FUNDP - Namur
Institut d'Informatique
Rue Grandgagnage, 21
B-5000 Namur

Amélioration du processus de la maintenance du logiciel par un système informatisé d'aide à la décision

ARNAUD COUNET

Promoteur : Prof. Naji Habra
Maîtres de stage : Prof. Alain April - Prof. Jean-Marc Desharnais

Mémoire présenté pour l'obtention
du grade de maître en informatique

Troisième Maîtrise
Année académique 2006-2007

Résumé

La maintenance et le support des logiciels d'une organisation n'est pas une tâche aisée, et les ingénieurs en maintenance de logiciel n'ont actuellement pas accès aux outils d'évaluation des stratégies afin d'améliorer les activités spécifiques à la maintenance de logiciel. Ce mémoire présente un système informatisé d'aide à la décision qui aide à localiser les meilleures pratiques dans un modèle de maturité spécifique à la maintenance du logiciel ($S3^m$). Les contributions de ce mémoire sont les suivantes : 1) une description de l'état de l'art en terme de maintenance du logiciel 2) un outil d'aide à la décision complémentaire au modèle de maturité afin d'aider les praticiens de la maintenance du logiciel en localisant les meilleures pratiques associées ; et 3) une description de l'approche basée sur la connaissance ainsi que du système employé par l'équipe de recherche.

Mots-clés :
maintenance du logiciel, modèle de maturité, système informatisé d'aide à la décision.

Abstract

Maintaining and supporting the software of an organization is not an easy task, and software maintainers do not currently have access to tools to evaluate strategies for improving the specific activities of software maintenance. This Master's Thesis presents a decision support system which helps in locating best practices in a software maintenance maturity model ($S3^m$). The contributions of this Master's Thesis are : 1) to describe the state of the art in term of maintenance 2) to instrument the maturity model with a support tool to aid software maintenance practitioners in locating specific best practices ; and 3) to describe the knowledge-based approach and system overview used by the research team.

Keywords :
software maintenance, maturity model, decision support.

Avant-propos

Ce mémoire est le fruit d'un stage de fin d'études réalisé à l'Ecole de Technologie Supérieure (ETS) de Montréal au Canada par un étudiant de troisième maîtrise en informatique aux Facultés Notre-Dame de la Paix (FUNDP) de Namur en Belgique.

Je tiens à remercier tous ceux sans qui ce travail n'aurait probablement pas pu être mené à bien.

En particulier, je remercie mon promoteur, le Professeur Naji Habra pour son aide précieuse dans la finalisation de la rédaction de ce document. Je remercie également les Professeurs Alain April et Jean-Marc Desharnais, maîtres de stage, pour leur aide continue aussi bien pendant le stage qu'après celui-ci.

Finalement, je souhaiterai dédier une partie de cet avant-propos aux personnes m'ayant soutenu moralement tout au long de ce travail, à savoir mes parents, Claire Storme, Stéphane Abran et Ernest Gregoire.

Table des matières

Résumé **1**

Abstract **1**

Avant-propos **2**

Introduction **10**

I Revue de littérature **11**

1 Maintenance du logiciel **12**
1.1 Fondamentaux 12
1.1.1 Définitions 12
1.1.2 Cycle de vie du logiciel 13
1.1.3 Nécéssité d'entreprise 14
1.1.4 Difficultés inhérentes 14
1.1.5 Acteurs de la maintenance 15
1.1.6 Problème de coût 16
1.2 Catégories de la maintenance 17
1.2.1 Maintenance corrective 17
1.2.2 Maintenance adaptative 17
1.2.3 Maintenance perfective 17
1.2.4 Maintenance préventive 17
1.3 Processus de la maintenance 18
1.3.1 En théorie 18
1.3.2 En pratique 20
1.4 Problématiques principales de la maintenance 20
1.4.1 Problèmes techniques 21
1.4.2 Problèmes de management 22
1.4.3 Estimation des coûts de la maintenance 23
1.4.4 Mesure de la maintenance du logiciel 23
1.5 Conclusion 24

2 Sofware Maintenance Maturity Model : $S3^m$ 25
2.1 Modèles de maturité en génie logiciel . . . 25
2.2 Fondations du modèle $S3^m$ . . . 28
2.2.1 Contexte de la maintenance de logiciel . . . 28
2.2.2 Classification des processus de la maintenance du logiciel . . . 29
2.2.3 Identification des domaines de processus . . . 31
2.2.4 Le concept de *Roadmap* . . . 32
2.2.5 Les niveaux du modèle de maturité . . . 34
2.3 La gestion du processus de la maintenance du logiciel . . . 36
2.4 La gestion des requêtes de la maintenance du logiciel . . . 38
2.5 L'ingénierie d'évolution . . . 40
2.6 Le support à l'ingénierie d'évolution . . . 41
2.7 Conclusion . . . 43

II Un système d'aide à la décision : $S3^m$DSS 44

1 Introduction 45

2 Présentation générale 46
2.1 Origines . . . 46
2.2 Objectifs . . . 49
2.3 Fonctionnement . . . 50
2.4 Classification . . . 52
2.4.1 Système expert . . . 52
2.4.2 Système informatisé d'aide à la décision . . . 53
2.4.3 Justification . . . 53
2.5 Conclusion . . . 54

3 Analyse des besoins 55
3.1 Catégories d'utilisateurs . . . 55
3.1.1 Evaluateur externe . . . 56
3.1.2 Evaluateur interne . . . 57
3.1.3 Administrateur . . . 58
3.1.4 Expert . . . 59
3.2 Cas d'utilisation . . . 60
3.2.1 Diagramme des cas d'utilisation - Vue *Utilisateur* . . . 61
3.2.2 Fonctionnalités - Vue *Utilisateur* . . . 61
3.2.3 Diagramme des cas d'utilisation - Vue *Administrateur* . . . 63
3.2.4 Fonctionnalités - Vue *Administrateur* . . . 63
3.2.5 Diagramme des cas d'utilisation - Vue *Expert* . . . 65
3.2.6 Fonctionnalités - Vue *Expert* . . . 65
3.3 Exigences non-fonctionnelles . . . 66
3.4 Conclusion . . . 67

4 Analyse conceptuelle **68**
4.1 Modélisation structurelle . . . 68
4.1.1 Diagrammes de robustesse . . . 68
4.1.2 Diagramme de classes . . . 69
4.2 Modélisation comportementale . . . 71
4.2.1 Diagrammes de séquences . . . 71
4.3 Modélisation de la persistance . . . 74

5 Architecture **76**
5.1 3-Tiers, en théorie . . . 76
5.2 3-Tiers, en pratique . . . 78
5.2.1 Couche Présentation . . . 79
5.2.2 Couche Métier . . . 79
5.2.3 Couche DAO . . . 80
5.3 Conclusion . . . 81

6 Choix technologiques **83**
6.1 HTML . . . 83
6.2 CSS . . . 85
6.3 JavaScript . . . 86
6.4 XML . . . 87
6.5 Java . . . 88
6.6 Java Server Pages . . . 88
6.7 Apache Tomcat . . . 89
6.8 Microsoft SQL Server . . . 90

7 Démarche corrective **91**
7.1 Tests unitaires . . . 91
7.2 Tests d'intégrations . . . 91

8 Interfaces **95**
8.1 Entrée dans le système . . . 95
8.2 Sélection des modes . . . 96
8.3 Section évaluateur . . . 97
8.4 Section administration . . . 98
8.5 Section expert . . . 98

9 Conclusion **100**

III Théorie des cas problèmes **101**

1 Introduction **102**

2 Analyse du raisonnement **103**
2.1 Raisonnement de l'expert . . . 103
2.2 Raisonnement proposé : *Arbre de décision* . . . 104
2.3 Conclusion . . . 108

3 Problèmes de la maintenance **109**
3.1 Approche théorique . . . 109
3.2 Approche empirique . . . 109
3.2.1 Priorités changeantes . . . 110
3.2.2 Méthodes de test inadéquates . . . 111
3.2.3 Difficultés dans la mesure de performance . . . 111
3.2.4 Documentation du système incomplète ou non existante . . . 111
3.2.5 Adaptation à un environnement d'affaires changeant rapidement . . . 111
3.2.6 Grand arriéré . . . 111
3.3 Classification des problèmes . . . 112

4 Cas problèmes typiques **114**
4.1 Peu de formations disponibles . . . 114
4.2 Gestion des priorités changeantes . . . 120

5 Conclusion **129**

Conclusions et travaux futurs **130**

Bibliographie **131**

ANNEXE 1 : Scénarios d'utilisation **133**

ANNEXE 2 : Diagrammes de robustesse **147**

ANNEXE 3 : Publication **151**

Table des figures

1 Démarche générale de la recherche 10

1.1 Modèle de cycle de vie en cascade 13
1.2 Processus de maintenance selon l'IEEE 18
1.3 Problématiques principales en maintenance du logiciel 20

2.1 Framework Quagmire 27
2.2 Diagramme de contexte de la maintenance du logiciel, adapté de [3] 28
2.3 Classification des processus-clés de maintenance, adapté de [3] 30
2.4 Domaines de processus CMMi et $S3^m$, adapté de [3] 31
2.5 Key Process Areas de $S3^m$, adapté de [3] 32
2.6 Partie 1 des *Roadmaps* du modèle $S3^m$, adapté de [3] 33
2.7 Partie 2 des *Roadmaps* du modèle $S3^m$, adapté de [3] 33
2.8 Partie 3 des *Roadmaps* du modèle $S3^m$, adapté de [3] 34
2.9 Partie 4 des *Roadmaps* du modèle $S3^m$, adapté de [3] 34
2.10 Niveaux de maturité de $S3^m$, adapté de [3] 35
2.11 Contexte de la gestion des processus [3] 36
2.12 Interactions des KPAs de la gestion des processus [3] 37
2.13 Interactions des KPAs de la gestion des processus [3] 39
2.14 Interactions des KPAs de l'ingénierie d'évolution[3] 40
2.15 Interactions des KPAs du support à l'ingénierie d'évolution[3] 42

2.1 Schéma conceptuel de COSMICXpert 47
2.2 Schéma conceptuel de SMXpert 48
2.3 Vue au niveau du système 50
2.4 Enchaînement des tâches dans $S3^m$DSS, adapté de [22] 51
2.5 Représentation d'un système expert, adapté de [10] 53

3.1 Diagramme des cas d'utilisation de l'utilisateur 61
3.2 Diagramme des cas d'utilisation de l'administrateur 63
3.3 Diagramme des cas d'utilisation de l'expert 65

4.1 Diagramme de robustesse - Authentification 69
4.2 Diagramme de robustesse - Consultation KB 69
4.3 Diagramme de robustesse - Ajout compte 69

4.4 Diagramme de robustesse - Ajout KB 69
4.5 Diagramme de classes 70
4.6 Diagramme de séquence - Authentification 71
4.7 Diagramme de séquence - Consultation KB 72
4.8 Diagramme de séquence - Ajout utilisateur 72
4.9 Diagramme de séquence - Ajout KB 73
4.10 Diagramme de séquence - Communication inter-servlets 73
4.11 Schéma conceptuel de $S3^m$DSS 75
4.12 Diagramme ERA du système$S3^m$DSS 75

5.1 Architecture 3-Tiers 77
5.2 Composantes d'une application web, adapté de [26] 77
5.3 Diagramme de déploiement de $S3^m$DSS 78
5.4 Structure de la couche UI de $S3^m$DSS 79
5.5 Structure de la couche Business de $S3^m$DSS 80
5.6 Structure de la couche DAO de $S3^m$DSS 81

6.1 Technologies de $S3^m$DSS 83
6.2 Exemple de code HTML tiré de $S3^m$DSS 84
6.3 Exemple de code CSS tiré de $S3^m$DSS 85
6.4 Exemple de code JavaScript tiré de $S3^m$DSS 86
6.5 Exemple de code XML tiré de $S3^m$DSS 87
6.6 Exemple de code Java tiré de $S3^m$DSS 88
6.7 Exemple de code JSP tiré de $S3^m$DSS 89
6.8 Exemple de code SQL tiré de $S3^m$DSS 90

8.1 Ecran d'entrée du système $S3^m$DSS 96
8.2 Menus du système $S3^m$DSS 96
8.3 Section utilisateur du système $S3^m$DSS 97
8.4 Section administrateur du système $S3^m$DSS 98
8.5 Section expert du système $S3^m$DSS 99

2.1 Arbre de décision orienté Index 105
2.2 Arbre de décision orienté Keyword 105
2.3 Arbre de décision orienté Concept de Maintenance 105
2.4 Arbre de décision orienté Cas Problème 106
2.5 Arbre de décision orienté Theme 107
2.6 Arbre de décision orienté Recommandation 107

3.1 Classification des problèmes de la maintenance, adapté de [7] 110
3.2 Classification des problèmes en catégories, adapté de [7] 112
3.3 Associations causales, adapté de [7] 113

1 Diagramme de robustesse - Authentification 147
2 Diagramme de robustesse - Désauthentification 147

3 Diagramme de robustesse - Consultation KB 148
4 Diagramme de robustesse - Consultation aide générale 148
5 Diagramme de robustesse - Consultation aide locale 148
6 Diagramme de robustesse - Consultation définition 148
7 Diagramme de robustesse - Ajout compte 149
8 Diagramme de robustesse - Retrait compte 149
9 Diagramme de robustesse - Modification statut 149
10 Diagramme de robustesse - Modification mot de passe 149
11 Diagramme de robustesse - Modification date d'expiration 150
12 Diagramme de robustesse - Ajout KB 150
13 Diagramme de robustesse - Modification KB 150
14 Diagramme de robustesse - Retrait KB 150

Introduction

Ce mémoire traite de l'amélioration du processus de la maintenance du logiciel par un outil d'aide à la décision. En partant de bonnes pratiques regroupées en un ensemble de recommandations et proposées sous forme d'un modèle de maturité, un certain nombre de problèmes engendrés par la maintenance du logiciel peuvent être résolus. Notre contribution personnelle consiste donc premièrement en la mise en relation de ces problèmes et de ces solutions éventuelles. Cette corrélation s'effectue par une première transition de problèmes théoriques soulevés dans le référentiel SWEBOK [2] vers un ensemble de cas problèmes fréquemment rencontrés par les experts en maintenance et regroupés dans une analyse effectuée par Dekleva [7]. Deuxièmement, un effort de recherche a été réalisé afin de cerner les recommandations appropriées à des cas spécifiques parmi l'ensemble des bonnes pratiques proposées. Finalement, un outil d'aide à la décision a été implémenté afin de permettre à l'utilisateur de prendre connaissance des pratiques adéquates à suivre lors de la survenance d'un problème préçis. La démarche générale de ce mémoire s'articule donc autour d'un axe de compilation de données existantes et de mise en relation de concepts hiérarchiquement différents ainsi qu'autour d'un axe applicatif ayant permis la conception et l'implantation d'un système d'aide à la décision utile aux organisations effectuant un processus de maintenance. La figure 1 présente la démarche générale employée dans la conception de ce mémoire.

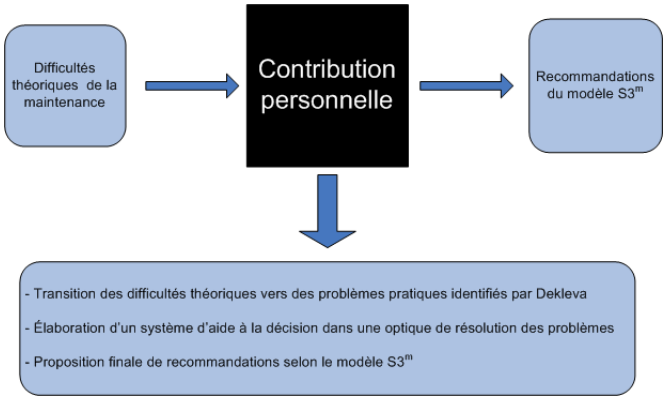


Fig. 1 – Démarche générale de la recherche

Première partie

Revue de littérature

1 Maintenance du logiciel

La plupart des évaluations montre qu'une entreprise informatique moyenne en génie logiciel dépense 60 à 70 pour cent de ses ressources globales en corrigeant, adaptant, augmentant, et restructurant ses programmes existants - une activité que nous appelons "maintenance de logiciel." Pourtant ce sujet extrêmement important suscite relativement peu d'attention dans la littérature technique. Ce manque de publication contribue à de nombreux malentendus et idées fausses. Dans cette première partie, nous décrirons précisément ce qu'est la maintenance de logiciel et définirons un certain nombre de généralités associées. Nous analyserons le processus particulier de la maintenance et distinguerons différentes problématiques liées au domaine de la maintenance du logiciel.

1.1 Fondamentaux

Dans cette section, nous traiterons de différents aspects connus ou non de la maintenance. Nous définirons ce qu'est la maintenance logicielle et où est sa place dans les modèles de cycle de vie du logiciel. Nous expliquerons pourquoi la maintenance est nécessaire et pourquoi certains éléments la rendent difficile à entreprendre. Nous définirons les acteurs les plus à même de travailler sur la maintenance d'un produit et finalement nous expliquerons pourquoi celle-ci est très coûteuse.

1.1.1 Définitions

Apporter une définition claire et précise de la maintenance logicielle est toujours une tâche assez difficile. De nombreux auteurs s'y sont essayés et ont contribué à quelques définitions traditionnellement acceptées en maintenance du logiciel. Voici quelques-uns de ces essais :

> *"changes that have to be made to computer programs after they have been delivered to the customer or user."* [16]

> *"... the performance of those activities required to keep a software system operational and responsive after it is accepted and placed into production."* [9]

"Maintenance covers the life of a software system from the time it is installed until it is phased out." [29]

"Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment." [11]

"... software product undergoes modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify existing software product while preserving its integrity." [13]

Plus généralement, on pourra définir la maintenance comme :

"The totality of activities required to provide cost-effective support to a software system. Activities are performed during the predelivery stage as well as the postdelivery stage. Predelivery activities include planning for postdelivery operations, supportability, and logistics determination. Postdelivery activities include software modification, training, and operating a help desk." [20]

1.1.2 Cycle de vie du logiciel

La maintenance fait partie intégrante du cycle de vie du logiciel. Quelque soit le modèle qui représente ce cycle, on constatera toujours que la maintenance en est le maillon final et considérée comme une activité d'après livraison du produit. La figure 1.1 montre le modèle de cycle de vie du logiciel en cascade.

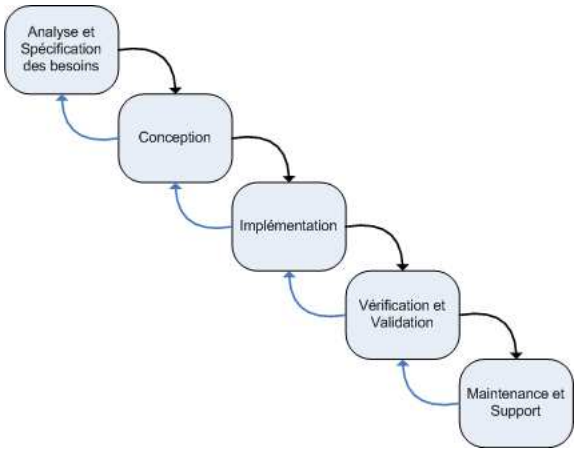


FIG. 1.1 – Modèle de cycle de vie en cascade

1.1.3 Nécéssité d'entreprise

Pendant la dernière phase du cycle de vie du logiciel, les utilisateurs commencent à travailler avec leur nouveau système. Dans certains cas, il s'agit pour ceux-ci d'une transition d'un système manuel vers un système automatique. Dans d'autres cas, de nombreux changements sont mis en place mais dans tous les cas, les utilisateurs doivent être formés à l'utilisation de leur nouveau système. L'utilisateur pourra trouver des choses qui ne fonctionnent pas correctement ou il désirera que de nouvelles fonctionnalités soient ajoutées au système. Ces remarques sont formulées aux mainteneurs du système dont le travail consiste à corriger ou améliorer le système. Les mainteneurs effectuent les corrections et adaptations approuvées, mettent en place les changements et l'utilisateur apprend à nouveau à utiliser le système. Ce cycle forme ce que l'on qualifie de *boucle de la maintenance* et permet de faire vivre un produit de nombreuses années. On parlera généralement d'une durée de vie de 15 à 20 ans pour un système.

Lehman [15] met en avant 2 lois de l'évolution des logiciels qui permettent d'expliquer pourquoi la phase de maintenance peut être considérée comme la plus longue phase du cycle de vie du logiciel. Sa première loi est qualifiée de "Law of Continuing Change", c'est-à-dire que le système doit nécessairement changer afin d'être toujours utile pour ses usagers. La seconde loi "Law of Increasing Complexity" signifie que la structure d'un programme se détériore à mesure que celui-ci évolue. Au fil du temps, la structure du code d'un programme se dégrade jusqu'à ce qu'il devienne plus rentable de le réécrire.

1.1.4 Difficultés inhérentes

Un constat fréquent est qu'il est assez difficile d'entreprendre la maintenance d'un système. Ceci est largement dut au fait que la phase de maintenance est une activité se déroulant après la livraison d'un produit. Schneidewind [23] établit que :

- Nous ne pouvons tracer le produit ni même le processus qui a créé le produit ;
- Les changements ne sont pas documentés de manière adéquate ;
- Il est difficile de suivre les changements ;
- Il y a un effet ondulatoire lorsque des changements sont effectués.

La combinaison d'une pauvre documentation logicielle et d'une complexité élevée d'un système rend le travail de maintenance souvent très difficile. Dans la plupart des cas, l'utilisateur d'un système considère les problèmes logiciels comme étant similaires aux problèmes techniques. Si un problème survient, il n'y a qu'à le régler. Beaucoup d'utilisateurs ont l'impression qu'ils peuvent demander n'importe quelle modification ou amélioration de système, pensant que les changements seront aisés. Ils ne comprennent pas que la plupart des actions de maintenance demande des changements majeurs dans la structure d'un programme. Les utilisateurs s'attendent à obtenir des résultats dans des délais similaires aux problèmes techniques. Ils s'attendent à ce que le logiciel soit comme le matériel et c'est cela le problème. Il est important de faire comprendre à l'utilisateur que la maintenance est bien plus que de fixer un bogue.

1.1.5 Acteurs de la maintenance

L'exécution de la maintenance d'un logiciel peut être réalisée par les développeurs du produit ou pas une organisation différente. Pigoski [19] décrit les avantages et inconvénients des deux approches. Il y a un certain nombre d'avantages à laisser la maintenance d'un logiciel à son développeur :

- Le développeur a la meilleure connaissance du système ;
- Il n'est pas nécessaire d'élaborer une documentation ;
- Un système de communication formel entre les mainteneurs et les développeurs ne doit pas être établi ;
- Les utilisateurs du système ne travaillent qu'avec une seule entreprise ;
- Une diversité dans le travail est engendrée, satisfaisant d'autant plus le personnel de l'entreprise de développement.

Malgré tout, un certain nombre d'inconvénients sont à déplorer dans cette approche :

- Le personnel peut vouloir quitter l'entreprise de développement si la maintenance devient trop lourde ;
- Les nouveaux employés peuvent être mécontents suite à la quantité de travail de maintenance ;
- Si les experts développeurs quittent l'entreprise, les personnes responsables de la maintenance peut ne pas être formés de manière adéquate ;
- Les développeurs passent souvent trop de temps à parfaire leur système ;
- Les développeurs initiaux sont souvent réassignés à de nouveaux projets ou à des projets prioritaires.

Similairement, un certain nombre d'avantages et d'inconvénients peuvent être pointés dans l'approche de transition d'un processus de maintenance à une entreprise extérieure. Pigoski [19] décrit ceux-ci :

- Une meilleure documentation est générée ;
- Des procédures formelles de transition sont créées ;
- Les mainteneurs connaissent les forces et les faiblesses du système ;
- Des procédures d'implémentation des changements sont établies ;
- Le moral est amélioré et les personnes désireuses d'effectuer de la maintenance réalisent un meilleur travail.

Dans cette approche, un certain nombre de désavantages sont aussi pointés :

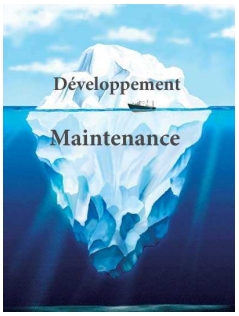
- La transition du système peut être lente ;
- Le moral peut baisser si de mauvaises personnes sont assignées au projet ;
- Il y a un réel besoin de formation ;
- Des problèmes de financement peuvent être rencontrés ;
- Prendre connaissance du système nécessite un certain temps ;

– Mettre en place l'entreprise de maintenance et ses équipements prend un certain temps ;
– Le support des utilisateurs peut en pâtir ;
– La crédibilité du support des utilisateurs peut en souffrir.

Etant donné ces avantages et inconvénients respectifs, aucune réponse type à la question posée ne peut être proposée. Comme dans beaucoup de situations, la gestion de la maintenance doit être réfléchie en fonction du contexte. Chaque entreprise doit peser le pour et le contre de ces avantages et inconvénients et les appliquer à leur environnement. Martin et McClure [16] ajouteront que les grandes entreprises devraient mettre en place une équipe de maintenance séparée afin d'améliorer le contrôle et la productivité.

1.1.6 Problème de coût

Quel que soit le modèle de cycle de vie du logiciel utilisé, la maintenance consomme une part majeure des ressources financières du cycle de vie. Différentes études faites depuis les années 70 ont montré que la part du budget allouée à la maintenance n'a cessé de croître. Dans le courant des années 90, une firme spécialisée en recherche marketing, le Gartner Group, a estimé que l'ensemble des entreprises américaines dépense environ 30 milliards de dollars annuellement pour la maintenance des systèmes.



On estime également que 45 % du budget total des entreprises américaines est alloué à la maintenance des logiciels. Les raisons de ce coût très élevé sont multiples et souvent mal comprises. En voici une liste non exhaustive :

– Schneidewind [23] relève que les programmeurs coûtent cher et qu'ils passent la plupart de leur temps de travail sur des problèmes de maintenance ;
– Osborne et Chikofsky [18] mettent en avant le fait que beaucoup d'entreprises paient actuellement le prix de développements pauvrement pensés et mal documentés par le passé ;
– Pigosky [20] parle des responsabilités des utilisateurs, ne sachant pas ce qu'ils veulent et bombardant sans cesse les mainteneurs de requêtes de changement ;
– Lehman [15] considère que les exigences de début de projet ne sont plus nécessairement les mêmes quelques années plus tard ;
– Pigosky [20] ajoute que le départ de commanditaires de projets provoque des incohérences dans la perception des fonctionnalités nécessaires aux nouveaux usagers ;

– On pourra noter une perte de temps dans la compréhension de code pauvrement documenté. Sachant que 40 à 60 % du temps de travail est employé à essayer de comprendre les programmes.

1.2 Catégories de la maintenance

Swanson [25] est le premier, en 1976, à examiner en profondeur la phase de maintenance. Il met en valeur trois catégories différentes de "difficultés" pouvant être résolues par un processus de maintenance. Ces trois catégories sont les suivantes :

– **Corrective :** Tout changement rendu nécessaire par des erreurs réelles (bugs induits ou résiduels dans un système) ;
– **Adaptative :** Tout effort lancé en raison de changements dans l'environnement dans lequel un système logiciel doit fonctionner ;
– **Perfective :** Tous les changements, insertions, suppressions, modifications, extensions, et perfectionnements faits à un système pour satisfaire les besoins d'évolution et/ou d'extension de l'utilisateur.

1.2.1 Maintenance corrective

La maintenance *corrective* fixe les différents bogues d'un système. Le système peut ne pas fonctionner comme il devrait le faire selon ses plans de conception. L'objectif de la maintenance corrective est donc ici de localiser les spécifications originelles afin de déterminer ce que le système était initialement sensé faire.

1.2.2 Maintenance adaptative

La maintenance *adaptative* fixe les différents changements qui doivent être faits afin de suivre l'environnement du système en perpétuelle évolution. Par exemple, le système d'exploitation peut être mis à jour et des modifications peuvent être nécessaires afin de s'accommoder à ce nouveau système d'exploitation. Malheureusement, l'utilisateur ne visualise pas les changements directs dans son application malgré les efforts du mainteneur.

1.2.3 Maintenance perfective

La maintenance *perfective* ou d'amélioration comprend tous les changements faits sur un système afin de satisfaire aux besoins de l'utilisateur. Si quelque chose n'était pas dans la conception originelle ou dans les spécifications du système et que l'utilisateur désire l'y ajouter, on parlera de maintenance perfective. Toutes les nouvelles demandes des utilisateurs sont de ce type.

1.2.4 Maintenance préventive

En plus de ces trois catégories de maintenance, l'IEEE a défini un quatrième type de maintenance : la maintenance *préventive*. Elle peut être définie comme la maintenance

executée afin d'empêcher des problèmes avant qu'ils surviennent [11]. Ce type de maintenance peut être considéré comme très important dans le cas de systèmes critiques tels que ceux liés à l'aéronautique par exemple.

1.3 Processus de la maintenance

1.3.1 En théorie

Les modèles de cycle de vie du logiciel sont assez bien connus mais peu de gens sont au courant de l'existence de modèles propres au processus de la maintenance. En 1993, l'IEEE sur une idée de Schneidewind [24] a publié un processus iteratif standard pour la gestion et l'exécution des activités de maintenance logicielle sous le nom de "IEEE Standard for Software Maintenance [11]". Ce modèle de processus comprend un ensemble d'entrées, de processus, de contrôles et de sorties en terme de maintenance. Le standard initie le processus de maintenance dès l'étape de livraison du produit à un utilisateur. La figure 1.2 présente le processus de maintenance selon l'IEEE. Une description des différentes phases s'en suivra.

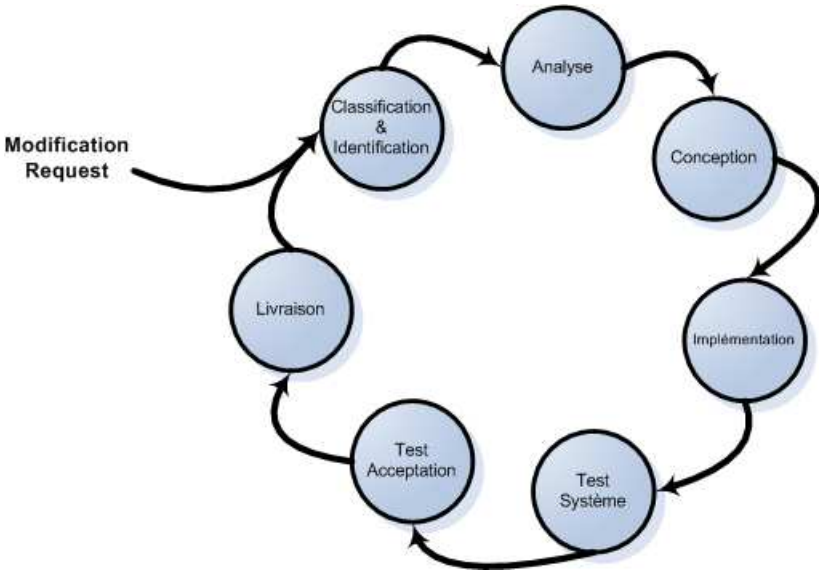


FIG. 1.2 – Processus de maintenance selon l'IEEE

Classification et Identification

Le processus démarre par la soumission d'une requête de modification (MR). L'entreprise de maintenance détermine tout d'abord le type de requête dont il s'agit (corrective, perfective ou adaptative). Une priorité est ensuite assignée à cette requête ainsi qu'un numéro unique. Toutes ces informations sont entrées automatiquement dans une base de données qui pourra être consultée tout au long du cycle.

Analyse

Une étude de faisabilité ainsi qu'une analyse détaillée sont effectuées durant cette phase. Le mainteneur prêtera attention aux impacts des modifications, il proposera plusieurs solutions alternatives et définira les coûts. L'analyse détaillée fournira différentes informations telles que des plans d'implémentation, des stratégies de test ou encore des identifications de fonctionnalités. A la fin de cette période d'analyse, l'ensemble des changements doit être totalement planifié.

Conception

Durant la phase de conception, l'ensemble des informations collectées pendant la phase précédente sont rassemblées et utilisées afin de construire le design de l'application. En fin de conception, la documentation du logiciel doit être mise à jour (plans de tests, design, analyse et exigences).

Implémentation

La phase d'implémentation comprend l'ensemble des processus de codage, de tests unitaires, d'intégration, d'analyse des risques et de revue de code. Toutes les documentations concernant le logiciel sont mises à jour.

Test Système

La phase des tests du système s'assure que l'ensemble des exigences originelles de l'utilisateur sont toujours rencontrées de même que les nouvelles ajoutées. La phase de tests est l'une des plus importante du cycle car elle permet de s'assurer que le nouveau produit est toujours satisfaisant et que les nouveaux changements ont été implémentés correctement.

Test d'acceptation

La phase de test d'acception s'établit sur un système totalement intégré et est réalisée par les clients, utilisateurs ou par un tiers parti. Les testeurs ont pour objectif de rapporter les erreurs et de déterminer si les fonctionnalités du système sont en accord avec leurs exigences.

Livraison

Durant cette phase finale, le fournisseur livre le nouveau système aux utilisateurs. Il effectue la configuration du système et la formation des utilisateurs. Après cette phase, le système est prêt à fonctionner de manière autonome.

1.3.2 En pratique

Bien que les standards tels que le processus de maintenance proposé par l'IEEE soient disponibles depuis près de 20 ans, nombreuses sont les entreprises qui ne travaillent pas encore avec un processus de maintenance bien défini. Schneidewind [23] justifie cela de la manière suivante :

> "The traditional view of the software development cycle has done a great disservice to the maintenance domain by representing it as one step at the end of the cycle."

Gageons néanmoins que dans les années à venir, ce processus ou ses dérivés prendra application dans le monde industriel tout comme le modèle du cycle de vie du logiciel l'a fait.

1.4 Problématiques principales de la maintenance

Afin d'assurer une maintenance efficace, un certain nombre de problématiques doivent être traitées. Comprenons que la maintenance soulève des défis uniques pour les ingénieurs du logiciel. La section suivante présente quelques-uns des problèmes théoriques principaux liés à la maintenance du logiciel. La figure 1.3 présente les différents sujets groupés.

Problèmes techniques
Compréhension limitée
Essai
Analyse d'impact
Maintenabilité
Problèmes de management
Alignement avec les objectifs de l'organisation
Embauche du personnel
Processus
Aspects organisationnels de la maintenance
Outsourcing
Estimation des coûts de la maintenance
Estimation des coûts
Modèles paramétrés
Expérience
Mesure de la maintenance du logiciel
Mesures spécifiques

FIG. 1.3 – Problématiques principales en maintenance du logiciel

Par la description de ces problématiques, le lecteur comprendra la complexité de la maintenance et la réelle nécessité d'un système d'aide à la décision tel que $S3^m$DSS. Cette section est donc présentée en guise d'ouverture vers des problématiques concrètes et traitées dans le système $S3^m$DSS. Le lecteur trouvera plus d'information à ce sujet au fil de la lecture de ce mémoire.

1.4.1 Problèmes techniques

Compréhension limitée

Par "compréhension limitée", on entend la rapidité avec laquelle quelqu'un peut comprendre ou faire un changement ou une correction dans un logiciel sur lequel cet individu n'a pas travaillé. Les études montrent que de 40 à 60% de l'effort de maintenance est consacré à la compréhension du logiciel à modifier. Il s'agit donc d'un sujet d'intérêt pour les mainteneurs. On notera que la compréhension devient des plus difficile lors de l'analyse d'un code source par exemple ou plus généralement lors de l'analyse de représentations orientées texte. Cette difficulté est accrue par la non disponibilité fréquente des développeurs initiaux. Ainsi les mainteneurs ont généralement une compréhension limitée du logiciel et un effort non négligeable doit être fait pour accroitre leur propre compréhension du logiciel [2].

Essai

Réaliser un essai complet sur un morceau majeur de logiciel est généralement significatif en terme de temps et de coûts. Trouver du temps pour réaliser des essais est souvent très difficile, sans parler du défi de coordination des essais lorsque différents membres de l'équipe de maintenance travaillent sur différents problèmes en même temps. On constatera, d'ailleurs, l'impossibilité fréquente d'effectuer des tests sur des fonctions critiques d'un logiciel nécessitant la mise hors-ligne du système [2].

Analyse d'impact

Une analyse d'impact décrit comment mener, à un coût efficace, une analyse complète des impacts d'un changement dans un logiciel existant. Par une connaissance de la structure et du contenu d'un logiciel, les mainteneurs réalisent l'analyse d'impact, identifient tous les produits des systèmes et des logiciels affectés par une demande de modification de logiciel et établissent une estimation des ressources nécessaires au changement. On notera que le vocabulaire de la maintenance parlera de demande de modification (MR pour Modification Request) ou rapport de problème (PR pour Problem Report) [2]. Arthur [4] définit les objets de l'analyse d'impact comme :

- Détermination de la portée d'un changement pour établir un plan et implanter le travail ;
- Développement d'estimations justes des ressources nécessaires pour effectuer le travail ;
- L'analyse de coûts/bénéfices du changement demandé ;
- Communication aux autres de la complexité d'un changement donné.

Cette analyse est, dans la plupart des cas assez difficile à entreprendre compte tenu de la complexité des systèmes.

Maintenabilité

L'IEEE [11] définit la maintenabilité comme l'aisance avec laquelle un logiciel peut être maintenu, amélioré, adapté ou corrigé pour satisfaire aux exigences spécifiées. Ces sous-caractéristiques de maintenabilité doivent être spécifiées, revues et contrôlées durant les activités de développement logiciel pour réduire les coûts de maintenance. Un succès amenant la qualité du logiciel à s'améliorer est difficile à atteindre car les sous-caractéristiques de maintenabilité ne sont généralement pas respectées durant le processus de développement. On comprendra que les développeurs sont assez préoccupés par une foule d'autres choses et négligent donc les exigences du mainteneur. La conséquence directe de cette attitude est un manque cruel de documentation du système. On notera finalement que la présence de processus normalisés et matures aident souvent à améliorer la maintenabilité d'un système [2]. Cette approche sera explicitée dans la partie consacrée au modèle $S3^m$.

1.4.2 Problèmes de management

Alignement avec les objectifs de l'organisation

On considère les objectifs de l'organisation comme le retour sur investissement des activités de maintenance. Bennett [5] énonce que : "le développement logiciel est habituellement basé par projet, avec une période de temps et un budget définis. L'emphase est mise sur la livraison dans les délais et dans le budget pour remplir les besoins de l'utilisateur. En contraste, la maintenance de logiciel a souvent comme objectif d'allonger la vie d'un système logiciel le plus longtemps possible. En plus, elle peut être conduite par le besoin de satisfaire la demande de l'utilisateur pour des mises à jour et des améliorations. Dans les deux cas, le retour sur investissement est bien moins clair : donc la vue par les gestionnaires seniors est souvent celle d'une activité majeure consommant de grandes ressources sans bénéfices clairs et quantifiables pour l'organisation."

Embauche du personnel

La maintenance n'est généralement pas vue comme un travail séduisant. Le personnel de la maintenance est souvent vu comme un personnel de seconde classe, ceci amenant une baisse de moral fréquente. L'embauche du personnel vise dès lors à attirer et à garder ses travailleurs dans son secteur [2].

Processus

Par la notion de processus, on comprend l'unicité de certaines activités de maintenance. On notera les nombreuses activités en commun avec le développement de logiciel mais néanmoins il existe plusieurs activités qu'on ne trouve pas dans le développement logiciel et posant un certain nombre de défis au manager [2].

Aspects organisationnels de la maintenance

Les aspects organisationnels décrivent comment identifier quelle organisation et/ou fonction sera responsable pour la maintenance du logiciel. Deux options principales sont proposées : les organisations de génie logiciel peuvent demeurer avec le développeur originel ou s'adresser à une équipe séparée. Nombreux sont les arguments pour et contre chacune des options, ils ont été présentés en section 1.1.5. Le choix sera fait sur base du cas par cas.

