l'équipe commerciale.

Suivant les affaires en cours de négociation, les développements étaient dirigés en fonction de fonctionnalités non existantes dans le PGI GESTIMUM Certaines affaires ne pouvaient aboutir que si des développements étaient effectués, ajoutant ou modifiant une ou des fonctionnalités.

L'exemple de projet « Budgets pluri annuels » cité ci-après montre cet état de fait. Dans ce projet nous verrons que le PGI GESTIMUM ne possédait pas la fonctionnalité demandée par le client et pour obtenir l'affaire et satisfaire les besoins du client, nous avons dû créer un module à part entière de gestion de budgets pluri annuels.

Le client, les Chambres Régionales d'Agriculture du Maroc ou CRA avait besoin d'un module de gestion budgétaire dans lequel les budgets pouvaient être répartis sur plusieurs années (5 ans) et les engagements à venir pouvaient être imputés aux différents budgets précédemment créés.

Le PGI GESTIMUM possède bien un module de gestion budgétaire mais seulement sur une année glissante (une période allant de 12 à 23 mois consécutifs), et il n'y a pas de système de contrôle en temps réel des dépenses engagées.

De plus, ce client avait souhaité un système d'alerte de dépassement de budget en temps réel, que nous avons développé en intégrant un système de contrôle des dépenses engagées en temps réel.

Ce système définissait le cas échéant un seuil d'alerte et l'impossibilité de dépasser le budget défini dans le paramétrage du module. A chaque ligne saisie dans un document de vente (bon de commande, facture, etc), le montant engagé était vérifié afin de contrôler le total par rapport au seuil d'alerte prévu.

Le système prévenait donc l'utilisateur par un message qu'il avait dépassé le seuil d'alerte mais le laissait poursuivre jusqu'à atteindre la quantité maximale autorisée par le budget.

Les contrôles étaient effectués lors de chaque opération affectant les documents de vente, à savoir:

1. La saisie d'une ligne dans un document de vente.
2. La manipulation de documents entiers :
    a. La copie d'un document de vente
    b. Le transfert d'un document de vente (un bon de commande en facture par exemple)

Prenons l'exemple du transfert d'un document de vente existant.

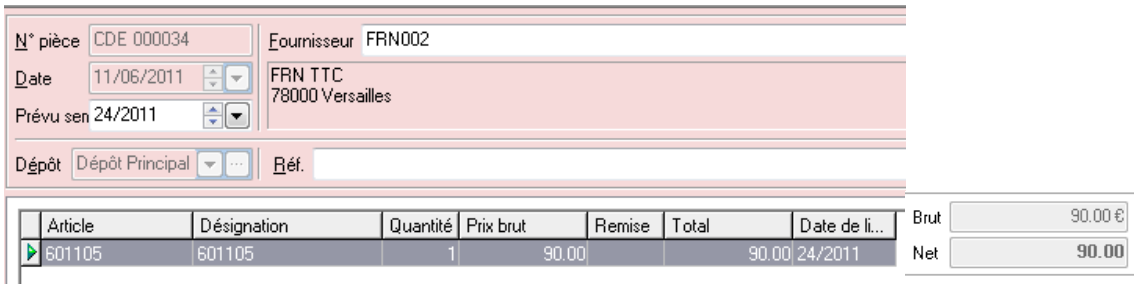


Figure 1. Un document de vente CDE 00034 et son total

La Figure 1. Un document de vente CDE 00034 et son total", montre un document de vente ayant une ligne déjà saisie (article 601105), son montant brut et net.

Nous tentons de transférer ce bon de commande en facture.

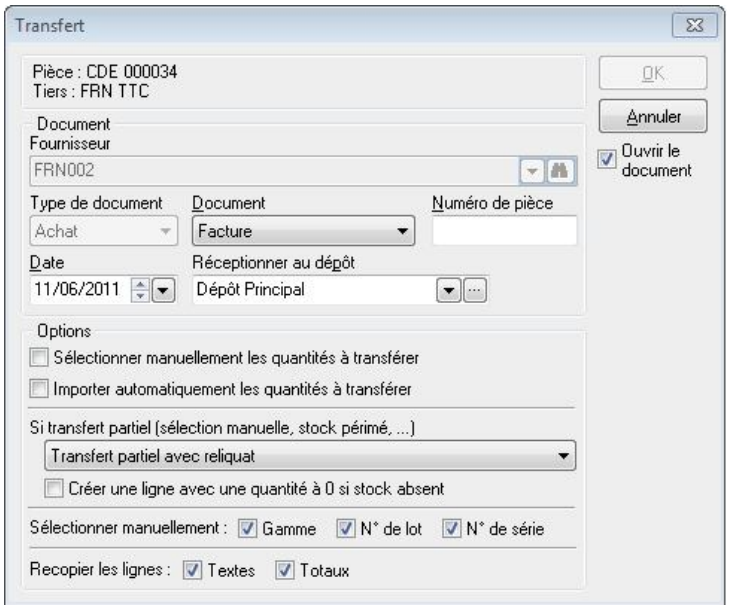


Figure 2. Transfert d'un document de vente

La Figure 2. Transfert d'un document de vente, montre la fenêtre de transfert du document de vente N°CDE000034 que nous essayons de transférer en une facture.

Lorsque nous appuyons sur le bouton OK, le système nous répond.

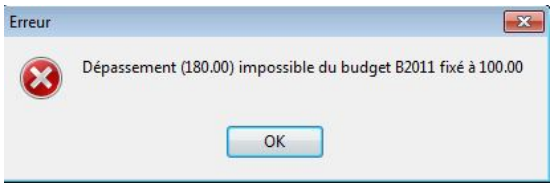


Figure 3. Message d'erreur dépassement budget

Dans la Figure 3. Message d'erreur dépassement budget, nous pouvons voir un message d'erreur affiché par le système, dans lequel il nous est indiqué que le budget B2011 est fixé à 100 et que le dépasser est impossible.

En effet nous tentons d'engager 180 (90 déjà engagés par un ou plusieurs utilisateurs plus 90 dans le document), ce qui est impossible d'après le paramétrage, le système nous prévient par un message d'erreur et annule l'opération de transfert que nous venons de tenter.

Ce système tenait compte de toutes les écritures effectuées par tous les utilisateurs simultanément et en temps réel.

Ce nouveau module a mobilisé l'équipe de développement pendant 6 mois durant lesquels nous n'avons pas pu corriger les anomalies déjà présentes ou nouvellement remontées à l'équipe de développement.

Un enjeu majeur de la société portait sur l'amélioration de la qualité logicielle. De ce fait j'ai proposé un ensemble de mesures qui visait à améliorer :

1. La qualité logicielle (*voir chapitre 3.4*).
2. La maintenabilité de la solution (*voir chapitre 3.2*).
3. Le nombre de régressions engendrées par les nouveaux développements (*voir chapitre 3.7*).
4. La visibilité des développements effectués et à effectuer (*voir chapitre 3.7*).
5. La feuille de route, plus claire et plus précise (*voir chapitre 3.7*).

Afin de pouvoir intégrer ces améliorations, j'ai initié un ensemble de mesures que j'ai présentées et faites valider par la direction.

L'ensemble de ces mesures se présente ainsi :

1. Un audit de l'existant, afin d'en comprendre les processus de développement utilisés, de les critiquer et de proposer les améliorations nécessaires (*voir chapitre 2.3*)

2. L'amélioration des processus de développement logiciel comprenant les demandes d'évolution et de maintenance ainsi que l'installation d'outils (ou moyens) de mesure permettant de contrôler l'efficacité de l'ensemble de méthodes mises en place. (*voir chapitre 3.1.1*)

3. L'amélioration de la qualité logicielle avec la mise en place des phases de recette et de tests de non régression automatiques. (*Voir chapitre 3.4*)

4. La fluidité des flux d'information par l'adoption d'un outil de gestion de cycle de vie logiciel. (*Voir chapitre 3.7.1*)

Ce plan a obtenu l'adhésion de l'équipe en place. Il s'appuie sur les standards tels qu'ISO 12207 et ISO 14764 qui constituent l'état de l'art en termes de développement logiciel et de maintenance.

L'objet du projet présenté dans ce mémoire consiste en la restructuration complète des processus de développement, de maintenance logicielle existante, en les adaptant aux standards du marché (ISO 12207 et ISO 14764) ainsi qu'en adoptant des outils pour la gestion de cycle de vie logiciel (Application Lifecycle Management ou ALM), la gestion de la phase de phase de recette et les tests de non régression automatiques. L'ALM permet de gérer l'ensemble du cycle de vie logiciel, depuis la conception jusqu'à la maintenance et passant par l'établissement de moyens de mesure de l'activité et du respect de la feuille de route définie.

Afin de satisfaire les nouvelles méthodes de travails j'ai eu la responsabilité de restructurer le département de développement logiciel afin de satisfaire aux exigences minimales du développement logiciel, en adoptant :

1. Des processus de développement logiciel standards.

2. En établissant des éléments de mesure objectifs afin de mesurer l'efficacité, la productivité des développements effectués et préparer l'avenir.

3. En mettant en place des méthodes de développement structurées (UML par exemple) qui permettent une meilleure appréhension des problèmes et une meilleure supportabilité des développements effectués.

Afin de tenir compte de l'aspect financier, deux personnes supplémentaires ont été embauchées pour ce projet.

Ce projet a été conçu par une équipe de 7 personnes réparties dans deux départements : développement et services. Mon travail a donc consisté à initier, diriger et clôturer l'ensemble des projets et diriger les équipes concernées, tout en gardant une part de développement logiciel pour continuer de satisfaire les demandes des clients.

En matière d'organisation, le processus de Gestion des demandes de développement devait-être refondu pour mettre en place un guichet unique et gérer ainsi le travail de l'équipe.

La restructuration des équipes ainsi que les méthodes de travail ont été revues pour satisfaire les exigences de la norme ISO 12207 qui propose un modèle de processus du cycle de vie logiciel.

La norme ISO 12207 impose un certain nombre de contraintes et l'adaptation à la taille de l'entreprise a été nécessaire. La réponse à la critique des processus existants en vue de l'adaptation de ces derniers à la norme ISO 12207, nous a amené à ne prendre que la partie de la norme correspondant aux équipes Service et Développement.

La méthode utilisée a été l'audit des processus existants (*voir chapitre 2.3*), la critique des résultats obtenus et la proposition d'un nouveau modèle de processus empruntés à la norme ISO 12207.

La recherche d'outils a été structurée pour tenir compte des impératifs de conformité au cahier des charges, maintenance et de budget, dans le sens où seuls les outils open source ont été sélectionnés.

Les outils utilisés pour l'audit ont été les cours d'Urbanisation des Systèmes d'Information et plus particulièrement la partie concernant le BPM (Bussines Process Management) suivi au CNAM.

En effet l'analyse des processus existants a été abordée comme des processus métier, étant donné que le métier de l'entreprise est le développement logiciel, le processus de production ne vise que la production du logiciel fabriqué par l'entreprise.

De ce fait l'activité de l'entreprise a pu être modélisée grâce à la modélisation des processus métier (Business Process Modeling), (*voir chapitre 2.3*)

La recherche d'outils de gestion du cycle de vie logiciel a été effectuée ainsi que l'adaptation aux besoins de l'entreprise de la solution choisie.

L'outil de gestion de cycle de vie logiciel a été sélectionné, il s'agit de la solution Redmine, développée en Ruby. J'ai adapté avec l'aide de 2 personnes la configuration de la solution aux besoins de l'entreprise

Les difficultés rencontrées ont été d'ordre managériales et non techniques, car il a fallu faire adopter les nouveaux processus de développement ainsi que le changement des méthodes de travail.

Même si tous les membres des équipes étaient favorables à une plus grande structuration de leur travail, il est toujours difficile de changer les habitudes de travail des membres d'une équipe.

Changer les outils et les méthodes de travail demande de la pédagogie (expliquer pourquoi on change et quels sont les bénéfices attendus du changement).

Obtenir l'adhésion des membres de l'équipe a été également un travail conséquent car les changements ne peuvent se faire rapidement mais plutôt progressivement afin de les faire adopter et pour que les nouvelles méthodes de travails soient bien appliquées il faut revoir leur application régulièrement.

Les processus de développement ont été modélisés sous forme de diagramme de flux (*voir chapitre 2.3*).

Ces diagrammes ont été montrés, expliqués et adoptés par tous lors de séances de travail dans lesquelles les schémas ont été élaborés, expliqués et finalement adoptés.

Le choix de l'outil de gestion de cycle de vie logiciel a fait l'objet d'un travail d'équipe afin que le choix soit fait grâce à des critères objectifs (voir chapitre 3.7.1 Choix d'un ALM) dont chaque membre a participé à l'élaboration.

Le choix de l'outil de tests automatiques dont l'objectif est l'automatisation des tests de non régression a également été structuré sous la forme d'une grille de critères, qui nous a permis de choisir en toute objectivité la solution la plus adaptée aux besoins de l'entreprise.

Nous avons également initié le projet de création des scénarii de tests et la définition des cas de tests afin d'alimenter les tests automatiques.

2 Situation des processus de développement.

Nous trouverons dans ce chapitre l'état des processus de développement lors de ma prise de fonctions au sein de la société Gestimum.

J'ai commencé par analyser l'ensemble des processus de développement, afin de déceler les carences éventuelles et ainsi pouvoir proposer des solutions pour :

2.1.1 Mettre en place des éléments de mesures objectifs

La mise en place d'éléments de mesures permet de quantifier le travail effectué, d'améliorer la performance par comparaison et par voie de conséquence de diminuer les coûts.

En quantifiant le nombre d'anomalies corrigées par exemple, il est possible de communiquer sur la prise en compte des demandes des clients.

Une note de version, contient le nombre de corrections, le nombre de fonctionnalités ajoutées.

En mesurant le nombre d'anomalies constatées d'une version à une autre, il est possible de mesurer le degré de maturité de la solution. A savoir : si une version contient de moins en moins d'anomalies alors on peut considérer que son degré de maturité a augmenté.

Les ressources mises à ma disposition ne me permettaient pas de me consacrer exclusivement à des corrections, il fallait aussi tenir compte des demandes d'évolution.

Toujours à cause des ressources dont nous disposions, nous avons

été obligés d'entamer une phase de stabilisation dans laquelle seules des corrections ont été effectuées, cela dans le but d'améliorer la perception de la solution par les clients.

2.1.2 Améliorer l'efficacité des développements.

L'efficacité des développements peut s'améliorer en adoptant des méthodes de développement simples contenant les éléments essentiels à un développement. Nous avons pour cela écrit des spécifications fonctionnelles, des spécifications organiques ou techniques et une phase de recette.

2.1.3 Mieux décrire les demandes via des spécifications fonctionnelles

Les spécifications fonctionnelles constituent la base de tout développement, elles sont la pierre angulaire de tout processus de développement.

En effet à partir des spécifications fonctionnelles on pourra déduire les spécifications organiques ou techniques.

Chaque fonctionnalité est décrite d'un point de vue utilisateur, en indiquant au développeur l'objectif de la fonctionnalité à coder.

Le développeur peut ainsi décider de la façon dont il va satisfaire l'objectif fixé par la spécification.

Dans les spécifications organiques on trouve des fonctions internes au code qui vont participer à la réalisation de la fonctionnalité décrite.

La qualité logicielle est également impactée par les spécifications fonctionnelles dans la mesure où chaque fonction décrite sera testée d'après sa définition.

Par la suite le manuel d'utilisation sera déduit des fonctionnalités décrites dans les spécifications fonctionnelles.

2.1.4 Améliorer la qualité logicielle

La qualité logicielle peut être améliorée par l'adoption de méthodes de développements systématiques, à savoir des tests effectués d'après les spécifications fonctionnelles.

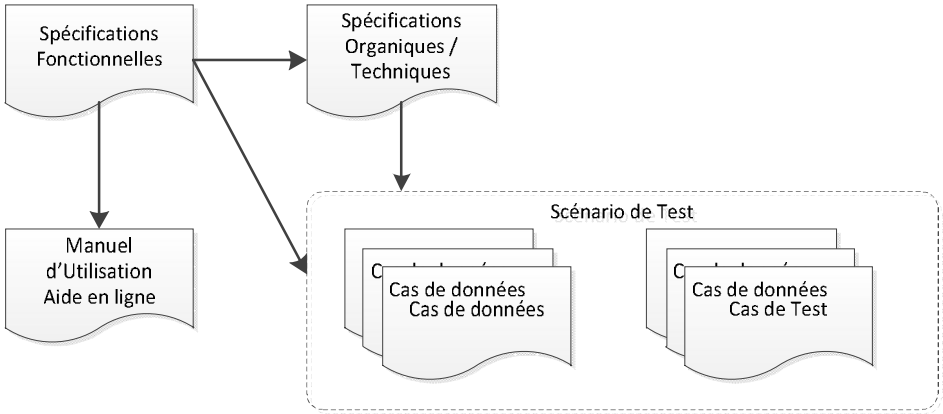


Figure 4. Spécifications fonctionnelles

Nous pouvons voir dans la Figure 4. Spécifications fonctionnelles, comment d'après les spécifications fonctionnelles les scénarii de tests peuvent être déduits.

Dans la phase de recette nous établissons clairement les consignes de tests et déterminons le temps nécessaire pour les effectuer.

2.1.5 Accroitre la productivité de l'équipe de développement

La productivité de l'équipe peut s'améliorer lorsque des objectifs clairs (spécifications et temps) sont définis.

A partir d'éléments de mesure mis en place, il est possible de quantifier objectivement ce qui est demandé aux membres de l'équipe de développement.

2.1.6 Améliorer la maintenabilité du produit

La maintenabilité est, dans le domaine informatique, la capacité pour des composants ou des applications à être modifiés, en vue d'apporter des corrections, de manière cohérente et à moindre coût. Plus généralement, dans l'industrie le terme exprime la capacité d'un système à être simplement et rapidement réparé et ainsi à diminuer les temps et les coûts d'intervention.

La maintenabilité du produit peut s'améliorer en adoptant des méthodes de développement standards du marché, comme par exemple l'utilisation de la modélisation UML pour l'écriture des spécifications fonctionnelles.

2.1.7 Augmenter la satisfaction des clients.

La satisfaction client a pu être augmentée car les développements allaient mieux correspondre à leur demande (spécifications validées par les clients), la qualité étant assurée par une phase de recette.

Les demandes d'évolution ainsi que les demandes de maintenance étaient soumises au service assistance sous des formes très diverses, à savoir :

1. Courriels expliquant brièvement la demande formulée par le client (interne ou externe).

   Par exemple : « J'ai besoin de limiter l'accès aux pièces commerciales »

   Dans cet exemple la demande n'est pas assez explicite pour être traitée. Nous ne savons pas de quelles pièces commerciales il est question, dans quel module du produit la limitation doit être effective, qui est concerné par la limitation et de quelle façon la limitation doit être faite.

2. Documents peu formatés ne comportant qu'une vue parcellaire de la demande (vue du rédacteur).

3. Courriels ne donnant que le titre de la fonctionnalité à développer et uniquement compréhensible par le demandeur !

Toutes ces demandes étaient formulées et aucune confirmation de la compréhension du besoin ou validation de la part des récepteurs n'était faite.

De ce fait les développements pouvaient ne pas correspondre aux souhaits du demandeur, ce qui conduit finalement à un développement non conforme et qui engendre des malfaçons ou des régressions.

Ces demandes étaient saisies dans un outil appelé Devteam (développé par la société mère EBP) dont le but était la gestion des projets de développement mais ne permettait pas l'élaboration de moyens de mesure objectifs de l'activité du département.

Par exemple il était impossible de savoir combien de demandes d'évolution avaient été formulées et corrigées en un lapse de temps.

L'absence d'indicateurs ne permettait pas de quantifier le travail fourni par le service développement. Or des indicateurs pertinents permettent de proposer des améliorations quantifiables et objectives par rapport à une quantité de demandes d'amélioration et d'évolutions effectuées.

Par exemple, avec des indicateurs pertinents (nombre de demandes par unité de temps), il est possible de calculer le nombre de demandes effectuées par un développeur en un temps donné, ce qui permet de calculer le temps moyen nécessaire pour coder une fonctionnalité.

Du fait de l'absence d'indicateurs il n'était pas possible de quantifier les ressources nécessaires, au niveau de l'équipe d'assistance mais aussi au niveau de l'équipe de développement.

Il était impossible donc de pouvoir donner le temps moyen de résolution d'un problème, ou le nombre de problèmes associés à une personne.

Un aspect crucial de la qualité de la solution, était que le service développement se trouvait submergé par des demandes sans spécification.

Non seulement le nombre de demandes était important mais aussi la qualité des demandes, car sans avoir les informations nécessaires et la bonne compréhension du problème il était très difficile de produire des développements de qualité.

Une conséquence était que, dans certains cas, une fonctionnalité pouvait être développée de la façon dont le développeur avait compris la demande et fait ses propres choix.

Il est à noter que le fait de ne pas avoir de spécifications, que ce soit pour les demandes de maintenance ou les demandes d'évolution, en provenance des clients internes (membres de Gestimum) ou externes (utilisateurs de la solution) ou de la part du service développement engendre plusieurs phénomènes :

1. Une situation paradoxale, car plus on demande de corriger (demandes de maintenance) ou de faire évoluer le produit (demandes d'évolution) sans clairement spécifier la demande et plus il y a des sollicitations à l'assistance. Car ces demandes étaient sensées améliorer le produit mais le résultat effectif est que malgré ces modifications le produit continue à engendrer des appels à l'assistance.

2. Un développement anarchique de la solution et dénué de contrôle de cohérence de l'ensemble, car il n'y a pas de ligne directrice (Roadmap) claire dans les développements.

   Un développement anarchique peut conduire à développer une nouvelle fonction qui fait le contraire d'une autre fonction déjà développée, ce qui peut conduire à postériori à des régressions (*voir chapitre 3.3.1 la non régression*), puisqu'aucune cohérence n'est assurée dans la suite des développements.

3. Un autre aspect de cette situation était l'absence d'une phase de recette bien déterminée, après les développements (maintenance ou évolution) car les

tests effectués reposaient essentiellement sur la bonne volonté des membres de l'équipe d'assistance.

Le temps passé par l'équipe assistance, à répondre à des sollicitations, ayant pour origine les régressions, pouvait être diminué en améliorant la qualité logicielle.

La régression en développement logiciel est une anomalie involontaire engendrée dans un programme lors de la modification ou l'ajout d'une nouvelle fonctionnalité.

Enfin une conséquence de cette organisation était l'absence, par manque de temps et de compétences en interne, d'une phase de tests de non régression, qui est primordiale afin de détecter les anomalies engendrées involontairement par un développement.

Afin de mieux gérer, d'organiser et de centraliser les demandes, et le cycle de vie logiciel, un processus d'acquisition d'un outil de gestion de vie logiciel ou ALM (Application Lifecycle Management) a été initié.

De plus un processus d'acquisition d'une solution de tests automatiques a été démarré également.

Les tests automatiques permettent de tester systématiquement toutes les fonctions d'un programme préalablement testé unitairement (chaque fontion est testée séparément), après une modification, pour s'assurer que des défauts n'ont pas été introduits ou découverts dans des parties non modifiées du logiciel (*voir chapitre 4.5 la non régression*).

Le logiciel de tests automatiques a besoin de scénarii qui décrivent les actions à effectuer et les contrôles à effectuer. L'écriture de ces scénarii ont fait l'objet d'un projet également.

Ces tests sont effectués quand le logiciel ou son environnement est modifié.

Dans le cas qui nous concerne, l'environnement du logiciel peut être le système d'exploitation qui exécute la solution GESTIMUM, Windows XP ou Windows 7 ne réagissent pas du tout de la même façon, face à des applications de type MDI (Multiple Document Interface ou Document à Interface Multiple) comme la solution GESTIMUM.

Une application MDI en Informatique, désigne l'organisation de l'interface graphique d'une application où des fenêtres "parentes" contiennent en leur sein des fenêtres enfants.

Le cas typique d'une application MDI consiste en la fenêtre principale de l'application, avec un menu et des barres d'outils, contenant une (sous-)fenêtre par fichier ou projet ouvert.

