UNIVERSITÉ DU QUÉBEC À MONTRÉAL

PROPOSITION D'UNE APPROCHE À BASE DE CAS POUR LA RÉUTILISATION DES UNITÉS DE PROGRAMMES

MÉMOIRE

PRESENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE DE GESTION

Par

DIPA DIABATÉ

AVRIL 2006

A mon Père Mamby Diabaté

A ma Mère Sétou Coulibaly

A toute la famille Diabaté

REMERCIEMENTS

Mes remerciements vont à l'endroit de Seaquest-Infotel, et à son président Raphaël Nbogni pour l'estime qu'il a portée en ma personne en m'accordant conjointement avec le laboratoire GDAC une bourse d'étude ainsi que la disponibilité des ressources matérielles qui m'ont permis d'effectuer mes recherches.

Je remercie chaleureusement Pr. Roger Nkambou pour son encadrement tout au long de ce travail, ainsi que son soutien indéfectible. Qu'il trouve dans cet ouvrage un témoignage de mon profond remerciement.

Mes remerciements vont à l'endroit du Pr. Martin Cloutier, Directeur de département de l'informatique pour son soutien et ses conseils tout au long de ma formation.

Je remercie Pr. Hafed Mili pour avoir accepté d'être examinateur et bon critique du travail. Je lui exprime toute ma gratitude.

Je remercie Dr Roland Yatchou pour le support précieux qu'il m'a accordé durant ce travail, ses conseils, sa disponibilité et la lecture de ce document m'ont été très utile. Je lui suis très reconnaissant.

Je remercie chaleureusement, Najmedine Jardack (MBA) pour ses conseils, son soutien indéfectible et son sens critique du travail. Je lui exprime toute ma gratitude.

Mes remerciements vont à l'endroit des personnes de qualité exceptionnelle, collègues et amis Valery Bévo, Hervé Donfack, Ghislain Ngantchaha et Kaufmann Bouhom pour l'intérêt qu'ils ont manifesté envers ce travail, pour la lecture et la correction dans le but d'améliorer ce travail. Je vous exprime toute ma gratitude.

J'exprime toute ma gratitude à Madame Sylvie Wamba, Mlle Mireille Zatcha, Alain Hounang, Ibrahim Maiga, pour leur soutien et l'amitié témoignée.

Je remercie également tout le personnel de Seaquest-Infotel et de TI.

Un grand merci aux familles Diarra, Tall, Diouf et Diessé pour leur soutien.

Je remercie tous ceux qui ont contribués à l'aboutissement de ce travail et que je ne peux remercier individuellement.

SOMMAIRE :

LISTE DES ABRÉVIATIONS

CIAO-SI	Conception intelligemment assisté par ordinateur d'un système d'information.
CBR	Case based reasoning.
UML	Unified Modeling Language
RUP	Rational Unified Process
EDS	Systèmes de Données Électroniques
LGI	Le Groupe Infotel Inc. / Seaquest-Infotel
GDAC	Laboratoire de Gestion, Diffusion et Acquisition des Connaissances, UQAM.
TI	Technologie Infotel.

LISTE DES TABLEAUX

LISTE DES FIGURES

RESUMÉ

L'amélioration constante de la qualité des logiciels et des processus de développement logiciel est depuis plusieurs décennies, au cœur des préoccupations et de nombreux travaux de recherche dans le domaine de l'ingénierie des systèmes d'information. Parmi les multiples solutions proposées, la réutilisation apparaît pour bien des chercheurs comme l'une des pistes les plus prometteuses pour la réduction du coût de production et l'amélioration de la qualité de systèmes [MILI 1999].

Plusieurs formes de réutilisation ont été introduites : la réutilisation des spécifications, la réutilisation des composants, la réutilisation des codes, la réutilisation des expériences, etc. [R.P. Diaz, 87], [Kruger 92], [NKambou et al. 2003, 2004]. Le but du projet CIAO-SI est de construire et de maintenir une mémoire d'expérience pour l'organisation pouvant supporter tout le cycle de développement.

Á ce jour, les travaux du projet CIAO-SI [NKambou et al. 2003, 2004] initié par le Groupe Infotel Inc et le laboratoire GDAC se sont limités aux phases d'analyse et de conceptions. Nous nous proposons dans notre travail de recherche d'étendre la réflexion à la phase d'implantation ('codage'). Ce volet vise à doter le système CIAO-SI d'un module permettant de rechercher et d'indexer les unités de programmes éprouvées, validées et stockées dans le but de leur réutilisation pour des nouveaux projets de développement logiciel.

La plupart des outils recensés utilisent des approches d'indexation qui se limitent à l'aspect descriptif du problème (caractéristique du problème, le contexte d'élaboration pour filtrer les résultats, utilisation des concepts pour définir les ontologies) et non à l'intégration de la solution proposée ; ce qui n'est pas approprié dans la réutilisation des codes. Nous avons proposé et implémenté une approche de réutilisation des unités de programmes basée sur le CBR et la classification à facettes.

L'implantation de cette approche dans le contexte de CIAO-SI, une plateforme de réutilisation dans les phases en aval de la réalisation, nous a permis d'obtenir des résultats encourageants. Finalement nous avons effectué une évaluation expérimentale par la méthode de Salton et cela nous a permis d'évaluer la pertinence de nos résultats

Mots clés : Réutilisation des codes, développement logiciel, indexation des codes, ontologie, CIAO-SI, raisonnement à base de cas

Chapitre 1 : Introduction

1.1 Contexte

La réutilisation dans le développement de logiciel constitue un défi majeur pour beaucoup de sociétés de développement de logiciels. Au delà de la réduction substantielle du coût de développement et l'amélioration de la productivité, il contribue significativement à améliorer la qualité du logiciel [NKambou et al. 2004]. Au niveau de la recherche, l'aide à la réutilisation est un secteur de recherche de plus en plus populaire auprès de nombreuses équipes de recherches. Il en a découlé de nombreux outils d'aide à la réutilisation. On pourrait citer par exemple DéjàVu [Burton, B.A., et al 1987], RSL Kibitzer [Barry Smyth et all, 2001], KBRAS [Yoelle S. Maarek et all, 1994] CAESAR [Frakes, W.B & Nejmeh, B.a, 1987]. Ces outils sont pour la plupart, sinon tous, spécifiques à une phase précise dans le processus de développement. Aucun outil ne couvre toutes les phases du processus de développement. Fort de ce constat, le projet CIAO-SI dans lequel s'inscrit notre travail s'est fixé pour objectif de construire une mémoire d'expertise qui servirait de base pour la réutilisation couvrant tout le cycle de développement. Pour le moment, seules les phases d'analyse et de conception ont été couvertes. Nous nous proposons donc dans le cadre de notre travail d'étendre la réflexion à la phase d'implémentation en abordant la question de la réutilisation du code.

1.2 Objectifs de la recherche

La réutilisation des unités de programmes constitue aujourd'hui un secteur très promoteur pour des compagnies de développement des logiciels. Nous avons recensé dans la littérature plusieurs études [Dard 94], [Morisio 02], [MILI 1989], [NKambou et al. 2003,2004], etc. relatant des expériences vécues dans les entreprises ainsi que plusieurs travaux de recherche initiés en vue d'apporter une solution aux

problèmes de réutilisation des unités de programme. Certains travaux ont abouti à des outils [Burton, B.A., et al. 1987], [Barry Smyth et al. 2001] [Frakes, W.B & Nejmeh, B.a, 1987] et d'autres à des méthodes spécifiques de réutilisation [Petro Diaz, 1987], [Kruger 1992], [Peter, 2004]. Notre travail s'inspire de ces travaux et consiste à doter l'outil CIAO-SI d'un module permettant la validation et le stockage ainsi que la recherche et l'indexation des unités de programmes éprouvées dans le but de les réutiliser dans de nouveaux projets.

1.3 Problématique

Les développeurs de logiciels désireux de réutiliser les unités de programme découlant d'un ou de plusieurs projets précédents font généralement face aux problèmes de recherche et de localisation du fragment de code approprié [Peter, 2004]. Cette difficulté représente un obstacle majeur à la réutilisation et constitue une problématique actuelle de recherche. Les outils existants d'aide à la réutilisation de codes offrent des supports passifs à la recherche et à l'indexation [NKambou et al. 2004]. Il en résulte que les résultats fournis sont moins pertinents (difficulté à trouver le code approprié, difficulté de comprendre la structure des bibliothèques de fonctions, ou de librairies de code parfois trop complexe, etc.). Pour l'outil CIAO, nous nous proposons de mettre en place un support d'aide à la réutilisation des unités de programmes. Comment mettre en place un tel support ?

L'approche CBR [Watson, 2002] nous semble la plus indiquée pour la mise en place d'un tel support [Peter, 2004]. Par conséquent, quatre questions majeures servent de fil conducteur dans le travail :

- Comment décrire et représenter une unité de programme sous forme de cas?
- Comment indexer un tel cas?

- Comment retrouver efficacement une unité de programme appropriée au contexte actuel?
- Comment assurer la pro-activité dans le support offert aux utilisateurs?

Une réponse à chacune de ces questions constituera une contribution certaine à la réutilisation des codes. Cependant la question d'adaptation des unités de programmes n'est pas abordée dans ce travail compte tenu du délai imparti.

1.4 Contributions.

Au terme de cette recherche, nos contributions ont été les suivantes :

- La proposition d'une description et d'une représentation des unités de programmes sous forme de cas réutilisables.
- La proposition d'une approche de recherche et d'indexation des unités de programme basée sur la définition de facettes
- La mise en place d'un système d'aide à la réutilisation des unités de programmes.
- L'intégration du système d'aide dans CIAO-SI en vue d'un support proactif dans un contexte de programmation.

1.5 Méthodologie

Nous avons opté pour l'utilisation le Case Based Reasoning (CBR) ou raisonnement a base de cas. Plusieurs chercheurs ont montré que cette approche procure un moyen efficace pour la réutilisation de codes [Peter, 2004]. Le CBR se définit comme une méthodologie de résolution de problème basée sur les connaissances contenues dans les cas antérieurs [Watson, 2002]. Il s'agit d'une approche de résolution de problème proche du raisonnement humain [Rabea, 2001].

Un cas symbolise un artefact qui peut être utile lorsqu'un nouveau problème se présente [Dussaux, 2001]. Cette méthodologie comprend quatre phases qui sont : la recherche, la réutilisation, la révision et l'apprentissage. Afin de compléter les dites phases et réaliser notre travail, nous nous servons du langage UML pour la représentation et la structuration des unités de programmes. Nous utilisons la classification à facette introduite par Petro-Diaz [Petro-Diaz, 87] pour réaliser l'indexation. Nous exploitons l'efficacité et la puissance des SGBD pour l'implantation de la structure de cas. Une évaluation expérimentale à l'aide de la méthode élaborée par Salton [Salton 89] nous permet d'évaluer la pertinence de nos résultats.

Le Groupe Infotel, une société de développement des applications, avec ses nombreux projets, nous sert de cadre expérimental pour l'évaluation de l'outil.

1.6 Structure du document

Le document est organisé en quatre chapitres, un premier chapitre d'introduction au contexte du sujet de mémoire. Il présente l'intérêt et la question de recherche faisant l'objet de ce travail.

Le deuxième chapitre fait un état de l'art sur la réutilisation des unités de programmes tout en mettant l'emphase sur les enjeux et les défis de la réutilisation de code.

Le troisième chapitre aborde le CBR et les techniques d'indexation appropriées à la réutilisation des unités de programmes. Il aborde également la représentation de code sous forme de cas.

Enfin le quatrième chapitre présente l'intégration de l'approche par classification à facette dans le système CIAO-SI.

Chapitre 2 : Réutilisation des codes : enjeux et défis

2.1 Historique

Dans cette section, nous établissons un lien entre le codage et les étapes subséquentes (conception, analyse, conditions, architecture, données, qualifications, méthodes d'essai, essais) du processus de développement logiciel. Il ne saurait avoir de programmation de code (fonctions, procédures ou package) sans une modélisation préalable. De plus, force est de constater que la réutilisation de code doit tenir compte de l'envergure ou de l'échelle du projet. Plus le projet est de grande envergure, plus la réutilisation est bénéfique [The technical ressource connexion 2001].

La réutilisation de code n'est pas un concept nouveau. En effet, dès 1969, Mcllroy a effectué plusieurs études sur les composants logiciels de masse [Mcllroy, 1969]. Il stipule que la réutilisation logicielle a été principalement concentrée sur la réutilisation de fonctions afin d'assurer l'uniformité des calculs. Les compagnies étaient favorables à cette orientation dans la mesure où elle assurait qu'un calcul spécifique d'ingénieur à travers tous leurs systèmes permet de retrouver la même valeur.

L'autre aspect important est que le développement des composants à des fins de réutilisation ne date pas des années 1990. Les jalons de cette technologie ont été jetés dans les années 60 en Norvège. En effet, Ole-Johan-Dahl et Kristen Nygaard [Dahl, O-J. and Nygaard, K. 1964] du centre de calcul norvégien ont développé un langage de programmation, Simula-67, pour soutenir la modélisation des simulations discrètes d'événement des processus scientifiques et industriels avec les représentations directes de vrais objets du monde. Simula-67 était un exemple d'importance des principes sains de technologie de la programmation tels que la modélisation de logiciel, l'encapsulation, et la séparation du monde des problèmes en

termes de composants bien définis. Plus tard, ces concepts sont devenus la base de la programmation orientée objet.

Le début de la croissance rapide du marché de technologie de composant (objet) était déjà en 1986 une approche révolutionnaire. Cox [Cox, 2001] a présenté le concept d'un "IC (composants intégrés) de logiciel. Sa vision était que nous devrions pouvoir établir le logiciel à partir de composants préconstruits de la même façon que les ingénieurs de matériel intègrent des morceaux pour construire des ordinateurs. Ce concept est également devenu le concept fondamental de la technologie d'objets pour plusieurs des fournisseurs parce qu'il était l'une des premières explications claires de la valeur marchande de cette technologie. IBM était l'un des plus grands partisans de ce concept et a commencé à migrer certains de ses magasins de développement à la programmation orientée composant.

Le concept de Cox a également conduit beaucoup de grandes sociétés à commencer à investir fortement en recyclant leur personnel et en mettant en application des initiatives de réutilisation des composants. Au début des années 90, beaucoup de sociétés de Wall Street commencent à bâtir des systèmes en utilisant C++ et beaucoup de compagnies de télécommunications ont commencé à établir les systèmes client/server en utilisant le Smalltalk. Un des exemples les plus annoncés était Xerox Corporation.

En 1992 David Taylor [Dard92] a décrit une expérience conduite sur des Systèmes de Données Électroniques (EDS). Une équipe de programmeurs SmallTalk a rénové un système industriel qui a été à l'origine écrit en PL/1. L'original PL/1 des systèmes a consisté en plus de 265,000 lignes de code et a pris 152 mois personnes d'effort durant une période de 19 mois. La version SmallTalk du système a consisté en 22,000 lignes de code et a pris 10 mois personnes d'effort sur une durée de 3.5. Le résultat est impressionnant.

En 1994, Xerox convertit tous ses systèmes de legs à EDS et a décidé de commencer à mettre en application tous les nouveaux systèmes en utilisant la technologie d'objet. D'autres compagnies (Shell Oil, Kash NKarry, la banque suisse, le Sprint, les Frères de Salomon, et la Banque Impériale Canadienne de Commerce) étaient également très favorables à la réutilisation. Malheureusement, la plupart de ces compagnies ne pouvaient pas démontrer clairement la réutilisation répandue. Seulement un petit pourcentage pouvait accomplir quelques niveaux de réutilisation dans un secteur d'activité.

