Table des matières

Introduction générale..... 1

1.1 Contexte 1

1.2 Problématique 1

1.3 Contribution..... 2

1.4 structure du mémoire 2

Chapitre I : Maintenance des logiciels……..…….. 4

1. Introduction 4

2. Les logiciels 4

2.1 L'évolution des logiciels 5

2.2 Le génie logiciel 5

2.3 Qualité des logiciels 5

3. Cycle de vie d'un logiciel 5

4. Modèles de cycle de vie d'un logiciel 7

4.1. Modèle en cascade 7

4.2. Modèle en V 7

4.3. Modèle en spirale 8

4.4. Modèle par incrément 8

5. Maintenance des applications 8

5.1. Types de maintenance 9

5.2. Activités de maintenance 9

5.3. Problèmes de maintenance 10

5.4. Techniques de maintenance 10

1. Rétro-ingénieurie 12

2.Réeingenieurie…… 12

3.Analyse d'impact……. 12

Chapitre II : Retro-ingénierie des logiciels……..……13

1. Introduction.....…..13

2. Origine et définition de la rétro-ingénierie……..13

3.Apporte la rétro-ingenieurie logicielle…….15

4.Demarche de la rétro-ingenieurie logicielle……20

5. La démarche de la retro*ingenieurie des logicielle orienté objet……...……..21

Conclusion……..22

Quelques Outils pour la Rétro-ingénierie……..……..…26

I.ptidej……..……..... 26

1. présentation de ptidej……27

2. Fonctionnalités de la solution : PEPS, Ptidej for EcliPSe…….31

3.Création d' une extension pour Peps…….38

II. Argo uml …….……42

1. Introduction…….42

2. Caractéristiques…….42

3. Intégration de l'outil Argo uml a netbeans pour la retro-ingénierie…52

III.Entreprise Architect ……55

1. représentation entreprise architect……..56

Chapitre III : Impact de changement……..…….. 57

1. Introduction 57

2. Définition l'impact de changement..... 57

3. Intérêt de l'Impact de changement orienté objet 57

4. Impact de changement orienté objet..... 58

4.1 Objectif…… 58

4.2 Model conceptuel. 58

4.2.1. Changement……. 58

4.2.2. Liens 58

4.3 Firewall de classe 59

5. Adaptation de model en java..... 59

Conclusion :…….……59

ChapitreI V : Approche proposé……..…60

1. Introduction et contribution……..…….……..…..60

2. Approche proposé……..……60

2.1. Extraction des composants ……60

2.3. Présentation graphique:……..61

2. Réalisation…….61

3.1. Outil de développement…….61

3.2 :mise en œuvre de l'outil……..62

4. Conclusion……66

Conclusion générale et perspective……..…….67

Bibliographie……..……69

ANNEXE……76

Table des figures

I-1 : Spécifications des besoins. . . . . ……. 6

I-2 : Cycle de vie de logiciel . . . . . …….. . 6

I-3 : Modèle en cascade. . . . . ……... 7

I-4 : Modèle en V. . . . . ……... 7

I.5 : Modèle en spirale. . . . . ……. . 8

I-6 : Répartition du temps entre les activités de maintenance. ……11

III-1 :L'analyse de l'impact basée sur la traçabilité entre les artefacts [42]. . . . . ……14

II.1 Modèle générale de la réingénierie logicielle……. .. 19

II.2 Les différentes phases de la réingénierie logiciele……..20

I.1 Le design de Ptidej…….26

I.2 :Situation de Ptidej UI par rapport au projet Ptidej…….……28

I.3 :Copie d'écran de l'interface Swing de Ptidej……..31

I.4 : Architecture du projet « Peps »……37

II.1 :Start menu…….55

II.2 : propriété d'entreprise Architect……56

II.3 : importation de code source…….57

II.4 : extraction de diagramme de classe……58

II.5 : matrice de dépendance ……..58

II.6 : matrice des liens entre les classe de system…….58

IV.1: Schéma globale de l'approche proposé……..…….60

IV.2 :architecture générale ….……..……61

I V.2:menu principal……..63

Iv.3 : chargement de code source……..……..……..63

IV.4 : extraction des composants de system……..……..64

IV.6 Représentation graphique (extrait du graphe global).....…..65

Liste des tableaux

Tableau III-1 : Principaux changements au niveau conceptuel…….15

Tableau II.2 : Exemples deChangement……..……..16

Tableau III.3 : Exemples de changements propres à C++……17

Rapport-Gratuit.com

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

Aperçue de mémoire

1.1 Contexte

La maintenance est la phase la plus coûteuse du cycle de vie du logiciel. Ceci peut, entre autres, s'expliquer par le fait qu'elle est la phase la plus longue qui ne s'ach ve qu'avec le retrait du logiciel. Par ailleurs, le ratio du coût de la maintenance par rapport au coût total du logiciel est passé de 40 à 60 % dans les années 80 à plus de 80% au d but de l'an 2000 [52]. Le coût élevé de la maintenance pousse les chercheurs et les industriels à se focaliser sur cette phase du cycle de vie d'un syst me afin d'essayer de la comprendre pour mieux la contrôler. Pour y arriver, il existe des techniques telles que la visualisation qui sert à montrer les tendances g n r ales du syst me d'une fa on intuitive [41]. Par ailleurs selon les lois d' volution des logiciels, comme indiqu dans [44], un syst me doit tre continuellement changé. Autrement, il apportera moins de satisfaction dans son environnement. En effet, durant la phase de maintenance, les programmeurs sont amenés à faire des changements aux systèmes pour de multiples raisons. Par exemple, les changements réalisés peuvent permettre de répondre à de nouveaux besoins des utilisateurs. On parle dans ce cas, de maintenance perfective. La maintenance peut aussi consister à apporter des changements afin de prévenir les bogues (pr ventive), de s'adapter un nouvel environnement (adaptative), ou pour corriger des erreurs (corrective). Ainsi, les logiciels doivent subir plusieurs changements pour prolonger la durée de leur vie.

1.2 Problématique

Apporter des changements aux syst mes existants s'av re indispensable. Cependant, un changement, même élémentaire, peut avoir des conséquences inattendues sur le fonctionnement global des syst mes. Par exemple, modifier la visibilit d'un attribut peut avoir des résultats non attendus sur le fonctionnement du système. Ces conséquences sont plus n fastes si l'on n'arrive pas d terminer les l ments qui sont affect s par le changement réalisé. Une fois ces éléments détectés, ils doivent être modifiés par les mainteneurs afin de garantir le bon fonctionnement du syst me. Ce besoin d'identifier les éléments affectés par un changement constitue l'objectif de la t che d'analyse de l'impact de changements. L'analyse de l'impact de changements tente de d terminer les parties modifier pour accommoder un changement. Les coûts énormes engendrés par les changements non contrôlés ont amen les chercheurs s'int resser aux techniques d'analyse d'impact des changements depuis longtemps. Par exemple, Weinberg [69] liste les erreurs de type logiciel qui ont coûté

le plus. Ces erreurs sont souvent dues uniquement à un changement dans un chiffre. Un peu moins de vingt ans plus tard, il tait toujours difficile de pr dire l'impact du passage l'an 2000 sur les syst mes informatiques. Ce probl me, connu sous le nom d'Y2K, a incit plusieurs organisations à inclure l'analyse de l'impact de changement explicitement dans le processus de développement et de maintenance des logiciels [7]. Afin de réaliser cette tâche, plusieurs approches furent proposées. On peut les classer principalement en deux catégories : la premi re cat gorie consiste appliquer des techniques d'analyse d'impact sur des mod les et des documents tels que ceux d crivant l'architecture [64] ou partir des besoins des utilisateurs [20]. La deuxième catégorie s'intéresse plutôt aux dépendances dans le code source afin d'identifier les parties du syst me qui doivent possiblement tre modifi es.

