Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique.

Re-ingénierie d'un système d'aide à la décision : Du raisonnement hybride au raisonnement par règle

par

Raja DALLAPE

Préliminaire du Mémoire présenté pour l'obtention du diplôme de

Maître en informatique

Année académique 2005-2006

Table des matières

Chapitre 1 : Introduction

1.1 Historique

La recherche en génie logiciel a permis de mettre à jour la méthode fonctionnelle COSMIC-FFP (COSMIC étant l'acronyme de Common Software Measurement International Consortium, FFP celui de Full Functional Point). COSMIC-FFP vise à améliorer la mesure des logiciels et est devenue en décembre 2002 une norme ISO/IEC avec pour numéro : 19761. Ensuite, un système à base de connaissances pour l'apprentissage de la méthode est né. Le projet de création du logiciel COSMICXpert venait de voir le jour. La première version de COSMICXpert a été développée par Tim Küssing, un étudiant allemand, lors de son stage à Montréal avec le professeur Desharnais. Mais cette première version présentait deux limites :

- le logiciel était en mode local, ce qui veut dire qu'il était difficilement accessible aux mesureurs
- les données ne pouvaient pas être validées facilement car un grand nombre de fichiers au format texte devaient être créés pour mémoriser que très peu d'information de la base de connaissances.

Une seconde version a été créée pour permettre une accessibilité, du logiciel, à plus grande échelle, ainsi qu'un système de mémorisation de données plus efficace. C'est le fruit du travail de François Gruselin et Julien Vilz, en automne 2002. Messieurs F. Gruselin et J.Vilz qui ont proposé une solution WEB, pour rendre le logiciel accessible à un plus grand nombre d'utilisateur. Ensuite le logiciel utilisait des fichiers XML, pour réduire le nombre de fichier servant à la mémorisation.
La seconde version de messieurs F. Gruselin et J. Vilz assurait déjà tous les besoins des mesureurs, il restait encore à rendre possible l'évolution de la base des connaissances.
Ce sont les stagiaires Christophe Duterme et Nicolas Fabry qui ont terminé la deuxième version du logiciel COSMICXpert en automne 2003. Ils ont également ajouté un mécanisme permettant de gérer la concurrence lors de l'utilisation des fonctionnalités d'évolutions, ou modification de la base des connaissances. L'intégrité de la base des connaissances était ainsi mieux préservée.
COSMICXpert était enfin complet et accessible depuis l'Internet. Mais le projet s'étant étalé sur plusieurs années, la documentation et le code source étaient vraiment trop hétérogènes, il n'y avait plus vraiment d'architecture calquée sur un modèle précis. Enfin beaucoup d'erreurs subsistaient.
En 2004, les stagiaires Stéphane Sandron et Benoît Vanderose ont tout d'abord refait la documentation de COSMICXpert pour la rendre plus homogène, ensuite ils ont donné une architecture de type « Modèle - Vue - Contrôleur » et finalement ils ont corrigé la majeure partie des erreurs du logiciel COSMICXpert.
Il reste néanmoins encore beaucoup à faire. Toutes les erreurs n'ont pas été corrigées, et les interfaces sont peu intuitives, elles ont été modifiées lorsque des ajouts ont été faits, mais n'ont jamais réellement fait l'objet d'une réflexion.
Enfin, les stagiaires Sandron et Vanderose ont, à partir du logiciel COSMICXpert, appliqué un processus de restructuration pour donner naissance à SM^{Xpert}. SM étant l'acronyme de Software Maintenance, le nom du logiciel indique presque déjà son utilité. En effet, son

domaine d’application est la maintenance de logiciel, et il permettra d’aider un utilisateur à évaluer la maturité d’un processus de maintenance et de résoudre les problèmes de celui-ci.

1.2 Contexte

Au fil du temps, les connaissances de l'homme s'accumulent de plus en plus et dans des domaines de plus en plus variés, à tel point qu'un humain seul ne puisse plus tout connaître. Il est donc souhaitable de mettre la machine au service de l'homme.
De nos jours l'informatique est de plus en plus présente dans la société. Cette informatisation de la société vient aussi avec le développement des réseaux. La combinaison de ces technologies fournit un outil puissant au service de l'homme. Un système à base de connaissances accessible via l'Internet peut aider l'humain pour une prise de décision.
La maintenance de logiciel est un domaine devenant de plus en plus vaste et de plus en plus présent en industrie. La maintenance n'est plus à son balbutiement, elle possède son propre processus et sa propre gestion stratégique afin de corriger des erreurs ou apporter des améliorations pour que les logiciels continuent à répondre de la manière la plus efficiente possible aux besoins de ses utilisateurs.
Il est donc intéressant de créer un système à base de connaissance pour aider les ingénieurs en maintenance dans leurs activités. SM^{Xpert} est un logiciel ayant ce but. Il basé sur les travaux du professeur Alain April de l'École de Technologie Supérieure (ÉTS) de Montréal. Dans le cadre d'une recherche doctorale, le professeur a proposé, sur base d'une liste de problèmes liés à la maintenance logicielle, un modèle de la maturité des processus nommé « Software Maintenance Maturity Model » (S^{3m}). Il permet d'évaluer et améliorer le processus de maintenance.
Le logiciel SM^{Xpert} est un logiciel, créé à partir d'une restructuration d'un autre logiciel (le logiciel COSMICXpert), ayant aussi pour but l'aide à la décision. Les principaux problèmes de ces logiciels sont que leurs interfaces sont très complexes pour l'expert désirant entrer ou modifier des informations dans la base de connaissance. Certains utilisateurs, dont des experts en maintenance, révèlent qu'il est impossible d'entrer des informations, ou en tout cas qu'il y sont parvenus en contournant les interfaces, c'est à dire en accédant directement aux fichiers XML dans lesquels la connaissance est mémorisée. Il est donc nécessaire d'améliorer les interfaces.
Ce projet a pour objectif principal de proposer des solutions d'amélioration des interfaces du logiciel SM^{Xpert}.

1.3 Problématique

L'interface actuelle du logiciel SM^{Xpert} est très complexe, le problème existait déjà pour le logiciel COSMICXpert dont il provient suite à un travail de restructuration. Les utilisateurs ne parviennent pas à utiliser correctement les fonctionnalités d'évolution ou de modification de la base des connaissances.
Comme nous l'avons vu les interfaces sont une partie cruciale pour la facilité d'utilisation d'un logiciel. La problématique étudiée dans ce document est l'amélioration des interfaces d'un système expert dont le domaine de connaissance est la maintenance de logiciel.

Poser le problème des interfaces, suppose d'autres questions :

1. Est-ce que des erreurs du logiciel n'empêchent pas la compréhension des interfaces ?
2. Est-ce que le logiciel reflète-t-il réellement ce que veulent les utilisateurs ?

1.4 Objectif

Il est proposé dans le cadre de ce projet de recherche d'améliorer l'interface utilisateur de type

« expert » du logiciel *SM*^{Xpert}. Tout d'abord une familiarisation avec cette interface est planifiée afin d'avoir une idée des problèmes qui se posent aux utilisateurs. Cette familiarisation a commencé par une première tâche qui consistait à entrer un cas problème dans la base de connaissance du logiciel *SM*^{Xpert}. La réussite de cette tâche a pris plusieurs essais. Les premiers essais consistaient à entrer de cas fictifs, et ensuite l'entrée de cas de maintenance réels.
Une fois cette étape réussie, il est intéressant d'évaluer la navigation dans la base de connaissance, grâce à l'interface prévue pour les utilisateurs. Cette familiarisation avec l'interface utilisateur a permis de faire un définition du projet à réaliser comme indiqué au premier tableau (Définition du projet) ci-dessous.
Une fois la définition du projet faite, ce dernier a pu débuter. Comme indiqué sur le deuxième tableaux ci-dessous (Planification du projet) le projet a débuté par une revue de littérature sur les interfaces permettant de faire une critique de l'interface actuelle du logiciel *SM*^{Xpert} et de définir une nouvelle solution. Après la définition de la nouvelle solution, il a fallut l'appliquer. C'est ce qui est expliqué en détail sur le troisième tableau ci-dessous (Exécution).
Le quatrième tableau ci-dessous (Interprétation) explique comment l'évaluation du nouveau logiciel sera faite.

1.4.1 Définition du projet

Motivation	Objet	Objectif	Utilisateurs
Améliorer l'interface du système d'aide à la décision *SM*^{Xpert}.	Identifier à partir des références pertinentes du domaine des interfaces utilisateurs de logiciel, les forces et les faiblesses de l'interface, des utilisateurs de type « expert », actuelle d'un logiciel d'aide à la décision (le logiciel *SM*^{Xpert}).	- Améliorer l'interface des utilisateurs de type « expert » actuelle. - Améliorer le modèle conceptuel du logiciel. - Améliorer la technique actuelle de gestion des données.	- Les personnes intéressées par le domaine des interfaces du logiciel. - Les experts de la maintenance. - Les utilisateurs de la maintenance - Les chercheurs en maintenance.

1.4.2 Planification du projet

Trois étapes ont été planifiées. Chaque étape se divise en 3.

Etapes	Intrants	Livrables
1 - Identifier dans les références pertinentes du domaine des interfaces utilisateurs de logiciel, l'information nécessaire à la conception d'interface de logiciel d'aide à la décision.	- Le modèle de maturité S^{3m}. Le logiciel *SM*^{Xpert} existant - Différentes publications scientifiques pertinentes. - Les parties 10 à 12 de la norme ISO 9241	- État de l'art sur les meilleures pratiques d'interfaces.
- Faire un sommaire de l'état de l'art, concernant les bonnes pratiques à utiliser pour concevoir des interfaces de logiciels		- Liste des principaux problèmes existants de l'interface des

d'aide à la décision.		utilisateurs de type « expert » du logiciel *SM*^{Xpert}.
	- Le cadre de référence pour l'amélioration de la qualité d'un logiciel d'aide à la décision (application à COSMICXpert et *SM*^{Xpert}.) développé par messieurs Stéphane Sandron et Benoit Vanderose.	- Proposition d'une nouvelle conception (uses cases, modèle de données, diagramme de classes, schéma entités-associations).
2 - Effectuer l'analyse des différents problèmes de l'interface actuelle du logiciel *SM*^{Xpert}, suivie d'une évaluation des impacts des changements du logiciel et de la technique actuelle de gestion de données.		- Étude de l'impact des changements sur le logiciel.
		- Choix des technologies recommandées pour satisfaire les exigences de la nouvelle version de *SM*^{Xpert}.
3 - Proposition d'une liste de modifications et d'ajouts pour résoudre les problèmes identifiés lors de l'étape 2.		

1.4.3 Exécution du projet

Préparation	Exécution	Analyse
Déploiement des technologies recommandées pour satisfaire les exigences de la nouvelle version de *SM*^{Xpert}.	Implémentation du système de gestion des utilisateurs de *SM*^{Xpert}.	Un logiciel où l'on peut enregistrer son profil en tant qu'utilisateur de la maintenance à des niveaux différents, expert de la maintenance. Ainsi que s'enregistrer en tant qu'administrateur.
Création de la base de données ainsi que du patron d'accès permettant la communication avec cette dernière.	Implémentation des fonctionnalités permettant de naviguer dans la base de connaissance.	Un logiciel permettant d'aide à la décision, avec gestion de ses utilisateurs.
	Implémentation des fonctionnalité permettant d'accroître/changer la base de connaissance de *SM*^{Xpert}.	Un logiciel permettant l'aide à la décision, avec gestion de ses utilisateurs et pouvant évoluer au niveau de sa connaissance.

	Réalisation de l'interface du système de gestion des utilisateurs. Réalisation de l'interface permettant d'accroître/changer la base de connaissance de SM^{Xpert}. Réalisation de l'interface de navigation dans la base de connaissance.	

1.5 Revue de littérature

Cette revue de littérature contient les références nécessaires dans le cadre de ce projet ayant pour but initial de changer les interfaces du logiciel SM^{Xpert}. Nous allons donc voir les meilleures pratiques d'interface. Mais suite à la familiarisation avec le logiciel, le but du projet a été redéfini. Le projet de consistait plus uniquement à un changement d'interfaces, mais aussi une correction des problèmes détectés, et le déploiement d'un système de gestion de base de données. Ces changements additifs ont provoqué non pas une restructuration, mais une ré-ingénierie du logiciel SM^{Xpert} jusque dans sa technique de raisonnement, passant d'un raisonnement hybride vers un unique raisonnement par règles. C'est pourquoi les références en matière de systèmes experts, système d'aide à la décision, ré-ingénierie et restructuration sont également présentées.

1.5.1 Les exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation

1.5.1.1 Introduction

Les exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation sont une source d'information importante dans le cadre de ce projet, car elles permettront dans un premier temps d'évaluer une interface et par la suite de servir d'outil pour améliorer cette même interface. Avant de voir les exigences proprement dites, il est tout d'abord intéressant d'expliquer les principes généraux.
Les principes généraux d'ergonomie seront donc tout d'abord présentés, suivis des règles d'ergonomie découlant de ces principes et permettant à un utilisateur de se familiariser rapidement au dialogue avec une interface.
Ensuite, nous verrons les recommandations relatives à la présentation de l'information qui fournissent afin d'organiser l'information de concevoir des objets graphiques suivant les attentes des utilisateurs.
Tous ces principes, recommandations, et règles peuvent également être appliqués à l'occasion de la spécification, du développement ou de l'évaluation d'un logiciel.