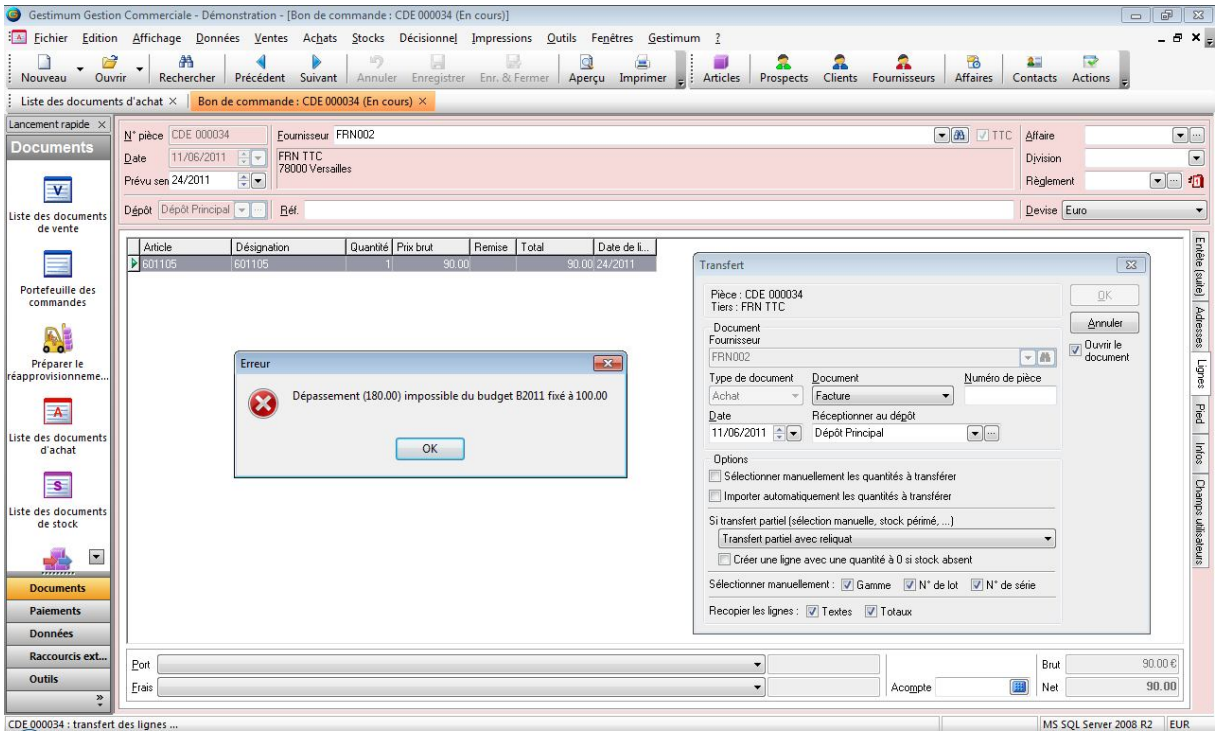


Figure 5. Application MDI

Nous pouvons voir dans la Figure 5. Application MDI", que la fenêtre principale contient la fenêtre « Bon de commande », un menu, et une barre d'outils.

La fenêtre active est signalée par un bouton de couleur orange, comme indiqué en Figure 6. Application MDI, fenêtre active :

Figure 6. Application MDI, fenêtre active

L'ensemble de ces processus sont inspirés d'ISO 12207 et adaptés au contexte et aux exigences de l'entreprise.

2.2 Etat de l'Art à prendre en compte

Le standard à prendre en compte est la norme ISO 12207 qui est un modèle de processus pour le développement logiciel.

Les processus de cycle de vie logiciel définis par ISO 12207 ont pour objectif de maintenir, de s'assurer de la présence des stratégies, des processus de cycle de vie logiciel et des procédures à usage de l'entreprise.

Mon rôle est de proposer les modifications nécessaires afin d'améliorer la qualité des développements. En m'appuyant sur les processus définis par ISO 12207 j'ai pu constater que l'adoption de certains processus adaptés à la structure de l'entreprise pouvait améliorer considérablement la qualité logicielle.

En effet, l'adoption de processus tels que ceux définis par ISO 12207 contribuent à l'amélioration globale de la qualité logicielle. Elle donne les outils nécessaires pour contrôler la qualité des livrables (documentation des développement, spécifications fonctionnelles et organiques). Elle assure que les décisions prises en termes de processus sont toujours respectées (respecter et faire

respecter les processus mis en place), que le cycle de vie logiciel soit respecté et que l'acquisition de nouveau produits est encadrée par des actions bien définies et contrôlées.

Afin de respecter les processus de cycle de vie logiciel définis par ISO 12207, un ensemble d'actions a été entrepris, à savoir :

1. Les procédures définissant les actions de chacun ont été formalisées sous forme d'un document illustré par la Figure 7. Cycle de vie d'une demande et mis à la disposition des équipes.

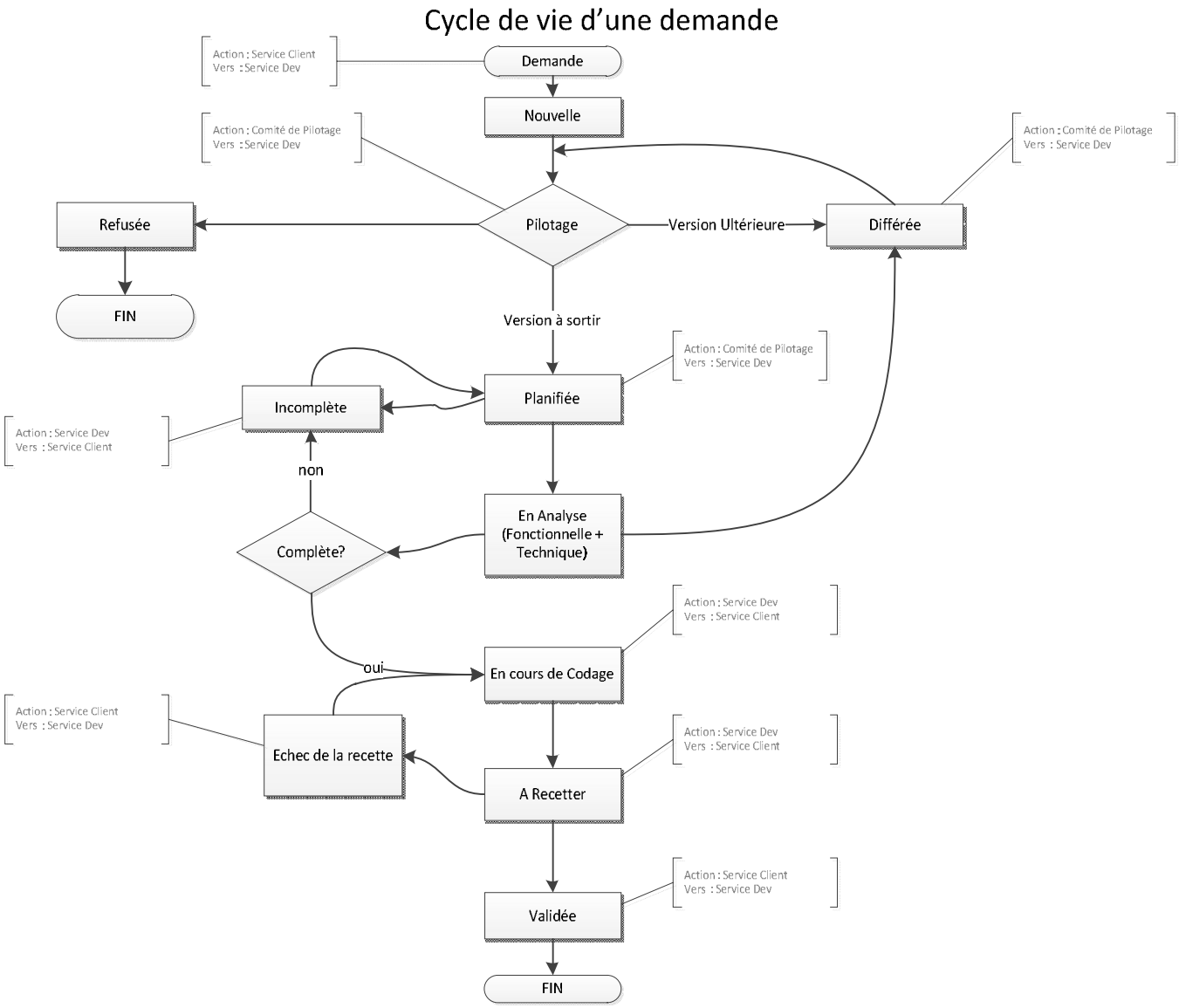


Figure 7. Cycle de vie d'une demande

Dans la Figure 7. Cycle de vie d'une demande", nous voyonsles différentes étapes qu'une demande peut suivre, avec les actions que chaque acteur doit effectuer et pour qui il doit les faire.

2. La responsabilité et les actions que chacun des acteurs doit exécuter à l'intérieur des procédures (sous forme d'un workflow) ont été clairement explicitées.

Figure 8. Cycle de vie, représentation d'une responsabilité

La Figure 8. Cycle de vie, représentation d'une responsabilité" nous montre que l'action de création d'une demande est faite par le département services et à destination du département développement.

La documentation des processus et des procédures afférentes ont été ajoutées au référentiel lié aux processus. Ainsi mis à disposition dans un endroit centralisé, les modifications, revues et améliorations étaient facilement accessibles.

2.2.1 Etat de l'art

« La standardisation est un processus par lequel on réfère un indice à une norme afin d'en comprendre le sens intégré dans un tout représentatif. »[i]

« Pour Henry Mintzberg, la standardisation est un procédé en entreprise qui permet de gagner du temps et de réduire les coûts. »[ii]

« Une norme industrielle est un référentiel publié par un organisme de normalisation comme Afnor, CEN, ISO, OASIS, ISO, IEC.

Comme la langue anglaise ne marque pas la différence entre norme et standard (« norme » se dit « standard » en anglais), on parle pour les normes de standards de jure et pour les simples standards de standard de facto.

La normalisation ou la standardisation est le fait d'établir respectivement des normes et standards industriels, c'est-à-dire un référentiel commun et documenté destiné à harmoniser l'activité d'un secteur. Elle est réalisée par des organismes spécialisés, qui sont le plus souvent soit des organismes d'État, soit des organisations créées par les professionnels d'un secteur d'activité donné.

Le développement logiciel est une activité qui requiert de la rigueur et de la méthode afin de transformer un besoin en une réalité informatique qui elle, satisfait une réalité économique. [iii]

Les organismes de standardisation tel que l'ISO et IEC ont établi des normes ou standards afin de poser un référentiel commun pour établir des processus de développement logiciel simples et efficaces, dans les trois grandes catégories qui sont :

- Les processus de base

  Dans le cas qui nous intéresse seul le processus « Développement du logiciel » a été pris en compte comme processus de base.

- Les processus de support

- Les processus organisationnels

2.2.2 ISO 12207

La norme ISO 12207 a pour objectif de poser la référence pour les processus du cycle de vie logiciel pris dans sa généralité avec des processus de base, des processus supports et des processus organisationnels. Cette norme décrit donc un modèle de processus.

Le but des modèles de processus de cycle de vie logiciel est de définir, de maintenir et d'assurer la disponibilité :

- des politiques ou procédés mis en place,

- des processus du cycle de vie,

- des modèles de cycle de vie,
- des procédures à destination de l'organisation.

Ces processus fournissent des politiques ou procédés de cycle de vie logiciel, les processus et procédures compatibles avec les objectifs de l'organisation, qui sont définis, adaptés, améliorés et maintenus pour soutenir les besoins des projets individuels dans le cadre de l'organisation, et sont susceptibles d'être appliqués en utilisant des méthodes efficaces et des outils éprouvés.

Les livrables de ce processus sont :

La résultante de l'implémentation réussie des processus de management de cycle de vie se décompose en :

1. Formalisation sous forme de politiques et procédures pour le management et le déploiement des modèles de cycle de vie.
2. Définition des responsabilités et de l'imputabilité.
3. Définition, maintenance et amélioration des processus de cycle de vie, modèles et procédures à usage de l'organisation.
4. Implémentation, priorisation, amélioration des processus.

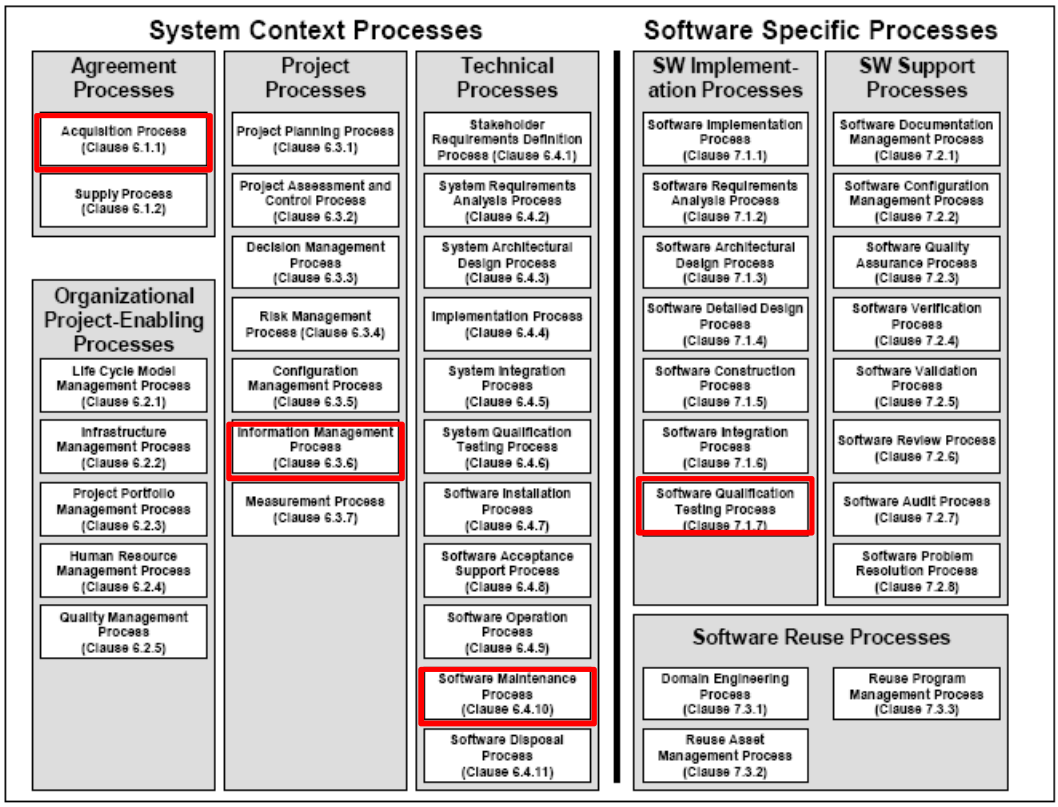


Figure 9. ISO 12207

ISO 12207 présente un ensemble de processus cités dans la Figure 9. ISO 12207", nous avons sélectionné (les processus encadrés en rouge) les processus les plus adaptés à la situation des équipes présentes au moment du choix.

Nous avons donc utilisé la partie « Acquisition process » ou processus d'acquisition pour ce qui est du choix de la solution de gestion de cycle de vie logiciel, ainsi que la solution de tests automatiques.

Nous avons utilisé la partie « Information Management Process » ou Processus Management de l'Information pour ce qui est de la partie qui concerne les flux d'informations dans la gestion de cycle de vie logiciel.

Pour la partie Tests nous avons utilisé la partie « System Qualification Testing process » ou Processus du Système Qualification et de Test.

L’objectif de ce processus est de confirmer que le logiciel produit et/ou service d’un processus ou un projet reflète les spécifications prédéfinies.

La partie maintenance s’est inspirée du processus « Software Maintenance Process » ou Processus de Maintenance Logiciel.

En génie logiciel, la maintenance du logiciel (ou maintenance logicielle) désigne les modifications apportées à un logiciel, après sa mise en œuvre, pour en corriger les défauts, en améliorer l'efficacité ou autres caractéristiques, ou encore adapter celui-ci à un environnement modifié (ISO/IEC 14764).

2.2.3 ISO 14764

Nous avons amélioré le développement logiciel en utilisant ISO 12207, la maintenance étant une tâche importante des équipes de développement logiciel réduites comme celles de la société Gestimum.

En génie logiciel, la maintenance du logiciel (ou maintenance logicielle) désigne les modifications apportées à un logiciel, après sa mise en œuvre, pour en corriger les fautes, en améliorer l'efficacité ou autres caractéristiques, ou encore adapter celui-ci à un environnement modifié.

Cette norme internationale distingue six processus de maintenance logicielle :

1. L'implémentation;
2. L'analyse et la résolution de problèmes;
3. La modification du logiciel;
4. L'acceptation de la modification par le demandeur;
5. La migration;
6. Et finalement, la mise à la retraite.

Il y a un certain nombre de processus, d'activités et de règles qui sont propres à la maintenance du logiciel, comme par exemple :

1. La transition : passation coordonnée du logiciel, de l'équipe de développement à l'équipe de maintenance;
2. Les ententes de services applicatives;
3. La priorisation des requêtes de modification et des rapports de problèmes;
4. L'acceptation ou le rejet d'une demande de modification selon le travail et la complexité plus ou moins grandes qu'elle implique; son renvoi à l'équipe de développement.

Une idée répandue est que le mainteneur ne fait que corriger des défauts (anomalies). Pourtant des études et des enquêtes indiquent depuis des années que plus de 80% des efforts de maintenance sont consacrés à des interventions autres que correctives, comme la modification d'une fonctionnalité. (Pigosky 1997[iv]).

Pigosky affirme dans son livre paru en 1997 que le montant de temps et d'argent dépensés en débogage, en personnalisation, en mise à jour, et en maintenance des logiciels dépasse de loin le montant dépensé pour l'acheter.

Cette notion est perpétuée par des utilisateurs qui font état de problèmes alors qu'ils ont affaire, en réalité, à des améliorations fonctionnelles du logiciel.

La maintenance et l'évolution des logiciels furent abordées pour la première fois par le Dr. Lehman, en 1969[v]. S'étendant sur une vingtaine d'années, ses recherches aboutirent à la formulation de huit règles de l'évolution d'un logiciel (1997). Elles ont mis en lumière le fait que la maintenance est un processus évolutif et que les logiciels évoluent avec le temps. En évoluant, ils deviennent plus complexes à moins qu'une action spécifique soit engagée pour en réduire la complexité.

1. (1974) Changement continu — Les systèmes informatiques doivent être continuellement adaptés ou ils deviennent progressivement moins satisfaisants.

2. (1974) Accroissement de la complexité — L'évolution des systèmes informatiques induit un accroissement de sa complexité à moins que du travail soit effectué pour la maintenance ou la réduction de la complexité.

3. (1974) Auto régulation — Les processus d'évolution des systèmes informatiques est auto régulé avec la distribution des produits et des processus de mesure proche de la normale.

4. (1978) Conservation de la stabilité organisationnelle (rythme de travail invariant) — Le taux effectif moyen de l'activité globale dans un système informatique qui évolue est invariant pendant travers le cycle de vie du produit.

5. (1978) Conservation de la connaissance — A mesure qu'un système informatique évolue, tout lui est associé, les développeurs, le personnel de vente, les utilisateurs, par exemple, sont tenus de maintenir la maîtrise de son contenu et d'effectuer les actions nécessaires afin d'assurer une évolution satisfaisante. Une croissance excessive diminue la maîtrise. La croissance incrémentale moyenne reste inchangée avec l'évolution du système.

6. (1991) Croissance Continue — Le contenu fonctionnel doit croitre continuellement afin de maintenir la satisfaction des utilisateurs au-delà du cycle de vie.

7. (1996) Déclin de la qualité — La qualité d'un système informatique semble décliner à moins qu'il soit rigoureusement maintenus et adaptés aux changements d'environnements opérationnels.

8. (1996) Système de Retour (d'abord défini en 1974 puis formalisé en tant que loi en 1996) — Processus d'évolution d'un système informatique multi-niveaux, multi-boucles, des systèmes de rétroaction multi-agents et doit être traité comme telle afin d'atteindre une amélioration significative sur toute base raisonnable.

E.B. Swanson a identifié, au départ, trois catégories de maintenance : la corrective, l'adaptative et la perfective.

Ces catégories ont été mises à jour par l'équipe de ISO/IEC 14764, avec l'ajout d'une quatrième catégorie :

1. La maintenance corrective : modification d'un progiciel effectuée après livraison afin de corriger les défauts rencontrés.

2. La maintenance adaptative : modification d'un progiciel effectuée après livraison pour qu'il reste utilisable dans un environnement qui change ou a changé.

3. La maintenance perfective : modification d'un progiciel effectuée après livraison pour en améliorer l'efficacité ou la maintenabilité.

4. La maintenance préventive : modification d'un progiciel effectuée après livraison pour en déceler et corriger les défauts latents avant qu'ils ne se manifestent.

Les problèmes majeurs de la maintenance du logiciel sont autant gestionnels que techniques.
Les problèmes de gestion sont :

1. l'alignement sur les priorités de la clientèle,
2. le choix des employés,
3. Le choix du responsable de la maintenance,
4. Justification de la valeur ajoutée et les coûts de l'équipe.

Les problèmes techniques sont liés à l'insuffisance

1. de la compréhension du logiciel,
2. de la documentation,
3. des tests,
4. de la mesure de la maintenabilité.

Les modèles de maturité du savoir-faire qui visent spécifiquement la maintenance du logiciel sont :

1. Le modèle de maturité de la maintenance du logiciel S3M (avril 2006), qui vise la petite maintenance logicielle[vi].

   Ce modèle S3M a été créé par une équipe composée de chercheurs et d'industriels. Il est disponible gratuitement sur le site du S3M.

   Pour le développement et les projets de maintenance le, CMMi[vii] est plus approprié.

   Il est maintenant largement employé par les entreprises d'ingénierie informatique, les directeurs des systèmes informatiques et les industriels pour évaluer et améliorer leurs propres développements de produits.

   Pour améliorer les processus d'infrastructure et d'opération l'ITIL[viii] est préférable.

   En effet ITIL (Information Technology Infrastructure Library pour « Bibliothèque pour l'infrastructure des technologies de l'information ») est un ensemble d'ouvrages recensant les bonnes pratiques (« best practices ») pour le management du système d'information.

   Rédigée à l'origine par des experts de l'Office public britannique du Commerce (OGC), la bibliothèque ITIL a fait intervenir à partir de sa version 3 des experts issus de plusieurs entreprises de services telles qu'Accenture,

Ernst & Young, Hewlett-Packard, Deloitte, BearingPoint ou PricewaterhouseCoopers.

C'est un référentiel très large qui aborde les sujets suivants :

1. Comment organiser un système d'information ?
2. Comment améliorer l'efficacité du système d'information ?
3. Comment réduire les risques ?
4. Comment augmenter la qualité des services informatiques ?

Le modèle S3M:

- donne des références aux modèles de références.
- couvre ISO 12207 (utilisé dans ce projet).
- couvre la norme internationale ISO 14764 de la maintenance du logiciel (utilisé dans ce projet).
- couvre ISO 90003:2004;
- couvre les articles pertinents du CMMi;
- est conforme à la norme ISO 15504 (SPICE).

2. Le modèle de maturité de la maintenance corrective (Kajko-Mattsson 2001).

Kajko-Mattsson et al. (2001) proposent le « Corrective Maintenance Maturity Model » (CM3) pour les mainteneurs Éducation et formation, un modèle de maturité pour le développement effectif de la maintenance des logiciels ingénieurs. Ce modèle de maturité est basée sur deux processus éducatifs de l'industrie et plusieurs modèles génériques de processus, parmi lesquels la CMM. Il définit trois niveaux de maturité: initial, défini et optimal. Ce modèle de maturité concerne la formation continue dans un environnement de l'industrie, qui est radicalement différent de celui d'un organisme éducatif. En revanche, le contexte de l'éducation n'est pas l'objet principal du CM3.

2.3 Processus à améliorer

Les processus de développement utilisés chez Gestimum, avant mon arrivée, étaient assez simples, dans la mesure où une seule personne était chargé de faire développer les nouvelles fonctionnalités ou de modifier des anomalies constatées par les utilisateurs (les clients) ou les utilisateurs internes de la société.

La taille de la structure imposait un certain nombre de contraintes, telles que :

1. Ressources limités de l'équipe de développement
2. Ressources limités de l'équipe de maintenance
3. Ressources limités de l'équipe services.

La modélisation du processus de développement existant, est présentée dans la Figure 11. Processus de développement avant modification".Elle montre toutes les étapes suivies à l'époque pour arriver à faire développer soit une nouvelle fonctionnalité, soit un correctif.

Il est à noter que ce processus ne contient pas les phases de développement classiques que l'on pourrait s'attendre à trouver au sein d'une équipe de développement logiciel. Celacaractérise le type de méthodes employées par la société à ce moment-là.

Ce minimalisme des processus de développement, était justifié par la taille de l'entreprise, dixit les dirigeants, et de ce fait toutes les tentatives passées, de mise en place de processus de développement plus performants ou plus optimisés, se sont soldées par des échecs, car les processus mis en place étaient jugés soit trop contraignants, soit non adaptés à la structure de l'entreprise.

Une conséquence logique et évidente de cette situation était une insatisfaction des clients et partenaires (sociétés d'informatique qui étaient en charge de distribuer la solution GESTIMUM auprès de leur clients), car la qualité logicielle n'était pas à la hauteur de leurs attentes ni des clients.