Ceci rend ardue la t che de d tection de l'impact des changements. Par ailleurs, les approches qui se limitent à utiliser les dépendances du code source peuvent être plus ou moins efficaces cause des probl mes li s l'identification de ces d pendances d'une mani re efficace [74].

L'objectif de ce m moire est de pallier les probl mes cit s ci-dessus en proposant une démarche qui est fondée sur la rétro-ingénierie des dépendances entre les classes du syst me afin de d tecter l'ensemble des classes affect es d'un syst me suite un changement.

1.3 Contribution

Le but de ce travail d'offrir aux concepteurs et aux mainteneurs un moyen pour mieu comprendre les cons quences des changements qu'ils comptent effectuer sur des programmes objets. En effet, comme l'affirment Pfleeger et Bohner [53], « plus on comprend un changement, mieux on peut le contrôler et donc minimiser son risque ». Pour y parvenir , nous proposons une démarche de la rétro-ingénierie.

Le choix de cette démarche est justifié par : l'analyse de l'impact se fait avant que le système ne soit réellement modifié et donc affecté. Ceci permet de réduire le coût et offre aux mainteneurs le choix entre plusieurs sc narios de maintenance selon l'ensemble des classes prédites comme affectées.

Pour atteindre notre objectif, nous optons pour une solution qui se focalise sur l'analyse du code source apr s modification. En effet, notre d marche consiste analyser le code source pour extraire uniquement les dépendances entre classes sans effectuer des analyses d'impact couteuses. Ces relations peuvent tre les liens d finis dans les diagrammes de classes d'UML (Unified Modeling Language) [59] tels que l'h ritage, les associations ou les agr gations. De plus, nous nous int ressons aux d tails d'invocations entre les classes.

1.4 Structure du mémoire

Le mémoire est organisé comme suit. Le premier chapitre présente une introduction générale sur la maintenance des logiciels

Ensuite, tude d'impact de changement des system orienté objet détecté dans le deuxième chapitre, après dans le troisième chapitre ont a fait un survol sur des principes de la rétro-ingénierie

des logiciels . Enfin, le dernier chapitre consulte la conception et l' laboration d'un outil d'aide la décision dédié à la problématique du choix du changement proposé.

1. Introduction

L'informatisation est le phénomène le plus important de l'époque contemporaine. Il touche toutes les institutions, et jusqu'à la vie personnelle de chacun. Actuellement, l'informatique est au cœur de toutes les grandes entreprises[2]. Le système d'information d'une entreprise est composé de matériels et de logiciels. Plus précisément, les investissements dans ce système d'information se répartissent généralement de la manière suivante : 20% pour le matériel et 80% pour le logiciel [74]. En effet, depuis quelques années, la fabrication du matériel est assurée par quelques fabricants seulement. Ce matériel est relativement fiable et le marché est standardisé. Les problèmes liés à l'informatique sont essentiellement des problèmes de logiciel.

2. Les logiciels

Traduction du terme anglais Software, le logiciel est un ensemble de programmes et d'applications qui permet à un ordinateur ou à un système informatique d'assurer une tâche ou une fonction en particulier (exemple : logiciel de comptabilité, logiciel de production).
Autrement dit, et de façon plus générale, un logiciel est un ensemble de programmes informatiques (du code) mais également un certain nombre de documents se rapportant à ces programmes et nécessaires à leur installation, utilisation, développement et maintenance: spécifications, schémas conceptuels, jeux de tests, mode d'emploi, …

2.1 L'évolution des logiciels

Pour rester utile, un système logiciel doit constamment évoluer. Ceci est principalement dû à l'accroissement des exigences de ses utilisateurs. Faire évoluer un système est un véritable défi impliquant la satisfaction des besoins suivants : découvrir la partie du logiciel concernée par cette évolution, trouver un moyen de la réaliser sans régression du système et enfin, pouvoir valider cette évolution.

2.2 Le génie logiciel

Le génie logiciel est l'ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi. Autrement dit, le génie logiciel est "l'art" de produire de bons logiciels, au meilleur rapport qualité/prix. Il utilise pour cela des principes d'ingénierie et comprend des aspects à la fois techniques et non techniques: le génie logiciel est basé sur des méthodologies et des outils qui permettent de formaliser et même d'automatiser partiellement la production de logiciels, mais il est également basé sur des concepts plus informels, et demande des capacités de communication, d'interprétation et d'anticipation. De fait, la "crise du logiciel" n'est toujours pas résolue. Le génie logiciel reste un "art" qui demande de la part

de l'informaticien une bonne formation aux différentes techniques (le "savoir"), mais également un certain entraînement et de l'expérience (le "savoir faire").

2.3 Qualité des logiciels

Si le génie logiciel est l'art de produire de bons logiciels, il est nécessaire de fixer les critères de qualité d'un logiciel:

Fiabilité (correction et robustesse) fonctionné dans des conditions anormales.
Extensibilité (maintenance) : facilité avec laquelle un logiciel se prête à sa maintenance, c'est-à-dire à une modification ou à une extension des fonctions qui lui sont demandées.
Ergonomie (simplicité et rapidité d'emploi, personnalisation): aptitude à communiquer de façon efficace avec l'utilisateur.

Efficacité : utilisation optimale des ressources matérielles.

Portabilité (éviter l'assembleur et les langages trop confidentiels) : facilité avec laquelle un logiciel peut être transféré sous différents environnements matériels et logiciels.
Réutilisabilité : aptitude d'un logiciel à être réutilisé, en tout ou en partie, dans de nouvelles applications.
Compatibilité : facilité avec laquelle un logiciel peut être combiné avec d'autres logiciels.
Vérifiabilité : facilité de préparation des procédures de test.
Coût: coût monétaire occasionné par le développement et la maintenance d'un logiciel lors de son cycle de vie.

Ces différents critères de qualité ne sont pas toujours compatibles ni même réalisables, et il est nécessaire de trouver des compromis. Dans tous les cas, les objectifs de qualité doivent être définis pour chaque logiciel, et la qualité du logiciel doit être contrôlée par rapport à ces objectifs.

3. Cycle de vie d'un logiciel

Le cycle de vie d'un logiciel est l'ensemble des étapes du développement d'un logiciel, de l'établissement des besoins du client jusqu'à l'achèvement du logiciel en tant que produit commercial. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimé, et la vérification de processus de développement.

Le cycle de vie permet de détecter les erreurs au plut tôt et ainsi de maitriser la qualité de logiciel, les détails de sa réalisation et le cout associées.
Le cycle de vie comporte généralement les activités suivantes :

- **Phase de spécification des besoins** : déterminer les besoins du logiciel (spécifications

générales, spécifications fonctionnelles, spécifications d'interface). Les spécifications des besoins servent à définir ce que doit faire le logiciel et non comment il doit être fait.