En 1997, le Groupe Gartner a publié un rapport sur la réutilisation orientée objet et a prévu qu'avant 2001 il y aurait 7 milliards de $ "de composants logiciels" sur le marché. À la fin des années 1980 et tout au long des années 1990, beaucoup de grandes sociétés ont investi massivement dans les technologies de réutilisation des composants. La plupart des projets initiés se sont rapidement métamorphosés en échecs. Malgré le fait qu'il n'y a aucun exemple de succès de réutilisation à grande échelle, le marché de technologie continue à promouvoir la réutilisation comme une solution de l'augmentation de la valeur ajoutée en technologie de l'informatique [NKambou et al. 2004]. De nos jours, force est de constater que la réutilisation logicielle dans les entreprises de développement logiciel occupe plus de place qu'on l'aura imaginé.

Parmi les nombreuses limites dans les tentatives de réutilisation des composants, le choix du langage de programmation occupait un aspect important ainsi que le choix de la technologie qui à l'époque était au stade embryonnaire et le non appui au projet de développement par les acteurs impliqués. Puisque ces raisons semblaient être à l'origine de ces échecs, alors les fournisseurs de technologie n'ont ménagé aucun effort pour relever le défi. En fait, IBM, HP, et plusieurs autres fournisseurs ont créé un joint-venture appelé Taligent. Le premier objectif de Taligent était de créer un ensemble standard d'objets réutilisables de C++. La vision de

Taligent de la réussite de la réutilisation des composants est d'adopter simplement une approche orientée objet.

IBM n'a pas donné suite au concept d'avoir un ensemble de cadres d'objets qu'il pourrait fournir en tant que composants réutilisables. Il y a plusieurs années, IBM a créé un projet appelé San Francisco pour créer une infrastructure réutilisable et des objets d'affaires en utilisant Java au lieu de C++. San Francisco a eu très peu de succès commercial minimal.

Une autre raison généralement citée du manque de succès en créant la réutilisation à grande échelle est le manque de normes. En 1993, un consortium de compagnies a créé le groupe de gestion d'objet (OMG) pour définir des normes de technologie d'objet. Les deux normes primaires qui ont évolué de l'OMG sont CORBA (architecture commune de courtier de demande d'objet) et UML (Unified Modeling Language). OMG a eu du succès en réalisant la large adoption de ces normes, cependant, il n'a toujours mené à aucun exemple de large réutilisation ou d'un marché de "composant logiciel".

Les défenseurs de CORBA ont cru qu'ils devraient concevoir des applications et distribuer des composants pour maximiser le potentiel de réutilisation. Le problème avec cette approche est qu'elle a des implications significatives sur l'exécution du système. Plusieurs des solutions de CORBA n'ont pas réalisé leurs conditions d'exécution.

En parallèle à l'effort de normes d'OMG, Microsoft faisait évoluer ses technologies bureautiques vers une approche orientée objet. La première étape de Microsoft dans l'évolution intégrait VisualBasic, C++ visuel, et tous ses outils de développement dans un modèle commun d'architecture appelé COM (modèle composant d'objet). Dans toute la deuxième moitié des années 90, l'architecture a continué à évoluer de COM, à COM+, jusqu'au NET d'aujourd'hui.

Aujourd'hui, les problèmes recensés par certains auteurs en ce qui concerne la réutilisation de codes sont encore d'actualité. Ces auteurs dénotent :

- le manque d'appui de la haute Direction de la compagnie dans la gestion des projets,
- le manque de méthodologies appropriées dans les projets de développement,
- le non établissement d'un ordre de priorité des cadres structurel, organisationnel, managérial et technique dans les sociétés [Mili, 1995],
- La mauvaise représentation des problèmes à résoudre [N.Belkin, 1985]
- le Manque d'outils pour représenter des composantes réutilisables [W.B.Frankes, 1988]
- La difficulté de trouver des composantes semblables [R.P.Diaz, 1988].

2.2 Problématique de la réutilisation des codes

Depuis plusieurs décennies, les mêmes raisons sont évoquées dans la perspective de résoudre les problèmes de réutilisation des codes. Plusieurs chercheurs et praticiens sont d'accord sur le fait que la mise en pratique d'un programme de réutilisation comporte d'énormes difficultés. D'énormes facteurs empêchent de mettre efficacement en pratique un programme de réutilisation. Il s'agit de:

- la non maturité du développement de logiciel, en tant que science ou discipline [Dijkstra, 1989]; [Shaw, 1990],
- la planification inadéquate du développement de logiciel [Brooks, 1975],
- la planification inadéquate du développement de la réutilisation [Woodfield, Embley et Scott, 1987],
- la gestion à court terme de la réutilisation [Gruman, 1988],

- le manque de méthodologie et d'outils de support à la réutilisation [Fischer, 1987].

La réutilisation systématique est difficile et risquée à gérer car dans son processus elle nécessite une cohérence efficace parmi les différents intervenants et ceci dans différents points de vues qui sont : organisationnel, économique, légal, cognitif, technique :

- Du point de vue organisationnel, la réutilisation a besoin d'un support à long terme. Ce support doit regarder la réutilisation des codes comme un investissement amortissable à moyen et à long terme.
- Du point de vue économique, la réutilisation des codes doit prendre en compte la notion de coûts et de bénéfices. Le coût encouru pour l'élaboration des codes réutilisables doit être amorti par l'utilisation de ces codes.
- Du point de vue légal, il faut définir quels sont les droits et les obligations des fournisseurs et des consommateurs des codes réutilisables.
- Du point de vue cognitif, les concepteurs et les utilisateurs ont besoin d'un modèle ou d'une représentation mentale du code réutilisable afin de la concevoir et de l'utiliser convenablement.
- Du point de vue technique, deux approches de réutilisation sont généralement utilisées: l'approche construction de blocs de codes, qui est basée sur le développement du nouveau logiciel à partir des codes existants, et l'approche « génératif » qui est basée sur la réutilisation des processus qui sont encodés dans un processeur tel qu'un compilateur, un générateur de code, etc.

Ces approches posent plusieurs problèmes lors de leur construction et leur utilisation [Horowitz et Munson, 1984].

Dans la section subséquente, nous abordons les intérêts et les bénéfices de la réutilisation des unités de programmes

2.3 Réutilisation des unités de programmes : Définitions et bénéfices

Pour Dubuque [Dubuque et al. 1991]. ''La réutilisation est le processus par lequel, au cours de la réalisation des activités de développement d'un système d'information dans un projet donné, on utilise les composants produits au cours des projets antérieurs ou concurrents''. Ces derniers distinguent deux types de composants logiciels, les composants essentiels à exploitation : les librairies de modules, le code, les procédures et guide d'utilisation, le matériel d'implantation et de transfert. Les composantes issues de la production des logiciels: modèles conceptuels, physiques, fonctionnels, organisationnels et technologiques. Les composants issus des modèles et leur codification : connaissances sur les systèmes d'informations (utilisations, domaine d'application, technologie) et sur le contexte des systèmes d'informations (entreprise, environnement). Contrairement à Dubuque, [Goldberg, 1990] identifie deux niveaux de réutilisation. Le premier niveau concerne le code produit lors de l'implantation, le second, les modèles produits à la conception et les connaissances utilisées pour la génération du logiciel. Traditionnellement, la réutilisation se limite au code uniquement [ho-hau Nguyen, et al. 2000].

Selon [Free-man, F., 1983] et [Krueger, C.W. 1992], ''la réutilisation consiste à utiliser des artefacts logiciels existants lors du développement d'une nouvelle application informatique. Ceci inclut les fragments de code source, ainsi que les structures de conception et d'implantation, les spécifications, la documentation, etc.''.

Plusieurs études [Biggerstaff, T.J. et Perlis, A.J. 1984] sur la réutilisation démontrent que 40% à 60% du code est réutilisable d'une application informatique à une autre. D'après [Lanergan, R.G. et Grasso, C.A. 1984], 60% de la conception et du code des applications d'affaires est réutilisable.

Plusieurs expériences en industrie [Basili, V.R. Rombach H 1994] suggèrent que la réutilisation contribue à accroître la productivité et la qualité. Lim [Lim et

W.C. 1994] suggère que la réutilisation accélère la livraison des produits et un retour sur l'investissement très avantageux.

Il existe plusieurs raisons pour lesquelles la réutilisation est souhaitable. Les deux raisons majeures sont l'augmentation de la productivité et l'amélioration de la qualité des logiciels [Tracz, W. 1987], [NKambou et al. 2004] En général, les systèmes développés à partir de la réutilisation des composants logiciels coûtent moins cher. Des gains de productivité se font dans la réduction des coûts de développement. Deux facteurs permettent d'expliquer ces gains de productivité: un délai de livraison plus court et l'élaboration du code avec moins d'erreurs. Le programmeur écrit moins de code, la réutilisation réduit la quantité de documentation et des jeux de tests et le système est plus facile à maintenir, car les programmeurs sont plus familiers avec les composants réutilisables qui ont servi à concevoir le système.

Pour bénéficier des avantages de la réutilisation de codes, un programme de réutilisation des codes doit tenir compte de plusieurs facteurs [W.B.Frankes, 1988]. Parmi ces facteurs, nous avons énumérés ceux qui semblent avoir un impact direct sur la réutilisation des unités de programmes. Il s'agit des facteurs:

- techniques - actuellement il y a beaucoup d'attention donnée aux facteurs techniques, dans les secteurs comme des environnements de soutien
- managériaux – l'organisation d'un environnement de technologie de la programmation pour favoriser et récompenser la réutilisation des codes. Un exemple d'une variable gestionnaire pourrait être une incitation économique des employés à créer des codes réutilisables, exprimées en dollars
- économiques - ceci traite notre capacité de découvrir si la réutilisation des codes payera au loin financièrement. Il s'agit de mettre l'emphase sur la viabilité économique pour la réutilisation des codes Une variable d'exemple pourrait être le niveau de paiement pour des créateurs des unités des codes

- légaux – ce sont les droits des créateurs et des consommateurs du logiciel réutilisable Une variable ici pourrait être si une loi limitant la responsabilité de fournisseur des codes est en place

Pour d'autres chercheurs, l'avantage concurrentiel est lié aux facteurs d'ordre organisationnel [Milli, 1999]. Il aborde différentes issues dont :

- Economiques : les issues économiques de la réutilisation des codes. Ils ont proposé plusieurs modèles de coût ayant pour l'évaluation, la prévision, l'analyse et les dépenses de réutilisation de logiciel [Mili 00]. Les caractéristiques des modèles proposés sont le cycle d'investissement, la fonction économique, les coûts, l'organisation de la réutilisation, la portée et les hypothèses.
- Organisationnelles : plusieurs publications obtiennent la conclusion de leur recherche de plusieurs facteurs d'organisation qui affectent le succès et l'échec de réutilisation, comme l'engagement de gestion qui est la chose préalable de succès [Morisio 02] les facteurs humains qui doivent être considérés pour supporter le changement de processus.
- Culturelles : Les questions culturelles ont effets uniques ou mal compris sur la réutilisation de logiciel [Dard, 94]. Il s'agit de la formation qui est souvent laissée de côté, la motivation ou l'engagement actif, en termes de la production et de la consommation.
- Techniques : Il s'agit de la récupération qui suppose qu'un utilisateur soumet une question bien définie à la bibliothèque de logiciel et s'attend à ce que le système identifie les actifs de bibliothèque qui satisfont la question et excluent des actifs sans rapport [Mili 99].

Nous avons présenté la réutilisation de code (historique, définition et bénéfices). Comme suggéré par Peter [Peter 2004] une des solutions appropriées à la

mise en œuvre de la réutilisation de codes est le CBR. Dans le prochain chapitre nous décrirons le CBR et les techniques d'indexations des unités de programmes.

Chapitre 3 : CBR et les techniques d'indexation des codes

3.1 Le CBR

Plusieurs études ont démontrées que le CBR est la méthodologie la mieux indiquée pour la réutilisation en générale [Sankar K. Pal and Simon C. K. Shiu, 2004] ; [C. Tautz, K-D Althoff, 1997] ; [Peter, 2004]. Dans le cadre de notre travail, nous avons opté pour cette méthode que nous allons adapter à notre contexte. Le raisonnement à base de cas (CBR) est une approche de résolution de problèmes qui utilise des expériences passées pour résoudre de nouveaux problèmes [Leake D. B 96]. L'ensemble des expériences forme une base de cas. Typiquement un cas contient au moins deux parties: une description de situation représentant un "problème" et une "solution" utilisée pour remédier à cette situation. Parfois, le cas décrit également les conséquences résultant de l'application de la solution (e.g. succès ou échec). Les techniques CBR permettent de produire de nouvelles solutions en extrapolant sur les situations similaires au problème à résoudre. Cette approche est adéquate pour les domaines où la similarité entre les descriptions de problèmes nous donne une indication de l'utilité des solutions antérieures. Les fondements du CBR proviennent de travaux en science cognitive menés par Roger Schank et son équipe de recherche durant les années 80 [Riesbeck C., Shank R 89] Leurs travaux ont mené à la théorie de la mémoire dynamique selon laquelle les processus cognitifs de compréhension, de mémorisation et d'apprentissage utilisent une même structure de mémoire. Cette structure, les "memory organization packets" (MOP), est représentée à l'aide de schémas de représentation de connaissance tels que des graphes conceptuels et des scripts [Luc Lamontagne – Guy Lapalme, 2002]. Watson définit le CBR comme une méthodologie de résolution de problème basée sur les connaissances contenues dans les cas [Watson, 2002] ou une approche de résolution de problèmes proche du raisonnement humain [Rabea, 2001]. Un cas (unité de programme) symbolise une

connaissance dont l'enseignement peut être utile lorsqu'un nouveau problème se présente [Dussaux, 2001]. Dans la suite nous envisageons la manière dont CIAO-SI peut s'adapter dans un contexte de réutilisation de code. Pour cela nous avons effectué les actions suivantes pour accomplir notre travail :

- Comprendre l'usage du CBR dans CIAO-SI pour fin de réutilisation
- Définir la structure de la base de cas dans le contexte du code
- Définir l'indexation et le facteur de mesure de similarité pour la recherche d'unité de programmes pertinents
- Intégrer le moteur CIAO-SI pour la mise en œuvre
- Tester et évaluer

Le cycle de vie du raisonnement à base de cas proposé par Jaczynski (figure 1) est un cadre approprié pour l'indexation et la récupération des unités de programmes (codes). Ce cycle inclut quatre processus essentiels : la recherche, la réutilisation, la révision et l'apprentissage.