1.5.1.2 Principes d'ergonomie

Ces principes ont pour but de rendre une interface plus adaptée à l'utilisateur, de réduire la charge cognitive et minimiser l'effort de mémorisation. Les principes ont encore deux autres buts, mais plus vastes encore. Ils ont pour but aussi de minimiser le multitâches et permettre

d'avoir un modèle mental clair d'une application.
Ces principes doivent prendre en compte les caractéristiques de l'utilisateur tels que : la capacité de concentration, les limites de la mémoire à court terme, les comportements d'apprentissages, le degré d'expérience dans l'activité du logiciel, la compréhension par l'utilisateur de la structure sous-jacente et du but du système avec lequel il va dialoguer.
Mais ces principes ne sont pas indépendants il est même probable de devoir faire un compromis entre avantages et inconvénients.
Enfin il faut prendre aussi en compte le champ d'application spécifique qui sont : les buts de l'organisation, des besoins du groupe (final) d'utilisateur concernés, les tâches que l'application doit permettre de réaliser et les technologies et ressources disponibles.

Voici la liste des principes [8] et [9] :

Adaptation de la tâche : « Un dialogue est considéré comme adéquat, pour une tâche, lorsqu'il permet à l'utilisateur de réaliser cette tâche de façon efficace et efficiente » [9].
Caractère auto descriptif : « Un dialogue est auto descriptif lorsque chaque étape du dialogue est immédiatement compréhensible grâce au retour d'information du système, ou est expliquée à l'utilisateur à sa demande » [9]. Il faut aussi informer l'utilisateur sur l'état d'un système. C'est-à-dire lui indiquer qu'une action risque de prendre du temps, lui indiquer si des données entrées ont été bien insérées, etc.
Contrôle explicite : « Ce principe signifie que même si c'est le logiciel qui a le contrôle, l'interface doit apparaître comme étant sous le contrôle de l'utilisateur et surtout exécuter des opérations uniquement à la suite d'actions explicites de l'utilisateur » [8].
Conformité aux attentes de l'utilisateur : « Un dialogue est conformes aux attentes de l'utilisateur lorsqu'il est cohérent et correspond aux caractéristiques de l'utilisateur, telles que la connaissance de la tâche, la formation, l'expérience et les conventions communément admises » [9].
Facilité à l'apprentissage : « Un dialogue permet l'apprentissage lorsqu'il soutient et guide l'utilisateur dans l'apprentissage de l'utilisation du système » [9].
Gestion des erreurs : « Un dialogue est tolérant aux erreurs si, malgré des erreurs d'entrée évidentes, le résultat prévu peut être obtenu soit sans action corrective, soit avec une action corrective minimale de la part de l'utilisateur » [9].
Flexibilité : « Un dialogue permet l'individualisation lorsque l'interface logicielle peut-être modifiée pour s'adapter aux besoins de la tâche, aux préférences individuelles et aux compétences de l'utilisateur » [9]. Une application doit pouvoir s'adapter à l'évolution cognitive d'un utilisateur et également à divers utilisateurs.
Cohérence : Ce qui s'affiche sur un écran doit être cohérent avec le monde réel, et le vocabulaire doit être le vocabulaire usuel d'un utilisateur. Exemple : si une application consiste à introduire les données d'une personne, il est préférable que ce qui s'affiche à l'écran soit directement perçu comme un formulaire. Il doit aussi y avoir une cohérence entre les données affichées à l'écran. Exemple : une uniformité dans les boutons de commande.
Concision : Eviter les manipulations trop longues afin de ne pas fatiguer un utilisateur. Exemple : trop d'étapes dans une procédure par exemple. Dans ce cas l'utilisation de valeur par défaut ou de macros est intéressante.

A règles d'ergonomie

Voici à présent les règles proprement dites, chacune des règles sera présentée en rapport avec le principe qu'elle tend à faire respecter. Il faut néanmoins préciser que certains principes sont suffisamment explicites pour ne pas nécessiter de règle.

Il existe de nombreux règles d'ergonomie, plus de 3000 [8]. Elles ne sont évidemment pas toutes applicables à chaque application ou utilisateur. Nous allons énoncer les règles permettant de souligner les points faibles, en matière de simplicité et d'intuitivité, d'une interface.

A.1 Adaptation de la tâche

Une interface permettant d'accomplir une tâche ne doit présenter que l'information permettant de l'exécuter.
Il faut que l'utilisateur puisse toujours avoir accès à un menu d'aide expliquant la tâche qu'il désire exécuter.
Toutes actions pouvant être automatisée doit être faite par le logiciel, il convient aussi de prendre en compte la complexité de la tâche (ex : utiliser un menu offrant les choix possibles lorsqu'il y a un ensemble d'entrées d'alternatives). Si il existe des possibilités de données d'entrée par défaut, l'utilisateur ne doit pas être obligé d'entrer ces valeurs.

A.2 Caractère auto descriptif

Si nécessaire, le logiciel se de doit de fournir un compte rendu, aux actions de l'utilisateur, qui se réfèrent strictement à la situation pour laquelle ils sont nécessaires. Et « si des conséquences graves peuvent résulter d'une de ses actions, une explication doit lui être fournie ainsi qu'une demande de confirmation » [9].
Ce compte rendu et ces explications doivent être adaptés au niveau de connaissance de l'utilisateur et lui permettre d'acquérir une compréhension globale du logiciel.
Le compte rendu et les explications se doivent d'être plus ou moins détaillées selon les besoins du l'utilisateur (exemple : en appuyant une fois sur la touche « Aide », l'utilisateur obtient une brève explication ; en appuyant deux fois il obtient une explication détaillée).
Lorsqu'il faut entrer des données, il convient que le logiciel indique à l'utilisateur la nature des données à entrer.
Il faut aussi informer l'utilisateur sur l'état d'un système. C'est-à-dire lui indiquer qu'une action risque de prendre du temps

A.3 Contrôle explicite

L'utilisateur doit donc contrôler le déroulement du dialogue, « si le dialogue a été interrompu il convient que l'utilisateur ait la possibilité de déterminer à quel endroit le reprendre chaque fois que la tâche le permet » [9].
L'utilisateur doit pouvoir annuler au moins la dernière étape d'un dialogue.
Enfin les méthodes d'interaction doivent être adaptées aux besoins et aux caractéristiques de l'utilisateur.

A.4 Facilité d'apprentissage

« Il convient que l'utilisateur dispose des règles et concepts sous-jacents utiles à l'apprentissage, afin de pouvoir constituer ses propres stratégies de regroupement et ses propres règles de mémorisation » [9].

A.5 Tolérance aux erreurs

L'application doit assister l'utilisateur dans la prévention et la détection d'erreurs d'entrée. Le système doit aussi empêcher qu'une erreur d'entrée provoque des résultats non prédéfinis et des défaillances du système.

« Dans le cas où le système peut corriger des erreurs automatiquement, il convient que le système avertisse l'utilisateur de l'exécution des corrections et qu'il laisse la possibilité d'ignorer les corrections » [9].

A.6 Flexibilité

Il convient que le logiciel permette à l'utilisateur de choisir entre différentes formes de représentation, selon ses préférences et la complexité de l'information à traiter.

1.5.1.3 Recommandations relatives à la présentation de l'information

La présentation de l'information est aussi importante que le dialogue, car il le facilite. L'un ne va pas sans l'autre. L'information doit présenter certaines caractéristiques permettant un maximum de tâches de perception de manière efficace, effective et satisfaisante.

Voici la liste de ces caractéristiques [10] :
- clarté : « le contenu de l'information est transmis rapidement et avec précision ».
- Discernabilité : « l'information affichée peut-être distinguée avec précision ».
- Concision : « les utilisateurs reçoivent uniquement les informations nécessaires à l'exécution de la tâche »
- Cohérence : « la même information est présentée de manière identique sur toute l'application, en toute conformité avec les attentes de l'utilisateur »
- Détectabilité : « l'attention de l'utilisateur est dirigée vers l'information demandée »
- Lisibilité : « information facile à lire »
- Compréhensibilité : « le sens est clairement compréhensible, sans ambiguïté, interprétable et reconnaissable » [10].

A Organisation de l'information.

C'est en organisant l'information qu'il est possible de respecter les caractéristiques présentées ci-dessus. L'organisation de l'information est facilitée par l'utilisation d'objets graphiques. Voici des recommandations relatives aux objets graphiques afin de respecter les caractéristiques nécessaires de l'information.

A.1 Les fenêtres

L'utilisation de fenêtres est plus appropriée lorsque :

- l'utilisateur surveille ou accède à plusieurs systèmes, applications ou processus simultanément
- l'utilisateur évalue, compare ou manipule plusieurs sources d'information ou plusieurs vues d'une source unique d'information.
- L'utilisateur travaille souvent alternativement entre des tâches, des systèmes, des applications, des dossiers, des sections ou des vues.
- l'utilisateur doit préserver le contexte d'une tâche plus vaste tout en effectuant des sous-tâches individuelles
- l'utilisateur doit prendre en charge des événements d'un système ou d'une application avant que les opérations de tâche primaire ne puisse continuer
- l'utilisateur doit avoir accès de façon occasionnelle à des composants de dialogue supplémentaires (ex : menu), situés à proximité du point de l'écran où se concentre l'activité en cours de l'utilisateur.

Lorsque plusieurs fenêtres sont utilisées il faut fournir une identification unique (ex : un nom de fenêtre), concevoir les positions et les dimensions des fenêtres par défaut de manière à minimiser le nombre d'opération que les utilisateurs doivent exécuter pour accomplir une tâche.
Lorsqu'une relation de subordination intervient entre fenêtres, les relations entre fenêtres principales et ses secondaires doivent toujours être visuellement apparentes.
« Il convient de pouvoir discerner visuellement les éléments de commandes de fenêtres exécutant différentes fonctions, et de les placer de manière cohérente et au même endroit dans chaque fenêtre » [10].
« Si cela est approprié à la tâche, il convient d'autoriser les utilisateurs à choisir leur format de fenêtrage préféré et à le sauvegarder comme format par défaut » [10].

A.2 Zones

« Les zones sont à placer de manière homogène, tout au cours du dialogue dans une application » [10]. « Le densité de l'information affichée doit être telle qu'elle ne puisse pas être perçue par l'utilisateur comme trop encombrée » [10]. Et si l'utilisateur doit discerner des relations entre des ensembles d'information affichés séparément, il est souhaitable d'afficher les deux ensembles d'information sur un seul écran.
Quant aux zones d'entrée/sortie, il convient, dans la mesure du possible, d'afficher dans la zone d'entrée/sortie toutes les informations exigées pour exécuter une tâche donnée. Si ce n'est pas possible, il convient :

1. de structurer les informations requises en sous-ensemble correspondant aux étapes des tâches ;
2. que ces sous-ensembles prennent en charge les sous-tâches appropriées et soient significatifs pour les utilisateurs prévus, et
3. que le partage de l'information n'entraîne aucune réduction des performances de la tâche.

A.3 Groupes

Le regroupement des informations sur l'écran aide l'utilisateur à percevoir, trouver et interpréter/comprendre plus facilement l'information.
« Il convient que les groupes soient perçus comme distincts de par l'espacement et l'emplacement » [10].

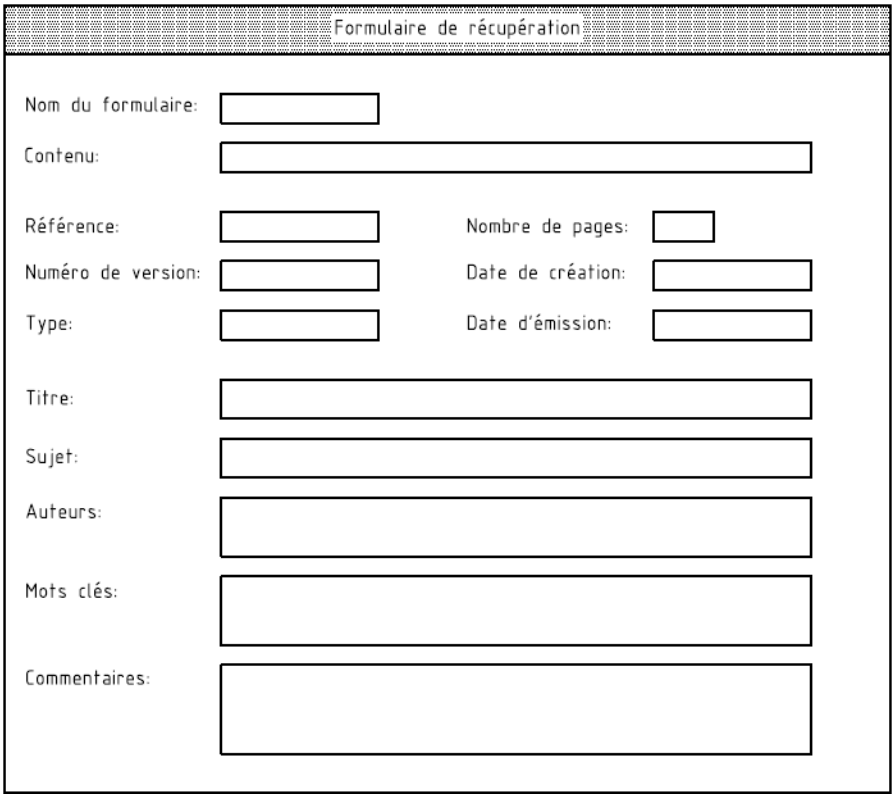

Figure 1 : exemple de regroupement [10]

Les lois suivantes facilitant l'utilisation de groupe.
Loi de rapprochement : les éléments très rapprochés dans l'espace sont perçus comme appartenant les uns aux autres.