En effet les clients formulaient des demandes mais de celles-ci ne voyaient que peu ou pas du tout le résultat de leurs requêtes dans les versions qui sortaient.

Ainsi j'ai eu l'occasion de parler avec un client suisse, en octobre 2010, qui avait constaté une anomalie en janvier 2007 et cette dernière n'était toujours pas corrigée.

Les besoins des clients n'étaient pas formalisés sous quelque forme que ce soit et seul la retranscription d'une conversation dans un courriel était saisie et envoyée au service développement.

Aucune spécification fonctionnelle n'étaient produite ni aucune expression de besoins non plus.

Ce type de fonctionnement dénote d'un manque de méthode apparent, car la demande n'était pas analysée d'un point de vue fonctionnelle et on ne s'assurait pas non plus que l'on avait bien compris le besoin du client.

Le service développement recevait donc des demandes de façon anarchique, non discriminée, non formatée, non priorisée et parfois sans contexte. Il s'en suivait une phase de tri des demandes. Tri effectué par une personne qui choisissait en fonction des critères qui lui étaient propres parmi l'ensemble des demandes et constituait ainsi le portefeuille des demandes pour la version à sortir.

Les développements avançaient de cette façon, ce qui engendrait de la frustration de la part des clients qui avaient formulé leurs demandes (souvent depuis longtemps) et qui ne voyaient pas les résultats car leur demande n'avait pas été choisie. Il est ainsi arrivé qu'un client attende plusieurs années pour qu'une de ses demandes soit finalement prise en compte, alors qu'elle avait été jugée opportune au moment où il l'avait formulée, mais seulement elle n'avait pas été désignée comme faisant partie de la prochaine sortie, et ce pendant trois ou quatre ans.

La phase de recette n'était pas formalisée et le temps nécessaire à cette phase n'était pas alloué à l'équipe d'assistance. C'est sur leur « temps libre » que les

tests étaient susceptibles d'être effectués. Ce qui a conduit bien souvent à livrer une version non testée, d'où des régressions importantes.

L'absence de :

a. Spécifications fonctionnelles

b. Spécifications Organiques ou techniques

c. Documentation

d. Phase de recette clairement établie

e. Tests de non régression

Tous ces manques m'ont conduit à proposer les changements décrits dans ce mémoire.

Les spécifications fonctionnelles sont la pierre angulaire de tout le cycle de vie logiciel, car on peut déduire :

1. Les spécifications organiques (avec les schémas UML éventuels) ou techniques.

   Les spécifications fonctionnelles décrivent la fonctionnalité à produire et le développeur peut ainsi décider de quelles fonctions il aura besoin pour produire la fonctionnalité décrite.

2. Le plan de tests

   La description d'une fonctionnalité sert de guide à l'équipe de tests.

3. L'aide en ligne ou le manuel de l'utilisateur.

   Les fonctionnalités décrites dans la spécification fonctionnelle servent aux utilisateurs.

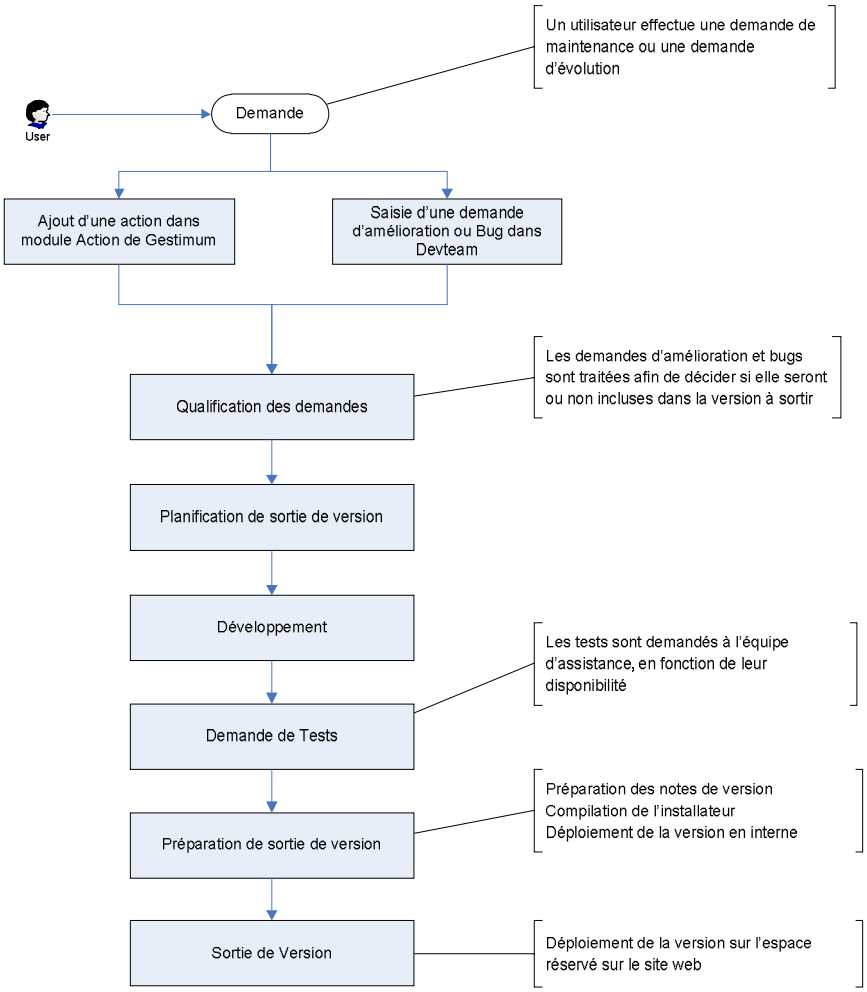


Figure 10. Cycle de vie logiciel utilisé avant modification

Dans la Figure 10. Cycle de vie logiciel utilisé avant modification", on peut voir qu'une demande était saisie dans 2 outils différents (l'ERP GESTIMUM et l'outil Devteam), puis il s'en suivait un sélection unilatérale des éléments à inclure dans la prochaine version, modification du planning de sortie, développement, demande de tests et préparation de sortie.

Toutes ces étapes (choix du contenu de la version par exemple) étaient conduites par une seule et même personne, sans concertation avec les autres membres des autres équipes.

Un autre aspect de ce processus à améliorer était le développement de la modification elle-même.

La critique que l'on peut faire de ce processus est que :

1. Aucune phase de spécification n'était présente.

2. Aucune phase de tests (mise à part la demande de tests) n'était présente.

3. Aucune phase de « debug » ou aller-retour avec le développement n'était présente.

4. Aucune phase de tests de non régression n'était présente.

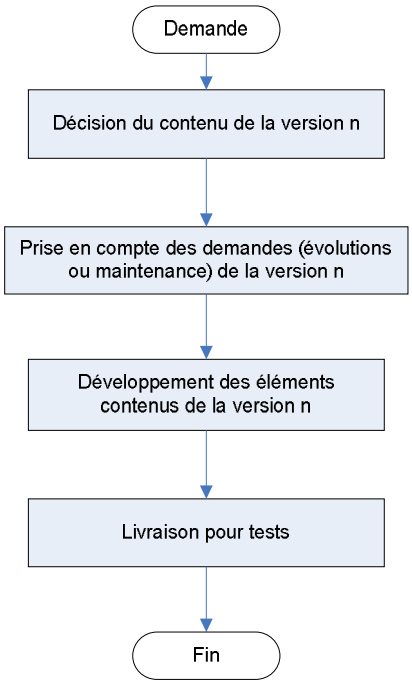


Figure 11. Processus de développement avant modification

La Figure 11. Processus de développement avant modification", nous montre la particularité de cette structure (de par sa taille) consistait dans des développements qui prenaient vie après une phase d'analyse extrêmement réduite, voire inexistante dans certains cas, sans documentation, sans analyse et sans aucune modélisation (UML ou autres).

En effet, cette phase était réduite à un document qui rassemblait, à la fois les besoins exprimés sous forme de copie d'un courrier électronique et les commentaires du rédacteur, le tout sans analyse fonctionnelle à proprement parler.

Par ailleurs une double saisie des demandes était effectuée dans le PGI (module Actions) et l'outil de gestion Devteam.

2.4 Outils Utilisés (situation avant modification)

La gestion des demandes de maintenance et autres demandes d'améliorations se faisait par l'ajout d'actions liées à un client dans le module actions de l'outil GESTIMUM Gestion Commerciale.

Chaque action au sens GRC (Gestion de Relations Client) du terme correspond donc à une demande et à la réponse apportée par la personne qui a traité la demande.

Chaque demande ou action d'un client était saisie dans cet outil afin de pouvoir revenir par la suite en cas de besoin. Un historique était ainsi gardé pour référence ultérieure.

La saisie des actions dans l'outil GESTIMUM était une des priorités des membres de l'équipe d'assistance car cette source d'information était centralisée dans la base de données de GESTIMUM.

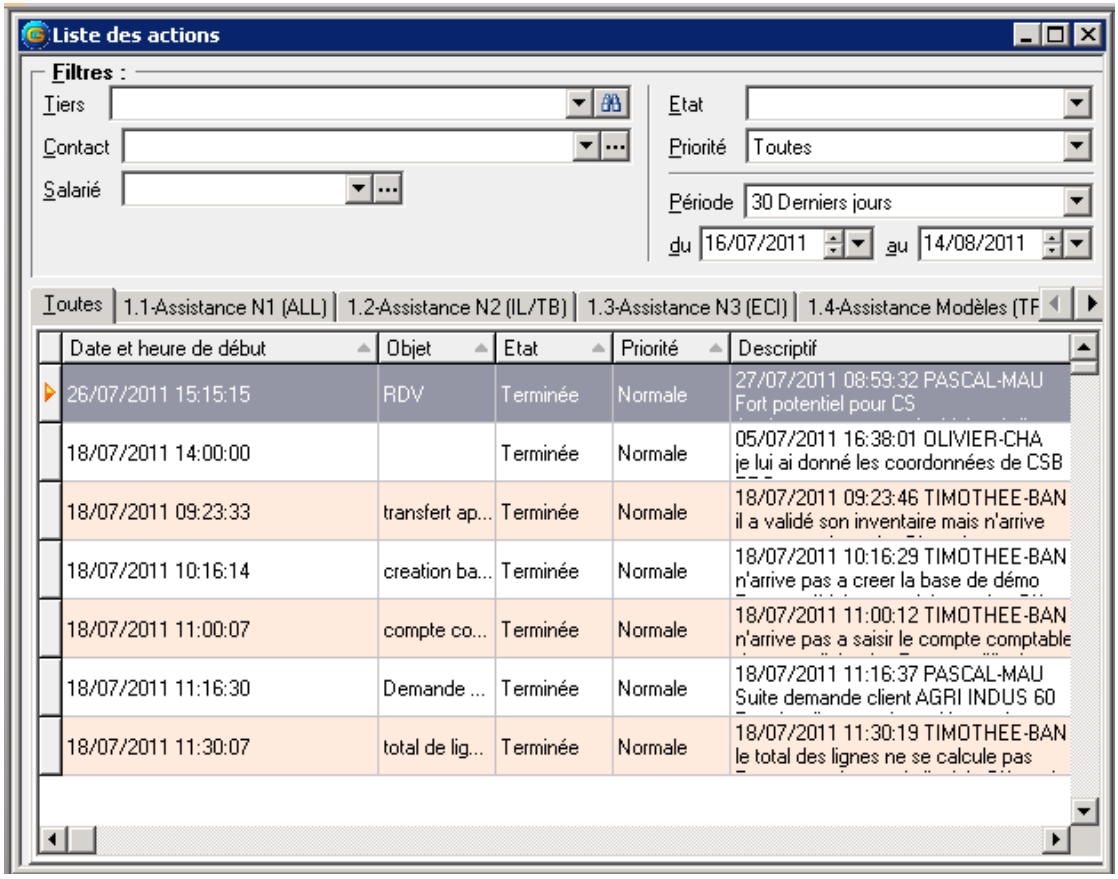


Figure 12. Saisie d'une action

Dans la Figure 12. Saisie d'une action", nous voyons le module actions de la solution GESTIMUM. Dans ce module il est possible d'inscrire des actions effectuées par un utilisateur à une date et un instant donné.

En parallèle, les demandes (d'évolution ou de maintenance) sont saisies dans l'outil Devteam qui rassemble les demandes d'améliorations et les dysfonctionnements constatés par les clients.

Ainsi les demandes des clients se trouvent à deux endroits (L'ERP GESTIMUM et l'outil Devteam) parfois en plusieurs exemplaires et avec des terminologies différentes.

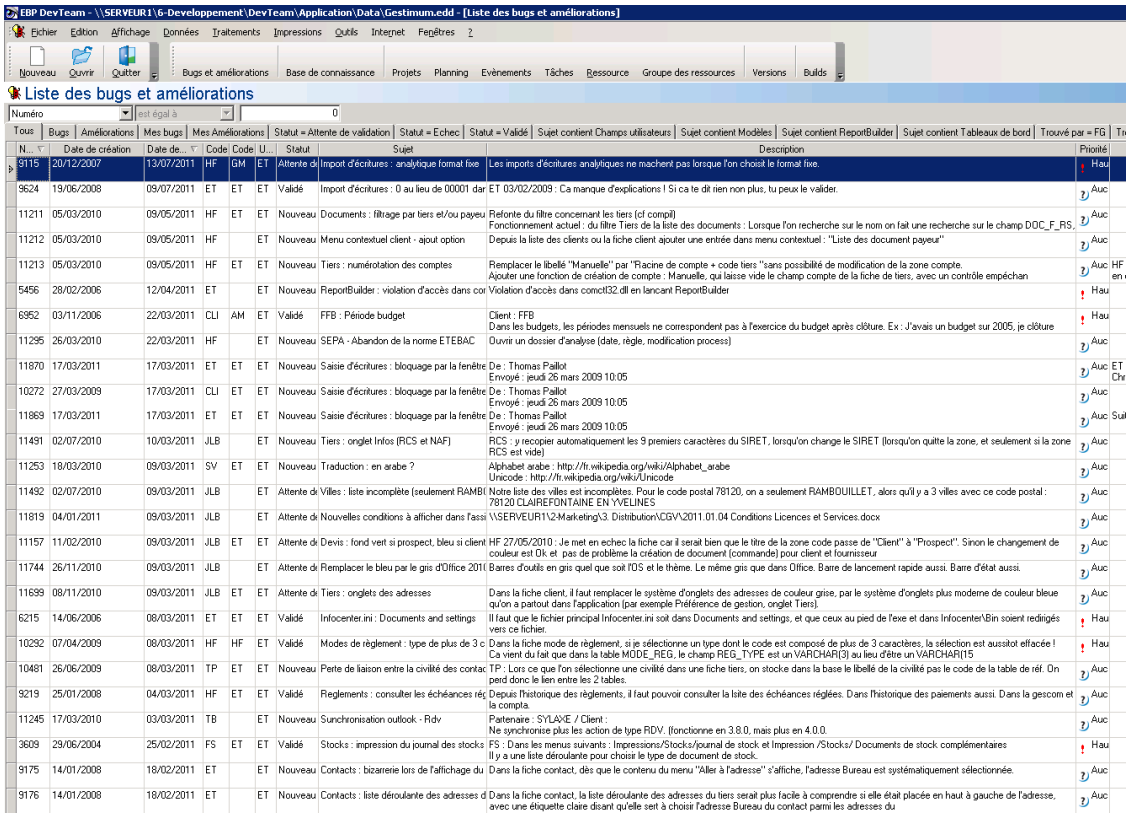


Figure 13. Liste des Bugs et Améliorations

La Figure 13. Liste des Bugs et Améliorations" nous montre une liste de bugs et améliorations saisies dans l'outil Devteam. Cet outil permettait de voir les anomalies remontées par les utilisateurs.

Un autre aspect de cette situation était que les développements (phase de codage) démarraient presqu'immédiatement après la décision de les faire, par un comité de pilotage restreint et conduit par une seule personne. Aucune phase d'analyse fonctionnelle et technique n'était faite. On pouvait se rendre compte par la suite que l'on avait passé trop de temps à développer une fonctionnalité qui ne servait qu'à un nombre limité d'utilisateurs.

Il n'y a pas dans ces processus, de phase de recette clairement établie, c'est-à-dire de phase où les développements sont testés. Il s'en suit des allers retours entre l'équipe de test et l'équipe de développement. Cela afin de corriger les développements. Il n'y a pas de phase de tests de non régressions, qui aident à accroitre la qualité de la solution produite.

3 Le nouveau système de développement et de maintenance.

Dans ce chapitre nous allons nous intéresser aux changements mis en place dans le système de développement et de maintenance Nous verrons également les contraintes d'un tel projet.

Les besoins de l'équipe de développement et d'assistance étaient de plusieurs natures, à la fois en structure, en méthodes de développement mais aussi en outils.

L'équipe d'assistance aux utilisateurs avait besoin de méthodes de recette ainsi que d'un moyen de communication efficace afin d'améliorer les interactions entre l'équipe de développement et l'équipe d'assistance.

L'équipe de développement avait besoin de méthodes de développement utilisant des standards éprouvés ainsi que d'outils efficaces pour communiquer avec l'équipe d'assistance.

- Nouvelle structure du système de l'équipe de développement

  L'équipe de développement a été divisée en deux parties qui traitent de la maintenance (correction de bugs) et de la modification du logiciel (nouvelles fonctionnalités).

  L'équipe service, dont fait partie l'équipe d'assistance, a été dotée d'un manager pour assurer la communication entre les deux départements.

  Le nouveau système de maintenance est basé sur un formalisme claire et simple, une liste des demandes à tester sans oublier le temps nécessaire pour effectuer ces tests..

- Nouvelle méthode de développement

Le nouveau système de développement est basé sur l'écriture de spécifications fonctionnelles des demandes d'évolution et maintenance et l'utilisation de diagrammes UML le cas écéant..

- Nouveaux Outils mis en place

  Ceux-ci permettent de répondre aux nouvelles exigences en termes de :

  1. Structure

     Mise en place d'un ALM afin de gérer le nouveau processus de développement.

     Mise en place d'un outil de tests automatisés afin d'effectuer des tests de non régression automatisés.

  2. Méthodes

     Documenter les demandes et gérer les spécifications liées aux demandes.

  3. Communication

     Assurer une communication efficace entre les équipes de développement et de test.

     Ouvrir le cycle de vie logiciel au monde extérieur et permettre une visibilité de l'état d'avancement des demandes.

Pour assurer une communication entre les deux systèmes, nous avons initié le projet de développer une passerelle bidirectionnelle entre l'ERP (Enterprise Ressource Planning) GESTIMUM et le système d'information du développement (Redmine).

3.1 Contraintes du projet

La mise en place des projets qui contribuent à la modification des processus de développement logiciel, font intervenir plusieurs équipes dans des domaines différents et des façons de travailler différentes. Il a fallu changer de méthodes. Par là des résistances aux changements sont apparues.

Ces résistances au changement ont plusieurs sources car les méthodes utilisées, avant modification, étaient considérées comme la meilleure façon de faire. De ce fait, la proposition de changement est considérée comme une non réponse aux besoins de développement.

Ces résistances aux changements constituent les contraintes de ces projets. Nous allons les énumérer et détailler leur contenu.

Les contraintes de cet ensemble de projets peuvent être classées en plusieurs catégories.

- Les contraintes de Coût

  L'ensemble des projets présentent un caractère stratégique pour l'entreprise car ils sont à la croisée des chemins entre le développement de la solution commercialisée et le développement de l'entreprise.

  Le coût des changements apportés devait se limiter au recrutement d'au plus 2 personnes, une dans chaque équipe (développement et assistance).

  Les outils acquis devaient également avoir un coût le plus faible possible.

- Les contraintes de Qualité

  L'objectif premier des projetsdu projet était d'augmenter la qualité des développements tout en diminuant les sollicitations à l'assistance.

En effet l'amélioration de la qualité dans tous les domaines du cycle de vie logiciel engendre des diminutions des demandes de maintenance et de ce fait une diminution des coûts liés à la maintenance.

L'amélioration de la qualité des développements engendrera mécaniquement une diminution des coût de maintenance, et c'est un des objectifs majeurs de ce projet.

- Les contraintes de Fonctionnalités

  La direction a émis le souhait de montrer aux clients des informations les concernant.

  Par exemple montrer à un client la feuille de route pour qu'il puisse voir que sa demande d'évolution ou de maintenance a bien été prise en compte et sera présente dans une future version.

  En effet, les outils utilisés auparavant ne permettaient pas d'ouvrir le système d'information à l'extérieur.

- Les contraintes Délais

  Ces transformations devaient être faites sans perturber les activités quotidiennes des équipes (maintenance ou évolution), et sans exéder une année pour la réalisation. Nouveaux processus

Dans le cas qui nous intéresse, la norme ISO 12207 a été appréhendée dans son ensemble car c'est la base des changements effectués, mais seule une partie des processus a été considérée et seuls ceuxdont la portée est en adéquation avec l'entreprise ont été utilisés.

La norme **ISO 12207** a pour objectif de poser la référence pour les processus du cycle de vie logiciel pris dans sa généralité avec :

- des processus de base :
    - Acquisition pour l'organisme lui-même,
    - Activités du fournisseur,
    - **Développement du logiciel**,
    - Exploitation du système informatique et des services associés,
    - Maintenance du logiciel.
- des processus supports :
    - **Documentation du logiciel et de son cycle de vie,**
    - Gestion de configuration,
    - Assurance qualité (PAQ) avec les revues, audit et vérification
    - Vérification,
    - **Validation,**
    - Revue conjointe,
    - Audit pour la vérification de la conformité aux exigences,
    - Résolution de problèmes et de non conformités en cours de développement et à l'exploitation.
- des processus organisationnels :
    - **Management de toutes les activités, y compris la gestion de projet,**
    - Infrastructure nécessaire à un processus,
    - **Pilotage et amélioration du cycle de vie**
    - **Formation du personnel.**

Pour nos besoins nous avons seulement considéré les processus écrits en caractères gras, à savoir :

3.1.1 Développement du logiciel

L'objectif de ce processus est de produire un logiciel exécutable qui reflète la conception de la solution.

- Les livrables de ce processus sont :

a. Critères de vérification ou contrôle définies pour toutes les parties intégrantes du logiciel par rapport à leur définition fonctionnelle.

   Il s'agit, ici, de fournir les spécifications fonctionnelles des demandes effectuées.

b. Production de modules logiciels définis par la conception.

   Il s'agit ici de définir les modules logiciels définis à la conception.

c. Traçabilité et consistance sont établies entre les modules logiciels et les spécifications fonctionnelles.

   Il s'agit ici de fournir un lien entre la conception et les modules créés.

d. Vérification des modules par rapport aux spécifications fonctionnelles.

   Il s'agit ici de fournir les moyens de contrôle des modules créés par rapport aux spécifications fonctionnelles.

   En résumé, comment tester ce qui a été développé.

3.1.2 Documentation du logiciel et de son cycle de vie

Le processus de management de la documentation consiste en la spécialisation du processus de management de l'information dans le groupe de processus de projets dans ce standard international.

- Les livrables de ce processus sont :

a. Définir la stratégie qui identifie la documentation à être produire pendant le cycle de vie du logiciel développé.

   Pour des raisons pratiques nous avons convenu que les demandes les plus complexes allaient être documentées de façon détaillée et le reste des demandes de façon plus succincte (dans l'outil de gestion du cycle de vie, Redmine).

b. Identifier les standards qui doivent être appliqués pour le développement de cette documentation.

c. Identifier la documentation à produire par le processus ou le projet.

   Dans le cas qui nous intéresse, la documentation produite est la spécification fonctionnelle.

d. Le contenu et l'objectif de toute la documentation est spécifié, revu et approuvé.

e. La documentation est développée et mise à disposition en accord avec les standards identifiés.

f. La documentation est maintenue en accord avec les critères définis.

3.1.3 Validation

Le processus de validation logiciel dans ce standard international est un processus de bas niveau du processus d'implémentation.

L'objectif de ce processus est de confirmer que le logiciel produit et/ou service d'un processus ou un projet reflète les spécifications prédéfinies.

- Les livrables de ce processus sont :

  a. Un document détaillant la a stratégie de vérification.

  b. Les critères de vérification de tous les logiciels requis sont identifiés

     Les critères utilisés dans ce cas sont vérifiés par les tests automatiques.

  c. Les activités de vérifications sont effectuées

  d. Les anomalies sont identifiés et enregistrées.

  e. Les résultats des activités de vérification sont mis à la disposition de toutes les personnes impliquées.

3.1.4 Management de toutes les activités, y compris la gestion de projets

L'objectif du processus de management des activités y compris la gestion de projets est de produire et communiquer des plans de projets utilisables et efficaces.

Ce processus détermine l'étendue du management de projets et des activités techniques. Il identifie les livrables de ce processus, les tâches. Iil établit les plannings de conduite des tâches, en incluant les critères d'achèvement et les ressources requises afin d'accomplir les tâches de ces projets.