Externalisation

L'externalisation ou *Outsourcing* devient une entreprise de plus en plus en vogue et la maintenance entre également dans ce créneau. On notera que les logiciels les moins critiques sont le plus souvent sujets à l'externalisation par crainte de perdre le contrôle de logiciels utilisés dans le coeur des affaires. Un des défis majeurs pour ceux qui fournissent le service externalisé est de déterminer la portée des services de maintenance requis et les détails contractuels. Différentes études montrent que 50% des fournisseurs de services externalisés proposent ces services sans aucun accord clair. Un autre défi identifié est la transition du logiciel vers le fournisseur de services externalisés [2].

1.4.3 Estimation des coûts de la maintenance

Estimation des coûts

Les estimations des coûts de maintenance sont affectés par beaucoup de facteurs techniques et non techniques. ISO/IEC 14764 énonce que "les deux approches les plus populaires pour l'estimation des ressources pour la maintenance de logiciel sont l'utilisation de modèles paramétrés et l'utilisation de l'expérience". On parlera le plus souvent d'une combinaison des deux [2].

Modèles paramétrés

Les modèles paramétrés se basent sur les données des projets et discutent de tous les aspects de l'estimation des coûts, incluant les points de fonction [2].

Expérience

L'expérience, par le jugement d'expert est une approche utilisée pour enrichir les modèles paramétrés. Elle se complète par différentes analogies et structures de décomposition de la tâche. On mettra en avant que la meilleur approche pour l'estimation de la maintenance est une combinaison d'expérience et de données empiriques fournies par différents programmes de mesure [2].

1.4.4 Mesure de la maintenance du logiciel

Il existe un certain nombre de mesures du logiciel communes à tous les domaines du génie logiciel. Ces mesures peuvent être un bon point de départ pour le mainteneur.

Mesures spécifiques

Différents auteurs présentent des mesures spécifiques à la maintenance. Ces mesures incluent des techniques standards pour chacune des quatre sous-catégories de maintenabilité [2] :

- *Analysabilité* : mesure de l'effort du mainteneur ou des ressources utilisées pour essayer de diagnostiquer les déficiences ou les causes des pannes ou pour identifier les parties à modifier ;
- *Changeabilité* : mesure de l'effort du mainteneur associé à l'implantation d'une modification spécifiée ;
- *Stabilité* : mesure les comportements inattendus du logiciel, incluant ceux rencontrés lors des essais ;
- *Essaiabilité* : mesure de l'effort du mainteneur et de l'utilisateur pour essayer de faire des essais sur le logiciel modifié.

1.5 Conclusion

Par le présent chapitre, nous avons défini les bases fondamentales de la maintenance. Nous avons distingué différentes catégories de maintenance existantes. Une description du processus standard de la maintenance a également été proposée. Finalement, nous avons décrit différentes problématiques propres au domaine de la maintenance et par là nous avons formé un lien vers la nécessité de fournir un outil d'aide à la décision permettant de contrer ces difficultés. Ces problématiques sont encore assez théoriques pour l'instant mais nous tenterons ultérieurement de les faire correspondre avec des cas problèmes typiques du monde de la maintenance. Le chapitre suivant proposera une description sous forme de modèle de maturité de diverses recommandations directement utilisables par les mainteneurs.

2 Sofware Maintenance Maturity Model : $S3^m$

Ce chapitre présente le modèle de maturité $S3^m$ pour *Software Maintenance Maturity Model*. Il s'agit d'un modèle structuré et organisé en niveaux de maturité qui présente un nombre important de pratiques exemplaires en maintenance ainsi que leurs références. Nous décrirons, tout d'abord, ce qu'est un modèle de maturité et pourquoi il est nécessaire de prendre en compte le modèle présenté. Nous définirons ensuite les fondations du modèle $S3^m$ en précisant son contexte d'émergence, en élaborant une classification des différents processus de la maintenance et en identifiant différents domaines de processus. Une description de la sémantique des différents niveaux de maturité est également présentée. Finalement, nous préciserons les domaines de processus propres au modèle de même que les interactions existantes entre ceux-ci.

2.1 Modèles de maturité en génie logiciel

Afin d'implémenter un processus de production de type industriel, le développement d'un processus contrôlable et répétable est nécessaire. Nelson [17] parle de *routine* ce qui réfère à :

> "predictable behavior models which are observable within organisations, from well defined technical procedures to more general policies and strategies."

Une organisation est considérée comme immature si ses routines ne sont pas prédéfinies mais plutôt définies sur le tas par le personnel et la direction. En contraste, une organisation est considérée comme mature si ses routines sont bien définies et communiquées aux employés.

Depuis une dizaine d'années, se poursuit un intérêt grandissant en matière d'évolution des processus et dans la poursuite de solutions aux problèmes orientés logiciel. Ceci a conduit les auteurs à un développement de modèles de référence de l'évolution des processus en niveaux de maturité croissants. Deming, Imai, Juran et Crosby sont considérés comme les pionniers du domaine, considérant que la qualité d'un produit est directement reliée à la qualité du processus sous-jacent. Selon ce paradigme, il est précisé que le meilleur moyen d'améliorer la qualité d'un logiciel est d'améliorer ses processus de développement et de maintenance. Ainsi afin de passer de processus plus ou moins chaotiques vers des

processus matures, un chemin d'évolution structuré est proposé et décrit sous forme d'un modèle de maturité.

Un modèle de maturité est conçu sur base de [3] :

- un consensus d'experts ;
- une connaissance des architectures des différents modèles existants ;
- des références à des standards et documents approuvés ;
- une addition de pratiques particulières et dédiées à un domaine spécifique ;
- une rationalisation de la classification de chaque pratique dans un niveau de maturité spécifique ;
- une expérimentation pour l'ajustement et l'amélioration du modèle.

L'architecture d'un modèle de maturité peut être de deux types : *continuous* ou *staged*. Une approche continue est typiquement utilisée pour des processus d'amélioration tandis qu'une approche par niveau est typiquement utilisée pour l'évaluation des fournisseurs.

La représentation continue propose les caractéristiques suivantes :

- permet de sélectionner la séquence d'amélioration rencontrant les objectifs du business et facilitant une meilleur gestion des risques ;
- permet la comparaison entre entreprises (par zone de processus).

La représentation par niveau propose les caractéristiques suivantes :

- suite documentée et ordonnée d'améliorations, démarrant de pratiques fondamentales et progressant à travers des niveaux successifs, chacun étant vu comme fondation du niveau suivant ;
- comparaisons internes et interentreprises possibles dans le cas de même approche par niveau ;
- classement facile résumant les évaluations et comparaisons entre entreprises.

Depuis les années 80, l'industrie et la communauté scientifique ont proposé un certain nombre de modèles de maturité. Sheard [28] présente, en figure 2.1, les modèles les plus connus et leurs standards.

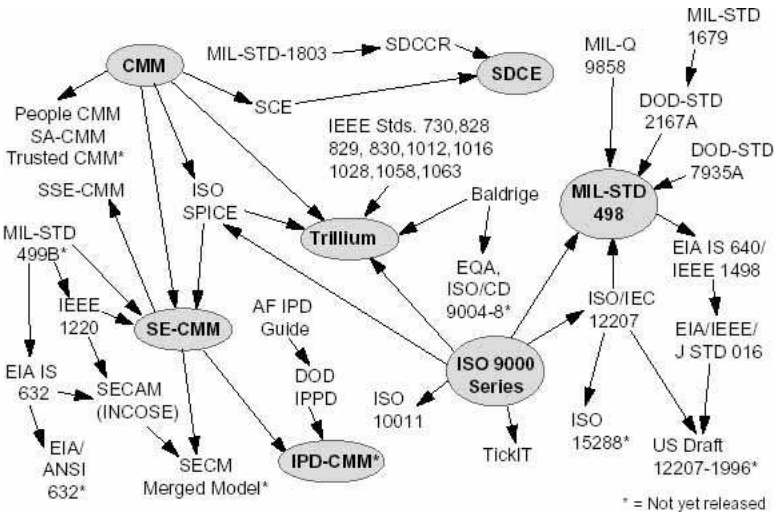


FIG. 2.1 – Framework Quagmire

La littérature traitant de maintenance du logiciel confirme que certains processus de maintenance sont uniques aux mainteneurs et ne font pas partie du processus classique de développement d'un logiciel. Une analyse de chacun de ces modèles a permis d'identifier un certain nombre de recommandations utiles aux mainteneurs mais aucun de ces modèles de maturité ne couvre l'entièreté des activités de maintenance, à savoir : la gestion de problèmes, l'acceptation des logiciels, la gestion des transitions du développement à la maintenance, l'établissement des SLAs, la planification des activités de maintenance, la gestion des requêtes d'événements et de services, le support quotidien des opérations et le rajeunissement de logiciel. Le lecteur comprendra dès lors la nécessité de concevoir un modèle de maturité propre à la maintenance de logiciels.

Le modèle de maturité $S3^m$ puise donc une grande partie de ses recommandations finales dans la littérature actuelle, reprenant les différents éléments déjà validés par les experts et les associant à des niveaux de maturité définis. Dans cette optique de recherche de recommandations existantes dans les modèles de maturité de la littérature, April [3] se base sur les caractéristiques suivantes :

- leurs pratiques détaillées sont documentées et disponibles ;
- les pratiques du modèle référencent directement ou indirectement la maintenance de logiciel ;
- le modèle et son utilisation sont récents ;
- le modèle est déployé et connu ;
- le modèle n'est pas orienté spécifiquement pour l'industrie, ce qui le rendrait non pertinent pour un public plus large.

Notons finalement que le modèle de maturité $S3^m$ comporte également un certain nombre de recommandations nouvelles imaginées par les auteurs.

2.2 Fondations du modèle $S3^m$

2.2.1 Contexte de la maintenance de logiciel

Avant d'entamer une description du modèle, il est important de comprendre le contexte dans lequel se trouve l'activité de maintenance de logiciel dans lequel les mainteneurs évoluent. On dénombre cinq interfaces avec lesquelles le processus doit traiter dans le contexte typique d'une organisation de maintenance :

- interface clients et utilisateurs (label 1) ;
- interface *up-front* et *help desk* (label 2) ;
- interface opérations (label 3) ;
- interface développeurs (label 4) ;
- interface fournisseurs (label 5) ;

La figure 2.2 présente le contexte de la maintenance en génie logiciel :

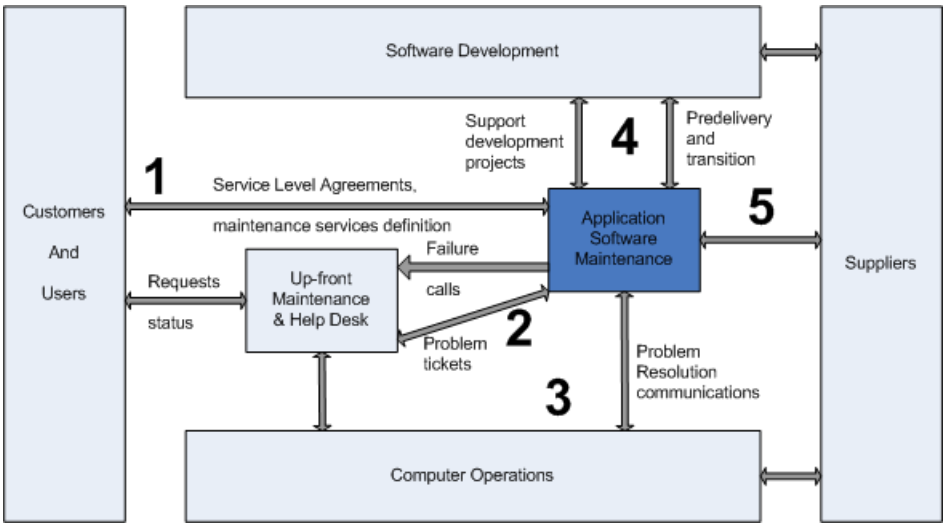


FIG. 2.2 – Diagramme de contexte de la maintenance du logiciel, adapté de [3]

La première interface décrit les relations entre les mainteneurs et les clients et/ou utilisateurs et consiste typiquement en la définition et la gestion des services de maintenance proposés. Ceci inclut la portée des services, les objectifs et priorités, le budget et les activités relatives à la satisfaction du client. Ces éléments sont regroupés en un accord contractuel nommé *Service Level Agreement* (SLA).

La seconde interface décrit le service de *help desk* qui est une pratique standard que les entreprises en IT/IS proposent afin de centraliser les demandes des utilisateurs. Une requête d'utilisateur, appelée billet, circulera typiquement entre le *help desk*, le service de maintenance et le service d'opérations avant qu'un problème soit clairement identifié. Ce

mécanisme de billets assure une communication efficace tentant de garantir une résolution rapide de problèmes.

La troisième interface décrit le service des opérations garantissant le support par l'infrastructure de l'application. Ce service supporte typiquement toutes les opérations associées aux stations de travails, réseaux et plateformes. Ils réalisent les opérations de sauvegarde et de restauration ainsi que l'administration des systèmes.

La quatrième interface décrit la relation entre les développeurs et les mainteneurs. Cette interface peut ne pas exister dans le cas où la maintenance est directement effectuée par les développeurs mais malgré tout beaucoup d'organisations choisissent de séparer le développement de la maintenance.

La cinquième interface s'oriente vers les relations avec les fournisseurs et "outsourcers".

2.2.2 Classification des processus de la maintenance du logiciel

Afin de développer un modèle de haut niveau en maintenance du logiciel, la première idée est d'identifier et de modéliser un ensemble complet de catégorie de processus, comme le suggèrent Wang et King [30]. Ce modèle doit regrouper tous les standards internationaux en terme de processus et d'activités. L'idée est donc de fournir une représentation similaire au standard ISO12207 mais orientée activités et processus de maintenance. Cette classification est décomposée en 3 classes :

- Processus primaires : processus liés à l'entreprise technique ;
- Processus de support : processus liés à l'entreprise de support ;
- Processus organisationnels : processus liés aux organisations telles IS, GRH, Finance, etc.

La figure 2.3 présente cette classification :

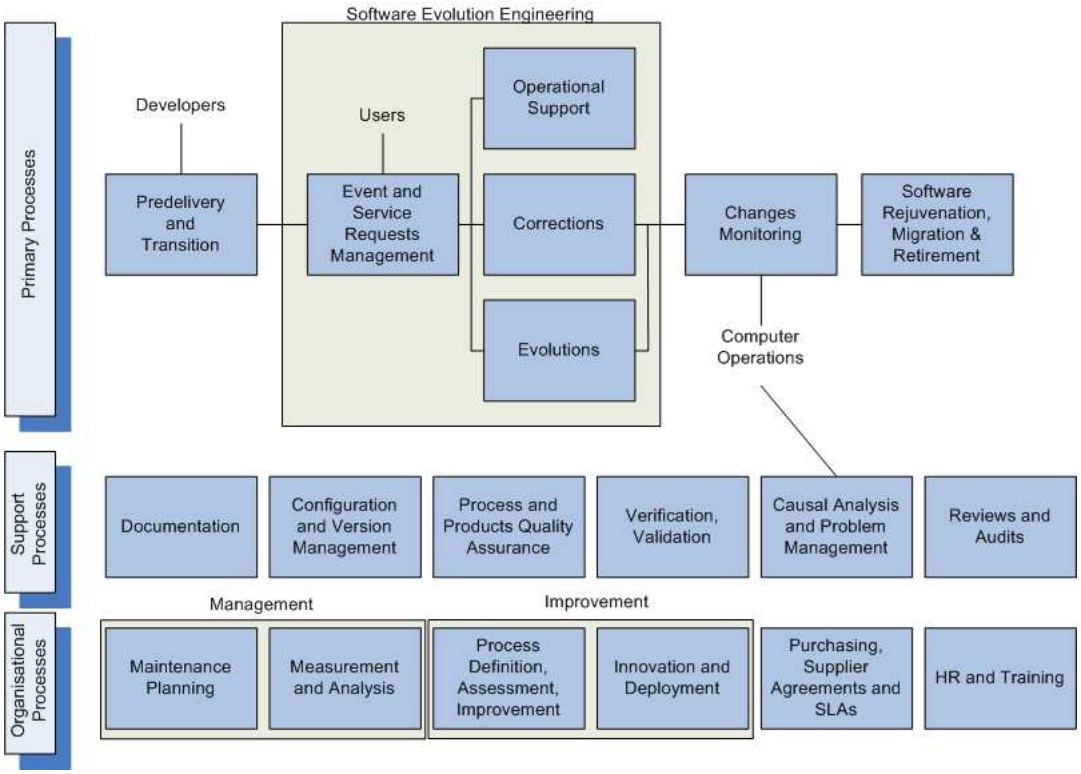


FIG. 2.3 – Classification des processus-clés de maintenance, adapté de [3]

Les processus opérationnels (notés également primaires) utilisés par l'entreprise de maintenance du logiciel débutent au démarrage d'un projet de développement par le processus de *Pre-delivery and Transition*. Une fois que le logiciel est entre les mains de l'équipe de maintenance, le processus de *Event and Service Request Managament* s'occupe des impacts quotidiens (PRs, MRs et requêtes de support). Ceux-ci sont redirigés vers les services adéquats en fonction du type de contrat des clients, de la nature de la requête et de sa taille. Lorsqu'une requête est acceptée, elle est assignée à l'un des services suivants : *Operational Support*, *Corrections* ou *Evolution* en fonction du type de maintenance à effectuer. Ensuite par le processus de *Change Monitoring*, un contrôle de l'application est effectué afin de s'assurer qu'aucune dégradation de la qualité du logiciel ne s'est produite.

Les processus de support sont réalisés par les développeurs, mainteneurs et opérateurs de l'entreprise. Ces processus incluent typiquement la documentation de processus ou encore la validation et vérification de processus. On notera que lorsqu'un problème survient et qu'il ne peut être identifié clairement, un processus d'analyse causale sera lancé.

Les processus organisationnels sont suivis par des départements de l'entreprise tels que la gestion des ressources humaines ou le département d'innovation. On notera que même s'il est important pour le mainteneur d'évaluer et de mesurer ces processus, il est plus important de définir et d'optimiser les processus opérationnels en premier lieu.

2.2.3 Identification des domaines de processus

La figure 2.4 présente les domaines de processus du CMMi comparativement à ceux proposés par $S3^m$. On remarquera les différences entre les deux modèles et notamment le fait que la maintenance du logiciel est centrée sur l'évolution du logiciel et non sur la phase de conception initiale.

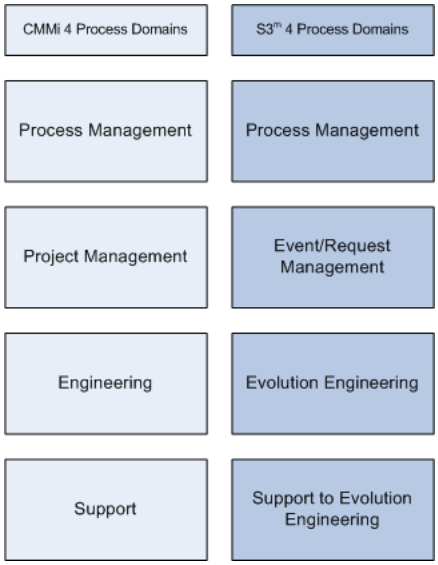


FIG. 2.4 – Domaines de processus CMMi et $S3^m$, adapté de [3]

Une *Key Process Area* (KPA) est un ensemble de pratiques clés (activités contribuant le plus à un processus) concernant un même domaine d'activités. Le modèle de maturité trie donc les pratiques clés en groupe logique : les *Key Process Area*. La figure 2.5 présente les processus clés que l'on peut retrouver dans chaque domaine de processus du modèle $S3^m$ et regroupés en KPA.

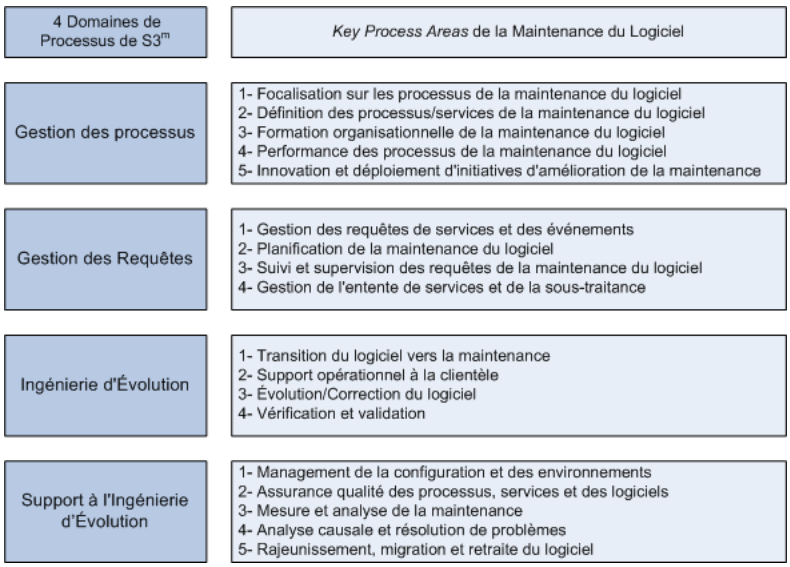


FIG. 2.5 – Key Process Areas de $S3^m$, adapté de [3]

2.2.4 Le concept de *Roadmap*

Une *roadmap* peut être définie comme un ensemble de pratiques liées et pouvant couvrir différents niveaux de maturité. L'ordre de ces pratiques clés est déterminé par un consensus d'experts s'accordant sur leur enchaînement et degré de priorité dans un processus spécifique. Notons également que les pratiques clés les plus fondamentales sont positionnées à un niveau de maturité le plus bas tandis que les pratiques clés les plus difficiles sont positionnées à des niveaux de maturité les plus élevés. Il y a donc une progression dans la complexité des recommandations en fonction du niveau du maturité.

L'architecture du modèle $S3^m$ s'articule donc autour de *domaines de processus* qui sont décomposés en *Key Process Area*. Ces derniers sont décomposés en *roadmaps*, elles-mêmes composées de *pratiques élémentaires* appartenant à l'un ou l'autre niveau du modèle. On notera que le modèle de maturité $S3^m$ possède 4 domaines de processus, 18 KPAs, 74 roadmaps et 443 pratiques. Les figures 2.6, 2.7, 2.8, 2.9 présentent l'ensemble des roadmaps ainsi que leurs liens avec les domaines de processus.

Process Domain	Key Process Area	Roadmaps
Process Management	Maintenance Process Focus	Responsability and communications
		Information gathering
		Findings
		Action plan
	Maintenance Process/ Service Definition	Documentation and standardisation of processes/services
		Process/service adaptation
		Communication process/services
		Repository of process/services
	Maintenance Training	Requirements, plans and resources
		Personal training
		Training of new personnel
		Training for project in predelivery and transition
		User training
	Maintenance Process Performance	Definition of maintenance measures
		Identification of baselines
		Quantitative management
		Prediction models
	Maintenance Innovation and Deployment	Research of innovations/ improvements
		Analysis of innovations/ improvement proposals
		Piloting selected innovations/ improvement proposals
		Deployment of innovations/ improvements
		Measurement of innovations/ improvement benefits

FIG. 2.6 – Partie 1 des *Roadmaps* du modèle $S3^m$, adapté de [3]

Process Domain	Key Process Area	Roadmaps
Event/Request Management	Event/Request Management	Communications and contact structure
		Management of events and service requests
	Maintenance Planning	Maintenance Planning (1 to 3 yrs)
		Planning project pre-delivery and transition
		Planning disaster recovery testing
		Capacity planning
		Versions and upgrade planning
		Impact analysis (PRs and MRs plans)
	Requests/software Monitoring and Control	Follow-up on planned approved activities
		Review and analysis of progress
		Urgent changes and corrective measures
	SLAs and Supplier Agreements	Customer account management
		Establishment of SLA and contracts
		Execution of services as per SLA and contracts
		Reporting, explanation and billing of services

FIG. 2.7 – Partie 2 des *Roadmaps* du modèle $S3^m$, adapté de [3]

Process Domain	Key Process Area	Roadmaps
Evolution Engineering	Pre-delivery and Transition Services	Involvement and communications with the developer, customer and purchasing
		Management and control of the pre-delivery and transition process
		Control of training and knowledge transfer
		Final transition preparation (products, environment and problem log)
		Participation in system and acceptance tests
	Operational Support Services	Production software monitoring
		After-hours support
		Business rules and functional support
		Ad hoc requests/reports/services
	Software Evolution and Correction Services	Detailed design
		Evolution/Correction (programming)
		Testing (unit, integration, regression)
		Documentation
	Verification and Validation	Reviews
		Acceptance tests
		Installation (move to production of a change or a version)

FIG. 2.8 – Partie 3 des *Roadmaps* du modèle $S3^m$, adapté de [3]

Process Domain	Key Process Area	Roadmaps
Support to Evolution Engineering	Configuration and Version Management	Change Management
		Baseline configuration
		Reservation, follow-up and control of products
	Process, Service and Software Quality Assurance	Objective evaluation
		Identification and documentation of non-conformances
		Communication of non-conformances
		Follow-up on corrections/ adjustements
	Maintenance Measurement and Analysis	Definition of measurement program
		Collection and analysis of measurement data
		Repository of maintenance measures
		Communication of measurement analysis
	Causal Analysis and Problem Resolution	Investigation of defects and failures
		Identification and analysis of causes
		Proposition of solutions
	Software Rejuvenation, Migration and Retirement	Re-documentation of product
		Restructuring of product
		Reverse engineering of product
		Re-engineering of product
		Product migration
		Product retirement

FIG. 2.9 – Partie 4 des *Roadmaps* du modèle $S3^m$, adapté de [3]

2.2.5 Les niveaux du modèle de maturité

Le modèle de maturité $S3^m$ étale ses pratiques sur 6 niveaux de capacité. Comme dans tout modèle de maturité, une entreprise débute par un niveau 0 et tente d'accéder aux niveaux supérieurs en suivant les principes associés à ceux-ci. On notera que plus

un processus de l'entreprise se situe dans un niveau élevé, plus les risques associés à ce processus sont réduits. La figure 2.10 présente les différents niveaux du modèle et les risques associés :

Level	Level name	Risk
0	Unstructured	Higher
1	Executed	
2	Managed	
3	Established	
4	Predictable	
5	Dynamic change	Lower

FIG. 2.10 – Niveaux de maturité de $S3^m$, adapté de [3]

Niveau 0 - Unstructured

Le niveau 0 décrit la situation où une entreprise n'est pas au courant des pratiques exemplaires suggérées par le modèle et de leurs intérêts.

Niveau 1 - Executed

Le niveau 1 décrit la situation où une entreprise reconnaît les pratiques exemplaires suggérées par le modèle mais ne les exécute qu'informellement. Les activités de l'entreprise dépendent principalement de connaissances individuelles non documentées de même que les processus ne sont pas réalisés d'une manière systématique et répétable.

Niveau 2 - Managed

Le niveau 2 décrit la situation où une entreprise met en place localement une pratique exemplaire suggérée par le modèle. La particularité de ce niveau est que l'implémentation de la pratique est localisée et non institutionnalisée, ce qui ne permet pas d'uniformiser les pratiques dans toute l'entreprise.

Niveau 3 - Established

Le niveau 3 décrit la situation où une entreprise respecte les caractéristiques du niveau précédent de même que les critères présentés ci-dessous :

- La pratique exemplaire suggérée par le modèle est exécutée ;
- Les procédures utilisées sont les mêmes dans toutes les unités de l'entreprise ayant une même fonction ;
- Un ensemble limité de mesures est développé, collecté, validé et utilisé ;
- Les employés savent comment exécuter un processus ;
- Le temps et les ressources nécessaires sont alloués afin d'atteindre les buts identifiés ;
- Des collections de techniques, de modèles et d'informations sont mises en place ;
- Le processus est constamment utilisé par le personnel ;
- Les caractéristiques du processus et les activités clés sont mesurées.

Niveau 4 - Predictable

Ce niveau est plus difficile à atteindre que le niveau précédent. Il met en avant des pratiques exemplaires exécutées formellement et gérées quantitativement en accord avec un but défini et dans des limites spécifiées.

Niveau 5 - Dynamic change

Le niveau 5 est l'étape la plus difficile à atteindre. Les pratiques exemplaires sont sous contrôle statistique et en accord avec un but émis dans des limites établies. Le principal objectif de ce niveau est l'aspect de continuité dans les améliorations des processus par des améliorations cumulatives des processus et technologies.

2.3 La gestion du processus de la maintenance du logiciel

Le premier domaine du modèle de maturité de $S3^m$ est consacré à la gestion des processus de la maintenance du logiciel. Ce domaine met en avant l'importance des ressources humaines dans l'activité de maintenance, en égard aux contacts journaliers avec les clients et l'exécution des processus découlant. Les mainteneurs utilisent les processus, les techniques et les outils mis à leur disposition pour satisfaire leurs clients et usagers. La figure 2.11 présente le contexte du domaine :

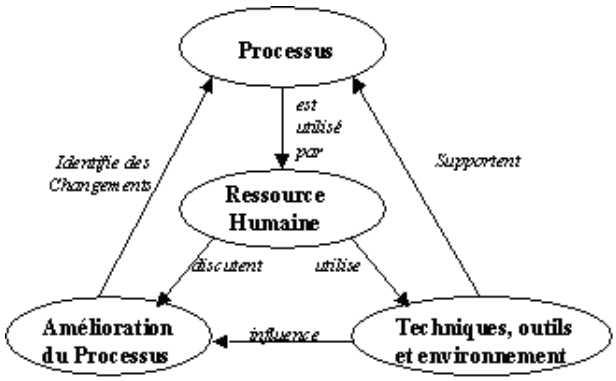


FIG. 2.11 – Contexte de la gestion des processus [3]

Ce domaine de processus couvre les 5 KPAs suivant :

- Focalisation sur les processus de la maintenance du logiciel ;
- Définition des processus/services de la maintenance du logiciel ;
- Formation organisationnelle de la maintenance du logiciel ;
- Performance des processus de la maintenance du logiciel ;
- Innovation et déploiement d'initiatives d'amélioration de la maintenance du logiciel.

La figure 2.12 présente les interactions des différents KPAs du domaine de la gestion des processus de la maintenance entre eux et avec les autres domaines de processus du modèle.

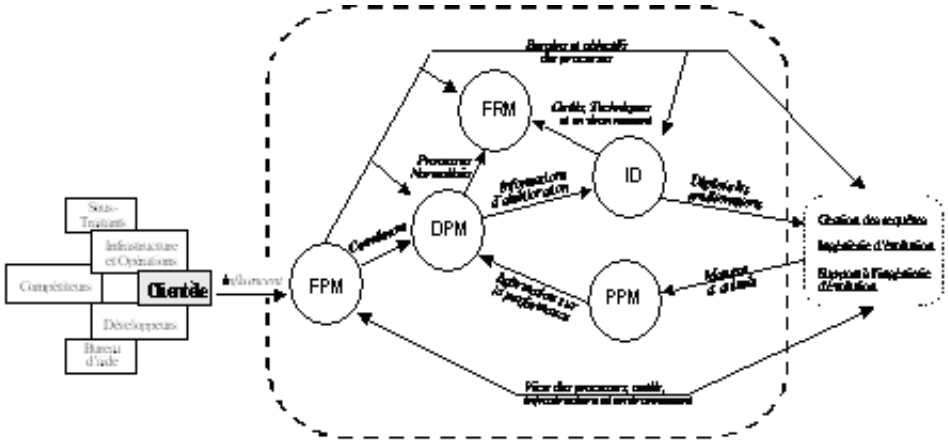


FIG. 2.12 – Interactions des KPAs de la gestion des processus [3]

Le KPA *Focalisation sur les processus de la maintenance* (FPM) aide l'organisation à planifier l'amélioration continue en recevant de l'information :

- de sa clientèle ;
- de l'expérience du vécu des ressources ;
- de leurs besoins de connaissances et compétences ;
- des processus actuels ;
- des techniques, technologies et outils actuels ;
- de l'environnement de travail ;
- des interfaces avec d'autres unités en Technologie de l'information.

Ces activités permettent de coordonner le KPA de *Définition des processus de la maintenance* (DPM) qui fait évoluer les processus génériques concernant les activités, techniques et outils qui seraient plus efficaces pour les employés dans leurs tâches quotidiennes de maintenance et d'évolution des logiciels [3].

Le KPA de *Formation organisationnelle de la maintenance* (FRM) identifie les besoins stratégiques d'éducation et de formation en se concentrant sur les processus de même que les aspects techniques, qui sont communs à plusieurs unités organisationnelles des technologies de l'information. En particulier, la formation est développée ou obtenue afin d'améliorer les compétences et connaissances requises pour exécuter les processus normalisés. Dans ce modèle, on réfère aux pratiques générales de formation des autres modèles mais plus utiles encore sont les concepts développés par Kajko-Mattsson qui offre un modèle de la maturité d'éducation et de formation pour la maintenance corrective du logiciel [3].

À un niveau de maturité plus avancé, le KPA *Innovation et déploiement* (ID) choisit et déploie des projets d'amélioration et des innovations qui améliorent l'habileté de l'organisation à rencontrer ses objectifs de qualité et de performance de l'exécution de

ses processus et des innovations technologiques. L'introduction de nouvelles techniques, outils et procédures devrait être considérée comme l'introduction de changements technologiques. On décidera donc de ces changements technologiques sur une base des faits en se servant de données et d'études de coûts/bénéfices et en effectuant des expériences et des déploiements contrôlés [3].

À un niveau plus avancé, le KPA de la *Performance des processus de la maintenance* (PPM) établit des objectifs quantitatifs :

- de la qualité et de la performance de l'exécution des processus normalisés ;
- des produits ;
- produits intermédiaires ;
- des logiciels applicatifs.

On mesure progressivement l'atteinte des objectifs d'affaires de l'organisation. L'organisation s'assure de coordonner, avec les unités organisationnelles de maintenance du logiciel, la mise en oeuvre :

- des définitions des mesures ;
- des objectifs de ces mesures ;
- des points de références de ces mesures ;
- du référentiel/dépôt des mesures ;
- des modèles de la prévision de la performance des processus normalisés.

Les unités organisationnelles de la maintenance du logiciel analysent les données de la performance de l'exécution des processus normalisés afin de développer une connaissance quantitative de la qualité de :

- ses livrables ;
- ses services ;
- de la performance de l'exécution de ses processus ;
- des technologies qu'elles utilisent [3].

2.4 La gestion des requêtes de la maintenance du logiciel

Le deuxième domaine du modèle de maturité de *S3^m* est consacré à la gestion des requêtes effectuées auprès des mainteneurs ainsi que les services qui y sont associés. Ce domaine de processus couvre les 4 KPAs suivant :

- Gestion des requêtes de services et des événements ;
- Planification de la maintenance du logiciel ;
- Suivi et supervision des requêtes de la maintenance du logiciel ;
- Gestion de l'entente de services et de la sous-traitance ;

La figure 2.13 présente les interactions des différents KPAs du domaine de la gestion des requêtes entre eux avec les autres domaines de processus du modèle.

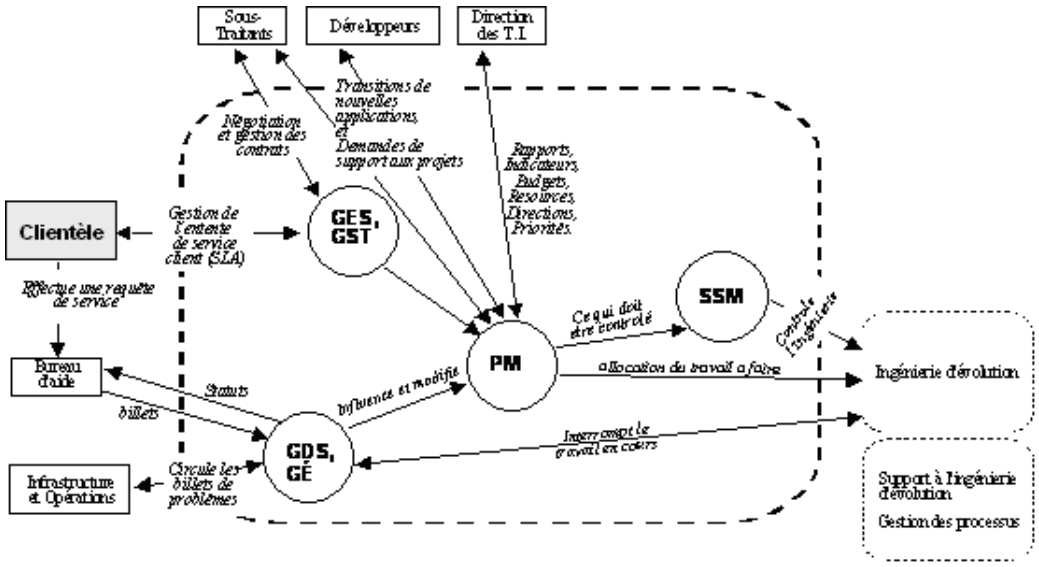


FIG. 2.13 – Interactions des KPAs de la gestion des processus [3]

Le KPA de *Gestion des requêtes de services et des événements* (GDS, GÉ) est le point d'entrée des communications quotidiennes avec la clientèle. On y reçoit les requêtes de support opérationnel, les requêtes de modifications (RM) et les requêtes de problèmes opérationnels (RP). Cet itinéraire doit identifier rapidement la priorité de la requête de la clientèle (avec le billet) qui peut circuler parmi tous les intervenants du processus de résolution de problèmes en technologie de l'information. C'est à cette interface que toutes les communications opérationnelles externes sont coordonnées. L'objectif de ce KPA est de s'assurer que l'unité organisationnelle de la maintenance traite les requêtes pour rencontrer les niveaux de services convenus avec le client [3].

Le KPA de la *Planification de la maintenance* (PM) aide l'organisation et chaque unité organisationnelle de la maintenance du logiciel à planifier l'allocation des ressources aux requêtes et ce dans un contexte où les priorités peuvent changer rapidement. C'est ici que la vue d'ensemble de toutes les requêtes se construit en faisant le tri des requêtes en fonction des priorités et de la disponibilité des ressources, et ce pour répondre aux requêtes :

- de la clientèle ;
- de l'équipe de développement ;
- des sous-traitants ;
- du groupe d'infrastructure et des opérations.

Par la suite, le travail est alloué directement aux ressources de la maintenance du logiciel [3].

Le KPA de *Gestion de l'entente de services et de la sous-traitance* (GES, GST) identifie les activités particulières - plus contractuelles - de la négociation et gestion de la relation

avec sa clientèle et ses partenaires. Ici, on doit bien définir et expliquer nos produits et services et s'assurer que la clientèle est satisfaite de l'entente conjointe. On s'assure également que nos sous-traitants font un partenariat à valeur ajoutée dans l'atteinte des objectifs communs [3].

Le KPA du *Suivi et de supervision de la maintenance* (SSM) contrôle les requêtes ayant été assignées aux ressources et qui sont actuellement en cours de réalisation [3].

2.5 L'ingénierie d'évolution

Le troisième domaine du modèle de maturité de $S3^m$ est consacré aux activités opérationnelles de la maintenance du logiciel. Le domaine décrit des pratiques associées au support opérationnel et au cycle de vie d'une requête de la maintenance . Ce domaine de processus couvre les 4 KPAs suivant :

- Transition du logiciel vers la maintenance ;
- Support opérationnel à la clientèle ;
- Évolution/Correction du logiciel ;
- Vérification et validation ;

La figure 2.14 présente les interactions des différents KPAs du domaine de l'ingénierie d'évolution entre eux avec les autres domaines de processus du modèle.