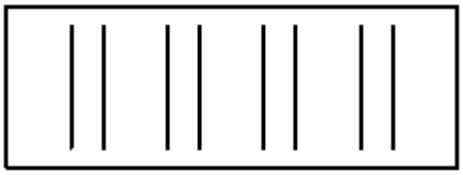

Figure 2 : illustration de la loi de rapprochement [8] et [10]

Loi de similarité : les éléments sont perçus comme appartenant les uns aux autres s'ils sont similaires. Dans l'exemple illustré l'observateur perçoit des colonnes et non des lignes.

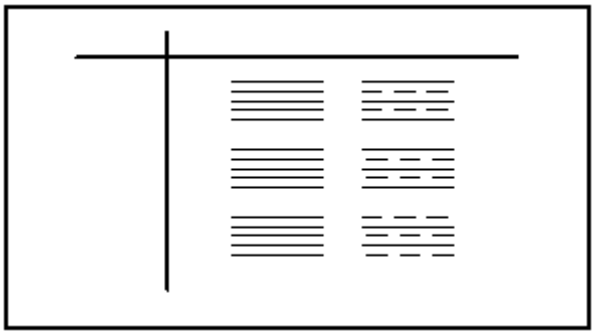

Figure 3 : illustration de la loi de similarité [8] et [10]

Loi de fermeture : les parties non existantes de la figure sont ajoutées ou les figures incomplètes sont automatiquement complétées. C'est le cas lorsque tous les groupes d'informations séparés dans l'espace et où l'observateur tente de créer une figure cohérente.

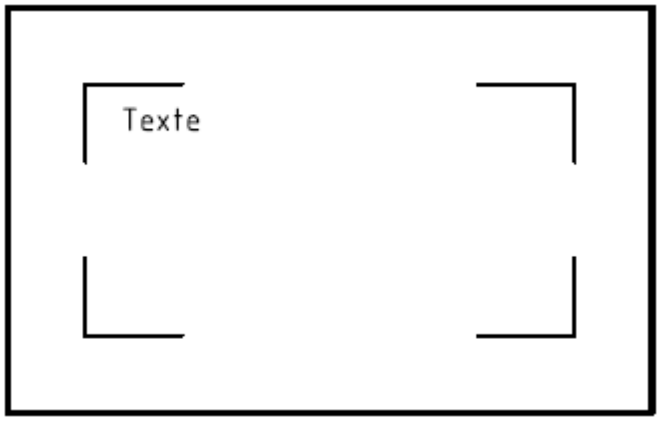

Figure 4 : illustration de la loi de fermeture [8] et [10]

A.4 Les listes

Utilisée aussi pour l'organisation de l'information, les listes doivent être organisées selon un ordre logique ou naturel adapté à la tâche (si pas possible, l'ordre alphabétique peut être envisagé).
« Les éléments et les groupes d'éléments d'une liste doivent être visuellement distincts afin de faciliter le balayage visuel » [10].
Le format des listes d'informations alphabétiques doit dépendre des conventions linguistiques (ex : justifier à gauche les listes verticales d'informations alphabétiques pour les langues se lisant de gauche à droite). Pour les informations numériques il est préférable de justifier à droite celles sans signe décimal (virgule ou point). Pour celle contenant des signes décimaux, il est préférable de les aligner par rapport au signe décimal.
Dans les listes numériques, il convient d'utiliser une taille de police fixe avec un espacement constant.
« Si une liste s'étend au-delà de la zone d'affichage disponible, il convient de fournir une indication de suite de liste » [10].

A.5 Les tableaux

Utilisés pour l'organisation d'informations dans des sous-ensembles visuels ayant un sens.
« La disposition des informations dans les tableaux doit être telle que les données les plus pertinentes pour les utilisateurs ou revêtant la plus grande priorité soient affichés dans le colonne la plus à gauche et que les données associées, moins significatives, figurent dans des colonnes situées plus à droite » [10].
« Si un tableau utilise des titres de colonnes et de rangées et qu'il s'étend au delà de l'affichage disponible, alors les titres associés aux colonnes et/ou aux lignes visibles restent également visibles » [10].

A.6 Les labels

Utilisés pour indiqués la signification des éléments d'information, lorsque le sens d'un élément d'écran n'est pas évident et clairement compréhensible pour les utilisateurs prévus.
Les labels doivent suivre une cohérence grammaticale (ex : utilisation cohérente de combinaisons nom-verbe).
L'emplacement du label se doit aussi d'être homogène, à proximité de l'élément d'information désigné. Leur format et leur alignement doivent aussi être homogène.

A.7 Les champs

« Il convient tout d'abord de bien faire la distinction visuelle entre les champs de saisie et les champs en lecture seule (ex : couleur, format, …). Si la tâche l'exige il convient de pourvoir distinguer l'information saisie par l'utilisateur des informations générées par le système dans les champs de saisie » [10].
Si un champ de saisie nécessite un format spécifique, il convient d'indiquer clairement les formats du champ de saisie, tandis que les champs de saisie fixes doivent clairement indiqués.

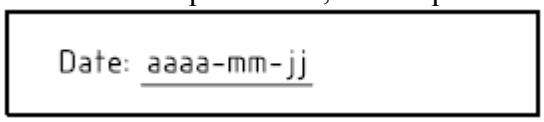

Figure 5 : exemple d'indication de format [10]

Figure 6 : exemple de longueur fixe indiquée [10]

A.8 Les techniques de codage

« Il faut concevoir les créations ou règles de code en impliquant les utilisateur prévus, et en prenant en compte leurs attentes et leurs tâches » [10]. Ceux-ci doivent avoir un sens, le niveau de signification augmente lorsqu'il existe des associations claires entre l'information codée et le sens prévu. Ainsi les codes mnémoniques sont conseillés plutôt que de codes arbitraires. Si ce n'est pas possible alors le sens doit être facilement accessible.
Ces codes doivent être visuellement distincts les uns des autres et utilisés de manière constante avec le même sens ou la même fonction.

A.9 Marqueurs

« Les symboles particuliers également connus sous le nom de « marqueurs », sont utilisés pour attirer l'attention sur des éléments alphanumériques choisis » [10]. « Il est préférable d'utiliser des marqueurs différents pour indiquer la sélection unique et la sélection multiple » [10].
Les marqueurs doivent être près de l'élément marqué, mais de manière à ce qu'ils n'apparaissent pas comme faisant partie des éléments affichés.

Figure 7 : exemple de marqueur [10]

1.5.1.4 Sommaire

Ainsi si une interface s'adapte à la tâche d'un utilisateur et qu'elle répond à ses attentes en lui fournissant l'information nécessaire à l'exécution de sa tâche, en utilisant son vocabulaire et en restant cohérente tout au long d'un dialogue. Et si l'utilisateur peut contrôler son dialogue et recevoir des comptes rendu de ses actions ainsi qu'un soutient et une guidance, il peut mieux comprendre le dialogue avec le terminal qu'il utilise et arrive plus facilement à son but.
Ces principes rendent donc une interface plus intuitive et plus facile d'utilisation et permettent à l'utilisateur d'accomplir une tache plus rapidement et correctement. De plus si un système est renforcé en le rendant capable de supporter et prévenir les erreurs de l'utilisateur ce dernier permettra d'accélérer encore d'avantage la réalisation d'une tâche. Et finalement si le système peut être individualisé il sera encore plus simple pour l'utilisateur de le comprendre

et de l'utiliser.
En addition avec ces principes, des objets graphiques conçus de manière cohérente et de façon à rendre leur utilisation plus naturelle ou à faciliter la lecture de l'information qu'ils contiennent rendent également le dialogue plus intuitif et l'exécution d'une tâche plus facile. En effet si l'information présentée est plus claire, plus concise et compréhensible cela permet déjà à l'utilisateur de réaliser des tâches de perception plus facilement. De plus une bonne organisation de l'information tant dans le fenêtrage que pour la résolution de l'écran permet d'éviter à l'utilisateur bien des manœuvres inutiles et déconcentrantes.
Ces principes et recommandations peuvent donc déjà permettre une critique du logiciel existant.

1.5.2 Les systèmes experts et systèmes informationnel d'aide à la décision

1.5.2.1 Introduction

Depuis l'avènement de l'informatique, l'homme a cherché à créer des système pouvant supporter tant ses prises de décision que la gestion de ses activités. Ainsi au fil du temps, un continuum d'expertise s'est installé allant des systèmes classiques de gestion aux systèmes expert.

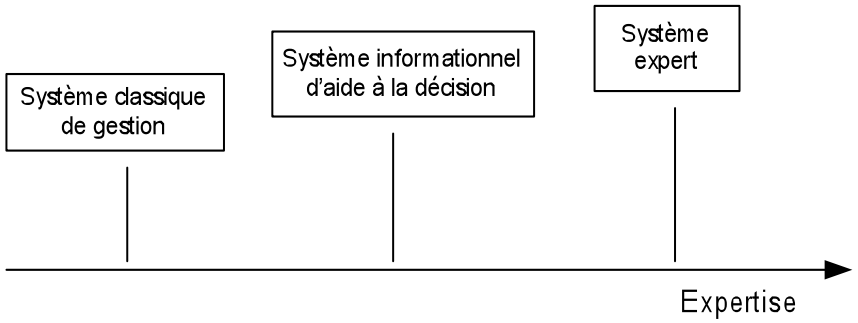

Figure 8 : continuum d'expertise [1]

Nous allons tour à tour présenter les concepts reliés à un système informationnel d'aide à la décision et les systèmes experts. Il existe plusieurs types de systèmes experts se différenciant par leur type de raisonnement. On distingue le raisonnement basé sur les règles et le raisonnement basé sur les cas.

1.5.2.2 Système informationnel d'aide à la décision

Un système d'aide à la décision est un système d'information, permettant de supporter des prises de décisions pour des situations dans lesquelles il n'est pas possible ou souhaitable d'automatiser entièrement le processus de décision [1].

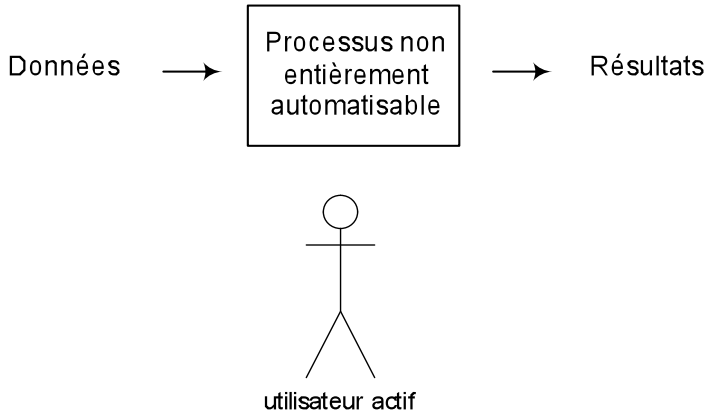

Figure 9 : système informationnel d'aide à la décision [1]

Un processus non entièrement automatisable est un processus dans lequel il doit y avoir une interaction, en l'occurrence ici pour la résolution d'un problème, entre l'utilisateur et le système.

Un système informationnel d'aide à la décision devrait idéalement être vu comme une « boite noire » avec laquelle un utilisateur peut communiquer aisément par un langage proche du langage naturel.

1.5.2.3 Systèmes experts

Un système expert est un outil pouvant reproduire des mécanismes cognitifs, dans un domaine précis, à partir d'une base de connaissance et d'un moteur d'inférence. C'est un logiciel simulant le mode de raisonnement d'un expert, pour acquérir de nouvelles connaissances à partir de faits et règles connus. Les systèmes experts sont des outils d'aide à la décision.

Le but des systèmes expert est d'obtenir l'expérience et l'expertise dans un domaine particulier. Les systèmes experts se veulent aussi évolutifs grâce à un module d'acquisition des connaissances. L'acquisition de connaissances ne signifie pas uniquement ajouter de nouvelles connaissances, mais aussi mettre en relation des nouvelles connaissances entre elles et avec les anciennes connaissances. Ce qui permet d'éviter d'écrire de nouveaux programmes. Une base de connaissance permet de mémoriser tout cela.