- Les livrables de ce processus sont :

b. Définition de l'étendue du projet.

c. Evaluation de la faisabilité des objectifs du projet avec les ressources disponibles et les contraintes.

d. Evaluation de l'effort à fournir pour la réalisation des activités et estimation des ressources nécessaires afin d'accomplir le travail.

e. Identification des interfaces entre le projet et les autres unités organisationnelles

f. Développement du plan d'exécution du projet.

g. Activation des plans d'exécution du projet.

3.1.5 Pilotage et amélioration du cycle de vie

L'objectif du processus de pilotage et d'amélioration du cycle de vie est de définir et d'assurer la disponibilité des politiques et procédures à usage de l'organisation en respectant l'étendue de ce standard international.

Ce processus fournit les politiques du cycle de vie, les processus et les procédures compatibles avec les objectifs de l'organisation. Ceux qui sont définis, adaptés, améliorés et maintenus pour répondre aux besoins de chaque projet dans le cadre de l'organisation. Ils sont capables d'être appliqués à l'aide de méthodes efficaces et outils éprouvés.

- Les livrables de ce processus sont :

a. Fourniture des politiques et procédures pour le management et déploiement du modèle de cycle de vie

b. Définition des responsables, des responsabilités et devoirs de chacun au sein du cycle de vie.

c. Définition, maintenance, amélioration des modèles et procedures du cycle de vie à l'usage de l'organisation.

3.1.6 Formation du personnel.

L'objectif du processus formation du personnel est de maintenir les compétences des ressources existantes en accord avec les besoins de l'entreprise.

- Les livrables de ce processus sont :

a. Identification des besoins de formation en fonction des projets

b. Identification de compétences à accroitre au sein des ressources existantes.

La maintenance logicielle a également bénéficié de ces changements car des nouveaux processus de maintenance inspirés d'ISO 14764 ont été adoptés et ainsi un nouveau système de qualité logiciel a été mis en place.

Cette norme internationale distingue six processus de maintenance logicielle :

1. L'implémentation;
2. **L'analyse et la résolution de problèmes**

3. **La modification du logiciel**
4. L'acceptation de la modification par le demandeur;
5. La migration;
6. Et finalement, la mise à la retraite.

Pour nos besoins nous avons considéré seulement les processus écrits en caractères gras, en effet ces processus sont plus en adéquation avec la structure de l'entreprise :

3.1.7 Analyse et résolution des problèmes

Le processus d'analyse et résolution des problèmes, qui est exécuté une fois que l'application est sous la responsabilité de l'équipe de maintenance.

Le mainteneur doit :

1. Analyser la requête, la confirmer (en reproduisant la situation)
2. Contrôler la validité de chaque requête
3. Investiguer et proposer une solution
4. Documenter la requête et la solution proposée

- Les livrables de ce processus sont :

a. Analyse des requêtes et reproductions effectuées

b. Documenter la solution proposée

3.1.8 Modification du logiciel

La modification du logiciel est la modification du produit après l'avoir livré, afin de corriger des erreurs, accroitre ses performances ou autres caractéristiques.

- Les livrables de ce processus sont :

  a. Documenter les modifications demandées pour les corrections des erreurs.

     Ce qui revient à fournir les spécifications fonctionnelles des modifications effectuées.

b. Documenter les modifications demandées pour l'amélioration du logiciel ou son évolution.

Fournir les spécifications fonctionnelles des demandes d'évolution demandées.

Il y a un certain nombre de processus, d'activités et de règles qui sont propres à la maintenance du logiciel, par exemple :

1. **La transition : la passation coordonnée du logiciel, de l'équipe de développement à l'équipe de maintenance.**

   Dans le cas qui nous intéresse il s'agit de montrer de façon claire et précise qu'une fonctionnalité est défaillante et qu'elle doit être corrigée.

   L'équipe de développement a été divisée en deux afin de séparer les activités de maintenance et de développement de nouvelles fonctionnalités.

   Du fait de cette organisation, les activités de maintenance ont été prises en charge à tour de rôle par l'un ou l'autre des développeurs de l'équipe en fonction de leur disponibilité.

2. Les ententes de services applicatifs.

3. La priorisation des requêtes de modification et des rapports de problèmes.

4. **Acceptation ou rejet d'une demande de modification selon le travail et la complexité plus ou moins grandes qu'elle implique; son renvoi à l'équipe de développement.**

   Le processus de gestion de vie logiciel que nous avons mis en place permet, grâce à un workflow (*voir chapitre 4*), après analyse de la demande.,Soit de rejeter soit de la confirmer et par là de l'inclure dans le plan de maintenance.

3.2 Le nouveau système de développement

Pour améliorer la qualité des développements, il était impératif d'adopter des méthodes fiables et d'utiliser des outils adaptés, qui allaient nous aider à accroitre la fluidité du passage de l'information entre les équipes (équipe de développement et test) et par voie de conséquence, la qualité du produit.

Les développements n'étaient pas documentés, il n'y avait pas de documents types, pas d'expression de besoins, pas de spécifications validées par le client final, qu'il soit interne ou externe.

L'exemple le plus frappant est la documentation de la solution GESTIMUM elle-même. C'est un Progiciel de Gestion intégré, qui est au centre de l'entreprise cible (PME moyenne de 10 à 250 salariés. Il plusieurs modules :

1. Un module de comptabilité
2. Un module de gestion commerciale qui gère les documents de vente (devis, bons de commande, factures, avoirs, factures financières, avoirs financiers)
3. Un module de gestion des tiers (Prospects, Clients et Fournisseurs)
4. Un module de gestion de stocks
5. Un module de gestion des budgets
6. Un module de gestions des affaires et planning des ressources
7. Un module de gestion des actions.
8. Un module de transfert électronique de documents. Plus communément appelé EDI (Echange de Données Informatisé)
9. Un module décisionnel, permettant la création de tableau de bords personnalisables et automatisables à volonté.

Tous ces modules ne sont pas apparus dès la première version mais le départ de cette solution est basé sur un seul document de quelques pages qui liste les souhaits du contenu de la solution, sans donner les indications nécessaires à un développement cohérent.

Une conséquence de cet état de fait, est la rédaction de spécifications fonctionnelles pour toute demande effectuée (maintenance ou évolution) à l'équipe de développement.

Tous ces éléments contribuent à augmenter la maintenabilité de l'application, car un développeur qui reprend le travail d'un autre peut se référer à cette documentation et aura une aide schématique qui le guidera dans ses développements.

En termes d'organisation, il a fallu également changer les méthodes de travail car les phases de livraison n'étaient pas définies, et le code restait modifiable au moment de la livraison de la version pour tests.

En effet, lors de la fourniture d'une version pour la phase de tests, le code de l'application n'était pas figé (ou sauvegardé) et les développements continuaient sur la même version.

Ainsi l'on pouvait continuer à développer et à fournir aux clients une version modifiée qui apportait un correctif sur un point particulier mais qui pouvait engendrer une régression non détectable car la phase de tests avait déjà commencé et les testeurs n'allaient pas tester à nouveau les points qu'ils avaient validés au préalable.

Une nouvelle méthode a été mise en place consistant à figer le code pendant toute la phase de recette. Cela garantit que la version livrée a bien été testée avec un environnement cohérent.

Figer le code signifie en fait que les développements de nouvelles fonctionnalités s'arrêtaient pendant la phase de recette pour ne travailler que sur les demandes de maintenance.

3.3 Formation de l'équipe

L'équipe avait besoin de formation en méthode de modélisation afin de concevoir les développements.

C'est pourquoi j'ai contacté un organisme de formation; et jai organisé des séances de formations dédiés à UML.

Nous avons abordé tous les éléments concernant UML (voir annexe I.11 pour le détail de la formation) et les avons appliqués à des cas concerts.

3.4 Le système qualité de la maintenance

La maintenance a également été revue car les demandes étaient trop souvent formulées de façon peu explicite et sujette à interprétation.

Afin d'accroitre la qualité de la maintenance, j'ai mis en place un formalisme minimum de demandes de maintenance.

Ce formalisme consiste tout simplement en la création d'un document contenant les éléments les plus importants pour faciliter l'analyse du problème.

Le document de demande de maintenance doit contenir les rubriques suivantes :

- <u>Description de l'anomalie</u>
    - Décrire les actions que l'utilisateur a effectué lors de la l'apparition de l'anomalie.

      Il est demandé ici de détailler toutes les actions que l'utilisateur a effectuées et qui ont abouti à l'anomalie décrite
    - Résultats escomptés et résultats obtenus.

L'utilisateur s'attend à un certain résultat, par exemple si l'utilisateur est en train de créer une facture d'un produit qui vaut 100 et une quantité de 1, le total devrait donc être de 100.

Or si le résultat est différent, cela veut donc dire que les résultats escomptés et les résultats obtenus n'ont pas coïncidé et que par conséquent il doit y avoir un problème, soit dans le paramétrage soit dans la solution GESTIMUM elle-même.

- Contexte de l'anomalie
    - Situer l'anomalie dans son environnement, quelle version du produit est utilisée.
    - Quels sont les éléments extérieurs à l'application (Système d'exploitation, applications installées sur la machine cliente.
    - Quelles sont les éléments serveur (Systèmes d'exploitation serveur utilisé, Version du serveur de base de données)
- Impact
    - Donnez une estimation de l'impact que cette anomalie a par rapport à l'entreprise et aux autres utilisateurs.
    - Il y a-t-il un ou plusieurs clients impactés par cette anomalie.
- Criticité
    - Est-ce que l'anomalie présente un caractère impératif de résolution car les résultats obtenus ne sont pas conformes (un mauvais calcul par exemple).
- Conditions de reproductibilité
    - Actions à effectuer pour reproduire l'anomalie
- Données techniques

- Base de données du client (son nom et son emplacement sur le serveur).

Ces informations permettent de délimiter le périmètre de l'anomalie et de mieux cerner le problème et de le traiter plus rapidement.

Un des changements les plus importants a été la création de phases de maintenance dans la feuille de route (Roadmap) du produit.

Nous avons créé des phases de stabilisation du produit. Pendant cette phase nous avons privilégié la correction d'anomalies par ordre d'importance (criticité) et d'impact.

Ces phases de stabilisation du produit nous ont permis de mieux nous concentrer sur les attentes des clients en termes de fiabilité de la solution.

Indirectement ces phases de stabilisation ont conduit à une diminution des sollicitations de l'assistance car plus le produit est fiable et moins les utilisateurs ont besoin d'assistance.

Un autre aspect du changement (structurel) demandé a été le recrutement d'un manager pour le département services.

Ce manager a pris en charge la phase de recette dans son ensemble et a aménagé le temps nécessaire pour que les tests soient effectués.

La Figure 14. Phase de Stabilisation et d'ajout de Fonctionnalités", nous montre les phases de stabilisation que nous avons mises en place, à savoir en mai 2011 qui a consisté en la correction de plusieurs dizaines d'anomalies jugées majeures.

Nous avons commencé par corriger un grand nombre d'anomalies afin de stabiliser le logiciel. C'est seulement après cette phase que nous avons continué d'ajouter des nouvelles fonctionnalités.

C'est un travail d'équipe (services et développement) que nous avons mis en place, avec un système de communication efficace et des réunions ponctuelles sur des points précis qui nous aident à mieux travailler ensemble.

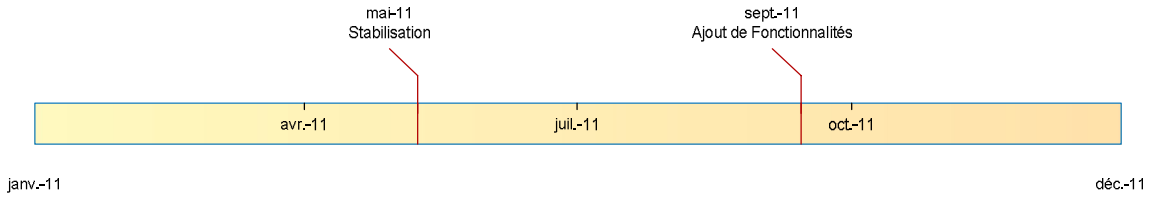


Figure 14. Phase de Stabilisation et d'ajout de Fonctionnalités

La phase d'ajout de fonctionnalité a débutée en Septembre 2011 après la phase de stabilisation, nous avons alors ajouté des fonctionnalités importantes pour les clients.

3.4.1 La non régression

La non-régression a été traitée comme un élément primordial de l'amélioration de la qualité logiciel, car elle permet de s'assurer que les modifications n'affectent pas l'existant et constitue ainsi une barrière contre les anomalies qui peuvent être engendrées involontairement par un développement.

Nous avons donc créé un projet dédié à la non-régression, qui a consisté en la création de scenarii de tests avec des cas de données prédéfinis. Nous avons choisi une solution d'automatisation des tests

3.5 Outils mis en place

Le problème principal de la structure mise en place, était non seulement l'absence de méthode, mais aussi le manque d'un outil capable de centraliser toutes les demandes (maintenance ou évolution), de les canaliser, de les répertorier, de les prioriser, de les faire vivre et les gérer dans son ensemble.

Nous avons donc créé un projet de recherche d'un ALM (Application Lifecycle Management).

Pour ce faire nous avons profité de nos réunions hebdomadaires pour confronter le point de vue de chacun des collaborateurs. Ils ont pu donner leur avis sur les fonctionnalités qui paraissaient indispensables ou souhaitées.

Le choix de l'ALM (Application Lifecycle Management) s'est porté sur une solution open source nommée Redmine. A nos yeux cet outil est de loin le meilleur, car a été pensé spécifiquement pour la gestion du cycle de vie logiciel.

La solution Redmine possède les modules suivants :

1. Gestion multi-projets.

   Les projets de développement peuvent être privés ou publics notamment. Le statut d'un projet reste très simple: soit actif, soit archivé.
   Il est possible de créer des sous-projets de n'importe quel projet (donc des sous-sous projets). Il est possible de copier un projet et donc de faire des modèles de projets.

2. Gestion des demandes.

   Le manager d'un projet choisit les types de demandes (dans le vocabulaire Redmine cela s'appelle un tracker) qui seront disponibles dans le projet. Les demandes pourront être assignées à un membre du projet. On peut assigner des demandes à un groupe de personnes.

   Il est possible d'associer des "Watchers" aux tâches, c'est à dire des personnes qui seront nécessairement averties par e-mail du déroulement de la tâche. On peut créer des tâches privées au sein d'un projet; ces tâches ne sont alors pas visibles de tous les membres du projet.

   Les demandes peuvent être déplacées dans un autre projet.

Redmine

Aperçu | Download | Activité | Roadmap | Demandes | Annonces | Wiki | Forums | Dépôt

Demandes

Filtres

Statut ouvert

Ajouter le filtre

Options

Appliquer Effacer

#	Tracker	Statut	Sujet	Mis-à-jour	Catégorie
10351	Feature	New	preview of a duplicate only shows the description	2012-03-01 16:45	Issues
10348	Defect	New	Wrong issue statuses assignment during issue copy	2012-03-01 15:46	Issues
10345	Feature	New	Filtering user names during issue creation	2012-02-29 15:24	Issues permissions
10341	Feature	New	New feature	2012-02-29 08:02	
10336	Defect	New	SQL error on "Group By clause" with Postgresql and JRuby	2012-03-01 15:15	Ruby interpreter support
10331	Feature	New	Per-project 'Issue Closed' status customization	2012-03-01 17:16	Issues workflow
10329	Defect	New	filter box UI can overlap into custom query UI at large font sizes	2012-02-27 17:30	UI
10328	Defect	New	Issue custom field is too narrow in IE 8.0.6001.18702 but OK in Firefox 9.0.1	2012-02-27 16:30	
10320	Defect	New	db migrate "ChangeAttachmentsContainerDefaults" error on SQLite3 1.3.5	2012-02-25 12:52	
10315	Feature	New	Auto addition to the group after login	2012-02-24 10:12	Accounts / authentication
10314	Feature	New	Make the only enabled activity in a project the default one for time entry	2012-02-24 09:53	Issues
10313	Feature	New	Historic	2012-02-24 09:17	Issues
10312	Defect	New	Can not attach empty files.	2012-02-24 08:01	Attachments
10310	Defect	New	Member search should distinguish people in case of duplicate names	2012-02-23 23:31	UI
10307	Feature	New	Enable tracking of external bugs	2012-02-23 14:14	
10303	Defect	New	Demo and test sites do not support basic authentication with REST API	2012-02-23 12:51	Website (redmine.org)
10294	Feature	New	Redmine search: improve, make better and smarter. (title matching first, then content)	2012-02-24 14:37	Search engine
10293	Feature	New	improve readability of subject changed message	2012-02-22 20:40	Email notifications
10292	Defect	New	replying to messages in forum is always considered spam	2012-02-22 20:41	Website (redmine.org)
10286	Feature	New	Auto-populate fields while creating a new user with LDAP	2012-02-21 00:07	Accounts / authentication
10284	Defect	Resolved	Note added by commit from a subproject does not contain project identifier	2012-02-20 22:59	SCM
10282	Feature	New	Copy wiki attachments on project copy	2012-02-20 19:34	Projects
10277	Defect	Confirmed	Redmine wikitext URL-into-link creation with hyphen is wrong	2012-02-29 03:09	Text formatting
10276	Feature	New	When assigning an issue, add possibility to pre-select a group	2012-02-19 20:20	Issues
10273	Feature	New	Offer ability to see issues due for a user across projects on their user page	2012-02-18 15:25	Issues

Figure 15. Redmine, aperçu module gestion des demandes

3. Gestion fine des droits utilisateurs définis par des rôles.

   Un utilisateur peut appartenir à un ou plusieurs projets. A chaque utilisateur sera associé un - ou plusieurs - rôle(s) par projet, les rôles définissant les permissions accordées aux utilisateurs dans un projet. Ainsi un utilisateur peut avoir des rôles différents selon le projet. Les rôles sont communs à tous les projets et sont très modulables.

   Il est possible de créer des groupes d'utilisateurs et donner les mêmes permissions à tous les utilisateurs dans un projet.

   Un utilisateur peut créer un projet sans avoir les droits d'administrateur.

   Nous avons créé des rôles en fonction des activités de chacun, par exemple le rôle développeur peut prendre en charge une demande et changer son statut mais seulement dans la limite prédéfinie pour que

l’équipe de test puisse intervenir, ensuite l’équipe de test peut prendre le relai et changer l’état de la demande en fonction des résultats des tests.

4. Gestion des documents

   Les documents sont associés à un projet, ils sont classés classés en catégories que l'administrateur de Redmine définit. Un système de filtre très bien fait permet de classer les documents par propriétaire, titre, date ou catégorie.

5. Les priorités paramétrables d’une demande

   L'administrateur définit les priorités possibles des demandes (urgent, normal...).

6. Gestion de groupes d'utilisateurs.

   Il est possible de regrouper des utilisateurs afin de leur attribuer différents paramètres, tels que des droits ou projets.

7. La gestion du temps

   Début, fin estimée, pourcentage de réalisation et possibilité d'ajouter des champs personnalisés

8. L'ajout de champs personnalisés

   Redmine permet de personnaliser les informations que l'on veut avoir pour les projets, utilisateurs, demandes, temps passé, utilisateurs, groupes.

   Les utilisateurs peuvent modifier eux-mêmes les champs personnalisés associés à leur compte

9. La modulation fine des statuts et la gestion des transitions de statuts par rôle.

Chaque demande peut évoluer avec un statut différent (nouveau, en cours, fermé..). Pour chaque rôle, on peut définir pour les différents types de demande, les transitions des statuts, c'est à dire les états possibles du statut futur en fonction du statut actuel.

10. Rapports d'anomalies, demandes d'évolutions.

    Un rapport envoyé de façon automatique sur les anomalies et demandes d'évolution peut parvenir à la personne assignée par le manager et en fonction de critères tels que :

    - Nombre de bugs non corrigés depuis plus de 7 jours.
    - Nombre de demandes d'évolution non modifiées depuis plus de 7 jours.
    - Attribution d'une anomalie à un développeur.

11. Wiki multi-projets.

    Un wiki est un site web dont les pages sont modifiables par les visiteurs afin de permettre l'écriture et l'illustration collaboratives des documents numériques qu'il contient.

    Il est donc possible pour les utilisateurs de voir les pages dédiées à un projet et de les modifier si besoin, ce qui peut être dangereux si les modifications ne sont pas contrôlées.

12. Forums multi-projets.

    Un forum est un espace virtuel qui permet de discuter librement sur plusieurs sujets divers.

    Dans ce cas il est possible d'ouvrir des sujets de discussions sur un thème en particulier, comme par exemple une fonctionnalité à rajouter ou une modification demandée.

13. News accessibles par RSS / ATOM.

    Pour suivre l'évolution d'une demande ou d'un projet (le flux RSS suit la norme ATOM).

14. Notifications par courriel (mail).

    Redmine est capable de communiquer avec le monde extérieur par le moyen de courriers électroniques, ces notifications peuvent déclenchées par modification de tout élément d'un projet.

15. Gestion de feuilles de route, GANTT, calendrier.

    Redmine est capable de construire des diagrammes de GANTT d'un projet, montrer le calendrier des demandes et fournir également une feuille de route qui est fonction des demandes d'évolution et de maintenance affectées à une version d'un projet de développement.

    La Figure 16. Redmine. feuille de route (roadmap) ", montre un exemple de feuille de route Redmine. Elle indique que la version 0.8 est prête à 99% car 259 demandes sont closes et 2 toujours ouvertes. Elle également les principales fonctionnalités présentes dans cette version.

Figure 16. Redmine. feuille de route (roadmap)

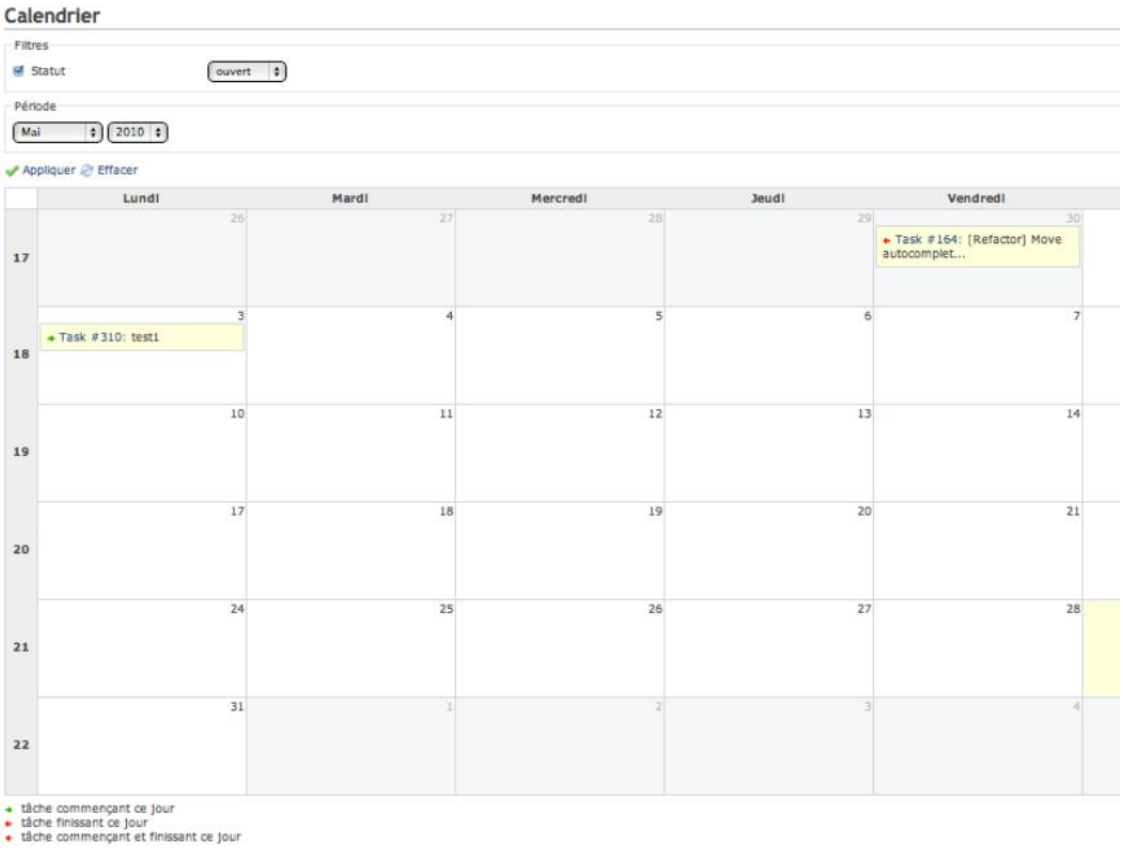


Figure 17. Redmine, Calendrier des demandes

La Figure 17. Redmine, Calendrier des demandes", montre le calendrier, dans lequel nous pouvons voir les demandes effectuées.