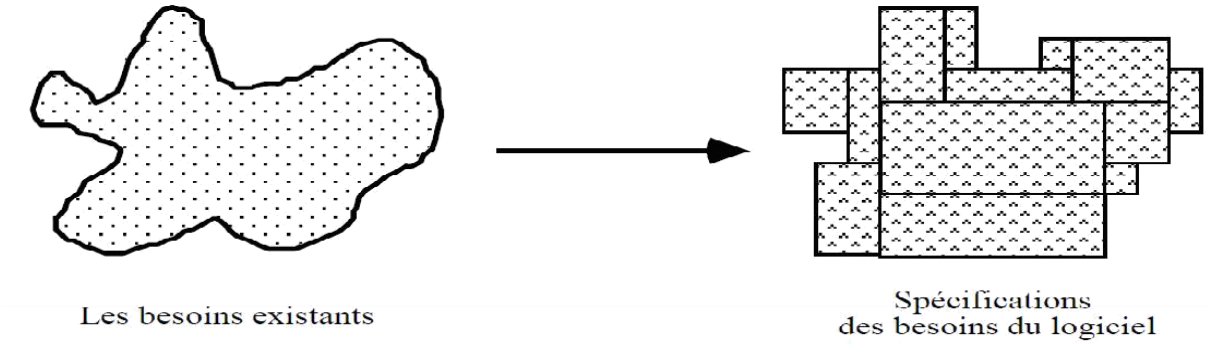


Figure I-1 : Spécifications des besoins[75]

¬ **Phase de conception du système et du logiciel :** En partant de la spécification des besoins, on décompose le système en deux parties : matériel et logiciel. C'est le processus qui consiste à présenter les diverses fonctions du système d'une manière qui permettra d'obtenir rapidement un ou plusieurs programmes réalisant ces fonctions.

¬ **Phase de réalisation et tests unitaires :** Lors de cette étape, on réalise un ensemble d'unités de programme écrites dans un langage de programmation exécutable. Les tests unitaires permettent de vérifier que ces unités répondent à leurs spécifications.

¬ **Phase de test du système :** On intègre les unités de programme et on réalise des tests globaux pour être sûre que les besoins logiciels ont été satisfaits**.** Le système est alors livré au client.

¬ **Phase de maintenance :** Une fois que le produit est accepté, il passe en phase de maintenance jusqu'à son retrait. Normalement (mais pas nécessairement) ceci est la plus longue étape du cycle de vie, l'activité de maintenance consiste à corriger les erreurs qui n'ont pas été découvertes lors des étapes antérieures du cycle de vie, à améliorer la réalisation des unités du système et à augmenter les fonctionnalités du produit au fur et à mesure que de nouveaux besoins apparaissent.

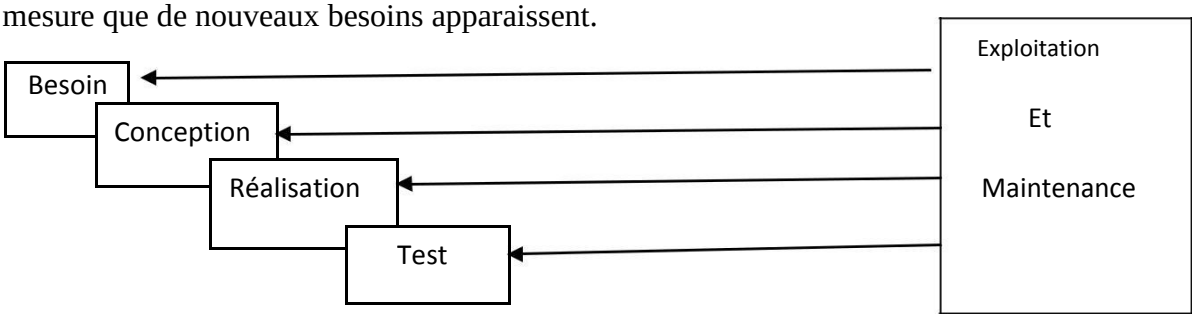


Figure I-2 : Cycle de vie de logiciel[75]

4. Modèles de cycle de vie d'un logiciel

4.1. Modèle en cascade

Dans ce modèle, le principe est très simple : chaque phase se termine à une date précise par la production de certains documents ou logiciels. Les résultats sont définis sur la base des interactions entre étapes et activités, ils sont soumis à une revue. Ce modèle est présenté en (Figure I-3) suivante :

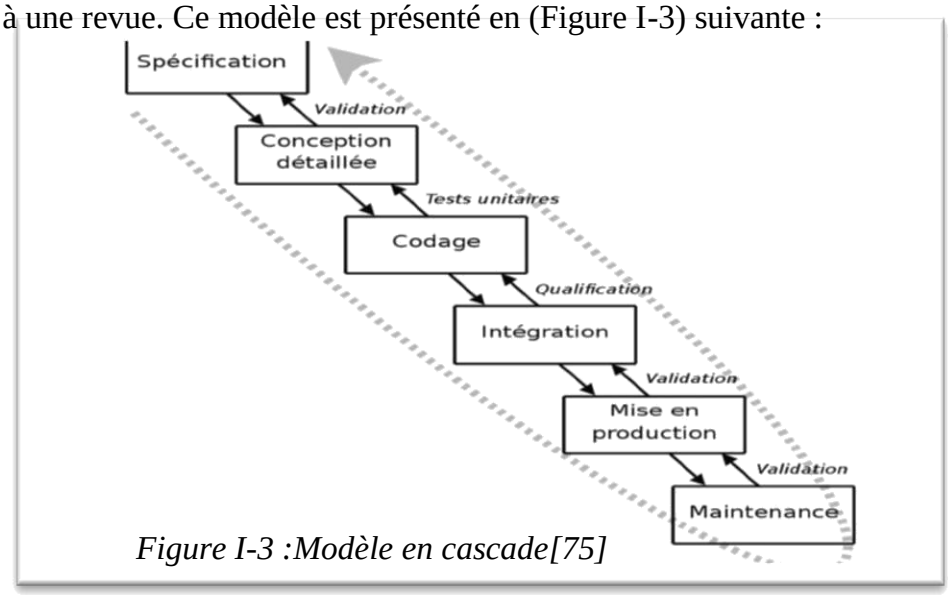


Figure I-3 :Modèle en cascade[75]

4.2. Modèle en v

Le modèle de cycle de vie en V (Figure I-4) part du principe que les procédures de vérification de la conformité du logiciel aux spécifications doivent être élaborées dès les phases de conception [89].

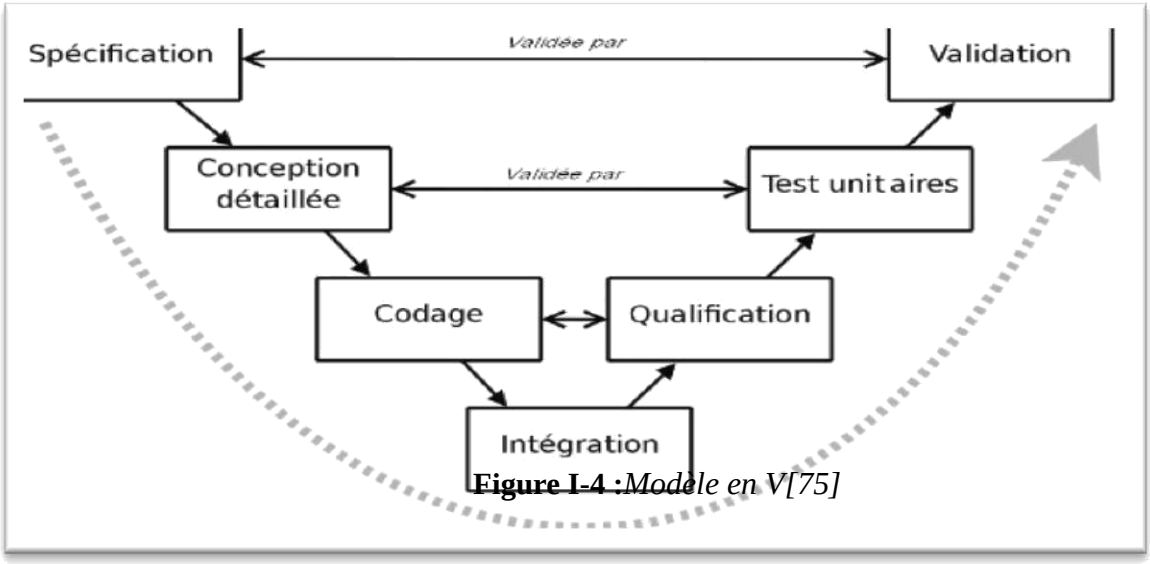


Figure I-4 :*Modèle en V[75]*

4.3. Modèle en spirale : chaque cycle de la spirale se déroule en quatre phases :

a. Détermination des objectifs, des alternatives, des contraintes à partir des résultats du cycle précédent et pour le premier à partir d'une analyse préliminaire des besoins;

b. Analyse des risques, évaluation des alternatives et, éventuellement maquettage ; c.