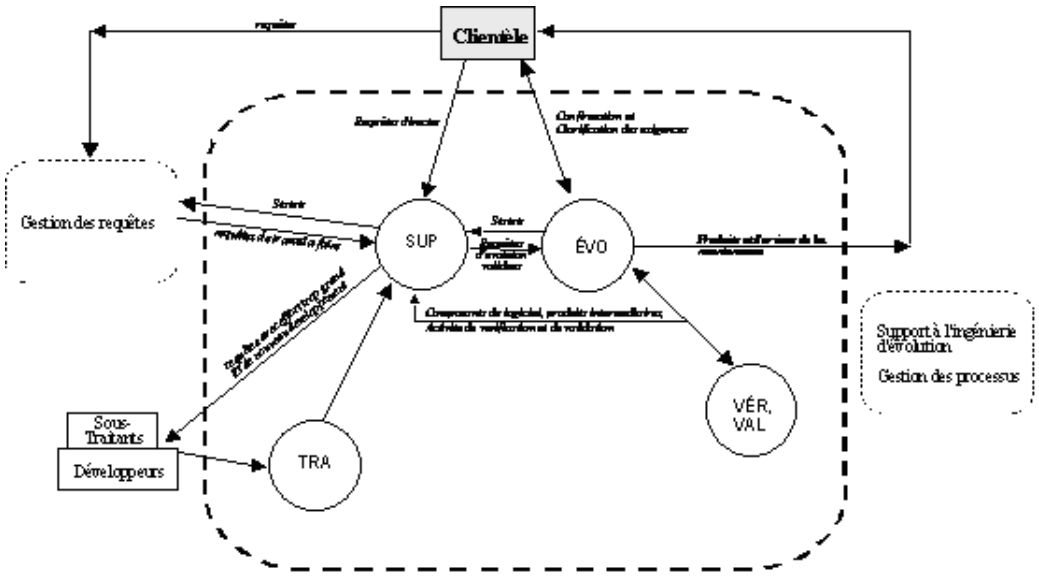


FIG. 2.14 – Interactions des KPAs de l'ingénierie d'évolution[3]

Le KPA de la *Transition du logiciel vers la maintenance* (TRA) est le point d'entrée des logiciels qui feront l'objet de maintenance. L'objectif principal de ce premier KPA

est d'assurer l'implication du mainteneur (peu importe de quelle organisation) pendant le développement du logiciel afin :

- d'identifier, recevoir et préparer de la formation et du transfert de connaissances ;
- d'influencer les caractéristiques de maintenabilité du logiciel pendant sa conception ;
- d'identifier, d'influencer, de compléter et de collecter la documentation utile pour la maintenance ;
- d'établir et publier auprès des intervenants le statut de chaque problème ;
- d'établir les détails de l'entente de services pour ce nouveau logiciel ;
- de préparer l'environnement de support/maintenance du logiciel ;
- d'effectuer les étapes formelles du processus de transition [3].

Le KPA du *Support opérationnel à la clientèle des logiciels* (SUP) traite les requêtes de services qui n'ont pas été résolues par le super-utilisateur ou le bureau d'aide. Les priorités étant déjà identifiées, on traitera dans cet itinéraire les requêtes qui ne nécessitent pas de modification du logiciel. Les requêtes de service nécessitant des modifications au logiciel sont acheminées à l'itinéraire d'évolution du logiciel [3].

Le KPA d' *Evolution du logiciel* (ÉVO) reçoit du KPA de Planification de la Maintenance, les requêtes de modifications autorisées. Le programmeur de la maintenance approfondira les analyses d'impact, effectuera la conception détaillée, les modifications, et les essais. On s'attarde ici à mettre en oeuvre les processus plus spécifiques de la maintenance qui sont similaires aux processus des développeurs [3].

Le KPA de *Vérification et de validation* (VÉR, VAL) contrôle les requêtes qui ont été assignées aux ressources et qui sont actuellement en évolution. À un niveau de maturité plus avancé, le management quantitatif dans ce KPA (VAL) s'assure que la collecte et l'analyse des données seront efficaces au niveau opérationnel. C'est à partir de ces données que les aspects décisionnels du suivi et du contrôle pourront aider dans la prévision de la demande et mieux supporter la planification [3].

2.6 Le support à l'ingénierie d'évolution

Le quatrième et dernier domaine du modèle de maturité de *S3^m* est consacré aux processus de support à l'ingénierie d'évolution de la maintenance de logiciel. Les processus de support sont disponibles et servent quand ils sont requis par les processus opérationnels. Ce domaine de processus couvre les 5 KPAs suivant :

- Management de la configuration et des environnements ;
- Assurance qualité des processus, services et des logiciels ;
- Mesure et analyse de la maintenance ;
- Analyse causale et résolution de problèmes ;
- Rajeunissement, migration et retraite du logiciel.

La figure 2.15 présente les interactions des différents KPAs du domaine du support à l'ingénierie d'évolution entre eux avec les autres domaines de processus du modèle.

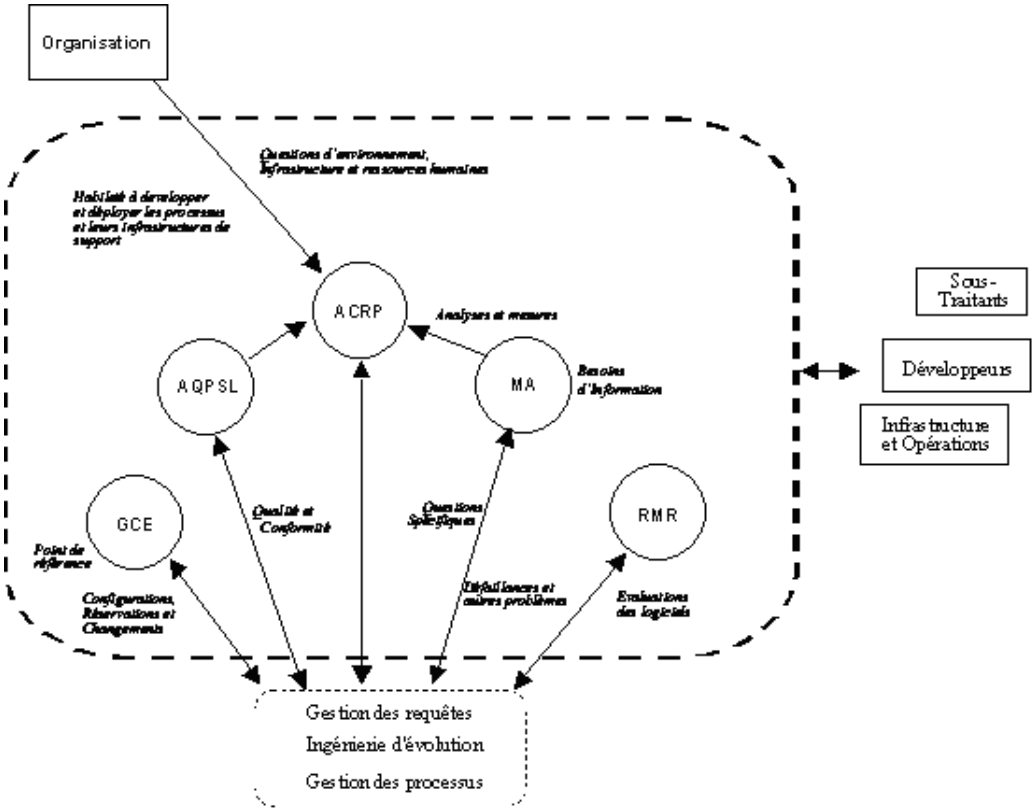


FIG. 2.15 – Interactions des KPAs du support à l'ingénierie d'évolution[3]

Le KPA du *Management de la configuration et des environnements du logiciel* (GCE) est un processus de support souvent partagé entre le développeur et le mainteneur dans une organisation des technologies de l'information. Un premier objectif du management de la configuration est d'établir le lien entre les différents produits intermédiaires tout au long d'un changement et d'en assurer le contrôle. Un objectif secondaire pour le mainteneur s'avère l'opportunité de s'impliquer (peu importe dans quelle organisation) pendant le développement d'un nouveau logiciel afin :

- d'identifier, de recevoir et de préparer la formation et le transfert de connaissances ;
- d'influencer les caractéristiques de maintenabilité du logiciel pendant sa conception ;
- de s'assurer d'identifier, influencer, compléter et collecter la documentation utile pour la maintenance ;
- d'établir et publier le statut de chaque rapport de problèmes auprès des intervenants ;
- d'établir les détails de l'entente de services pour ce nouveau logiciel ;
- de préparer l'environnement de support/maintenance du logiciel ;
- d'effectuer les étapes formelles du processus de transition [3].

Le KPA de l'*Assurance qualité des processus, services et des logiciels* (AQPSL) traite les requêtes de services qui n'ont pas été résolues par le super-utilisateur ou le bureau d'aide. Les priorités étant déjà identifiées, on traitera ici les requêtes ne nécessitant pas de modification du logiciel. Les requêtes nécessitant des modifications au logiciel sont acheminées au KPA d'*Evolution du logiciel* [3].

Le KPA d'*Analyse causale et résolution de problèmes* (ACRP) reçoit des informations de différents processus. Ces informations sont nécessaires pour l'étude de questions/situations générales devant être prises en compte pour régler des problèmes d'ensemble et guider les améliorations majeures et innovations. On cherche ici à étudier des moyens d'améliorer la performance d'ensemble de la maintenance du logiciel [3].

Finalement le KPA de *Rajeunissement, migration et retraite du logiciel* (RMR) effectue les analyses d'optimisation du portfolio des logiciels de la clientèle et propose des activités de :

- redocumentation ;
- restructuration ;
- rétroingénierie ;
- migration ;
- retraite des logiciels afin d'optimiser les coûts de la maintenance.

Chaque proposition est documentée et soumise à la clientèle et aux intervenants afin qu'ils en tiennent compte dans leurs planifications annuelles [3].

2.7 Conclusion

En conclusion, nous avons décrit les bases du modèle *S3^m*. Nous sommes partis du constat que l'activité de maintenance est délaissée dans les modèles de maturité traditionnels pour en aboutir à la description du modèle structuré *S3^m*. Nous avons décrit l'architecture du modèle de même que les niveaux de maturité qu'il met en place. Nous avons ensuite examiné les différents KPA proposés afin de cerner les interactions existantes au sein d'une entreprise de maintenance suivant le modèle. La prochaine partie du mémoire va décrire un outil d'aide à la décision proposant une série de recommandations tirées du modèle présenté en fonction de cas problématiques pour les équipes de maintenance.

Deuxième partie

Un système d'aide à la décision : $S3^m$DSS

1 Introduction

Cette deuxième grande partie du mémoire consiste en la présentation du système $S3^m$DSS. Ce projet est le fruit du travail réalisé lors du stage effectué à l'Ecole de Technologie Supérieure de Montréal. Il consiste en la mise en place d'un système à base de connaissance spécifique au monde de la maintenance. Comme tout système de ce type, deux volets sont à distinguer. Une partie technique comportant les différents mécanismes mis en place pour le bon déroulement de l'application et une partie orientée contenu, délivrant un certain nombre d'indications aux utilisateurs du système. Cette deuxième partie de mémoire présente avant tout la partie technique et orientée mécanismes de l'application.

Nous décrirons le système dans son ensemble tout en couvrant les différentes étapes de son cycle de vie. Une description détaillée des origines, objectifs et fonctionnalités de l'application est présentée dans le *chapitre 2*. Une analyse des besoins initiaux effectuée auprès du commanditaire du projet et présentée dans le *chapitre 3*. Celle-ci sera articulée autour des différentes catégories d'utilisateurs impliqués dans le projet, des cas d'utilisations prévu pour satisfaire à leurs exigences et des exigences non fonctionnelles qui ont pu être décelées. Une description conceptuelle du système sera ensuite entreprise au moyen de différents diagrammes, tentant d'aborder toutes les facettes du système en *chapitre 4*. Pour ce faire, nous analyserons la modélisation structurelle de $S3^m$DSS au moyen de diagrammes de robustesse et d'un diagramme de classes. La modélisation comportementale sera présentée sous forme de diagrammes de séquence afin de bien saisir la dynamique de l'application tandis que la modélisation de la persistance sera décrite au moyen de diagrammes conceptuels et ERA. Une analyse de l'architecture employée, tout en rappelant les principes théoriques sous-jacents est présentée en *chapitre 5*. Les différentes technologies employées dans la confection du système seront décrites et justifiées dans le *chapitre 6*. Nous aborderons ensuite dans le *chapitre 7* les techniques de tests employées pour contrôler le bon fonctionnement du prototype. Une description des interfaces du système sera finalement présentée dans le *chapitre 8* tandis que le *chapitre 9* conclura cette présentation du système $S3^m$DSS en rappelant les grands axes du projets.

2 Présentation générale

Présenter un système dans ses moindres facettes n'est jamais une tâche aisée. Ce chapitre tentera d'aborder le système $S3^m$DSS d'un point de vue assez générale permettant ainsi de se constituer une première opinion afin d'aborder les chapitres suivants dans les meilleurs termes. Le point 1 exposera les origines multiples du projet ayant amenés le système $S3^m$DSS à voir le jour. Le point 2 traitera des grands objectifs que véhicule le système tandis que le point 3 présentera le *modus operandi* général mis en place afin de satisfaire les besoins des utilisateurs du système. Finalement, le point 4 conclura ce chapitre tout en invitant le lecteur à compléter sa connaissance du système dans la suite de cet ouvrage.

2.1 Origines

Le système $S3^m$DSS trouve sa première origine dans le logiciel COSMICXpert. Ce système a été chapeauté par Jean-Marc Desharnais et implémenté graduellement par Tim Küessing, Julien Vilz et François Gruselin [8]. Son objectif principal est d'offrir un apprentissage de la méthode de mesure fonctionnelle COSMIC-FFP qui bien souvent est mal interprétée par ses praticiens.

La méthode de mesure fonctionnelle permet de mesurer la taille d'un logiciel en terme de ses fonctionnalités, en s'appuyant sur la méthode des points de fonctions. Elle est assez importante comme le décrit [1] :

« Des mesures sont nécessaires pour analyser tant la qualité que la productivité du développement ou de la maintenance des logiciels. D'une part, des mesures techniques sont nécessaires pour quantifier la performance technique des produits ou des services du point de vue des développeurs. Des mesures techniques peuvent être utilisées pour des analyses d'efficacité ; pour l'amélioration de la performance du design entre autres. D'autre part, les mesures fonctionnelles sont nécessaires pour quantifier la performance des produits ou des services du point de vue de l'utilisateur ; pour l'analyse de la productivité entre autres. Les mesures fonctionnelles doivent être indépendantes des techniques de développement et des décisions d'implantation. Elles peuvent donc être utilisées pour comparer la productivité des différentes techniques et technologies. »

COSMICXpert s'appuie sur une base de connaissance et sur un moteur d'inférence. La

base de connaissance est composée d'un ensemble de mots-clés, concepts topologiques, cas problèmes, thèmes, faits et recommandations. La figure 2.1 présente le schéma conceptuel de COSMICXpert montrant comment ces différents concepts sont agencés entre eux.

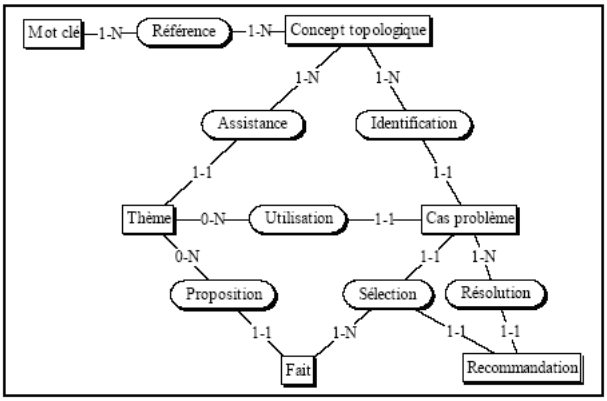


FIG. 2.1 – Schéma conceptuel de COSMICXpert

A partir d'un mot-clé, l'utilisateur est invité à répondre à une série de questions relatives à un problème qu'il désire résoudre (ex : identifier un *entry point*) et spécifique à la mesure fonctionnelle COSMIC-FFP. Le moteur d'inférence du système tentera de dériver une recommandation à l'utilisateur en fonction de ses propres réponses. Notons finalement que le lecteur intéressé pour consulter ce système à l'adresse suivante : http://www.gelog.etsmtl.ca/cosmicxpert/login.jsp.

A la suite de cette expérience réussie, l'idée a été d'appliquer le même raisonnement au monde de la maintenance et tout particulièrement au modèle de maturité $S3^m$ [3]. Ceci constitue le système SMXpert, aboutissement d'un travail de restructuration de COSMICXpert réalisé par Stéphane Sandron et Benoît Vanderose [22]. SMXpert s'appuie également sur une base de connaissance comportant différents éléments tels que des index, mots-clés, concepts de maintenance, cas problèmes, questions, faits et recommandations. La figure 2.2 présente le schéma conceptuel de SMXpert montrant comment ces différents concepts sont agencés entre eux.

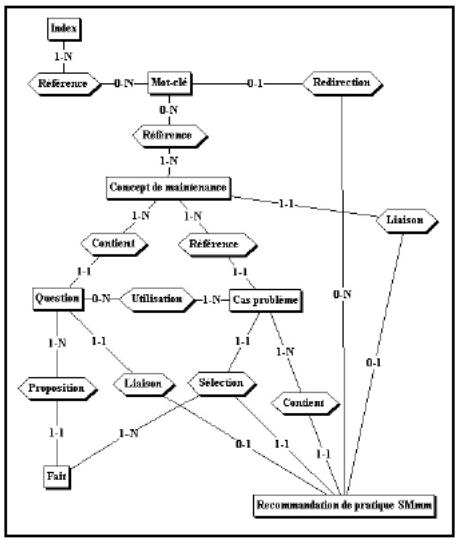


FIG. 2.2 – Schéma conceptuel de SMXpert

La partie moteur d'inférence n'a quant à elle pas pu être adaptée au domaine de la maintenance compte tenu de la nature différente des problèmes. Ce problème a subsisté jusqu'à ce jour et n'est pas toujours bien compris par les différents acteurs du projet. Néanmoins, après réflexion, une explication de ce échec peut être avancée. Le moteur d'inférence souhaité devrait travailler sur des questions auxquelles sont liés des pourcentages de pondération. Ces questions amènent, selon les réponses, à des recommandations. Dans la pratique, l'usage fait des pondérations attribuées à chaque question et recommandation a été complètement détourné. En réalité, peu importe leurs valeurs car elles sont réadaptées pour correspondre au modèle.

De plus, notons que la théorie sous-jacente à ce moteur d'inférence s'énonce ainsi : "la théorie de la certitude semble appropriée pour l'expert de la mesure dont la problématique de résolution de problèmes est de nature binaire." Or dans le cas du système qui nous occupe, nous ne traitons pas de problématique binaire mais bien des problématiques ternaires, quaternaires ou plus encore. Il ne s'agit plus ici d'identifier un objet ou de décrire sa véracité ou sa fausseté. Notons finalement que la théorie de la certitude, tel que supposée implémentée repose sur une incertitude des faits et une incertitude des règles. Or cette dernière n'a pas trouvé d'implémentation dans la pratique. On comprend, dès lors, pourquoi cette approche système expert est à mettre en doute. Gageons toutefois qu'après un certain nombre d'années d'essais et d'erreurs, le système aie trouvé sa direction tout en éclaircissant les points nébuleux qui ont conduits à sa répétition.

Par la suite, une série d'améliorations ont été poursuivies par Raja Dallape dans le cadre de son mémoire [6]. Ces travaux ont eu pour principal but d'améliorer la prise en main du système par une refonte complète de l'interface selon des principes communément reconnus en IHM. Une migration de la base de connaissance conçue initialement en XML

vers une base de donnée de type SQL Server a également été intégrée.

Les résultats proposés par le système SMXpert n'ont au final pas abouti à quelque chose de concluant pour ses commanditaires. On pourra même noter une certaine divergence d'esprit concernant le moteur d'inférence que devrait proposer le système. Mon implication dans le projet débute sur ce point. Ma première tâche fût de reconstruire un système qui avait perdu son caractère initial de système expert. Au fur et à mesure des avancements, nous nous sommes butés sur le problème chronique du moteur d'inférence. Il semble que la théorie sous-jacente au système expert ne soit pas encore bien maîtrisée, de même que le processus de décision de l'expert en maintenance non encore cerné. Nous avons donc décidé de poursuivre l'expérience vers un système à la tournure légèrement différente et proposant un accès aisé aux différentes informations comprises dans le modèle de maturité $S3^m$. Une approche 'système expert' pourra sur base des résultats de cette nouvelle application être enclenchée dans un avenir proche. Il était évident que ce point devait être clarifié afin de ne pas travailler sur un système dont l'apparence reflétait une autre dimension que son contenu. Ceci a conduit au système $S3^m$DSS pour Software Maintenance Maturity Model Decision Support System.

2.2 Objectifs

Au fil des rencontres avec les experts en maintenance, un certain nombre d'objectifs ont été fixés afin de palier à différents problèmes du monde de la maintenance et du modèle de maturité $S3^m$. Le système que nous présentons tente de satisfaire à ces objectifs.

Nous proposons d'établir une correspondance entre des problèmes liés à la maintenance et le modèle $S3^m$. Ces problèmes, présentés en détail dans la partie III, sont basés sur différents sondages réalisés par [7] et qui tentent de hiérarchiser les problèmes les plus courants en maintenance. Face à ces problèmes récurrents, un ensemble de recommandations peut être proposé, celles-ci sont reprises sous forme de bonnes pratiques dans [3]. Il est néanmoins ardu pour l'intéressé de trouver dans le modèle $S3^m$ les bonnes pratiques directement applicables à un cas précis. Ce travail de recherche et d'adaptation, habituellement exercé par un expert en maintenance, est donc celui du système $S3^m$DSS. Notons qu'à l'heure actuelle, le système ne se base que sur les pratiques exemplaires relatives aux niveaux 0, 1 et 2 du modèle. Les niveaux suivants étant encore pour l'instant *non pro bono publico*.

Parallèlement à cet objectif principal de correspondance, on notera que le système propose par voie de conséquence une meilleure visibilité pour le modèle $S3^m$ ainsi qu'une certaine vulgarisation de ses recommandations. Il permet en tout cas de familiariser les entreprises avec les principes de maintenance. Une passerelle commerciale pourra également être sous-entendue si l'on considère que l'utilisateur désire poursuivre son processus d'amélioration en réalisant une évaluation plus poussée et/ou en achetant les bonnes pratiques des niveaux supérieurs. Finalement, on retiendra que le système permet de promotionner le cycle de maintenance souvent délaissé par les entreprises et les auteurs.

Notons également que la direction prise par le système est maintenant assez claire : elle permet de proposer des pratiques adéquates en égard à un problème particulier de la maintenance. Par le passé, une certaine confusion s'était faite ressentir sur les possibilités du système. Tantôt outil d'évaluation du niveau de maturité d'une entreprise selon la méthode $S3^m$, tantôt système expert permettant d'évaluer un problème du mainteneur. Le volet d'évaluation est, pour l'instant, mis de côté mais il pourra ultérieurement être poursuivi dans le cadre d'un autre projet.

Pour terminer, notons qu'une des exigences principales du système est son accessibilité de par l'Internet. Ceci se justifie par la nature expérimentale du projet, nécessitant la validation d'experts éloignés mais aussi par la facilité de distribution du système via la simple communication d'une adresse et d'un compte utilisateur.

2.3 Fonctionnement

A présent que nous connaissons les éléments qui ont fait émergé $S3^m$DSS et les objectifs principaux du système, décrivons brièvement le mode de fonctionnement principal que le système propose à l'utilisateur. Une description plus détaillée des acteurs et des fonctionnalités peut être retrouvée dans le chapitre 3.

Selon [27], la première activité dans la construction d'un système à base de connaissance est la définition d'une analyse de tâches. Cette analyse de tâches débute, à un haut niveau, avec la définition d'un index de termes. Cet index comprend un ensemble de mots communément utilisés en ingénierie logiciel. La figure 2.3 propose une représentation schématique de ces concepts. A partir de l'index, un sous-ensemble plus restrictif de mots est identifié. Ce sous-ensemble est une liste de mots-clés reconnus expressément en maintenance logicielle. Chaque mot-clé est ensuite connecté à l'un ou l'autre concept de maintenance.

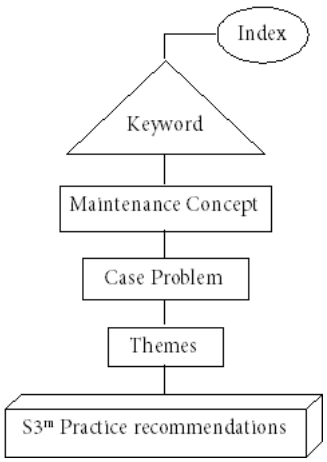


FIG. 2.3 – Vue au niveau du système

Un concept de maintenance est un concept trouvé dans le Software Maintenance Body of Knowledge [2] et dans l'ontologie présentée par Kitchenham [14]. Chaque problème de maintenance identifié par Dekleva a été traduit en un cas problème et connecté à l'ontologie de la maintenance logicielle. Chaque cas problème est lié à un ensemble de thèmes (ou questions) qui aident l'utilisateur du système à naviguer dans les différentes parties du modèle de maturité $S3^m$ et qui propose une série de recommandations sous forme de bonnes pratiques. Le lien entre les concepts de maintenance et le modèle de maturité est réalisé au moyen des thèmes. Ces derniers sont des questions qui ont été développées de telle manière que l'on sautera de noeud en noeud dans l'ontologie. Le lecteur averti découvrira que les thèmes combinent souvent différents concepts de maintenance. Pour chaque bonne pratique existe un thème lié (ou un choix) que l'utilisateur peut sélectionner (aussi appelé faits) et qui au final l'amènera à une série de recommandations. Cette correspondance 1-1 entre les thèmes et les recommandations contribuera à la composition d'un ensemble de recommandations directement adaptées au contexte de l'utilisateur. Une description plus poussée du raisonnement décrit est proposée dans la partie III. La figure 2.4 présente sous forme d'un schéma de déroulement le *modus operandi* du système.

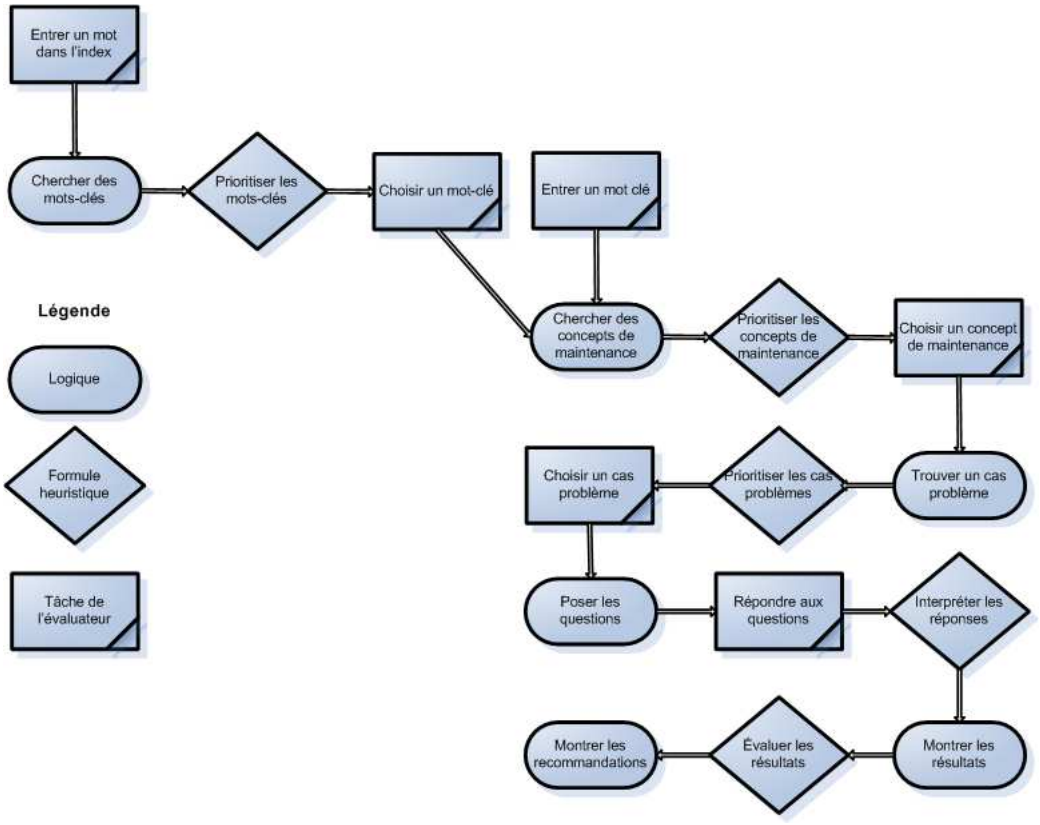


FIG. 2.4 – Enchaînement des tâches dans $S3^m$DSS, adapté de [22]

Derrière tout cela, une distinction entre les ingénieurs internes de la maintenance et les clients de la maintenance a été établie. Nous pensons que les mêmes problèmes sont en

jeu pour ces deux types d'utilisateur mais qu'ils doivent être adaptés légèrement selon les acteurs. Dans ce cas, quand un client de la maintenance utilisera le système, les questions seront adaptées à sa compréhension, de même que les recommandations proposées.

2.4 Classification

Il peut être intérressant pour le lecteur de classifier le type de système que forme $S3^m$DSS. La question posée de par l'historique du système est de savoir si nous avons affaire à un système expert ou un système d'aide à la décision. Cette partie présente clairement ce que l'on entend par système expert et système informatisé d'aide à la décision. Une justification sera ensuite proposée afin de trancher la question selon les différentes discussions entreprises autour du sujet.

2.4.1 Système expert

Desharnais [8] définit un système expert de la façon suivante :

> *An expert system is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problem or giving advice*

De cette définition, il en déduit qu'un système expert possède un certain nombre de connaissances, que ce soit dans des algorithmes, des listes d'exemples, des questions/réponses, etc. On remarque aussi qu'une bonne partie des connaissances d'un système expert appartient à un domaine spécifique. Une collection de connaissances provenant de différents domaines sont rarement l'objet d'une expertise. Finalement, la connaissance doit permettre de résoudre des problèmes autres que des problèmes d'accès à une documentation en ligne (hyperliens) et peut aider à résoudre des problèmes ou à donner des avis.

Un système expert est généralement composé d'une base de connaissances (et de faits), d'un moteur d'inférence et de différentes interfaces utilisateurs. La base de connaissances comprend des faits et des règles selon un mode de représentation tel que les réseaux sémantiques, les objects structurés, les réseaux neuronaux ou les règles de production. Le moteur d'inférence permet d'expliciter la connaissance selon un mode de représentation spécifique. Desharnais [8] ajoute qu'un moteur d'inférence est caractérisé par un cycle de base, une stratégie de recherche et une méthode de chaînage. Finalement, pour intéragir avec le système, une interface conviviale doit être ajoutée.

Notons que les systèmes experts présente également un module d'acquisition des connaissances, leur permettant d'être aussi évolutifs que possible. Ce module permet l'ajout de nouvelles connaissances mais aussi la mise en relation de nouvelles connaissances entre elles et avec les anciennes connaissances. Cette démarche permet d'éviter d'écrire de nouveaux programmes. Une base de connaissance permet de mémoriser tout cela.

Fichefet [10] présente schématiquement les systèmes experts de la façon suivante :

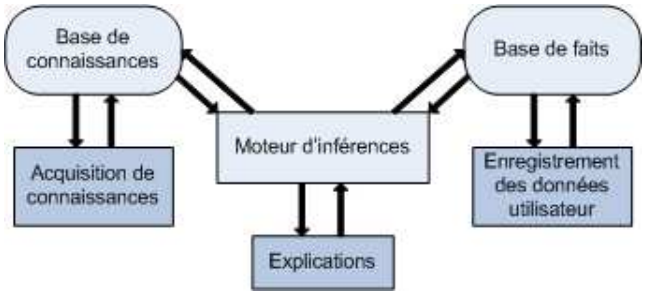


FIG. 2.5 – Représentation d'un système expert, adapté de [10]

2.4.2 Système informatisé d'aide à la décision

Selon [21] : "L'aide à la décision est l'activité de celui qui, par des voies dites scientifiques, aide à obtenir des éléments de réponse à des questions que se posent des acteurs impliqués dans un processus de décision, éléments concourant à éclairer la décision en vue de favoriser un comportement des acteurs qui soit de nature à accroître la cohérence entre l'évolution du processus d'une part, les objectifs et ou les systèmes de valeurs au service desquels ces acteurs se trouvent placés d'autre part."

Un Système Interactifs d'Aide à la Décision (SIAD) ou Decision Support System (DSS) est, quant à lui, un systèmes d'information interactifs construits à l'aide de technologies d'information et destinés à supporter les activités des gestionnaires. On notera que les systèmes experts peuvent être considérés commes des SIAD intelligents. Dallape [6] ajoute que : "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".

2.4.3 Justification

Une description détaillée du raisonnement instauré par le système $S3^m$DSS est proposée en partie III. Néanmoins afin de fournir une justification appropriée, rappelons que ce raisonnement se décompose en deux phases : *la sélection d'un cas problème* et *la déduction de recommandations*. De plus, un arbre de décision peut être construit de telle sorte que ses feuilles sont les recommandations finales proposées à l'utilisateur. A la question de savoir si le présent système peut être considéré comme un système d'aide à la décision, la réponse est positive. Nous avons bien là affaire à une application permettant d'éclairer la décision de l'utilisateur sur une question précise. Ajoutons également que le système est intéractif et permet donc le dialogue avec l'utilisateur par un langage proche du langage naturel.

Nous ne pouvons néanmoins pas considérer le système comme un système expert à part entière. Ci-haut nous avons décrit les différents composants d'un tel système, à savoir une base de connaissances (et de faits), un moteur d'inférence et une interface conviviale. Le système $S3^m$DSS possède bien une base de connaissances composées de mots-clés,

concepts de maintenance ou encore de recommandations. Il possède bien un ensemble de faits posés par l'utilisateur, de même qu'une interface d'intéraction avec l'utilisateur via un navigateur web. Néanmoins le moteur d'inférence n'est pas présent dans le système en tant que tel. Nous avons bien un mécanisme de dérivation de recommandations mais il est manifestement statique et matérialisé sous forme d'un arbre de décision. L'approche "rules based reasoning" est donc à exclure. Aucun mécanisme d'acquisition autonome de connaissances n'est proposé. L'approche "case based reasoning" pourrait donc convenir dans la limite où l'utilisateur serait le seul validateur de ses choix et non le système mais cette approche se heurte à l'absence d'un module d'acquisition de connaissances autonome. Le système comporte également un volet d'interactivité qui peut entrer en opposition avec la classe des systèmes experts. Il semble donc que qualifier $S3^m$DSS comme système expert soit un terme trop fort bien que certains points le font paraitre comme tel. N'ayant pas pu satisfaire à l'ensemble des propriétés d'un système expert, nous nous limiterons donc à son appelation de système d'aide à la décision tout en gardant à l'esprit que plus qu'une querelle d'intellectuels, le système ne cherche pas à rentrer dans l'une ou l'autre catégorie par vantardise mais plutôt à suivre le raisonnement de l'expert et par là à satisfaire l'utilisateur final.

2.5 Conclusion

En conclusion, $S3^m$DSS est un système d'aide à la décision permettant de proposer à l'utilisateur un certain nombre de bonnes pratiques en matière de maintenance de logiciel. Il est le résultat d'une tentative d'adaptation du système COSMICXpert au domaine de la maintenance tout en ayant précisé son domaine d'application après un certain nombre d'essais/erreurs révélés par SMXpert. Une description du mode de fonctionnement du système a également été présentée, celle-ci sera complétée dans la partie III.

3 Analyse des besoins

L'analyse des besoins est la première phase de réalisation d'un projet. C'est celle qui conditionne sa réussite dans la mesure où elle définit les besoins réels de ceux qui vont utiliser le résultat final. Phase de communication et d'échange, elle est souvent le reflet du résultat final. Nécessitant rigueur et méthode, c'est une des phases les plus difficiles de la conduite de projet. Dans ce chapitre, nous éliciterons les fonctionnalités du système offertes aux utilisateurs, et ce à partir des différentes informations dont nous avons pris connaissance. Le point 1 identifiera et décrira les différents utilisateurs du système. Le point 2 définira les cas d'utilisation disponibles pour chacune des classes d'utilisateur. Le point 3 exposera les différentes exigences non fonctionnelles du système. Le point 4 conclura ce chapitre.

3.1 Catégories d'utilisateurs

Cette partie a pour but de décrire les différentes catégories d'utilisateurs qui interagissent avec le système. Le système $S3^m$DSS présente trois mode d'interactions pour quatre types d'utilisateurs différents : l'utilisateur de type « évaluateur externe », l'utilisateur de type « évaluateur interne », l'utilisateur de type « expert » et l'administrateur. Une nuance a été introduite entre les utilisateurs de type « évaluateur externe » et « évaluateur interne ». En effet, les problèmes cernés par ces deux métiers sont différents mais néanmoins ils peuvent se résoudre dans la plupart des cas d'une même façon. Il a donc été nécessaire d'adapter l'interaction avec le système en fonction de l'origine de l'utilisateur. Les cas problèmes de même que les questions posées et les recommendations proposées seront adaptés à la situation.

Pour chaque catégorie d'utilisateurs, une description de du rôle et des principales caractéristiques sera précisée. Une définition de l'environnement de travail nécessaire à l'utilisation du système ainsi qu'une liste des tâches pouvant être effectuées est également présentée. Ces tâches seront ultérieurement décrites de manière plus complète. Enfin une analyse de l'expertise des systèmes informatiques et de la motivation à utiliser le système est proposée pour chaque classe d'utilisateur.

3.1.1 Evaluateur externe

Il s'agit de l'évaluateur considéré comme client d'un processus de maintenance. On parlera d'évaluateur "externe" car l'acteur se situe en dehors du service de maintenance, ou d'une manière générale en dehors de l'entreprise effectuant la maintenance d'un logiciel proprement dit. Dans le cadre de l'exploitation d'un de ses logiciels, il effectuera une demande de maintenance d'une plusieurs fonctionnalités. Il utilisera le système $S3^m$DSS afin d'améliorer sa coordination avec le service en charge de la maintenance de ses logiciel en consultant la base de connaissance proposée.

Caractéristiques

- Attributs physiques :
  - Age : sans importance ;
  - Sexe : sans importance.
- Attributs mentaux :
  - Bonnes capacités intellectuelles.
- Qualifications et connaissances :
  - Connaissance de l'anglais ;
  - Connaissances de base en informatique.

Environnement

- Localisation : sans importance du moment qu'un accès à Internet est possible ;
- Matériel : stations disposant d'un clavier, un écran, une souris, Firefox 1.x ou Internet Explorer 6.x et une JRE 1.5.x ou supérieur ;
- OS : famille Windows avec un support de la technologie Java.

Liste de tâches

- Authentification ;
- Désauthentification ;
- Consultation KB ;
- Emission suggestion ;
- Consultation aide générale ;
- Consultation aide locale ;
- Consultation définition.

Expertise des systèmes informatiques

Elle peut être faible mais néanmoins l'évaluateur doit pouvoir utiliser une interface Web.

Motivation à utiliser le système

Elle est élevée car elle permet un gain d'efficacité très important dans la coordination et les résultats proposés par la service de maintenance.

3.1.2 Evaluateur interne

Il s'agit de l'évaluateur considéré comme participant à un processus de maintenance. On parlera d'évaluateur interne car l'acteur se situe directement dans le service de maintenance, ou d'une manière générale au sein de l'entreprise effectuant la maintenance d'un logiciel proprement dit. Dans le cadre de la maintenance d'un de ses logiciels, il utilisera le système $S3^m$DSS afin d'améliorer la coordination avec ses clients et/ou les pratiques en vigueur au sein de son service en consultant la base de connaissance proposée. Cette démarche de correction et d'adaptation pourra apporter une plus-value non négligeable au service et à l'entreprise.

Caractéristiques

- Attributs physiques :
  - Age : sans importance ;
  - Sexe : sans importance.
- Attributs mentaux :
  - Bonnes capacités intellectuelles.
- Qualifications et connaissances :
  - Connaissance de l'anglais ;
  - Connaissances de base en informatique.

Environnement

- Localisation : sans importance du moment qu'un accès à Internet est possible ;
- Matériel : stations disposant d'un clavier, un écran, une souris, Firefox 1.x ou Internet Explorer 6.x et une JRE 1.5.x ou supérieur ;
- OS : famille Windows avec un support de la technologie Java.