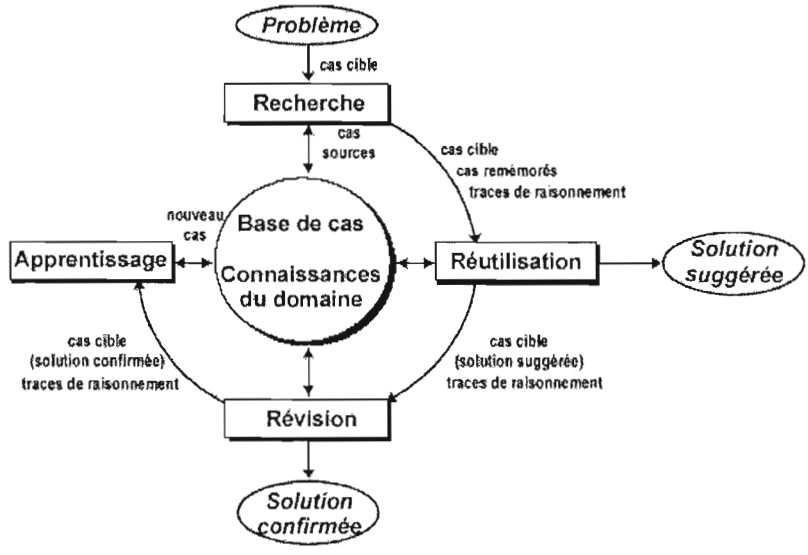

Figure 1 : Cycle de raisonnement à base de cas (Jaczynski, 1998)

La recherche: cette phase permet de déterminer les cas les plus proches du problème à résoudre. La procédure de recherche est habituellement implantée par une sélection des plus proches voisins ("k-nearestneighbors") ou par la construction d'une structure de partitionnement obtenue par induction. L'approche des plus proches voisins utilise des métriques de similarité établie pour mesurer la correspondance entre chaque cas et le nouveau problème à résoudre. L'approche par induction génère un arbre qui réparti les cas selon différents attributs et qui permet de guider le processus de recherche.

La Réutilisation: suite à la sélection de cas lors de la phase de recherche, le système CBR aide l'usager à modifier et à réutiliser les solutions de ces cas pour résoudre son problème courant. En général, on retrouve deux approches pour la réutilisation de cas. Par réutilisation structurelle, on obtient une nouvelle solution en modifiant des solutions antérieures et en les réorientant afin de satisfaire le nouveau problème. Par réutilisation dérivationnelle, on garde, pour chaque cas passé, une trace des étapes qui ont permis de générer la solution. Pour un nouveau problème, une nouvelle solution est générée en appliquant l'une de ces suites d'étapes.

La révision : permet d'affiner la solution grâce à l'évaluation de l'expert. Peu de systèmes CBR font de la révision (adaptation) complètement automatique. Pour la plupart des systèmes, une intervention humaine est nécessaire pour générer partiellement ou complètement une solution à partir d'exemples. Le degré d'intervention humaine dépend des bénéfices en termes de qualité de solution que peut apporter l'automatisation de la phase d'adaptation

L'apprentissage : permet de mettre a jour les éléments du raisonnement en prenant en compte l'expérience qui vient d'être réalisée. La base de cas est éventuellement enrichie du nouveau cas.

Dans le cadre de ce travail, nous nous limitons aux phases de recherche et de réutilisation des unités de programmes. Cependant nous présentons un aperçu des phases de révision et d'apprentissage.

CIAO-SI a été conçu pour supporter les projets de développement selon la méthodologie RUP (Rationnal Unifyed Process). Un projet dans ce contexte est vu sur sept phases comme le montre la figure 2 :

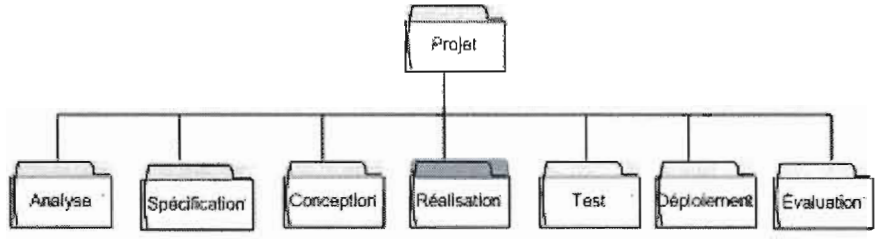

Figure 2 : Phases de la méthodologie RUP

Dans le cadre de ce mémoire, notre travail se limite à la gestion des artefacts découlant de l'étape de réalisation du projet.

3.2 L'indexation dans le CBR

3.2.1 Notion d'index

L'indexation est l'assignation d'une étiquette à un article [Prieto-Díaz 1987]. L'article se définit comme un cas. Ici le cas étant une unité de programme (code). Cette étiquette doit préciser les circonstances dans lesquelles le cas a été appris (enregistré) et la situation dans la laquelle il est préférable de le réutiliser. De ce point de vue, l'indexation se définit comme l'interprétation d'une situation dont le seul but est de déterminer l'étiquette du cas qui convient le mieux à cette situation. L'implantation d'un index nécessite qu'il soit efficace, abstrait, concret et évolutif.

Dans la riche revue de littérature, trois (3) grandes catégories d'indexation se démarquent. Il s'agit de:

- **l'indexation formelle :** elle insiste sur l'exactitude et la conformité de la solution du cas à utiliser. Cette approche se base sur le fait que la correspondance entre le cas cible stocké et le cas source ne peut se réaliser de façon efficace et sans ambiguïté que si le cas est représenté en utilisant un symbolisme mathématique stricte.
- **l'indexation par mots clés** : A la différence de l'indexation formelle, l'indexation par mots clés utilise une représentation textuelle pour la description du cas. Les cas sont indexés et classés dans la base de cas suivant les mots clés contenus dans leur description. Pour effectuer une recherche, un analyseur de texte extrait les mots clés contenus dans la description du problème faite par l'utilisateur en langage naturel et ces mots sont utilisés pour localiser les cas dont la description contient ces mots.
- **l'indexation par ontologique** : Elle est actuellement la plus utilisée et la plus prometteuse pour l'indexation des cas [Fernández et al. 1995]. Au lieu de baser la similarité entre cas sur une correspondance syntaxique, l'approche ontologique met en exergue la correspondance sémantique entre les différents concepts sélectionnés par l'utilisateur et ceux contenus dans la description du cas. Cette ontologie doit définir les concepts importants du domaine et les relations entre ces concepts. Les concepts issus de l'ontologie sont utilisés pour indexer les différents cas stockés. Plusieurs applications à base de cas tirent profit de cette technique pour l'indexation des cas. Parmi ces applications, on peut citer Rebuilder qui utilise l'ontologie WordNet pour l'indexation des diagrammes de classes stockés en cas [Gomes et al. 2002a ; Gomes et al. 2002]. Reboot et ROSA utilisent une classification à facettes introduite par Priezto-Diaz pour l'indexation de composants logiciels [Sindre et Kalson, 1993] ; [Bjørnar et al. 1994] ; [Prieto-Díaz.et Freeman, 1987].

Parmi les nombreuses techniques de la réutilisation de codes et de composants logiciels recensées dans la littérature [Kruger 92], [Petro-Diaz, 87], [Arango, 98], [Devanbu, 1989], [Rosales, 1988], [Frakes. 1988], [Sindre et Kalson, 1993], [Donfack 2004] notre choix s'est porté sur la technique de la classification facettée de Petro-Diaz.

3.2.2 La classification facettée

La notion de facette est définie comme suit: « *Any component of a compound subject, or the set of classes produced by one characteristic of division; manifestation of a fundamental category in a particular class, e.g. crop facet in agriculture is manifestation of fundamental category personality, or entity* » [Langridge 1992, 74]. Exprimé très simplement, une facette est une catégorie. Taylor en parle en ces termes: «*set of clearly defined, mutually exclusive, and collectively exhaustive aspects, properties, or characteristics of a class or specific subject*» [Maple 1995]. On notera que le terme Facette est utilisé à la fois dans le contexte de la structure d'un plan de classification et dans celui de la représentation du sujet d'un document [Langridge1992].

S.R. Ranganathan, bibliothécaire indien de la première moitié du 20e siècle, fut le premier à introduire la notion de facette dans le domaine de la gestion documentaire. On lui doit également l'établissement des bases théoriques de l'analyse par facettes [Maple, 1995]. Selon lui la classification à facettes est une classification non énumérative qui présente les sujets symbolisés regroupés en facettes selon certaines catégories fondées sur un critère précis de division. Les classifications à facettes ne sont pas des classifications hiérarchiques au sein desquelles on trouverait, de déduction en déduction, les symboles des sujets à indexer. Au contraire, chaque facette de termes symbolisés (ou foci) fournit des éléments isolés (isolates), obtenus à partir d'une caractéristique commune de division et appartenant à une catégorie précise, que le bibliothécaire doit assembler selon les instructions qui lui sont données par les indicateurs de facettes : le système à facettes permet donc, en théorie, la

combinaison illimitée de termes ressortissant à un domaine déterminé [Canonne 1993, 48].

La classification à facettes est une méthode de classification d'objets à laquelle s'est beaucoup intéressé [Prieto-Díaz 88, 87]. Cette méthode est généraliste et peut a priori être appliquée à toute forme d'objets, la seule condition requise étant de pouvoir en fournir une description cohérente permettant de différencier un objet d'un autre.

Une classification à facettes place les objets de la collection dans un espace à n dimensions appelées facettes. Chaque facette est un ensemble fini de termes. Ainsi, un élément de la collection peut être (au moins partiellement) caractérisé par le n-uplet de termes représentant ses coordonnées. On pourra noter que pour obtenir un système de classification effectif, la recherche des facettes, et des termes des composants doit être faite par un expert à partir d'une partie représentative de la collection. Une facette peut donc être considérée comme un point de vue de la collection d'objets qu'elle permet de classifier. Le tableau ci-dessous, tiré de [Petro Diaz, 1991], présente un exemple de classification à facettes (simplifié) pour les composants d'UNIX. L'ensemble de ces utilitaires est scindé en quatre vues : la fonction réalisée par le composant (by action), les objets manipulés par la fonction (by object), la structure de données sur laquelle opèrent la fonction (by data structure) et le système auquel appartient la fonction (by system).

Classification à facette des composants d'UNIX				
Facette	{by action}	{by object}	{by data structure}	{by system}
Termes	get put update append check detect locate search evaluate compare make build start	file-names identifies line-numbers character number expression entry declaration line pattern	buffer tree table file archive	line-editor text-formatter

Tab. 1 Classification à facette des composants d'UNIX

Dans le cas particulier d'une collection de fonctions, il est intéressant de classifier les fonctions suivant leur type. Le type des fonctions agit comme une facette dotée d'une sémantique. On obtient alors une classification à facettes comportant une seule facette qui présente une infinité de termes (les types).

Ces classifications vont permettre de retrouver aisément des documents et des portions de code déjà existantes, simplifiant ainsi le développement d'une application. Elles vont de plus autoriser un rangement des nouveaux documents et composants logiciels au fur et à mesure sans avoir à réorganiser la classification de départ. En enfin, elles peuvent aboutir à trier les bibliothèques puisqu'il sera possible de déterminer statistiquement quels sont les documents et composants les plus utilisés et quels sont ceux qui ne le sont que rarement, voire jamais.

[Prieto-Díaz 87, 88,91] argumente en faveur des classifications à facettes. Il pense en effet que celles-ci sont plus adaptées aux applications mentionnées ci dessus que les classifications habituelles. Ses expérimentations montrent que le principe des facettes évite non seulement de mal ranger un objet d'une collection, mais encore qu'il permet de retrouver ces objets de façon beaucoup plus aisée.

Le modèle de classification à facettes pose lui aussi des problèmes de cohérence. Il ne s'agit pas à proprement parler d'un problème de cohérence entre les différentes vues, mais plutôt de savoir si les différentes facettes sont bien indépendantes et si la classification qui en découle est complète. Ainsi, il faut toujours être capable d'insérer un objet dans la bibliothèque ; autrement dit, cet objet doit pouvoir se glisser dans les vues déjà existantes. Prieto-Díaz résout ce problème en confiant la création des facettes et des termes à un expert [Benjamin Sigonneau, 2003], celui ci se basant sur un échantillon représentatif de la bibliothèque pour mener son travail à bien.

La classification à facette pose les bases d'une approche par ontologie en ce sens que les termes utilisés sont complétés de leur explication détaillée. Ces

explications sont connues d'avance par les utilisateurs ce qui permet leur prise en compte tant dans la description de codes que lors de la recherche. Dans cette approche, seul l'aspect fonctionnel du cas est pris en compte. Le système résultant s'avère donc approprié pour la réutilisation du code et des connaissances de domaines.

Recherche de code existant. Il apparaît intuitivement qu'une bonne classification permet de répondre en grande partie au problème de la réutilisation. Une bonne classification permet en effet de trouver les composants recherchés s'ils existent, des composants similaires (c'est-à-dire des composants proches de celui recherché mais à adapter) sinon. Prieto-Díaz considère que les classifications classiques ne sont pas adaptées à une telle tâche ; une classification à facettes lui semble bien mieux remplir les critères d'une bonne classification.

La réutilisation des codes ou des composants logiciels est d'un intérêt majeur pour le développement de logiciel. Elle présente en effet de nombreux atouts, parmi lesquels on peut citer un temps et des coûts de développement réduits ou encore une meilleure fiabilité des programmes développés. Cependant, rendre la réutilisabilité effective pose des problèmes. Le problème de la réutilisation de code existant peut se découper en deux sous problèmes. Le premier est celui de l'existence ou non de la fonctionnalité que l'on souhaite (ré) utiliser ; quand bien même le code correspondant à cette fonctionnalité existerait déjà, il reste cependant à le retrouver. Le second problème est celui de l'intégration de cette fonctionnalité au sein de l'application en cours de développement. Garlan, Allen et Ockerbloom [David Garlan et al. 95] expriment la difficulté de cette tâche.

Nous avons présenté le CBR, les techniques d'indexations recensées dans la revue de littérature. Nous avons décrit notre choix (la classification facettée) comme étant la technique la mieux adaptée pour la recherche et l'indexation des unités de programmes [Petro-Diaz1998]. Dans le prochain chapitre nous représentons le code

en termes de cas et nous décrirons l'utilisation des facettes pour l'indexation des unités de programmes.

3.3 Représentation de codes en termes de cas.

3.3.1 Utilisation des facettes pour l'indexation des codes

Les facettes ci-dessous symbolisent les connaissances de domaines pour lesquels la recherche des unités de programme s'effectue. Il s'agit de:

Domaine d'application : le domaine d'application est l'ensemble des connaissances associées à un projet de développement d'application spécifique. Dans le cadre de cette recherche, le domaine d'application constitue l'ensemble des applications informatiques développées par Le Groupe Infotel, pendant les 15 dernières années. Ces applications existes dans différentes versions (Tableau 4 : Tableau d'évaluation expérimentale). L'utilisation du domaine d'application est primordiale, sinon essentielle dans la recherche et l'indexation des unités de programme. Nous savons bien qu'un programmeur n'a pas toujours la facilité de retrouver les codes ou les portions d'unités de programme dans les différents systèmes ou moteurs de recherche. Généralement le nombre de réponses fourni par les moteurs de recherche exige du programmeur de faire des choix judicieux afin que le résultat sélectionné soit relativement conforme à sa requête. Nous avons projeté d'implanter notre système de réutilisation des codes de telle sorte que le système offre au programmeur une liste de domaines d'applications existants. S'il choisi ''GRH'' (domaine d'application de gestion des ressources humaines), alors toute sa recherche est focalisée sur toutes les unités de programmes relatives à la gestion des ressources humaines. Si par contre il choisi comme facette le domaine d'application de la PAIE, alors la recherche et l'indexation seront focalisées sur ce domaine.