Développement et vérification de la solution retenue, un modèle «classique » (cascade ou en V) peut être utilisé ici ;

d. Revue des résultats et vérification du cycle suivant.

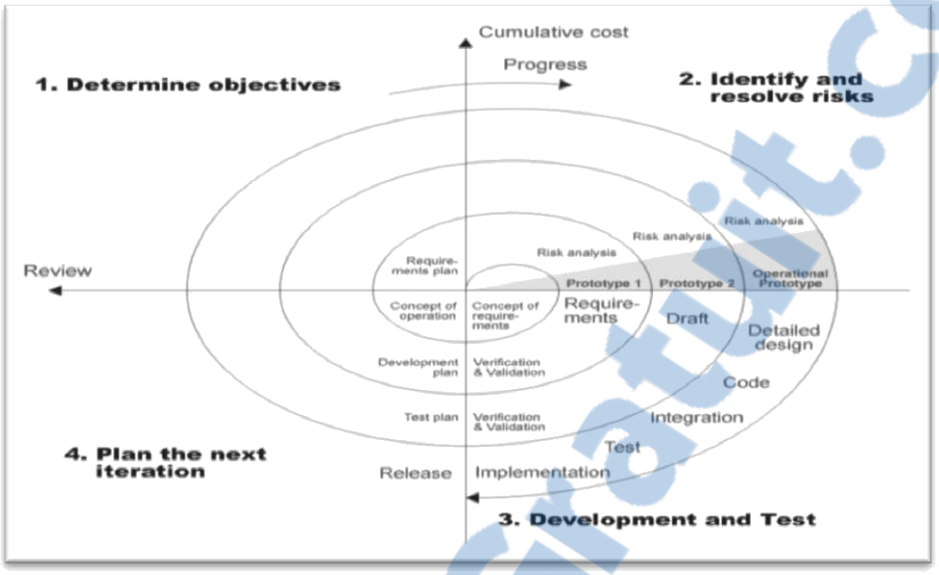


Figure.5 :Modèle en spirale[75]

L'analyse préliminaire est affinée au cours des premiers cycles. Le modèle utilise des maquettes exploratoires pour guider la phase de conception du cycle suivant. Le dernier cycle se termine par un processus de développement classique.

Ce modèle a été moins expérimenté que les deux précédents. Sa mise en œuvre demande de grandes compétences et devrait être limitée aux projets innovants à cause de l'importance qu'il accorde à l'analyse des risques. Néanmoins, ce dernier concept peut être appliqué aux autres modèles.

4.4. Modèle par incrément : Dans les modèles précédents, un logiciel est décomposé en composants développés séparément et intégrés à la fin du processus. Dans les modèles par incrément, un seul ensemble de composants est développé à la fois, des incréments viennent s'intégrer à un noyau de logiciel développé au préalable. Chaque incrément est développé selon l'un des modèles précédents [76].

5. Maintenance des applications

La compréhension d'un programme représente l'une des tâches les plus importantes du processus de maintenance. Les personnes chargées de la maintenance épuisent 40 à 60% de leurs temps à lire le code et à essayer de comprendre sa logique. C'est une activité laborieuse qui augmente le coût de la . En effet, l'équipe qui a développé le système n'est pas toujours celle qui assure sa maintenance, les documents de spécifications sont parfois incomplets et le code source constitue souvent la seule source d'informations fiable. La plupart des problèmes associés à la maintenance du logiciel sont causés par la

Rapport-Gratuit.com

méthode utilisée pour concevoir et développer le système. De plus, la maintenabilité est un facteur de qualité qui n'est pas incorporé dans le processus de développement.

5.1. Type de maintenance :

Généralement on considère trois types de maintenance : maintenance corrective adaptative, et perfective. et d'autre auteurs comme Glass et Noiseux dans [81] ont proposé d'autres types de la maintenance préventive et urgente(emergency maintenance). Nous définissons ces différents types dans le paragraphe suivant :

a. Maintenance corrective

C'est le type de maintenance le plus évident. On s'assure qu'un logiciel en opération continue à satisfaire ses spécifications. En pratique, c'est l'activité de corriger les erreurs résiduelles qui auraient dues être découvertes lors de la phase de test. Comme on ne pourra jamais découvrir toutes les erreurs lors de la phase de test, ce type de maintenance est inévitable. Il ne constitue toutefois qu'environ 17% de l'effort total de maintenance [78].

b. Maintenance adaptative

Ce type de maintenance consiste à modifier un logiciel en réponse à un changement intervenu dans son environnement logiciel ou matériel. Le standard IEEE la définit comme suit [80]: "*modification of a software product performed after delivery to keep a computer program usable in a changed or changing environnement*". Ainsi, la portabilité d'un logiciel d'une plate-forme (système d'exploitation ou processeur) à une autre relève de cette catégorie de maintenance. La maintenance adaptative compte pour environ 18% de l'effort total de maintenance [78].

c. Maintenance perfective

La maintenance perfective fait référence aux modifications d'un programme en vue d'augmenter ses performances ou ses fonctionnalités. Ce type de maintenance s'effectue généralement à la demande des utilisateurs qui veulent toujours avoir plus de fonctionnalités. C'est la principale activité de maintenance, elle constitue entre 60 à 70% de l'effort total de maintenance [78].

d. Maintenance urgente

C'est un cas particulier de maintenance corrective. L'IEEE en donne la définition suivante [80]: " *unscheduled corrective maintenance performed to keep a system operational* ".
ce type de maintenance permet de garder les system opérationnel suite a une anomalie non programmé.

e.Maintenance préventive

consiste a modifier un programme pour prévenir des difficultés futurs .

par exemple modifier un programme pour augmenter sa maintenabilité .
ce type de maintenance est relativement rare.

La maintenance doit être donc d'avantage percue comme un procesus d'évolution d'un programme que la simple détection et correction ponctuelle.

Tout programme est appelé a évolué pour satisfaire de nouvelle exigence (maintenance perfective) et s'adapter a un environnement changeant sans cesse (maintenance adaptative).ces deux activité totalise 80% de l'effort globale de la maintenance.

5.2. Activité de maintenance

Carma McClure [82] observe que l'activité d'un programmeur lors de la maintenance peut se ramener à trois tâches principales : compréhension du code, modification, et revalidation.

a. Comprendre le code et les changements à effectuer est une tâche essentielle pour modifier un programme. Le programmeur doit avoir une complète compréhension du programme, sinon le risque d'introduire des erreurs après modification est grand.

b. Le programmeur doit modifier le programme pour que ce dernier soit conforme à ses spécifications actuelles (maintenance corrective) ou nouvelles (maintenance adaptative ou perfective). Il doit s'assurer que ses modifications n'altèrent pas l'intégrité du système ou n'augmentent pas sa complexité. Il doit également faire attention aux effets de bord (introduction d'autres erreurs).

c. Après toute modification, un programme doit être re-testé (revalidé) pour s'assurer que les modifications ont été correctement effectuées et qu'il n'y pas eu d'effets de bord.