« La base de connaissance est un élément capital, c'est la représentation des connaissances de l'expert et la description d'heuristiques utiles » [*Web02*].

« Les systèmes experts sont aussi capables de résoudre d'eux-mêmes les problèmes, de prendre des décisions et de raisonner sur des problèmes » [1]. La résolution se fait par manipulation de connaissances ou par déduction. Ainsi il existe plusieurs types de systèmes experts, chacun réglant des problèmes par d'une manière qui lui est propre. Ce sont ces différentes façons de régler les problèmes qui classifient les systèmes experts. On distingue le raisonnement par cas et le raisonnement par règle.

A. Le raisonnement basé sur les cas

Le raisonnement basé sur les cas, le « case based reasonning » est une technique de modélisation du raisonnement qui se sert de l'importance du passé pour résoudre un

problème. C'est une conséquence simple de la nature humaine. En effet il n'est pas rare d'entendre : « évitons un nouveau … » qui ici, est une référence négative mais il existe évidemment des références positives. Cette technique de modélisation du raisonnement à partir de précédent est fortement utilisée en psychologie cognitive, en enseignement et autres professions comme dans le droit, pour la résolution de cas juridique, grâce à la jurisprudence qui a un rôle important.
« Le précédent sert de cadre de réflexion » [*Web03*] et est très intéressant pour de nouveaux problèmes non encore résolus.

« Le "case based reasoning" est une des techniques de raisonnement analogique. Il signifie raisonner à partir de cas ou d'expériences anciennes pour résoudre un problème, critiquer des solutions, expliquer des situations anormales ou interpréter des situations » [*Web01*].

Une des pionnière de cette technique de modélisation est Janet Kolodner et voici sa définition : « Cased-based reasoning can mean adapting old solutions to meet new demands, using old cases to explain new situations, using old cases to critique new solutions, or reasoning from precedents to interpret a new situation (much like lawyers do) or create an equitable solution to a new problem (much like labour mediators do) » [*Web01*].

Abran et Desharnais [2] présentent les suppositions de J. Kolodner quant à l'utilisation du raisonnement par cas :

- Régularité : le monde est essentiellement régulier et prévisible. Les mêmes actions faites dans les mêmes conditions donneront les mêmes résultats (ou très similaires)
- Répétition : les événements tendent à se répéter. Raisonner par cas est donc très susceptible d'être utile dans le futur.
- Consistance : des petits changements de l'environnement requièrent des petits changements de raisonnement et aussi dans la solution.

Il existe deux styles de raisonnement à base de cas :

1. le style résolution de problème, à partir d'ancienne solutions, utilisées comme guide. Ce raisonnement est utile dans la résolution de problème comme par exemple la planification de tâches.
2. le style interprétatif, on interprète une nouvelle situation dans le contexte d'une ancienne situation. Ce type de raisonnement est intéressant pour l'argumentation, peser le pour et le contre.

Ces deux styles sont forts dépendants de la capacité humaine à la recherche de cas appropriés et aussi à l'interprétation humaine. Il est donc nécessaire de bien classer les cas, et ce n'est pas un exercice évident.
Par contre l'être humain est plus à l'aise pour le raisonnement analogique. En effet lorsqu'il y a une part d'expérience pour une situation nouvelle mais comportant des similarités, les cas du passé « transmettent » leur logique pour la nouvelle situation. Même un novice peut se servir des cas, la justification sera plus difficile. Par contre la résolution, grâce au raisonnement basé sur des cas, est performante lorsque l'utilisateur a une certaine expérience dans le domaine.
Un ordinateur résolvant des problèmes par analogie est appelé un « Case base reasonning system » (CBR Systems) il est doté d'une base de connaissance qui est constituée des cas. Les

CBR Systems sont des systèmes particuliers de raisonnement et ont une diversité d'application dans beaucoup de domaine.

A.1 Qu'est-ce qu'un Case-Based Reasonning tool

"Sur la plan cognitive un Case-Based Reasonning tool (*CBR tool*) est un outil capable d'utiliser une connaissance spécifique venant de situations problématiques expérimentées auparavant.
[…] Un CBR tool est aussi une approche incrémentale d'apprentissage. Une nouvelle expérience est retenue à chaque fois qu'un problème est résolu, le rendant immédiatement disponible pour des problèmes futurs » [2]

A.2 Le processus de raisonnement par cas

Le processus principal d'un raisonnement par cas se décompose en 4 étapes [3] :

Étape 1 : Les cas sont organisés dans la base des connaissances. Il faut, pour résoudre les problèmes, un nombre initial de cas problème dans la base des connaissances. Ces cas initiaux doivent être indexés et organisés en une structure utilisable. C'est par l'utilisation de cette structure que l'utilisateur va, dans la première étape chercher les cas les plus appropriés. Par exemple grâce à l'analyse multi critères.
Etape 2 : Choisir le meilleurs cas et l'adapter au cas courant ou utiliser sa solution comme guide.
Dans le contexte d'une adaptation d'un cas au cas courant, il se peut qu'un cas corresponde exactement. Pour ce qui est de qui est de l'utilisation de la solution comme guide, il faut en réalité construire la solution pour le nouveau cas.
Etape 3 : Tester la nouvelle solution. Dans cette étape la solution du cas problème en cours est testée. C'est à l'utilisateur de décider si la solution est acceptable pour son cas problème.
Etape 4 : Ajouter une nouvelle connaissance à la base de connaissance. Dans cette étape, l'utilisateur décide si la solution, trouvée à l'étape précédente, devrait être ajoutée à la base de connaissance pour en faire une future référence.

A.3 Les avantages du raisonnement par cas

Voici la liste des avantages cités dans la littérature concernant le raisonnement par cas:

- Résolution de problèmes avec un domaine partiellement connu. En effet lorsque le domaine est partiellement définit le raisonnement par cas est plus approprié. En d'autres mots les domaines traités par les CBR systems sont des domaines imparfaitement définis mais riches en expérience ;
- Plus approprié pour des problèmes nouveaux ;
- Proche du raisonnement humain. Il est souvent naturel de résoudre un problème en se servant de sa propre expérience ;
- Raisonnement efficace. L'utilisateur se focalise sur les aspects, de la résolution d'un problème, qui sont importants. De plus le CBR system aide à éviter de prendre une mauvaise direction qui pourrait engendre de nouveaux problèmes ;

- Acquiert rapidement des connaissances. La grande partie des connaissances se trouvent dans les cas, collecter des cas n'est pas une tâche difficile et beaucoup de domaines ont déjà des cas existants ;
- Une seule explication pour un problème précis. Grâce à l'approche par cas, on se réfère à un cas et la solution de ce cas est fournie.

B. Le raisonnement basé sur les règles

Le raisonnement par règle, le « *rules based reasonning* » vient d'un autre comportement typiquement humain aussi. Comme déjà mentionné le raisonnement est un processus cognitif permettant d'obtenir de nouveaux résultats, ou bien de faire la vérification d'un fait.
Le raisonnement par règle s'inscrit dans une logique mathématique, en utilisant la déduction, l'abduction ou l'induction.
Voici une représentation schématique [*Wiki03 raisonnement*] avec les notations classiques de la logique (→ pour l'implication) :

Déduction	Abduction
a a → b	b a → b
b	A

La règle d'abduction se lit ainsi:
1. si *b* est vrai
2. et si *a implique b*
3. alors A est vrai.

La règle de déduction se lit ainsi :
1. si *a* est vrai
2. et si *a implique b*
3. alors *b* est vrai.

Exemple : SI un animal à six pattes ALORS insecte
Il n'est possible de conserver la cohérence qu'avec la déduction, « si la théorie initiale est cohérente, alors toute théorie qui en est une conséquence déductive reste cohérente » [3].
Appliquer une de ces 3 règles consiste à faire un raisonnement simple. Le raisonnement sera qualifié de déductif si il s'appuie sur la déduction et hypothétique si il s'appuie sur l'abduction ou l'induction.

Un ordinateur résolvant des problèmes par application de règle est un « Rule base reasonning system » (CBR Systems) il est aussi doté d'une base de connaissance qui est constituée de l'ensemble des règles.
L'application de ces règles est faite par le moteur d'inférence, qui élabore la solution en choisissant les règles de production et leur séquence d'utilisation.

B.1 Le moteur d'inférence

Le moteur d'inférence [1] d'un système expert est en fait un compilateur de représentation interne. Il applique des règles que choisit un « *scheduleur* ». Un « *scheduleur* » cherche des règles applicables dans une situation ainsi que leur priorités, et choisi une règle parmi toutes les candidates.

Le moteur d'inférence garantit aussi la cohérence, il écarte les incohérences, met à jour le contexte pendant l'inférence, c'est-à-dire qu'il remet à jour les règles applicables. Et enfin il garantit que les conclusions les plus plausibles soient retenues.

B.2 La limite du raisonnement par règle

Le « rule based reasonning » a une limite qu'il faut signaler.
Prenons par exemple [*Web04*]:
le programme " ticket " formé des règles suivantes:
- J'ai le ticket avec X si
- X me fait un sourire furtif et
- X maintient un contact oculaire prolongé

Dans le cas où le sourire et l'œillade sont là, est-ce vrai que j'ai le " ticket "?
Car X pourrait être aveugle.
Il faut ajouter une nouvelle condition à la règle:
- et X non-aveugle
Sans parler de la définition des mots utilisés:

- Un sourire est furtif si sa durée est inférieure à une seconde
- Un sourire n'est pas un mouvement involontaire
- etc.

« En fait le « rule based reasonning » est approprié pour la résolution de tâches bien construites » [4]. Tout le champ des connaissances doit être exhaustif.

1.5.2.4 Sommaire

De plus en plus l'informatique assiste l'homme dans ses activités, dans un premier temps pour ses activités de gestion et dans un second temps pour ses activités de décision. Ainsi sont apparus les systèmes informationnels d'aide à la décision dans lesquels l'homme joue un grand rôle et ensuite les systèmes experts capables de résoudre par eux même des problèmes. On distingue deux types de systèmes experts : les systèmes experts basés sur le raisonnement par cas et les systèmes expert basés sur le raisonnement par règles.

1.5.3 Ré-ingénierie et restructuration

1.5.3.1 La restructuration

Le terme restructuration est vu comme une transformation code vers code. C'est un changement de la structure du code, du design. Cela peut être le passage d'un système non structuré vers un système structuré.
Chikofsky et Cross [5] nous donnent une définition : « La restructuration est la transformation d'une forme de représentation vers une autre au même niveau d'abstraction, tout en préservant le comportement externe du système ».

Beaucoup de restructurations peuvent être faites en connaissant la structure mais sans pour autant connaître le contenu ni même le but d'un système. Par exemple il est possible de

changer une imbrication d'instruction conditionnelle en un « case » ou le contraire, sans pour autant connaître le domaine ou le but du problème.

Cependant la restructuration possède des utilités, il est intéressant de l'utiliser pour une meilleure observation d'un système nécessitant des changements qui amélioreraient le système.

La restructuration est parfois utilisée comme une forme de maintenance de prévention pour améliorer l'état physique d'un système en prenant de nouveau standard.

Enfin la restructuration peut impliquer aussi un ajustement du système pour répondre par la suite à de nouvelles contraintes ne nécessitant pas de réévaluation d'un niveau d'abstraction plus élevé.

1.5.3.2 Ré-ingénierie

L'essence même de la ré-ingénierie est de refaire radicalement la conception des processus d'une organisation. Lors d'un changement au lieu de réviser chacune des fonctionnalités d'une firme, la re-ingénierie propose de repenser le processus complet. C'est une série de nouvelles fonctionnalités qui verront le jour.
La ré-ingénierie au fil du temps a été mal vue car elle s'assimilait avec licenciements massifs.
Mais cet amalgame est faux, ce sont les compagnies qui ont appelé : re-ingénierie, leur réductions de personnel.
Les grands partisans de la ré-ingénierie sont Michael Hammer et James Champy. Dans leurs livres tels que *Reengineering the Corporation*, *Reengineering Management*, et *The Agenda* ils expliquent que beaucoup trop de temps est gaspillé à la révision.

La ré-ingénierie est un processus délicat, il faut avant toutes choses que ce processus aboutisse. Pour cela il faut l'accord des personnes qui savent comment tourne une organisation. C'est un processus souvent tentant, quel dirigeant n'aurait pas l'idée de prendre quelque chose de défectueux et de le concevoir sous une autre forme. Mais la ré-ingénierie ne doit pas viser uniquement ce qui doit être sans se soucier du pourquoi c'est présentement ainsi. Il ne faut donc pas être réticent envers un processus de ré-ingénierie, mais être prêt, savoir ce qui doit être oublié et ce qui sera réinventé, ainsi qu'avoir l'accord des personnes concernées.

La ré-ingénierie comme nous l'avons vu s'applique aux organisations, mais elle peut aussi s'appliquer à beaucoup d'autre domaines comme l'informatique, avec les même problèmes et exigences. Chikofski et Cross [5] donnent une définition plus adaptée à l'informatique de la re-ingénierie : « la re-ingénierie est l'examen et l'altération d'un système pour le reconstituer dans une nouvelle forme et en implémenter cette nouvelle forme ».