Langage de programmation des codes : Dans le cadre de cette recherche, nous nous sommes intéressés de prime à bord à tous les langages de programmations

utilisés dans la production des codes informatiques d'application à la gestion. Il s'agit de Java Beans, du Cobol, de C++, du PL/Sql, du fortran, etc. Un programmeur qui recherche un code ou une portion de code, n'a pas toujours le choix de retrouver les unités de programmes souhaitées avec le langage de programmation qu'il maîtrise. Il ressort alors la nécessité d'implémenter une facette ''langage de programmation des codes'' qui permet d'orienter le programmeur dans son choix. Les valeurs de ces facettes sont diverses, de basic a C++. Cependant la liste n'est pas exhaustive, à tout moment le système peut être enrichi.

Outil de création (génération/modification) des codes : Les outils utilisés pour la création (génération, modification) des codes informatiques sont assez diversifiés [Susan Elliott Sim Charles L.A. Clarke Richard C. Holt 98]. Pour guider le programmeur dans le choix des codes recherchés, il est important de lui donner la possibilité de sélectionner l'outil qui a servi à créer, modifier ou à générer le code trouvé.

Type de programme : Pour rechercher un code informatique, le programmeur doit être en mesure de sélectionner une fonction(F) qui lui retourne une valeur, une procédure pour un traitement spécifique (P) ou un package, ensemble de code informatique intégré (Pz) .

Concept de programmation : nous avons projeté d'établir trois niveaux de concepts de programmation des codes. Dépendamment du niveau sélectionné (1 : Programmeur débutant ; 2 : Programmeur moyen ; 3 : Programmeur avancé), le programmeur peut facilement choisir le code de programme à comprendre et à adapter dans son contexte de travail.

Production de la documentation : Une portion de code non documenté n'est pas lisible et vraisemblablement ne répond pas au code d'éthique de la programmation. Lorsque le programmeur recherche une unité de programme pour implémentation ou modification dans le but de sa réutilisation. Il s'assure que le

résultat de sa recherche lui fournit une unité de programme documentée, lisible et compréhensible.

En se focalisant sur l'ontologie du domaine d'application du projet CIAO-SI, et l'approche de réutilisation des unités de programme basée sur la classification à facette, nous représentons la structure initiale d'un cas dans la figure 3

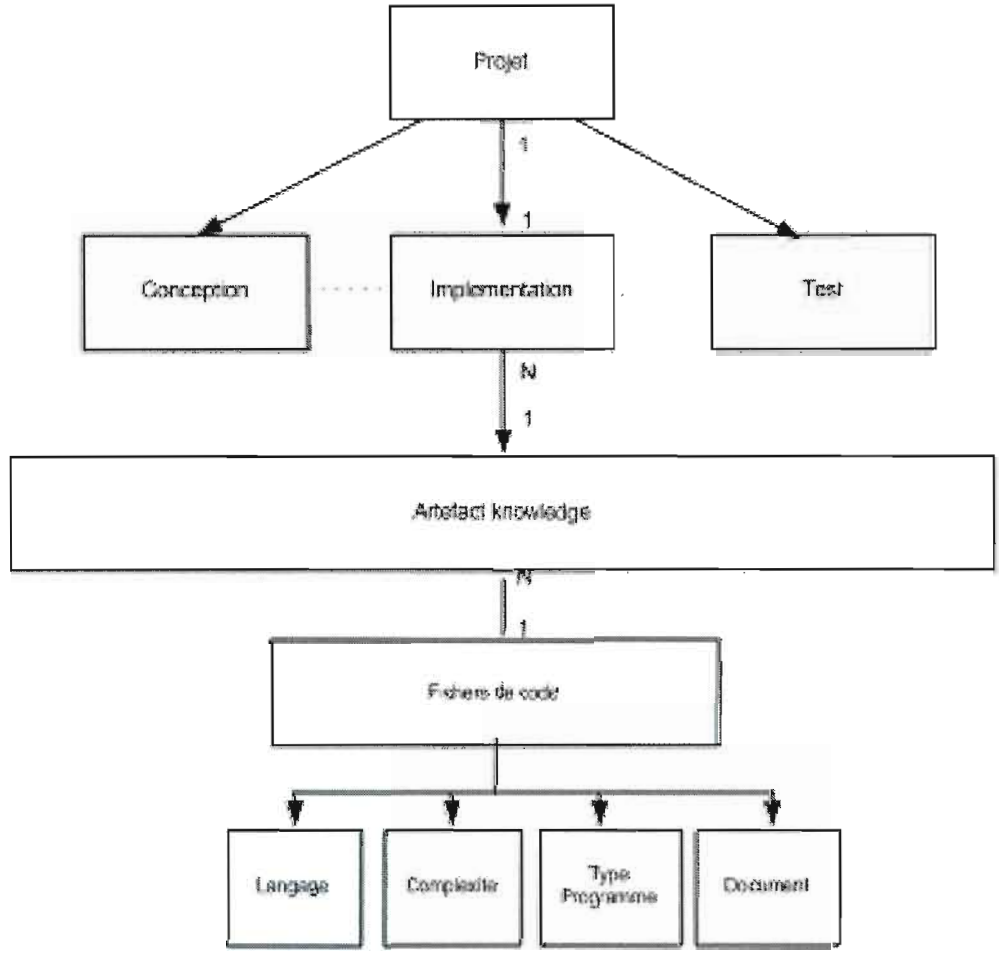

Figure 3: Structure initiale d'un cas

La structure proposée regroupe l'ensemble des connaissances associées aux unités de programme représentées sous le nom de fichier de code. Le fichier de code utilise les connaissances issues des phases conception, et implémentation. La phase de test sert de vérification de la phase précédente.

Les facettes du fichier de code pour la recherche et l'indexation sont : le domaine d'application, le langage de programmation, le concept de programmation, le type de programme (package, fonction, et procédure) la production de la documentation (programme documenté ou non), les outils de création, de génération ou d'adaptation. La structure du cas défini (unité de programme) représentée sous forme de modèle de classe regroupe toutes les classes et les connaissances de domaines de codes.

La figure 4 représente la structure du cas.

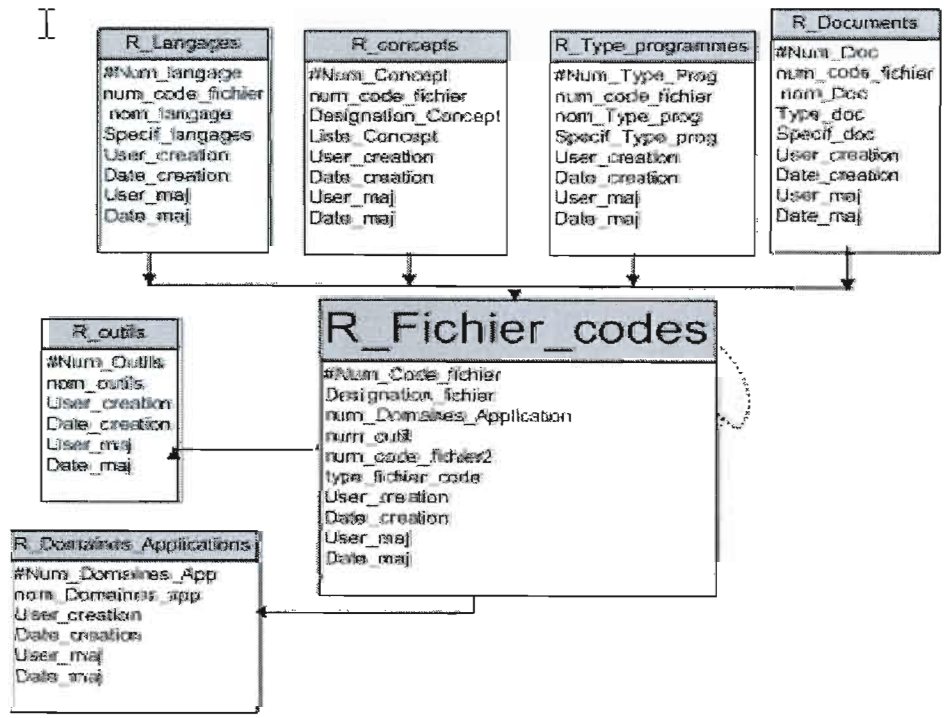

Figure 4 : Structure de cas (unité de programme)

Nous avons décrit, dans cette section, les facettes qui composent la structure de cas. Dans ce que suit nous avons, à travers les exemples, les différentes possibilités que nous offrons dans le choix de code pour la construction d'une application informatique.

3.3.2 Recherche et d'indexation de code informatique

Supposons qu'un programmeur veut mettre en place une application de gestion des ressources humaines pour une société de production du lait. Il a en sa possession toute la documentation produite par les phases subséquentes de développement d'un projet informatique (analyse des besoins, conception, modélisation).Le programmeur se sert de ses cas d'utilisation pour produire un module applicatif constitué d'une ou de plusieurs fonctions, ou procédure ou package.

En tout premier lieu, le programmeur s'enregistre dans l'outil comme développeur. Dans ce cas, il a la possibilité d'avoir accès à tous les domaines d'application existants dans la base des cas ainsi que tous les outils de développement et de modélisation.

Ensuite le programmeur choisit dans la liste des domaines d'applications, la gestion des ressources humaines. Il est alors dirigé vers une fenêtre lui montrant la liste des domaines d'application existants (GRH, PAIE, IMMO, COMPTA, GCO, BAT, STO, UtIL, ect.). Ces caractéristiques sélectionnées, seront utilisées comme critères de recherche et de récupération du code.

Le tableau ci-dessous correspond à l'exemple de recherche et d'indexation de codes informatiques.

Critères de recherche et de récupération des codes	
Facettes	**Exemple de valeur des facettes**

Domaine d'application	GRH
Langage de programmation	C++
Outil de génération/création/modification du code	Case Oracle
Type de programme (fonction, procédure, package)	Procédure
Concept de programmation (débutant, moyen, avancé)	moyen
Programme documenté (Description, commentaires, jeux de test)	Oui

Tableau 1 : Critères de recherche

A chaque critère choisi, un poids est affecté indiquant l'importance du critère pour le programmeur. Apres la sélection et l'instanciation des critères de récupération, le système affiche les unités de programmes (cas) retrouvées et proches des critères choisis.

Pour chaque cas (unité de programme) pré choisi, une valeur de (1% à 100%) lui est attribuée correspondant au taux de document (défini) retourné répondant aux critères de sélection.

Ainsi le programmeur a le choix de télécharger le cas (l'unité de programme) approprié et répondant à ces critères de recherche ''le cas trouvé''. Le programmeur peut alors modifier le code trouvé conformément aux spécifications de la nouvelle application. Mais le stockage du cas modifié nécessite l'avis d'un expert.

3.3.3 Description du Cycle CBR pour la réutilisation des codes

La forme la plus évidente de réutilisation de logiciel est la réutilisation de code qui consiste à copier des morceaux de code des projets précédents et modifiés

pour utiliser au besoin courant [Peter, 2004]. Les développeurs de logiciels qui essayent de réutiliser des travaux précédents ont souvent des problèmes de localisation de fragment de code approprié. Ce comportement se produit dans toute la communauté de logiciel et est également une des aspirations dans la réutilisation du code. Le raisonnement à base de cas (CBR) est une technique d'intelligence artificielle permettant de stocker des expériences précédentes et de les réutiliser pour résoudre de nouveaux problèmes. Le CBR est le candidat parfait pour créer un outil pour la réutilisation de code [Peter, 2004]. Les réalisateurs peuvent stocker les fragments de code utilisés précédemment et les rechercher une fois requis. Ceci sauvera non seulement le temps de recherche mais aussi aide également des développeurs à se concentrer sur l'essentiel de leur travail.

Parmi les outils jusque là recensés, Déjà vu [Smyth, B. and P. Cunningham, 92] est un outil qui profite de CBR hiérarchique pour la réutilisation et la génération de code. Il met l'accent sur la décomposition de problème plutôt que la structuration de solutions (le programme). Cette façon de faire rend difficile la détermination des décompositions appropriées de l'espace de problème dont le résultat est moins conforme à celui souhaité.

Un autre outil de réutilisation de code le KBRAS et CÉSAR [K. Zeroual, P.N. Robillard, 92] [NKambou et al. 2003], s'adresse à des phases particulières du processus de technologie de la programmation. La réutilisation de soutien de KBRAS des composants de logiciel en classifiant et en recherchant des conditions pour d'autres applications de logiciel. CÉSAR est un système de raisonnement à base de cas basé pour construire des programmes, dans un domaine spécifique, des composants existants de programme. CÉSAR aide des utilisateurs construisant une première ébauche de nouveau programme des spécifications données, en utilisant des programmes de bibliothèque de logiciel existant. La contrainte principale pour ces outils est que l'utilisateur doit être un expert en matière de domaine pour comprendre comment les programmes de bibliothèque doivent être décomposés en termes de leurs

caractéristiques fonctionnelles. Notre choix s'est porté sur l'approche CBR. Etant donné qu'il est la méthodologie la mieux indiquée pour la réutilisation des unités de programmes [Peter, 2004]

Recherche et la récupération des codes, la classification à facette introduite par Petro Diaz est bien appropriée pour la recherche des unités de programmes [Petro Diaz 98]. Il s'agit dans un premier temps de définir les facettes de recherche du cas (Domaine d'application, outils de création/génération et d'adaptation du code, langage de programmation, degré de complexité, type de programme (package, fonction, procédure) et le document produit (code documenté ou pas)). Cette technique introduite Cet outil fournit un support dynamique pour aider le développeur dans le choix de la récupération du cas stocké dans la base de connaissances.

Nous avons présenté le CBR et l'approche classification facettée comme la technique appropriée à la réutilisation du code. Nous avons donné les détails de cette technique dans le chapitre actuel. Dans le prochain chapitre, nous proposons une implémentation et une intégration d'un support de réutilisation de codes dans le cadre de CIAO-SI basée sur la classification à facette.

Chapitre 4 : Intégration de Réutilisation de codes à CIAO-SI

4.1 Présentation des cas d'utilisation

4.1.1 Cas d'utilisation d'exploitation

La figure5 présente le cas d'utilisation d'exploitation qui montre les besoins fonctionnels des utilisateurs en matière de réutilisation des unités de programmes.