5.3. Problèmes de maintenance

La maintenance des logiciels se heurte à un certain nombre de difficultés. Martin et Osborne dans **[**77**]** ont relevé quelques facteurs qui font de la maintenance une activité délicate :

- Mauvaise conception des programmes.
- Utilisation de code non-structuré
- Utilisation de plusieurs langages de programmation dans un même programme (logiciel).

Mais une difficulté majeure est celle que pose la compréhension d'un programme (program understanding).
La compréhension de programme comprend la lecture de la documentation, l'analyse du code source et l'identification des changements à effectuer. Selon McClure[78]**,** des trois activités de la maintenance, la compréhension de programme est celle qui consomme le plus de temps (voir figure I.1).

5.3.1. Compréhension de programme

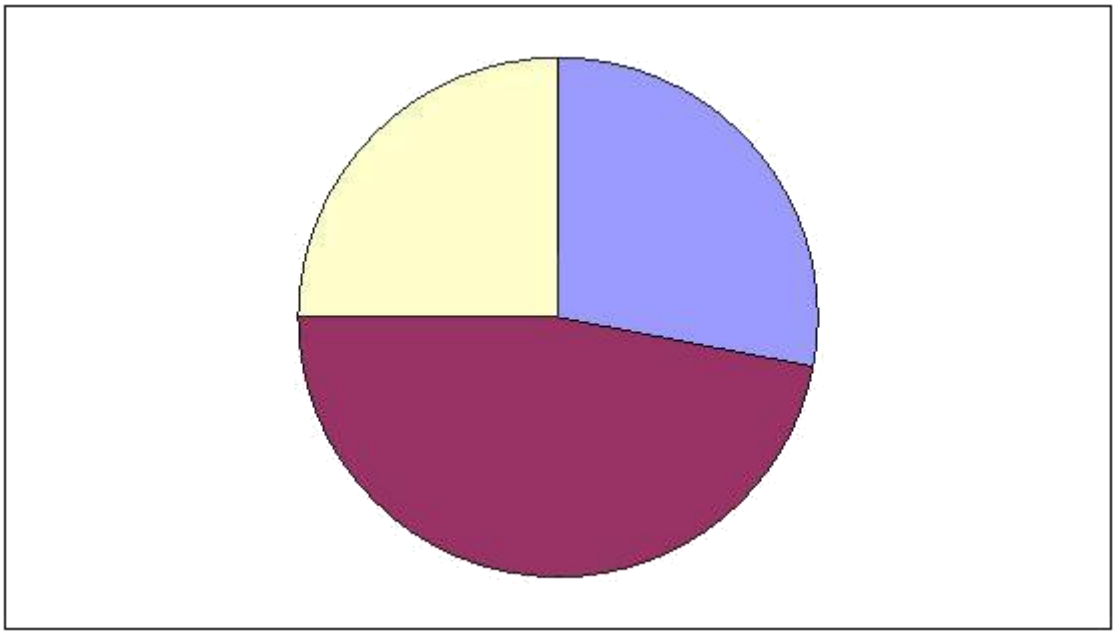
Beaucoup d'organisations sont confrontées au problème de maintenir des logiciels désuets, s'exécutant sur une grande variété de plate-forme; écrit dans des langages obsolètes et qui ont fait l'objet de nombreuses opérations de maintenance, ce qui a conduit à une dégradation de leur structure. La maintenance de tels systèmes est devenue très complexe et très coûteuse. Assez souvent, un programmeur qui fait la maintenance d'un logiciel n'a pas participé à sa phase de développement. Parfois la documentation dont il

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

dispose est incomplète, inadaptée, voire inexistante. Il passe de ce fait un temps important à essayer de comprendre les intentions de tous ceux qui avant lui sont intervenus sur le code.

La compréhension de programme comprend la lecture de la documentation, l'analyse du code source et l'identification des changements à effectuer. Selon McClure [78], des trois activités de la maintenance, la compréhension de programme est celle qui consomme le plus de temps . Une autre difficulté liée à la compréhension et la modification d'un système est l'évaluation de l'impact d'un changement et des erreurs qu'il peut introduire dans le code. Toujours selon McClure [78], le risque d'introduire une erreur quand seulement 200 lignes de code sont changées varie entre 35 et 75%.



Modification 25%

Compréhension 47%

Vérification 28%

FigureI-6 : *Répartition du temps entre les activités de maintenance*

L'objectif de la compréhension de programme est d'acquérir suffisamment de connaissances sur un code pour pouvoir le faire évoluer de façon disciplinée. Comprendre un système consiste essentiellement en l'identification de ses composants et des relations qui existent entre eux [88]. Pour faciliter la compréhension d'un programme, ce processus d'identification doit être supporté par des techniques appropriées.

5.4. Techniques de maintenance

Dans le domaine du logiciel, une plus grande partie de l'effort était accordée à la phase de développement. L'objectif était de livrer un produit qui satisfait les besoins dans les délais et dans les limites du budget sans se préoccuper

de la qualité du logiciel. Avec l'augmentation des coûts de la maintenance, de nombreuses techniques ont vu le jour. Ci-dessous, nous présentons brièvement quelques-unes de ces techniques.

1. **Rétro ingénierie :** Généralement pour concevoir un logiciel, on passe de la conception au code. La rétro ingénierie prend le chemin inverse. Chikofsky et Cross [87] ont défini la rétro ingénierie par: *"reverse engineering is the process of analyzing a subject system to identify the system's component and their interrelationships and create representations of the system in another form or at a higher level of abstraction".* C'est le processus d'analyse d'un logiciel pour identifier ses composants et leurs relations dans le but de générer une représentation du système à un plus haut niveau d'abstraction.

2. **Réingénierie :** Selon Chikofsky et Cross, la réingénierie consiste en : *"the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form "* [88]. C'est un processus composé de l'analyse du système et puis sa reconstitution. Ces deux étapes sont effectuées par les techniques de rétro ingénierie et de *"Forward Engineering"*.

Cette dernière consiste a transiter d'une représentation de haut niveau obtenue par l'étape précédente pour produire un nouveau système .certaine auteurs ajoutent une troisième étape la restructuration .elle transforme la structure interne d'un logiciel sans changer le niveau d'abstraction et sans modifier ou ajouter de nouvelles fonctions .en outre elle permet une meilleurs compréhension et facilite sa maintenance.

3. analyse d'impact : pratiquement lorsqu'une proposition de changement parvient a l'équipe de maintenance elle est transformer en terme de logiciel afin de décider si le problème doit être retenue ou rejeté.une fois acceptez il faut Localiser l'origine de l'anomalie.

L'analyse d'impact de changement s'impose a cette étape afin de (i)déterminer tous les composants de system qui doivent être changé en conséquence de changement initiale, (ii)estimer les ressources pour implanter le changement .

Ce résultats permet de donner aux de prendre une décision sur la meilleur solution à mettre en œuvre ou annuler la requête de changement.

Conclusion

Dans ce chapitre, nous avons présenté la maintenance, ses différents types ainsi que ses Techniques et les problèmes rencontrés durant cette phase, qui est une phase importante dans le cycle de vie d'un logiciel.

Dans le chapitre suivant, nous abordons l'impact de changement qui est l'une des techniques souvent préconisée dans la phase de maintenance de logiciel.