Liste de tâches

- Authentification ;
- Désauthentification ;
- Consultation KB ;
- Emission suggestion ;
- Consultation aide générale ;
- Consultation aide locale ;
- Consultation définition.

Expertise des systèmes informatiques

Elle peut être faible mais néanmoins l'évaluateur doit pouvoir utiliser une interface Web.

Motivation à utiliser le système

Elle est élevée car elle permet un gain d'efficacité très important dans la compréhension du modèle de maintenance $S3^m$.

3.1.3 Administrateur

Il s'agit de l'utilisateur qui va administrer le système et gérer les profils et demandes des différents utilisateurs du système. Son rôle est de nature strictement technique, il n'interviendra pas dans les modifications de contenu du système.

Caractéristiques

- Attributs physiques :
  - Age : sans importance ;
  - Sexe : sans importance.
- Attributs mentaux :
  - Bonnes capacités intellectuelles.
- Qualifications et connaissances :
  - Connaissance de l'anglais ;
  - Connaissances avancées en informatique.

Environnement

- Localisation : sans importance du moment qu'un accès à Internet est possible ;
- Matériel : stations disposant d'un clavier, un écran, une souris, Firefox 1.x ou Internet Explorer 6.x et une JRE 1.5.x ou supérieur ;
- OS : famille Windows avec un support de la technologie Java.

Liste de tâches

- Actions des utilisateurs ;
- Actions des experts ;
- Ajout compte ;
- Retrait compte ;
- Modification statut ;
- Modification mot de passe ;
- Modification date d'expiration.

Expertise des systèmes informatiques

Elle doit être élevée car le bon fonctionnement de l'ensemble du système dépend principalement des décisions prises par l'administrateur mais elle peut être diminuée par la technologie utilisée se présentant sous forme d'interfaces Web.

Motivation à utiliser le système

Elle est moyennement élevée car elle est assez répétitive.

3.1.4 Expert

Il s'agit de l'utilisateur qui va opérer différents changements dans la base de connaissance en ajoutant différents scénarii afin de rendre la base de connaissance beaucoup plus complète et précise. Il est considéré comme un expert du domaine de la maintenance car le contenu qu'il propose d'insérer relève exclusivement de la maintenance logicielle et sera directement accessible par les autres catégories d'utilisateur. Une attention toute particulière sera apportée au contenu qu'il insère dans le système car il n'est sujet qu'à sa propre validation.

Caractéristiques

- Attributs physiques :
  - Age : sans importance ;
  - Sexe : sans importance.
- Attributs mentaux :
  - Bonnes capacités intellectuelles.
- Qualifications et connaissances :
  - Connaissance de l'anglais ;
  - Connaissances de base en informatique ;
  - Connaissances avancées en $S3^m$ ou plus généralement en maintenance du logiciel.

Environnement

- Localisation : sans importance du moment qu'un accès à Internet est possible ;
- Matériel : stations disposant d'un clavier, un écran, une souris, Firefox 1.x ou Internet Explorer 6.x et une JRE 1.5.x ou supérieur ;
- OS : famille Windows avec un support de la technologie Java.

Liste de tâches

- Actions des utilisateurs ;
- Ajout de connaissances dans la KB ;
- Modification de connaissances dans la KB ;
- Retrait de connaissances dans la KB.

Expertise des systèmes informatiques

Elle doit être élevée car les informations de l'ensemble du système dépendent principalement de l'activité de l'expert mais elle peut être diminuée par la technologie utilisée se présentant sous forme d'interfaces Web.

Motivation à utiliser le système

Elle peut être très variable.

3.2 Cas d'utilisation

Cette partie a pour but de décrire les différentes fonctionnalités attendues par les utilisateurs du système. Des diagrammes des cas d'utilisation sont utilisés pour donner une vision globale du comportement fonctionnel du système. Ces diagrammes des cas d'utilisation sont différenciés pour des raisons de clarté. Une description détaillée des fonctionnalités sous-entendues dans ces diagrammes est ensuite présentée afin de saisir le sens des cas d'utilisation. L'ensemble de ces fonctionnalités est également décrit sous forme de scénarios en annexe.

Remarque : Dans la suite de cette analyse, nous regrouperons les évaluateurs externes et évaluateurs internes en tant qu'utilisateur. Nous ne les différencierons pas car ils ne proposent pas de mécanismes différents si ce n'est une adaptation des connaissances proposées en fonction de l'acteur comme explicité auparavant.

3.2.1 Diagramme des cas d'utilisation - Vue *Utilisateur*

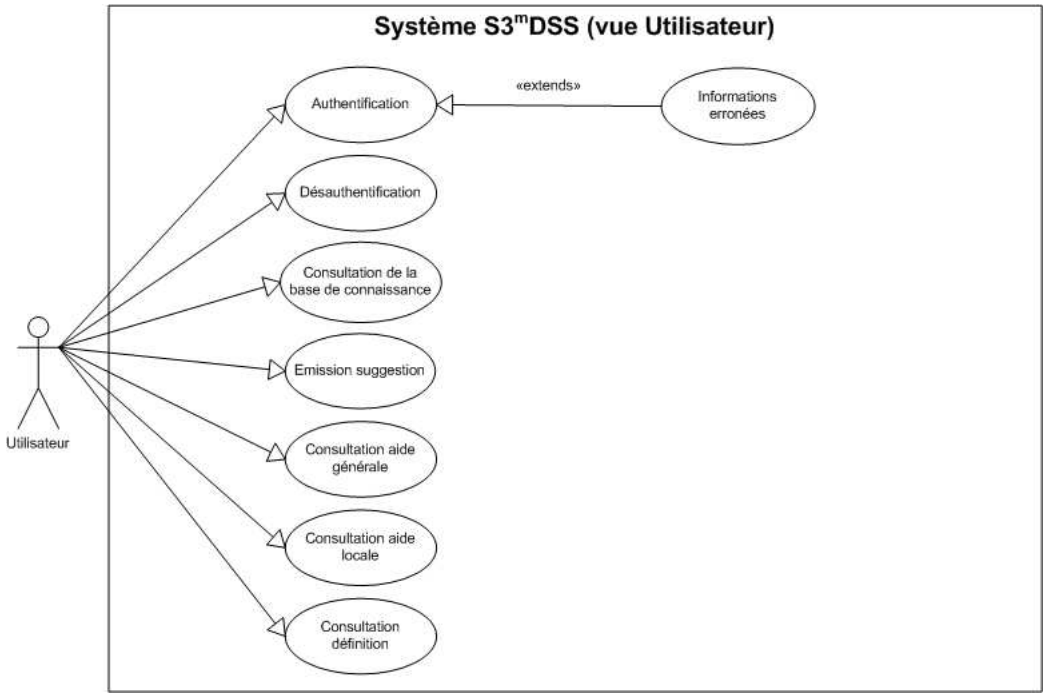


FIG. 3.1 – Diagramme des cas d'utilisation de l'utilisateur

Description : Ce système comprend l'ensemble des actions que les utilisateurs du système peuvent réaliser.

3.2.2 Fonctionnalités - Vue *Utilisateur*

Authentification

Description : Un utilisateur souhaite s'authentifier auprès du système. Pour ce faire, le système lui propose un mécanisme d'accès par la saisie d'un login et d'un mot de passe. Ces informations sont fournies à l'utilisateur par l'intermédiaire de l'administrateur.

Désauthentification

Description : Un utilisateur souhaite se désauthentifier du système. Pour ce faire, le système lui propose un mécanisme de déconnexion par un simple bouton de l'interface.

Consultation de la base de connaissance

Description : Un utilisateur souhaite consulter la base de connaissance du système. Celui-ci propose un mécanisme d'accès pas à pas aux connaissances du système. Le déroulement de l'action passe par un mot-clé, un concept de maintenance, un cas problème et une liste de questions pour aboutir à un ensemble de recommandations accessibles par un simple

lien hypertexte. Notez que la séquence peut débuter par la saisie d'un mot dans l'index référençant un mot-clé.[1]

Emission suggestion

Description : Un utilisateur souhaite émettre une suggestion à l'auteur du système. Pour ce faire, le système lui un mécanisme de lancement automatique de son client mail favori pré rempli de l'adresse du destinataire.

Consultation aide générale

Description : Un utilisateur souhaite obtenir une aide générale quant à l'utilisation du système. Pour ce faire, le système propose un mécanisme d'accès à une aide générale sur le contenu du logiciel.

Consultation aide locale

Description : Un utilisateur souhaite obtenir une aide localisée concernant un concept du système. Pour ce faire, le système propose un mécanisme d'accès à une aide spécialisée sur le concept impliqué.

Consultation définition

Description : Un utilisateur souhaite obtenir une définition approfondie d'un terme présent dans la base de connaissance. Pour ce faire, le système propose un mécanisme d'accès à la définition du terme en cliquant sur l'élément désiré.

[1] Cette fonctionnalité se veut assez générique pour ne pas répéter inutilement les différentes étapes impliquées dans la consultation des recommandations.

3.2.3 Diagramme des cas d'utilisation - Vue *Administrateur*

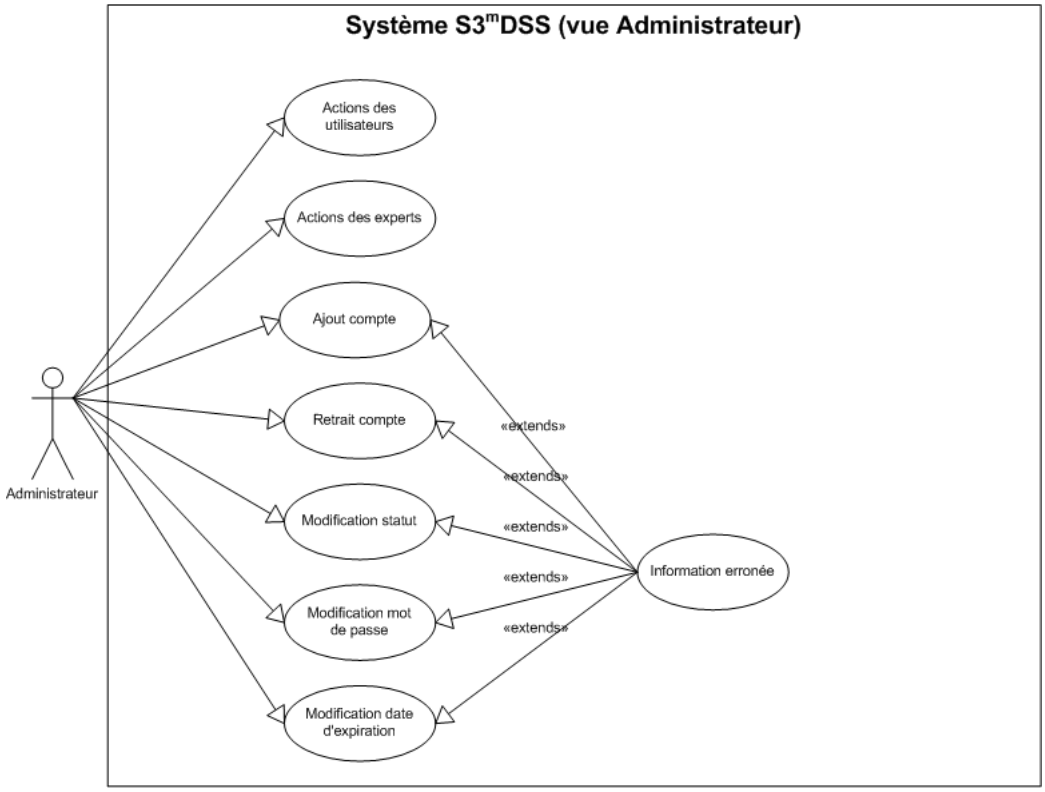


FIG. 3.2 – Diagramme des cas d'utilisation de l'administrateur

Description : Ce système comprend l'ensemble des actions que les administrateurs du système peuvent réaliser.

3.2.4 Fonctionnalités - Vue *Administrateur*

Actions des utilisateurs

Description : Il s'agit ici des fonctionnalités disponibles pour l'utilisateur : *Authentification*, *Désauthentification*, *Consultation de la base de connaissance*, *Emission suggestion*, *Consultation aide générale*, *Consultation aide locale* et *Consultation définition*[2].

Actions des experts

Description : Il s'agit ici des fonctionnalités disponibles pour l'utilisateur : *Ajout dans la base de connaissance*, *Modification dans la base de connaissance* et *Retrait dans la base de connaissance*.

[2]On notera que la vue administrateur permet de consulter les connaissances des évaluateurs internes et externes en même temps.

Ajout compte

Description : Un administrateur souhaite ajouter le droit d'accès au système à un nouvel utilisateur. Pour ce faire, le système lui propose un formulaire comportant un login, un mot de passe, une date d'expiration et un statut à accorder au nouvel utilisateur.

Retrait compte

Description : Un administrateur souhaite retirer le droit d'accès au système à un utilisateur. Pour ce faire, le système lui propose de selectionner un utilisateur à supprimer parmi la liste des utilisateurs du système.

Modification statut

Description : Un administrateur souhaite modifier le statut d'un utilisateur du système. Pour ce faire, le système lui propose un formulaire dans lequel il sélectionnera un utilisateur et un nouveau statut à lui associer.

Modification mot de passe

Description : Un administrateur souhaite modifier le mot de passe d'un utilisateur du système. Pour ce faire, le système lui propose un formulaire dans lequel il sélectionnera un utilisateur et un nouveau mot de passe à lui associer.

Modification date d'expiration

Description : Un administrateur souhaite modifier la date d'expiration d'un compte d'un utilisateur du système. Pour ce faire, le système lui propose un formulaire dans lequel il sélectionnera un utilisateur et une nouvelle date d'expiration à associer à son compte.

3.2.5 Diagramme des cas d'utilisation - Vue *Expert*

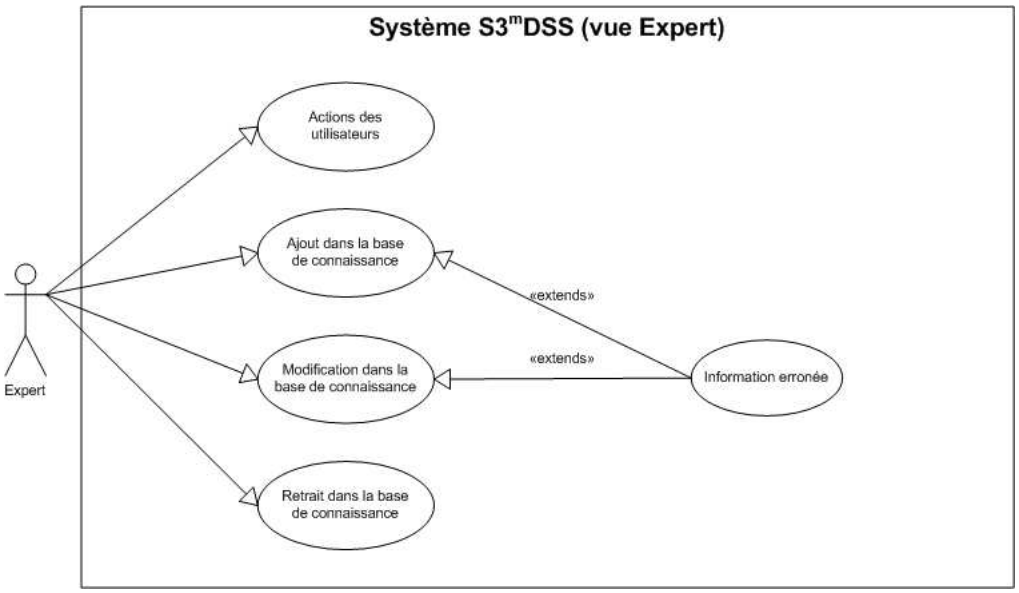


FIG. 3.3 – Diagramme des cas d'utilisation de l'expert

Description : Ce système comprend l'ensemble des actions que les experts du système peuvent réaliser.

3.2.6 Fonctionnalités - Vue *Expert*

Actions des utilisateurs

Description : Il s'agit ici des fonctionalités disponibles pour l'utilisateur : *Authentification*, *Désauthentification*, *Consultation de la base de connaissance*, *Emission suggestion*, *Consultation aide générale*, *Consultation aide locale* et *Consultation définition*[3].

Ajout dans la base de connaissance

Description : Un expert souhaite faire évoluer la base de connaissance en ajoutant un index, mot-clé, concept de maintenance, cas problème, question ou recommandation. Celui-ci peut également revoir les relations agençant ces différentes notions. Le système lui propose de choisir parmi l'un de ces éléments puis affiche un formulaire à compléter.

Modification dans la base de connaissance

Description : Un expert souhaite faire évoluer la base de connaissance en modifiant un mot clé, index, concept de maintenance, cas problème, question ou recommandation.

[3]On notera que la vue expert permet de consulter les connaissances des évaluateurs internes et externes en même temps.

Celui-ci peut également revoir les relations agençant ces différentes notions. Le système lui propose de choisir parmi l'un de ces éléments puis affiche un formulaire à compléter.

Retrait dans la base de connaissance

Description : Un expert souhaite faire évoluer la base de connaissance en supprimant un mot clé, index, concept de maintenance, cas problème, question ou recommandation. Celui-ci peut également revoir les relations agençant ces différentes notions. Le système lui propose de choisir parmi l'un de ces éléments puis affiche une liste des éléments disponibles dans le système.

3.3 Exigences non-fonctionnelles

Les points suivants ont pour but de préciser les caractéristiques du système qui n'ont pas été décrites de façon formelle durant les interviews. Ces exigences non-fonctionnelles constituent un point très important de l'analyse des besoins puisqu'elles déterminent les conditions d'utilisation offertes par le système. Il est important de noter que le système d'exploitation utilisé pour les tests est Microsoft Windows XP SP2 et est basé sur des versions de Apache Tomcat, Java et SQL Server correctement configurées et compatibles. Aucune garantie sur les résultats n'est fournie dans d'autres conditions.

Efficacité

Le système se doit d'être le plus efficace possible afin de permettre aux utilisateurs un confort et des performances optimales. Ces performances sont d'autant plus importantes qu'elles garantissent le dynamisme du système. Le temps de réaction que le système mettra avant de fournir une quelconque réponse à un utilisateur n'excédera par une minute. Il est à notre que ces critères ne sont valides que sous condition d'une disponibilité technique optimale du système. Toute dégradation des performances, due à une défaillance technique ayant un impact sur le système, ne sera pas prise en compte.

Ergonomie

Afin d'offrir la meilleure interface possible et dans la mesure du possible, des analyses seront effectuées sur des interfaces existantes et avec des utilisateurs de profils distincts dans le but de définir un design aussi proche que possible de leur attentes et besoins.

Sécurité

Bien que la sécurité d'un système de données tel que présent ne soit pas une priorité, le système tâchera de garantir la pérennité des données ainsi que l'accès aux informations.

Scalability

Le système devrait pouvoir supporter une augmentation importante du nombre d'utilisateurs et de données relatives au domaine de la maintenance sous réserve d'une amélioration possible de l'infrastructure physique disponible.

Utilisabilité

Le système offrira une interface permettant à chaque catégorie d'utilisateurs de réaliser ses tâches le plus simplement et le plus efficacement possible.

3.4 Conclusion

Ce chapitre a permis de définir formellement les différentes catégories d'utilisateurs impliqués dans le système ainsi que la nature de la différenciation entre évaluateurs internes et externes. Les différents cas d'utilisation du système ont été présentés de même qu'une description des différentes fonctionnalités répondant aux attentes des utilisateurs. Finalement, une analyse des exigences non-fonctionnelles a été proposée.

4 Analyse conceptuelle

Ce chapitre a pour but de présenter par divers diagrammes l'ensemble du système $S3^m$DSS. Nous aborderons la modélisation structurelle de l'application au moyen de diagrammes de robustesse et d'un diagramme de classes puis décrirons la modélisation comportementale du système au moyen de diagrammes de séquence et terminerons par une description de la modélisation des données stockées en base de connaissance.

4.1 Modélisation structurelle

Dans cette partie, nous décrirons la structure de l'application au moyen de diagrammes de robustesse et d'un diagramme de classe. Nous aurons ainsi une bonne vue du système tel qu'il a été implémenté tout en mettant l'accent sur la transition qui a été effectuée entre les cas d'utilisation de l'analyse des besoins et la structure du système actuel. Notons également que nous ne présenterons pas de diagrammes de composants, jugeant que le lecteur trouvera assez d'information dans les diagrammes présents et dans la description détaillée de l'architecture faite dans le chapitre 5.

4.1.1 Diagrammes de robustesse

Les diagrammes de robustesse permettent de raffiner les use cases présentés en Annexe 1 pour en déduire un premier jet d'une découpe en composantes et une esquisse de l'interaction entre ces composantes. Ci-après se trouvent quelques diagrammes de robustesse des fonctionnalités implémentées dans le système. Le lecteur est invité à consulter l'Annexe 2 pour prendre connaissance de la totalité de ces diagrammes

Remarque : Il est à noter que dans tous les diagrammes de robustesse repris ci-dessous (excepté le cas d'authentification), l'acteur sera considéré comme étant préalablement authentifié avant d'effectuer l'opération.

Authentification

Description : L'acteur souhaite s'authentifier auprès du système.

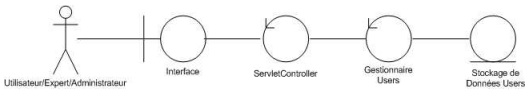


FIG. 4.1 – Diagramme de robustesse - Authentification

Consultation KB

Description : L'acteur souhaite consulter des informations relatives à son problème de maintenance.

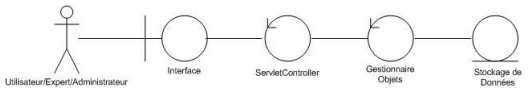


FIG. 4.2 – Diagramme de robustesse - Consultation KB

Ajout compte

Description : L'acteur souhaite ajouter un nouvel utilisateur dans le système.

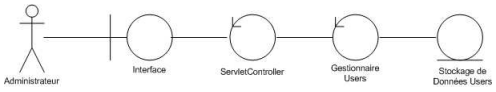


FIG. 4.3 – Diagramme de robustesse - Ajout compte

Ajout de connaissances dans la KB

Description : L'acteur souhaite ajouter un élément de connaissance en KB

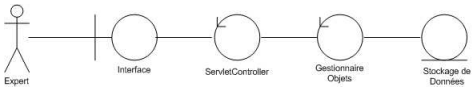


FIG. 4.4 – Diagramme de robustesse - Ajout KB

4.1.2 Diagramme de classes

Le diagramme de classes présente les classes du système $S3^m$DSS ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques. Une classe est un ensemble de fonctions et de données (attributs) qui sont liées ensembles par un champ sémantique. Les classes

sont utilisées dans la programmation orientée objet. Elles permettent de modulariser un programme et ainsi de découper une tâche complexe en plusieurs petits travaux simples.

La figure 4.5 présente le diagramme de classes du système. On notera que les éléments de couleur rose correspondent à la couche Présentation et les rouges à la couche DAO tandis que les éléments verts correspondent aux objets et les bleus aux différents gestionnaires et contrôleurs composant la couche Métier.

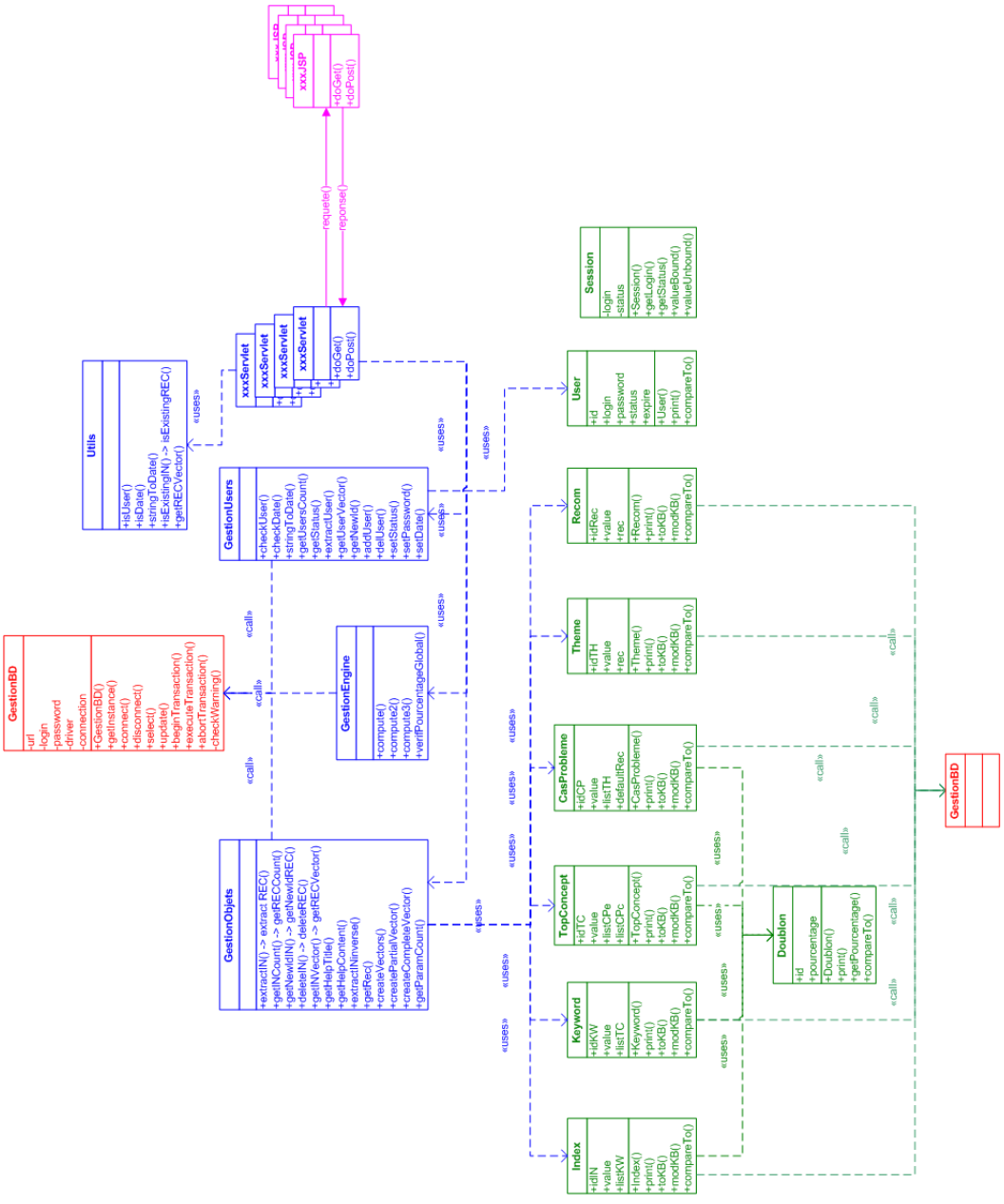


Fig. 4.5 – Diagramme de classes

4.2 Modélisation comportementale

Dans cette partie, nous décrirons le comportement de l'application lors de certaines actions entreprises par l'utilisateur. En ce sens, nous nous attacherons à la dynamique de l'application par le biais de diagrammes de séquences permettant de représenter les collaborations entre gestionnaires et objets selon un point de vue temporel. L'accent est mis sur la chronologie des événements. Ces diagrammes de séquences servent à illustrer les cas d'utilisation présentés précédemment.

4.2.1 Diagrammes de séquences

Ci-après se trouvent les diagrammes de séquences des différentes fonctionnalités. Ne sont pas présentés ici tous les diagrammes de séquences possibles. Seuls ceux qui sont réellement utiles pour la bonne compréhension du système seront présents. La grande majorité des diagrammes non présentés peuvent être déduits des diagrammes présents.

Remarque : Il est à noter que dans tous les diagrammes de séquence ci-après (excepté le cas d'authentification), l'authentification permettant la réalisation de l'opération aura été effectuée. Nous utiliserons dans les diagrammes ci-après le terme utilisateur pour identifier indifféremment l'évaluateur interne et l'évaluateur externe. Notez également que les diagrammes ont été épurés afin de mieux faire transparaître les interactions ciblées.

Authentifier un utilisateur

Description : L'acteur souhaite s'authentifier auprès du système.

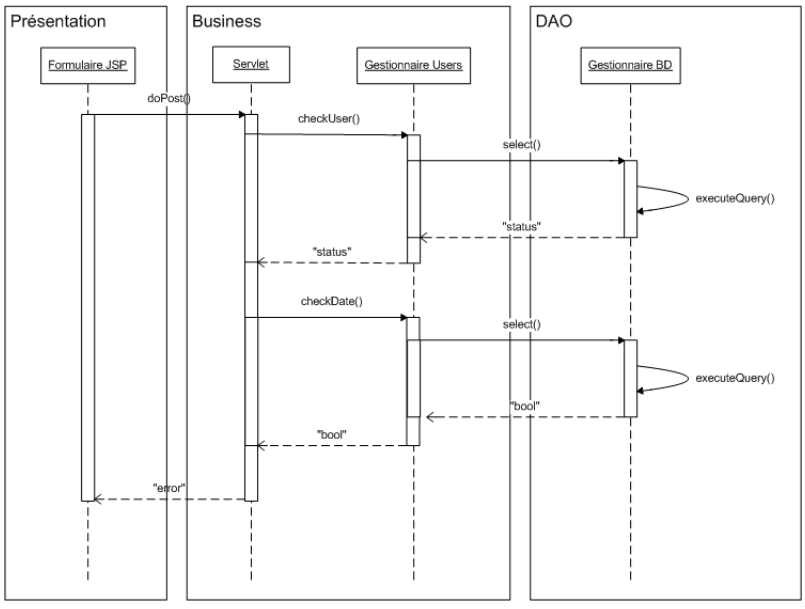


FIG. 4.6 – Diagramme de séquence - Authentification

Consulter une connaissance en KB

Description : L'acteur souhaite consulter une connaissance disponible dans le système. Le cas de l'affichage d'un mot-clé fera office d'exemple. On notera que les interactions inter-servlets ont été masquées.

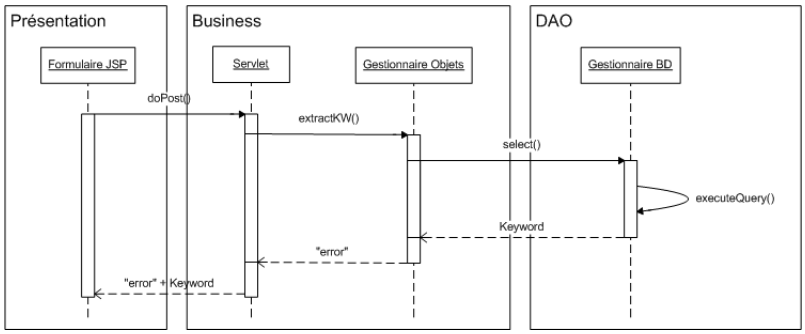


FIG. 4.7 – Diagramme de séquence - Consultation KB

Ajouter un compte utilisateur

Description : L'acteur souhaite ajouter un utilisateur dans le système.

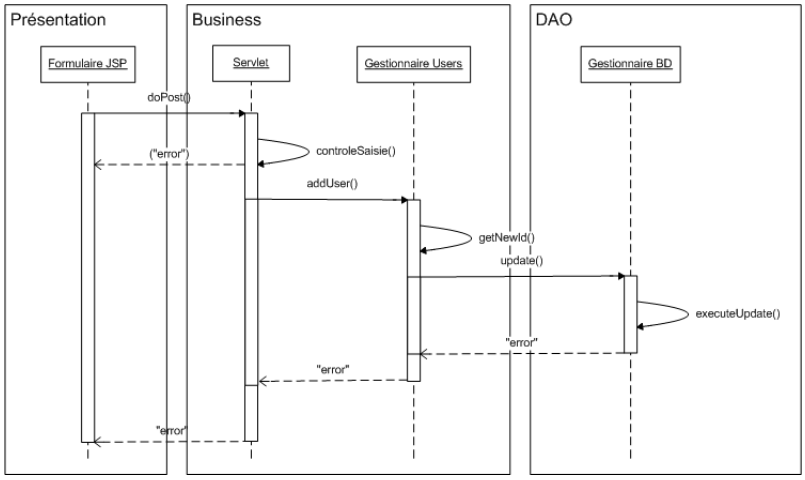


FIG. 4.8 – Diagramme de séquence - Ajout utilisateur

Ajouter de la connaissance en KB

Description : L'acteur souhaite ajouter une connaissance dans le système. Le cas de l'ajout d'une recommandation fera office d'exemple.

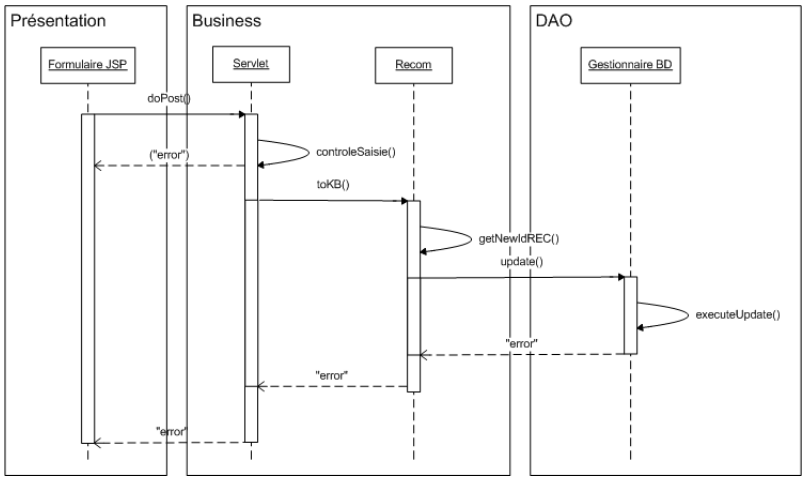


FIG. 4.9 – Diagramme de séquence - Ajout KB

Communication inter-servlets

Description : La communication inter-servlets ayant été masquée ci-haut, il me semblait important de consacrer un diagramme de séquence à ce moyen de communication et ce afin de clarifier les choses.

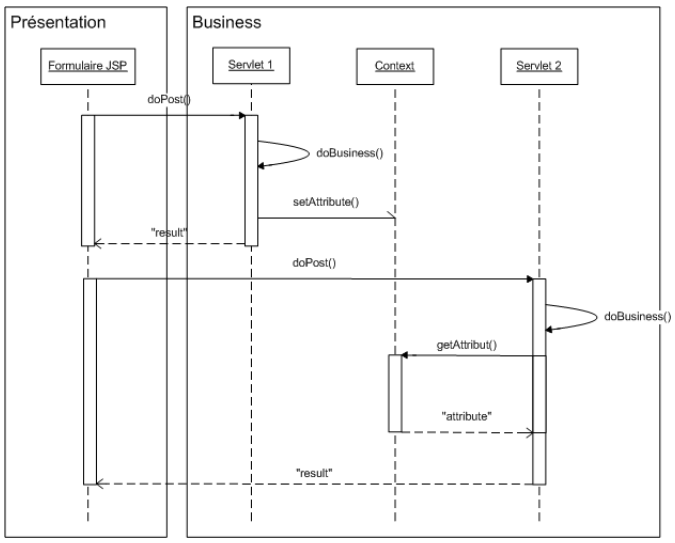


FIG. 4.10 – Diagramme de séquence - Communication inter-servlets

4.3 Modélisation de la persistance

Dans cette partie, nous traiterons de la modélisation de la persistance des données dans le système $S3^m$DSS. Pour ce faire, nous décrirons les données devant être conservées dans le système. Il s'agit des données ayant un haut degré de contenu susceptible d'aider l'utilisateur dans la recherche de recommandations appropriées. Ceci sera présenté sous forme de plusieurs diagrammes.

Notons préalablement que l'application est passée d'un système de sauvegarde basé sur l'XML à un système de base de données de type SQL Server. Cette modification vers un système de base de données a permis d'offrir de meilleurs temps de réaction à l'application. En contre partie, afin d'éviter de lourds travaux de restructuration, les relations existantes entre entités ont continué à être gérée directement dans le langage Java. En conclusion, les mécanismes de relations entre tables de la base de données sont inexistants et s'effectuent directement dans le code. Ce fait ne perturbe en rien l'application qui garanti déjà des temps de réponse adéquats.

Le système $S3^m$DSS est composé d'un ensemble de concepts. Ces concepts font partie intégrante du déroulement des tâches que l'utilisateur doit effectuer mais ils sont également source d'information. Cette information doit être sauvegardée afin d'en assurer la pérennité. Avant de présenter la structure de sauvegarde de ces éléments, il est bon de les définir précisément :

- **Mots de l'index :** Ensemble de mots communément utilisés en génie logiciel.
- **Mots-clés :** Ensemble de mots génériques pouvant être reliés directement avec un concept de maintenance.
- **Concepts de maintenance :** Ensemble de concepts liés au domaine de la maintenance établissant un lien entre un mot-clé et un cas problème dans un voeux de généralisation.
- **Cas problèmes :** Ensemble de problèmes typiques du monde de la maintenance.
- **Thèmes :** Ensemble de questions pertinentes et aiguillant un utilisateur vers une recommandation appropriée.
- **Recommandations :** Ensemble de bonnes pratiques propres au domaine de la maintenance et décrites dans la méthodologie $S3^m$.

La figure 4.11 présente, par un schéma conceptuel, les différents concepts cités et les relations qui existent entre ceux-ci.

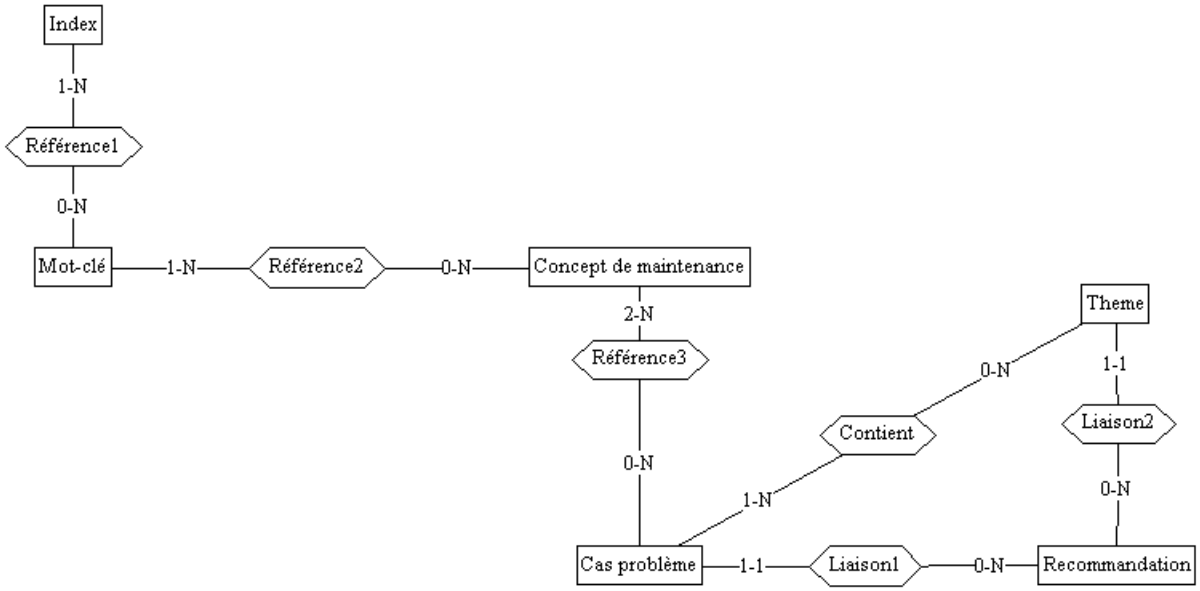


FIG. 4.11 – Schéma conceptuel de $S3^m$DSS

D'un point de vue technique, la figure 4.12 présente les tables définies en base de données.