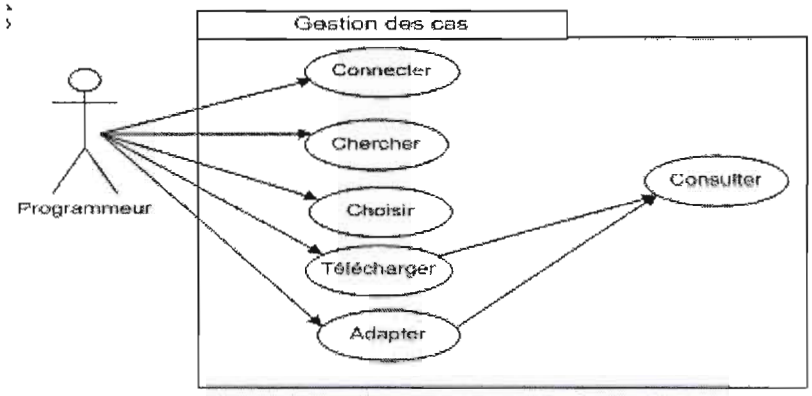

Figure 5: Cas d'utilisation d'exploitation

Le programmeur est le seul utilisateur de ce cas d'utilisation. A la suite de l'exécution de la séquence de « cas d'utilisation d'exploitation », il a le choix entre bâtir l'unité de programme à partir des cas existants ou non. S'il opte pour la construction sur la base d'un cas existant, la fonction de recherche est automatiquement activée. Cette fonction lui permet de retrouver dans la base de cas, un ou plusieurs cas proche(s) du domaine d'application. Basée sur la représentation du problème sous forme d'indicateur, la recherche classifie les cas contenus dans la base de cas suivant les valeurs des indicateurs contenues dans la description du problème. Un calcul de similarité entre les cas sources sélectionnés et le cas cible est

ensuite effectué. Ce calcul permet à l'utilisateur d'apprécier la distance entre l'unité de programme à développer (cas cible) et l'unité de programme contenue dans le cas (cas source). A la fin de la recherche, un ou plusieurs cas pertinents sont sélectionnés par l'utilisateur en fonction de leur degré de similarité et des choix effectués par l'utilisateur en fonction de la description des cas. Les cas ainsi sélectionnés sont téléchargés par l'utilisateur pour fin de consultation, de modification dans l'environnement de développement souhaité. L'utilisateur pourra ainsi bénéficier de toute l'expérience contenue dans ces cas pour adapter son cas. Afin de permettre à l'utilisateur de fournir une meilleure description de problème, il peut se servir à tout instant du cas d'utilisation « consulter cas » pour avoir les informations sur les différents indices entrant dans la description du problème. L'opération de recherche ainsi décrite constitue un élément fondamental pour le système CIAO-SI, au-delà de la réalisation de la première étape du cycle CBR, elle permet d'établir une relation entre l'unité de programme en cours de développement (problème) et les unités de programmes contenus dans la base sous forme de cas (expérience).

Cas d'utilisation « chercher/choisir un cas »

Le cas d'utilisation « chercher une unité de programme » permet de soumettre le système dans une phase de recherche des codes dépendamment des critères de recherche. Les critères de recherche représentent les facettes des codes informatiques [Petro Diaz 98]. Il s'agit du domaine d'application, du langage qui sert à créer le code, de l'outil de génération du code, etc. Selon que le code recherché est stocké dans la base de cas, le système retourne tous les cas possibles. A chaque cas est attribué une valeur de pertinence ou pourcentage.

Cas d'utilisation « télécharger/adapter un cas »

Ce cas d'utilisation permet de visualiser la liste des cas disponibles dans le système. Il permet aussi de télécharger le cas du programmeur le plus proche de son

champ de recherche. La liste des fonctions, leur description détaillée et les pourcentages attribués à chaque fonction retournée lors de la recherche guident le programmeur dans son choix. En outre il permet de mettre à jour soit les informations sur un cas ou simplement d'adapter le code téléchargé. Aucun outil ou langage n'est cependant à ce niveau exigé au programmeur pour la modification du code

Le tableau ci-dessous donne un aperçu des scénarios d'exécution des différents cas d'utilisation dans le domaine de la réutilisation des unités de programmes par le programmeur.

Définition des cas d'utilisation du système- Programmeur	
Connecter	• Saisir le nom de connexion, • Saisir le mot de passe, • Cliquer sur le bouton de connexion
Chercher	• Sélectionner le domaine d'application (exemple : Gestion des ressources humaines, gestion de la paie, etc.), • Sélectionner l'environnement de création, de génération du code (C++ Builder, Oracle Case, SysBase, etc.) • Sélectionner le langage de programmation du code (Java, C++, PL/SQL, etc.) • Sélectionner le type de programme (fonctions, procédures, package), • Sélectionner le concept de programmation (débutant, moyen, avancé), • Sélectionner l'option documentation (Oui pour un code documenté et non pour un code non documenté) • Saisir les mots clés de recherche • Cliquer sur le bouton ''Rechercher'' pour lancer le processus de recherche des unités de programmes répondant aux critères de recherche, • Cliquer sur le bouton ''Annuler'' pour mettre fin au processus de recherche
Choisir	• Affichage par le système des cas les plus proches du domaine de la recherche, • Affichage par le système de la description détaillée de toutes les unités de programmes retournées au cours de la recherche • Sélectionner le cas le plus proche de sa recherche

Télécharger	• Télécharger le cas sélectionné, • Enregistrer le cas sélectionné dans un fichier,
Adapter	• Modifier le cas téléchargé • Adapter le cas téléchargé au contexte • Enregistrer le cas adapté dans un fichier

Tableau 2: Cas d'utilisation du système -programmeur

4.1.2 Cas d'utilisation de stockage

Ce cas d'utilisation est réservé à la seule approbation de l'expert du domaine des applications. A la suite de l'exécution de la séquence de « Stockage de cas », l'expert est tenue a consulter tous les cas modifiés et adaptés au contexte du programmeur. Il passe en revue les cas, les évalue et juge de la pertinence de ces cas à répondre aux exigences établies dans la description des cas susceptibles d'être considérés comme des nouveaux cas. Dépendamment de son jugement les cas modifiés sont considérés soit comme des «nouveaux cas» et seront stockés dans la base des cas, soit s'ils ne répondent pas aux critères d'exigibilité de l'expert, ils seront simplement ignorés.

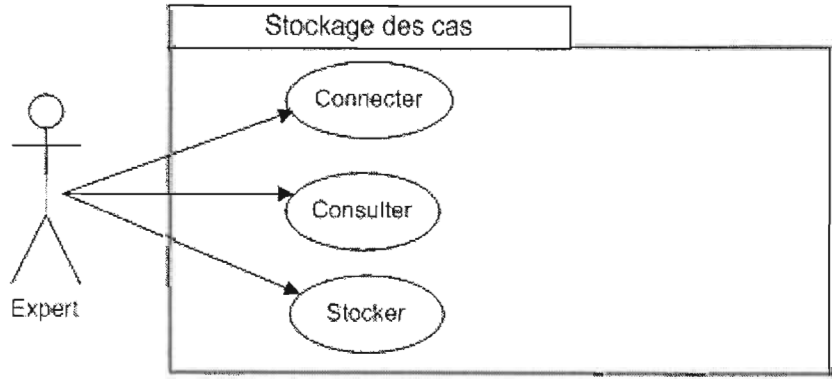

Figure 6: Stockage du cas par l'expert

Définition des cas d'utilisation du système- Expert	
Connecter	• Saisir le nom de connexion, • Saisir le mot de passe, • Cliquer sur le bouton de connexion
Consulter	• Afficher les nouveaux cas (modifier, créer, adapter) • Evaluer les nouveaux cas
Stocker	• Lecture du cas adapté, • Approuver le cas adapté, • Stocker le nouveau cas dans la base de connaissance

Tableau 3: Cas d'utilisation du système -Expert

Cas d'utilisation « Stocker un cas »

Ce cas d'utilisation permet de stocker dans la base de connaissance le nouveau cas (cas adapté). Le stockage d'un cas nécessite l'aval de l'expert du domaine. En d'autres mots c'est l'expert du système CIAO-SI chargé des codes informatiques qui juge lequel des codes modifiés ou crées sera stocké.

4.1.3 Implémentation

Le module de gestion de la base de cas comprend un gestionnaire de critères, un gestionnaire de cas et un module de recherche. Le gestionnaire de critère permet de définir et de mettre à jour les informations sur les indices qui sont utilisés au cours de la recherche. Cette fonctionnalité est très importante, vu le rôle que jouent les indices pour la recherche des cas similaires. La gestion de cas propose des fonctions pour créer, adapter, sauvegarder et consulter les informations sur les cas. Les fonctions offertes par ce module permettent de réaliser les trois dernières étapes du cycle CBR. Le sous-module de recherche permet de retrouver dans la base de cas, les cas similaires au projet en cours. Il offre des fonctions pour la réalisation de la

première étape du cycle CBR. Ci-dessous les interfaces de recherche et d'indexation des unités de programmes qui ont été implémentées

La figure 7 est l'interface de présentation de CIAO-SI sur laquelle est intégrée notre technique de recherche et d'indexation des unités de programmes

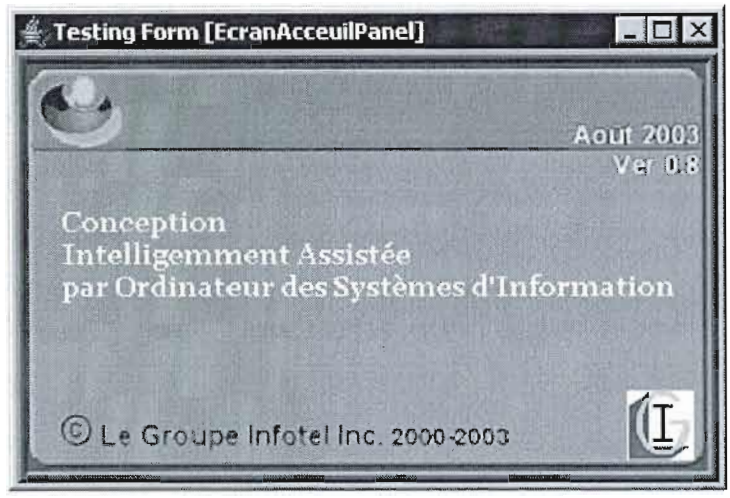

Figure 7: Ecran d'accueil de CIAO-SI

A la suite de l'exécution de l'interface CIAO-SI l'utilisateur est invité à se connecter (figure8). Trois choix sont possibles dépendamment du profil d'utilisateur implémenté dans le système (Administrateur, Expert ou programmeur)

Soit il se connecte comme administrateur du système CIAO-SI, pour la gestion des aspects managériaux et techniques. Soit comme expert de domaine pour la consultation et la validation des différents processus de développement et des unités de programmes. Comme programmeur, l'utilisateur fait des recherches spécifiques dans des domaines particuliers (Gestion des ressources humaines, gestion de paie, gestion des finances, etc.)

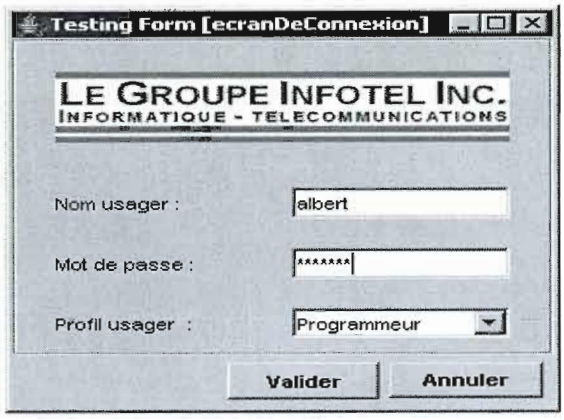

Figure 8: Ecran de connexion de CIAO-SI

Ci-dessous, la figure 9, elle est l'interface de sélection des valeurs des facettes introduites lors de l'implémentation du système. Le programmeur, au lieu de saisir les valeurs des facettes, a le privilège de les choisir à travers une base de données initialement enrichie. La liste de ces valeurs n'est pas exhaustive, cependant pour un premier jeu de test, elle nous semble suffisante.

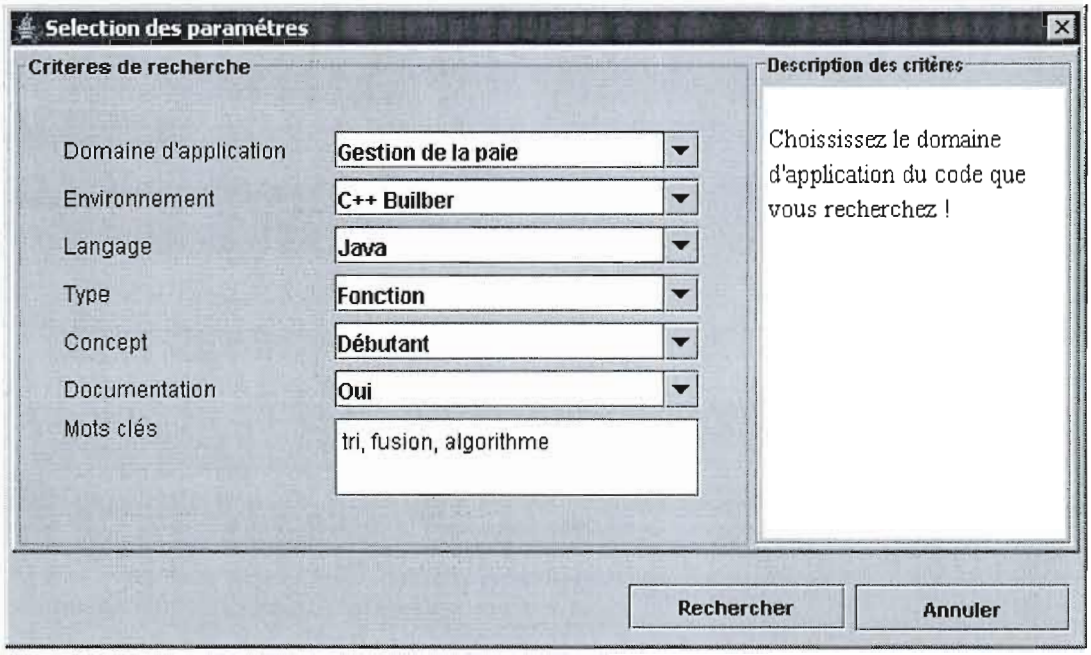

Figure 9: Ecran de sélection des facettes pour la recherche des codes

La figure 10 présente les résultats de recherche des unités de programmes, conformément aux valeurs sélectionnées par le programmeur et aux unités de programmes disponibles dans la base de cas.

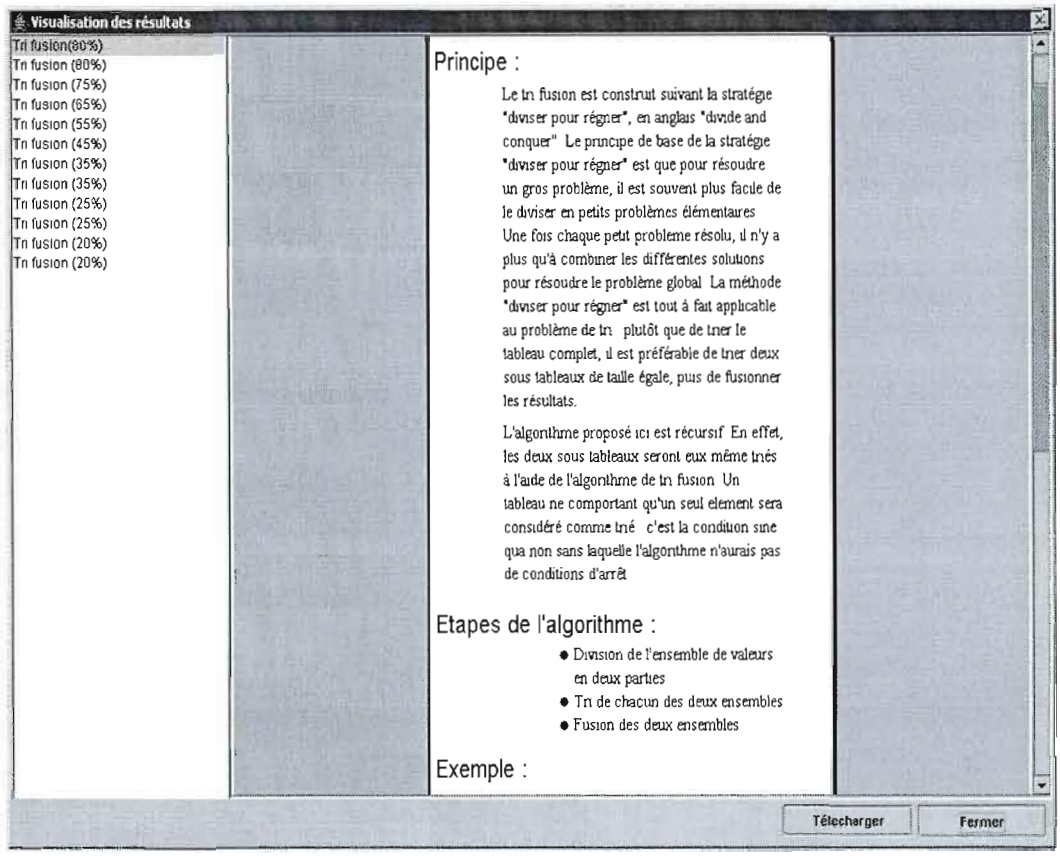

Figure 10: Consulter un cas

4.2 Intégration au Système CIAO-SI.

Le système CIAO-SI tire profit du raisonnement à base de cas pour la gestion des connaissances dans le domaine du développement logiciel. Le système met à la disposition des utilisateurs un certain nombre de fonctionnalités.