1. Introduction

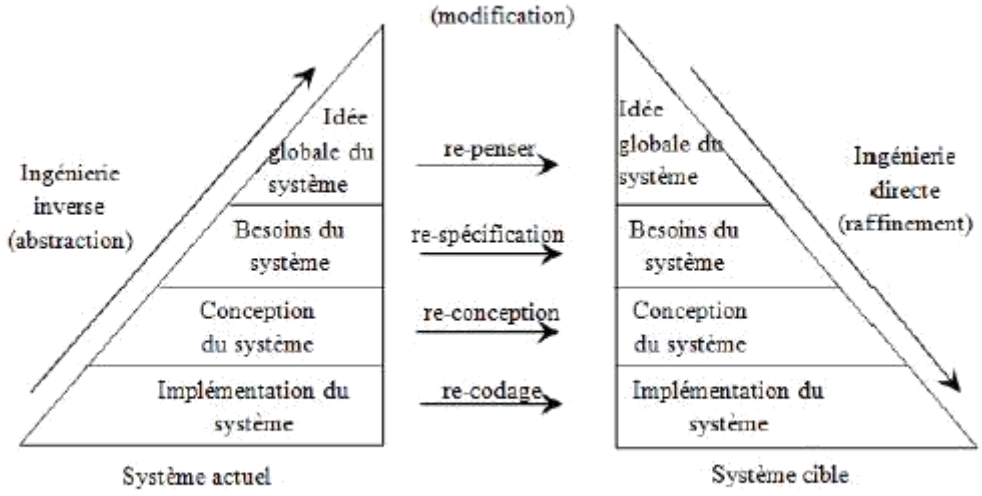
Dans ce chapitre nous explorons la rétro-ingénierie en présentant ses différents domaines d'application dans la littérature. Nous nous sommes intéressés particulièrement à la rétro-ingénierie liés au domaine du génie logiciel. Nous déterminons ses apports et ses étapes principales, ses approches existantes et les différents niveaux d'abstraction visés par la rétro-ingénierie orientée objet.

2. Origine et Définition de la rétro-ingénierie

D'après [4], la rétro-ingénierie revient aux années de la révolution industrielle. Elle est semblable à la recherche scientifique où les chercheurs essayent d'inventer des plans et des cartes aux phénomènes naturels. La seule différence est que la rétro-ingénierie concerne les objets créés par l'homme (des appareils mécanique, des composants électronique, des programmes informatique, …). D'une façon générale, la rétro-ingénierie est le processus d'extraction de connaissances ou de concevoir des représentations à partir de tout ce qui est construit par l'homme.

Dans le contexte de l'ingénierie logicielle, le terme rétro-ingénierie a été défini en 1990 par Chikofsky et Cross dans [3] comme le processus d'analyse d'un système pour (1) identifier les composants du système et leurs interrelations, et (2) créer des représentations du logiciel dans une autre forme ou dans un niveau d'abstraction plus élevé. La rétro-ingénierie a été traditionnellement vue comme un processus en deux étapes : l'extraction des informations et l'abstraction. L'étape d'extraction des informations analyse les artefacts du logiciel pour recueillir des données brutes, tandis que l'étape d'abstraction crée des vues et documents orientés utilisateur à partir des données brutes [25]. Chikofsky et Cross ont présenté une structure de base pour les outils implémentant la rétro-ingénierie .

Le logiciel étudié est analysé et les résultats de cette analyse sont stockés dans une base d'informations. Par la suite, ces informations sont utilisées pour produire différentes vues du logiciel, comme les diagrammes, les rapports, et les métriques.[34]



IGURE 1.2: Modèle général de la réingénierie du logiciel source originale

3. Apports du rétro-ingénierie logiciel

La rétro-ingénierie est très utile aux développeurs des logiciels. En effet, ils l'utilisent comme un moyen de coopération et collaboration autour des logiciels qui ne disposent pas[36]
(ou qui disposent partiellement) de la documentation. Ils peuvent l'utiliser aussi pour étudier et améliorer les logiciels concurrents.

De plus, la rétro-ingénierie permet d'évaluer la qualité et la robustesse d'un logiciel. Elle est également le moyen essentiel dans les différentes taches de l'étape de maintenance qui représente une étape très importante dans le cycle de vie de logiciel. En effet, une enquête effectuée aux USA en 1986 auprès de 55 entreprises a révélé que 53% du budget total d'un logiciel est affecté à la maintenance [27]. La rétro-ingénierie facilite les taches de maintenances car elle permet de proposer aux personnes intervenant dans la maintenance des représentations plus abstraites qui leur permet de comprendre le logiciel pour le tenir à jour en fonction des nouveaux besoins des utilisateurs.

En plus de faciliter la maintenance de logiciels, six autres objectifs de la rétro-ingénierie ont été identifié dans [3] :

· Faire face à la complexité.
· Générer différentes vues.
· Recouvrir des informations perdues.
· Détecter l'effet de bord.
· Synthétiser des abstractions de haut niveau.
· Faciliter la réutilisation.

4. Démarche de réingénierie logicielle

En référence à la définition traduite et donnée par Marin Flower[1] dans « *Refactoring – Improving the Design of Existing Code* » , La ré-ingénierie logicielle est un processus ayant pour objectif de transformer et améliorer un logiciel à partir du modèle existant sans toutefois modifier son comportement externe ». L'objectif de cette démarche est donc, au mieux, d'en conserver les performances actuelles, mais surtout de lui apporter de l'évolutivité.

Cette démarche opère en plusieurs étapes, qui peuvent varier plus ou moins selon le contexte et les objectifs souhaitables[38]. Cependant le diagramme ci-dessous donne une vision théorique des différentes phases de ré-ingénierie inspirant l'approche menée dans ce projet.

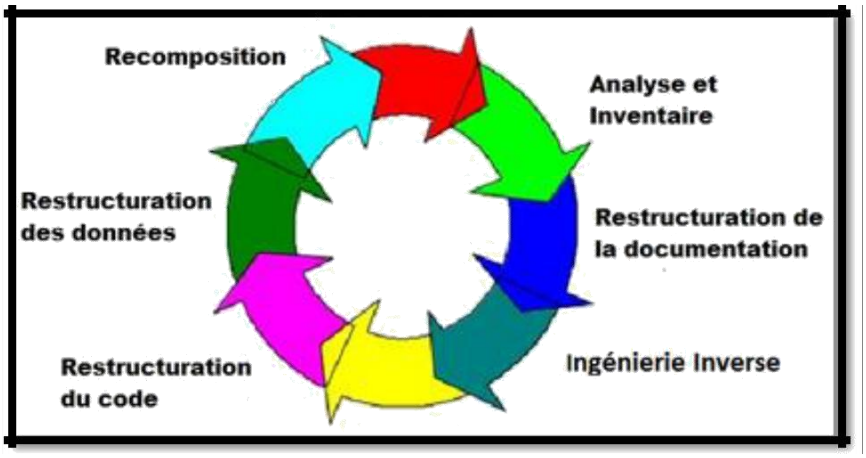


Figure III.2 Les différentes phases de la réingnierie logicielle (ref:fr.wikipedia.org/wiki/Réingénierie_logicielle)

Comme on peut le voir sur le diagramme de l'illustration 2, le processus est une roue. La démarche de ré-ingénierie peut-être incrémentale. Il est tout à fait possible d'effectuer le processus complet sur une partie du système – cas de notre étude -, puis de réitérer la démarche sur une autre partie. C'est notamment le type d'approche préconisée par les méthodes Agiles[1]

Comme évoqué précédemment, ce diagramme propose une vision théorique que l'on adapte en fonction de nos besoins.

- **Phase d'analyse et d'inventaire**

  Cette phase est la première dans notre cas aussi. Il s'agit là de comprendre le fonctionnement du logiciel. Une prise en mains des technologies utilisées suivie de l'analyse du code et des structures de données.