16. Historique.

Pour les demandes et le wiki on peut suivre toutes les mises à jour.Qui les a faites et à quelle date.

17. Intégration avec divers systèmes de suivi de versions : SVN, CVS, Mercurial, Git, Bazaar & Darcs.

Un logiciel de gestion de versions (ou VCS en anglais, pour Version Control System) est un logiciel qui permet de stocker un ensemble de fichiers sources en conservant la chronologie de toutes les modifications qui ont été effectuées. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes.

Les logiciels de gestion de versions sont utilisés notamment en ingénierie du logiciel pour conserver le code source relatif aux différentes versions d'un logiciel.

18. Identification possible via LDAP (ainsi qu'Active Directory) et CAS (à travers un plugin pour ce dernier).

    J'ai mis en place une connexion s'appuyant sur l'annuaire de l'entreprise (Active Directiory). De cette manière les utilisateurs peuvent s'authentifier avec les mêmes informations qu'ils utilisent pour ouvrir leurs sessions sur leurs postes de travail.

19. Multilingue (25 langues).

20. Support de plusieurs bases de données : MySQL, PostgreSQL ou SQLite.

Figure 18. Interface Redmine

Une fois la solution mise en place il manquait un lien avec l'ERP par le biais des actions.

Voici un descriptif de ce projet qui nous aurait permis de remonter des demandes clients vers l'outil de gestion de cycle de vie logiciel (Redmine) et inversement pouvoir remonter l'avancée d'une demande vers l'ERP.

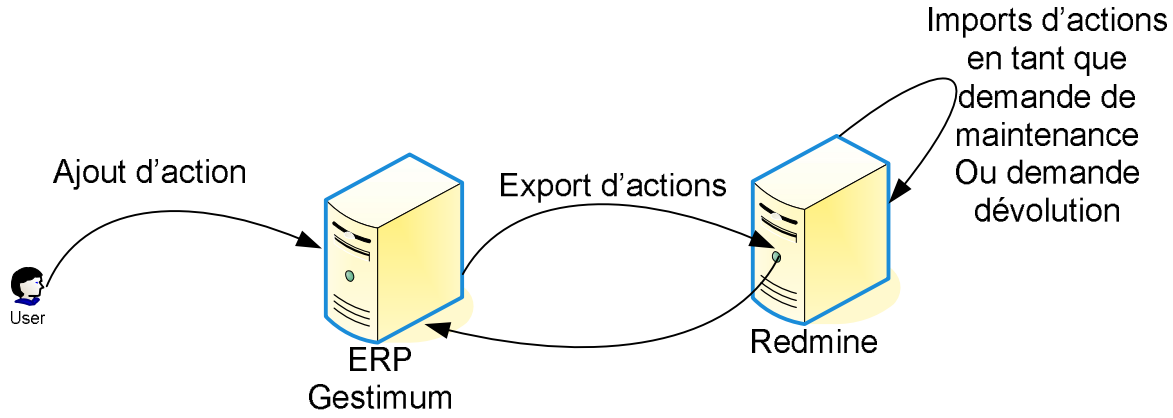


Figure 19. Passerelle ERP - Redmine

3.6 Présentation de l'ensemble des projets, gestion et conduite de changement.

Les changements apportés à la structure de l'équipe, les méthodes de travails, les outils utilisés, ont nécessité la mise en place d'un ensemble de projets.

L'ensemble des fonctions souhaitées ont été confrontées à un panel de solutions choisies, installées et testées.

Dans ce chapitre nous allons décrire l'ensemble des projets, la façon dont nous les avons menées, les changements opérés et la façon dont les changements ont été implémentés.

3.6.1 Nouveau processus de développements

Le projet nouveaux processus de développement a été effectué en analysant les processus utilisés puis en les critiquant pour en proposer une modification qui s'appuie sur un standard en la matière, à savoir ISO 12207.

Le résultat de ce projet a été l'adoption d'un nouveau modèle de processus de développement, dont les étapes sont formalisées dans un document qui montre le flux d'information avec les acteurs concernés, les livrables et les interactions entre les différents membres des équipes.

Ce modèle de développement a été formalisé sous la forme d'un schéma représentant le workflow (voir Figure 20. Processus de Développement d'une demande") qui a été conçu, expliqué à l'ensemble des acteurs concernés.

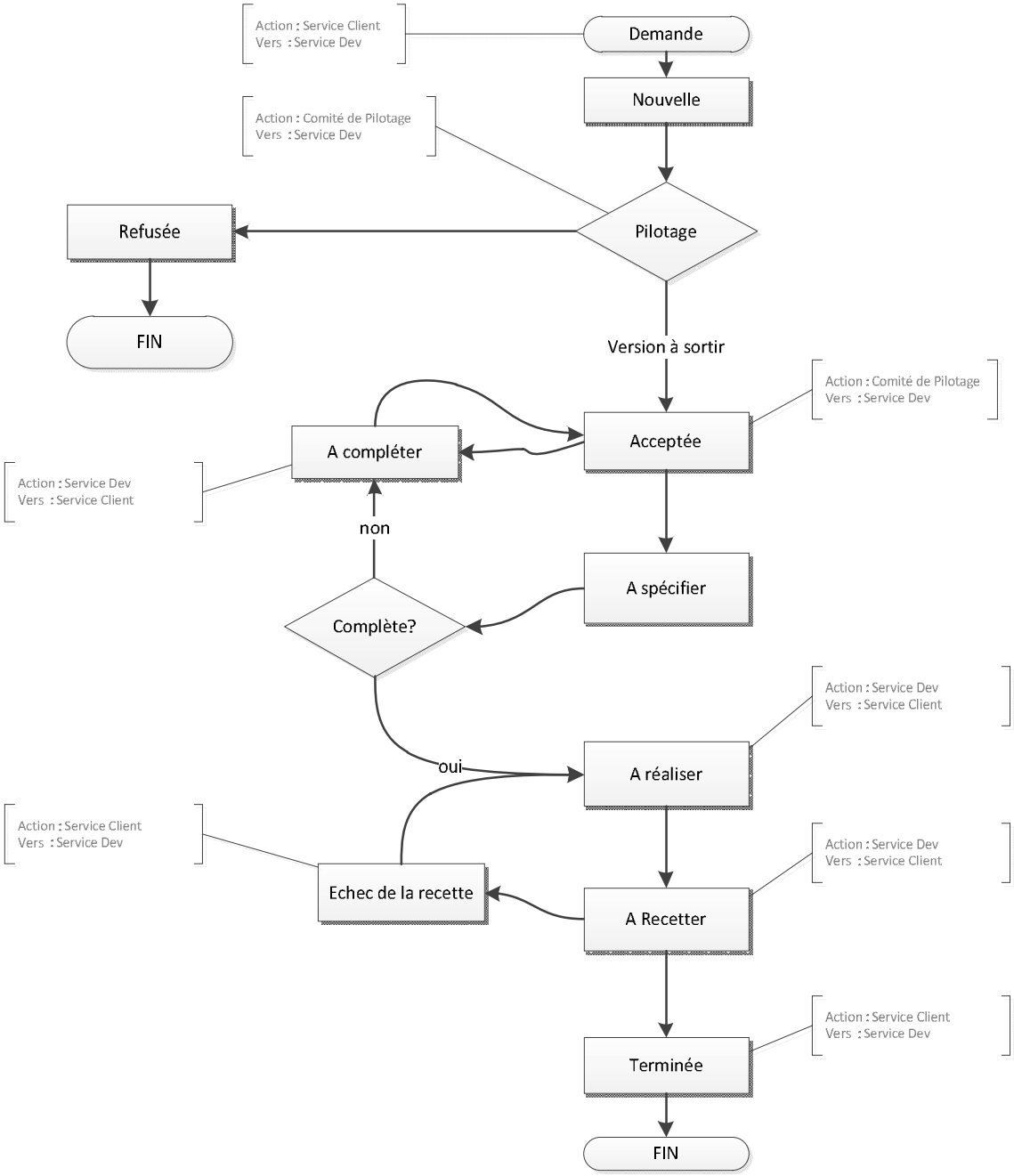


Figure 20. Processus de Développement d'une demande

La Figure 20. Processus de Développement d'une demande, montre le nouveau processus de développement dans lequel toutes les phases de développement sont représentées.

Ces phases ont été implémentées (*voir chapitre 4.2*) dans un workflow dans l'ALM afin de compartimenter les différents rôles de chaque membre des équipes concernées.

Il a été ainsi possible d'identifier de façon certaine qui est en train de travailler sur une demande, dans le cas où il faut poser des questions sur l'implémentation de telle ou telle fonction ou modification et à quelle phase la demande en est-elle.

3.6.2 Nouveau processus de qualité logiciel

La qualité logicielle a été améliorée par l'adoption de processus issus d'ISO 14764.

Dans le workflow mis en place grâce à l'outil Redmine (*voir chapitre 3.7.1*) Choix d'un ALM) il est possible de valider les demandes clients après analyse et de changer le statut de cette dernière.

De plus, la phase de recette a été constituée formellement, à savoir que celle-cia été identifiée dans le cycle de vie des développements et dans le planning des développements. Les plages de temps nécessaires aux tests ont été réservées à l'équipe par leur manager.

Les tâches à effectuer lors de la phase de recette ont été formalisées et une liste exhaustive des tests a été créée.

Les interactions avec les autres membres de l'équipe ont été formalisées par des points d'avancements réguliers et l'équipe de développement a été placée en mode debug, à savoir que les demandes de l'équipe de tests étaient traitées immédiatement.

Toutes ces mesures font partie de la norme ISO 14764.

3.7 Processus d'acquisition

Une étude a été menée sur l'opportunité de la mise en place d'un ALM avec la définition des exigences dans le contexte de l'entreprise.

La norme ISO/CEI 12207 s'applique à des acquisitions de logiciels, de prestations logicielles ou du développement de logiciels spécifiques.

Nous nous sommes inspirés de la norme ISO 12207 (voir chapitre 2.2.2 ISO 12207) car les processus d'acquisition de logiciels sont une référence importante et donnent un cadre de référence pour la mise en place d'une telle solution.

Pour ce faire nous avons créé une expression de besoin qui rassemble les objectifs, les critères d'adoption de la solution et la procédure de sélection du produit.

Une recherche a été menée afin de sélectionner les solutions existantes qui pouvaient correspondre aux critères adaptés à l'entreprise (par exemple, un coût d'acquisition faible).

L'acquisition du produit a suivi un processus de sélection qui respectait une matrice de décision laquelle nous a permis de choisir la solution selon des critères objectifs et pondérés selon les besoins de l'entreprise.

Les critères (voir chapitre 3.7.1 Choix d'un ALM) de sélection tenaient compte des contraintes de l'entreprise, de l'existant et des objectifs futures de la solution choisies.

3.7.1 Choix d'un ALM

ID	Nom de tâche	Début	Terminer	Durée
1	Définition des critères de choix	10/01/2011	24/01/2011	2,2s
2	Pondération des critères	24/01/2011	07/02/2011	2,2s
3	Sélection des solutions	07/02/2011	21/02/2011	2,2s
4	Acquisition des solutions	21/02/2011	28/03/2011	5,2s
5	Tests des solutions acquises	28/03/2011	30/05/2011	9,2s
6	Sélection de la solution	30/05/2011	13/06/2011	2,2s

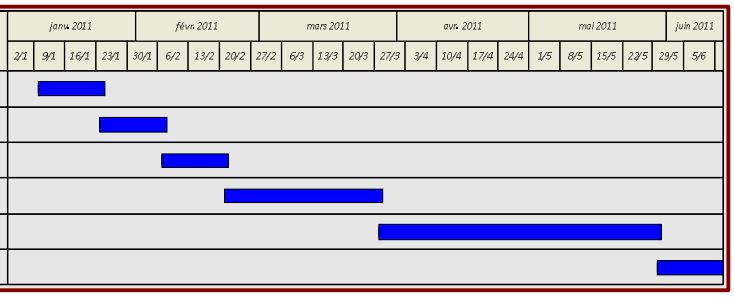


Figure 21. Choix d'un ALM

La Figure 21. Choix d'un ALM", montre les étapes que nous avons suivies pour la sélection de l'outil qui allait devenir notre système d'information qui gère le cycle de vie logiciel.

Nous avons écrit des spécifications fonctionnelles (ensemble de fonctions souhaitées dans la solution) afin de gérer le projet de façon précise et objective, ainsi nous nous sommes appuyés sur un document structuré, ces spécifications ont été revues, adoptées et comprises par tous.

Voici la liste des fonctionnalités attendues que nous avons sélectionnées pour faire notre choix.

1. Gestion du cycle de vie logiciel.

1.1 Gestion des projets.

- Pourcentage d'avancement.

  Le pourcentage d'avancement d'un projet est calculé pour renseigner sur sa progression projet vers son objectif.

1.2 Gestion des tâches

- Ordonnancement manuel des tâches.

  Cette fonction permet l'élaboration d'un plan d'action permettant de déterminer les séquencements des tâches identifiées précédemment.

- Pourcentage d'avancement d'une tâche

  Cette fonction permet de voir le pourcentage de progression des tâches dans le projet.

- Temps prévu

Cette fonction indique le temps donné et prévu pour effectuer les tâches dans un projet pour sa progression.

- Temps passé

Cette fonction indique le temps passé et pris pour effectuer les tâches dans un projet lors de sa progression.

- Intervenants (ressource)

Cette fonction permet de voir qui a ou est intervenu sur la demande ou la tâche.

- Statut d'avancement (modifiable)

Cette fonction permet de donner un état sur les tâches à fournir (complète, incomplète,…).

1.3 Planification des tâches

Cette fonction sert à l'organisation des tâches (date-durée) de manière précises pour les tâches à fournir.

1.4 Gestion des ressources humaines

Cette fonction sert à gérer les ressources humaines disponibles pour le projet.

1.5 Regroupement des tâches par sujet.

- Nombre de niveaux des sous-tâches > 10

1.6 Possibilité d'ouvrir ou fermer une arborescence de tâches.

Cette fonction permet de voir un ensemble de tâches reliées entre-elles par des liens de dépendance.

1.7 Pouvoir déplacer une ou plusieurs tâches dans un projet.

1.8 Génération de diagrammes de Gantt

Le diagramme de GANTT permet la planification des tâches nécessaires à la réalisation du projet.

1.9 Génération / Impressions d'états.

1.10 Copier / Coller d'images dans les tâches et projets.

Cette fonction sert à donner la possibilité d'insérer des copies d'écrans afin d'améliorer la compréhension des problèmes posés.

1.11 Ajout de fichiers joints à une tâche ou un projet.

Cette fonction donne la possibilité par exemple de joindre tout document nécessaire au traitement de la tâche à effectuer (spécifications fonctionnelles ou demande de maintenance).

1.12 Tableaux de bord personnalisables

- Vue par ressource
- Vue globale du projet avec les tâches, bugs / améliorations correspondantes et leur lien (tache – bug)

1.13 Listes filtrables

Permet de sélectionner un sous-ensemble d'éléments d'une liste en fonction d'un critère inscrit dans le filtre.

1.14 Champs utilisateurs / personnalisables.

1.15 Tag possibles (tâches, bugs, etc)

2. Gestion de la remontée de Bug / Demandes d'améliorations.

2.1 Copier / Coller d'images et fichiers.

Attribuer un bug à un projet ou à une tâche.

2.2 Attribuer une criticité

Cette fonction sert à déterminer la nature du bug c'est-à-dire bloquant ou pas bloquant.

2.3 Attribuer un impact

Cette fonction détermine l'étendue de l'anomalie.

2.4 Gestion des Statuts

2.5 Sources du bug

Cette fonction permet de montrer la provenance (de quellle partie du logiciel)du bug et l'écrire dans un commentaire

2.6 E-mail sur statut.

Cette fonction permet l'envoi d'e-mails en cas de changements dans le projet.

2.7 Rendre visible le statut d'avancement.

3. Prérequis Techniques

3.1 Base de données

La base de données doit être de type SQL

3.2 Interface

L'interface doit être de type Web et donc pas de client lourd, pas d'application à installer sur les postes clients (le navigateur Web est déjà installé).

3.3 Open Source

Langage utilisé doit être dans un langage supportable (PHP 5, mini et objets, Java, Ruby).

Utilisation de Frameworks (outils utiles pour fonctionnement).

Nous avons utilisé une grille décision pour faire notre choix.

Nous avons appliqué des coefficients selon l'importance relative du critère.

Fonctionnalités attendues	Coeff.	Devteam	Endeavour	Web2projet	REDMINE	PROJECT-OPEN	Topcased 4	IBM RTC
I Pré-requis techniques								
Base de données [10]	3	1	1	1	1	1		1
Base SQL (SQL Server[0] > MySQL[0] > reste)	1	1	1	1	1	1		
Interface								
Web. PAS de client lourd	3	0	1	1	1	1		1
Open source	3		1	1	1	1	1	1
Le langage utilisé doit être supportable (PHP 5 utilisant l'objet, java…)	3	1	1	1	1	1	1	1
Frameworks utilisés	1	0	1	1	1		1	1
II Gestion du cycle de vie logiciel								
Gestion des projets	3	1	1	1	1	1		
% d'avancement	3	1	1	1	1	1		
Gestion des tâches	3	1	1	1	1	1		
Ordonnancement des tâches	3	1	1	1	1	1		
% d'avancement	3	1	1	1	1	1		
statut d'avancement	3	1	1	1	1	1		
intervenants	3	1	1	1	1	1		
Tesmps prévu	3	1	1	1	1	1		
Temps passé	3	1	1	1	1	1		
Planification des tâches	3	1	1	1	1	1		
Gestion des ressources (humaines)	2	1	1	1	1	1		
Regroupement des tâches par sujet	2	1	1	1	1	1		
Niveaux de sous-tâches >= 10	2	1	1	1	1	1		
Possibilité d'ouvrir ou fermer une arborescence	2	1	0	1	0	1		

de tâches								
Pouvoir déplacer une ou plusieurs tâches dans un projet	2		0	1	1			
Génération de diagrammes de Gant	3		1	1	1	1		
Génération d'impression d'états	2	1	1	1	1	1		
Copier/coller d'images dans les tâches et projets	3		0	0	0	0		
Ajout de fichiers joints à une tâche ou un projet	3	1	1	1	1	1		
Tableaux de bord personnalisables	2	1	1	1	1	1		
Vue par ressource	2	1	0	1	1	1		
Vue globale par projet avec les tâches, bugs/améliorations correspondantes et leur lien (tâche - bug)	2	0	1	1	1	1		
Listes filtrables	3	1	1	1	1	1		
Champs utilisateurs / personnalisables	3	1	0	1	1			
Mots-clés (tags) possibles (tâches, bugs, etc.)	2	1	1	1	1	1		
III Gestion de la remontée de bugs/demandes d'améliorations								
Copier/coller d'images et fichiers	3	0	0	0	0	0		
Sources du bug	3	1	1	1	1			
Attribuer un bug à un projet ou une tâche	3	1	1	1	1	1		
Attribuer une criticité	3	1	0	1	1	1		
Attribuer un impact	3	1	0	1				
Gestion des statuts	3	1	1	0	0			
Notifications de changement de statut (par mail)	3	0	1	1	1			
Statut d'avancement visible de l'extérieur	2							
IV Editeur								
Pérénité de l'éditeur	3			0	3			3
Note globale de l'outil : /104		77	79	91	**95**	76	7	22

Figure 22. Grille de décision ALM

Il apparaît donc claire que la solution qui obtient la note la plus élevée est Redmine, c'est donc cet outil que nous avons mis en place.

3.7.2 Choix d'un outil de Tests automatiques (non-régression)

Ce projet a consisté en l'élaboration de cas de tests et des scenarii de tests afin de couvrir tout le spectre fonctionnel de la solution (voir chapitre 4.5 la non régression).

En parallèle de la création de la documentation des tests, nous avons démarré un projet de choix d'une solution de tests automatiques.

Pour effectuer le choix, nous avons procédé de la même façon que pour le choix de lALM en produisant une grille de décision qui nous a permis de choisir de façon objective la solution la plus adaptée à nos besoins.

Dans la mesure où la solution GESTIMUM est développée en utilisant le langage Pascal et l'Environnement de Développement Intégré (EDI) Delphi, le choix de la solution de tests automatisés a été considérablement restreint.

A savoir que seulement deux solutions pouvaient satisfaire un des besoins les plus importants pour nous : l'inspection des objets des fenêtres.

L'inspection d'objets, consiste en la possibilité d'un programme de pouvoir voir et donc de contrôler les valeurs de certaines zones d'une fenêtre d'un autre programme.

Cette inspection est primordiale dans les tests automatisés d'une application MDI, car c'est d'après les valeurs que l'application va positionner que le programme testeur va pouvoir déterminer si le comportement est cohérent ou pas.

La grille de décision a été réduite à sa plus simple expression car le choix ne pouvait se faire que sur un seul élément qui était le prix de la solution, et seule la solution TestComplete répondait à cette exigence.

Les tests automatisés consistent en une batterie de test qui couvrent tout le spectre fonctionnel de la solution.

Le spectre fonctionnel est couvert par la création de workflows qui sont ensuite traduits en scripts de la solution de tests automatiques.

Cette tâche a été confiée à une personne qui traduit les scénarii en scripts pour la solution de tests automatiques.

3.8 Conduite de changement

Ce projet a été réalisé avec la collaboration de l'équipe. Ainsi pour obtenir l'adhésion de tous et éviter les conflits liés à la résistance au changement, il m'est apparu important d'impliquer l'équipe au complet dans toutes les décisions liées à leur activité.

De cette façon les spécifications fonctionnelles ont été élaborées avec l'aide et les idées de tous, en prenant en compte les apports de chacun.

Cette méthode a porté ses fruits car les explications étaient moins nécessaires sur le bien-fondé des changements, étant donné que les éléments avaient déjà été évoqués par chacun lors de réunions hebdomadaires consacrées aux projets.

L'adoption des nouveaux processus a été simplifiée dans la mesure où l'équipe a constaté que les nouveaux processus amenaient une simplicité et une cohérence des actions menées par l'équipe.

4 Changements apportés, aux équipes et aux méthodes de travail.

Les changements opérés à travers les différents projets sont la conséquence de l'audit des processus existant et des propositions de changement faites et validés par la direction de l'entreprise.

Le premier changement consiste en la séparation des activités de développement et de maintenance (changements structurels).

Le changement le plus significatif est l'adoption d'un ALM (Application LifeCycle Management) qui centralise, gère et donne de la visibilité sur tout le cycle de vie logiciel à travers un workflow (voir chapitre 4.3).

Un autre changement de taille, est l'introduction de l'analyse fonctionnelle systématique, ce qui s'est traduit par la création du rôle d'analyste dans l'ALM (voir chapitre 4.3.1 Redmine - Rôles).

L'application des processus issus d'ISO 14764 nous a conduit également à adopter une solution de tests automatiques, ce qui constitue un changement majeur.

4.1 Changements structurels

Les équipes ont été restructurées de telle sorte qu'elles aient la capacité de faire face aux demandes.

En effet l'équipe de développement a été divisée en 2 parties qui constituent les grands rôles du développement logiciel, à savoir, le développement et la maintenance.

A tour de rôle un membre fait partie de l'équipe de maintenance quand l'autre membre fait partie de l'équipe de développement.

Ceci permet de varier les activités de chacun des membres de l'équipe, en les alternant entre création de nouvelles fonctionnalités et modification de fonctionnalités existantes.

4.2 Changements des méthodes

Les méthodes de travail ont également évoluées. J'ai introduit des notions fondamentales telles que:

1. La modélisation

   L'utilisation d'UML pour la modélisation des demandes en vue de leurs spécifications.

2. Les spécifications fonctionnelles

   Cette notion est fondamentale pour tout développement logiciel.

En fonction des cas, j'ai demandé l'écriture de « cas d'utilisations » afin de mieux décrire les besoins exprimés.

Les « machines à état » ou « automate à états fini », nous aident à modéliser l'enchainement des différentes fenêtres des assistants proposés (les assistants sont des fenêtres qui posent des questions à l'utilisateur et réagissent en fonction des choix fait par l'utilisateur) par la solution GESTIMUM.

Il est ainsi possible de modéliser les conséquences des choix des utilisateurs dans une fenêtre et de décrire ce qui sera affiché dans les fenêtres suivantes.

Les diagrammes de séquences nous permettent maintenant de modéliser les différents états d'un document de vente ainsi que les actions nécessaires afin de le faire changer d'état.

Avec les diagrammes de séquence, nous pouvons modéliser un document de vente à partir de sa création en passant par la duplication, le transfert en un autre document puis l'archivage.

La Figure 23. Diagramme de Séquence - Devis -Facture" montre un exemple de diagramme de séquence qui modélise les étapes du transfert d'un devis vers une facture.