1.5.3.3 Sommaire

Nous avons deux processus, permettant la remise à niveau, bien distincts : la ré-ingénierie et la restructuration. Un consiste à examiner un système et le refaçonner entièrement, avec une possibilité d'implication de changement de comportement de ce système. Tandis que le second ne change pas le comportement.

Chapitre 2 La maintenance de logiciel

Comme nous le savons le logiciel *SM*^{Xpert} a pour domaine d'application la maintenance de logiciel. Nous allons donc voir les connaissances de base en maintenance de logicielle.

2.1 Introduction

Tout d'abord, un logiciel commence par son développement, ensuite une fois fini et satisfaisant aux exigences du client, ce logiciel devra évoluer pour satisfaire les besoins d'un environnement en progression. De plus certaines anomalies peuvent être découvertes lors de l'utilisation de ce logiciel fini, un processus de maintenance de logiciel est donc nécessaire. Mais cela ne signifie pas que le cycle de vie de la maintenance débute lors de la mise en production d'un logiciel. «Le cycle de vie de la maintenance débute bien avant par une participation active des responsables de la maintenance tout au long du processus de développement » [11].

Il existe de multiples définitions de la maintenance, ces définitions ont été émises de manière successive dans l'histoire de la maintenance.
Ainsi en 1983 Martin & McLure dans *Software Maintenance: The Problem and its Solutions* donnaient cette définition : « Changements qui doivent être effectués sur un logiciel après sa livraison à l'utilisateur ». Quelques années plus tard la définition s'est quelque peu affinée pour donner : « Modification d'un logiciel, après sa livraison, afin de corriger des défaillances, améliorer sa performance ou d'autres attributs ou de l'adapter suite à des changements d'environnements » *IEEE 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology*.
Finalement, la dernière définition en date est encore plus détaillée. Elle est donnée par le SWEBOK (Software Engineering Body of Knowledge) en 2004: « la totalité des activités qui sont requises afin de procurer un support, au meilleur coût possible, d'un logiciel. Certaines activités débutent avant la livraison du logiciel, donc pendant sa conception initiale, mais la majorité des activités ont lieu après sa livraison finale (l'équipe de développement ayant maintenant terminé son travail et étant affectée à d'autres travaux) »
On dénote une certaine évolution dans les différentes définitions. En effet la première définition est assez allusive, ne précisant même pas en quoi consiste une modification. La deuxième définition précise, quant à elle, ce que signifie les modifications et ce qui les provoque. Indirectement on sait aussi que la maintenance n'est pas une activité temporelle ne se réalisant qu'une seule fois.
Enfin nous pouvons remarquer que dans la dernière définition, des critères comme l'optimisation entrent compte, le début des activités de la maintenance est redéfinit. On voit donc, par le biais de ces définitions que progressivement les connaissances en maintenance se sont étoffées.

La maintenance de logiciel a pris au fil du temps de plus en plus d'importance, elle ne peut plus se réaliser avec négligence il faut une organisation pour mener à bien des activités de maintenance. Ce chapitre présente l'ensemble des connaissances nécessaires à la maintenance de logiciel.

2.2 Différence entre développement et maintenance de logiciel

Le développement de logiciel possède des similitudes avec la maintenance, ce sont deux activités parfois difficiles à distinguer, surtout lorsque les développeurs de logiciel effectuent eux même la maintenance de ce dernier. Il y a en effet beaucoup d'activités communes entre le développement et la maintenance de logiciel (analyse, conception, codage, gestion de la configuration, essais, revues et documentation). Mais pour chacune de ces activités, le contexte est différent. Pour la maintenance, ces activités sont faites dans une optique d'amélioration ou de correction avec un délai souvent plus bref et des ressources humaines restreintes.
Les outils utilisés pour le développement peuvent également servir à la maintenance, mais à nouveau, leur contexte est différent.

Un ingénieur de la maintenance doit donc avoir les qualités requises d'un développeur en plus des aptitudes requises pour les activités propres à la maintenance. Car la maintenance possède bien évidement ses activités propres. En voici un résumé :

1. L'équipe de maintenance doit pouvoir gérer les événements et requêtes de modifications venant de la clientèle, tout en continuant ses travaux. Il faut donc un processus d'acceptation ou de rejet du travail. L'arrivée de ces événements est aléatoire, alors que pour le développement tout est prévu sur un plus long terme et planifié avec un échéancier et aboutit sur une livraison du produit. Il est évidemment impossible, vu le caractère aléatoire des demandes de planifier tout dans un processus budgétisé comme pour le développement de plus «les travaux sont ordonnés de manière à satisfaire l'utilisateur, à court terme, et s'assurer du bon fonctionnement quotidien des logiciels en opérations » [11].
2. Il faut évaluer les requêtes de demande et les classer par ordre de priorité. Il faut estimer l'effort requis pour modifier le logiciel existant. « Si l'effort estimé est peu élevé et peut être accommodé à l'intérieur des ressources et disponibilités de l'équipe de maintenance pour gérer le quotidien de la maintenance, la requête est alors exécutée à l'intérieur de la gestion de la file d'attente » [11]. Si l'effort estimé est élevé par rapport au temps limite adjugé par une organisation ou par rapport à sa marge de manœuvre budgétaire, alors la requête devient un nouveau projet avec un budget spécifique, une équipe et un échéancier. Il faut aussi préciser que les priorités peuvent changer rapidement et à n'importe quel moment suivant les requêtes et prendre aussi priorité sur un travail en cours.
3. Comme signalé, les requêtes de modifications arrivent de manière aléatoire, la charge de travail de la maintenance est donc gérée par « une technique de file d'attente, souvent supportée par un logiciel de bureau d'aide '*help desk*' ».

« Une dernière grosse différence entre le développement et la maintenance est que le développement est « *requirement-driven* » alors que la maintenance est « *event-driven* », ce qui signifie que le stimuli qui initie la maintenance est un événement imprévisible » [12].

2.3 Qui fait la maintenance

Il existe deux modèles distincts en entreprise. Dans le premier modèle organisationnel, la maintenance est effectuée par les développeurs même. Dans le second modèle c'est une organisation de maintenance, indépendante des développeurs, qui effectue la maintenance.

Mais dans le premier modèle organisationnel, souvent les développeurs n'aiment pas faire de la maintenance et préfère travailler sur le développement de nouveaux projets.
On soupçonne plus facilement un manque de transparence des organisations effectuant leur propre maintenance de logiciel, ainsi les logiciels livrés seraient de moins bonne qualité et peu documentés afin qu'il soit plus difficile pour une autre équipe de maintenir le logiciel. Il y aura par conséquent des avantages, du à ce gros inconvénient, de faire de la maintenance avec une équipe indépendante : « une meilleure documentation des logiciels, un processus plus formel et contrôlé de transition du logiciel développé, une plus grande spécialisation du programmeur de maintenance, une meilleure gestion des requêtes de changements et une plus grande satisfaction des employés » [11].

2.4 Le processus de la maintenance du logiciel

Au tout début de l'industrie du logiciel, la maintenance et le développement n'étaient pas distincts. Cela a donné une vue du cycle de vie du développement dans laquelle la maintenance était considérée comme une étape finale du cycle. La maintenance a été longtemps vue sous cet angle et parfois même complètement ignorée. Alors que pourtant les premiers modèles de cycle de vie pour le processus de maintenance sont apparus en réalité très tôt dans l'industrie du logiciel.
Ils étaient « en général coupés en 3 phases.
1) Compréhension,
2) modification et
3) validation du changement au logiciel » [11].

Ces cycles de vies ont été améliorés, et ces 15 dernières années des consensus internationaux ont définit la terminologie et sur les cadres de références pertinents à la maintenance et qui sont actuellement utilisés.

Voici un diagramme général du cycle de vie de la maintenance. Bien évidemment chaque organisation peut faire son propre cycle de vie, mais dans l'ensemble il devrait être proche de celui proposé.

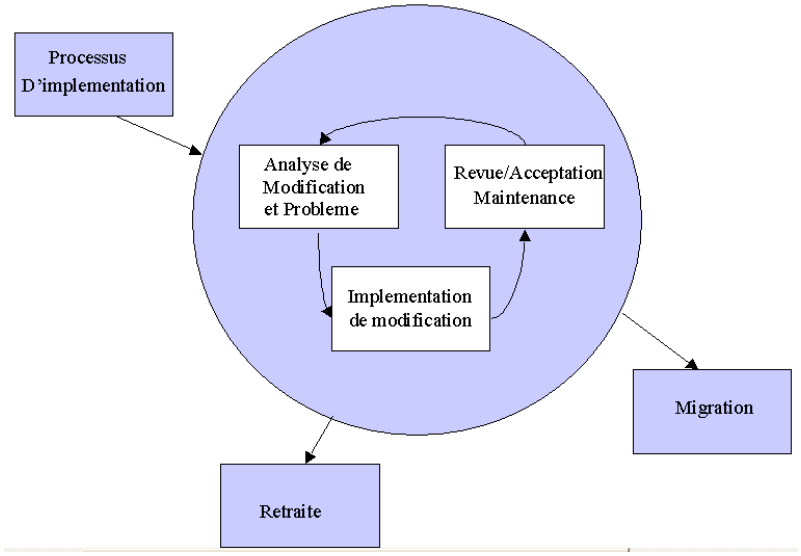

Figure 10 : Meta processus de la maintenance du logiciel [11]

Voici plus en détail les différents processus

- Le processus de l'analyse des modifications et problèmes. Ce processus commence lors de la prise en charge du logiciel par le groupe de maintenance. Il faut y analyser chaque requête (exemple : modification), c'est-à-dire vérifier leur validité et documenter la requête, ensuite chercher une solution et la documenter pour finalement obtenir une autorisation d'effectuer les modifications.
- « Le processus de l'implantation d'une modification autorisée. On s'attarde ici à mettre en œuvre les processus similaires à ceux du développement » [11].
- Le processus de revue/acceptation d'une modification: c'est une validation de la solution en la soumettant à la personne ayant émis la requête.
- Le processus de migration : c'est un processus d'exception, il entre en jeu lorsque le logiciel doit par exemple changer de plate-forme, sans pour autant subir de modifications fonctionnelles.
- Le processus de retrait: c'est également un processus exceptionnel, qui consiste à « enlever un logiciel de l'environnement opérationnel » [11].
- Le processus d'implémentation est plus externe aux processus opérationnel de la maintenance. Ce sont les activités de préparation et de transition du logiciel. C'est-à-dire par exemple : création du plan de maintenance, préparation de la gestion des problèmes identifiés pendant le développement, etc.

2.5 Les Catégories de travaux de la Maintenance du Logiciel

La maintenance vient d'un événement pouvant être un rapport de problème ou bien une demande de changement, ce sont les raisons d'une demande de maintenance qui permettent de définir l'activité de maintenance nécessaire [12].
La maintenance se divise en 4 catégories d'activités : corrective, adaptative, perfective et préventive. Ces activités ont été catégorisées suivant deux critères :
Critère 1: le travail consiste soit en une correction soit en une amélioration au logiciel
Critère 2: le travail est soit proactif, soit réactif.

	Correction	Amélioration
Proactive	Préventive	Perfective
Réactive	Corrective	Adaptative

Catégorie de maintenance [11]

Correction : « corriger une défection. Exemple une anomalie entre le comportement attendu d'un produit et le comportement observé » [12].
La correction préventive est un changement de l'implémentation sans changer les exigences.
Amélioration : « implémentation d'un changement du système qui change son comportement » [12]. L'amélioration adaptative ajoute de nouvelles exigences, tandis que l'amélioration perfective est un changement des exigences existantes.

Il est important de savoir distinguer si on fait de la maintenance corrective ou perfective, car il se peut qu'un problème rapporté par un utilisateur soit la demande d'une fonctionnalité nouvelle qui à l'origine n'était pas requise et dans ce cas le rapport de problème mène à une amélioration.
Les mainteneurs ont besoin de comprendre le produit et d'avoir différents outils. La maintenance corrective pourrait se faire en étant seulement capable de trouver et localiser tous les changements à effectuer dans le code. Tandis que pour la maintenance perfective il faut une large compréhension du produit. Ceci est très possible avec une documentation de qualité. Mais également avec des outils, comme les outils de mesure de complexité cyclomatique

permettant d'avoir une idée de la complexité d'un logiciel, ou alors localiser les endroits sensibles du code. C'est une information importante pour la gestion de la priorité par exemple.
Nous verrons dans les problèmes liés à la maintenance que la documentation joue un rôle important et nous verrons aussi dans la définition de l'ontologie de la maintenance que la documentation est un facteur ayant également un rôle.

2.6 Ontologie de la maintenance de logiciel

Après avoir vu les connaissances fondamentales de la maintenance, il serait intéressant d'en avoir une description plus riche encore.
Pour cela, une ontologie de la maintenance sera présentée. «L'ontologie est une spécification de la conceptualisation » [12]. C'est par le biais d'une identification des facteurs influençant la maintenance que cela sera réalisé.