Fonctionnalité du système CIAO

Le système CIAO-SI est conçu autour d'un ensemble de modules. Ces différents modules interagissent pour offrir aux utilisateurs un ensemble de fonctionnalités. Ces fonctionnalités tournent autour de la gestion des cas, la gestion de l'espace de travail, l'assistance à l'utilisateur et la gestion des différentes bases de connaissance.

Gestion de la mémoire de cas

Le système CIAO-SI gère un ensemble d'artefacts produits au cours du cycle de développement du logiciel. Une structure et une représentation de cas [Nkambou et al. 2004] élaboré à cet effet permettent de stocker et de gérer les informations sur un projet (ensemble d'artefacts) suivant le processus de développement (documentation, planification, code, diagrammes...). Le système CIAO-SI offre des facilités pour la production et la modification des artefacts. Pour un nouveau projet, les artefacts sont produits sur la base de ceux existants. Il tire profit de la technique présentée plus haut pour proposer à l'utilisateur des projets similaires (cas source) sur lesquels, ils pourront se baser pour construire leur application. Ainsi au lieu de construire l'application sur la base de l'expérience personnelle de chacun, le système offre des moyens pour réutiliser et adapter des solutions aux problèmes similaires stockés dans la base de cas.

Gestion des bases de connaissances du domaine

Le système CIAO-SI gère une base de connaissances pour les différents domaines d'application. Les connaissances de chaque domaine d'application sont issues de l'ontologie du domaine associé (ontologie de concept et ontologie de tâche). L'utilisation des ontologies à ce niveau présente une importance capitale pour le système. Elles permettent non seulement de décrire et d'indexer les différents cas mais aussi aident à fournir une assistance spécifique au domaine au cours de la phase d'adaptation. Au cours de cette phase, elles permettent notamment de faire des

suggestions, des recommandations et des validations des différents modèles dépendamment du domaine de l'application en cours de développement.

Gestion des bases de connaissances du processus

Le système CIAO-SI gère aussi les connaissances sur les processus de développement et celles associées aux unités de programmes. Pour chaque processus de développement, une ontologie correspondante est développée. Cette ontologie précise les différentes phases du cycle de production, les différents artéfacts et le rôle des différents intervenants. Dans le but d'illustrer notre réflexion à travers un exemple concret, une ontologie a été développée pour le processus de développement RUP (Rational Unifié Procès). L'ontologie du processus MERISE est en cours développement. Cependant le système est ouvert à tout autre processus. Pour chaque unité de programmes est défini un ensemble de facettes pour lesquelles sont attribuées des valeurs. Cependant le système est ouvert à l'ajout de nouvelles facettes.

Assistance au programmeur

Le système CIAO-SI offre une assistance au concepteur et principalement durant la phase d'adaptation des cas. Cette assistance se compose d'un tutoriel, de recommandations, de suggestions et de validation des différents artéfacts produits. Elle se base sur les développements récents issus du domaine du E-Learning pour proposer au développeur un service de qualité. Les ontologies de domaines et de processus sont utilisées pour proposer au concepteur une assistance spécifique en fonction du domaine d'application et du processus de développement utilisé. Contrairement aux approches existantes qui proposent une assistance statique basée sur une méthodologie comme UPEDU [Robillard et Kruchten, 2002] (assistance à l'utilisation de RUP), notre approche se distingue par son aspect dynamique. En effet, l'assistance offerte au concepteur par le système CIAO-SI est variable d'un concepteur à l'autre suivant le profil de ce dernier, le processus utilisé, le domaine d'application considéré et des problèmes rencontrés. Cette fonction est d'une

importance clé pour le système car elle procure au programmeur des facilités pour produire rapidement des unités de programmes de qualité acceptable.

Architecture du système CIAO-SI

L'architecture du système CIAO-SI (figure 11) laisse apparaître un ensemble de modules qui interagissent pour réaliser les fonctions ci-dessus présentées.

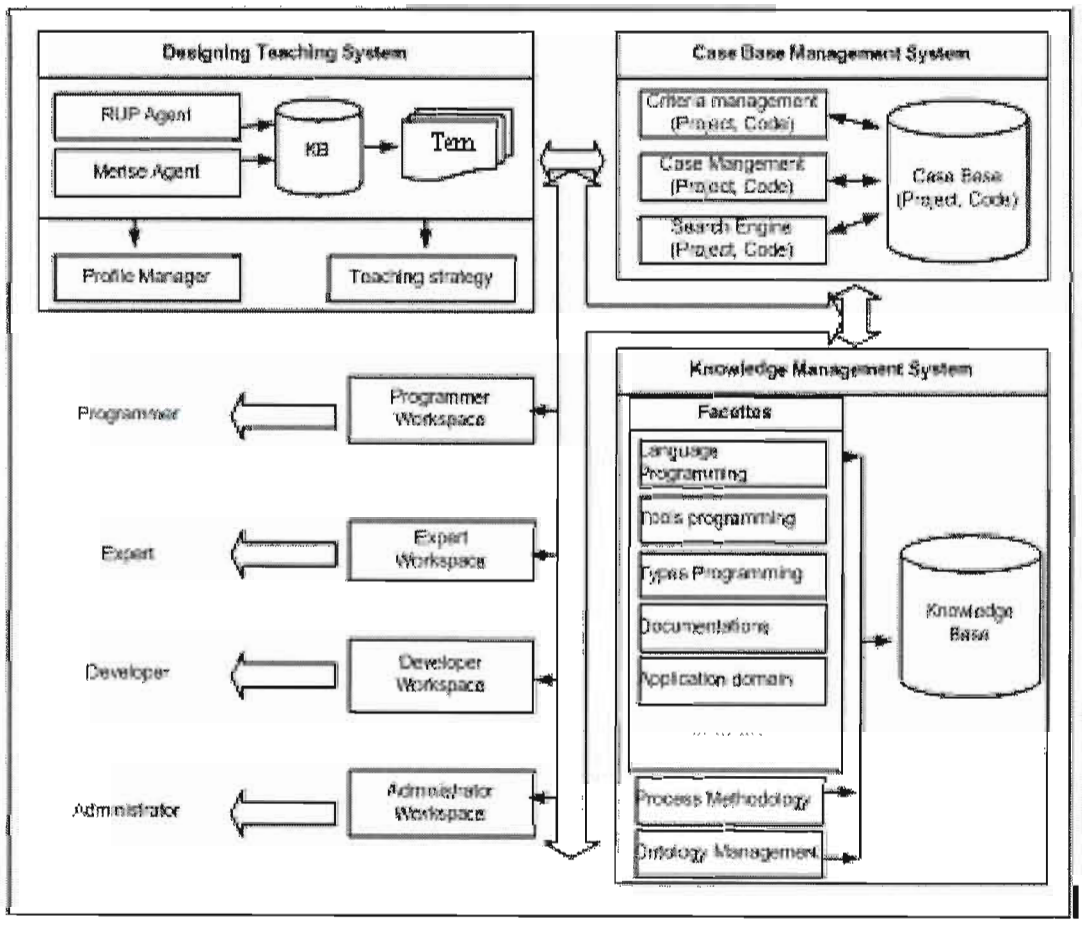

Figure 11: Architecture du système CIAO-SI

Le système CIAO-SI présente quatre profils utilisateur : l'administrateur, l'expert, le développeur et le programmeur. Chaque profil contient des informations personnelles sur l'usager permettant de gérer sa session de travail. L'administrateur

se charge des fonctions classiques d'administration (gestion des utilisateurs, configuration du système). L'expert enrichit et met à jour les différentes bases de connaissances du système à travers une interface apprêtée à cet effet. Le développeur utilise le système pour construire des solutions logicielles selon le domaine d'application. Le programmeur utilise le système pour la recherche, l'indexation des unités de programmes afin de trouver la solution ou les solutions proches de son domaine de recherche. Outre les différents profils utilisateurs, l'architecture du système CIAO-SI présente trois grands modules : le module de gestion des connaissances, le module de gestion de la base de cas et le module d'assistance.

Le module de gestion des connaissances se compose des sous module de gestion des connaissances de domaine d'application et du sous module de gestion des connaissances sur le processus. Le sous-module de gestion des connaissances du domaine propose un ensemble d'outils permettant de définir, de représenter et de stocker l'ontologie de domaine. Le sous-module de gestion des connaissances sur le processus propose des outils pour la gestion des connaissances sur le processus. Toutes ces connaissances sont stockées dans une base de connaissance commune.

Le module d'assistance offre des fonctions d'assistance lors de la phase de modification des cas. Il se compose des agents dédiés en fonction du processus utilisé, d'un gestionnaire de profil utilisateur et d'un élaborateur de stratégie. Les agents dédiés utilisent les informations de la base de connaissances du domaine et du processus et de leur connaissance propre pour offrir aux utilisateurs un service d'assistance très appréciable. Le gestionnaire de profil enregistre et restitue toutes les informations nécessaires sur le profil des différents utilisateurs. L'élaborateur de stratégie se base sur les informations sur le profil de l'utilisateur, le domaine d'application et le processus de développement pour construire une stratégie d'assistance (critique, coaching).

4.3 Mise en œuvre et Evaluation expérimentale

Pour la mise en œuvre du système, nous nous sommes servis des technologies actuellement utilisées par la plupart des développeurs d'applications. Il s'agit notamment de Java [Horton, 2001] comme langage de programmation et de MYSQL comme serveur de base de données.

Objectif et procédure expérimentale

Nous savons que la réutilisation des unités de programmes dans le développement logiciel est depuis plusieurs décennies une préoccupation majeure dans les sociétés de développement logiciel. Plusieurs expériences en industrie [Basili, V.R. Rombach H 1994] suggèrent que la réutilisation contribue à accroître la productivité et la qualité). Lim et W.C. [Lim et W.C, 1994] estiment que la réutilisation accélère la livraison des produits et un retour sur l'investissement très avantageux. En général, les systèmes développés à partir de la réutilisation des unités de programmes coûtent moins cher. Des gains de productivité se font dans la réduction des coûts de développement. Deux facteurs permettent d'expliquer ces gains de productivité : un délai de livraison plus court et l'élaboration du code avec moins d'erreurs. Le programmeur écrit moins de code, la réutilisation réduit la quantité de documentation et des jeux de tests et le système est plus facile à maintenir, car les programmeurs sont plus familiers avec les composants réutilisables qui ont servi à concevoir le système.

Le Groupe Infotel Inc. Une société de développement de logiciel orientée dans la livraison des logiciels développés sur mesure depuis plus de quinze ans nous a servi de cadre expérimental de notre travail de recherche. Elle possède à son actif plus d'une dizaine de progiciels avec différentes versions. Il s'agit des progiciels Ressources humaines (GRH1, GRH2, GRH3), de la paie (PAIE1, PAIE2, PAIE3), des immobilisations (IMMO1, IMMO2) de la comptabilité générale (CG1, CG2, CG3), de la comptabilité analytique (CA1, CA2), de la Trésorerie (TR1, TR2), de la

comptabilité budgétaire (CB1, CB2, CB3), des stocks (STO1, STO2), de la gestion commerciale (GC).

Nous nous sommes basés au moyen de rappel et de mesures de précision de Salton pour évaluer notre travail. Rappelons que ''Le rappel'' est le rapport des documents appropriés recherchés par rapport au nombre de documents appropriés dans la base de données tandis que ''la précision'' est le rapport du nombre de documents appropriés recherchés par rapport au nombre de documents recherchés.

Une source de difficulté avec des études basées sur le rappel et les mesures de précision est que toutes les deux exigent des jugements au sujet de la pertinence d'un document. De tels jugements de pertinence, qui doivent être faits par les juges humains, sont incertains.

L'approche d'évaluation de la réutilisation des unités de programmes que nous avons élaborés dans le cadre de ce travail est d'établir une étude comparative entre la recherche et la réutilisation des codes informatiques dans les précédents projets de réalisation des applications en utilisant l'approche de recherche de la réutilisation des codes (recherche utilisation des codes informatique existant selon la seule appréhension du programmeur dans la réalisation de nouveaux projets) et la réutilisation des codes avec l'outil CIAO-SI. (Tableau 4 : Tableau d'évaluation de l'expérimentation).

Les valeurs ont été attribuées par l'expert selon son appréhension qui en jugeant le temps nécessaire pour un programmeur de retrouver les codes.

Tableau d'évaluation de l'expérimentation

Tableau comparatif de recherche et de réutilisation des unités de programmes			
Domaine d'application	**Désignation**	**Recherche et utilisation des codes informatiques pour les projets LGI** (estimé approximative de la capacité de retrouver et de réutiliser les codes)	**Recherche et utilisation des codes fournis par l'outil CIAO-SI pour les projets LGI** (estimé approximative de la capacité de retrouver et de réutiliser les codes)
GRH	Gestion des ressources humaines	10-40%	50-90%
PAIE	Gestion de la Paie	10-40%	50-90%
IMMO	Gestion des immobilisations	10-40%	50-90%
STO	Gestion des stocks	10-40%	50-90%
GCO	Gestion commerciale	10-40%	50-90%
CG	Gestion de la comptabilité générale	10-40%	50-90%
CA	Gestion de la comptabilité analytique	10-40%	50-90%
CB	Gestion de la comptabilité budgétaire	10-40%	50-90%
TR	Gestion de la trésorerie	10-40%	50-90%

Tableau 4: Tableau comparatif des résultats

Recherche et utilisation des codes informatiques au LGI :

Le programmeur affecté à un projet de développement donné se sert des documents produits lors des phases d'analyse de conception et de modélisation dudit projet. Ces documents lui servent de référence pour la réalisation des unités de programmes ou des fonctions du projet. De prime a bord comme tout programmeur

de la société commence à fouiller les répertoires des codes informatiques dudit domaine d'application pour retrouver les codes de programmes semblables ou tout simplement un code répondant au mieux à son besoin. En général les codes retrouvés dans la majeure partie des cas ne répondent pas aux besoins ni aux spécifications fonctionnelles du système en cours de réalisation. Le constat est d'autant plus réel, qu'il s'avère que les unités de programmes sont soit mal documentées ou tout simplement incomplètes. Le programmeur est confronté à différents choix. Soit il décide de recommencer à zéro un travail qui fort probablement peut s'avérer existant, mais simplement non disponible à cause du non respect des procédures de stockage des codes ou soit il va essayer de modifier le code trouvé et l'adapté à son contexte. Pour un programmeur d'expérience, la tache peut être moins difficile. Cependant rien ne garantie que le nouveau code crée ou modifié va respecter les règles de l'art dans la réalisation des unités de programmes (documentation détaillée, règles de gestion bien décrites, clarté dans l'écriture des codes, etc.). D'ou le pourcentage (0 à 40%) attribué par l'expert qui estime qu'il y a toujours plus de travail à faire pour l'adaptation du code retrouvé lors de la recherche. Les documents de spécifications techniques ou fonctionnelles retrouvés sont incomplets, les codes retrouvés sont mal documentés.