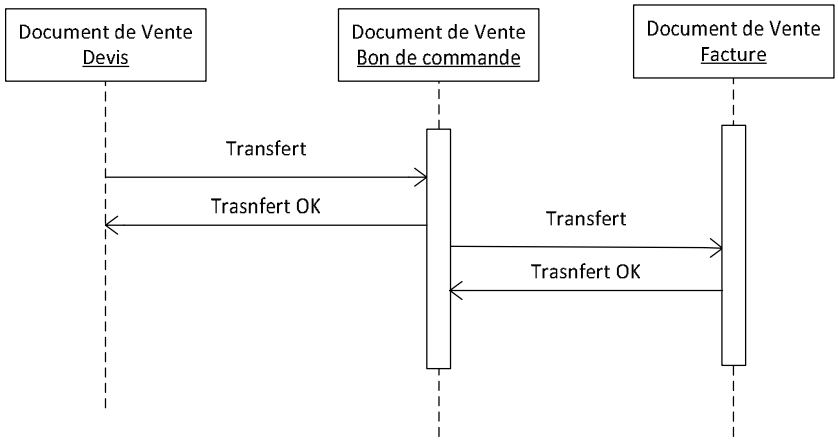


Figure 23. Diagramme de Séquence - Devis -Facture

4.3 Changements Fonctionnels, mise en place de l'ALM

La mise en place de l'ALM Redmine a bouleversé les façons de travailler des équipes, dans la mesure où cet outil a introduit un moyen de communication efficace entre les différentes équipes.

Le workflow du cycle de vie d'une demande a été implémenté dans l'outil Redmine et a donné ainsi le moyen de connaître l'état d'une demande (statut dans Redmine), son état d'avancement mais aussi le responsable de la demande et tous les échanges concernant une demande en particulier.

Nous allons voir le cycle de vie d'une demande afin de montrer comment les différents rôles interviennent et comment elles sont représentées dans Redmine.

Le workflow dans Redmine est basé à la fois sur les différents statuts d'une demande (Figure 24. Statuts d'une demande), les rôles et les types de demande (appelés trackers dans Redmine).

Statuts de demandes

Nouveau statut | Mettre à jour l'avancen

Statut	% réalisé	Valeur par défaut	Demande fermée	Trier
Nouvelle	0	✔		
Acceptée	10			
A complèter	10			
A spécifier	10			
A réaliser	30			
A intégrer	70			
A recetter	70			
Echec du recettage	50			
Terminée	100		✔	
Refusée	100		✔	

Figure 24. Statuts d'une demande

En suivant le workflow du cycle de vie d'une demande, c'est-à-dire en modifiant le statut d'une demande, l'outil Redmine permet d'en connaître l'état d'avancement.

La Figure 24. Statuts d'une demande, montre les différents états d'avancement d'une demande ainsi que le pourcentage d'avancement correspondant à chaque état.

Les demandes, leur contenu et leur statut étaient modifiées, par les différents acteurs : rapporteurs, analystes, développeurs et testeurs, ces modifications reflètent la situation de la demande au sein du workflow.

4.3.1 Redmine - Rôles

Redmine définit des rôles, correspondant aux permissions sur les différents projets Redmine.

Ainsi l'on pourra définir des rôles et les associer aux différents acteurs des projets, des demandes, des documents, des fichiers, etc.

Les rôles définis dans Redmine dans le cas qui nous intéresse, sont :

Rôles

Nouveau rôle

Rôle	Trier	
Manager		Supprimer
Rapporteur		Supprimer
Analyste		Supprimer
Développeur		Supprimer
Testeur		Supprimer

Figure 25. Redmine - Rôles définis

Chaque Rôle a des caractéristiques différentes, à savoir :

- Manager

  Le manager est responsable de l'équipe de développement, dont le rôle est de coordonner les actions de l'ensemble de l'équipe de développement.

- Rapporteur

  Le rapporteur est un membre appartenant ou non à l'équipe de développement et qui a pour mission la création des demandes d'évolution et de maintenance.

- Analyste

  L'Analyste est un membre de l'équipe services qui est en charge de l'analyse fonctionnelle des demandes en vue de leur développement.

- Développeur

  Le développeur est responsable du codage de la demande, c'est-à-dire de la traduction du besoin en une réalité informatique.

- Testeur

  Le testeur est la personne qui teste le produit des développeurs afin de s'assurer que la demande a bien été codée en respectant les spécifications fonctionnelles.

Ainsi le manager peut mettre en place des projets pour que les rapporteurs créent des demandes, puis que les développeurs les traitent et que les testeurs puissent tester ce que les développeurs ont produit.

Le rapporteur ne peut que créer ou modifier une demande. Voici comment (Figure 26. Redmine - Droit du Rôle Rapporteur) dans Redmine les droits sur demande sont définis :

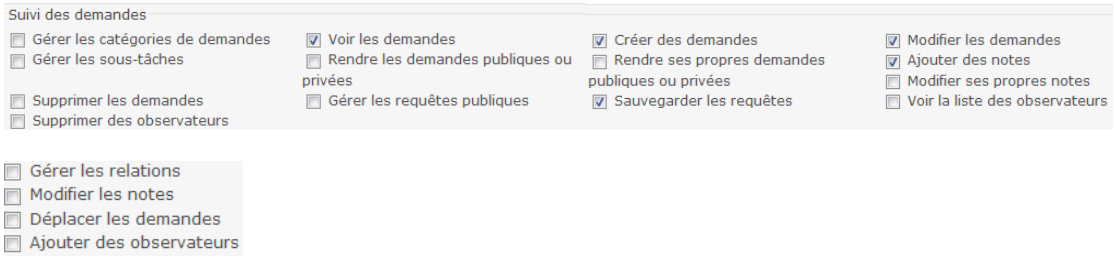


Figure 26. Redmine - Droit du Rôle Rapporteur

Le rôle développeur et le rôle testeur ont des droits différents, comme le montre la Figure 26. Redmine - Droit du Rôle Rapporteur", car ils doivent intervenir dans d'autres parties du développement.

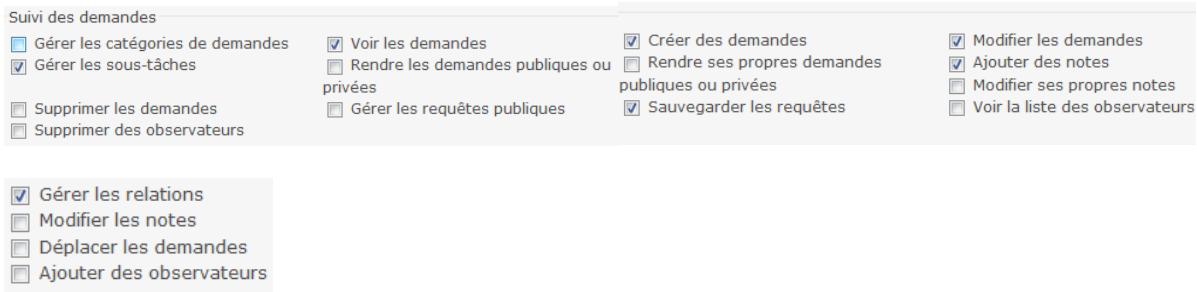


Figure 27. Redmine - Droits du Rôle Développeur et Testeur

Redmine permet de synthétiser (voir Figure 28. Redmine - Synthèse des permissions") les droits sur tous les objets qui constituent cette solution et ainsi de pouvoir clairement et facilement définir le périmètre d'action (via des permissions) de chaque rôle.

Cette synthèse est représentée sous forme d'une matrice, dont la première colonne représente la permission sur un objet (Projet, forum, calendrier, documents, fichiers, diagramme de Gantt et les demandes).

Chaque case à l'intersection d'une ligne et une colonne représente la permission accordée (cochée) ou pas (décochée) pour chacun des rôles.

Par exemple un manager peut créer un projet (case cochée dans la colonne manager) mais un développeur ne peut pas (case décochée dans la colonne développeur).

Permissions	Manager	Développeur	Rapporteur	Testeur	Analyste
Créer un projet	☑	☐	☐	☐	☐
Modifier le projet	☑	☐	☐	☐	☐
Choisir les modules	☑	☐	☐	☐	☐
Gérer les membres	☑	☐	☐	☐	☐
Gérer les versions	☑	☑	☐	☐	☐
Créer des sous-projets	☑	☐	☐	☐	☐
Forums de discussion					
Gérer les forums	☑	☐	☐	☐	☐
Poster un message	☑	☑	☑	☑	☑
Modifier les messages	☑	☐	☐	☐	☐
Modifier ses propres messages	☑	☑	☑	☐	☐
Supprimer les messages	☑	☐	☐	☐	☐
Supprimer ses propres messages	☑	☐	☐	☐	☐
Calendrier					
Voir le calendrier	☑	☑	☑	☑	☑
Publication de documents					
Gérer les documents	☑	☐	☐	☐	☐
Voir les documents	☑	☑	☑	☑	☑
Publication de fichiers					
Gérer les fichiers	☑	☑	☐	☐	☐
Voir les fichiers	☑	☑	☑	☑	☑
Gantt					
Voir le gantt	☑	☑	☑	☑	☑
Suivi des demandes					
Gérer les catégories de demandes	☑	☑	☐	☐	☐
Voir les demandes	☑	☑	☑	☑	☑
Créer des demandes	☑	☑	☑	☑	☑
Modifier les demandes	☑	☑	☐	☑	☐
Gérer les relations	☑	☑	☐	☐	☐
Gérer les sous-tâches	☑	☑	☐	☐	☐
Rendre les demandes publiques ou privées	☑	☐	☐	☐	☐
Rendre ses propres demandes publiques ou privées	☑	☐	☐	☐	☐
Ajouter des notes	☑	☑	☑	☑	☑
Modifier les notes	☑	☐	☐	☐	☐
Modifier ses propres notes	☑	☐	☐	☐	☐
Déplacer les demandes	☑	☐	☐	☐	☐
Supprimer les demandes	☑	☐	☐	☐	☐
Gérer les requêtes publiques	☑	☐	☐	☐	☐
Sauvegarder les requêtes	☑	☑	☑	☑	☑
Voir la liste des observateurs	☑	☐	☐	☐	☐
Ajouter des observateurs	☑	☐	☐	☐	☐
Supprimer des observateurs	☑	☐	☐	☐	☐

Figure 28. Redmine - Synthèse des permissions

4.3.2 Redmine – Workflow - Manager

Le cycle de vie logiciel, est suivi grâce à la mise en place d'un workflow, qui est basé sur les rôles et les statuts des demandes.

Dans la terminologie Redmine un tracker est un type de demande qui sert à piloter un workflow.

La Figure 29. Redmine - Workflow – Manager, nous montre la représentation du workflow d'une anomalie (tracker anomalie) pour le rôle manager, sous forme d'une matrice.

Cette matrice est composée d'une première colonne qui est le statut de départ (intitulé « Statut actuel ») de l'objet anomalie et les statuts autorisés.

Un manager peut faire passer une demande du statut « nouvelle » (à la création de la demande) au statut « acceptée », à « A compléter » ensuite « A spécifier » puis revenir au statut « à compléter » et ainsi modifier tous les états depuis n'importe quel état vers n'importe quel état.

Statut actuel	Nouveaux statuts autorisés							
	Nouvelle	Acceptée	A compléter	A spécifier	A réaliser	A Recetter	Terminée	Refusée
Nouvelle	☐	☑	☑	☑	☑	☑	☑	☑
Acceptée	☑	☑	☑	☑	☑	☑	☑	☑
A compléter	☑	☑	☑	☑	☑	☑	☑	☑
A spécifier	☑	☑	☑	☑	☑	☑	☑	☑
A réaliser	☑	☑	☑	☑	☑	☑	☑	☑
A Recetter	☑	☑	☑	☑	☑	☑	☑	☑
Terminée	☑	☑	☑	☑	☑	☑	☑	☑
Refusée	☑	☐	☐	☐	☐	☐	☐	☐

Figure 29. Redmine - Workflow – Manager

La manager est un cas à part car il a tous les droits sur les statut d'une demande d'un projet.

La Figure 30. Redmine - Etats possibles des demandes avec le rôle Manager montre les états possibles d'une demande lorsque l'on a le rôle manager. On peut en ffet passer d'un état vers tous les autres et cela dans les 2 sens.

Les différents états possibles : Nouvelle (état initial), Acceptée, A compléter, A Spécifier, A Réaliser, A Recetter, Terminée et Refusée sont représentés par les cercles comportant leur noms. Les différentes actions qui font passer une demande d'un état à un autre sont représentées par le nom de l'état cible dans les arcs.

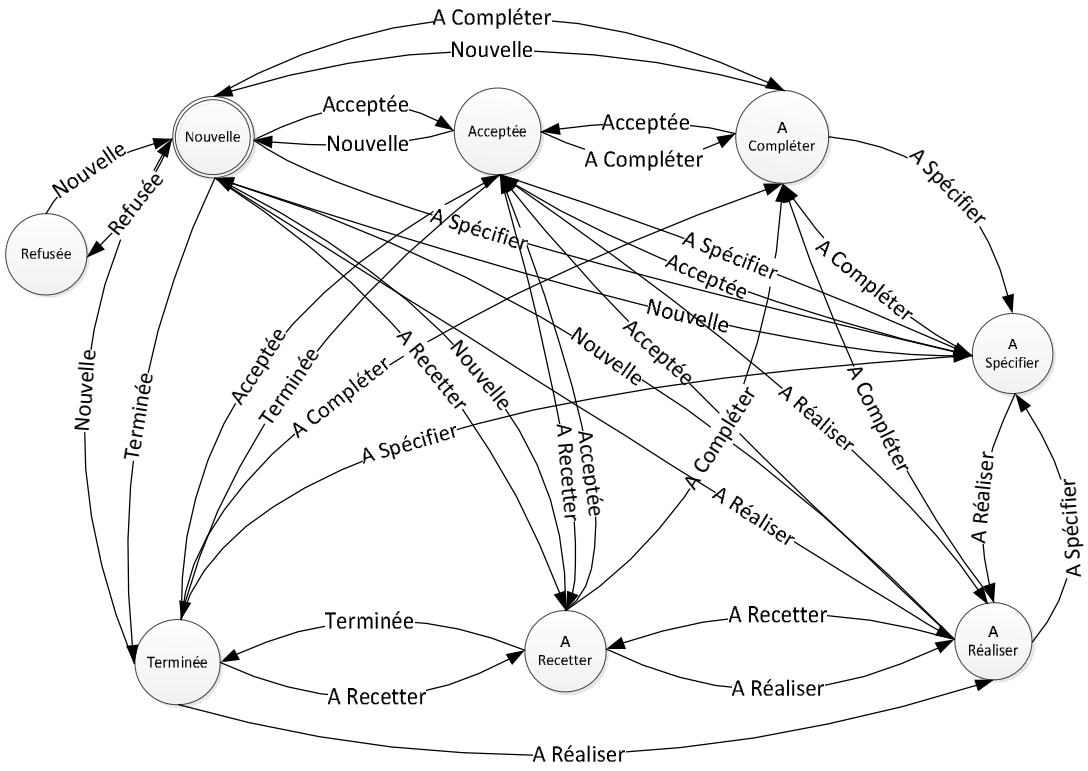


Figure 30. Redmine - Etats possibles des demandes avec le rôle Manager

4.3.3 Redmine – Workflow – Rapporteur – Anomalie

Le rôle rapporteur confère à l'utilisateur qui en est désigné le droit d'intervenir dans le processus afin de signaler des anomalies ou de faire des demandes d'évolution, mais il n'est qu'un créateur de demandes, il ne peut modifier les statuts de ces dernières.

La Figure 31.Redmine - Workflow - Rapporteur" nous montre que le rôle rapporteur peut créer une demande mais en aucun cas modifier son statut.

Ce rôle ne peut pas faire passer la demande dans un autre état après l'avoir créé.

Statut actuel	Nouveaux statuts autorisés							
	Nouvelle	Acceptée	A compléter	A spécifier	A réaliser	A Recetter	Terminée	Refusée
Nouvelle	☑	☐	☐	☐	☐	☐	☐	☐
Acceptée	☐	☐	☐	☐	☐	☐	☐	☐
A compléter	☐	☐	☐	☐	☐	☐	☐	☐
A spécifier	☐	☐	☐	☐	☐	☐	☐	☐
A réaliser	☐	☐	☐	☐	☐	☐	☐	☐
A Recetter	☐	☐	☐	☐	☐	☐	☐	☐
Terminée	☐	☐	☐	☐	☐	☐	☐	☐
Refusée	☐	☐	☐	☐	☐	☐	☐	☐

Figure 31.Redmine - Workflow - Rapporteur

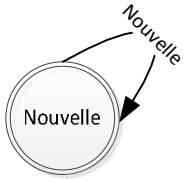


Figure 32. Redmine - Etats possibles d'une demande avec le rôle rapporteur

La Figure 32. Redmine - Etats possibles d'une demande avec le rôle rapporteur" montre que le rôle rapporteur ne permet que l'ajout d'une nouvelle demande et aucune autre action n'est possible sur l'état de cette dernière.

4.3.4 Redmine – Workflow – Analyste – Anomalie

Le rôle analyste, confère à l'utilisateur le droit d'accepter une demande (statut « acceptée »), de demander un complément d'information au rapporteur (statut « A compléter »), de commencer l'analyse (statut « A spécifier ») et une fois l'analyse terminée changer son statut à « A réaliser ».

Statut actuel	Nouveaux statuts autorisés							
	Nouvelle	Acceptée	A compléter	A spécifier	A réaliser	A Recetter	Terminée	Refusée
Nouvelle	☐	☑	☑	☑	☑	☐	☐	☐
Acceptée	☐	☐	☑	☑	☑	☐	☐	☐
A compléter	☐	☑	☐	☑	☑	☐	☐	☐
A spécifier	☐	☑	☑	☐	☑	☐	☐	☐
A réaliser	☐	☑	☑	☑	☐	☐	☐	☐
A Recetter	☐	☐	☐	☐	☐	☐	☐	☐
Terminée	☐	☐	☐	☐	☐	☐	☑	☐
Refusée	☐	☑	☐	☐	☐	☐	☐	☑

Figure 33.Redmine - Workflow - Analyste – Anomalie

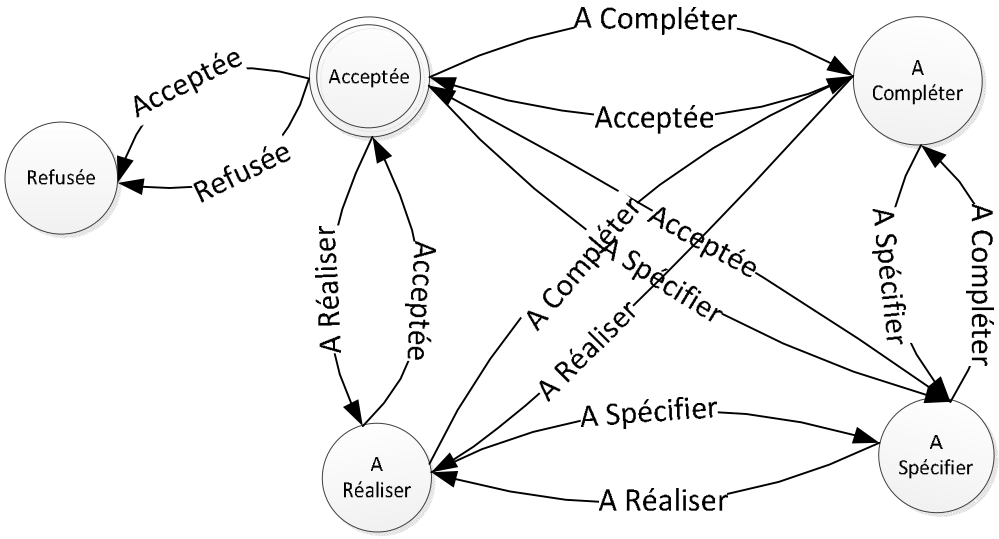


Figure 34. Redmine - Etats possibles d'une demande avec le rôle Analyste

Figure 34. Redmine - Etats possibles d'une demande avec le rôle Analyste montre que le rôle Analyste peut faire passer une demande de l'état « Acceptée » (état initial) à « A compléter », « A Spécificier », et « A Réaliser » et ce dans les deux sens.

Ce workflow correspond dans le processus de développement à la partie illustrée par la Figure 33.Redmine - Workflow - Analyste – Anomalie.

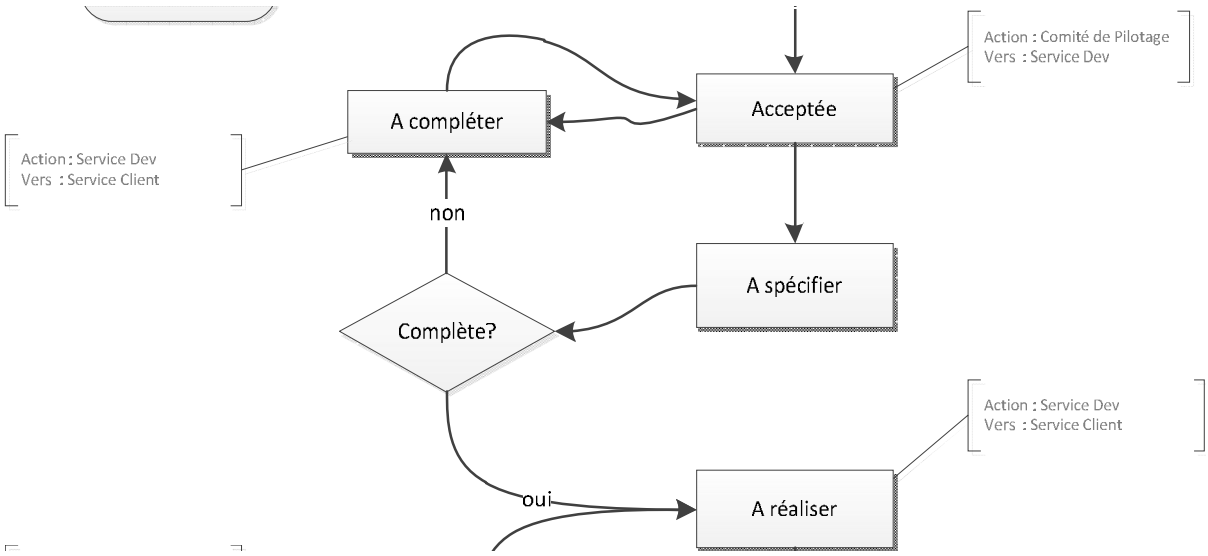


Figure 35. Processus de Développement, Analyse

4.3.5 Redmine – Workflow – Développeur - Anomalie

Nous allons voir les différents workflows du rôle développeur par rapport à une demande de maintenance, en l'occurrence une anomalie.

Figure 35. Processus de Développement, Analyse" nous montre que le rôle développeur peut prendre en compte une demande, statut *à réaliser*, et changer son statut pour *à recetter* un fois qu'il a fini son travail.

Le développeur ne peut faire passer le statut de la demande à « terminée » ce qui signifirait que la demande a été validé par la recette.

Statut actuel	Nouveaux statuts autorisés							
	Nouvelle	Acceptée	A compléter	A spécifier	A réaliser	A Recetter	Terminée	Refusée
Nouvelle	☐	☐	☐	☐	☑	☑	☐	☐
Acceptée	☐	☐	☐	☐	☑	☑	☐	☐
A compléter	☐	☐	☐	☐	☑	☑	☐	☐
A spécifier	☐	☐	☐	☐	☑	☑	☐	☐
A réaliser	☐	☐	☐	☐	☑	☑	☐	☐
A Recetter	☐	☐	☐	☐	☑	☐	☐	☐
Terminée	☐	☐	☐	☐	☐	☐	☐	☐
Refusée	☐	☑	☐	☐	☐	☐	☐	☐

Figure 36. Redmine - Workflow - Développeur - Anomalie

La Figure 36. Redmine - Workflow - Développeur - Anomalie nous montre la partie du nouveau processus de développement qui concerne les actions d'un développeur.

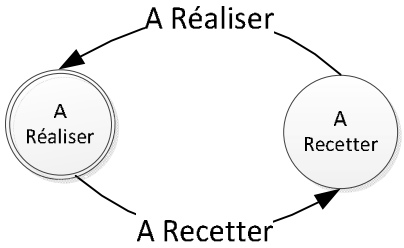


Figure 37. Redmine - Etats possibles d'une demande avec le rôle Développeur

La Figure 37. Redmine - Etats possibles d'une demande avec le rôle Développeur" montre que le rôle développeur permet de faire passer un demande de l'état « A Réaliser » (état initial) à l'état « A recetter » seulement et ce dans les seux sens.

Par l'intermédiaire du workflow développeur sur le tracker anomalie (Figure 38. Processus, partie développeur") nous pouvons circonscrire les actions du développeur et ainsi nous conformer à la norme ISO 12207 qui nous indique qu'il est nécessaire de définir les autorisations et les devoirs de chacun.