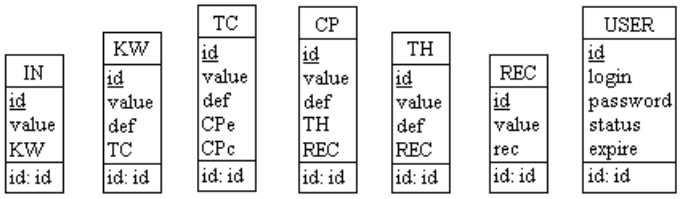


FIG. 4.12 – Diagramme ERA du système$S3^m$DSS

5 Architecture

L'architecture logicielle décrit d'une manière symbolique et schématique les différents composants d'un ou de plusieurs programmes informatiques, leurs interrelations et leurs interactions. Contrairement aux spécifications systèmes résultantes de l'analyse fonctionnelle, l'architecture logicielle, produite lors de la phase de conception, ne décrit pas ce que doit réaliser un programme mais plutôt comment il doit être conçu de manière à répondre aux spécifications. Dans ce chapitre, nous nous attacherons à décrire l'architecture du système $S3^m$DSS. Pour ce faire, nous commencerons par rappeler les principes de base de l'architecture 3-Tiers tout en appliquant la théorie au développement web. Dans un deuxième temps, nous décrirons et justifierons l'architecture du système $S3^m$DSS dans la pratique.

5.1 3-Tiers, en théorie

L'architecture trois tiers est un modèle logique d'architecture applicative visant à distinguer très nettement trois couches au sein d'une même application. Cette dernière est alors considérée comme un empilement de trois couches, niveaux ou strates dont on a clairement défini le rôle. La figure 5.1 schématise ces concepts. Dans un contexte de développement web, on parlera de :

- La présentation de données : correspondant à l'interface web qui permet à l'utilisateur de piloter l'application et d'en recevoir les informations.
- Le traitement métier des données : implémentant les algorithmes « métiers » de l'application. Cette couche est indépendante de toute forme d'interface avec l'utilisateur. Elle doit pouvoir être utilisable aussi bien avec une interface console qu'une interface web, par exemple. Il s'agit généralement de la couche la plus stable de l'architecture, ne changeant pas même si l'on modifie l'interface utilisateur ou la manière dont on accède aux données persistantes de l'application.
- L'accès aux données persistantes : s'occupant de l'accès aux données persistantes, le plus souvent au sein d'un SGBD mais pouvant provenir de bien d'autres technologies.

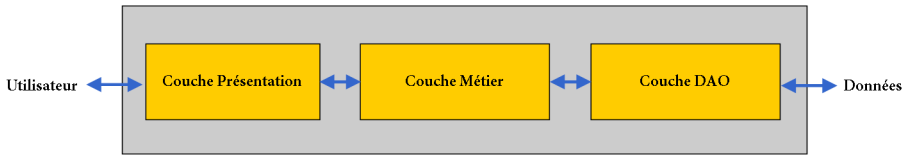


FIG. 5.1 – Architecture 3-Tiers

Les différentes couches du modèle communiquent entre elles au travers d'un « modèle d'échange ». Chacune des trois couches propose un ensemble de services rendus qui sont mis à disposition de la couche supérieure. On interdira alors qu'une couche invoque les services d'une couche plus basse que la couche immédiatement inférieure ou plus haute que la couche immédiatement supérieure, chaque couche communique avec ses voisines immédiates. La figure 5.2 présente ce concept. Chaque couche possède un rôle et une interface de communication bien définie, ceci permet dans l'absolu de faire évoluer individuellement ces strates sans induire de changement dans les autres couches. On notera cependant qu'une nouvelle fonctionnalité de l'application peut avoir des répercussions dans plusieurs couches.

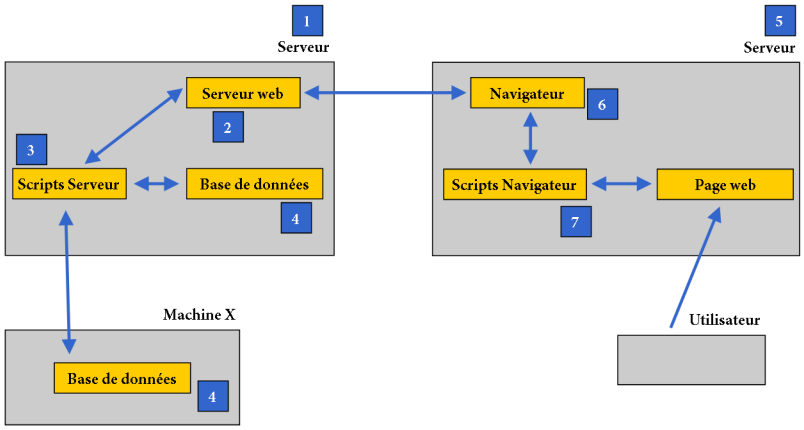


FIG. 5.2 – Composantes d'une application web, adapté de [26]

Sur un plan plus technique, l'approche 3-Tiers permet de décomposer une application en trois parties distinctes :

- L'interface : c'est à ce niveau qu'intervient le dialogue entre l'utilisateur et le système ; les échanges entre l'interface et le processus se font via des événements.
- Le processus : il décrit la logique de l'application ; les communications entre le processus et les objets se font via des requêtes.
- Les objets : ils référencent les données et les règles spécifiques du métier ; ils sont appelés dès que nécessaire.

Finalement, on considèrera les avantages multiples de cette architecture :

- La manipulation des données est indépendante du support physique de stockage ;
- Le moteur d'accès aux données est centralisé ;
- La maintenance des traitements est facilitée ;
- La vision des traitements depuis la couche de présentation est amplement simplifiée ;
- Le portage d'un environnement graphique à un autre est très aisé ;
- Le travail en équipe est optimisé.

5.2 3-Tiers, en pratique

Le système $S3^m$DSS a été conçu selon l'architecture 3-Tiers telles qu'elle a été présentée ci-dessus. Il semble, cependant, intéressant de décrire précisément comment l'architecture a été respectée dans le cadre du projet. Une vue de haut niveau sous forme de diagramme de déploiement est présentée par la figure 5.3. Nous nous attacherons ensuite à décrire plus précisément les 3 couches présentes dans le système $S3^m$DSS.

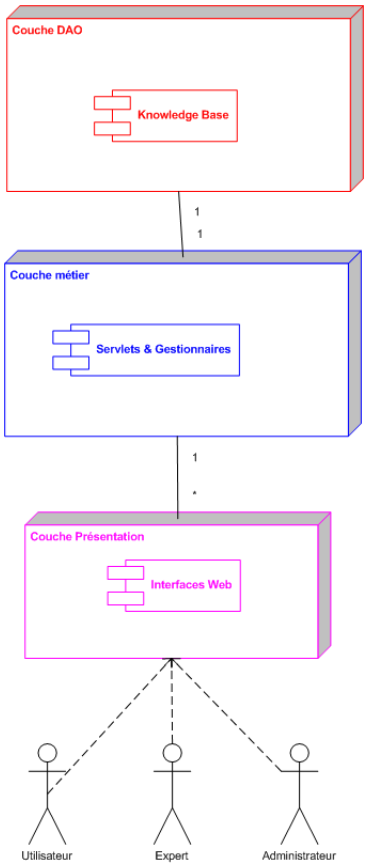


FIG. 5.3 – Diagramme de déploiement de $S3^m$DSS

5.2.1 Couche Présentation

La couche Présentation regroupe toutes les pages JSP correspondants aux vues des utilisateurs. Y sont inclus l'ensemble des formulaires au moyen desquels l'utilisateur peut introduire des données dans le système. Un feedback est aussi systématiquement renvoyé à l'utilisateur lorsqu'il effectue une action. A ce propos, Java propose un mécanisme qualifié d'exception et consistant à effectuer les instructions dans un bloc d'essai (le bloc try) qui surveille les instructions. Lors de l'apparition d'une erreur, celle-ci est lancée dans un bloc de traitement d'erreur (le bloc catch) sous forme d'un objet appelé Exception. Le bloc de traitement d'erreur va lever l'exception. Dans la pratique, ces erreurs ont toutes été dirigées de manière générique vers l'interface et renvoyées distinctement via un message à l'utilisateur selon le type d'erreur survenu. La figure 5.4 décrit précisément les gestionnaires présents dans la couche Présentation du système $S3^m$DSS.

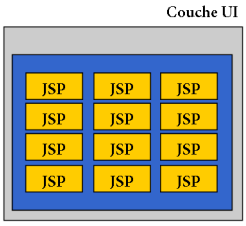


FIG. 5.4 – Structure de la couche UI de $S3^m$DSS

Notons qu'un effort particulier a été opéré afin de garantir un minimum de code Java au sein de l'HTML formant les pages JSP. Ceci afin de ne pas mélanger différents types de programmation et facilitant la maintenance du système. La séparation des tâches entre couches n'en est que d'autant plus respectée puisque la couche présentation concentre son effort sur l'affichage. Notons finalement que ce type de programmation permet la révision facile des interfaces par des infographistes sans soucis de changement du business de l'application ni manipulation de langages complexes.

5.2.2 Couche Métier

La couche Métier comprend l'ensemble des fonctionnalités business du système S3mDSS. Son objectif principal est de regrouper l'ensemble des opérations de 'Business' du système. Elle peut être subdivisée en trois grandes composantes :

- Contrôleur : son rôle est d'effectuer les contrôles usuels sur les saisies de l'utilisateur ;
- Business : son rôle est d'effectuer les fonctions métier de l'application ;
- Objets : son rôle est de fournir un ensemble d'objets avec lesquelles le système peut opérer.

Contrôleur

La composante « Contrôleur » comprend un ensemble de gestionnaires appelés "servlets". Chacun d'entres eux est associé à une vue (ou formulaire) de la couche Présentation.

Lors de tout envoi d'information par l'utilisateur au système, le servlet adéquat prend la main du côté serveur. Il est le seul point de communication entre la couche Présentation et la couche Métier. Il effectue un contrôle de saisie sur les informations reçues et, le cas échéant, averti l'utilisateur des erreurs qu'il a pu trouver. Le servlet appellera ensuite les fonctions nécessaires de la composante Business afin de traiter l'information puis renverra une réponse à l'utilisateur. Cette chaîne de communication peut être modélisée de la manière suivante : UI¡–¿servlet(¡–¿servlet)¡–¿gestionnaire(¡–¿gestionnaire)¡–¿DAO.

Business

La composante « Business » comprend un ensemble de gestionnaires traitant directement avec la couche DAO. Ces gestionnaires *engine*, *users* et *objets* regroupent respectivement les méthodes de traitement propres au moteur de calcul des recommendations, aux utilisateurs et aux objets du système. La communication relative à l'extraction de données entre la couche métier et la couche DAO s'effectue via ces gestionnaires. Ils ont également la charge de transmettre des informations directement exploitables par les autres composantes de la couche Métier.

Objets

La composante « Objets » comprend un ensemble d'objets avec lesquelles le système travail. Ces objets regroupent les différentes informations disponibles en KB et disposent également de méthodes spécifiques leur permettant de mettre à jour la base de connaissance.

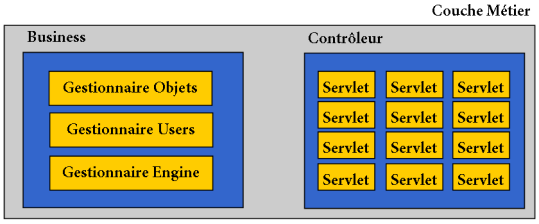


FIG. 5.5 – Structure de la couche Business de $S3^m$DSS

5.2.3 Couche DAO

La couche DAO comprend l'ensemble des mécanismes d'accès à la base de connaissance. Dans le premier prototype de $S3^m$DSS, la base de connaissance a été implémentée via un ensemble de fichiers XML. Cette technique, encouragée par le commanditaire, a montré ses faiblesses lorsque le système a commencé à être alimenté de connaissance. Il s'avère en effet que même si l'idée est honorable, les mécanismes de « parsing » de fichiers sont inefficaces lors du traitement de milliers de lignes. L'exemple le plus représentatif est la constitution d'une liste des mots de l'index présents dans la base de connaissance afin d'en afficher une liste directement exploitable par l'utilisateur. Cette simple construction demande un certain nombre d'accès aux fichiers XML qui devient rapidement un poids

pour le système. Différentes techniques d'optimisation ont été réalisées dans le but de diminuer les temps de traitements. Un exemple parmi d'autres a été de profiter des temps de saisie d'information dans les formulaires par l'utilisateur pour permettre au système de mettre en mémoire un certain nombre d'éléments.

Malgré tout, après différents tests, il est devenu vital pour l'interactivité du système de passer à un second prototype construit autour d'une base de donnée. Il est en pratique composé d'un gestionnaire de base de données de type JDBC permettant d'accéder au SGBD et donc indirectement aux données présentes en BD.

Le changement de type de stockage de données, bien qu'il puisse être considéré comme une faiblesse dans l'analyse technique initiale, est en soi à considérer comme un élément positif pour le système. Il a, en effet, montré la facilité avec laquelle un composant peut-être modifié dans une architecture de type 3-Tiers. Les modifications n'ont pas perturbé les autres couches de l'architecture et se sont faites assez rapidement dans leur ensemble. Il a tout de même été nécessaire de construire une petite application permettant de faire migrer l'ensemble des données présentes sous formes de fichiers XML vers des tables disponibles en base de données.

Notons également que la couche DAO suit le modèle de création « singleton » et ce afin de faciliter la réutilisation et la maintenance du code mais aussi afin de garantir un nombre d'instanciation raisonnable du gestionnaire sur le serveur. Le modèle de création « singleton » sert à contrôler le nombre d'instances d'une classe présent à un moment donné. Ce modèle est très pratique dans le cas présent où la classe se trouve sans état et effectue toujours les mêmes traitements. Notons finalement que le singleton limite le nombre d'instances en mémoire et est parfois perçu comme une fabrique particulière. On pourra citer les avantages de distinction de la couche DAO permettant un regroupement des accès à la base de connaissance ainsi qu'une grande modularité ayant permis l'utilisation d'un SGBD à la suite d'un traitement XML. La figure 5.6 décrit précisément les gestionnaires présents dans la couche DAO du premier et du second prototype $S3^m$DSS.

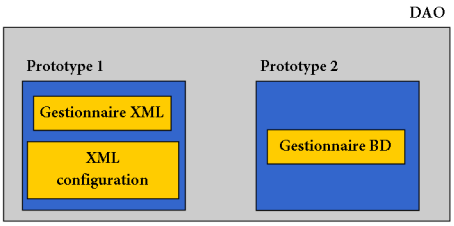


FIG. 5.6 – Structure de la couche DAO de $S3^m$DSS

5.3 Conclusion

En conclusion, nous avons vu les avantages d'une architecture 3-Tiers dans un environnement de développement Web ainsi que son déploiement dans le système $S3^m$DSS. Elle

est composée d'une couche de présentation, d'une couche métier et d'une couche DAO. La couche présentation comporte l'ensemble des vues de l'utilisateur. La couche métier est subdivisée en 3 composantes : contrôleur, business et objets. La couche DAO comporte, quant à elle, les mécanismes d'accès à la base de connaissance. Un cas d'application pratique de la modularité de l'architecture 3-Tiers a également été précisé.

6 Choix technologiques

L'implémentation du système $S3^m$DSS s'est construite autour de certains choix technologiques qu'il parait important de décrire et justifier. Il est à noter que le système s'articule autour de nombreuses technologies : *Java*, *Java Server Pages*, *JavaScript*, *HTML*, *CSS*, *XML*, *Apache Tomcat* et *Microsoft SQL Server*. Ce cocktail de technologie n'a pas été utilisé sans raison. Cette section se focalisera sur les différentes technologies introduites dans le système en les présentant d'une manière formelle mais aussi en les justifiant dans le contexte du système qui nous occupe.

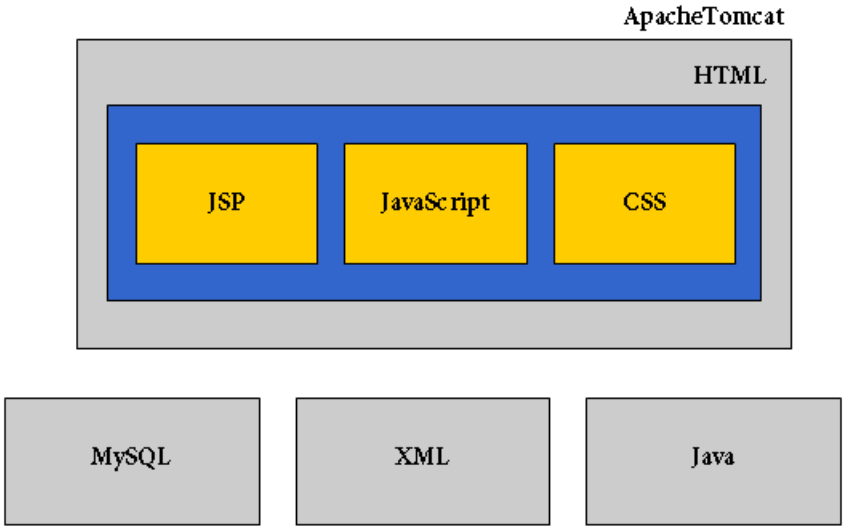


FIG. 6.1 – Technologies de $S3^m$DSS

6.1 HTML

Description

L'Hypertext Markup Language ou généralement HTML est un langage de programmation utilisé pour écrire des pages web. La fonctionnalité première de l'HTML est de pouvoir insérer des hyperliens dans du texte et donc de créer de l'hypertexte. L'HTML est

considéré comme un langage de description de documents mais aussi comme un langage de balisage. Ceci est compris dans le sens où les balises servent de zone de délimitation ou de contenu en elles-mêmes.

Les documents HTML sont identifiés par une URL et interprétés par les navigateurs web après transmission via le protocole de communication http d'un serveur http à un client. Il est possible qu'une partie du document soit interprétée directement au sein du serveur. Ce principe est, en l'occurrence, en vigueur avec la technologie JSP. Notons finalement, qu'un voeux d'interopérabilité a motivé un travail commun sur les spécifications de l'HTML afin de permettre à ces documents d'être accessibles sur des plates-formes et navigateurs différents : ordinateurs personnels, téléphones cellulaires, appareils portables, et ainsi de suite (adapté de [36]). La figure 6.2 montre un exemple de code HTML.

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html;
       charset=ISO-8859-1">
4      <link rel="stylesheet" type="text/css" href="../
          default.css" />
5          <title>Help</title>
6  </head>
7  <body bgcolor=#808080>
8      <h5>Maintenance Concept</h5>
9      <aide>It is the intersection between ...</aide>
10 </body>
11 </html>
```

FIG. 6.2 – Exemple de code HTML tiré de $S3^m$DSS

Justification

L'utilisation du langage de programmation HTML au sein du système $S3^m$DSS se justifie dans le besoin fondamental de créer un système accessible par l'Internet. Ceci est du au caractère expérimental du système. Il est en effet essentiel, dans une première phase, de pouvoir distribuer le logiciel facilement à divers experts en maintenance ou clients potentiels et, par la suite, de pouvoir collecter leurs remarques. Le moyen retenu a donc été de travailler sur un système accessible par le web. Ce choix nous a donc poussé à travailler sur une interface de type HTML mais ayant un certain nombre de particularités permettant de faire tourner un serveur de données ainsi qu'un système de calcul de recommandations en back-office, rôle du couple JSP-Java.

6.2 CSS

Description

Le langage Cascading StyleSheets (CSS) ou feuilles de style en cascade est utilisé pour décrire la présentation d'un document structuré écrit en HTML. La direction de ce langage incombe au World Wide Web Consortium (W3C). L'objectif principal de ce langage est de séparer la structure et la présentation d'un document. Cette séparation fournit un certain nombre de bénéfices, permettant d'améliorer l'accessibilité, de changer plus facilement de structure et de présentation, et de réduire la complexité de l'architecture d'un document. On constate cependant que CSS est géré différemment selon les navigateurs web utilisés. En l'occurrence, Mozilla Firefox reste celui qui interprète le mieux CSS tandis qu'Internet Explorer reste le pire lorsqu'il s'agit de rendre correctement les feuilles de style paramétrées suivant les règles du W3C. C'est à ce titre que l'utilisation du premier est conseillée pour l'utilisation du système (adapté de [32]). La figure 6.3 montre un exemple de code CSS.

```
1 th.login {
2 font-family: inherit;
3 font-size:large;
4 border: thin solid #6495ed;
5 padding: 5px;
6 text-align:left;
7 background-color: #D0E3FA;
8 background-image: url(sky.jpg);
9 }
```

FIG. 6.3 – Exemple de code CSS tiré de $S3^m$DSS

Justification

L'utilisation du langage de CSS au sein du système $S3^m$DSS se justifie, dans un premier temps, par la réduction de la complexité de l'architecture des pages HTML du système. Cet effort, loin d'être anecdotique, a du être intégré compte tenu du fait que les pages web du système comportent déjà un mélange de langages HTML et JSP complexifiant le code. Dans une moindre mesure, les optimisations faites par le biais de CSS peuvent être comparées à celles faites sur le code JSP directement intégré dans les pages HTML. Dans un deuxième temps, se situant dans un contexte de maintenance, il sera beaucoup plus aisé à l'avenir de revoir le système grâce à la distinction logique qui a été opérée entre structure et présentation.

6.3 JavaScript

Description

JavaScript est un langage de programmation de type script, orienté objets à prototype, principalement utilisé dans les pages web. Conçu par la Netscape Communications Corporation, le langage était initialement nommé LiveScript et développé pour proposer un langage de script côté serveur. La version orientée client suivra quelques temps plus tard et sera rebaptisée JavaScript suite à une entente entre Netscape et Sun Microsystems. Le langage de programmation fût rapidement intégré au navigateur web Netscape Navigator dont le succès contribua à l'adoption rapide de JavaScript dans le développement web orienté client. Notons également que JavaScript implémente les spécifications du langage ECMAScript, standard que l'on peut retrouver dans le document ECMA-262 rédigé par l'European Computer Manufacturers Association, organisation de standardisation active dans le domaine de l'informatique.

D'un point de vue pratique, JavaScript peut être directement intégré au sein de pages web et exécuté sur le poste client. Le navigateur web prend alors en charge l'exécution des morceaux de code appelés *scripts*. Le langage est vu comme un complément de l'HTML, proposant ce que l'on appelle parfois l'HTML dynamique. En somme, l'utilisation de JavaScript permet de réaliser des services dynamiques, parfois futiles ou strictement cosmétiques mais néanmoins souvent nécessaires à l'obtention de fonctionnalités désirées (adapté de [35]). La figure 6.4 montre un exemple de code JavaScript.

```
1 <script language="JavaScript">
2 function submitForm(type){
3       document.formEXP.tmp.value=type;
4       document.formEXP.submit();
5    }
6 </script>
```

FIG. 6.4 – Exemple de code JavaScript tiré de $S3^m$DSS

Justification

L'utilisation du langage de programmation JavaScript au sein du système $S3^m$DSS se justifie dans la complémentarité des fonctionnalités que ce langage apporte à l'HTML. Les possibilités de ce dernier sont parfois assez limitées et il a été nécessaire de recourir au JavaScript afin d'apporter un certain dynamisme au système tel que la formation de "pop-up". Notons également que l'utilisation du JavaScript est assez limitée dans le système mais que dans la majorité des cas, elle s'est révélée essentielle dans le sens où le langage s'est retrouvé à la croisée de l'HTML et du Java. Il a permis de faire le lien entre les servlets Java et les "vues-utilisateur" dans des cas où il était très ardu de faire autrement.

6.4 XML

Description

L'Extensible Markup Language (XML) ou langage de balisage extensible est un langage de balisage générique. Son développement a commencé en 1996 et est devenu une norme du W3C depuis 1998. Son objectif est de faciliter la communication automatisée des contenus entre systèmes d'informations hétérogènes. Ceci est réalisé par une structuration des données un peu comme dans l'HTML par le biais de balises. Notons également qu'à proprement dit, XML est une famille de technologies, XML 1.0 définit les « balises » et les « attributs » mais autour de cette spécification, un nombre de plus en plus important de modules facultatifs fournissant des ensembles de balises et d'attributs ou des lignes directrices pour des tâches particulières ont été définis (adapté de [33]). La figure 6.5 montre un exemple de code XML.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ct-list>
3   <ct>
4     <id>0</id>
5     <value>.Empty</value>
6     <CPe>0-100</CPe>
7     <CPc>0-100</CPc>
8   </ct>
9 </ct-list>
```

FIG. 6.5 – Exemple de code XML tiré de $S3^m$DSS

Justification

Le choix d'effectuer un système de sauvegarde de données via la technologie XML est en soi une bonne idée. Le langage permet facilement de stocker un certain nombre d'éléments au sein de fichiers distincts. C'est d'ailleurs ce choix qui a été fait dans les systèmes analogues à $S3^m$DSS et qui m'a été proposé au lancement de la phase de conception du système. Néanmoins, cette technologie souffre d'un défaut majeur : le traitement. Traiter du XML demande des bibliothèques dédiées afin d'effectuer un « parsing » sur les fichiers. Ce « parsing » fonctionne dans l'absolu très bien mais commence à montrer ses faiblesses lorsque les fichiers deviennent très gros. Ce cas problème est survenu lorsque la base de connaissance du système a commencé à se remplir et que les fichiers XML ont atteint plusieurs milliers de lignes. Le système a donc perdu de son interactivité. Une modification a par la suite été faite afin de passer à un système de SGBD et garantir ainsi des temps d'extraction de données réduits. Ceci est repris plus précisément dans le chapitre **??**.

6.5 Java

Description

Java est une technologie développée par Sun Microsystems composée d'un langage de programmation et d'un environnement d'exécution. La langage Java a la particularité d'être portable sur plusieurs systèmes d'exploitation par le biais de la Java Virtual Machine (JVM), programme interprétant le code Java et le convertissant en code natif. C'est cette garantie de portabilité qui a fait la réussite de Java dans les architectures client-serveur en facilitant la migration entre serveurs, très difficile pour les gros systèmes. La figure 6.6 montre un exemple de code Java.

```
1 static GestionBD gBD = null;
2   public static GestionBD getInstance() throws
       SQLException{
3     if (gBD==null) {
4       gBD = new GestionBD();
5       gBD.connect();
6     }
7     return gBD;
8   }
```

FIG. 6.6 – Exemple de code Java tiré de $S3^m$DSS

Justification

L'utilisation du langage de programmation Java au sein du système $S3^m$DSS se justifie dans sa facilité de combinaison avec les technologies JSP, HTML, XML et SQL Server. En choisissant Java comme langage, un gain précieux de temps a également été fait, dans le sens où le langage est déjà connu et ne nécessite pas de période d'adaptation pour le maîtriser.

6.6 Java Server Pages

Description

Le Java Server Page ou JSP est une technologie basée sur Java permettant à la fois de générer dynamiquement du code HTML ou tout autre type de page web, mais aussi d'ajouter des extensions au HTML par la création de balises spéciales à intégrer dans le code de la page web. Les JSP sont compilées par un compilateur afin de devenir des servlets Java. Un compilateur JSP peut générer un servlet Java en code source Java qui peut à son tour être compilé par le compilateur Java, ou peut générer le pseudo-code Java interprétable directement. En pratique, JSP met en place un conteneur de servlets

fournissant les services réseaux par lesquels les requêtes et réponses sont émises. Il décode toutes les requêtes et formate les réponses dans le format approprié. L'approche servlets garanti un temps de lancement réduit, une gestion du multi-threading ou encore du suivi de session, important pour la sécurité (adapté de [34]). La figure 6.7 montre un exemple de code JSP.

```
<%
   int error=(Integer)request.getAttribute("error");
   switch(error){
   case 0:
   %>
   <center><h6>User correctly added to system</h6></
      center>
   <%
   break;
   case 1:
   %>
    <center><h6>Login already exists</h6></center>
   <%
   break;
}
%>
```

FIG. 6.7 – Exemple de code JSP tiré de $S3^m$DSS

Justification

L'utilisation du langage de programmation JSP au sein du système $S3^m$DSS se justifie premièrement dans la nécessité d'ajouter un certain dynamisme aux pages HTML du système et dans la facilité d'interaction avec le serveur Java.

6.7 Apache Tomcat

Description

Apache Tomcat est un serveur d'applications libre issu du projet Jarkata qui agit comme un conteneur de servlet J2EE. Tomcat a l'avantage d'implémenter les spécifications des servlets et des JSP de Sun Microsystems. On peut également considérer Tomcat comme un serveur http, celui-ci incluant un serveur http interne. Notons également que Tomcat est écrit en langage Java, il peut donc s'exécuter via la JVM sur n'importe quel système d'exploitation (adapté de [31]).

Justification

L'utilisation du serveur d'application Tomcat est une nécessité pour faire tourner le système $S3^m$DSS, il se justifie donc par l'utilisation des langages associées JSP. Néanmoins, le système propose nombre de versions et devient au fils des années un standard incontournable.

6.8 Microsoft SQL Server

Description

Microsoft SQL Server est un SGBD développé et commercialisé par Microsoft. Ce système était développé à l'origine par Microsoft et Sysbase. Suite à un différent entre les deux sociétés, chacune a continué le développement de son côté. Microsoft a choisi de le commercialiser sous le nom de SQL Server. De son côté Sybase, pour éviter toute confusion, a renommé Sybase SQL Server en Sybase Adaptive Server Enterprise. Microsoft SQL Server fait désormais partie de la stratégie technique de Microsoft en matière de base de données (adapté de [37]). La figure 6.8 montre un exemple de code SQL.

```
1 CREATE TABLE 'USER' (
2 'id' INT NOT NULL ,
3 'login' VARCHAR( 100 ) NOT NULL ,
4 'password' VARCHAR( 100 ) NOT NULL ,
5 'status' VARCHAR( 100 ) NOT NULL ,
6 'expire' VARCHAR( 8 ) NOT NULL
7 );
8 ALTER TABLE 'USER' ADD PRIMARY KEY ( 'id' );
```

FIG. 6.8 – Exemple de code SQL tiré de $S3^m$DSS

Justification

L'utilisation du serveur de bases de données SQL Server se justifie dans un sens général où il a fallu passer d'un mode de stockage via XML à une technologie de SGBD beaucoup plus rapide. D'autre part, le type de serveur a été imposé par l'infrastructure présente au sein du laboratoire de développement où le système doit être déployé.

7 Démarche corrective

Le test est un processus manuel ou automatique, qui vise à établir qu'un système vérifie les propriétés exigées par sa spécification, ou à détecter des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification. Ainsi le test vise à mettre en évidence les erreurs d'un logiciel. Cependant il n'a pour objectif ni de diagnostiquer la cause des erreurs, ni de corriger les erreurs, ni même de prouver la correction d'un programme [12]. Il existe différentes façon de classer les tests informatiques. Parmi celles-ci figure le classement par niveau de détail, dépendant essentiellement de l'endroit où l'on se situe dans le cycle de vie. Dans le chapitre suivante, nous présenterons les jeux de test principaux effectués au sein du système, se subdivisant en 2 parties : les tests unitaires et les tests d'intégration.

7.1 Tests unitaires

Le test unitaire est un procédé permettant de s'assurer du fonctionnement correct d'une partie déterminée d'un logiciel. Il s'agit pour le programmeur de tester un module, indépendamment du reste du programme, ceci afin de s'assurer qu'il répond aux spécifications fonctionnelles et qu'il fonctionne correctement en toutes circonstances. Cette vérification est considérée comme essentielle, en particulier dans les applications critiques. Le test unitaire doit être rejoué après une modification du code afin de vérifier qu'il n'y a pas de régressions (l'apparition de nouveaux dysfonctionnements) [39].

Les tests unitaires ont été largement utilisés tout au long de la conception du système. Ils ont permis de mettre à jour un nombre conséquent de problèmes. Néanmoins, au fils du temps, les interactions entre les différents modules étant très importantes, il est devenu difficile d'effectuer des tests sur des morceaux de code sans les intégrer dans un jeu de tests faisant intervenir de nombreux autres composants. Mais considérons malgré tout que chaque module a été testé singulièrement lors de sa phase de codage et ce afin de s'assurer de leur bon fonctionnement.

7.2 Tests d'intégrations

Le test d'intégration est un test qui se déroule dans une phase d'un projet informatique suivant les tests unitaires. Il consiste, une fois que les développeurs ont chacun validé leurs

développements ou leurs correctifs, à regrouper leurs modifications ensemble dans le cadre d'une livraison. L'intégration a pour but de valider le fait que toutes les parties développées indépendamment fonctionnent bien ensemble [38].

Afin d'effectuer une première correction du code et de vérifier que toutes les fonctionnalités ont bien été implémentées, nous avons utilisé une *check-list* informelle. Les tests sont regroupés en thèmes. Chaque thème comporte un certain nombre de rubriques à tester. Ces rubriques ont été combinées entre elles afin de reproduire tous les cas de figure. La *check-list* est présentée ci dessous :

- Phase d'authentification
  - Login correct
  - Login incorrect
  - Password correct
  - Password incorrect
- Droits d'utilisation
  - Contrôle administrateur
  - Contrôle expert
  - Contrôle evaluateur interne
  - Contrôle evaluateur externe
- Fonctionnalités diverses
  - Accès à l'aide générale
  - Accès à la suggestion
  - Accès logout
- Fonctionnalités administrateur
  - Ajout utilisateur
    - Login correct
    - Login incorrect
    - Password correct
    - Password incorrect
    - Date d'expiration correct
    - Date d'expiration incorrect
  - Retrait utilisateur
  - Modification statut
  - Modification password
    - Password correct
    - Password incorrect
  - Modification date d'expiration
    - Date d'expiration correct
    - Date d'expiration incorrect
  - Vue utilisateurs
- Fonctionnalités expert
  - Help
  - Ajout recommandation

  - Modification recommandation
  - Retrait recommandation
  - Ajout thème
  - Modification thème
  - Retrait thème
  - Ajout cas problème
  - Modification cas problème
  - Retrait cas problème
  - Ajout concept topologique
  - Modification concept topologique
  - Retrait concept topologique
  - Ajout mot-clé
  - Modification mot-clé
  - Retrait mot-clé
  - Ajout index
  - Modification index
  - Retrait index
- Fonctionnalités evaluateurs
  - Recherche index
    - Mot connu
    - Mot inconnu (avec/sans casse)
    - Tableau associé valide + help associée
  - Recherche mot-clé
    - KW list
    - Bouton search
    - Bouton definition
    - Bouton ?
  - Recherche TC
    - Bouton ?
    - TC correct (Nom et pourcentage)
    - Help associée
    - Garde puce
  - Recherche CP
    - Bouton ?
    - CP correct (Nom et pourcentage)
    - Help associée
    - Garde puce
  - Recherche TH
    - Bouton ?
    - TH correct (Nom)
    - Help associée
    - Garde Facts
  - Recherche REC
    - Bouton ?

- REC correct (Nom)
- Recommandation associée Gen
- Recommandations associées Combinées

8 Interfaces

Une interface permet d'échanger des informations entre l'utilisateur humain et la machine. L'interface Web est un exemple d'interface homme machine constituée de pages web. Une interface Web est généralement accessible par un navigateur Web. Ce chapitre présente quelques interfaces web types proposée par le système $S3^m$DSS. Les choix effectués dans la présentation des données seront motivés sans pour autant entrer dans une analyse complète de type IHM. Le lecteur pourra consulter l'entièreté des interfaces du système en le consultant directement à l'adresse suivante : http://www.gelog.etsmtl.ca/S3mDSS/.

8.1 Entrée dans le système

La figure 8.1 présente le premier écran disponible à l'entrée du système. L'utilisateur est invité à insérer son login et mot de passe pour utiliser le système. Cette étape initiale permet de filtrer les informations ultérieures qui sont présentées à l'utilisateur en fonction du type de compte attribué à celui-ci. Un lien permet également à l'utilisateur non enregistré de faire une demande d'inscription au projet par le biais d'un client mail. Le logo présentant l'application a été entièrement refait pour correspondre au nouveau nom du système. Notons finalement que les couleurs de l'application telles que le fond d'écran ou les tableaux sont générées à partir d'une feuille CSS et homogènes avec le reste de l'application.

FIG. 8.1 – Ecran d'entrée du système $S3^m$DSS

8.2 Sélection des modes

La figure 8.2 présente les différents menus de navigation disponibles dans le système. Notons que ces barres sont accessibles depuis n'importe quel écran du système. La barre supérieure permet de naviguer entre les différentes fonctionnalités primaires de l'application. Elle est entièrement ajustée selon le type d'utilisateur que le système détecte. Elle permettra, entre autre et le cas échéant, de basculer entre les modes administrateurs, experts et évaluateurs mais aussi de consulter l'aide générale du système, d'émettre une suggestion ou de quitter l'application. La barre inférieure permet, quant à elle, de démarrer une recherche dans la base de connaissance. La partie de gauche permet d'entreprendre sa recherche par la saisie d'un mot dans un dictionnaire présent dans le système. La partie de droite permet de sélectionner un mot-clé par une liste mais aussi de prendre connaissance de sa définition.

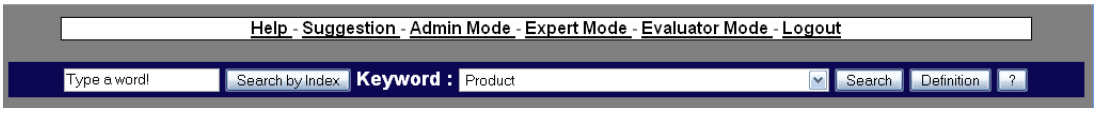


FIG. 8.2 – Menus du système $S3^m$DSS

8.3 Section évaluateur

La figure 8.3 présente la section utilisateur du système. Elle est composée d'un ensemble de tableaux regroupant un certain nombre d'éléments. Les tableaux constituent des concepts qu'il est possible de préciser par le biais des boutons d'aide disponibles. Notons que les tableaux sont construits dynamiquement et apparaissent à mesure que l'utilisateur sélectionne l'un ou l'autre concept. Le dernier tableau renvoie un hyperlien vers un ensemble de recommendations que l'utilisateur pourra consulter. Les choix entre éléments se font via l'utilisation de puces à cliquer. Une définition plus détaillée ou un complément d'information est systématiquement proposé pour chaque élément et accessible par simple clic. Notons finalement que le cadrage de la fenêtre est automatiquement effectué afin d'éviter un défilement incessant. Le lecteur pourra comparer les différents tableaux avec le *modus operandi* détaillé dans la section 2.3.