Recherche et utilisation des codes fournis par l'outil CIAO-SI

L'outil CIAO-SI est doté d'un module assistance à la conception et l'adaptation des cas. Ce module est activé lors du choix du profil de l'utilisateur du système. Selon qu'il soit administrateur du système ou programmeur ou aussi l'expert du domaine [NKambou 2004]. Le module d'assistance du système CIAO-SI offre une assistance au programmeur pendant la phase de résolution de son problème en lui fournissant un cadre de sélection des critères de recherche et principalement durant la phase d'adaptation des cas. Cette assistance se compose d'un tutoriel, de recommandations, de suggestions et de validation des différentes unités de

programmes produites. Elle se base sur les développements récents issus du domaine du E-Learning pour proposer au programmeur un service de qualité. Les ontologies de domaine et de processus sont utilisées pour proposer au programmeur une assistance spécifique en fonction du domaine d'application et du processus de développement utilisé. Notre approche se distingue par son aspect dynamique. En effet, l'assistance offerte au concepteur par le système CIAO-SI est variable d'un concepteur à l'autre suivant le profil de ce dernier, le processus utilisé, le domaine d'application considéré et des problèmes rencontrés. Cette fonction est d'une importance clé pour le système car elle procure au programmeur des facilités pour produire rapidement des unités de programmes de qualité acceptable.

Le programmeur se connecte à travers son profil utilisateur pour l'exécution du module de réutilisation de codes. A la suite de l'exécution du module, le programmeur a le choix entre bâtir l'unité de programme à partir des unités de programmes existantes (cas) ou non. S'il opte pour la construction sur la base d'un cas existant, la fonction de recherche est automatiquement activée. Cette fonction lui permet de retrouver dans la base de cas, un ou plusieurs cas proche(s) du domaine d'application. Basée sur la représentation du problème sous forme d'indicateur, la recherche classifie les cas contenus dans la base de cas suivant les valeurs des indicateurs contenues dans la description du problème. Un calcul de similarité entre les cas sources sélectionnés et le cas cible est ensuite effectué. Ce calcul permet à l'utilisateur d'apprécier la distance entre l'unité de programme à développer (cas cible) et l'unité de programme contenu dans le cas (cas source). A la fin de la recherche, un ou plusieurs cas pertinents sont sélectionnés par l'utilisateur en fonction de leur degré de similarité et des choix effectués par l'utilisateur en fonction de la description des cas. Les cas ainsi sélectionnés sont téléchargés par l'utilisateur pour fin de consultation, de modification dans l'environnement de développement souhaité. L'expert a donc estimé que les documents de spécifications techniques et fonctionnelles ainsi que les unités de programmes retrouvées répondent au mieux à

l'attente du programmeur. D'ou le pourcentage (50 a 90%) attribué pour signifier le résultats de recherche. Il est vrai que le programmeur aura toujours le choix de sélectionner le cas qui lui semble pertinent de son domaine de recherche.

En définitif, l'outil CIAO-SI est plus approprié pour la réutilisation des connaissances dans le domaine du développement des logiciels et en particulier la réutilisation des unités de programmes pour la réalisation de nouveaux projets.

Conclusion

Synthèse :

Dans le cadre de ce mémoire, nous devions doter l'outil CIAO-SI d'un module permettant la validation et le stockage ainsi que la recherche et l'indexation des unités de programmes éprouvées dans le but de les réutiliser dans de nouveaux projets. Notre Travail consistait alors à faire :

- La proposition d'une description et d'une représentation des unités de programmes sous forme de cas réutilisables.
- La proposition d'une approche de recherche et d'indexation des unités de programme basée sur la définition de facettes
- La mise en place d'un système d'aide à la réutilisation des unités de programmes.
- L'intégration du système d'aide dans CIAO-SI en vue d'un support proactif dans un contexte de programmation.

Nous avons ainsi contribué à la description et la représentation des unités de programmes sous forme de cas réutilisables. La mise en œuvre de cette structure de cas a nécessité l'utilisation du langage UML (Unified Modeling Langage). Nous avons proposé et implémenté une approche de réutilisation des unités de programmes basée sur le CBR et la classification à facettes. L'implantation de cette approche dans le contexte de CIAO-SI, une plateforme de réutilisation dans les phases en aval de la réalisation, nous a permis d'obtenir des résultats encourageants. Pour tester la pro-activité de notre approche, nous avons exploité l'efficacité et la puissance des SGBD par l'implantation de la structure de cas ainsi que la technique de recherche et d'indexation. Finalement nous avons effectué une évaluation expérimentale par la méthode de Salton et cela nous a permis d'évaluer la pertinence de nos résultats.

Aujourd'hui, très peu de systèmes de réutilisation de codes disponibles sur le marché utilisent la classification facettée appliquée au CBR pour la recherche et l'indexation dans les industries.

Limites de recherche:

Le modèle de classification à facettes pose des problèmes de cohérence. Il ne s'agit pas à proprement parler d'un problème de cohérence entre les différentes vues, mais plutôt de savoir si les différentes facettes sont bien indépendantes et si la classification qui en découle est complète. Prieto-Díaz résout ce problème en confiant la création des facettes et des termes à un expert [Benjamin Sigonneau, 2003], celui-ci se basant sur un échantillon représentatif de la bibliothèque de fonctions pour mener à bien son travail.

Nous considérons qu'il faut apporter des améliorations significatives à l'approche d'évaluation de la pertinence des unités de programmes considérée par l'expert comme un nouveau cas. Bien attendu aujourd'hui encore, l'expert de domaine est sollicité pour évaluer les unités de programmes modifiées et adaptées dans le but de leur stockage. Cette approche d'évaluation n'est pas encore au stade de l'automatisation. L'automatisation est une piste pour nos travaux futurs.

Perspectives de recherche:

Nous comptons faire une étude empirique de l'usage du module de réutilisation des unités de programmes dans les entreprises de développement d'applications, dans le but de tester et d'évaluer la pertinence de recherche des unités de programmes. Les données sur l'utilisation de notre module et sur la satisfaction des utilisateurs pourraient nous permettre de tirer de nouvelles conclusions sur la technique implémentée.

Bibliographie

[Aamodt, Plaza 1994] A.Aamodt, E. Plaza 1994. <<Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches>> AICom - *Artificial Intelligence Communications, IOS Press*, Vol. 7 No 1, pp. 39-59.

[Arongo, Guilermo, 88] Arongo, Guilermo Francisco 88. <<Domain engineering for software reuse>> UMI 300N ZEE red University of California, (Irvine), 1998 (These).

[Basili et Rombach, 1994] Basili, V. et Rombach H. 1994. <<Experience Factory>>. *Encyclopedia of Software Engineering.* Vol. 1, John Wiley & Sons, pp. 476-496.

[Belkin & Croft 1985] N. Belkin & W. Croft 1985. <<Retrieval techniques>> *M. Williams, editor, Annual Review of Information Science and Technology (ARIST)*, Vol. 22, Ch. 4, pp. 109-145. Elsevier Science Publishers B.V., 1985.

[Bejar 91] Bejar, I. I. Chaffin R., and Embretson S., 91. <<Cognitive and Psychometric Analysis of Analogical Problem Solving>>, Springer-Verlag, 1991.

[Benjamin, 2003] Benjamin Sigonneau 2003. <<Concept de vue et génie logiciel>> – étude bibliographique Canonne, André. 1993. Vocabulaire élémentaire des classifications. Liège : CÉFAL. (Collection Bibliothèque du bibliothécaire, 2).

[Bevo, Nkambou, Donfack 2003] Valery Bevo, Roger Nkambou, Hervé Donfack 2003. <<Toward A Tool for Software Development Knowledge Capitalization>>. *Proceedings of the 2nd international conference on information and knowledge sharing. ACTA* press, Anaheim, pp. 69-74.

[Bhansali S. 1993] <<Architecture-driven reuse of code in KASE>> *Proc. of the 5th Conf on Software Eng. & Knowledge Eng.*, 1992, pp. 100-109.

[Bjørnar, T., Solveig, B., Knut, T. et Gorm, S. 1994]. <<*ROSA* = Reuse of Object-oriented Specifications through Analogy: A Project Framework>>. No 21 Département de science de l'Information. Université de (Bergen).

[Biggerstaff T. Richter .C 1987] <<Reusability Framework Assessment Directions>> *IEEE Software*, Vol. 10 No 3 Mars 1987. PP 41-49.

[Brooks 1975] Brooks, J. (1975). <<Telephone: The First Hundred Years>>. Harpter & Row 1975.

[Cox Rosa 2001] <<The Myth and Reality of Reuse A TRC White Paper>> The Technical Resource Connection. 2001

[Chung-Hong and Joseph, 95] Chung-Hong Luang and Joseph E. Urban, 95<<An approach to the classification of domain models in support of analogical reuse>>. PP.169- 178; 1995

[Cunningham et Zenobi 2001] Cunningham, P. et Zenobi, G. 2001. <<Case Representation Issues for Case-Based Reasoning from Ensemble Research>>. Technical Report TCD-CS-2001-10. Department d'Informatique Trinity College Dublin.

[Dahl and Nygaard, 1964] Dahl, O-J. and Nygaard, K 1964. The SIMULA Project. <<Technical Progress Report 1>>. Juillet 1964. NCC Doc. (D).

[Dard 94] Dave Dard, Ed Comer, 94. <<Why do so many reuse programs fails>>, IEEE Software, Sep 1994, pp114-115.

[David Garlan, Robert Allen, and John Ockerbloom 95] David Garlan, Robert Allen, and John Ockerbloom, 95.. <<Architectural mismatch why reuse is so hard>>. *IEEE Software*, Vol.12 No.6, PP.17-26, 1995.

[Donfack, Nkambou et Bévo, V. 2004] Donfack Hervé, Nkambou Roger et Bévo Valery. 2004. <<Case Retrieval in Software Engineering Capitalization Tool Using Dynamic Inductive Tree and Inverted List>>. *Artificial Intelligence and Application* (AIA'04), Accepted.

[Donfack, 2004] Donfack, Hervé. 2004. << Recherche de cas par arbre inductif dynamique et liste inversée: application à la capitalisation des connaissances dans le développement logiciel>>. Mémoire de maîtrise, Montréal, Canada, Département d'informatique - Université du Québec à (Montréal).

[Delacroix 99] Q. Delacroix, 99. <<Un système pour la recherche plein texte et la consultation hypertexte de documents techniques>>. Thèse de l'université Blaise Pascal Clermont Ferrand II. 1999.

[Dubuque, Proulx, Rolland S 1991] Dubuque A., Proulx J-M, Rolland S 1991. le projet mobilisateur <<le macroscope informatique et le génie logiciel>>, *ICO Québec*, Automne 1991, PP 76-79.

[Dussaux, 2001] Dussaux, G. 2001<<IronWEB – Construction collective de bases de connaissances sur le Web". Thèse de Doctorat. INSA de Rouen 2001.

[Dijkstra 1989] Edsger W. Dijkstra 1989. <<On the Cruelty of Really Teaching Computer Science>>. Communications of the ACM 32(12), pp 1398-1404, 1989.

[Engelhart, 2001] Engelhart, P., M.2001. << Knowledge Management in Software Engineering>>: *A State-of-the-Art-Report*, Air Force Research Laboratory, Information Directorate/IFED, 32 Brooks Road, Rome, NY 13441-4505. 29 Novembre 2001

[Ermine, 2003] Ermine, J. 2003. <<La gestion des connaissances: points de vue et expérience>>http://www.cefrio.qc.ca/allocutions/Ermine_HECMontreal_201103.pdf.

[Fernández Chamizo 1995] C., González Calero, P.A., Hernández Yáñez, L., Urech Baqué, A. 1995. <<Case- Based Retrieval of Software Components>>. *Expert Systems with Applications*, Vol. 9, No. 3, pp. 397-421.

[Frakes et Gandellal, 97] Frakes W.B. & Gandellal P.B. 97. <<Classification, Storage and retrieval of reusable components>> *Panel Session*: Information Retrieval and Software Reuse 1997.

[Frakes et Nejmeh, 1987] Frakes, W. et Nejmeh, B. 1987. <<A Software Reuse System through Information Retrieval>>. *Proceedings of 20 the Annual Hicss, Kona, Hicss, Hona, Hi.* Vol. 2, pp 30-36.

[Gerald and Prieto-Díaz 88] Gerald Jones and Rubén Prieto-Díaz 88. <<Building and managing software libraries>> *COMPSAC'88, Proceedings of the 12th Annual International Computer Software and Applications Conference*, pp. 228-236, 1988.

[Gerhard,1987] Gerhard F.1987. <<Cognitive view of reuse and redesign>>. I*EEE Software*, Vol.4 No.4 pp.60-72, 1987.

[Gerhand, Henninger et Redmile 91] Gerhand Fischer, Scott Henninger, David Redmile 91. <<Cognitive tools for locating and comprehending software objects for reuse>> *Proceedings of the 13th international conference on Software engineering* Mai 1991.

[Goldberg 1990] Goldberg 1990. <<Information Models, Views, and Controllers>> *Dr. Dobb's Journal* Juillet 1990

[Gomes, Paulo, Nuno, José et Carlos 2002a] Gomes, P. Francisco, C., Paulo, P., Nuno, S., Paulo, C., José, F., et Carlos B. 2002a. <<Case Retrieval of

Software Designs using WorldNet>>. *Proceedings of the European Conference on Artificial Intelligence (ECAI'02).* pp.245-249.

[Gomes, Paulo, Nuno, José et Carlos, 2002b] Gomes, P. Francisco, C., Paulo, P., Nuno, S., Paulo, C., José, F., et Carlos B. 2002b. <<Experiments on Cased Retrieval of Software Designs>> *6th European workshop on CBR (ECCBR 2002)*, pp.118-132.

[Gruman, 1979] Gruman L, 1979. <<Value distribution for holomorphic maps in n, Math>> 245 (1979), pp.310 1 (1988), pp. 47- 86.

[Grosz 92] Grosz 92. <<Building information system requirements using generic structures>> *Proc. of the 16th Intel Computer Software & Applicatins Conf (COMPSAC)*, 1992, pp. 200-205.

[Haining et Etzkorn, 2004] Haining Yao, Letha Etzkorn, 2004. <<Towards a semantic based approach for software reusable component classification and retrieval"*ACME, Huntsville, Alabama, USA*. Avril 2004

[Herrmann 86] D. J. Herrmann and R. Chaffin, 1986. <<Comprehension of semantic relations as a function of the definitions of relations>> *Human Memory and Cognitive Capabilities: Mechanisms and Performances, F. Klix and H. Hagendorf, Eds., Elservier Science Publishers B.V., North-Holland,* 1986. pp. 311-319.

[Horowitz et Munson 1984] Horowitz et Munson 1984. <<An expansive view of Reusable Software>> *IEEE Transactions on Software Engineering.* Pp 477-487. Septembre 1984

[Horton, 2001] Horton, I. 2001. Maîtrisez Java 2 (JDK 1.3). *CampusPress.*

[Jurisica 97] Jurisica Igor 97. <<Similarity-based retrieval for diverse bookshelf software repository users>> *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research* Novembre 1997.