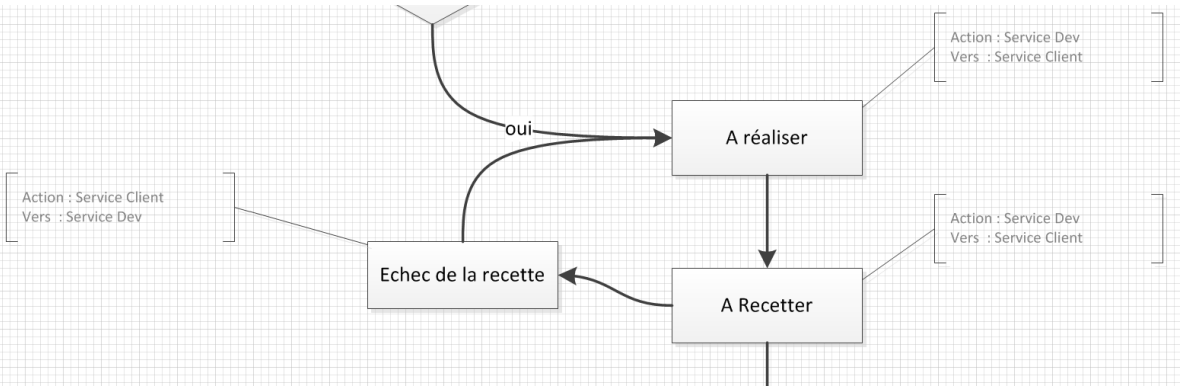


Figure 38. Processus, partie développeur

4.3.6 Redmine – Workflow – Testeur – Anomalie

Le rôle testeur permet à l'utilisateur de commencer les tests; il suffit pour cela que l'utilisateur choisisse une demande dans la liste des demandes ayant le statut « A recetter ».

De cette façon nous remplissons les éxigeance d'ISO 14764, à savoir qu'il est nécessaire de fournir clairement une documentation aux testeurs sur les tests à effectuer.

La Figure 39. Redmine - Workflow - Testeur – Anomalie", montre que le rôle testeur peut intervenir sur les demandes « A recetter » (changées par le développeur) et que si le test est concluant, alors changer son statut à « Terminée », mais si le test échoue alors le testeur doit changer le statut et le positionner sur « A réaliser ».

De cette façon le développeur verra qu'une demande est « revenue » dans sa liste des demandes à réaliser.

Le testeur doit indiquer les motifs pour lesquels il estime que le test a échoué.

Statut actuel	Nouveaux statuts autorisés							
	Nouvelle	Acceptée	A compléter	A spécifier	A réaliser	A Recetter	Terminée	Refusée
Nouvelle	☐	☐	☐	☐	☑	☑	☑	☐
Acceptée	☐	☐	☐	☐	☐	☑	☑	☐
A compléter	☐	☐	☐	☐	☐	☐	☐	☐
A spécifier	☐	☐	☐	☐	☐	☐	☐	☐
A réaliser	☐	☐	☐	☐	☑	☑	☑	☐
A Recetter	☐	☐	☐	☐	☑	☐	☑	☐
Terminée	☐	☐	☐	☐	☑	☑	☐	☐
Refusée	☐	☐	☐	☐	☐	☐	☐	☐

Figure 39. Redmine - Workflow - Testeur – Anomalie

La Figure 41. Processus partie testeur, nous montre la partie du cycle de vie dans laquelle le testeur peut intervenir à partir des demandes dont le statut est « A recetter » et faire revenir si besoin est cette demande au développeur si les tests échouent, dans le cas contraire la demande est terminée.

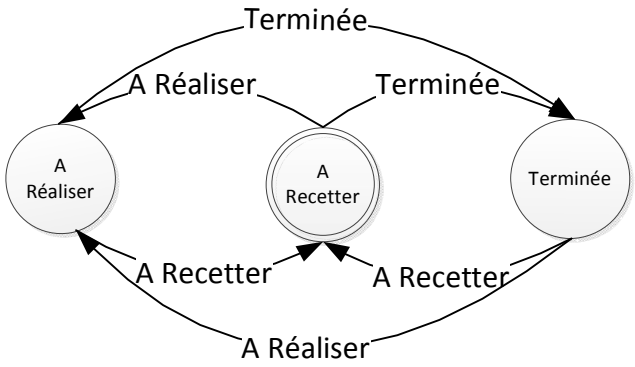


Figure 40. Redmine - Etats possibles d'une demande avec le rôle Testeur

La Figure 40. Redmine - Etats possibles d'une demande avec le rôle Testeur" montre qu'à partir de l'état « A Recetteré (état initial) le rôle testeur peut valider une demande parce que le test a réussi et faire passer la demande dans l'état « Terminée ». Mais si le test échoue alors la demande revient à l'état « A Réaliser » afin que le développement soit repris.

Cette partie du cycle de vie est équivalente au workflow implémenté et illustré par la Figure 39. Redmine - Workflow - Testeur – Anomalie.

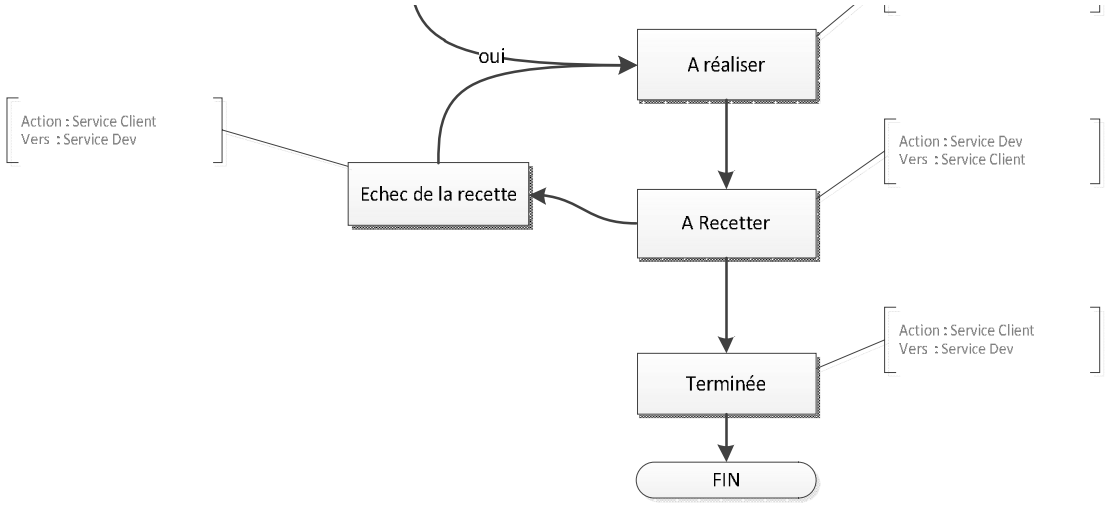


Figure 41. Processus partie testeur

Nous venons de voir comment dans Redmine le cycle de vie avait été implémenté et comment en fonction de chaque rôle les utilisateurs pouvaient intervenir sur l'état d'une demande.

4.4 Exemple d'application du workflow

Nous allons voir l'application du workflow d'une demande de maintenance à travers Redmine et les différents acteurs.

Informations générales sur la demande :

Anomalie #13566

Mettre à jour | Surveiller | Dupliquer | Copier | Déplacer | Supprimer

Tri des colonnes incorrect dans la fenêtre de paramétrage des grilles

Ajouté par **Emmanuel Triballier** il y a **6 mois.** Mis à jour il y a **environ un mois.**

Statut:	Terminée	**Début:**	14/10/2011
Priorité:	Moyenne	**Echéance:**	14/10/2011
Assigné à:	-	**% réalisé:**	100%
Catégorie:	ERP-Propriétés des grilles		
Version cible:	4.5.2 (498)		
Rapportée par:	Jean-Luc Brison		
Constatée dans la version:	4.5		
Apparue dans la version:			
Sévérité de l'anomalie:	Gênante		
Clients impactés:	Tous les clients		
Noms des clients:	GESTIMUM - JLB 4.2 svp		
A chiffrer:			
Reproduction de l'anomalie:			
Estimation de l'effort:			
Fourchette de temps:			
Modèle-état ?:	Non		
Publiable:	Oui		
Texte à publier:	Nouvelle méthode de tri dans l'onglet Colonnes de la boite de dialogue de paramétrage des grilles, insensible aux majuscules et minuscules.		
Publié:	Oui		

Figure 42 Informations sur une anomalie

Dans les informations que Redmine nous fournit sur une anomalie, il est possible de voir qui a rapporté l'anomalie « Rapporté par », ainsi que la version dans laquelle cette anomalie a été constatée « Constatée dans la version ».

Une information importante est la possibilité de publier des informations sur cette anomalie, le champ « texte à publier » sert à créer de façon automatique un fichier contenant les notes de version ou « lisez-moi »

Cette anomalie sera corrigée dans la version cible 4.5.2, désigné par le champ « version cible ».

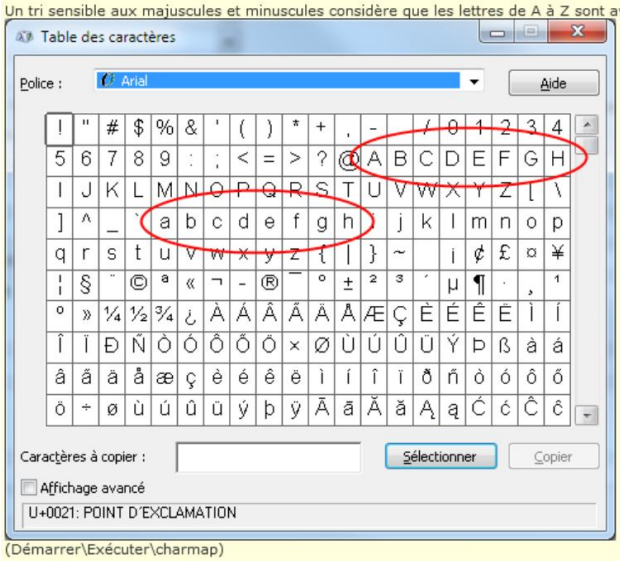
Cette anomalie comporte également la mention gênante dans la catégorie « Sévérité de l'anomalie ». Cette mention priorise l'anomalie afin de lui assigner une version cible.

Redmine offre la possibilité de créer automatiquement la roadmap de la version en question 4.5.2, qui est en fait l'ensemble des demandes (maintenance et évolution) assignées à cette version et de voir l'état d'avancement de la version en question.

Description de demande

Dans la boite de dialogue de paramétrage des grilles, il faut revoir la méthode de tri des colonnes : le tri sensible à la casse doit être remplacé par un tri insensible à la casse.

Un tri sensible aux majuscules et minuscules considère que les lettres de A à Z sont avant les lettres de a à z. On le voit dans la table des caractères :



(Démarrer\Exécuter\charmap)

L'utilisateur qui cherche "TVA", à la fin des T, ne va pas trouver, car il va être classé au début des T, avant "Table" !

Figure 43. Descriptif d'une demande

Une fois l'anomalie créé, il convient de statuer sur l'acceptation ou non de l'anomalie afin qu'elle rentre dans le cycle de développement de la version 4.5.2

Mis à jour par Stéphane Venet il y a 2 mois

- **Statut** changé de *Nouvelle* à *Acceptée*
- **Sévérité de l'anomalie** changé de *Moyenne* à *Gênante*

Figure 44. Acceptation d'une demande

L'anomalie étant maintenant acceptée, elle est en développement

Mis à jour par Emmanuel Triballier il y a environ un mois

- **Statut** changé de *Acceptée* à *A recetter*
- **% réalisé** changé de *0* à *60*

Fait. Tri désormais insensible à la casse.

Figure 45. Fin de développement d'une demande

Le développement est terminé et la correction de l'anomalie peut être recettée, le statut a changé de « Acceptée » à « A recetter ».

Mis à jour par Jonathan Fontaine il y a environ un mois

- **Statut** changé de *A recetter* à *Terminée*
- **% réalisé** changé de *60* à *100*

Figure 46. Recette d'une anomalie

L'anomalie a été corrigée, puis la correction a été recettée avec succès, le statut passe donc « A recetter » à « Terminée »

Mis à jour par Jonathan Fontaine il y a environ un mois #5

- **Publié** changé de *Non* à *Oui*

Figure 47. Publication de la correction d'une anomalie

Une fois le cycle terminé, il est possible de publier ou non le résultat des actions menées pour satisfaire cette demande.

Le statu « Publié » étant maintenant positionné à oui, cela veut dire le résultat de tout le workflow va pouvoir être montré dans la roadmap.

4.5 Tests automatiques

Nous avons élaboré des scénarii de tests qui partent d'un schéma illustré par la **Erreur ! Source du renvoi introuvable.** qui décrit toutes les étapes nécessaires à la création d'un document de vente.

Ces scénarii créés sont ensuite traduits en scripts qui sont ensuite interprétés par la solution TestComplete et simulent l'utilisation réelle du produit testé.

Les tests automatiques sont effectués par la solution choisie (voir chapitre 3.7.2 Choix d'un outil de Tests Automatiques) : « TestComplete ».

Cette solution a la particularité de simuler les actions d'un utilisateur en adressant les objets (et leur propriétés) des fenêtres de l'application.

Nous avons ainsi constitué une bibliothèque de scénarii qui couvrent tous les domaines fonctionnels de l'application.

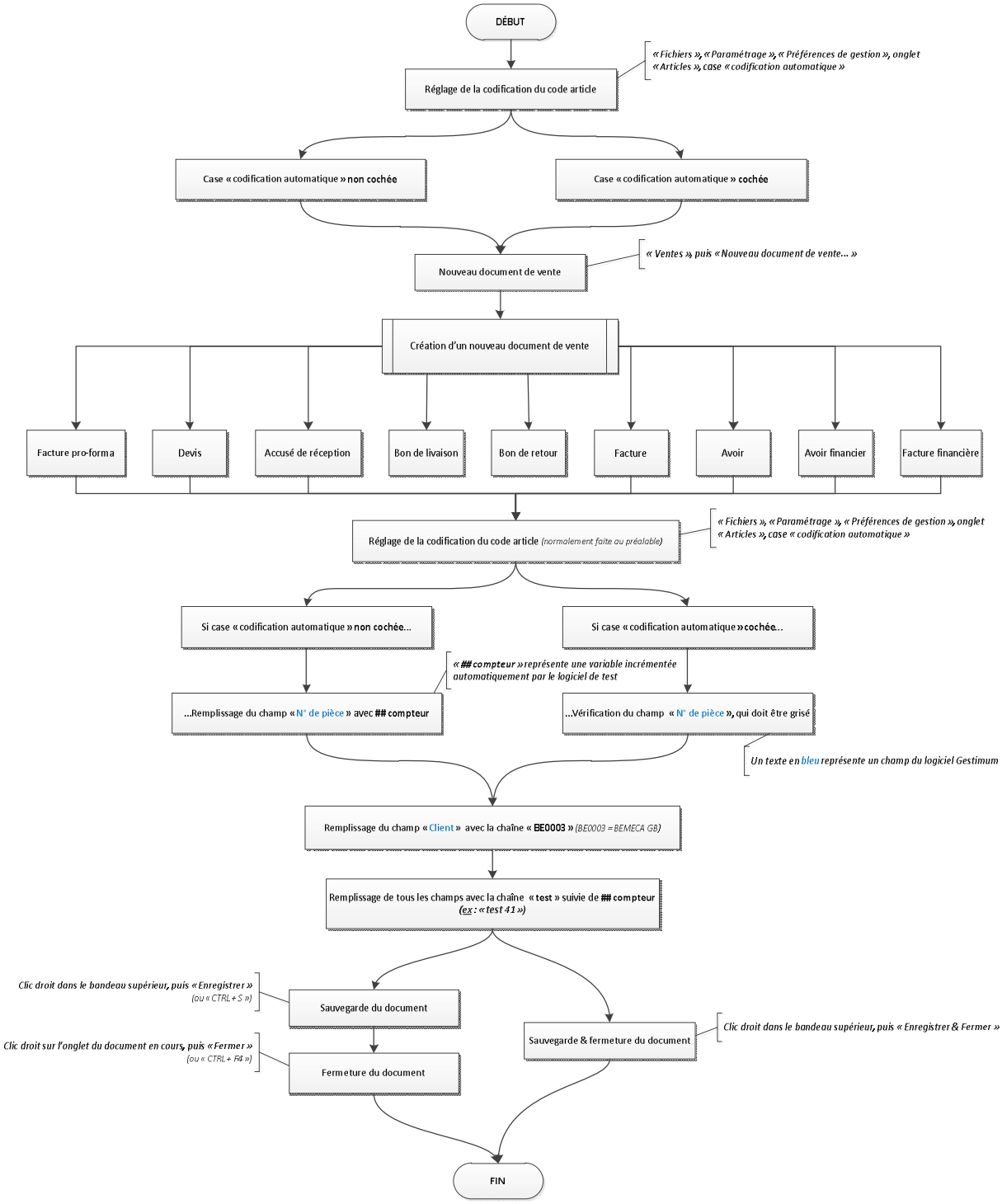


Figure 48. Scénario de création de tous les documents de vente

4.6 La non régression

La non régression a été traitée de telle sorte que, d'une part nous nous sommes dotés d'une solution de tests automatiques et d'autre part nous avons préparé ces tests en créant des scenarii de tests et des cas de tests (voir chapitre 4.5).

La solution de tests automatiques possède des capacités intéressantes pour la solution GESTIMUM car elle peut observer les objets crées par GESTIMUM.

La Figure 49. Capture d'une zone" nous montre comment la solution de tests automatiques peut capturer le contenu d'une zone de la fenêtre, ici le prix brut qui peut être comparé à ce que la fenêtre est en train de montrer.

Il est ainsi possible de contrôler le déroulement des opérations effectuées, par exemple en rentrant un prix brut dans la ligne et en comparant le résultat en bas à droite.

La solution TestComplete est capable de dérouler le scénario et de déterminer si la solution GESTIMUM se comporte comme escompté ou non.

Le déroulement des scripts nous aident à faire des tests de non régression automatiques.

C'est en analysant les résultats de ces tests qu'il nous est possible de constater qu'une fonction existante renvoie bien les résultats escomptés, ce qui signifie qu'il n'y a pas de régression pour la fonction testée.

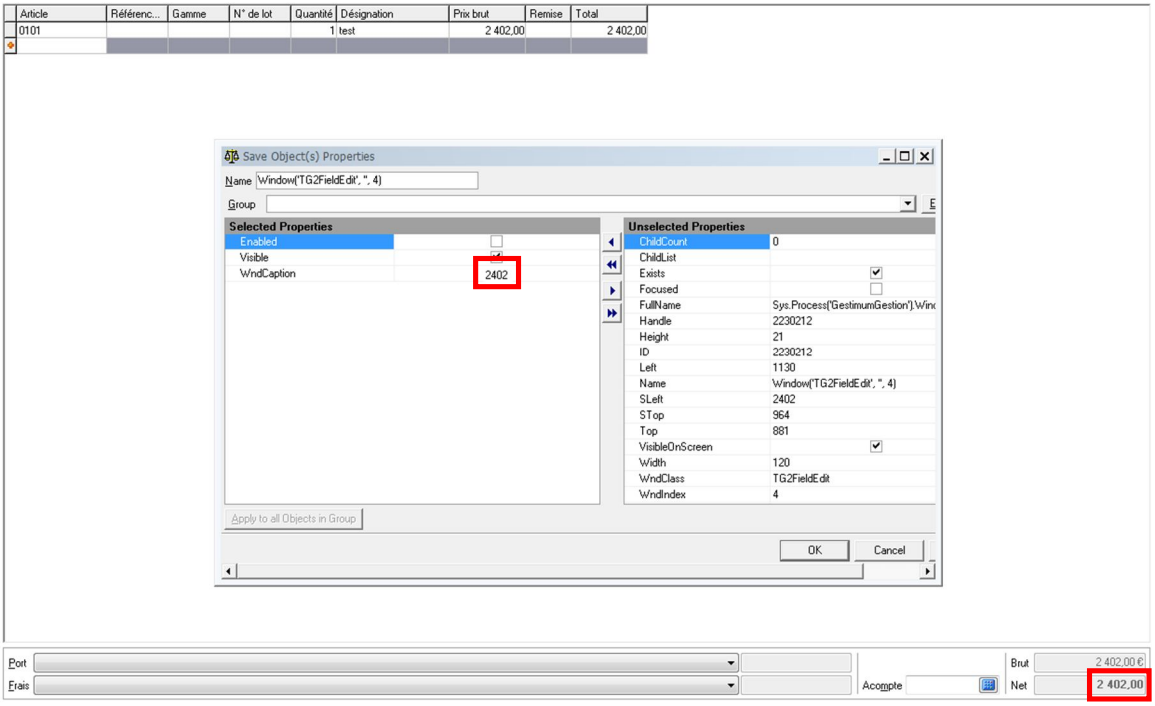


Figure 49. Capture d'une zone

4.7 La recette

La phase de recette a été mise en place, grâce à Redmine et aux changements de management qui ont été opérés.

La Figure 50. Redmine - Liste des demandes à recetter, montre les demandes que l'équipe de tests doit valider car elles ont le statut « A Recetter »

Il suffit pour cela d'ouvrir chaque demande , regarder son contenu, effectuer le test demandé et changer son statut en fonction du résultat du test.

#	Tracker	Statut	Priorité	Sujet	Assigné à	Mis-à-jour
5	Anomalie	A Recetter	Normal	Adresse non affichée	Emmanuel Triballier	15/03/2012 20:38
4	Anomalie	A Recetter	Normal	Total HT non conforme	Emmanuel Triballier	15/03/2012 20:38
3	Anomalie	A Recetter	Normal	Notification de dépassement de budget incorrecte	Emmanuel Triballier	15/03/2012 20:38
2	Anomalie	A Recetter	Normal	Budget non conforme	Emmanuel Triballier	15/03/2012 20:38
1	Anomalie	A Recetter	Normal	Affichage Euro non prise en compte	Emmanuel Triballier	15/03/2012 20:38

Figure 50. Redmine - Liste des demandes à recetter

Cette liste montre bien l'exhaustivité des demandes à valider et donne ainsi tous les éléments nécessaires aux testeurs.

Ces changements constituent la refonte des processus de développement, la mise en place de l'ALM, la mise en place de TestComplete, solution de tests automatiques pour la non régression ainsi que la mise en place d'une phase de recette.

5 Conclusion

Nous venons de voir comment un éditeur de logiciels de petite taille a développé un progiciel de gestion intégré nommé GESTIMUM.

Les méthodes employées pour le développement de cette solution ont été analysées, critiquées puis modifiées.

Le résultat de cette analyse m'a donc conduit à proposer les changements suivants:

Diminution du temps, amélioration de la qualité.

1. Restructuration de l'équipe de développement

   L'équipe de développement a été scindée en deux afin de parallèlement les besoins de correction de bugs et d'ajout de nouvelles fonctionnalités.

   Un des résultats obtenus par cette structuration a été une meilleure prise en compte des demandes (maintenance et évolution) et une optimisation des ressources de développement.

   Nous avons ainsi parallélisé les traitements des demandes de maintenance et les demandes d'évolution.

   Une partie de l'équipe étaient dédiée pendant un temps prédéfini au demandes de maintenance, pendant que l'autre partie de l'équipe développait les nouvelles fonctionnalités.

2. Mise en place de nouvelles méthodes de développement

   Formalisation des demandes à l'aide d'UML.

   - Formalisation des besoins et création de Spécifications fonctionnelles avec cas d'utilisation.

- Formalisation des interactions en utilisant la modélisation via des machines à état, ou des diagrammes de séquence.

  La mise en place de l'utilisation d'UML a aidé à améliorer la qualité logiciel, sa maintenabilité et son évolutivité.

  En effet la formalisation des demandes à l'aide d'UML en utilisant certains diagrammes seulement, tels que : diagramme de séquences, diagrammes d'état et les machines à état finis, ont aidé à l'amélioration de la qualité en améliorant l'analyse des demandes.

  La maintenabilité a pu être améliorée car la modélisation permet d'appréhender une fonctionnalité complexe sous forme de graphe.

  L'évolutivité a aussi pu être améliorée car les schémas créés (diagrammes de séquences, diagrammes d'états, machine à états, etc) d'une fonctionnalité peuvent être utilisés pour faire évoluer la solution en moins de temps qu'auparavant.

3. Mise en place d'un ALM

   La mise en place de l'ALM a été sans nul doute le changement le plus visible et le plus important car les résultats obtenus grâce à ce dernier sont :

   1. Une vision globale des développements en cours.
   2. Une vision globale des développements à venir (Roadmap).
   3. Une transmission cohérente des informations entre les différentes équipes, via les différents workflows mis en place.
   4. Une phase de recette intégrée au cycle de vie du développement , grâce au workflow et ainsi mieux cadrée.