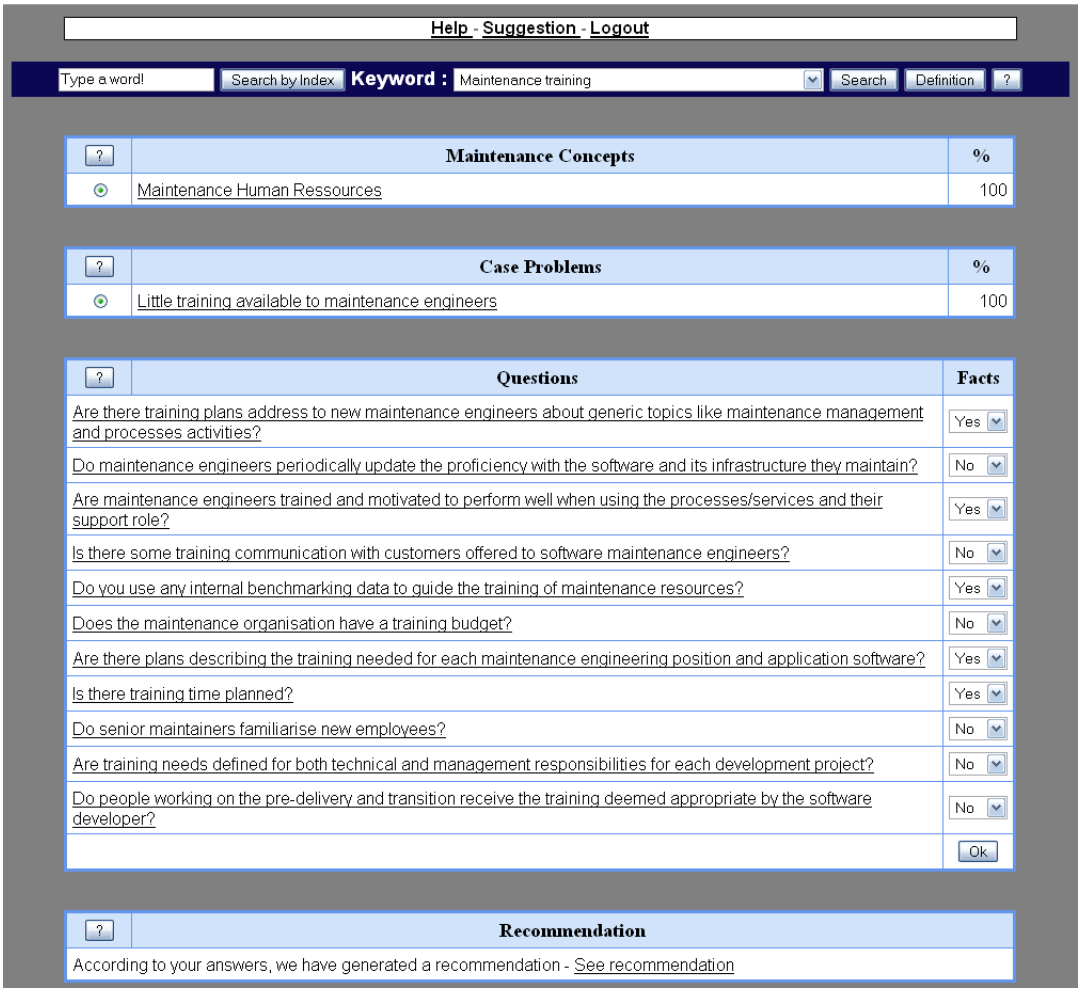


FIG. 8.3 – Section utilisateur du système $S3^m$DSS

8.4 Section administration

La figure 8.4 présente la section administrateur du système. Elle est composée d'un menu de sélection permettant d'effectuer les différentes actions de gestion de compte. Lorsque l'administrateur cliquera sur l'une de ces fonctionnalités, un formulaire lui sera proposé afin qu'il puisse remplir différentes informations dans le système. Dans l'exemple ci-dessous, la fonctionnalité d'ajout d'utilisateur est présentée. Notons que la mise en place de listes déroulantes a été systématiquement proposée lorsqu'elle était possible et ce afin d'éviter les erreurs de saisie.

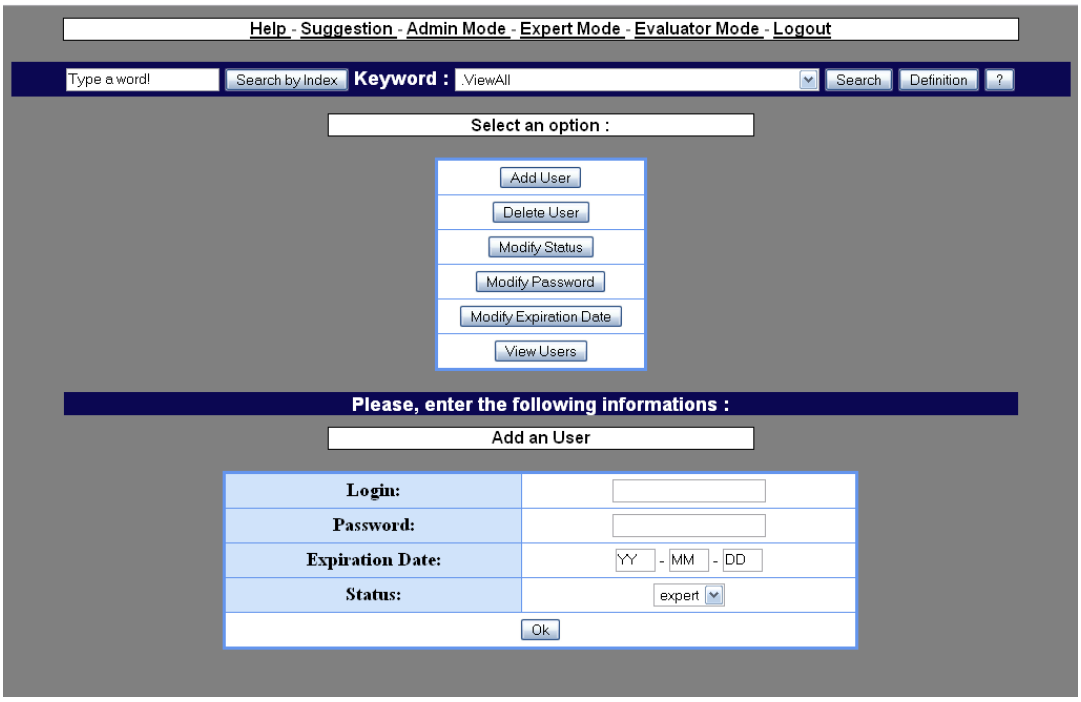


FIG. 8.4 – Section administrateur du système $S3^m$DSS

8.5 Section expert

La figure 8.5 présente la section expert du système. Elle permet de faire évoluer la base de connaissance. Un premier menu invite l'expert à choisir un élément et une action a entreprendre avec celui-ci. Les actions sont l'*ajout*, la *modification* ou le *retrait* tandis que les éléments sont les concepts typiquement que nous avons largement précisés dans les pages précédentes. A la sélection d'un élément, un formulaire est proposé à l'expert afin qu'il complète l'élément. Notons qu'une aide est disponible afin de profiter pleinement de toutes les fonctionnalités proposées dans ces formulaires. Un système de glissement inter-listes a également été mis au point afin de faciliter les manipulations de l'expert tout en diminuant les erreurs de saisie.

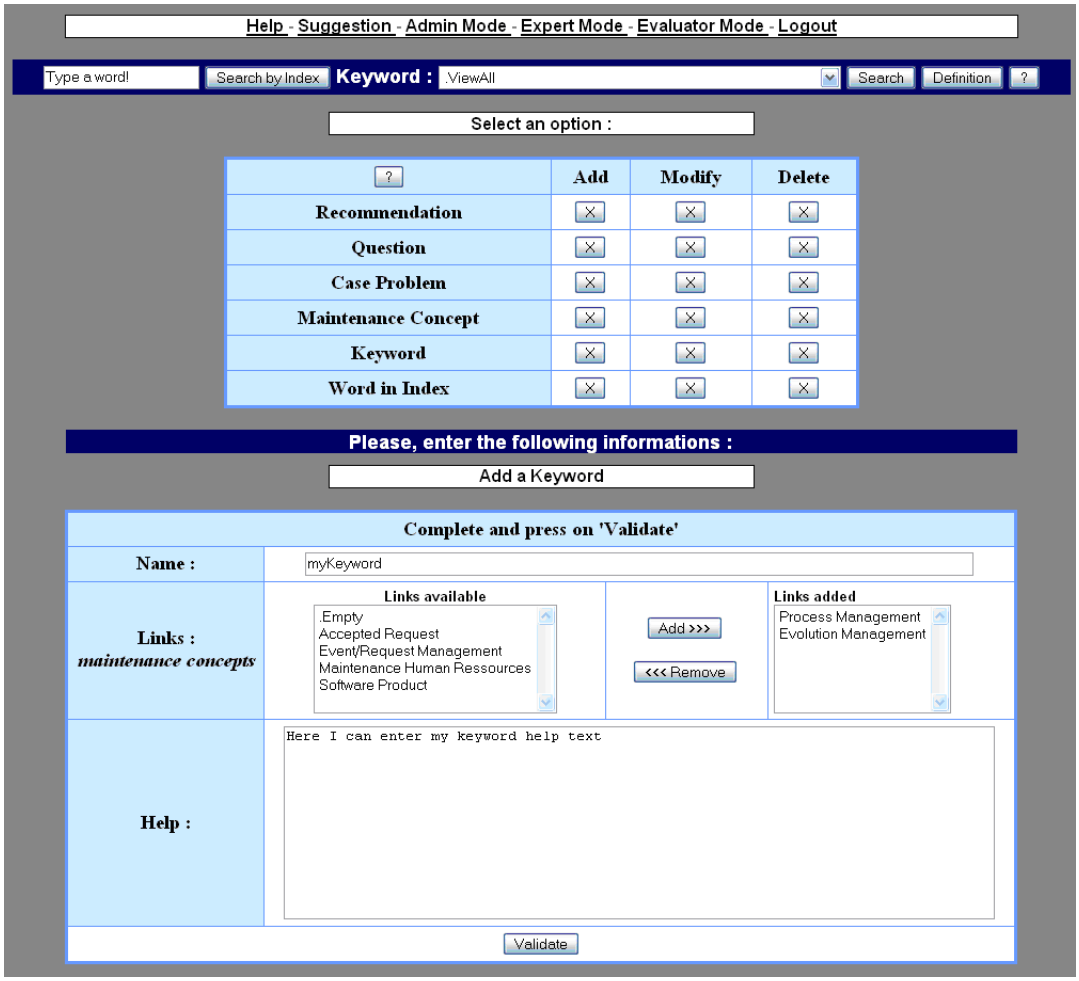


FIG. 8.5 – Section expert du système $S3^m$DSS

9 Conclusion

Lors de cette deuxième partie, nous avons présenté le système $S3^m$DSS depuis la description des attentes des commanditaires jusqu'aux technologies mises en place pour réaliser l'application. Nous avons décrit les origines multiples du logiciel venant à la fois des systèmes experts COSMICXpert et SMXpert. Nous avons parcouru les grandes fonctionnalités proposées aux utilisateurs. Une description de ces mêmes utilisateurs a été entreprise afin de distinguer les possibilités de chacun dans l'application. Nous en avons déduit des cas d'utilisation typiques ainsi qu'un certain nombre d'exigences non fonctionnelles. L'architecture 3-Tiers a également été présentée, se voulant assez modulaire pour les améliorations futures du système. Différents diagrammes commentés ont présenté l'application sous différentes perspectives. Une justification des technologies employées dans l'application a également été présentée, de même qu'un recueil des tests effectués ainsi qu'une présentation rapide des interfaces du système. Il reste à présent à décrire le coeur du système, c'est à dire, le contenu de sa base de connaissance. Ceci sera le sujet de la dernière partie de ce mémoire.

Troisième partie

Théorie des cas problèmes

1 Introduction

Cette dernière grande partie du mémoire consiste en la présentation du contenu du système $S3^m$DSS. Comme tout système d'aide à la décision, deux volets sont à distinguer. Une partie technique comportant les différents mécanismes mis en place pour le bon déroulement de l'application et une partie orientée contenu, délivrant un certain nombre d'indications aux utilisateurs du système. Cette troisième partie de mémoire présente avant tout la partie contenu de l'application.

Nous définirons, en *chapitre 2* , le raisonnement typique des experts du domaine de la maintenance parallèlement au raisonnement proposé par le système $S3^m$DSS sous forme d'arbres de décision. Cette démarche nous garantira la bonne heuristique développée par l'application. Nous préciserons ensuite, dans le *chapitre 3*, les différents problèmes de la maintenance en rappelant les problématiques théoriques présentées en première partie puis en se basant sur une approche empirique menant à des questionnements très pratiques. Finalement, nous proposerons, en *chapitre 4*, différents cas problèmes allant de l'énoncé d'une problématique réelle vers une piste de solution à l'instar de ce que propose le système $S3^m$DSS.

2 Analyse du raisonnement

Le système $S3^m$DSS, comme nous l'avons expliqué, propose un certain nombre de recommandations aux utilisateurs. Pour aboutir à ces bonnes pratiques, un raisonnement implicite est proposé sous forme d'un arbre de décision. Ce chapitre explique, dans un premier temps, le raisonnement d'un expert en maintenance de la soumission d'un problème par le client à sa résolution. Dans un deuxième temps, nous décrirons le raisonnement que le système $S3^m$DSS propose et analyserons l'arbre de décision engendré.

2.1 Raisonnement de l'expert

Le rôle de l'expert en maintenance comporte de multiples facettes. L'une d'entre elles consiste en un rôle de consultance pour l'usager désirant résoudre un problème particulier. Ces problèmes peuvent être de différentes natures mais ils doivent en tout cas être liés au domaine de la maintenance. Le guide SWEBOK [2] distingue un certain nombre de ces problèmes théoriques, Dekleva [7] présente les problèmes les plus couramment rencontrés par les usagers. Cette section présente la démarche théorique générale d'un expert en maintenance lorsqu'il fait face à une requête d'un client, qu'il fasse partie du personnel technique de la maintenance ou de la clientèle de la maintenance.

Lorsqu'un client (utilisateur ou mainteneur) présente un problème, l'expert en maintenance tente d'y faire correspondre un domaine de processus parmi les domaines présentés en partie 1 et étant : *la gestion du processus de la maintenance du logiciel*, *la gestion des requêtes de la maintenance du logiciel*, *l'ingénierie d'évolution* et *le support à l'ingénierie d'évolution*. Une description des multiples avantages à maîtriser ce domaine particulier peut également être proposée, chaque domaine possèdant une liste de bénéfices associés. Cette première phase est en réalité une étape de ciblage qui consiste à faire correspondre un problème avec un sous-ensemble de pratiques exemplaires agrégées en domaine.

Par la suite une série de questions sont posées au client. Ces questions vont permettre d'évaluer le niveau de maturité actuel du client dans le domaine associé à son problème. L'expert propose ensuite au client d'implanter les pratiques du niveau supérieur proposées par le modèle $S3^m$. S'il s'agit de la première expérience du client avec le modèle, l'expert pourra directement puiser les références adéquates dans le modèle et les proposer au client. Cette deuxième phase est en réalité une étape de proposition de recommandations afin de répondre au problème spécifique que le client a soumis à l'expert en maintenance.

Il s'agit précisément de la démarche que nous désirons instaurer dans le logiciel $S3^m$DSS.

2.2 Raisonnement proposé : *Arbre de décision*

La démarche proposée par le système $S3^m$DSS s'articule autour d'un arbre de décision. On considère qu'un arbre de décision est une représentation graphique de toutes les résultats et chemins possibles aboutissant à la résolution d'un problème. On parlera également de procédure de classification. Un tel arbre est composé de noeuds décisionnels auxquels sont associés un certain nombre de liens vers des choix. En progressant du noeud initial (racine) vers un noeud terminal (feuille), un certain nombre de choix seront émis et définiront une solution au problème de base.

Les noeuds considérés dans le système sont les concepts principaux du logiciel, à savoir : index, mots-clés, concepts de maintenance, cas problèmes, thèmes et recommandations. Cet arbre peut être peut-être scindé en deux grandes parties : la *sélection d'un cas problème* et la *déduction de recommandations*. La première étape consiste à sélectionner le cas problème présent dans le système et similaire au problème de départ de l'utilisateur. La seconde étape consiste à déduire les recommandations qui apporteront des zones de solution à l'utilisateur.

L'utilisateur du système doit, dans un premier temps, spécifier correctement le problème qui l'occupe. Pour se faire, nous avons mis en place un certain nombre de mécanismes permettant de cibler l'utilisateur vers un sous-ensemble de cas problèmes utiles. Cette première démarche s'effectue en trois temps. Premièrement l'utilisateur entre un mot dans un dictionnaire. Celui-ci est analysé et un certain nombre de mots-clés associés sont affichés par le système. Deuxièmement, en se basant sur les pourcentages d'association fournis par le système et la nature du problème qu'il désire identifier, l'utilisateur choisit un mot-clé générique permettant de se rapprocher du contexte de son problème. Troisièmement, par la sélection du mot-clé, une série de concepts de maintenance sont proposés, une démarche semblable est proposée. Le lecteur comprendra par la que ces mécanismes permettent d'orienter l'utilisateur vers des concepts génériques renvoyant eux mêmes à des problèmes bien particulier. Cette technique permet de partir d'un mot très spécifique, l'index, pour aboutir par des inférences successives à des concepts génériques, confortant par la même occasion les choix réalisés. L'utilisateur pourra dès lors visualiser si le sens de sa recherche a bien été compris par le système. Les figures 2.1, 2.2 et 2.3 proposent une vue schématique et partielle de l'arbre de décision relié à cette démarche.

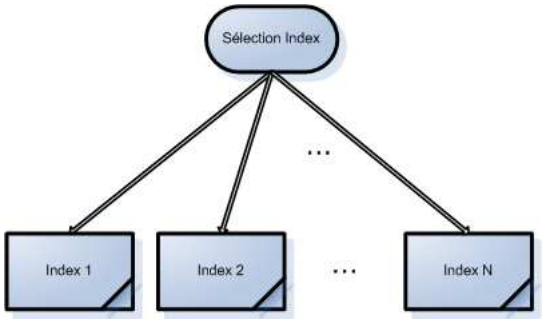


FIG. 2.1 – Arbre de décision orienté Index

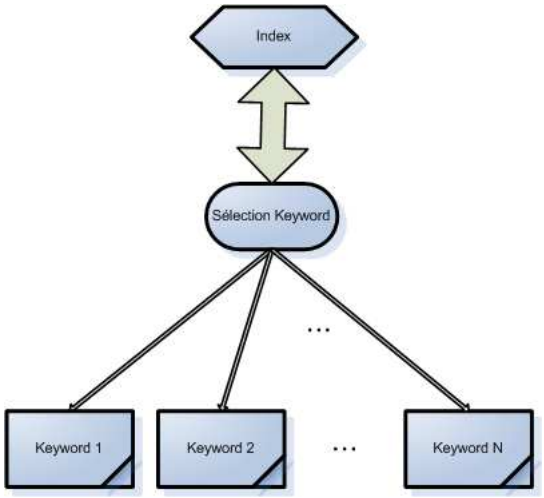


FIG. 2.2 – Arbre de décision orienté Keyword

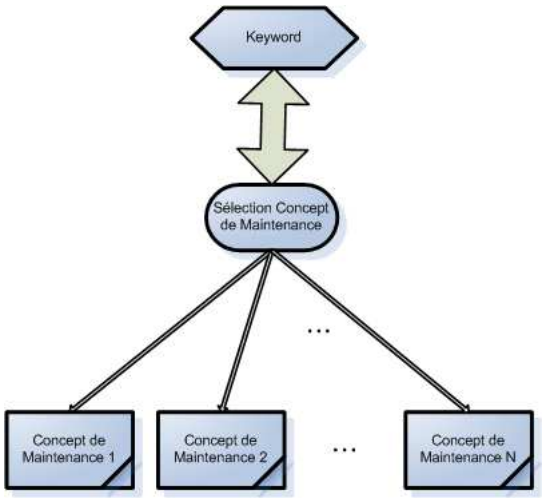


FIG. 2.3 – Arbre de décision orienté Concept de Maintenance

A ce stade, le système propose à l'utilisateur un certain nombre de cas problèmes dans lesquels il devrait retrouver le problème qui l'occupe. La deuxième phase de déduction de recommandations peut débuter. L'utilisateur répond à un certain nombre de questions permettant au système de lui proposer les bonnes pratiques pouvant solutionner son problème de départ. Notons que l'ordonancement des questions est réalisé de telle manière que les premières questions portent sur des pratiques de base pour arriver à des pratiques plus poussées dans les niveaux de maturité.

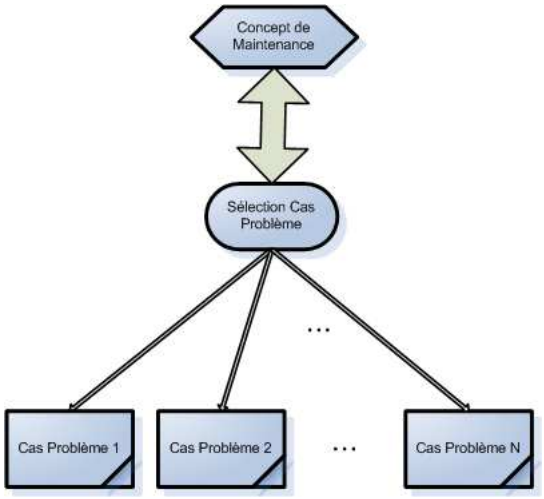


FIG. 2.4 – Arbre de décision orienté Cas Problème

Le système procède ensuite de la manière suivante : selon les réponses de l'utilisateur, il décide d'afficher ou non les recommandations associées à chaque question puis compile l'ensemble des recommandations implicitement demandées pour construire une feuille de synthèse qui sera proposée à l'utilisateur. La figures 2.5 et 2.6 propose une vue schématique et partielle de l'arbre de décision relié à cette démarche.

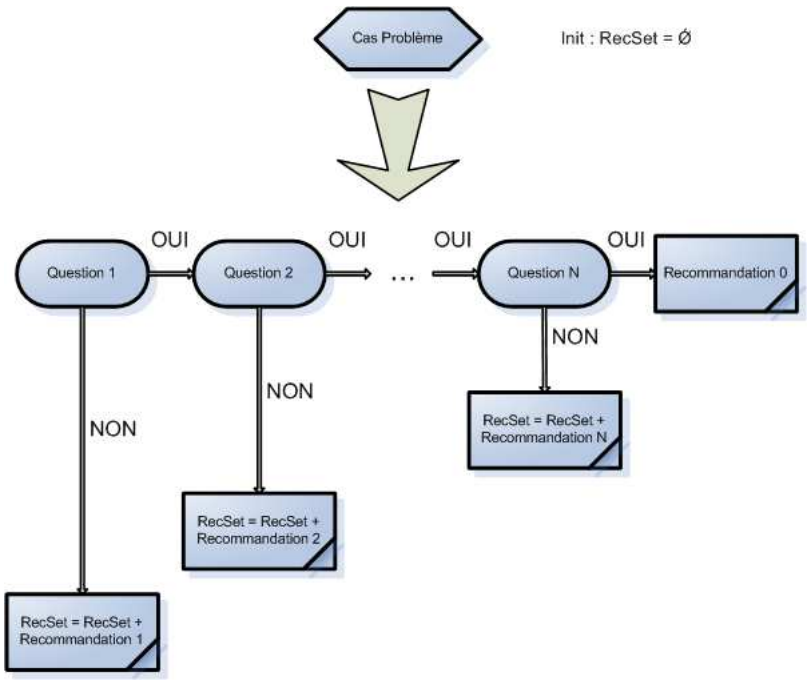


FIG. 2.5 – Arbre de décision orienté Theme

Notons que chaque recommandation finale propose un énoncé ainsi qu'une explication plus détaillée de la nouvelle démarche à adopter. Un lien a également été ajouté vers l'ouvrage de référence de la méthode $S3^m$ afin de permettre à l'utilisateur d'approfondir sa connaissance. De plus, différents hyperliens vers des pratiques proposées dans la littérature sont également proposés afin d'approfondir la question.

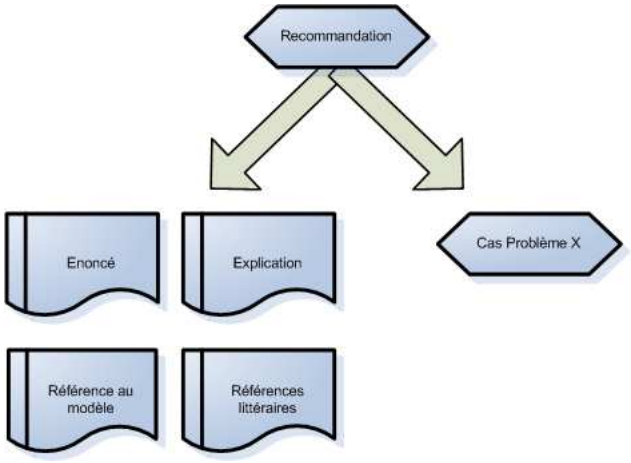


FIG. 2.6 – Arbre de décision orienté Recommandation

Finalement, remarquons la possibilité pour l'utilisateur d'effectuer un retour en arrière parmi les concepts. Cette possibilité est indispensable compte tenu du fait que l'utilisateur peut, selon ses choix, aboutir à des concepts non désirés. Il sera donc aisé pour lui de revenir une étape en arrière afin de corriger d'éventuelles erreurs de direction. Notons également qu'une recommandation finale peut, dans certains cas, renvoyer à un sous cas problème lorsque le système ne peut dériver une bonne pratique directement.

2.3 Conclusion

Nous avons ici, dans un premier temps, brièvement décrit la démarche d'un expert en maintenance lors d'un cas de consultance. Nous avons mis en évidence deux phases dans cette démarche : une phase de ciblage du domaine se rapportant au problème suggéré et une phase de questionnement permettant d'élaborer un certain nombre de recommandations. Dans un deuxième temps, nous avons analysé la démarche proposée par le logiciel $S3^m$DSS. Au travers d'un arbre de décision, nous avons également mis en évidence ces deux mêmes phases. Cette similitude valide le bien fondé de l'approche que nous proposons. La suite de cette partie proposera une description technique plus détaillée de l'analyse initiale, de la conception et de l'implémentation du système.

3 Problèmes de la maintenance

3.1 Approche théorique

En partie 1, nous avons cerné les problématiques principales de la maintenance. Pour rappel, le guide SWEBOK [2] distingue plusieurs grandes catégories de problèmes :

- Problèmes techniques ;
- Problèmes de management ;
- Problème d'estimation des coûts de la maintenance ;
- Problème de mesure de la maintenance du logiciel.

Au sein de ces catégories, sans rentrer dans une répétition des concepts, nous pouvons retrouver différentes problématiques : *compréhension limitée*, *essai*, *analyse d'impact*, *maintenabilité*, *alignement avec les objectifs de l'organisation*, *embauche du personnel*, *processus*, *aspects organisationnels de la maintenance*, *externalisation*, *estimation des coûts*, *modèles paramètrés*, *expérience* ou encore *mesures spécifiques*.

Ces problématiques sont pour l'instant assez théoriques, dans le sens où il s'agit avant tout d'une collection de problèmes rencontrés dans la littérature mais aussi par ce qu'elles sont relativement génériques et ne reflètent pas directement les problèmes rencontrés sur le terrain par la majorité des entreprises de maintenance. Dans cette optique de rapprochement vers les problèmes typiques de l'utilisateur, Dekleva [7] a réalisé une série de sondages au cours de conférences regroupant un certain nombre d'entreprises.

3.2 Approche empirique

L'approche proposée par Dekleva [7] consiste à identifier et quantifier les problèmes principaux en maintenance du logiciel. Une explication est aussi donnée concernant l'importance de ces problèmes. Cette approche est intéressante à la fois pour les chercheurs mais aussi pour les entrepreneurs. Les premiers peuvent ainsi axer leurs recherches dans les domaines les plus cruciaux tandis que les seconds peuvent comparer leur environnement et leur expérience avec leurs semblables. Dans notre cas, cette étude nous permettra d'obtenir un ensemble de problèmes typiques de la maintenance et de tenter des les résoudre via notre système d'aide à la décision. Le caractère très pratique de ces problèmes rend également l'usage du système d'autant plus intéressant qu'il permet d'y trouver des pistes

de solution très facilement.

L'étude a débuté en Avril 1990 lors d'une conférence annuelle de la SMA. Un ensemble de 68 personnes, toutes dans le domaine du génie logiciel depuis une quinzaine d'années ont été interrogées sur leurs problèmes en terme de maintenance. L'étude se base sur la méthode Delphi et consiste en un sondage en plusieurs tours aboutissant à un consensus sur les problèmes majeurs de la maintenance, chaque tour permettant d'améliorer la construction du consensus précédemment proposé. La figure 3.1 présente les problèmes identifiés et classés par les participants de cette étude :

Rank	Maintenance problem
1	Managing changing priorities
2	Inadequate testing techniques
3	Difficulty in measuring performance
4	Absent or incomplete software documentation
5	Adapting to rapid changes in user organizations
6	A large backlog of requests for change
7	Difficulty in measuring/demonstrating the maintenance team's contribution
8	Low morale due to lack of recognition and respect for maintenance engineer
9	Not many professionals in the domain, especially experienced ones
10	Little methodology, few standards, procedures and tools specific to maintenance
11	Source code in existing software complex and unstructured
12	Integration, overlap and incompatibility of existing systems
13	Little training available to maintenance engineers
14	No strategic plans for maintenance
15	Difficulty in understanding and meeting user expectations
16	Lack of understanding and support from IS/IT managers
17	Maintenance software runs on obsolete systems and technologies
18	Little will or support for reengineering existing software
19	Loss of expertise when a maintenance engineer leaves the team or company

FIG. 3.1 – Classification des problèmes de la maintenance, adapté de [7]

Parmi ces problèmes, il peut être intéressant d'en préciser quelques-uns. Compte tenu du classement effectué par Dekleva, nous nous attacherons à décrire en quelques lignes le contexte des 6 problèmes décelés comme étant les plus importants aux yeux des mainteneurs.

3.2.1 Priorités changeantes

En raison de la longueur des tâches de maintenance, de nouvelles demandes arrivent fréquemment avant que les tâches en cours soient terminées. Il y a toujours quelque chose de plus critique ou un autre utilisateur avec un problème. Une grande partie du temps est gaspillée à l'arrêt et au redémarrage d'une tâche de maintenance. Les mainteneurs considère ce problème comme une bataille constante. Certains mettent également en cause la longueur du cycle de changement (adapté de [7]).

3.2.2 Méthodes de test inadéquates

Il existe un manque de compréhension et d'utilisation de méthodes de test, un manque de temps, un manque de compréhension des données relatives aux tests de régression, et un manque d'exigence de tests rigoureux comme standard nécessaire au passage d'une application en phase de production. On notera que des environnements de tests innapropriés peuvent conduire à ces problèmes. Certains expliquent que les tests ont un coût trop élevé, d'autres argumentent que les méthodes de test sont trop contraignante à réaliser lors de petits changement dans un système (adapté de [7]).

3.2.3 Difficultés dans la mesure de performance

Il est difficile de mesurer la performance individuel et d'un groupe de mainteneurs. Cette difficulté peut provenir d'une mauvaise définition des tâches de la maintenance, condition nécessaire à la constitution de mesures. On posera également que mesurer objectivement la performance humaine est toujours très difficile à réaliser et par voie de conséquence à interpréter. On notera également le manque d'outils permettant de mesurer la performance du personnel (adapté de [7]).

3.2.4 Documentation du système incomplète ou non existante

Un manque de documentation précise fait baisser la productivité de la maintenance et augmenter drastiquement la courbe d'apprentissage pour les nouveaux mainteneurs. Les personnes bien informées sont coincées dans leurs travaux en raison de la documentation. Cette difficulté est souvent rencontrée dans le cadre de vieux systèmes dont la documentation associée est devenue inappropriée. Un manque de standard dans la documentation peut aussi conduire à une incomprehension (adapté de [7]).

3.2.5 Adaptation à un environnement d'affaires changeant rapidement

L'environnement d'affaires change à un rythme très rapide. Un système est déjà obsolète lorsqu'il est implanté. L'adaptation est particulièrement difficile dans le cas de vieux systèmes en raison d'un mauvais code, d'une complexité élevée, ou d'une vieille technologie. Ce problème est assez dépendant de l'industrie et de la zone fonctionnelle supportée par le système. On constate généralement que dans le domaine des assurances, le changement est fréquent alors que dans le monde de la finance, une certaine stabilité est de mise. On notera aussi que le changement de matériel peut parfois entraîner plus de problème qu'un modèle d'affaires variant. Notons finalement les réactions diverses au changement de la part des utilisateurs, tantôt ouverts, tantôt fermés (adapté de [7]).

3.2.6 Grand arriéré

Les utilisateurs sont non satisfaits et impatients. Il y a tellement de choses à faire qu'un petit nombre de requêtes ne sont jamais adressées. Cet arriéré peut être dut à des changements internes provoquant un allongement des délais. On parlera aussi d'un

manque de support managériale à cet égard mais aussi d'exigences changeantes de la part de l'utilisateur (adapté de [7]).

3.3 Classification des problèmes

La figure 3.2 montre la classification des problèmes présentés précédemment en quatre catégories de problèmes. Ces catégories permettent d'introduire des liens de causalité entre les différentes problématiques.

Classification des problèmes en catégories
Maintenance Management
Lack of managerial understanding and support
Large backlog
Changing priorities
Lack of maintenance methodology, standards, procedures, and tools
Inadequate testing methods
Performance measurement difficulties
Organizational Environment
Strategic plans
Adopting to the rapidly changing business environment
Lack of support for reengineering
Contribution measurement difficulties
Personnel Factors
Low morale due to the lack of recognition and respect
High turnover causing a loss of expertise
Lack of maintenance personnel, particularly experienced maintainers
Maintainers' lack proper training
Understanding and responding to business needs
System Characteristics
Program code is complex and unstructured
System documentation is incomplete or nonexistent
Antiquated systems and technology
Integration of overlapping and incompatible systems or subsystems

FIG. 3.2 – Classification des problèmes en catégories, adapté de [7]

Différentes corrélations entre problèmes ont été calculées par Dekleva [7] afin de déterminer s'il existait des liens de causalité entre les problématiques présentées selon les catégories définies. Ce dernier a pu déduire que les problèmes de maintenance de l'environnement organisationnel cause des problèmes de management. D'autre part, la gestion de la maintenance peut provoquer certaines perturbations chez le personnel. Les problèmes associés aux caractéristiques du système perturbent généralement aussi ce même personnel. La figure 3.3 présente ces différentes conséquences :

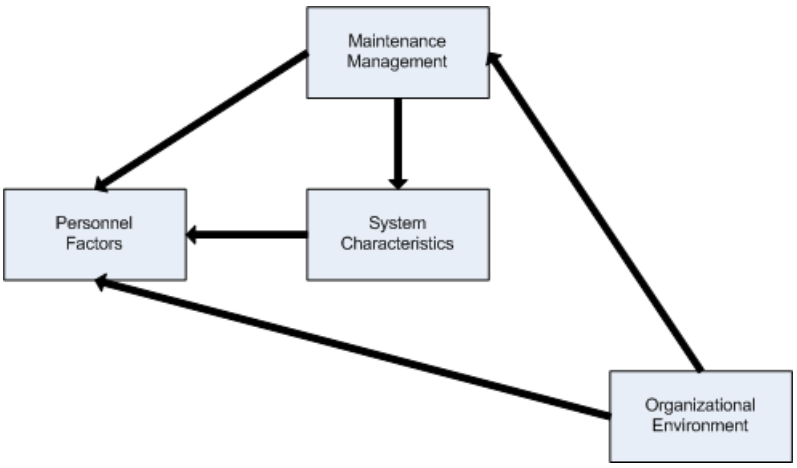


FIG. 3.3 – Associations causales, adapté de [7]

Finalement, notons que les problèmes liés au management sont les plus rencontrés. Ils représentent à eux seuls 50% des problèmes majeurs selon les répondants. Ceci est important pour au moins deux raisons. La première est que l'introduction de pratiques de gestion améliorées ne requière pas d'investissement substantiel et peut être construit graduellement. La seconde raison de la dominance de cette catégorie est qu'elle est fortement liée aux autres. On estime que chaque problème engendré par le management cause au moins un autre problème attenant à une seconde catégorie. Une résolution des problèmes de management pourrait donc non seulement résoudre ses difficultés propres mais aussi un certain nombre de problèmes indirectement causés.

4 Cas problèmes typiques

Ce chapitre présente quelques cas problèmes proposés par le système ainsi que les solutions apportées en terme de recommandation. Un replacement de la problématique est également effectué au sein des concepts de la maintenance. L'accent ici est principalement mis sur le contenu de ces problématiques. Le lecteur pourra retrouver l'ensemble de ces problèmes et d'autres non retranscrits dans le système $S3^m$DSS.

4.1 Peu de formations disponibles

Par ce cas problème, nous voulons décrire le fait que les entreprises de maintenance du logiciel ne mettent pas en place des activités de formation structurées. Souvent, la gestion des ressources humaines en terme de plans de formation, d'activités de formation et de gestion de carrière est mal établie. Les responsables ne sont pas au courant de l'importance d'aligner les activités de formation des mainteneurs avec les nouveaux projets. On constatera une absence de responsables en terme de formation. Le statut du mainteneur est par là aussi dévalorisé par rapport à celui d'un développeur. Un sentiment de délaissement de la part de l'entreprise est souvent ressenti.

La première étape consiste à localiser le domaine de processus et les KPAs associés à cette problématique. Il nous semble, à cet égard, judicieux de se concentrer sur le domaine de processus consacré à la gestion du processus de la maintenance du logiciel. Le KPA directement exploitable sera celui consacré à la formation en maintenance. Ce dernier tente de développer les compétences et le niveau des techniciens de la maintenance et de leurs responsables. On tentera d'identifier les besoins en formation, des plans de formation et le profil requis pour le personnel. L'engagement individuel du personnel dans leur propre éducation et formation est essentielle et doit être promotionnée. Les nouveaux employés doivent être bien formés et supervisés à mesure qu'ils s'intègrent dans les équipes de maintenance. Notons que la formation doit inclure une formation sur les processus de l'entreprise aussi bien qu'une formation sur les logiciels et technologies employées. April [3] regroupe synthétiquement les buts de ce KPA de la manière suivante :

- Identifier, demander et obtenir les ressources requises pour la formation et l'éducation des ingénieurs en maintenance ;
- Harmoniser la formation de l'entreprise, la formation locale planifiée et la formation par projet ;

– Assurer une force de travail compétente et motivée pour la maintenance ;
– Motiver les ingénieurs en maintenance par une promotion de l'éducation et une formation sur les processus, logiciels et technologies.

Notons que l'utilisateur du système $S3^m$DSS pourra accéder à ce problème par le biais du mot-clé : *Maintenance training*. Celui-ci le redirigera vers le concept de maintenance généraliste nommé *Maintenance Human Ressources*. A partir de ce point, un certain nombre de questions vont être posées à l'utilisateur afin de prendre connaissance des pratiques déjà en place dans l'entreprise et de proposer un feuillet de recommandations adaptées à la situation de l'utilisateur. Les questions que nous avons mis en place dans le système sont les suivantes :

1. Existe-t-il dans votre entreprise des plans de formation adressés aux nouveaux ingénieurs en maintenance concernant des sujets généraux comme l'activité de management ou des processus ?
2. Les ingénieurs en maintenance mettent-ils à jour périodiquement leurs compétences avec le logiciel qu'ils maintiennent et son infrastructure ?
3. Est-ce que les ingénieurs en maintenance sont formés et motivés à exécuter correctement leur travail en utilisant les processus, services et leur rôle de soutien ?
4. Existe-t-il une formation à la communication entre clients et ingénieurs en maintenance ?
5. Employez-vous des données de *benchmarking* internes pour guider la formation des ressources de maintenance ?
6. Est-ce votre entreprise possède un budget consacré à la formation ?
7. Y a-t-il des plans décrivant la formation requise pour chaque position dans l'entreprise et chaque logiciel d'application ?
8. Y a-t-il du temps de formation prévu ?
9. Les mainteneurs expérimentés familiarisent-ils de nouveaux employés ?
10. Est-ce que les besoins de formation sont définis pour les responsabilités techniques et de gestion lors de chaque projet de développement ?
11. Les gens travaillant à la pré-livraison et à la transition reçoivent-ils la formation considérée comme appropriée par le développeur du logiciel ?

En fonction des réponses fournies par l'utilisateur, une compilation de recommandations est proposée. Notons toute fois que si l'utilisateur satisfait à l'ensemble des propositions, aucune recommandation ne peut lui être proposée directement. En effet, le système travaille sur base des pratiques élémentaires des niveaux 0, 1 et 2 du modèle de maturité $S3^m$. Un message invitera donc l'utilisateur à contacter l'auteur du modèle afin de se procurer les recommandations des niveaux supérieurs moyennant payement. Remarquons que la majorité des entreprises ne satisfait pas à l'ensemble des bonnes pratiques des niveaux inférieurs. Ce cas de figure est dès lors limité à quelques précurseurs.

La suite de cette section présente les recommandations proposées à l'utilisateur. Le numéro de la recommandation correspond au numéro d'une des questions présentées ci-haut. Le lecteur comprendra que l'utilisateur répondant par la négative à la question x se verra proposé la recommandation x. Par recommandation, nous entendons généralement l'énoncé d'une pratique de base à réaliser au sein de l'entreprise de l'utilisateur, un texte explicatif apportant quelques détails supplémentaires à l'énoncé, des liens vers la littérature permettant d'approfondir le sujet si nécessaire, et un lien vers le numéro de la pratique dans le livre consacré au modèle $S3^m$.

Recommandation 1

Les plans de formation du nouveau personnel traitent de sujets généraux en gestion, processus et en maintenance du logiciel.