La figure 1 montre les facteurs qui influencent le processus de maintenance et leur classification.

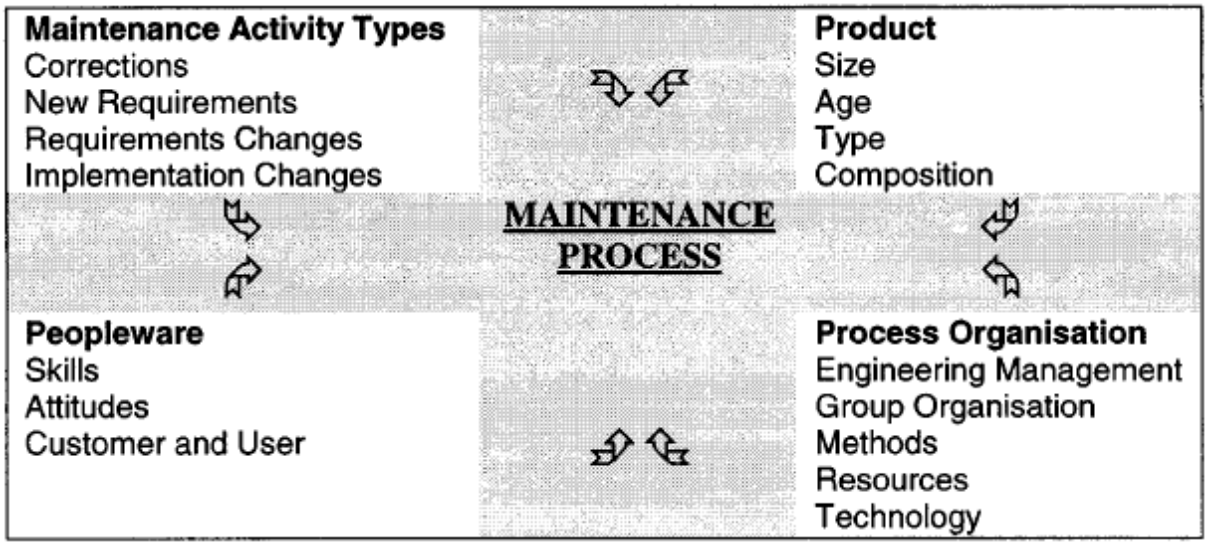

Figure 11 : Vue d'ensemble du domaine des facteurs influençant la maintenance de logiciel [12]

A présent il faut décrire les facteurs ainsi que leur caractéristiques ayant un impact sur l'activité de la maintenance.

2.6.1 Le produit

Voici l'ontologie du produit de la maintenance

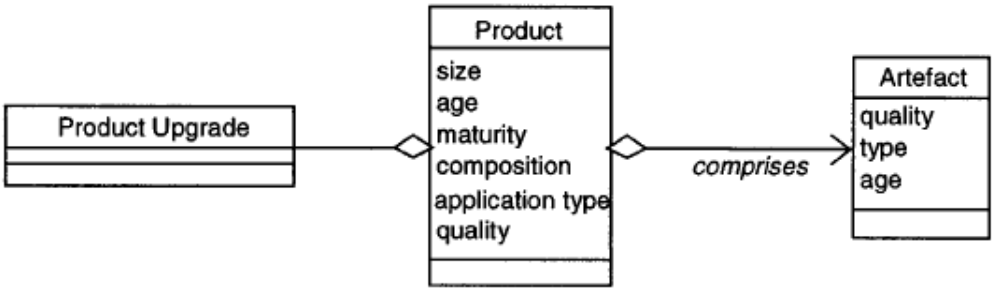

Figure 12 : ontologie du produit en maintenance [12]

Le produit	Logiciel ou package à modifier
Mise à jour du produit	Nouvelle version du produit, un nouveau patch, etc
Artefact	Le code, mais aussi les documents relatifs au code. Ces documents sont le design, la spécification des exigences, etc

Définition de l'ontologie du produit en maintenance

La taille du Produit

Le nombre et l'organisation d'une équipe varieront en fonction de la taille du produit.
Par exemple : plus la taille du produit sera grande plus il sera susceptible que tous les membres d'une même équipe ne comprennent pas le produit de la même façon. Ce qui rend difficile le diagnostique des problèmes et l'identification des modifications à réaliser, car il y a plus de probabilité de malentendu. Ainsi l'activité de la maintenance sur de gros produits peut être moins productive que sur de petits produits.
Une personne est souvent affectée à la maintenance d'un petit produit, alors qu'il y aura déjà une équipe pour un produit de taille moyenne et plusieurs équipes pour un produit de grande taille.

Type ou domaine d'application

Il a été observé des différences majeures de productivités entre différents produits provenant de différents domaines. Par exemple « la maintenance du système de sécurité doit à tout prix préserver la fiabilité du système » [12], c'est une contrainte du domaine d'application fait varier la productivité de la maintenance.

Age du produit

Un vieux produit avec une vieille technologie de développement sera difficile à maintenir. Il y a avant tout un gros risque qu'il soit difficile de trouver quelqu'un ayant encore les connaissances des technologies désuètes. Et il sera aussi difficile de retrouver les concepteurs de ce produit, ainsi que la documentation. Il se peut donc qu'il y ait des parties du logiciel que personne ne puisse comprendre.
On présume donc qu'en général la maintenance sera plus performante pour des produits récents que pour de vieux produits.

Maturité du produit

La maturité ne doit pas être confondue avec l'age. C'est le cycle de vie du logiciel après sa sortie. L'activité de maintenance change suivant l'étape du cycle de vie dans laquelle se trouve le produit.

Etape du cycle de vie	Tâche la plus courante	Nombre d'utilisateur
Stade de l'enfance : arrivées des rapports d'erreurs par les utilisateur initiaux, après livraison du logiciel	correction	Petit
Stade de l'adolescence : le nombre d'utilisateur augmente, les rapports d'erreurs sont prédominants, et il commence a y avoir des demandes de modifications du comportement du système	Corrections et changement des exigences	En croissance
Âge adulte : il n'y a presque plus d'erreur dans le produit. Les demandes de nouvelles fonctionnalités sont prédominantes, et si elles sont acceptées par un grand nombre d'utilisateur, elles seront faites. Si les modifications s'enchaînent, alors il y aura besoin de restructurer le code pour éviter un design dépréciant.	Nouvelles exigences et implémentation de ces exigences	Maximum
Sénile : il existe des nouveaux produits disponibles et il reste seulement peu d'utilisateur à supporter. Ordinairement il ne reste que des corrections à fournir	Correction	Décroissant

Cycle de vie du produit après livraison

La composition du produit

La composition d'un produit affecte également la maintenance et plus particulièrement les aptitudes en maintenance ainsi que les outils nécessaires. « Si le produit est généré du design, alors les ingénieurs de la maintenance doivent avoir accès aux outils de génération de code. »

Qualité du produit

La qualité fournie par le processus de développement d'un logiciel ajoute des contraintes au processus de maintenance. Il est évidemment plus facile de maintenir des produits de bonne qualité, sachant que « qualité » ici signifie : documentation, structure du produit,…
Ce facteur est encore plus important lorsqu'il n'y a pas moyen d'entrer en contact avec l'équipe ayant développé le produit.
Il faut remarque que l'existence d'une documentation n'implique pas que le produit soit pour autant de qualité, il faut également que cette documentation soit de qualité. B. A. Kitchenham et Al définissent une documentation de qualité comme une documentation complète, précise et lisible.

2.6.2 Les activités de maintenance

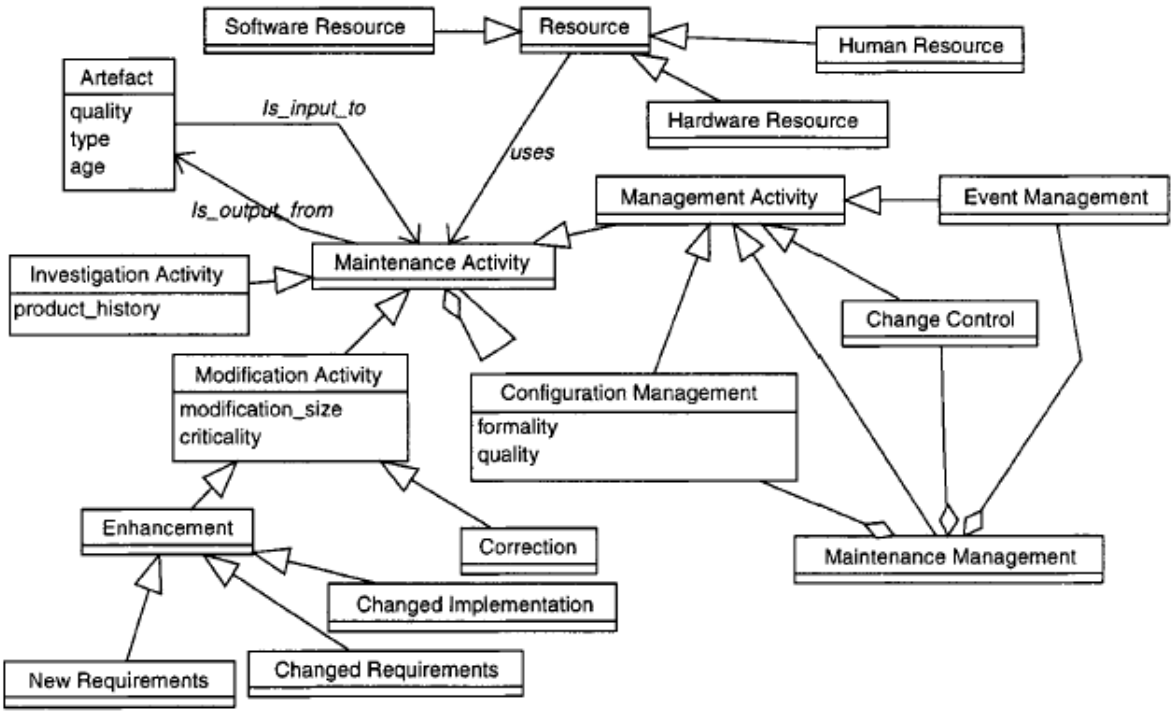

Figure 13 : ontologie de l'activité de maintenance [12]

Activité d'investigation	Évaluer l'impact d'une modification
Activité de modification	(cfr : catégories de travaux de la maintenance de logiciel).
Activité de gestion	Activités de gestion des requêtes, activités permettant de s'assurer de l'intégrité d'un produit en maintenance et de ses différentes versions.
Ressource	Ressources humaine, matérielle et logicielle permettant une activité de maintenance

Définition de l'ontologie de l'activité de maintenance

La maintenance commence soit à cause des utilisateurs, soit à cause mainteneurs. Les causes peuvent être un rapport d'erreur ou une demande de changement. Ces causes seront étudiées par un ingénieur en maintenance, dans l'étape de l'*analyse des modifications et problèmes* du processus de maintenance, afin de déterminer si des modifications de maintenance sont nécessaires ou pas.

Les catégories maintenance ont précédemment été présentées. Il est utile de rappeler rapidement les deux activités principales sans pour autant s'attarder sur leurs variantes :

- Correction : « corriger une défection. Exemple une anomalie entre le comportement attendu d'un produit et le comportement observé » [12].
- Amélioration : « implémentation d'un changement du système qui change son comportement » [12].

Importance d'une amélioration/correction

Une caractéristique des activités de la maintenance affectant la productivité et l'efficacité est l'importance de l'activité de maintenance. Une large amélioration/correction demandera un plus gros effort et plus d'ingénieurs en maintenance et particulièrement plus si le produit est également important.

Connaissance du produit

Le degré de compréhension et les outils à dispositions sont aussi des facteurs. Lors d'une maintenance corrective, les aptitudes nécessaires peuvent se réduire à pouvoir corriger des fautes de codes et ainsi procéder seulement à des changements locaux, mais lorsqu'il s'agit d'une amélioration il faut évidemment connaître le produit.