- **Phase de construction de la documentation**

  Au fil de l'analyse décrite ci-dessus la documentation est apportée et enrichie. D'une part l'intégration de commentaires dans le code, permettant notamment de générer une documentation Java. D'autre part une documentation rédigée sous forme de descriptions des principes de fonctionnement du système ou de descriptions de certains éléments importants.

- **Phase de restructuration du code**

Cette phase permet de réorganiser le code, sans impacter le fonctionnement du système. Des tests sont donc effectués en parallèle afin de vérifier que l'on ne régresse pas fonctionnellement. Il s'agit notamment de scinder les méthodes et les classes trop longues, de les renommer si besoin, d'éliminer les redondances ou encore de rendre le code plus générique pour une meilleure évolutivité dans le temps.

- **Phase de reconstruction des données**

Il s'agit ici d'intervenir sur la structure des données. En fonction de la restructuration du code réalisé précédemment, la structure de données pourra être modifiée. Dans le cadre de cette étude, peu de modifications sont envisagées à ce niveau. Cependant, certaines évolutions du système peuvent mener à l'intégration de nouveaux éléments dans la structure de données.

- **Phase d'intégration de nouveaux éléments**

Il s'agit là d'apporter des nouvelles fonctionnalités. Celles-ci peuvent avoir un effet sur la structure de données, plus en tant qu'ajout de nouveaux éléments qu'en tant que modification d'éléments existants[39].

Même s'il n'est pas nécessaire d'avoir effectué toutes les phases précédentes sur l'ensemble du système, il est évidement préférable que les évolutions s'appuient sur des « bases saines », soit un code évolutif. Cela ne signifie pas non plus que ce même code ne doive pas un jour ou l'autre repasser par une phase de restructuration de code.

5. La rétro-ingénierie des logiciels orientés objet :

[2] Methodes Agiles : methodes de développement et de gestion de projets informatiques se voulant plus pragmatiques que les méthodes traditionnelles. Elles cherchent notamment à impliquer au maximum les différents acteurs du projet. Elles reposent sur le principe de cycles de développements incrémentaux, adaptatifs et itératifs. (Source : Wikipedia.fr)

Chapitre II : Rétro-ingénierie des logiciels

La compréhension des d'applications orientées-objet est une tâche difficile qui se doitde surmonter les particularités de ce paradigme. En effet, même si ses particularités sont les points forts de ce type de programmation, elles élèvent l'analyse et la compréhension des applications orientées-objet à un autre niveau de difficulté. Le cas du polymorphisme et la liaison dynamique rendent les traditionnels outils d'analyse inadéquats.
L'objectif essentiel de la rétro-ingénierie orientée-objet est de voir le système par une vue descriptive dans le niveau d'abstraction adéquat [6]. Pour mieux comprendre les fonctionnalités d'un logiciel il est très nécessaire de choisir le bon niveau d'abstraction [23] :

Niveau instruction

Ce niveau inclut l'exécution de chaque instruction du code. La majorité des outils ne supporte pas cette vue à l'exception des débogueurs. Ce niveau d'abstraction va bien avec les activités spécifiques de maintenance comme la fixation des bugs.

Niveau communication inter-thread

Ce niveau s'occupe de visionner les interactions entre les threads du système. Peu sont les outils qui prisent en charge ce type d'interaction, et si c'est le cas, souvent les autres niveaux sont omis comme dans Jinsight [1O].

Niveau objet

Ce niveau concerne la visualisation des interactions des méthodes entre les objets.
Ce niveau peut être utile pour détecter les fuites de mémoire et autres étranglements de performance. La majorité des outils supporte ce niveau.

Niveau classe

Dans ce niveau, les objets de la même classe sont substitués par le nom de leurs classes.
Ce niveau, qui est supporté par la majorité des outils, va bien avec les activités qui requièrent de haut-niveau de compréhension du comportement du système, tels que le recouvrement de la documentation, et la compréhension de quelles classes implémentent une caractéristique particulière.

Niveau architectural

Ce niveau consiste à grouper des classes dans des clusters et montrer comment les composants du système inter-actent les uns les autres[56].

Tutoriel sur quelques outils de la retro-ingénierie

I.Ptidej :

· ce jour et à notre connaissance, il n'existe aucun outil qui permet d'obtenir automatiquement un modèle abstrait précis à partir du code source d'un programme.
Même les outils industriels comme Rational Rose sont incapables de retro-concevoir le code source d'un programme précisément et se contentent d'en fournir une représentation graphique. Un ensemble de définitions a été développé et implanté dans un outil.
– L'implantation et le développement d'un ensemble de définitions dans l'outil Ptidej qui permet de
• Analyser statiquement le code source d'un programme

• Identifier des relations abstraites entre classes, telles que les relations d'association, d'agrégation et de composition
• Appeler un solveur de contraintes pour identifier les microarchitectures similaires à la solution d'un patron de conception donné
– L'intégration de cet outil avec l'environnement de développement Eclipse

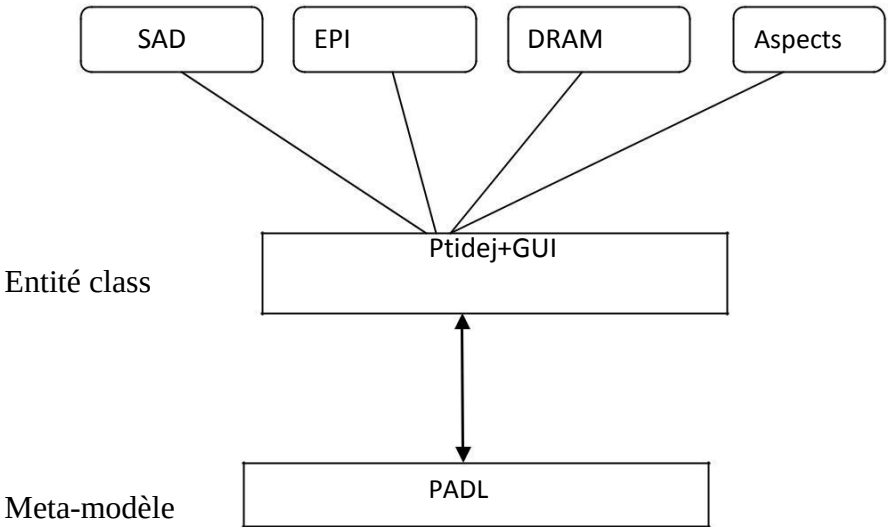


Figure IV.1 : Le design de Ptidej

Quatre outils ont été ajoutés à Ptidej:

- SAD: (Software Architectural Defects)un outil de détection et de correction Des défauts de l'architecture logicielle.
- EPI: (Efficient Pattern Identification)outil d'identification efficace D'événements de modèles de conception dans un pro-
gramme.
- DRAM: (Adjonction relationnelle dynamique marix)outil de visualisation des Données statiques et dynamiques de programmes avec matrice d'adjacence.

- Aspects: un outil de modélisation et de com Mettre des métriques sur l'abstraction orientée

Tous ces outils visent à évaluer la qualité
Des programmes orientés objet et aider à maintenir -
Et les développeurs dans l'amélioration de leurs programmes
En soulignant la conception de leurs programmes.