À ce niveau de maturité, le nouvel employé recevra une courte introduction au fonctionnement de l'unité organisationnelle. Cette formation inclura une revue du système de gestion des requêtes, des librairies des codes sources, de la bibliothèque de documentation, des règles de compilation et des accès aux données d'opération. Ceci implique que vous avez documenté les processus de maintenance du logiciel. Si ce n'est pas le cas, consultez la définition de processus.

Référence dans le modèle $S3^m$: Pro3.1.3

Liens annexes : [ISO12207 ; SEI02 ; ISO90003 :2004 ; Kaj01c-ME-1.1]

Recommandation 2

Le personnel de la maintenance se familiarise et se re-familiarise avec le logiciel qu'il maintient sur une base régulière.

Il est nécessaire que les ingénieurs logiciels assignés à la maintenance du logiciel possèdent une connaissance générale de la fonctionnalité du logiciel ainsi qu'une connaissance détaillée de ses composants. Le niveau de connaissance va influencer la productivité de l'employé. Des études récentes citent le manque de connaissances générales d'une application comme cause de problèmes. Les mainteneurs, de par la nature de leur travail, n'ont pas toujours la possibilité d'avoir la vue d'ensemble d'un logiciel complexe. Il faut donc se re-familiariser avec le logiciel régulièrement. Cela veut également dire de visiter la clientèle pour bien connaître l'utilisation de la fonctionnalité. Pour ce faire, on doit allouer et planifier des périodes pour ces activités. Le « People CMM » recommande cette activité en proposant d'améliorer la connaissance en architecture et dans les domaines spécifiques d'application.

Référence dans le modèle $S3^m$: Pro3.2.1

Liens annexes : [Kaj01c-ME-2.3]

Recommandation 3

Le personnel de la maintenance est formé et motivé afin de donner une bonne performance dans le processus et dans son rôle.

Cette pratique exemplaire tente de motiver les mainteneurs à bien connaître leurs processus et la manière d'effectuer le travail. On désire amener le personnel à faire des recommandations et à s'impliquer personnellement dans l'amélioration. On doit arriver à motiver l'individu à faire son travail d'une manière disciplinée. Chaque objectif derrière un processus doit être expliqué. Les managers et le personnel de la maintenance doivent faire un effort additionnel dans ces aspects descriptifs détaillés. Des études de Kajko-Mattson démontrent que même si le processus est bien documenté, il est rarement suivi tel que prescrit. Pour implanter cette pratique exemplaire, on doit :

- identifier les endroits où il y a manque de formation ;
- identifier les faiblesses des formations actuelles ;
- revoir les documents de formation de processus ;
- évaluer la connaissance des processus de la maintenance ;
- inviter le personnel à formuler des suggestions pour améliorer la formation en processus.

Référence dans le modèle $S3^m$: Pro3.2.2

Liens annexes : [ISO12207 ; SEI02 ; ISO90003 :2004 ; Kaj01c-ME-2.4]

Recommandation 4

De la formation en communication avec la clientèle est offerte au personnel de la maintenance du logiciel.

Le mainteneur effectue son travail directement avec la clientèle. Il est donc nécessaire d'offrir de la formation sur les communications orales ainsi que sur les communications interpersonnelles. Ce besoin varie en fonction du rôle du personnel.

Référence dans le modèle $S3^m$: Pro3.2.3

Liens annexes : [Kaj01c-ME-2.8]

Recommandation 5

Les comparaisons de formation des ressources du logiciel sous forme d'étalons internes pertinents sont utilisées afin de recueillir des données qui orientent les efforts de formation des ressources la maintenance.

L'étalonnage interne est une technique de comparaison pour fin d'amélioration. Des auteurs décrivent comment les organisations de la maintenance peuvent utiliser des comparaisons entre elles, afin de mieux comprendre les similitudes, les différences et ainsi

identifier les pratiques exemplaires internes à l'organisation. Pour rencontrer cette pratique exemplaire, les managers de la maintenance devront avoir effectué une comparaison des investissements en formation des autres spécialités du logiciel et des autres ressources de l'organisation. Cette comparaison doit être représentée sous forme graphique et servir aux discussions avec les managers et le groupe de formation sur les besoins de formation.

Référence dans le modèle $S3^m$: Pro3.2.4

Liens annexes : [S3m]

Recommandation 6

Les ressources financières sont disponibles au niveau de l'unité organisationnelle, pour l'éducation et la formation de chaque ressource.

L'unité organisationnelle de la maintenance du logiciel possède un budget pour la formation. Ce budget reflète les besoins individuels et sont alloués en fonction des besoins. Une documentation locale est disponible qui décrit :

- les cours complétés avec succès ;
- les activités de formation internes suivies par les employés ;
- les plans futurs de formation.

Référence dans le modèle $S3^m$: Pro3.2.5

Liens annexes :[Kaj01c-ME-2.1]

Recommandation 7

Il existe un plan d'éducation et de formation pour l'organisation et pour chaque logiciel. Ce plan énonce les besoins de formation, les cours offerts, les crédits, les ressources disponibles et le calendrier des activités.

Le plan d'éducation et de formation doit couvrir les perspectives suivantes : éthique et professionnalisme, la formation continue, les processus internes, les processus des groupes interfaces, l'infrastructure technique, le logiciel et ses règles d'affaires/structures de données.

Référence dans le modèle $S3^m$: Pro3.2.6

Liens annexes : [Nie02 6.7, activité 1, SEI TP activité 1]

Recommandation 8

Du temps pour la formation personnelle est planifié.

À ce niveau de maturité, on peut observer que le personnel prend environ deux heures/-semaine afin de se familiariser avec des pratiques exemplaires qui contribuent à l'efficacité de leur travail. Le personnel peut aussi assister à des présentations internes de collègues.

Référence dans le modèle $S3^m$: Pro3.2.7

Liens annexes : [Kaj01c-ME-2.2]

Recommandation 9

La personne senior fait l'introduction des nouveaux employés et de la relève pour la maintenance des logiciels applicatifs sous sa responsabilité.

L'introduction aux processus, activités, infrastructure et techniques de la maintenance spécifiques à un logiciel applicatif donné, doit être effectuée pour le nouvel employé. Cette activité traite les aspects suivants :

- Documentation et librairies du code source et objet ;
- Éditeurs, outil de comparaison de programmes et fichiers de données, compilateurs, assembleurs, liens, chargeurs des systèmes opérants, débogueurs, simulateurs, émulateurs, analyseurs statiques de code source, outils de test, outils de documentation et les systèmes de gestion des bases de données ;
- Horaire d'opération et de couverture du support opérationnel.

Référence dans le modèle $S3^m$: Pro3.2.8

Liens annexes : [S3m]

Recommandation 10

Les besoins de formation technique et en gestion sont précisés pour chaque projet en transition (c'est-à-dire nature de la formation, personnes visées, moment où la formation est offerte, etc.).

La transition est un des aspects les plus importants et des plus négligés du développement du logiciel. À ce niveau de maturité, la formation offerte par le projet sur sa transition et sa maintenabilité, est minimale (c'est-à-dire environ trois jours). Pour rencontrer cette pratique exemplaire, le mainteneur doit évaluer et réviser la proposition initiale de formation des mainteneurs. Il doit aussi établir une communication bidirectionnelle pour énoncer et clarifier ses exigences pendant le développement afin :

- d'effectuer une transition douce et
- de mieux assumer ses responsabilités pour la maintenance du nouveau logiciel.

Référence dans le modèle $S3^m$: Pro3.2.9

Liens annexes : [S3m]

Recommandation 11

Les personnes qui travaillent à la transition d'un projet reçoivent la formation jugée appropriée par le développeur.

Le développeur doit posséder un devis de ce qui est requis comme connaissances minimales requises, éducation et formation spécialisée pour maintenir le logiciel. Cette formation doit avoir été planifiée et suivie par le personnel de la maintenance qui aura la responsabilité de maintenir ce logiciel avant qu'ils en aient la responsabilité finale.

Référence dans le modèle $S3^m$: Pro3.2.10

Liens annexes : [S3m]

4.2 Gestion des priorités changeantes

Par ce cas problème, nous voulons décrire le fait que la durée des activités de maintenance est assez longues et que de nouvelles demandes émanant des clients viennent perturber ces activités avant la complétude des tâches. Il y a toujours quelque chose de plus critique à réaliser ou un autre problème du client à résoudre. On constatera donc qu'une grande partie du temps de maintenance est gaspillée dans les phases d'arrêt et de démarrage des activités de maintenance.

La première étape consiste à localiser le domaine de processus et les KPAs associés à cette problématique. Il nous semble, à cet égard, judicieux de se concentrer sur le domaine de processus consacré à la gestion des requêtes de la maintenance du logiciel. Les KPAs directement exploitables sont *la gestion des événements et requêtes* ainsi que *la planification de la maintenance*. On remarquera que ce cas problème trouve sa solution dans plusieurs KPAs de par sa complexité.

Le KPA de gestion des événements et requêtes assure que les erreurs, les requêtes de support opérationnel et l'évolution du logiciel sont identifiées pro activement, priorisées, re-dirigées et réalisées afin de rencontrer les exigences introduites dans le contrat de SLA connu du client. Un événement, s'il n'est pas résolu, peut résulter en une non conformité face aux objectifs du SLA et peut être la cause de mauvais temps de réponse à une requête spécifique du client ou il peut rentrer en conflit avec les exigences précédemment approuvées par les deux parties. April [3] regroupe synthétiquement les buts de ce KPA de la manière suivante :

- Faire la collecte proactive de toutes les requêtes de services (internes et de la clientèle) ;
- Bien identifier, communiquer et obtenir le consensus sur les priorités des requêtes en attente et en évolution ;
- S'assurer que l'on travaille sur les bonnes priorités ;
- Surveiller le comportement du logiciel dans les dernières 24 heures et les infrastructures ;

– Interrompre le travail en cours lorsqu'un événement de plus haute priorité survient ;
– Communiquer pro activement les événements et leurs temps planifiés de résolutions.

Le KPA de planification de la maintenance assure la création et la mise à jour de plans décrivant les activités courantes et à prévoir de la maintenance du logiciel. Le planning de maintenance est typiquement élaboré à partir de trois perspectives : organisationnelle, tactique et opérationnelle. Ces perspectives documentent les demandes en terme de service de maintenance. Les données de planification doivent être étudiées afin de produire un document justifiant les allocations de buget et de ressources en fonction des priorités et des demandes de service. April [3] regroupe synthétiquement les buts de ce KPA de la manière suivante :

– Faire la collecte proactive de toutes les requêtes (court terme, moyen terme et long terme) et en établir la planification détaillée ;
– Bien identifier, communiquer et obtenir le consensus sur les priorités des requêtes actuellement en attente et en évolution ;
– Identifier les contrôles requis pour chaque type de service et logiciel opérationnel de la maintenance du logiciel ;
– Les systèmes de reprises de pannes font l'objet d'essais, d'audits et de planification ;
– Il existe une planification détaillée du contenu des versions d'un logiciel ;
– Effectuer la mise à niveau des logiciels qui proviennent des fournisseurs de progiciels ou de l'introduction de nouveautés de l'infrastructure et des opérations ;
– Établir une justification d'allocation des nouvelles requêtes et de la gestion de la capacité ;
– Informer les intervenants des travaux exécutés, des travaux en attente et des travaux en évolution en fonction des priorités établies ;
– Informer les intervenants sur l'état des budgets et l'utilisation des ressources.

Notons que l'utilisateur du système $S3^m$DSS pourra accéder à ce problème par le biais des mots-clés : *Event and service request management*, *Event management*, *Capacity planning*, *Disaster recovery planning*, *Impact analysis*, *Maintenance planning*, *Monitoring and control of service requests and events*, *Reviews*, *Urgent changes* et *Versions and upgrade planning*. Ceux-ci le redirigeront vers le concept de maintenance généraliste nommé *Event/Request management*. A partir de ce point, un certain nombre de questions vont être posées à l'utilisateur afin de prendre connaissance des pratiques déjà en place dans l'entreprise et de proposer un feuillet de recommandations adaptées à la situation de l'utilisateur. Les questions que nous avons mis en place dans le système sont les suivantes :

1. Les clients/utilisateurs agissent-ils différemment selon la nature de leur contact avec l'organisation de maintenance ?
2. Chaque client/utilisateur ou système a-t-il un point de contact documenté qui fournit des services de maintenance ?
3. Chaque demande du client et événement du système de client sont-ils enregistrés par l'organisation de maintenance ?

4. Est-ce que chaque demande du client est d'abord admise ou rejetées avant d'être assignée ?
5. Est-ce que des demandes acceptées de modification sont assignées, d'une manière préliminaire, à une future version du logiciel ?
6. La maintenance du logiciel édite-t-elle un plan de maintenance ayant une vue de un à trois ans ?
7. Y a-t-il un plan de maintenance élaboré et mis à jour annuellement selon une procédure documentée ?
8. Les activités de planification de la maintenance suivent-elles les normes de l'organisation ?
9. La planification de la maintenance considère-t-elle les possibilités de rajeunissement de l'ensemble des logiciels ?
10. Le plan de maintenance considère-t-il les besoins en ressources (personnel, infrastructure et outils) ?
11. Le responsable de la maintenance assigne-t-il dynamiquement la charge de travail selon la demande ?
12. Est-ce que le plan de la maintenance est utilisé pour faire le suivi des activités et pour communiquer le statut des requêtes ?
13. La priorité des demandes d'un client est-elle assignée en collaboration étroite avec l'utilisateur, selon une procédure documenté ?
14. Est-ce que des rapports de problème sont redirigés au groupe approprié de support seulement lorsque le support courant a documenté son intervention ?
15. Y a-t-il un plan de support de capacité relatif aux demandes de support et de modification ?

En fonction des réponses fournies par l'utilisateur, une compilation de recommandations est proposée. Notons toute fois que si l'utilisateur satisfait à l'ensemble des propositions, aucune recommandation ne peut lui être proposée directement. En effet, le système travaille sur base des pratiques élémentaires des niveaux 0, 1 et 2 du modèle de maturité $S3^m$. Un message invitera donc l'utilisateur à contacter l'auteur du modèle afin de se procurer les recommandations des niveaux supérieurs moyennant payement. Remarquons que la majorité des entreprises ne satisfait pas à l'ensemble des bonnes pratiques des niveaux inférieurs. Ce cas de figure est dès lors limité à quelques précurseurs.

La suite de cette section présente les recommandations proposées à l'utilisateur. Le numéro de la recommandation correspond au numéro d'une des questions présentées ci-haut. Le lecteur comprendra que l'utilisateur répondant par la négative à la question x se verra proposé la recommandation x. Par recommandation, nous entendons généralement l'énoncé d'une pratique de base à réaliser au sein de l'entreprise de l'utilisateur, un texte explicatif apportant quelques détails supplémentaires à l'énoncé, des liens vers la littérature permettant d'approfondir le sujet si nécessaire, et un lien vers le numéro de la pratique dans le livre consacré au modèle $S3^m$.

Recommandation 1

Une approche individuelle de gestion des requêtes et des événements est principalement basée sur des relations personnelles entre le programmeur de la maintenance et une ressource de l'unité organisationnelle du client

Référence dans le modèle $S3^m$: Req1.1.2

Recommandation 2

Il y a un point de contact unique pour fournir une aide directe aux clients.

Référence dans le modèle $S3^m$: Req1.2.1

Recommandation 3

Chaque requête et événement font l'objet de création d'une requête de modifications (Rm) ou d'un rapport de problèmes (Rp) du logiciel qui sert de billet de travail de la maintenance.

Pour rencontrer cette pratique, on doit s'assurer que chaque requête et événement sont documentés. Par exemple, un numéro d'identification unique est assigné avec en plus un sommaire de la requête.

Référence dans le modèle $S3^m$: Req1.2.2

Liens annexes : [IEEE Std 1219, 4.1]

Recommandation 4

Chaque requête et événement font l'objet de catégorisation, prioritisation et d'une estimation préliminaire de leur taille et de leur ampleur ?

On classe les requêtes selon les catégories de services définies et normalisées. Le responsable du logiciel fait l'étude de compromis pouvant s'appliquer à une nouvelle requête de services. Expliquez qu'il y a beaucoup d'enjeux lors de l'évaluation d'une nouvelle requête. L'équipe de la maintenance procédera donc à une évaluation sur l'introduction d'un changement face à l'impact sur la stabilité et la disponibilité du logiciel. On doit aussi juger de ce qui est nécessaire, en termes de processus formels (c'est-à-dire des planifications, analyses d'impact, essais, etc.), versus l'urgence de la requête et du coût pouvant être facturé au client.

Référence dans le modèle $S3^m$: Req1.2.3

Liens annexes : [IEEE Std 1219, 4.1.2]

Recommandation 5

Les requêtes de modifications acceptées sont assignées, d'une manière préliminaire, à une version future du logiciel.

Assigner la requête à une version future du logiciel.

Référence dans le modèle $S3^m$: Req1.2.4

Liens annexes : [IEEE Std 1219, 4.1]

Recommandation 6

Au niveau de l'organisation, il existe une politique relative aux plans des unités organisationnelles. Ces plan comprennent l'objet, la portée, les buts, les objectifs, les biens livrables et certains autres éléments importants.

À ce niveau de maturité, on doit s'assurer de rencontrer les normes de la planification de l'organisation. L'unité organisationnelle de la maintenance devra donc, par exemple :

- Connaître le cycle de la planification de l'organisation ;
- Déposer ses planifications selon les formats et échéances requis ;
- S'assurer de la qualité du contenu ;
- Utiliser et mettre à jour les planifications pendant l'année ;
- Rendre compte des écarts.

Référence dans le modèle $S3^m$: Req2.2.1

Liens annexes : [SEI SPP engagement 2] [ISO 9001-3 5.4.1 et 5.4.2.2] [Cam94]

Recommandation 7

Le plan de la maintenance est élaboré et mis à jour annuellement, conformément à une procédure documentée.

Un processus de planification annuelle de la gestion du logiciel est institué. Le plan de la maintenance du logiciel couvre, entre autres, la maintenance planifiée et imprévue.

Référence dans le modèle $S3^m$: Req2.2.2

Liens annexes : [SEI SPP activité 6] [ISO 9001-3 5.4] (Réf. IEEE Std. 1058.1) [Zit96 MCA2.02/01]

Recommandation 8

Les activités de planification de l'unité organisationnelle de la maintenance suivent les normes de l'organisation et sont coordonnées avec le développeur et les infrastructures et opérations.

Pour rencontrer cette pratique exemplaire, on doit pouvoir voir l'harmonisation des plans de ces trois unités organisationnelles. Pour ce faire, on devra retrouver des activités internes conjointes de planification ou des communications intergroupes sur l'échange des planifications préliminaires.

Référence dans le modèle $S3^m$: Req2.2.3

Recommandation 9

La planification annuelle introduit les possibilités de rajeunissement pour les logiciels de la maintenance.

Le plan de la maintenance discute des logiciels qui pourraient profiter d'activités majeures de rajeunissement. Pour ce faire, on doit présenter une étude de cas expliquant quelle activité spécifique de rajeunissement pourrait profiter au logiciel. Cette étude explique sommairement les coûts et bénéfices et sera présentée à l'utilisateur pour approbation lors de sessions de gestion du compte et de signature d'ententes de services annuelles.

Référence dans le modèle $S3^m$: Req2.2.4

Recommandation 10

La planification des infrastructures et les outils de soutien d'ingénierie d'évolution du logiciel sont préparés annuellement.

Le plan de la maintenance possède, entre autres, un volet infrastructure et outils de soutien d'ingénierie.

Référence dans le modèle $S3^m$: Req2.2.5

Liens annexes : [SEI SPP activité 14] [Cam94]

Recommandation 11

Le responsable de la maintenance est désigné comme responsable des activités de planification sur les requêtes visant son logiciel opérationnel.

À ce niveau de maturité, le responsable de la maintenance dicte les approches de solutions et supervise le traitement des requêtes. Il est donc responsable, par exemple :

- d'entretenir le contact avec la clientèle ;
- d'influencer les priorités relatives en tenant compte de toutes les perspectives du logiciel opérationnel, par exemple :
  - les changements aux infrastructures et opérations ;
  - les projets actuellement en développement et en transition ;
  - les cycles annuels de production ;
  - les défaillances actuelles ;
  - les ententes contractuelles avec les fournisseurs ;
  - les autres priorités des dirigeants des Technologies de l'information.
- d'énoncer des analyses d'impact sommaires et de vérifier les conclusions de ses programmeurs ;
- de s'assurer que les autres ressources de la maintenance connaissent les priorités établies ;
- de s'assurer que les autres ressources de la maintenance travaillent aux priorités établies ;
- que les processus de la maintenance sont suivis.

Référence dans le modèle $S3^m$: Req2.2.17

Liens annexes : [SEI SPT& O engagement 1]

Recommandation 12

Le plan de la maintenance est utilisé pour faire le suivi des activités et pour communiquer l'état de situation des requêtes.

À ce niveau de maturité, le plan consiste principalement en une liste priorisée des requêtes de la maintenance avec un sommaire d'analyse pour chacune. IEEE1219 recommande que cette liste provienne du système de gestion des requêtes de la maintenance. Cette liste doit être gérée, contrôlée et communiquée pour démontrer l'état de la situation de la demande de services de la maintenance du logiciel. Cette liste doit être gérée conformément à une procédure établie.

Référence dans le modèle $S3^m$: Req2.2.18

Liens annexes : [SEI SPP activité 6] [SEI SPT& O activité 1] [ISO9003 :2004 5.4] [IEEE1219, 4.2]

Recommandation 13

L'assignation des priorités aux différentes requêtes s'effectue en étroite collaboration avec le client, conformément à une procédure documentée.

Pour rencontrer cette pratique, il faut que la liste des requêtes soit régulièrement approuvée par la clientèle. Cette pratique a plusieurs objectifs secondaires, par exemple :

- Le client est tenu informé des priorités relatives à ses requêtes ;
- La liste des requêtes circule et est maintenue à jour ;
- Le client contrôle les travaux ;
- On prévient les plaintes en ce qui a trait à la lenteur du service pour une requête spécifique ;
- Des discussions peuvent être entamées sur les moyens alternatifs d'accélérer le traitement des requêtes en attente.

Plusieurs organisations de la maintenance ont de la difficulté à évaluer et à prioriser les requêtes. Selon Kajko-Mattss, une manière de prioriser rapidement les rapports de problèmes est d'y assigner une valeur par défaut. On peut donc assigner le problème à une version ultérieure du logiciel et viser initialement (à l'aide de notre connaissance des temps de résolution moyens) une période de résolution. Déterminer à quelle version la requête peut être assignée initialement s'effectue en fonction de plusieurs critères décrits par Martin et McLure.

On peut aussi aider la prise de décision en élaborant des analyses financières de chaque requête. C'est-à-dire que les aspects financiers sont étudiés et pris en compte dans l'analyse d'impact. Ces analyses sont basées sur :

- un modèle des coûts ;
- un modèle des bénéfices ;
- des règles sur les méthodes de calcul devant être utilisées.

Le modèle des coûts tient compte des coûts initiaux (correction ou modification) et subséquents (maintenance et exploitation) des logiciels. Ces coûts doivent faire état de toutes les ressources nécessaires, qu'il s'agisse de ressources internes ou externes à l'organisation, humaines, financières, matérielles ou informationnelles.

Le modèle des bénéfices distingue les bénéfices selon qu'ils sont :

- mesurables en unités monétaires ;
- mesurables au moyen d'indicateurs de gestion ;
- non mesurables (intangibles).

Le modèle de bénéfice sépare les bénéfices financiers selon qu'ils sont récupérables ou non par l'organisation .

Référence dans le modèle $S3^m$: Req2.2.19

Liens annexes : [Cam94, 8.3.2.1] [Cam94, 8.3.2.4] [Cam94, 8.3.2.5]

Recommandation 14

Toutes les questions relatives au fonctionnement intergroupes sont documentées et négociées. Les problèmes non résolus sont acheminés aux gestionnaires concernés.

Une modification à un logiciel existant peut affecter plusieurs acteurs. Pour rencontrer cette pratique exemplaire, l'autorisation d'une requête spécifique inclut l'accord des groupes affectés (par exemple : groupes opérationnels, développeurs, bureau d'aide, infrastructures et opérations, etc.). La ressource de la maintenance ne peut pas, à ce niveau de maturité, procéder à un changement et ensuite placer les intervenants devant un fait accompli.

Référence dans le modèle $S3^m$: Req2.2.20

Liens annexes : [SEIIC activité 6]

Recommandation 15

L'unité organisationnelle de la maintenance met en oeuvre un processus de gestion de la capacité fondé sur l'ensemble des planifications et des ententes de services.

Un processus de gestion de la capacité est développé, prenant en compte les différentes planifications et les ententes de services de la clientèle. La gestion de la capacité est composée d'un certain nombre d'activités ayant pour objectif l'évaluation de la capacité de traiter la demande de la clientèle en fonction des ressources disponibles. Pour rencontrer cette pratique, le mainteneur doit démontrer que la demande n'excède pas la capacité des ressources de son unité organisationnelle en fonction des ententes de services. Pour en savoir plus sur la gestion de la capacité de services et la gestion de la capacité des ressources, consultez [ITI01 6.2.2 et 6.2.3].

Référence dans le modèle $S3^m$: Req2.2.21

Liens annexes : [ITI01 6.2]

5 Conclusion

Lors de cette troisième partie, nous avons précisé le raisonnement de l'expert en maintenance et l'avons comparé à celui proposé dans le système $S3^m$DSS en terme d'arbres de décision. Nous sommes aboutis à une démarche analogue, ceci garantissant la validité du système. Nous avons ensuite fait glisser les problématiques présentées dans le guide SWEBOK [2] vers des problèmes très concrets proposés par Dekleva [7]. Cela nous a permis d'élaborer une nouvelle classification entre problèmes mais aussi différents liens de causalité. Finalement, nous avons présenté une série de cas problèmes tirés de l'analyse précédente et directement implanté dans le système $S3^m$DSS. Cette dernière étape permet de prendre connaissance d'une partie du contenu de l'application.

Conclusions et travaux futurs

Blablabla c'est la conclusion

Bibliographie

[1] Abran A., Desharnais J.-M., Oligny S., St-Pierre D. et Symons C., *Measurement Manual 2.2*, UQAM, Canada, 2002.

[2] Abran A. et Moore J.W., *SWEBOK : Guide to Software Engineering Body of Knowledge*, Los Alamitos CA, 2004.

[3] April A., *S3^m - Model to Evaluate and Improve the Quality of Software Maintenance Process*, Montreal, Canada, 2005.

[4] Arthur L.J., *Software Evolution : The Software Maintenance Challenge*, John Wiley & Sons, 1988.

[5] Bennett S., *Software Maintenance : A Tutorial, Software Engineering*, IEEE Computer Society Press, 2000.

[6] Dallape R., *Re-ingénierie d'un système d'aide à la décision : Du raisonnement hybride au raisonnement par règle*, FUNDP, Belgique, 2005.

[7] Dekleva S.M., *Delphi study of software maintenance problems*, International Conference on Software Maintenance, IEEE Computer Society Press, Los Alamitos CA, 1992.

[8] Desharnais J.-M., *Application de la mesure fonctionnelle COSMIC-FFP : une approche cognitive*, UQAM, Canada, 2003.

[9] Federal Information Processing Standards Publication (FIPS PUB 106), *Guideline on Software Maintenance*, 1984.

[10] Fichefet J., *Recherche Opérationnelle : Notes de cours*, FUNDP, 2006.

[11] *IEEE std 1219 : Standard for Software Maintenance*, IEEE Computer Society Press, Los Alamitos, CA.

[12] Institude of Electrical and Electronics Engineers, *Norme 829*, http ://www.ieee.org.

[13] *ISO 12207 : Information Technology - Software Life Cycle Processes*, Geneva Switzerland.

[14] Kitchenham B. A. et Al., *Towards an Ontology of Software Maintenance*, Journal of Software Maintenance : Research and Practise, 1999.

[15] Lehman M.M., *On understanding laws, evolution, and conversation in the large-program life cycle*, The journal of Systems and Software, 1980.

[16] Martin J. et McClure C., *Software Maintenance : The Problem and its Solutions*, Englewood Cliffs, NJ, 1983.

[17] Nelson et Al., *An Evolutionary Theory of Economic Change*, Belknap Press, Cambridge, Massachusset and London, 1982.

[18] Osborne W.M. et Chikofsky E.J., *Fitting Pieces to the Maintenance Puzzle*, IEEE Software 10-11, 1990.

[19] Pigoski T.M., *Encyclopedia of Software Engineering*, John Wiley & Sons, New York, 1994.

[20] Pigoski T.M., *Practical Software Maintenance - Best practices for managing your software investment*, Wiley Computer Publishing, 1996.

[21] Roy B., *Cahiers du Lamsade n°97*, Univ. Paris-Dauphine, 1992.

[22] Sandron S., Vanderose B., *Cadre de référence pour l'amélioration de la qualité d'un logiciel à base de connaissances*, FUNDP, Belgique, 2004.

[23] Schneidewind N.F., *The state of software maintenance*, IEEE Transactions on Software Engineering, 1987.

[24] Schneidewind N.F., *Software Maintenance : The Need for Standardization. Processing of the IEEE*, 1989.

[25] Swanson E.B., *The Dimensions of Maintenance - Proceedings of the Second International Conference on Software Engineering*, 1976.

[26] Tahe S., *Introduction à la programmation WEB en JAVA - Servlets et pages JSP*, Université d'Angers, France, 2002.

[27] Van Heijst G., Schreiber A.T., Wielinga A., *Using explicit ontologies in KBS Development*, University of Amsterdam, Department of Social Science Informatics, 2003.

[28] Sheard S., *The Framework Quagmire*, Software Productivity Consortium, 2001.

[29] Von Mayrhausser T.E., *Software Engineering - Methods and Management*, Academic Press, Inc., San Diego CA, 1990.

[30] Wang Y. et King G., *Software Engineering Processes - Principles and Applications*, CRC Press, 2000.

[31] Wikipédia : L'encyclopédie libre, article consacré à Apache Tomcat.

[32] Wikipédia : L'encyclopédie libre, article consacré aux Cascading Style Sheets.

[33] Wikipédia : L'encyclopédie libre, article consacré à l'Extensible Markup Language.

[34] Wikipédia : L'encyclopédie libre, article consacré aux Java Server Pages.

[35] Wikipédia : L'encyclopédie libre, article consacré au JavaScript.

[36] Wikipédia : L'encyclopédie libre, article consacré à l'Hypertext Markup Language.

[37] Wikipédia : L'encyclopédie libre, article consacré à Microsoft SQL Server.

[38] Wikipédia : L'encyclopédie libre, article consacré aux Tests d'intégration.

[39] Wikipédia : L'encyclopédie libre, article consacré aux Tests unitaires.

ANNEXE 1 : Scénarios d'utilisation

Cette annexe présente une explication plus détaillée des scénarios d'utilisation du système par les différentes catégories d'utilisateur.

Pré-conditions globales :
- Le système fonctionne correctement et est prêt à accepter une requête.

Post-conditions globales :
- Le système fonctionne correctement et est prêt à accepter une requête.

Scénarios - Vue *Utilisateur*

Authentification

Description : Un utilisateur souhaite s'authentifier auprès du système.

Pré-conditions :
– L'utilisateur n'est pas encore authentifié.
– L'utilisateur dispose des informations d'authentification.

Post-conditions :
– L'utilisateur est authentifié par le système.

Flux normal :

Utilisateur	Système
1. L'utilisateur manifeste son intention de s'authentifier	
	2. Le système demande les informations d'authentification
3. L'utilisateur fournit ces informations	

	4. Le système vérifie que les données introduites sont correctes 5. Le système signale à l'utilisateur qu'il est authentifié

Informations erronées "extends" Authentification

Description : Un utilisateur souhaite s'authentifier auprès du système mais il ne dispose pas de données valides.

Pré-conditions :
– Les informations d'authentification sont invalides.

Post-conditions :
– L'utilisateur n'est pas authentifié par le système.
– L'utilisateur n'a pas accès aux informations du système.

Flux alternatif :

Utilisateur	Système
Idem Use Case Authentification - Authentification jusqu'à l'action 4	
	5. Le système signale à l'utilisateur que les données introduites sont inexistantes ou non valides.

Désauthentification

Description : Un utilisateur souhaite se désauthentifier auprès du système.

Pré-conditions :
– L'utilisateur est authentifié.

Post-conditions :
– L'utilisateur est désauthentifié par le système.

Flux normal :

Utilisateur	Système

1. L'utilisateur manifeste son intention de se désauthentifier	
	2. Le système désauthentifie l'utilisateur 3. Le système signale à l'utilisateur qu'il est désauthentifié

Consultation de la base de connaissance

Description : Un utilisateur souhaite consulter la base de connaissance du système.

Pré-conditions :
- L'utilisateur est authentifié.
- L'utilisateur a une idée fixe sur ce qu'il recherche.

Post-conditions :
- L'utilisateur a obtenu l'information qu'il souhaitait connaître.

Flux normal :

Utilisateur	**Système**
1. L'utilisateur manifeste son intention de se consulter la base de connaissance	
	2. Le système propose d'entrer un mot clé ou de chercher un mot clé par index
3. L'utilisateur entre un mot clé ou cherche un mot clé par index	
	4. Le système propose un ensemble de concepts de maintenance relatifs au mot clé choisi ainsi qu'un pourcentage symbolisant l'exactitude du lien entre le mot clé et le concept de maintenance
5. L'utilisateur choisit un concept de maintenance	
	6. Le système propose une série de cas problèmes relatifs au concept de maintenance choisi ainsi qu'un pourcentage symbolisant l'exactitude du lien entre le concept de maintenance et le cas problème
7. L'utilisateur choisit un de ces cas problèmes	
	8. Le système propose une série de questions relatives au cas problème choisi

9. L'utilisateur répond à l'ensemble des questions 10. Le système propose une recommandation en fonction des réponses de l'utilisateur	

Emission suggestion

Description : Un utilisateur souhaite émettre une suggestion à l'auteur du système.

Pré-conditions :
- L'utilisateur est authentifié.
- L'utilisateur a une suggestion à émettre à l'auteur du système.

Post-conditions :
- L'utilisateur a émis une suggestion à l'auteur du système.

Flux normal :

Utilisateur	Système
1. L'utilisateur manifeste son intention d'émettre une suggestion à l'auteur du système	
	2. Le système ouvre le logiciel de gestion de courriel par défaut de l'utilisateur
3. L'utilisateur entre ses suggestions 4. L'utilisateur envoie ses suggestions	
	5. La suggestion de l'utilisateur est envoyée à l'auteur du système

Consultation aide générale

Description : Un utilisateur souhaite obtenir une aide générale quant à l'utilisation du système.

Pré-conditions :
- L'utilisateur est authentifié.
- L'utilisateur a besoin d'aide.

Post-conditions :
- L'utilisateur a pris connaissance de l'aide générale du système.

Flux normal :

Utilisateur	Système
1. L'utilisateur manifeste son intention d'obtenir une aide générale quant à l'utilisation du système	
	2. Le système propose à l'utilisateur une page d'aide sur le fonctionnement général du système

Consultation aide locale

Description : Un utilisateur souhaite obtenir une aide localisée concernant un concept du système.

Pré-conditions :
– L'utilisateur est authentifié.
– L'utilisateur a besoin de clarifier un concept du système.

Post-conditions :
– L'utilisateur a pris connaissance de l'aide localisée d'un concept du système.

Flux normal :

Utilisateur	Système
1. L'utilisateur manifeste son intention d'obtenir une aide localisée sur l'un des concepts du système	
	2. Le système propose à l'utilisateur une page d'aide sur le concept du système qui pose problème

Consultation définition

Description : Un utilisateur souhaite obtenir une définition approfondie d'un terme présent dans la base de connaissance.

Pré-conditions :
– L'utilisateur est authentifié.

– L'utilisateur a besoin de plus d'information quant au sens d'un terme du système.

Post-conditions :
– L'utilisateur a pris connaissance de la définition du terme posant problème.

Flux normal :

Utilisateur	Système
1. L'utilisateur manifeste son intention d'obtenir la définition d'un terme présent dans le système	
	2. Le système propose à l'utilisateur une page de définition du terme qui pose problème

Scénarios - Vue *Administrateur*

Actions des utilisateurs

Idem Use Case Authentification
Idem Use Case Informations erronées "extends" Authentification
Idem Use Case Désauthentification
Idem Use Case Consultation de la base de connaissance

On notera que la vue administrateur permet de consulter les connaissances des évaluateurs internes et externes en même temps.

Idem Use Case Emission suggestion
Idem Use Case Consultation aide générale
Idem Use Case Consultation aide locale
Idem Use Case Consultation définition

Actions des experts

Idem Use Case Ajout dans la base de connaissance
Idem Use Case Modification dans la base de connaissance
Idem Use Case Retrait dans la base de connaissance

Ajout compte

Description : Un administrateur souhaite ajouter le droit d'accès au système à un nouvel utilisateur.

Pré-conditions :
- L'administrateur est authentifié.
- Un utilisateur a émis une demande d'accès au système.
- L'administrateur connaît le login, mot de passe, statut et date d'expiration à accorder au nouvel utilisateur.

Post-conditions :
- Le nouveau compte utilisateur a été créé.
- L'utilisateur a accès au système.

Flux normal :

Administrateur	Système
1. L'administrateur manifeste son intention d'ajouter un nouveau compte utilisateur	
	2. Le système demande à l'administrateur les informations nécessaires pour créer un nouveau compte
3. L'administrateur fournit ces informations	
	4. Le système vérifie que les données introduites sont correctes
	5. Le système signale à l'administrateur que le compte a été créé

Retrait compte

Description : Un administrateur souhaite retirer le droit d'accès au système à un utilisateur.

Pré-conditions :
- L'administrateur est authentifié.
- L'administrateur souhaite retirer l'accès au système à un utilisateur.
- L'administrateur connaît le login de l'utilisateur à retirer du système.

Post-conditions :
- Le compte utilisateur a été créé.
- L'utilisateur n'a plus accès au système.

Flux normal :

Administrateur	Système
1. L'administrateur manifeste son intention de supprimer un compte utilisateur	
	2. Le système propose à l'administrateur les logins des comptes utilisateurs actifs
3. L'administrateur sélectionne le compte à supprimer	
	4. Le système signale à l'administrateur que le compte a été supprimé

Modification statut

Description : Un administrateur souhaite modifier le statut d'un utilisateur du système.

Pré-conditions :
- L'administrateur est authentifié.
- L'admnistrateur souhaite modifier le statut d'un utilisateur du système.
- L'administrateur connaît le login de l'utilisateur en question ainsi que le nouveau statut à accorder.

Post-conditions :
- Le statut de l'utilisateur a été modifié.

Flux normal :

Administrateur	Système
1. L'administrateur manifeste son intention de modifier le statut d'un utilisateur	
	2. Le système propose à l'administrateur les logins des comptes utilisateurs actifs
3. L'administrateur sélectionne le compte à modifier	
	4. Le système propose à l'administrateur les différents statuts disponibles
5. L'administrateur sélectionne le nouveau statut de l'utilisateur	
	6. Le système signale à l'administrateur que le statut de l'utilisateur a été modifié

Modification mot de passe

Description : Un administrateur souhaite modifier le mot de passe d'un utilisateur du système.

Pré-conditions :
- L'administrateur est authentifié.
- L'administrateur souhaite modifier le mot de passe d'un utilisateur du système.
- L'administrateur connaît le login de l'utilisateur en question ainsi que le nouveau mot de passe.

Post-conditions :
- Le mot de passe de l'utilisateur a été modifié.

Flux normal :

Administrateur	Système
1. L'administrateur manifeste son intention de modifier le mot de passe d'un utilisateur	
	2. Le système propose à l'administrateur les logins des comptes utilisateurs actifs
3. L'administrateur sélectionne le compte à modifier	
	4. Le système propose à l'administrateur d'introduire un nouveau mot de passe
5. L'administrateur introduit le nouveau mot de passe de l'utilisateur	
	6. Le système signale à l'administrateur que le mot de passe de l'utilisateur a été modifié

Modification date d'expiration

Description : Un administrateur souhaite modifier la date d'expiration d'un compte utilisateur.