2.6.3 L'environnement humain (peopleware)

Le développement de logiciel ainsi que sa maintenance demande beaucoup de ressources humaines travaillant en groupes. Ainsi le facteur humain et social ne peut pas être ignoré en tant que facteur affectant la maintenance. Nous allons donc voir l'ontologie de l'environnement humain. Il n'y a pas de définition de chaque élément de l'ontologie car leurs noms sont suffisamment parlant.
Dans le processus de maintenance, il y a deux types d'équipe :

- l'équipe s'occupant de la maintenance (« *engineers* » sur la Figure 14)
- l'équipe s'occupant du client (« *managers* » sur la Figure 14)

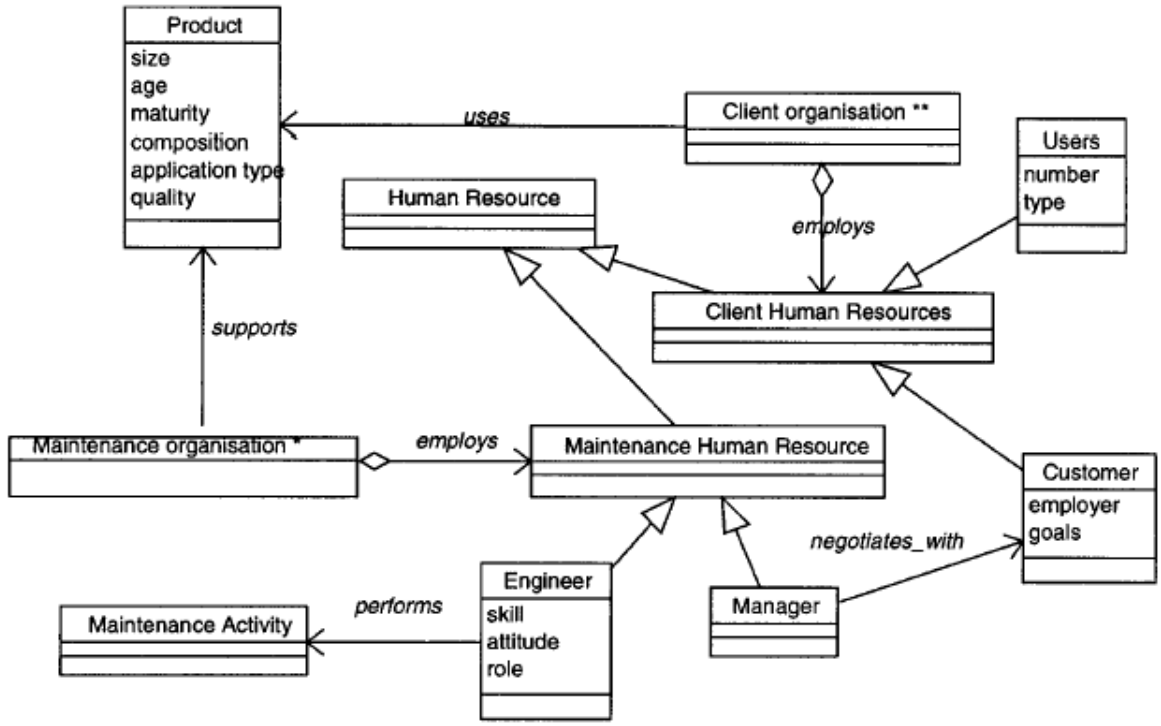

Figure 14 : ontologie de l'environnement humain [12]

L'équipe s'occupant de la maintenance

La motivation

Les motivations et les attitudes du personnel influent sur leurs activités, les gens ont une perception négative de la maintenance, ceci sera expliqué plus en détails dans les problèmes de la maintenance, mais ici disons seulement que cela affecte directement la performance de la maintenance.

Séparation entre développeur et ingénieur de la maintenance

Le fait d'avoir ou non une séparation stricte entre équipe responsable du développement et équipe responsable de la maintenance est important. Certaines organisations n'ont pas cette séparation, cela dépend du produit développé. Le produit est en continuelle évolution et de nouvelles versions sortent régulièrement. Le développeurs font eux même les correction et les améliorations sont inclues dans un processus continu. Ainsi les outils permettant le développement et ceux permettant la maintenance ne sont pas différents.
Le cas contraire, celui où il y a clairement une séparation, et voire même deux compagnies différentes, une de développement et une de maintenance, il est clair que les mainteneurs auront besoin d'outils spécialisés (des outils de rétro ingénierie par exemple).

L'équipe s'occupant des clients

Diversité

La diversité de la clientèle qui agrandi les possibilités de taches de maintenance. Plus elle sera grande, plus il y aura différents types de problèmes.

Financement

La volonté d'amélioration du logiciel par les clients. Le financement provient de ces derniers, il va de soit que la maintenance dépend de leur satisfaction. En effet si les exigences ne sont pas atteintes au cas où elles seraient mal comprises, les clients ne seront pas satisfaits.

2.6.4 Le processus de maintenance

Il faut faire une distinction entre le processus de maintenance réalisé par les ingénieurs en maintenance pour une modification spécifique et le processus d'organisation de la gestion des requêtes de maintenance des clients et des mainteneurs eux-mêmes.

Processus pour une modification spécifique

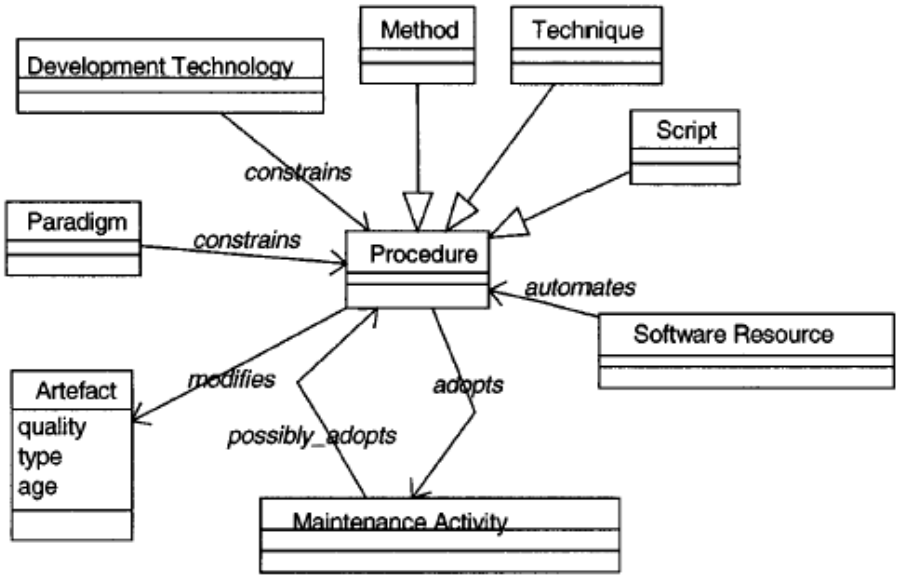

Figure 15 : ontologie d'une modification spécifique [12]

Paradigme	La philosophie adoptée durant le développement initial du produit.
Procédure	La procédure même d'une activité de maintenance. Cela peut être une méthode, une technique ou un manuscrit (« *script* »). Certaines procédures peuvent être choisies pour réaliser une activité spécifique.
Méthode	Une procédure systématique définissant les pas et les heuristiques afin de réaliser une activité de maintenance.
Manuscrit technique	Une procédure utilisée pour accomplir une activité, mais étant moins rigoureusement définie qu'une méthode

Paradigme de développement

Le paradigme de développement d'un logiciel affecte la performance de la maintenance, plus précisément la contraignent. Les mainteneurs ont besoin des aptitudes nécessaire à la compréhension du langage utilisé.

Technologies de développement

Les technologie permettant le développement d'un logiciel présentent un risque, il se peut qu'elle deviennent un jour inutilisables. Il faut donc s'assurer de ce risque pour les logiciels ayant un long cycle de vie.

Enfin le degré d'automatisation des procédures influe aussi la performance.

L'organisation de la maintenance

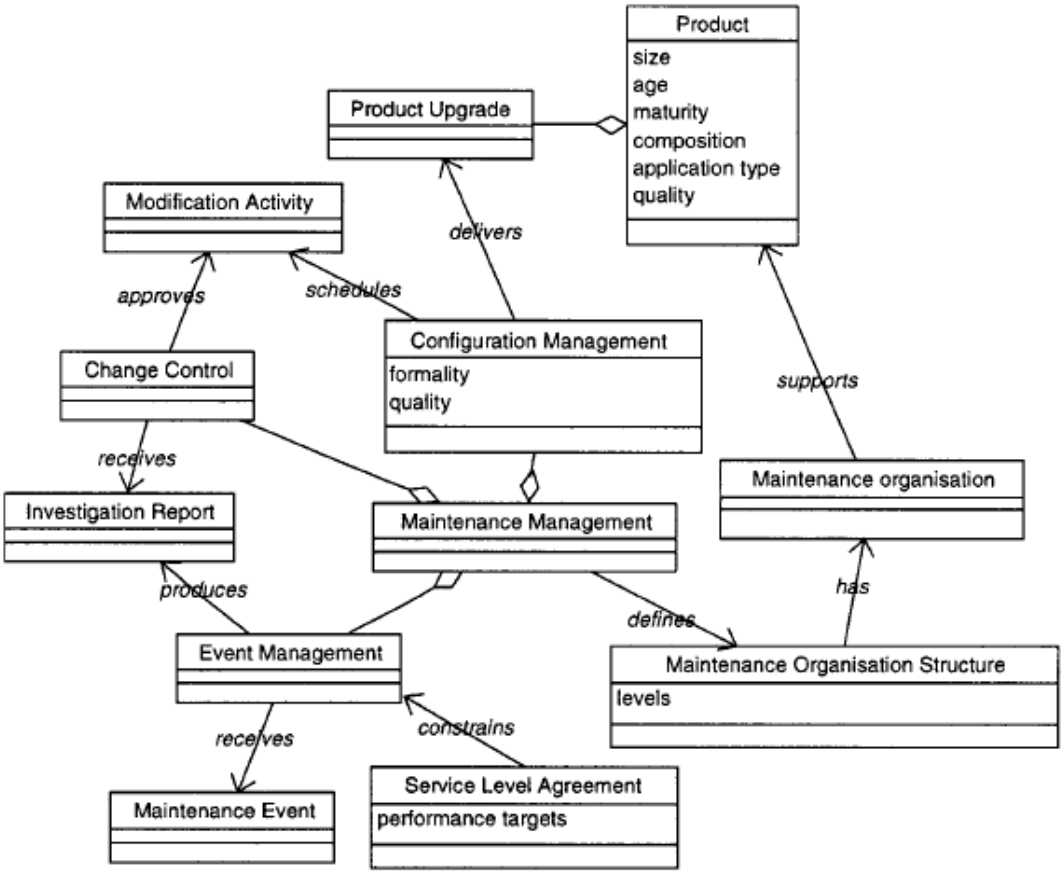

Figure 16 : ontologie de l'organisation de la maintenance [12]

« Service Level Aggrement » (SLA)	Accord entre le fournisseur de service de maintenance et le client qui spécifie ses attentes de service
La gestion des configurations	C'est la gestion des versions d'un produit, et de son état de modification.
Le contrôle de changement	Le processus responsable d'évaluer les résultats d'une investigation de maintenance et de décider si le changement aura lieu
Evénement	Un rapport de problème ou une requête de changement.

Requêtes

L'influence de la quantité de requêtes de maintenance sur les performances de l'organisation de la maintenance n'est pas négligeable, exemple si il y a trop de demandes, il pourrait très bien manquer de ressources humaines.

Outils

Il est important d'avoir des outils de support à la maintenance. La plupart des organisations ont leurs outils de contrôle de configuration et il existe aussi beaucoup d'outils de support à la gestion des événements permettant de récupérer les demandes et de les tracer.

2.7 Problèmes de la maintenance du Logiciel

Le problème majeur en maintenance de logiciel n'est pas de nature technique mais c'est bien sa gestion.
On distingue deux catégories de problèmes de la maintenance :

1. Les problèmes internes venant de la perception des employés et gestionnaires de la maintenance de logiciel
2. Les problèmes externes venant de la perception de la clientèle.

2.7.1 Problèmes internes

Plus un logiciel fait objet de maintenance, plus la complexité de sa structure et sa taille s'accroissent. Et progressivement l'effort de maintenance devient élevé. Ceci est un problème technique fréquent car les logiciels sont de plus en plus susceptibles de faire l'objet de maintenance et ce plusieurs fois. Mais comme déjà précisé les problèmes techniques ne sont pas les plus importants.

Le caractère aléatoire des requêtes, et la difficulté de les comprendre. L'obligation de réagir rapidement aux pannes et de s'assurer que le service est rapidement et bien restauré, c'est-à-dire respecter un niveau de service établi afin de garder la confiance de la clientèle. La gestion de priorités changeantes car les organisations des utilisateurs changent vite. Tout cela crée une très mauvaise et stressante atmosphère pour une organisation de maintenance.
A cette atmosphère, une condition à laquelle on ne pense pas forcément vient s'ajouter : la motivation. En effet « en maintenance la motivation peut poser problème car l'activité de maintenance peut être perçue comme moins importante que le développement » [12]. Il est très difficile de mesurer la contribution de la maintenance, ainsi que sa performance, il y a par conséquent un manque de considération et de respect de la maintenance. Les novices y sont le plus souvent assignés et elle est aussi perçue comme une punition. Les mainteneur, du au manque de considération se voient aussi contraints « d'accepter, avec résignation, le prochain logiciel développé, peu importe son niveau de qualité » [11]. Ils sont donc aussi contraints de résoudre des problèmes qui ne devaient pas être sensés régler. Les ingénieurs de la maintenance se retrouvent ainsi avec pour impression d'être les seuls responsables du bon fonctionnement, de gestion des logiciels et du support à la clientèle.

Les ingénieurs de la maintenance ont également peu de support technique et d'outils spécifiques à la maintenance et doivent parfois faire face à une documentation du logiciel incomplète ou absente. Cela arrive souvent lorsque le budget pour le développement n'est pas suffisant, dans ce cas les étapes d'analyses sont moins prises en compte. Mais ce souci d'économie est une fasse idée car la maintenance sera plus onéreuse. Le problème de documentation vient directement du processus de développement, on voit donc qu'un processus de développement mature garanti une certaine facilité de maintenance, non pas spécialement par la documentation mais également par la qualité du code.