4. Mise en place d'une solution de tests automatisés

Cette solution nous a permis de commencer un long travail de modélisation des fonctionnalités présentes dans la solution GESTIMUM afin d'en automatiser les tests.

Tous ces changements nous ont conduits à constater que la mise en place des méthodes éprouvées, telle qu'ISO 12207 et ISO 14764 étaient bénéfiques pour l'entreprise.

En effet la résultante de ces changements a été une diminution des bugs constatés par les clients, une meilleure prise en compte des demandes car les demandes étaient systématiquement analysées et validées.

Même si tous les projets prévus à l'origine n'ont pas tous été exécutés, il n'en demeure pas moins que les souhaits de l'entreprise en termes d'accroissement de la qualité, ont été satisfaits.

Une des conséquences de cet accroissement de la qualité est une diminution des coûts de développement car si la qualité augmente, les demandes de maintenance diminuent et c'est donc un objectif atteint de ce point de vue-là.

Ce que nous pouvons retenir de cette expérience, c'est que les méthodes issues du monde industriel (ISO) en matière de taylorisation du travail (les workflows) induisent des changements de visions au sein des équipes. Il faut penser non pas à leur seul travail mais aussi à la place de leur travail dans un ensemble cohérent qui est la solution développée.

Glossaire

Annexes

I.1 Qu'est-ce que l'ISO

ISO est l'acronyme de International Organization for Standardization et a été fondé en 1947 et basée en Suisse.

Le nom pourrait avoir différents acronymes en différentes langues (« IOS » en anglais, « OIN » pour Organisation internationale de normalisation en français), ses fondateurs ont décidé de lui donner un nom court et à usages multiples. Ils ont choisi « ISO » dérivé du Grec isos qui veut dire égale à. Quelque-soit le pays, quelque-soi la langue, le nom court de l'organisation est le même: ISO.

Son objectif est de promouvoir le développement de la standardisation dans le monde afin de faciliter les échanges de biens et des services.

ISO (International Organization for Standardization) est le plus important développeur et éditeur de standard Internationaux.

ISO est un réseau d'instituts nationaux de 162 pays, un membre par pays, avec un secrétariat central à Genève qui coordonne le système.

ISO est une organisation non gouvernementale qui constitue un point entre le secteur public et le secteur privé.

D'une part les instituts membres font partie des structures gouvernementales de leur pays ou sont mandatés par leur gouvernements respectifs.

D'un autre côté d'autres membres ont leur racines dans le secteur privé ayant été fondés par des partenariats nationaux d'associations d'industries.

Ainsi, ISO arrive à un consensus de solutions qui satisfont les besoins des marchés et les larges besoins des sociétés.

Structure de l'ISO

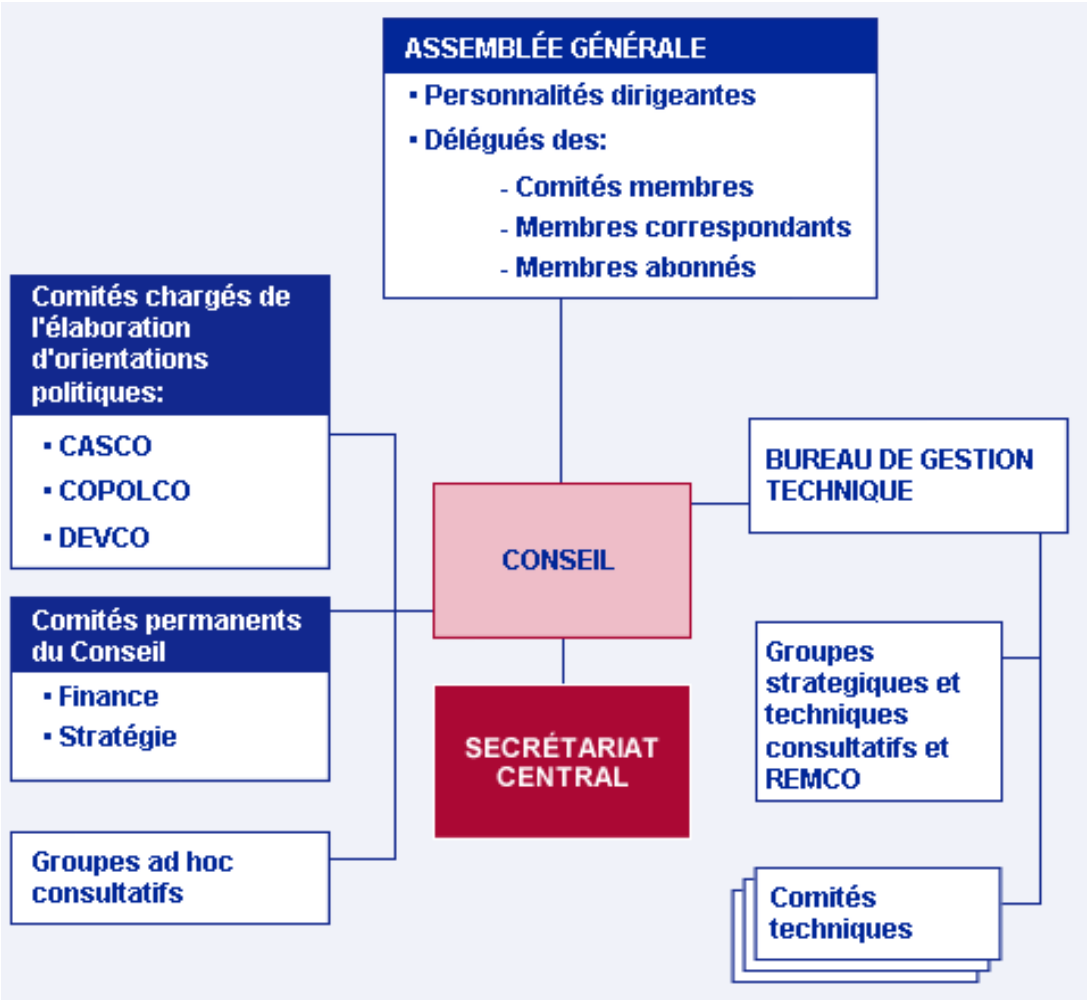


Figure 51. Structure de l'ISO

La structure de l'ISO présentée en Figure 51. Structure de l'ISO" est composée d'un conseil, un secrétariat général et de :

- Assemblée Générale
- Personnalités dirigeantes
- Membres de l'ISO
- Comités chargés de l'élaboration d'orientations politiques

- Comités permanents du Conseil
- Groupes ad hoc consultatifs
- Bureau de gestion technique
- Groupes stratégiques et techniques consultatifs et REMCO
- Comités techniques

I.2 Fonctionnement de l'ISO[ix]

Les orientations de l'ISO sont guidées par un Plan stratégique approuvé pour une période de cinq ans par les membres de l'ISO. Les membres de l'ISO, représentants ultimes de l'ISO pour leurs propres pays, sont divisés en trois catégories : les comités membres (membres à part entière), les membres correspondants et les membres abonnés. Seuls les comités membres ont le droit de vote.

L'Assemblée générale, qui a lieu chaque année, est une réunion des principaux dirigeants de l'ISO et des délégués nommés par les comités membres. Les membres correspondants et les membres abonnés peuvent assister à l'assemblée générale en qualité d'observateurs. Parmi les personnalités dirigeantes figurent notamment le Président, personnalité de premier plan de la normalisation ou de l'économie, le Vice-président (questions de politique), le Vice-président (gestion technique), le Trésorier et le Secrétaire général. L'ordre du jour de l'Assemblée générale comprend, entre autres, les actions liées au rapport annuel de l'ISO, le Plan stratégique et les incidences financières, et le rapport financier annuel du Trésorier sur le Secrétariat central de l'ISO.

Les Statuts de l'ISO stipulent que, si l'Assemblée générale est l'autorité ultime de l'organisation, la plupart des fonctions de gouvernance de l'ISO sont assurées par le Conseil, conformément à la politique établie par les comités membres.

Le Conseil se réunit deux fois par an et ses membres sont renouvelés par rotation de manière à assurer la représentativité de tous les membres de l'ISO.

Tous les comités membres sont éligibles en vue de l'élection/la nomination au Conseil. Différents comités chargés de l'élaboration d'orientations politiques relèvent du Conseil.

Ces comités, qui fournissent une assistance stratégique pour l'élaboration du travail des normes sur les aspects intersectoriels, sont notamment le CASCO (évaluation de la conformité); le COPOLCO (politique en matière de consommation), et le DEVCO (questions relatives aux pays en développement). Ces comités sont ouverts à tous les comités membres et membres correspondants.

Le Bureau de gestion technique (TMB) qui fait rapport au Conseil, est lui-même responsable de la gestion d'ensemble des travaux techniques, y compris d'un certain nombre de groupes consultatifs techniques et stratégiques. Les comités membres sont éligibles pour être nommés/élus au TMB selon un ensemble de critères établis par le Conseil.

Les opérations sont gérées par le Secrétaire général (principal fonctionnaire exécutif) qui fait rapport au Conseil. Le Secrétaire général siège au Secrétariat central de l'ISO à Genève (Suisse) avec un personnel restreint qui fournit le soutien administratif et technique aux membres de l'ISO, coordonne le programme décentralisé d'élaboration des normes et publie le résultat de ces travaux. Le Secrétariat central de l'ISO assure également le rôle de secrétariat des organes de gestion, des comités chargés de l'élaboration d'orientations politiques et de leurs organes subsidiaires.

Les Comités Techniques (CT)[x] sont chargés des travaux techniques.

Les CT qui peuvent impacter l'ingénierie du logiciel sont :

CT 10 Dessins Techniques

CT 20 Espace et véhicules aériens

CT 46 Information et Documentation

CT 145 Symboles Graphiques

CT 154 Documents et données dans l'administration, le commerce et l'industrie

CT 159 Ergonomie

CT 176 Management de la qualité et assurance qualité

CT 184 Systèmes d'Automatisation Industrielle

I.3 Qu'est-ce que l'IEC

La Commission électrotechnique internationale (CEI) ou International Electrotechnical Commission (IEC) en anglais, est l'organisation internationale de normalisation chargée des domaines de l'électricité, de l'électronique et des techniques connexes. Elle est complémentaire de l'Organisation internationale de normalisation (ISO), qui est chargée des autres domaines.

La CEI est composée de représentants de différents organismes de normalisation nationaux. La CEI a été créée en 1906 et compte actuellement 69 pays participants. Les normes CEI sont reconnues dans plus de 100 pays.

Originellement située à Londres, la Commission a rejoint ses quartiers généraux actuels de Genève en 1948. La CEI dispose de trois centres régionaux à Singapour, São Paulo et Worcester (Massachusetts).

La CEI a été l'instrument du développement et de la distribution de normes d'unités de mesure, notamment le gauss, l'hertz, et le weber. Elle contribua également à proposer un ensemble de références, le système Giorgi, qui finalement fut intégré au système international d'unités (SI) dont l'ISO est responsable. [xi]

Structure de l'IEC

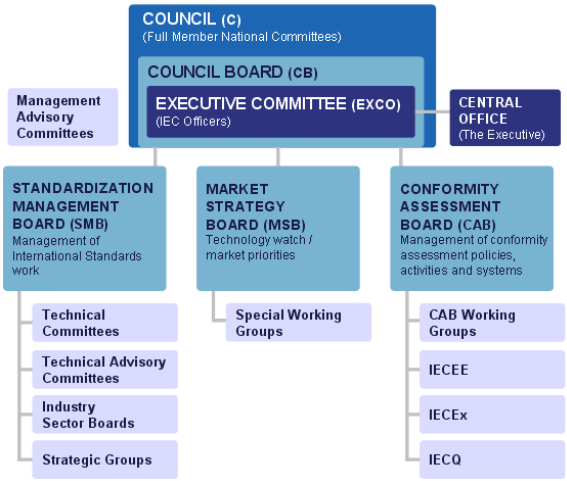


Figure 52. Structure de l'IEC

I.4 Fonctionnement de l'IEC

Les normes sont établies par des experts regroupés dans des groupes de travail internationaux et qui représentent les constructeurs, utilisateurs de matériels, ainsi que des représentants de laboratoires d'essais, des consultants et des universitaires spécialistes du domaine considéré.

Ces experts sont mandatés par leur comité national, ils sont chargés de rédiger des projets qui sont soumis au vote des comités nationaux CEI et qui sont adoptés comme normes internationales lorsque que la majorité de votes positifs requise est obtenue (en général une majorité des 2/3).

Les CT (comités techniques) sont chargés des travaux d'ordre technique. Les CT qui peuvent avoir une influence dans l'ingénierie du logiciel sont :

CT 45 Documentation Nucléaire

CT 56 Fiabilité et Maintenabilité

CT 65 Processus industriels de mesure / control.

I.5 ISO et IEC

Ces deux organismes se sont réunis en Joint Technical Committee, c'est en 1987 qu'est créé le JTC1 afin de devenir l'organe de référence de la normalisation des technologies de l'information au niveau mondial.

Comité technique commun à ses deux parents (JTC1 est un sigle dont le développement est Joint Technical Committee 1, Comité technique commun n° 1), il réunit les compétences de l'ancien TC97 de l'ISO relatif aux logiciels (langages de programmation, codage de l'information…) et les compétences de comités techniques de la CEI en matière de matériels : microprocesseurs, imprimantes, par exemple.

Les normes publiées par le JTC1, quoique appelées souvent normes ISO, sont reconnaissables par leur numéro de nomenclature qui commence par les deux sigles ISO/CEI. Le total de normes publiées par le JTC1 depuis sa création était de 1993 en 2006. Le comité comporte 28 pays membres et 44 observateurs.

I.6 Structure du JTC1

Le JTC1 est composé des comités techniques suivants (considérés par les deux parents comme des sous-comités, d'où leur sigle en SC) :

SC1 – Vocabulaire

SC2 – Tables de caractères et codage de l'information

SC3 – Télécommunication et échange d'information entre systèmes

SC7 – Ingénierie du Logiciel

SC11 – Media magnétiques flexibles pour échange de données digitales

SC14 – Représentation des éléments de données

SC15 – Etiquetage et structures des fichiers

SC17 – Cartes d'identification et services associés

SC18 – Traitement des documents et communications reliées

SC21 – Récupération d'informations, transfert et management pur OSI.

SC22 – Langages de programmation, leur environnement et système d'interfaces logiciel

SC23 – Cartouches de disques optiques pour l'échange d'informations

SC24 – Imagerie informatique et traitement de l'image

SC25 – Interconnexion des équipements des technologies de l'information

SC26 – Systèmes à microprocesseurs

SC27 – Techniques de sécurité des technologies de l'information

SC28 – Equipement de bureaux

SC29 – Représentation codée des images, audio et information multimédia / Hypermedia

SC31 – Techniques d'identification et de captage automatique des données

SC32 – Gestion et échange de données informatisées

SC34 – Langages de description et de traitement de documents

SC35 – Interfaces d'utilisateurs

SC36 – Informatique pour l'éducation, la formation et l'apprentissage

SC37 – Biométrie

Le sous-comité n°7 ou SC7 [xii] est celui qui nous intéresse dans le cas de notre panorama des normes liées à l'ingénierie logicielle.

Le SC7 est chargé de l'ingénierie et de la qualité des logiciels et systèmes, il couvre l'ensemble des normes sur les méthodes, processus et services sur la totalité du périmètre de l'informatique. De nombreuses normes concernent le cycle de vie logiciel, systèmes, les tests, les services IT (série ISO/IEC 20000).

Le SC7 est composé de dix groupes de travail, tels que :

Group de Travail n°2 : Système de documentation

Group de Travail n°4

Outils et environnement

Group de Travail n°6

Evaluation et méthodes de mesure

Group de Travail n°7

Gestion du cycle de vie

Group de Travail n°8

Support des processus de cycle de vie

Group de Travail n°9

Intégrité logicielle

Group de Travail n°10

Recensement des processus

Group de Travail n°11

Définition des données de l'ingénierie logicielle

Group de Travail n°12

Mesure des tailles des fonctions

Group de Travail n°13

Mesure des processus logiciels.

I.7 ISO 12207 outil complémentaire du management

ISO 12207 est complémentaire au management institutionnalisé.

ISO 12207 donne un ensemble puissant, complet mais flexible de block de construction du cycle de vie logiciel pour projets et organisations afin de l'utiliser de façon pertinente et efficace.

I.8 Ce que ISO 12207 n'est pas

Prescription : ne dit pas comment faire

- Adapté aux évolutions technologiques
- N'interfère pas avec les décisions technologiques
- Un substitut d'un management systématique ou discipliné

Standard pour méthodes, techniques et modèles

- Ne prescrit pas de méthodes de management et d'ingénierie
- Ne prescrit pas de langage de programmation
- Ne prescrit pas d'environnement d'ingénierie logicielle

- Ne prescrit pas modèles de cycle de vie ou de développement

Standard pour produits

- Requière de la documentation de la production de groupes spécifiques
- Mais prescrit l'interdiction de formats, contenu explicit ou media
- Un produit d'une organisation standard utilisable avec 12207

Standard pour les mesures

- Beaucoup de tâches ont besoin de mesures et d'indicateurs
- Mais prescrit l'utilisation de mesures / indicateurs non spécifiques
- Utiliser ISO / IEC 9126 pour se guider.

I.8 Architecture du cycle de vie

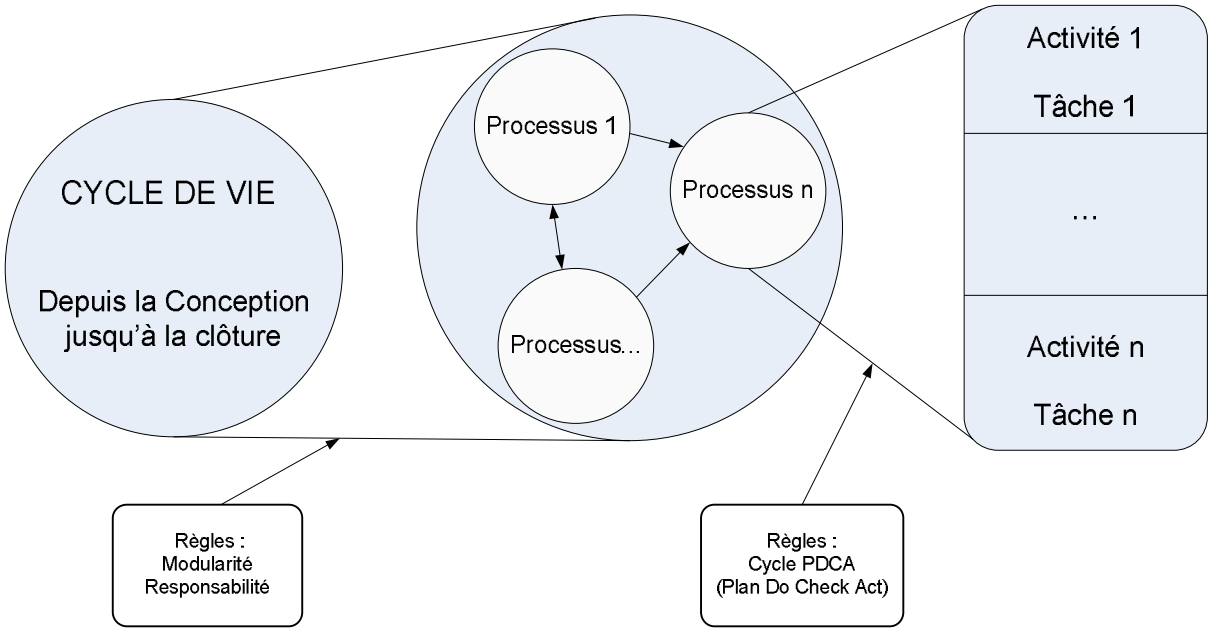


Figure 53. Architecture du cycle de vie

I.10 Règles pour le partitionnement du cycle de vie

Modularité

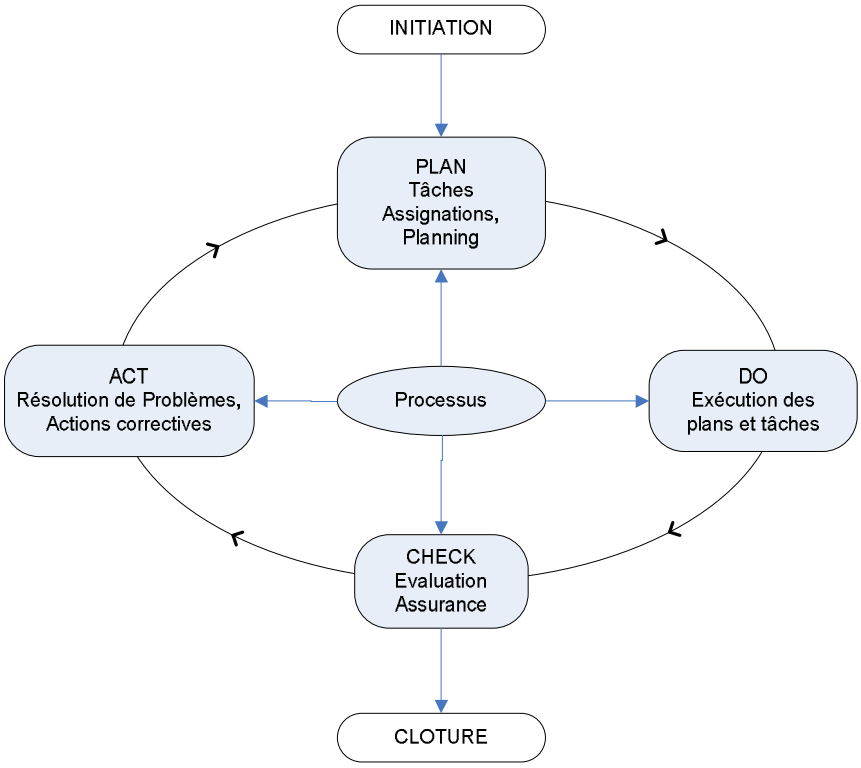
- Cohésion (Fonctionnelle): les tâches d'un processus sont fonctionnellement liées
- Couplage (Interne) : les liens parmi les processus sont minimes
- Associations :
    - Si une fonction est utilisée par plus d'un processus, alors la fonction devient un processus.
    - Si le processus X est invoqué par le processus A et seulement le processus A, alors le processus X appartient au processus A
    - Exception : seulement pour des explications futures

Responsabilité

- Chaque processus est sous une responsabilité
- Une fonction dont les composantes sont sous différentes responsabilités ne devrait pas être un processus.
- Mettre en contraste avec un processus pour un sujet monolithique exemple : Management de la qualité

A noter que le cycle de vie lui-même n'a pas été partitionné dans le temps car le cycle de vie logiciel suit le cycle de vie du système parent.

Partitionnement d'un processus en cycle activités PDCA[xiii]



I.11 Contenu formation UML

Chapitre 1 Introduction à la modélisation objet

1.1 Le génie logiciel

1.1.1 L'informatisation

1.1.2 Les logiciels

1.1.3 Le génie logiciel

1.1.4 Notion de qualité pour un logiciel

1.2 Modélisation, cycles de vie et méthodes

1.2.1 Pourquoi et comment modéliser ?

1.2.2 Le cycle de vie d'un logiciel

1.2.3 Modèles de cycles de vie d'un logiciel

---

[i] Standardisation : http://fr.wikipedia.org/wiki/Standardisation

[ii] Standardisation : http://fr.wikipedia.org/wiki/Standardisation

[iii] Normalisation : http://fr.wikipedia.org/wiki/Normalisation_%28industrie_et_service%29

[iv] Practical Software Maintenance: Best Practices for Managing Your Software Investment, Thomas M. Pigoski, John Wiley & Sons, Inc. New York, NY, USA ©1996

[v] Lehman, M. M.; J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski (1997). "Metrics and laws of software evolution—the nineties view".

[vi] D'après le site officiel du S3M: http://www.s3m.ca/fr/aboutUs/faq.html

[vii] CMMi : Sigle de Capability Maturity Model + Integration

CMMi, sigle de Capability Maturity Model + Integration, c'est un modèle de référence, un ensemble structuré de bonnes pratiques, destiné à appréhender, évaluer et améliorer les activités des entreprises d'ingénierie.

CMMi a été développé par le Software Engineering Institute de l'université Carnegie Mellon, initialement pour appréhender et mesurer la qualité des services rendus par les fournisseurs de logiciels informatiques du Département de la Défense US (DoD).

[viii] ITIL: Information Technology Infrastructure Library.

[ix] Fonctionnement : http://www.iso.org/iso/fr/about/governance_and_operations.htm

[x] Comités Techniques : http://www.iso.org/iso/fr/standards_development/technical_committees/list_of_iso_technical_committees.htm?sector_filter=

[xi] IEC : http://fr.wikipedia.org/wiki/Commission_%C3%A9lectrotechnique_internationale

[xii] SC7 : http://www.jtc1-sc7.org/

[xiii] PDCA : Plan Do Act Check