Pré-conditions :
- L'administrateur est authentifié.
- L'administrateur souhaite modifier la date d'expiration d'un compte utilisateur
- L'administrateur connaît le login de l'utilisateur en question ainsi que la nouvelle

date d'expiration.

Post-conditions :
– La date d'expiration du compte de l'utilisateur a été modifiée.

Flux normal :

Administrateur	Système
1. L'administrateur manifeste son intention de modifier la date d'expiration d'un compte utilisateur	
	2. Le système propose à l'administrateur les logins des comptes utilisateurs actifs
3. L'administrateur sélectionne le compte à modifier	
	4. Le système propose à l'administrateur d'introduire une nouvelle date d'expiration
5. L'administrateur introduit la nouvelle date d'expiration du compte de l'utilisateur	
	6. Le système signale à l'administrateur que la date d'expiration du compte de l'utilisateur a été modifiée

Informations erronées "extends" UC ajout compte à UC modification date d'expiration

Description : Un administrateur souhaite ajouter ou modifier des informations dans le système mais l'information saisie n'est pas valide. (Exemple : login déjà attribué, date non valide...

Pré-conditions :
– idem Use case ajout compte à modification date d'expiration.
– L'administrateur introduit une information non valide.

Post-conditions :
– Le scénario n'est pas poursuivi.

Flux alternatif :

Administrateur	Système
Idem Use Case ajout compte à modification date expiration jusqu'à l'action 5	
	5. Le système constate que l'information entrée est erronée. 6. Le système signale le problème à l'administrateur.

Scénarios - Vue *Expert*

Actions des utilisateurs

Idem Use Case Authentification
Idem Use Case Informations erronées « extends » Authentification
Idem Use Case Désauthentification
Idem Use Case Consultation de la base de connaissance

On notera que la vue Expert permet de consulter les connaissances des évaluateurs internes et externes en même temps.

Idem Use Case Emission suggestion
Idem Use Case Consultation aide générale
Idem Use Case Consultation aide locale
Idem Use Case Consultation définition

Ajout dans la base de connaissance

Description : Un expert souhaite faire évoluer la base de connaissance en ajoutant un mot clé, index, concept de maintenance, cas problème, question ou recommandation. Celui-ci peut également revoir les relations agençant ces différentes notions.

Pré-conditions :
- L'expert est authentifié.
- L'expert souhaite faire évoluer la base de connaissance en y ajoutant un élément.
- L'expert dispose des informations nécessaires à introduire dans la base de connaissance.

Post-conditions :
- La base de connaissance a été modifié.

Flux normal :

Expert	Système
1. L'expert manifeste son intention d'ajouter un élément dans la base de connaissance	
	2. Le système propose un domaine de modification parmi les index, mots clés, concepts de maintenance, cas problèmes, questions et recommandations
3. L'expert sélectionne parmi les domaines celui qui l'intéresse	
	4. Le système propose une liste d'informations à remplir
5. L'expert complète les informations nécessaires	
	6. Le système signale à l'expert que la base de connaissance a été modifiée

Modification dans la base de connaissance

Description : Un expert souhaite faire évoluer la base de connaissance en modifiant un mot clé, index, concept de maintenance, cas problème, question ou recommandation. Celui-ci peut également revoir les relations agençant ces différentes notions.

Pré-conditions :
- L'expert est authentifié.
- L'expert souhaite faire évoluer la base de connaissance en y modifiant un élément.
- L'expert connaît l'élément qu'il désire modifier.
- L'expert dispose des informations nécessaires à introduire dans la base de connaissance.

Post-conditions :
- La base de connaissance a été modifié.

Flux normal :

Expert	Système
1. L'expert manifeste son intention d'ajouter un élément dans la base de connaissance	

	2. Le système propose un domaine de modification parmi les index, mots clés, concepts de maintenance, cas problèmes, questions et recommandations
3. L'expert sélectionne parmi les domaines celui qui l'intéresse	
	4. Le système propose une liste des éléments disponibles dans le domaine choisi
5. L'expert sélectionne l'élément qu'il désire modifier	
	6. Le système propose une liste d'informations à modifier
7. L'expert modifie les informations nécessaires	
	8. Le système signale à l'expert que la base de connaissance a été modifiée

Retrait dans la base de connaissance

Description : Un expert souhaite faire évoluer la base de connaissance en supprimant un mot clé, index, concept de maintenance, cas problème, question ou recommandation. Celui-ci peut également revoir les relations agençant ces différentes notions.

Pré-conditions :
- L'expert est authentifié.
- L'expert souhaite faire évoluer la base de connaissance en y supprimant un élément.
- L'expert connaît l'élément qu'il désire modifier.
- L'expert dispose des informations nécessaires pour supprimer l'élément dans la base de connaissance.

Post-conditions :
- La base de connaissance a été modifié.

Flux normal :

Expert	Système
1. L'expert manifeste son intention de supprimer un élément dans la base de connaissance	

	2. Le système propose un domaine de modification parmi les index, mots clés, concepts de maintenance, cas problèmes, questions et recommandations
3. L'expert sélectionne parmi les domaines celui qui l'intéresse	
	4. Le système propose une liste des éléments disponibles dans le domaine choisi
5. L'expert sélectionne l'élément qu'il désire supprimer	
	6. Le système signale à l'expert que la base de connaissance a été modifiée

Informations erronées "extends" UC ajout en KB à UC modification en KB

Description : Un expert souhaite faire évoluer la base de connaissance en ajoutant ou modifiant un mot clé, index, concept de maintenance, cas problème, question ou recommandation mais il introduit une information erronée.

Pré-conditions :
- Idem Use Case ajout en KB à modification en KB.
- L'expert introduit une information non valide.

Post-conditions :
- Le scénario n'est pas poursuivi.

Flux alternatif :

Expert	Système
Idem Use Case ajout en KB à modification en KB jusqu'à l'action 7	
	8. Le système constate que l'information entrée est erronée. 9. Le système signale le problème à l'expert.

ANNEXE 2 : Diagrammes de robustesse

Cette annexe présente l'ensemble des diagrammes de robustesse de l'application.

Remarque : Il est à noter que dans tous les diagrammes de robustesse repris ci-dessous (excepté le cas d'authentification), l'acteur sera considéré comme étant préalablement authentifié avant d'effectuer l'opération.

Authentification

Description : L'acteur souhaite s'authentifier auprès du système.

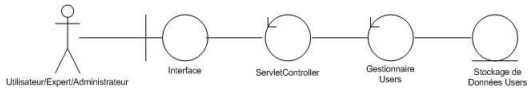


Fig. 1 – Diagramme de robustesse - Authentification

Désauthentification

Description : L'acteur souhaite se désauthentifier du système.

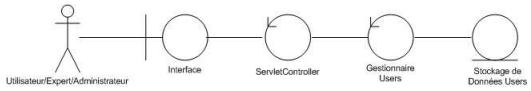


Fig. 2 – Diagramme de robustesse - Désauthentification

Consultation KB

Description : L'acteur souhaite consulter des informations relatives à son problème de maintenance.

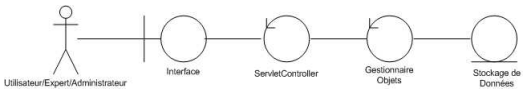


FIG. 3 – Diagramme de robustesse - Consultation KB

Consultation aide générale

Description : L'acteur souhaite consulter l'aide générale du système.

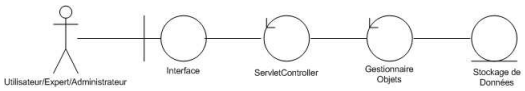


FIG. 4 – Diagramme de robustesse - Consultation aide générale

Consultation aide locale

Description : L'acteur souhaite consulter l'aide locale relative à un élément du système.

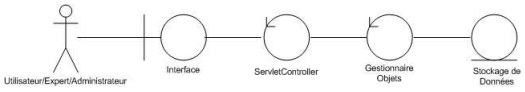


FIG. 5 – Diagramme de robustesse - Consultation aide locale

Consultation définition

Description : L'acteur souhaite consulter la définition d'un élément du domaine de la maintenance.

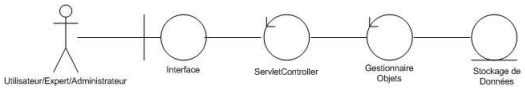


FIG. 6 – Diagramme de robustesse - Consultation définition

Ajout compte

Description : L'acteur souhaite ajouter un nouvel utilisateur dans le système.

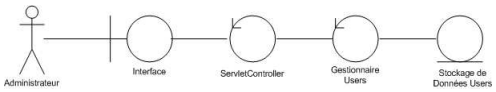


FIG. 7 – Diagramme de robustesse - Ajout compte

Retrait compte

Description : L'acteur souhaite retirer un utilisateur du système.

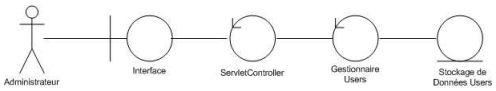


FIG. 8 – Diagramme de robustesse - Retrait compte

Modification statut

Description : L'acteur souhaite modifier le statut d'un utilisateur du système.

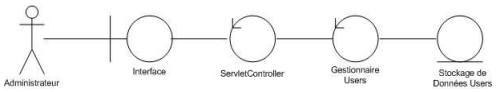


FIG. 9 – Diagramme de robustesse - Modification statut

Modification mot de passe

Description : L'acteur souhaite modifier le mot de passe d'un utilisateur du système.

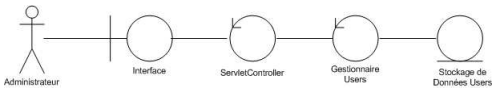


FIG. 10 – Diagramme de robustesse - Modification mot de passe

Modification date d'expiration

Description : L'acteur souhaite modifier la date d'expiration d'un utilisateur du système.

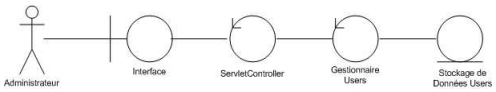


FIG. 11 – Diagramme de robustesse - Modification date d'expiration

Ajout de connaissances dans la KB

Description : L'acteur souhaite ajouter un élément de connaissance en KB

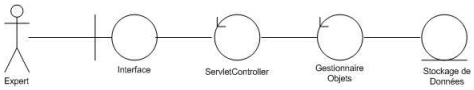


FIG. 12 – Diagramme de robustesse - Ajout KB

Modification de connaissances dans la KB

Description : L'acteur souhaite modifier un élément de connaissance en KB

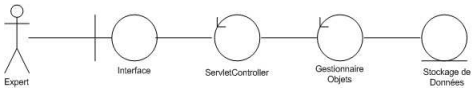


FIG. 13 – Diagramme de robustesse - Modification KB

Retrait de connaissances dans la KB

Description : L'acteur souhaite retirer un élément de connaissance en KB.

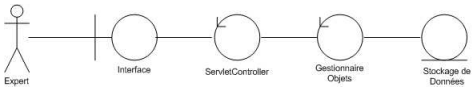


FIG. 14 – Diagramme de robustesse - Retrait KB

ANNEXE 3 : Publication

Software Maintenance Maturity Model ($S3^m$DSS)
A Decision Support System

Alain April, Jean-Marc Desharnais, Arnaud Counet
Ecole de Technologie Supérieure, Montreal, Canada
Faculté Universitaire Notre-Dame de la Paix, Namur, Belgium
Email : alain.april@etsmtl.ca , jmdeshar@ele.etsmtl.ca , arnaud.counet@gmail.com

Keywords : Knowledge-based system, software maintenance, maturity model, ontology, decision support.

Abstract : Maintaining and supporting the software of an organization is not an easy task, and software maintainers do not currently have access to tools to evaluate strategies for improving the specific activities of software maintenance. This article presents a knowledge-based system which helps in locating best practices in a software maintenance maturity model ($S3^m$). The contributions of this paper are : 1) to instrument the maturity model with a support tool to aid software maintenance practitioners in locating specific best practices ; and 2) to describe the knowledge-based approach and system overview used by the research team.

1. INTRODUCTION

Knowledge transfer of a large number of best practices, described in a maturity model, has proved difficult [1]. This is especially true during the training stage for an assessor or a new participant in a process improvement activity. It is also challenging to quickly refer to, or access, the right practice, or subset of practices, when trying to answer specific questions during or after a maturity evaluation.

The maturity model $S3^m$ contains a large number of concepts and information which are structured in many successive levels [2], [14]. The first is called the process domains level, and reflects the main process knowledge areas of a maturity model. In the $S3^m$, there are 4 process domains (*process management*, *maintenance request management*, *software evolution engineering* and *support to software engineering evolu-*

tion). Each process domain is broken down into one or more key process areas (KPAs). These KPAs logically group together items which conceptually belong together. A KPA is further divided into *roadmaps* with one or more *best practices*, spanning five $S3^m$ maturity levels. The complete $S3^m$ has 4 domains, 18 KPAs, 74 roadmaps and 443 best practices. It would be beneficial to have a knowledge-based system (KBS) to help access this complex structure and large amount of information. A potential solution to this problem would be to develop a knowledge-based system for the $S3^m$. This KBS should be available for both maintainers and maintenance client. The proposed modelling of a software maintenance KBS is based on the van Heijst methodology [3], which consists of constructing a task model, selecting or building an ontology [4], mapping the ontology onto the knowledge roles in the task model and instantiating the application ontology with this specific domain knowledge. According to van Heijst, there are at least five different types of knowledge to be taken into account when constructing such a system : tasks, problem-solving methods, inferences, the ontology and the domain knowledge (see Figure 1). Van Heijst uses the different types of knowledge in a more generic way than we do in this document, and these have been adapted for us by [13]. For van Heijst, domain knowledge refers to a collection of statements about the domain [4]. The domain of this specific research is software maintenance, and it is divided into 4 process domains. Examples of statements are presented in section 3. At a high level, the ontology refers to a part of the software maintenance ontology proposed by [5] presented in section 4. The problem solving methods and tasks are described at length in section 5. The tool environment and conclusion, as well as future work, are presented in sections 6 and 7. Section 2 begins by presenting the goals of the $S3^m$ architecture.

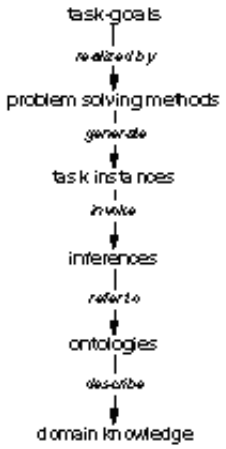


Figure 1 - The different components of knowledge models [3]

2. GOALS OF THE S3m ARCHITECTURE

The $S3^m$ was designed as a customer-focused benchmark for either :

- Auditing the software maintenance capability of a service supplier or outsourcer ; or
- Supporting the process improvement activities of software maintenance organizations.

To address the concerns specific to the maintainer, a distinct maintenance body of knowledge is required. The $S3^m$ is also designed to complement the maturity model developed by the SEI at Carnegie Mellon University in Pittsburgh [6] by focusing mainly on practices specific to software maintenance. The architecture of the model locates the most fundamental practices at a lower level of maturity, whereas the most advanced practices are located at a higher level of maturity. An organization will typically

mature from the lower to the higher maturity level as it improves. Lower-level practices must be implemented and sustained for higher-level practices to be achieved.

Rank	Maintenance problem
1	Managing fast-changing priorities
2	Inadequate testing techniques
3	Difficulty in measuring performance
4	Missing or incomplete software documentation
5	Adapting to rapid changes in user organizations
6	A large number of user requests in waiting
7	Difficulty in measuring/demonstrating the maintenance team's contribution
8	Low morale due to lack of recognition
9	Not many professionals in the field, especially experienced ones
10	Little methodology, few standards, procedures or tools specific to maintenance
11	Source code complex and unstructured
12	Integration, overlap and incompatibility of systems
13	Little training available to personnel
14	No strategic plans for maintenance
15	Difficulty in meeting user expectations
16	Lack of understanding and support from IT managers
17	Maintenance software running on obsolete systems and technologies
18	Little will for reengineering applications
19	Loss of expertise when employees leave

Tab 1 - Top maintenance problems [8]

3. S3m AND KNOWLEDGE STATEMENTS

Software maintainers experience a number of problems. These have been documented and an attempt made to rank them in order of importance. One of the first reported investigations was conducted by Lientz and Swanson [7]. They identified six problems related to users of the applications, to managerial constraints and to the quality of software documentation. Other surveys have found that a large percentage of the software maintenance problems reported are related to the software product itself. This survey identified complex and old source code which was badly documented and structured in a complex way. More recent surveys conducted among attendees at successive software maintenance conferences [8] ranked perceived problems in the following order of importance (see Table 1). These are also examples of knowledge statements about the domain of software maintenance. Key to helping software maintainers would be to provide them with ways of resolving their problems by leading them to documented best practices.

There is a growing number of sources where software maintainers can look for best practices, a major challenge being to encourage these sources to use the same terminology, process models and international standards. The practices used by maintainers need to show them how to meet their daily service goals. While these practices are most often described within their corresponding operational and support processes, and consist of numerous procedures, a very large number of problem-solving practices could be presented in a KBS which would answer their many questions about those problems. Examples are presented in section 6. Maintenance client problems could also be linked to these internal problems because of the impacts it can occur. When using the software maintenance ontology in the KBS, it was necessary to consider the structure of the maturity model relationship between the many process domains, roadmaps and practices. This problem is addressed next.

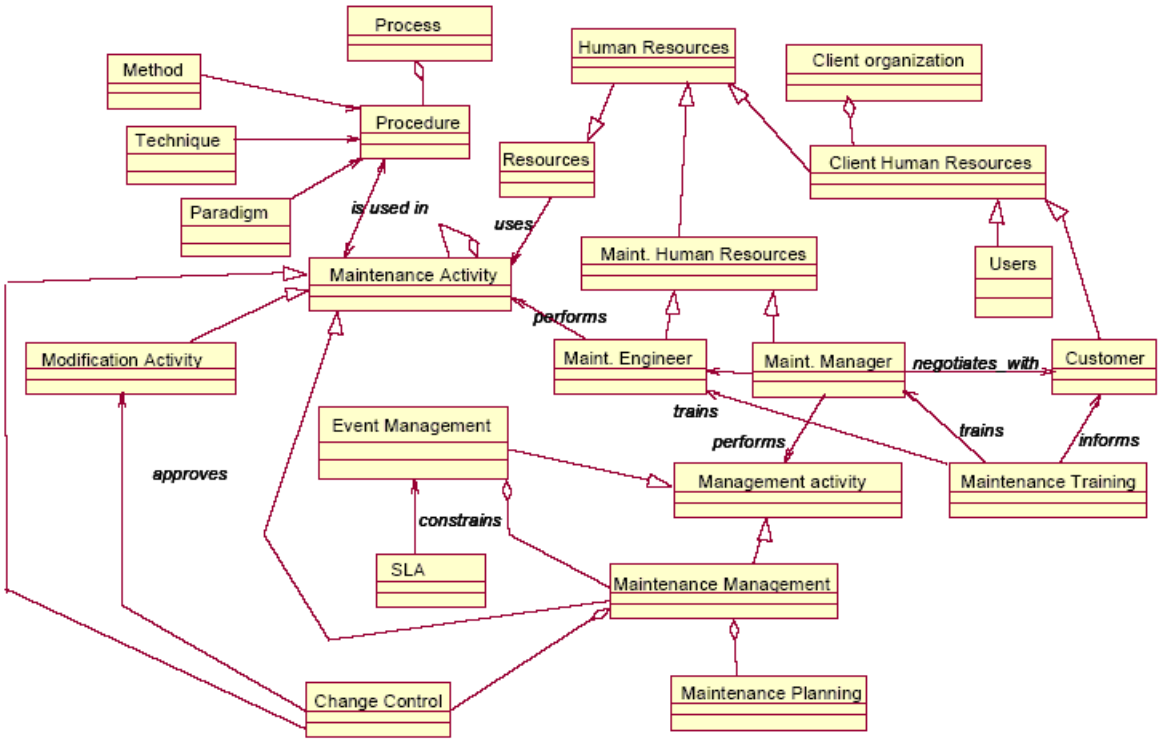


Figure 2 - Part of the software maintenance ontology of (Kitchenham and et al., 1999)

4. ONTOLOGY OF THE SOFTWARE MAINTENANCE BODY OF KNOWLEDGE

We elected to implement only a subset of the ontology developed by Kitchenham et al. [5] and Ruiz et al. [9] for the initial trial of this research project. The Kitchenham ontology was chosen because its author is well known in Software Engineering maintenance. The following authors also write on the subject [9], [10] and [11] from the point of view of the knowledge system. Figure 2 describes the different maintenance concepts considered surrounding a software maintenance activity. Software maintenance is highly event-driven, which means that some maintenance activities are unscheduled and can interrupt ongoing work. This subset of the ontology represents many, but not all, the concepts involved in responding to the questions related to the first problem identified by Dekleva : "Managing fast-changing priorities". Maintainers agree that this is the most important problem they face. How can they handle the fast-changing priorities of the customer ? Solutions to this problem are likely to be found by using many paths through the maintenance concepts of the ontology. Navigation through these concepts should lead to associated concepts which are conceptually linked and likely to contribute to a solution, like the need for better event management, change control, maintenance planning, Service Level Agreements, maintenance manager negotiation, training, procedures, and so forth. Many more concepts must be in-

volved to contribute to all aspects of the solution, but our purpose is to show the utility of a KBS in the software maintenance domain, and it therefore starts with a constrained number of concepts. Maturity models typically include the detailed best practices that could be of help in solving this type of problem. The main issue is that the best practice locations and their interrelationships are hidden in the layered architecture of the maturity model, specifically in its process domains, KPAs and roadmaps. It is therefore necessary to find a way to link this layered architecture with the maintenance concepts of the ontology and proceed to analyze the tasks required to build a KBS to support the maintainers in their quest for solutions. The next section describes the navigation concepts that have been implemented in *S3*^{m}*DSS*. The user of the KBS navigates using a sequence of tasks that will lead him through a further sequence of tasks.

5. HIGH LEVEL VIEW OF S3m KBS

According to [3], the first activity in the construction of a KBS is the definition of task analysis. Task analysis begins, at a high level, with a definition of an index of terms. This index includes words commonly used in software engineering (see Figure 3). From this index, a subset of more restrictive words is identified. This subset is a list of keywords recognized specifically in software maintenance. Each keyword is then connected to one or more maintenance concepts. A maintenance concept, in software maintenance, is a concept found in the Software Maintenance Body of Knowledge and ontology (see Figure 2). Every maintenance problem identified by Dekleva has been translated into a case problem and connected to the software maintenance ontology. Each case problem is then linked to a set of themes (questions) which help the user of the KBS to navigate into a part of the maturity model that will propose recommendations in the form of best practices. The link between the maintenance concepts and the maturity model is made in the themes concept. Themes are questions which have been developed to hop from node to node in the ontology. A close look at Figure 2 reveals that the themes concept can combine different maintenance concepts and, finally, create a set of recommendations of the maturity model. For every best practice, there is a linked theme (or choice) from which the user can select (also called facts) which will lead to a final specific set of recommendations. This 1-1 matching between theme and recommendation will contribute to a composed set of recommendations directly adapted to the user context. Above all of this, a distinction between internal maintenance engineers and maintenance client has been made. We think that the same problems are involved for both side but we need to adapt the way we ask. In this case, when a maintenance client uses the system, themes are adapted to his understanding. Provided recommendations are some kind of invitation to his maintainer to follow different rules. This could both help maintenance enterprise but also client one.

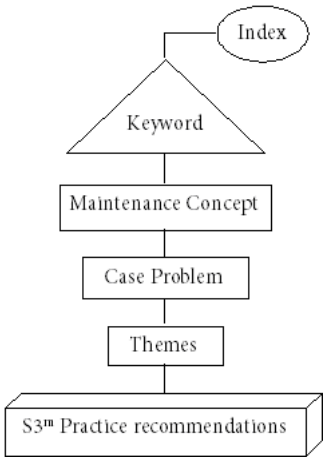


Figure 3 - High-level view of $S3^m$

Expanding the 6 high-level tasks in Figure 3, we propose 12 detailed tasks which will help identify a subset of best practices related to the S3m.

6. TOOL ENVIRONMENT

Next will explain how the tool environment was built. Then it will be explain how the KBS helps the user answer a practical question and how the expert insert a complete case problem.

6.1. ENVIRONMENT

The $S3^m$DSS was built using Java, Java Server Pages, JavaScript, CSS and HTML, and supports the $S3^m$. This combination of technologies was selected for its easy access via the Internet. Behind that, a SQL Server database was added in order to manage the knowledge base. This choice was justified by the lack of reactivity that XML parsing proposed before. The architecture is based on a 3-tiers model providing easy maintainability and composed by a presentation layer, business layer and DAO layer. The business layer has been split into 2 parts : the first regrouping all the controlling servlets and the second regrouping all the business methods. Servlets assure the right communication between the presentation layer and the business layer while business methods communicate with the DAO layer. The $S3^m$DSS KBS was developed by Counet, a Master's degree student from the University of Namur, Belgium, during a research exchange program [12].

#	Questions
A	Are there training plans address to new maintenance engineers about generic topics like management and processes activities?
B	Do maintenance engineers periodically update the proficiency with the software and its infrastructure they maintain?
C	Are maintenance engineers trained and motivated to perform well when using the processes/services and their support role?
D	Is there some training communication with customers offered to software maintenance engineers?
E	Do you use any internal benchmarking data to guide the training of maintenance resources?
F	Does the maintenance organisation have a training budget?
G	Are there plans describing the training needed for each maintenance engineering position and application software?
H	Is there training time planned?
I	Do senior maintainers familiarise new employees?
J	Are training needs defined for both technical and management responsibilities for each development project?
K	Do people working on the pre-delivery and transition receive the training deemed appropriate by the software developer?

Tab 2 - KBS questions

Currently, more than 550 words and 70 keywords have been introduced into the system. 5 of the 19 maintenance problems identified by Dekleva have been solve but there is still a great deal of work required to populate the knowledge base for all the $S3^m$ practices and allow users to obtain answers to all the software maintenance problems. $S3^m$DSS is composed of 3 different

interface types : administrator, expert and user layout. The administrator interface manages access to $S3^m$DSS, while the expert interface gives the expert the option of adding new index words, keywords, concepts, cases, themes and recommendations. Next will show in practice how the KBS helps the user answer the following question : How to improve maintenance training ?

6.2. USER INTERFACE

First of all, the user enters a word that will identify a suggested keyword. As an example, the user enters : *training*. He will select a keyword that will help the KBS find the most closely related KPA and roadmap concepts. Currently, the system presents the following keyword : *maintenance training*. The KBS presents the maintenance concepts, which are related to KPA and roadmap, to the user. It will present the concepts in order of priority to the user. A percentage is related to each concept. The expert has previously established this percentage. The user chooses one or multiple concepts, *maintenance human resources* in our example. The KBS presents the case problems associated with the maintenance concept selected to the user. It will present the case problems in order of priority to the user. A percentage is related to each problem. The expert has previously established this percentage. The user chooses one or multiple case problems, *little training available to maintenance engineers* in our example. With this case problem, there are 11 themes presented to the user in the form of questions (see Table 2). The user will find facts for each practice (theme). He can answer yes or no to any of the themes. In function of the facts chosen, the system composed a set of recommendations to the user. Figure 4 shows how the KBS will recommend the following solution (simplified for this paper) : RecSet.

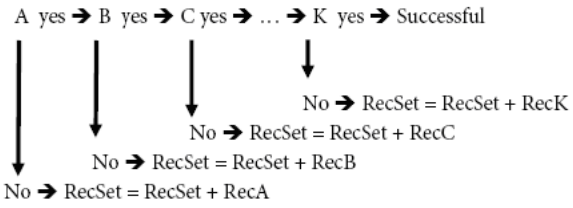


Figure 4 - KBS recommendation mechanism

Figure 5 (next page) shows an example of the user layout in the previous case problem. The user layout is made up of 4 dynamic tables representing all the concepts we discuss before. Each table is displayed step by step by user selection and associated with a help function. In the top of the layout, a toolbar has been inserted to start every research by typing a word into the system or selecting a keyword. Next will show in practice how a maintenance expert can enter a case problem into the KBS.

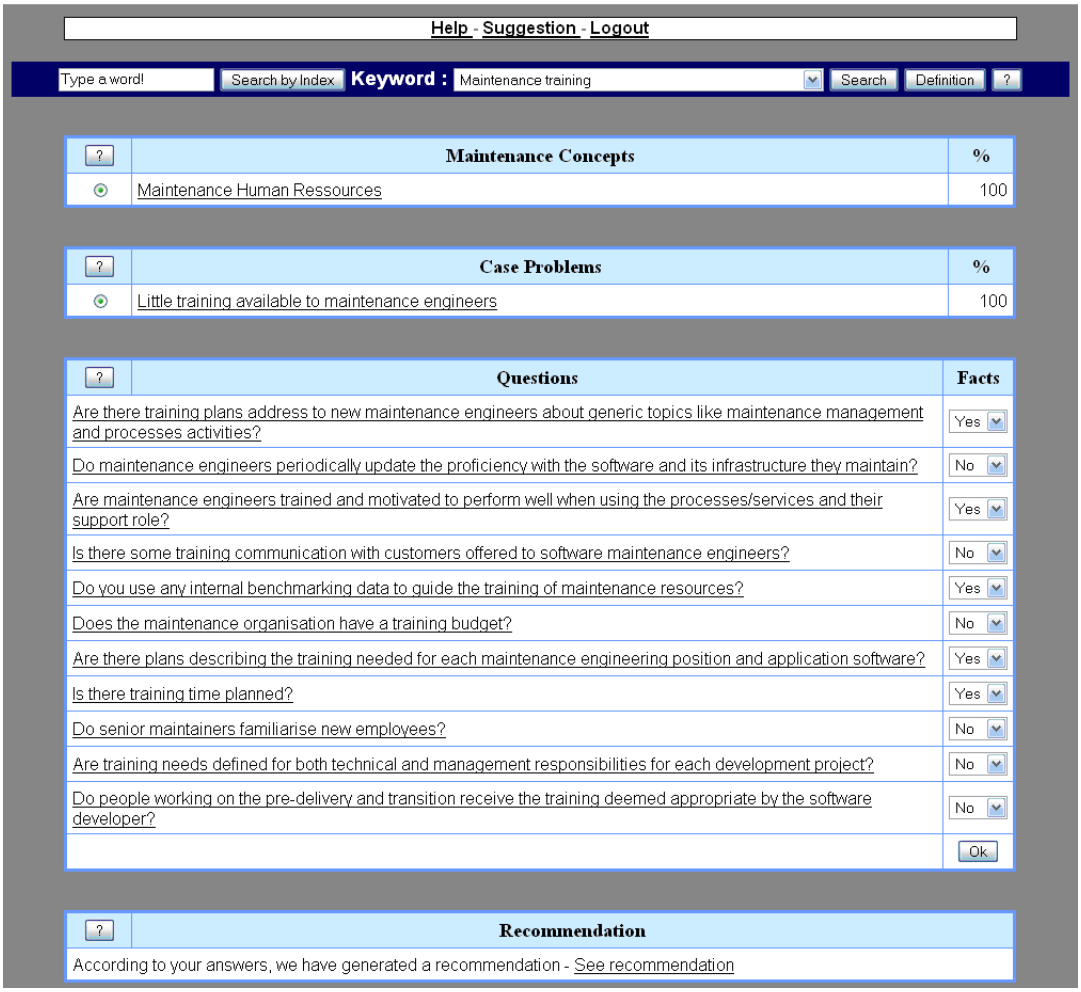


Figure 5 - $S3^m$DSS user interface layout

6.3. EXPERT INTERFACE

Figure 6 (next page) shows an example of the expert layout. A purified table propose expert to add, modify or delete high level view elements. Expert can also add a complete case to the KBS by respecting the following recommendation, question, case problem, maintenance concept, keyword and word order because of the links between elements. Below the top table, a form is proposed where expert can fill information like element name, help content or links with upper or lower elements. All existing elements are accessible by conventional html lists and can be added very easily by selecting and pressing a button. When validation button is pressed, an additional form shown in Figure 7 appears. Experts can then complete association percentages between linked elements. Note that experts can use HTML mark-ups into recommendation text to add hyperlinks, lists or tables.

7. CONCLUSION AND FUTURE WORK

Identifying the best practices in a maturity model is a difficult task, considering

their number and the multiple appropriate answers associated with each of them. Our hypothesis is that a KBS could help in finding an appropriate recommendation. The next step in this research project is to populate the KBS, validate the results with experts in the domain and determine whether or not the KBS is a useful support tool for training on the content of the maturity model. The $S3^m$DSS tool is available on **http ://www.gelog.etsmtl.ca/S3mDSS/**

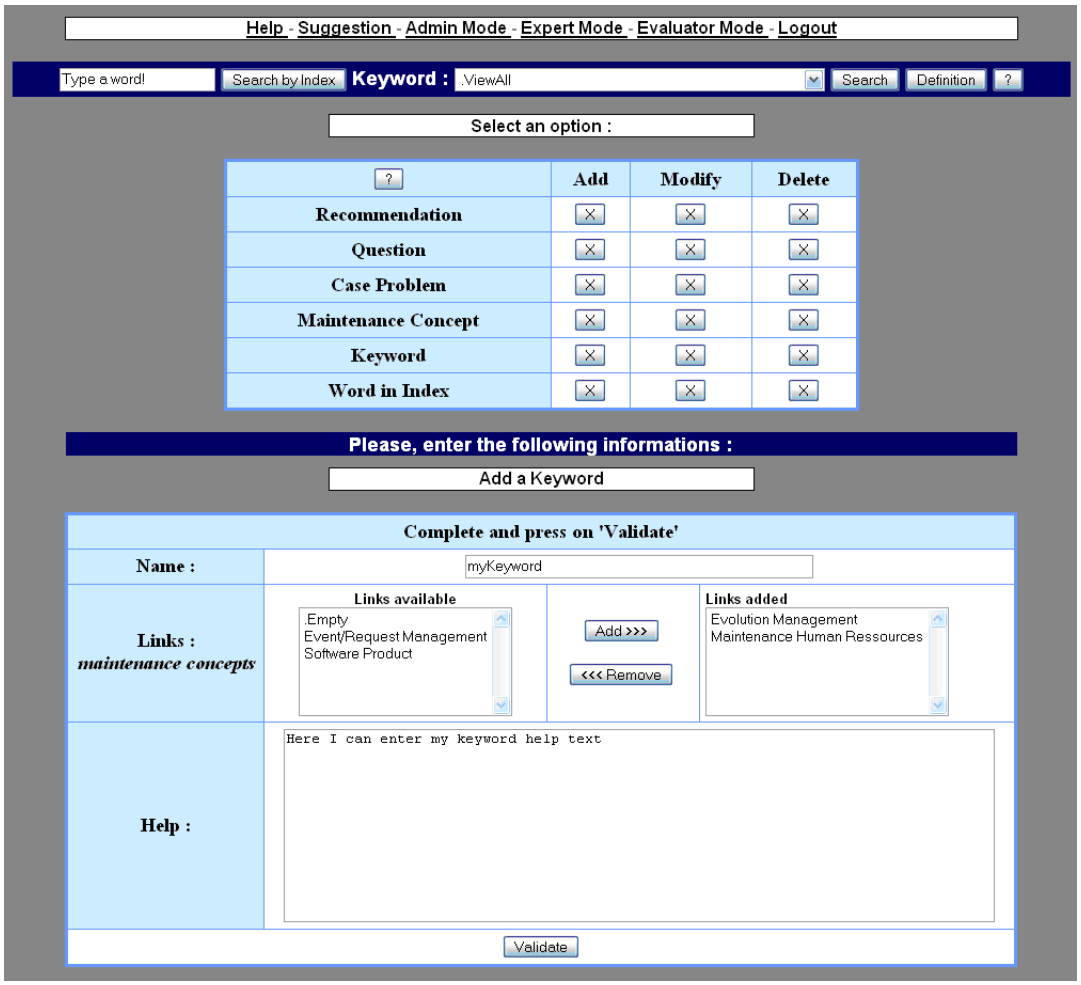


Figure 6 - $S3^m$DSS expert interface layout

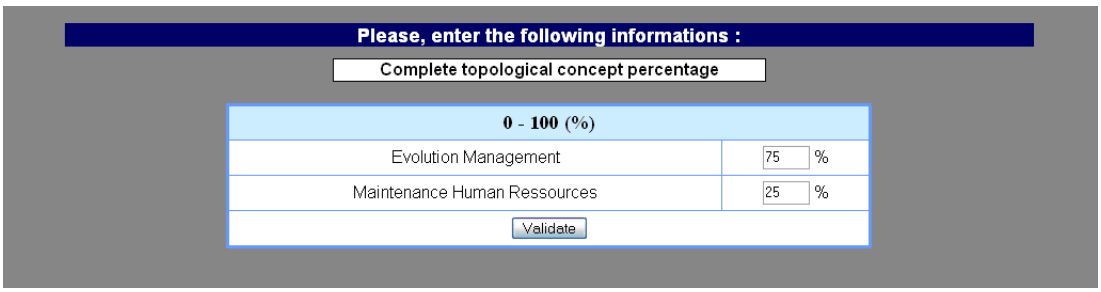


Figure 7 - $S3^m$DSS expert form layout

REFERENCES

[1] Abran, A., Moore, J. W., Bourque, P., Dupuis, R. and Tripp, L., *Guide for the Software Engineering Body of Knowledge (SWEBOK)*, Ironman version, IEEE Computer Society Press : Los Alamitos CA,2004 ; 6-1-6-15, Montréal, http ://www.swebok.org [27 January 2005].

[2] April, A., Abran, A. and Dumke, R. *SMCMM Model to Evaluate and Improve the Quality of the Software Maintenance Process : Improvements, traceability and conformity to standards*, CSMR 2004 8th European Conference on Software Maintenance and Reengineering, (2004a) Tampere (Finland).

[3]Van Heijst, G., Schreiber, A. T. and Wielinga, A., *Using Explicit Ontologies in KBS Development*, 2003 University of Amsterdam, Department of Social Science Informatics, Amsterdam, 1997.

[4] Uschold, M. and Jasper, R. (2001), *An ontology for the management of software maintenance projects*, In Industrial Knowledge Management : a microlevel approach,Bedford (UK), pp. 549-563.

[5] Kitchenham, B. and et al. (1999), *Towards an Ontology of Software Maintenance*, J. Softw. Maint :Res. Parct., 11(6) :365-389.

[6] CMMi (Ed.) (2002) *Capability Maturity Model Integration for Software Engineering (CMMi)*, Version 1.1, CMU/SEI-2002-TR-028, ESC-TR-2002-028, Carnegie Mellon University.

[7] Lientz, B. and Swanson, E. (1981), *Problems in Application Software Maintenance*, Communications of the ACM, 24, 11, 763-769.

[8]Dekleva, S. M., *Delphi Study of Software Maintenance Problems*, International Conference on Software Maintenance (CSM 1992) (1992) IEEE Computer Society Press : Los Alamitos CA.

[9] Ruiz, F., Vizcaino, A., Piattini, M. and Garcia, F. (2004) *International Journal of Software Engineering and Knowledge Engineering*, 14, 3 323-349.

[10] Vizcaíno, A. Favela, J. and Piattini, M., *A multi-agent system for knowledge management in software maintenance*, KES 2003 (2003), Springer Verlag, Oxford, UK.

[11] Dias, M., G. Anquetil, N. and Oliveira, K. M. (2003), *Organizing the Knowledge Used in Software Maintenance*, Journal of Universal Computer Science, 9, 7 64-658.

[12] Counet, A. (2007), *Amélioration du processus de la maintenance du logiciel par un système informatisé d'aide à la décision*, FUNDP, Namur, Belgium.

[13] Desharnais, J.-M., *Application de la mesure fonctionnelle COSMIC-FFP : une approche cognitive*, UQAM, Montréal, 2004.

[14] April, A., Abran A. and Dumke, R., *Assessment of Software Maintenance Capability : A model and its Design Process*, IASTED 2004, Conference on Software Engineering (2004b), Innsbruck (Austria).