Il est aussi très difficile de retracer la création d'un logiciel et de ses changements. De plus les changements ne sont que très rarement documentés.

Il y a aussi des problèmes indirectement liés aux problèmes internes de la maintenance comme par exemple le fait que la maintenance n'est pas enseignée comme elle est vécue en industrie. « Il en résulte un manque de connaissances et techniques de la maintenance pour les candidats potentiels pour des emplois en maintenance du logiciel » [11].

Un dernier phénomène non négligeable est la culture. Certaines cultures sont plus basées sur la qualité et ont l'idée que l'amélioration augmente régulièrement la satisfaction du client. Ces cultures donnent donc une place plus importante à la maintenance.

2.7.2 Problèmes externes

Un premier problème est que la grande majorité des coûts dans le cycle de vie des logiciels semblent liées à la maintenance. A. April [11] rapporte que les études faites dans ce sens appuieraient cette croyance parmi les utilisateurs.
La maintenance est un travail laborieux, et les salaires des programmeurs prennent une grosse partie des coûts. Mais ces coûts sont des coûts justifiés, ce qui rend la maintenance si chère à la vue des utilisateurs provient d'autres problèmes.

Le caractère aléatoire des multiples requêtes des utilisateurs doit pouvoir être géré par des mécanismes de gestion de file d'attente. Car, il arrive souvent que les utilisateurs ne précisent pas leurs priorités ou alors les changent eux-mêmes constamment et il peut arriver que certaines de leurs priorités soient oubliées ou même qu'une tâche de maintenance en cours soit postposées. Il se peut également que des pannes surviennent. Ces problèmes sont bel et bien des problèmes de gestion à nouveau. Et cela peut devenir frustrant pour un utilisateur d'attendre à tel point que ce dernier finisse par proposer ses propres solutions ou même sous traite chez un externe la maintenance.

Mais est-ce que cette vision de lenteur est-elle entièrement bien fondée ? A. April [11] reprend une étude de Lientz et Swanson dans *Software Maintenance Management*, qui, sur base d'un sondage auprès de 487 groupes de maintenance, a rapporté que 55% des requêtes des utilisateurs étaient en réalité des nouvelles exigences. Mais ce n'est pas tout, les requêtes des utilisateurs ne constituent pas la seule source de requête. Il y a aussi les requêtes des opérateurs, des gestionnaires de projets et des études de re-ingénieries qui elles aussi viennent chambouler les priorités.
La clientèle n'est pas consciente que les services de maintenance font en réalité plus que de la maintenance. Une fois encore une maintenance mieux gérée éviterait cette confusion.

Un dernier point à souligner sur cette vision de lenteur de service, c'est qu'elle est aussi due à une incompréhension de la différence entre matériel et logiciel. Le matériel est palpable, il est facile de détecter un composant défectueux et de le remplacer, sans pour autant changer les autres composants. Ce n'est pas encore le cas pour la maintenance de logiciel.

2.8 La mesure pour la maintenance

Maintenant que nous avons vu pas mal de problème en maintenance, il serait intéressant de savoir comment contrôler ces problèmes. Pour pouvoir contrôler, il faut pouvoir prédire, comparer, évaluer, les outils de mesure sont utiles dans ses activités. En ce qui concerne la maintenance c'est prédire le comportement de ressources, mesurer le processus de maintenance et son produit, évaluer le service ou le comparer.

2.8.1 L'estimation des ressources

L'estimation des ressources (nombre d'employés, budgets, etc) se fait le plus souvent suivant deux approches. Une première basée sur l'expérience et une seconde en utilisant des modèles tels que COCOMO-maintenance.

2.8.2 La mesure du processus de la maintenance

L'objectif ici est de mettre sous contrôle le processus de la maintenance.
Bien que la mesure de la maintenance ressemblera fortement à la mesure du développement, ces deux mesures doivent être distinctes car leurs exigences sont différentes. Voici plusieurs exemples de mesure du processus de la maintenance.

Le premier exemple est un ratio simple, le *'Temps moyen de changement'* (représentant le temps moyen pour effectuer un changement sur un logiciel après son développement) est une autre perspective intéressante à mesurer. Une interprétation possible de ce ratio est que son importance est un indice de la 'maintenabilité' d'un logiciel. Si le temps moyen de changement d'un logiciel est court alors c'est qu'il est facilement maintenable.

Il existe aussi une mesure du coût de la 'maintenabilité' du logiciel (coût de changement sur un logiciel après son développement). En utilisant le ratio du nombre de défaillances en relation avec le coût du projet initial de développement. On peut ainsi savoir quel processus de développement offre un coût de maintenance plus faible.

2.8.3 La mesure du produit de la maintenance

Il existe peu de mesure spécifique à la maintenance pour mesure son produit. Mais il existe beaucoup de mesure du code source, mesure pour le développement, donnant des informations sur la 'maintenabilité' d'un logiciel.

Comme déjà précisé la qualité du logiciel a un impact sur la maintenance, il est important d'avoir un logiciel de qualité. Il existe des mesures de la qualité d'un logiciel et prenant en compte l'aspect 'maintenabilité'.

Une mesure possible pour le produit est l'effort nécessaire pour analyser et effectuer des modifications sur un logiciel ou mesurer les attributs du logiciel qui vont avoir un impact sur l'effort de maintenance.
Les mesures des attributs d'un logiciel viennent en général du code. Il est possible de mesurer de différentes façons la complexité de la structure en étudiant une représentation graphique du code source. L'exemple le plus connu est celui de la complexité cyclomatique de McCabe.

Il est aussi possible de mesurer la structure du code. Une programmation structurée est basée sur une décomposition du problème en sous problèmes gérés par des composants du logiciel. Les mesures de couplages aident à déterminer si les composants du logiciel sont indépendantes ou pas. On peut ainsi savoir grâce à cette indépendance si une modification d'un composant aura de grosses répercutions sur le reste des autres composants.

Une dernière mesure du produit est celle de la documentation interne. Elle fournit aussi des informations, en effet la documentation aide le programmeur à comprendre l'utilisation de telle ou telle variable ainsi que la but d'une méthode.

2.8.4 La mesure du service

On distingue deux catégories des mesures du service :
- l'entente de service
- l'étalonnage de la maintenance du logiciel.

L'entente de service

Définissons tout d'abord ce qu'est l'entente de service. L'entente de service, le 'Service Level Agreement' (SLA), est l'objectif de performance d'une organisation. Le SLA est donc un élément important de la satisfaction du client.
Pour y parvenir les organisations séparent souvent les activités de support en rôles bien définis permettant ainsi la spécialisation d'une équipe dans chacune des activités. On retrouve ainsi une partie du personnel s'occupant du client, une autre partie s'occupant de la correction et une autre du perfectionnement du logiciel.

Dans la plupart des situations on trouve 3 niveaux [12]:

- Level 1 : l'équipe de « help desk » n'a pas de compétence technique, et est seulement responsable de recevoir les problèmes et de savoir quel technicien de support pourra aider l'utilisateur
- Level 2 : les techniciens de support savent comment communiquer avec l'utilisateur, comprendre leur problème et les conseiller
- Level 3 : les mainteneurs sont autorisés à faire des changements du produit

L'étalonnage de la maintenance

L'étalonnage, plus connu sous sa traduction anglophone 'benchmarking', consiste pour une entreprise en une comparaison. Il faut donc, pour une entreprise, avant toute comparaison pouvoir se connaître (ses propres processus, etc). Il est possible de faire une comparaison interne (entre les secteurs de l'organisation), une comparaison externe avec les compétiteurs en ayant recours à une firme spécialisée dans l'étalonnage.
Voici quelques exemples de mesures comparées [11] :

- Points de fonctions supportés par personne (développement maison)
- Points de fonctions supportés par personne (développement maison + progiciels)
- Coût par point de fonction supporté
- Nombre de langages de programmation supportés
- % de type de programmation (Maintenance, Améliorations, Nouvelles applications)

2.9 Conclusion

Nous voyons que la maintenance est à présent une discipline à part entière avec son propre cycle de vie, indépendant de celui du développement, elle a un processus propre avec des étapes bien définies, des activités qui lui sont spécifiques comme la gestion d'une file d'attente des requêtes. Elle peut être effectuée par les développeurs ou bien par une organisation indépendante.

La maintenance a également une ontologie propre, avec des facteurs l'influençant comme le personnel, le produit, etc.

Le problème majeur en maintenance de logiciel n'est pas de nature technique mais c'est bien sa gestion. Il faut gérer une file d'attente de requêtes arrivant aléatoirement, peu d'outils spécifiques à la maintenance, elle est perçue comme moins importante et leur tâche est beaucoup plus difficile lorsque les développeurs n'ont pas effectué un travail de qualité. Enfin la maintenance est aussi vue, à tort, comme coûteuse et lente. Mais c'est sans compter sur un comportement parfois ingérable des utilisateur et sur une méconnaissance de la maintenance et de ce qui relève de son champs d'action.

La mesure en maintenance est aussi très importante, elle permet de garder le contrôle de la maintenance de logiciel grâce dans un premier temps à l'estimation des ressources. Dans ce cas l'expérience peut être utile, mais il existe aussi des modèles. L'estimation des ressources n'est pas le seul outil de mesure, la mesure du processus de la maintenance ainsi que du produit de la maintenance nous fournissent des informations sur la 'maintenabilité' d'un logiciel, ou de l'effort à fournir pour le maintenir. Enfin la mesure du service est aussi importante, d'une part en se mesurant par rapport aux autre et ensuite en regardant sur soi-même par une mesure de la capacité à satisfaire l'utilisateur.

Bien que la maintenance ait beaucoup de liens avec le développement de logiciel, son processus est différent, ses résultats le sont également. Elle peut reprendre beaucoup d'outils utiles au processus de développement, mais l'utilisation est contextuellement différente. *SM*^{Xpert} se veut un outil propre à la maintenance, il est la preuve matérielle de l'existence de cette activité en génie logiciel.

Références

[1] **J-M Jacquet**, « Technique d'Intelligence Artificielle » - *Namur: Facultés universitaires Notre-Dame de la Paix, Année académique 2004-2005.*
[2] **A Abran et J-M Desharnais**, « Applying a Functional Measurement Method : Cognitive Issues » - *Montréal, Quebec, Canada : International Workshop on Software Measurement (IWSM'01), 28-29 août 2001.*
[3] **Li D. Xu**, « Case-Based Reasonning : A major paradigm of artificial intelligence » - *IEEE Potentials, 1994.*
[4] **L. An, J. Yan et L. Tong,** « An Intergrated Rule-Based and CaseBased Reasoning System for Customer Service Management » - *IEEE International Conferene an e-Business Enguneering (ICEBE'05), 2005.*
[5] **E. J. Chikofsky et J. H. Cross**, « Reverse Engineering and Design Recovery: a Taxonomy » - *IEEE, janvier 1990.*
[6] **S. Perera et I. Watson**, « An integrated approach for Case-Based design and estimating » - *London : IEE, 1995.*
[7] **J-M Desharnais, A Abran, A Mayers, L. Buglione et V. Bevo**, « Knowledge Modeling for the Design of a KBS in the functional Size Measurement Domain ».
[8] **M. Noirhomme – Fraiture**, « Human Computer Interaction » - *Namur : Facultés universitaires Notre-Dame de la Paix, Année académique 2003-2004.*
[9] **Association Française de Normalisation**, « *exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation – Partie 10* » - *Paris La Défense Cedex, 1996.*
[10] **Association Française de Normalisation**, « *exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation – Partie 12* » - *Paris La Défense Cedex, 1999.*
[11] **A. April et A. Abran**, « Modèle d'Evaluation de la Capacité à Maintenir le Logiciel (MÉC^{ML}) » - *Université de Quebec : Ecole de Technologie Supérieure, Montréal, Quebec, Canada, 2005.*
[12] **B. A. Kitchenham et Al**., *« Towards an Ontology of Software Maintenance »* - *Journal of Software Maintenance : Research and Practise, 1999.*

[*Wiki01*] http://fr.wikipedia.org/wiki/Raisonnement
[*Wiki02*] http://fr.wikipedia.org/wiki/Syst%C3%A8me_expert
[*Wiki03*] http://fr.wikipedia.org/wiki/R%C3%A9tro-ing%C3%A9nierie

[*Web01*] http://tecfa.unige.ch/tecfa/publicat/schneider/these-daniel/wmwork/www/phd_45.html
[*Web02*] http://www.tripalium.com/fiches/auto-evaluation/sysexpert.html
[*Web03*] http://tecfa.unige.ch/tecfa/publicat/schneider/these-daniel/wmwork/www/phd_48.html
[*Web04*] http://villemin.gerard.free.fr/LogForm/Raisonne.htm
[*Web05*] http://www.strassmann.com/pubs/reengineering.html
[*Web06*] http://www.change-management.com/change-management-overview.htm