[Jaczynski, 1993] Jaczynski, M. 1993. <<Etude du raisonnement par cas : recherche des cas similaires et utilisation des ensembles floues>>. *Rapport de stage de DEA informatique Université de Nice Sophia-Antipolis* 1993

[Jones, 1984] Jones T.C 1984. <<Reusability in programming: A survey of the State of the Art>> I*EEE Transactions on Software Engineering,* PP 498-501. Septembre 1984

[Katsuro, Yokomori and al. 2003] Katsuro Inoue, Reishi Yokomori and al. 2003. <<Component rank: Relative significance rank for software component search>>. *Proceedings of the 25th International Conference on Software Engineering* May 2003.

[Krueger 92] Charles W. Krueger, 1992. <<Software reuse>>, *ACM Computing Surveys (CSUR),* Vol.v.24 No.2, pp.131-183, Juin 1992.

[Maniez, 1999] Maniez, Jacques 1999. <<Des classifications aux thésaurus : du bon usage des facettes>> *Documentaliste - Sciences de l'information,* Vol. 36, No 4-5, pp.249-262.

[Maiden, 92] Neil Maiden, Alistair Sutcliffe, 1992. <<Exploiting reusable specifications through analogy>>, *Communications of the ACM*, Vol.35 No.4, pp.55-64, Avril 1992

[Maiden 93] N. A. M. Maiden and A. G. Sutcliffe, 1993. <<Requirements engineering by example: an empirical study>> *Proc. of lEEE Int'l Symposium on Requirements Eng.*pp. 104-111. 1993

[Martin, 1990] Martin J 1990. <<Information Engineering>>, Book3. D*esign and Construction*, Prentice-Hall 1990.

[Maple, Amanda. 1995] Maple, Amanda. 1995. <<Faceted access>>. *Review of the literature.http://wwwsul.stanford.edu/depts/music/mlatest/BCC/BCC-Historical/95WGFAM2.html.*

[Marie Conte Andrew and al. 1998] Marie T. Conte Andrew R and al. 1998. <<A Study of Code Reuse and Sharing Characteristics of Java Applications>> *Department of Electrical and Computer Engineering Coordinated Science Laboratory*, Mai 1998.

[McIlroy 1969] McIlroy 1969. <<Édition de références sur la réutilisation de logiciel>> *Conférence de l'OTAN de New York* sur la technologie de la programmation. 1969

[Michèle Hudon 2001] Michèle Hudon 2001. <<Analyse des facettes pour la classification des documents institutionnels au gouvernement du Québec>> Rapport préparé pour le Groupe de travail en classification et indexation Octobre 2001.

[Mitchell, Howse et Maung 1995] Richard Mitchell, John Howse, Ian Maung 1995. <<As-a: a relationship to support code reuse>> *Journal of Object-Oriented Programming*, Vol. 8, No, pp 25-33 & 55 and is SIGS Publications 1995

[Mineau et al. 94] Amina Arfi, Robert Godin, Hafedh Mili, Rokia Missaoui, Mineau Guy W. 94: <<Interface Hierarchy of a Class Library>> *COODBSE* pp: 42-57. 1994

[Mili et al. 95] Hafedh Mili, Fatma Mili, and Ali Mili 1995. <<Reuse Software: Issues and Research Directions>>, I*EEE Trans. Software Engineering*, Vol. 21, No. 6, Juin 1995.

[Mili & al, 97] A. Mille, A. Napoli 97. <<Aspects du raisonnement à partir de cas>>. A*ctes des 6ème journées nationales du PRC GDR Intelligence Artificielle. HERMES* 1997.

[Mili et al, 99] Ali Mili, Sherif Yacoub, Edward Addy, Hafedh Mili, 99. <<Toward an Engineering Discipline of Software Reuse>>, *IEEE Software*, Sep/Oct, 1999.

[Mili, 99] Mili, 99<<A survey on reuse: Research fields and Challenges>> *State of the of Reuse and CBSE.* 1999

[Mili 2000] Ali. Mili, S. Fowler Chmiel, R. Gottumukkala, L. Zhang, 2000. <<An Integrated Cost Model for Software Reuse>>, *Proceeding of International Conference of Software Engineering, Limerick (Ireland)*, 2000.

[Mili, Yacoub, 2000] Mili. A, Yacoub S.M, 2000. <<A Comprehensive Analysis of Domain Engineering Methods: A Controlled Case Study>> *International Conference on Software Engineering (ICSE'22)*, Workshop on Software Product Lines: Economics, Architectures, and Implications, Limerick, (Ireland), Pages 20-31, Juin 2000.

[Morisio 2002] Maurizio Morisio, Michel Ezran, Colin Tully, 2002. <<Success and Failure Factors in Software Reuse>>, *IEEE Transaction on Software Engineering,* Vol. 28, No.4, April 2002. pp 340-357,"Software Reuse: Silver Bullet", *IEEE Software,* Septembre et Octobre 2001

[Moorman Zaremki and Wing, 97] Amy Moorman Zaremki and Jeannette M. Wing 97. <<Specification matching of software components>> *ACM Transactions on Software Engineering and Methodology (TOSEM),* Vol. 6, I. 4 Octobre 1997.

[Neighbors 84] J. M. Neighbors, 1984. <<The Draco approach to constructing software from reusable components>> *IEEE Trans. on Software Engineering*, Vol. 10, No. 5, pp. 564-574, Septembre. 1984

[Ngantchaha , Nkambou, and Bevo, 2004] Ngantchaha G., Nkambou, R. and Bevo, V. 2004. <<Software Engineering Knowledge Capitalization Using CBR: Case Structure>> *Artificial Intelligence and Application (AIA'04)*, Accepted.

[Ngantchaha, 2004] Ngantchaha Ghislain. 2004. <<Gestion des cas dans un système de capitalisation de connaissances basée sur le raisonnement à partir de cas :

application au système ciao-si>> Mémoire de maîtrise, Montréal, Canada, Département d'informatique - Université du Québec à (Montréal).

[Nguyen, Abran, Bayard and DeChantal, 1997] Nguyen, T. Abran, A. Bayard, R. and DeChantal, O. 1997. <<Les concepts de la réutilisation du logiciel et la pratique institutionnelle dans des entreprises québécoises>>. *Le génie logiciel et ses applications, dixièmes journées internationales (GL97)*, (Paris), EC2D.

[Lamontagne et Lapalme, 2002] Lamontagne Luc et Guy Lapalme 2002. <<Raisonnement à base de cas textuel - état de l'art et perspectives futures>> *Revue d'intelligence artificielle*, Vol.16 No.3 pp.339-366, 2002.

[Lamrous, 99] Lamrous S 99. <<Modélisation et réalisation d'un système prototype interactif de recherche d'information multimédia à forte composante textuelle>>, Thèse de doctorat, UT Compiègne, Juin 1999.

[Lanergan, 1984], Lanergan RG et Grasso, CA 1984. <<Software Engineering with Reusable Design and. Code>>. *IEEE Transactions on Software Engineering*, Vol.10, No 5 Septembre 1984.

[Langridge, Wilton. 1992] Langridge, Derek Wilton. 1992. <<Classification: its kinds, elements, systems, and applications>> *London Bowker-Saur, in association with the Centre for Information Studies*, Charles Sturt University, Wagga Wagga, N.S.W. (Topics in library and information studies).

[Leake, 96] Leake D.B 96. <<Case-Based Reasoning: Experiences, Lessons, and Future Directions>> *AAAI Press/MIT Press, Menlo Park*, CA, 1996.

[Paulo Gomes& al. 2002] Paulo Gomes& al. 2002<<Case Retrieval of Software Designs using WordNet>> *Proceedings of the European Conference on Artificial Intelligence* 2002 (ECAI'02).

[Peter 2004] Peter-Min-Sheng Hsieh, 2004 <<CBR in code reuse. EPS Engineering Postgraduate Society>> *The BECA Postgraduate Poster Competition 2004*'' Software Engineering The University of (Auckland) 2004.

[Podgurski and Pierce 93] Andy Podgurski and Lynn Pierce 93. <<Retrieval reusable software by sampling Behavior>>*ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 2, I. Juillet 1993.

[Poulin 93] J. S. Poulin and K. P. Yglesias, 1993 <<Experiences with a faceted classification scheme in a large reusable software library (RSL) >> *Proc. of the 17th Intel Computer Software & Applications Conf (COMPSAC)*, pp. 90-99. 1993.

[Premkumar Devambu, 91] Premkumar Devambu, Peter G.Selfridge, Ronald J Brachman Bruce W.Ballard, 91. <<Lassie: a knowledge-based software information systems>> *Communications of the ACM,* Vol. 34 I.5 Mai 1991

[Premkumar & Jone 97] Premkumar Devanbu & Mark A. Jone 97. <<The use of description logics in KBSE system>> *At&T Laboratories research ACM Transactions on Software Engineering and Methodology (TOSEM).* Vol. 6, I.2, Avril 1997

[Prieto-Díaz & Freeman, 1987] Prieto-Díaz, R& Freeman, P. 1987. <<Classifying Software for Reusability>>. *IEEE Software*. Vol. 4, No 1. pp. 6-16. 1987

[Prieto-Diaz et Arango,91a] Ruben Prieto-Diaz , G. Arango,91a. <<Domain Analysis and Software Systems Modeling>>, *IEEE Computer Society Press, Los Alamitos*, CA, 1991

[Prieto-Diaz, 91b] Rubén Prieto-Díaz 91b, <<Implementing faceted classification for software reuse>>, *Communications of the ACM*, Vol.34, No.5, pp.88-97, Mai 1991

[Prieto-Diaz 93] Rubén Prieto-Díaz,93. <<Status Report: Software Reusability>>, *IEEE Software*, Vol.10 No.3, pp.61-66, Mai 1993

[Rabea, Ezzat et El-Zoghabi, 2001] Rabea, A., Ezzat, A. et El-Zoghabi, A. 2001. <<Applying a Case-Based Reasoning to Help Desk Application>> http://www.geocities.com/a_ragab75/paper.HTM

[Richard, Yoelle et Maarek 91] Richard helm and Yoelle S. Maarek 91. <<Integrating information retrieval and domain specific approaches for browsing and retrieval in object-oriented class libraries>>. Pp. 47 - 61. 1991 Phoenix, (Arizona), United States

[Riesbeck, Shank, 89] Riesbeck C., Shank R., 89. <<Inside Case-Based Reasoning>>, *Lawrence Erlbaum Associates*, 1989.

[Robillard, 2002] Robillard P.N. & Kruchten P. (2002). <<Software Engineering Processes: With the UPEDU>>. *Pearson Addison Wesley; 1st edition.*

[Sankar and Simon, 2004] Sankar K. Pal and Simon C. K. Shiu 2004. Foundations of Soft Case-Based Reasoning, John Wiley & Sons, 2004

[Salton & al 83a] G. Salton & M.J. McGill, 83a <<Introduction to modern information retrieval>>, Ed. Mc Graw Hill, 1983.

[Salton & al 89] G. Salton, M. Smith 89. << On the Application of Syntactic methodologies in Automatic Text Analysis>>, *Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'89)*. Cambridge, (Massachussets) USA, Juin 1989.

[Shaw et al,1990] Shaw K.T. Paw U, X.J. Zhang, W. Gao, G. den Hartog, and H.H. Neumann. 1990. <<Retrieval of Turbulent Pressure Fluctuations at the Ground Surface Beneath a Forest>>. *Boundary-Layer Meteorology*. Vol. 50, pp.319-338.

[Schoen, 1991] Schoen, E. 1991<<Intelligent Assistance for the Design of Knowledge- based Systems>> Ph.D. Thesis, Stanford University. (1991).

[Henninger 97] Henninger, S. 97 <<An Evolution Approach to constructing effective software reuse repositories>>. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 6 I.2 , Avril 1997

[Sindre et Kalson 1993] Sindre, G. et Kalson 1993. <<A method for software reuse through large component libraries>>. *Proceedings of 5th International Conference on Computing and Information (ICCI'93),* Sudbury, (Canada), pp 464-468. Mai 26-28 1993

[Smyth et Cunningham, 1992] Smyth, B.et Cunningham, P. 1992. <<Déjà Vu: A Hierarchical Cased Based Reasoning System for Software Design>>. *Proceedings of 10 the European conference on Artificial Intelligence (ECAI'92).* Chichester, England: John Wiley & Sons (Vienna), Austria. pp. 587-589)

[Stefania et Manzoni 2001] Stefania Bandini, Sara Manzoni 2001 <<Application of fuzzy indexing and retrieval In case based reasoning for design>> *Proceedings of the 2001 ACM symposium on Applied computing* Mars 2001

[Susan et al, 1998] Susan.E. Sim, C.L.A. Clarke, and R.C Holt, 1998. <<Archetypal Source Code Searching>> *Proceedings of the 6th International Workshop on Program Comprehension*, pp. 180-187, (Ischia), Italie, 1998.

[Sutcliffe 93] A.Sutcliffe, N.A.M.Maiden, 93<<Bridging the requirements gap: policies, goals and domains>>, *7th International Workshop on Software Specification and Design*, Redondo Beach, Californie, USA, Décembre 1993.

[Sutcliffe & Maiden, 1994] Sutcliffe A.G & Maiden N.A.M 1994. <<Requirements Critiquing Using Domain Abstractions>>, *Proceedings of IEEE Conference on Requirements Engineering, IEEE* Computer Society Press, pp.184-193.

[Tautz et Althoff, 1997] Tautz.C, Althoff .K-D, 1997. <<Using Case-Based Reasoning for Reusing Software Knowledge>>, in *Proceedings of the Second International Conference on Case- Based Reasoning Research and development*, 1997, pp. 156-165

[Tracz, 1987] Tracz, W.1987. <<Software reuse: motivators and inhibitors" *Proceedings of the Thirty Second IEEE Computer Society International Conference (COMPCON 87)*, Vol. 7, pp. 358-363. Février 1998

[Vijayan et Storey] Vijayan Sugumaran, Veda C. Storey, 2003. <<A semantic-based Approach to component retrieval>> *ACM SIGMIS Database*, Vol. 34 I.3. Août 2003

[Watson, 2002] Watson, I. 2002. <<Applying Knowledge Management: Techniques for Building Organizational memories>>. *Proceedings of 6th European Conference on Case Based Reasoning (ECCBR 2002)* LNAI 2416, Springer. Vol. 2416, No 6. pp 6-12

[Woodfield et al, 1987] Embley, D. W., & Scott, D. T. (1987) <<Can programmers reuse Software>> *IEEE Software*, pp. 52-59. Juillet 1987

[Yoelle et al, 1994] Yoelle S. Marker, Daniel M. Berry, and Gail E. Kaiser, 1994. <<GURU: Information Retrieval for Reuse, Landmark Contributions in Software Reuse and Reverse Engineering>> edited by P. *Hall, Unicom Seminars, Ltd.*, 1994.

[Young-Jun 2001] Young-Jun Kim 2001. <<Software Reuse Model using Case-Based Object Approach>> *Proceedings of 2nd International Conference on*

Information Technology Based Higher Education and Training, Kumamoto, Japan Juillet 4-6, 2001.

[INRIA] <<Introduction to CBR*Tools>>. Site Internet: http://www-sop.inria.fr/axis/cbrtools/usermanual-eng/Introduction.html