Eclipse

La plate-forme de développement Eclipse est devenue en quelques années la plate-forme de référence dans l'industrie comme dans le milieu universitaire. Elle offre un environnement cohérent pour intégrer des extensions et faciliter leur accès et leur utilisation par les développeurs.
Eclipse est composé de trois parties
– Workbench
– Plug-in Development Environment (PDE)
– Standard Widget Toolkit (SWT)
Eclipse offre aussi une interface pour travailler avec CVS
La suite d'outil de rétro-conception Ptidej est un ensemble d'outils et de modèles avancés permettant l'analyse et la manipulation du code source de systèmes logiciels et d'autres sources de données pour en évaluer la qualité et en faciliter la compréhension.
A l'heure actuelle, il n'existe pas de pont entre Eclipse et Ptidej alors que les objectifs du projet Ptidej rentrent parfaitement dans les objectifs de la plate-forme Eclipse.

1.Présentation de ptidej :

Le projet Ptidej offre une suite d'outils extensibles pour reconstruire le modèle d'un programme et pour l'analyser à l'aide de plusieurs interfaces utilisateur. Il comporte 297+ packages, 2075+ classes et 340+ interfaces pour plus de 112 300 lignes de code Java. Il est activement développé et maintenu depuis 2001, suivant des conventions de codage stricts et des standards de qualité.
La suite Ptidej est activement utilisée à travers le monde pour effectuer des analyses de programmes informatiques et pour créer divers outils.

Parmi les projets qui constituent la suite Ptidej, on compte les ensembles de projets :

Caffeine Dynamic trace generator and analyser for Java using Prolog

Rapport-Gratuit.com

DRAM Dynamic Relationships with Adjacency Matrices
OADynPPaC Outils pour l'Analyse Dynamique et la mise au Point de Programmes de Programmes avec Contraintes
CPL Common Ptidej Library
PADL Pattern and Abstract-Level Description Language
EPI Efficient Pattern Identification
POM Primitives, Operations, Metrics
JChoco Java library to build explanation-based constraints solvers
Ptidej Solvers Explanation-based constraint solvers to identify (complete and approximate) occurrences of design patterns in PADL models)
SAD Software Architectural Defects
Ptidej Pattern Trace Identification, Detection and Enhancement in Java
Ptidej UI Ptidej graphical interfaces project set
Programmes tiers DOT, InfoVis, SugiBib
P-MART Pattern-like Micro-Architecture Repository

Etapes a suivre :
– Implanter l'interface graphique de Ptidej dans Eclipse
– Développer Ptidej pour atteindre une intégration complète
– Implanter toutes les fonctionnalités disponibles
– Visualiser le modèle correspondant à l'aide d'un éditeur de diagrammes directement dans Eclipse
– Afficher les résultats de l'appel d'un solveur de contraintes pour identifier les micro-architectures similaires à la solution d'un patron de conception donné
– Faire de tests JUnit

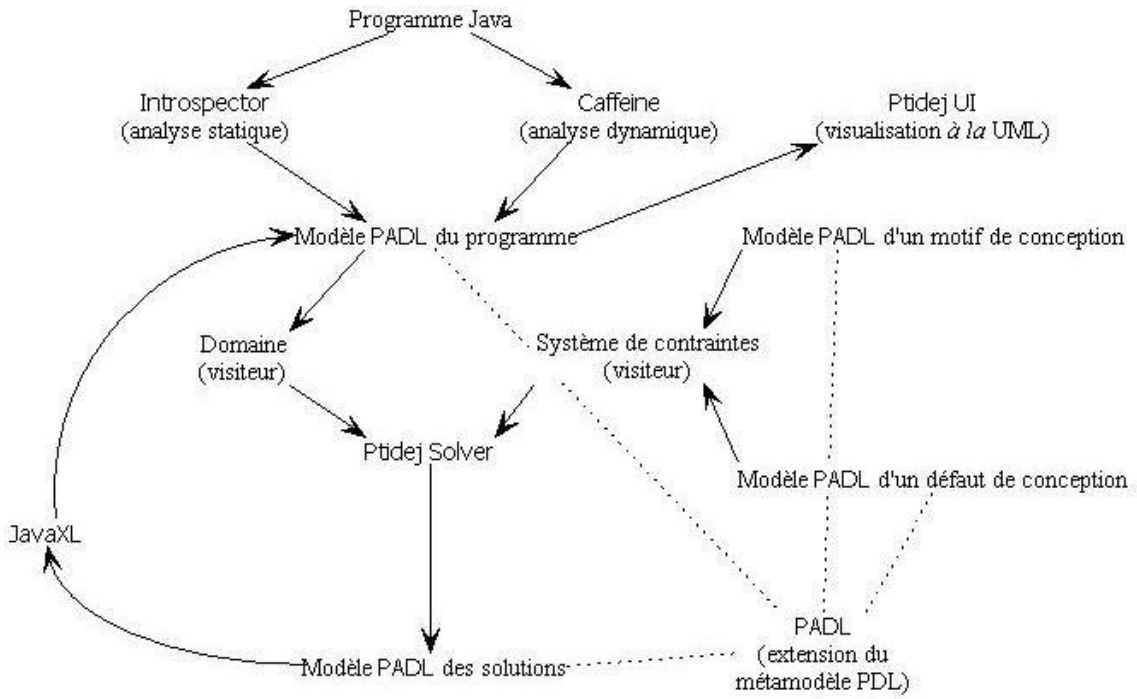


Figure IV.2 :Situation de Ptidej UI par rapport au projet Ptidej

La structure des modules d'extension par défaut
– Le dossier bin
– Le dossier src
– Les éléments du module d'extension à la racine
– Le fichier .template
– Le fichier build.properties
– Le fichier plugin.xml (Manifeste)

Voici un exemple de manifeste :

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
id="idduplugin"
name="nom du plugin"
version="1.0.0"
provider-name="Université de Montréal"
class="test.testPlugin">
<runtime>
<library name="boot.jar"/>
</runtime>
<requires>
<import plugin="org.eclipse.core.resources"/>
<import plugin="org.eclipse.ui"/>
</requires>
<extension point="org.eclipse.ui.editors">
<editor
name="Exemple d'éditeur XML"
icon="icons/sample.gif"
extensions="e"
ContributorClass="org.eclipse.ui.texteditor.
BasicTextEditorActionContributor"
class="test.editors.XMLEditor"
id="test.editors.XMLEditor">
</editor>
</extension>
</plugin>
```

Les fonctions implantées
1. Traitement des fichiers .class
a. Les analyser dans le même éditeur de diagrammes b.
Les analyser dans un nouvel éditeur de diagrammes
[3] Traitement des fichiers .jar
[4] Traitement des fichiers .java
[5] Traitement des fichiers .ptidej
[6] Traitement des répertoires qui
contiennent des fichiers .class
[7] Traitement des répertoires qui
contiennent des fichiers .java
[8] Traitement des paquetages Java

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

Les classes implantées (26)

AddFolderAction.java (D)
AddJavaAction.java (D)
AddJDTPackage.java (D)
AddJDTPackageAction.java (D)
AddNewFolderAction.java (L)
AddNewJavaAction.java (L)
AddNewJDTPackage.java (L)
AddNewJDTPackageAction.java (L)
PopActionClass.java (D)
PopActionJava.java (D)
PopActionJavaPackage.java (D)
PopActionNewClass.java (L)
PopActionNewJava.java (L)
PopActionNewJavaPackage.java (L)
PopActionNewPackage.java (L)
PopActionNewPtidej.java (L)
PopActionPackage.java (D)
PopActionPtidej.java (D)
DiagramEditor.java (D)
DiagramEditorContributor.java (D)
PtidejDiagramEditor.java (L)
PtidejDiagramEditorContributor.java (L)
ClassFileFilter.java (D)
ClassJarDirFilter.java (D)
Explorer.java (D)
Util.java (D)

Parmi les interfaces utilisateurs complètement fonctionnelles, on peut compter sur Ptidej Viewer Swing Application, dont voici une capture d'écran :

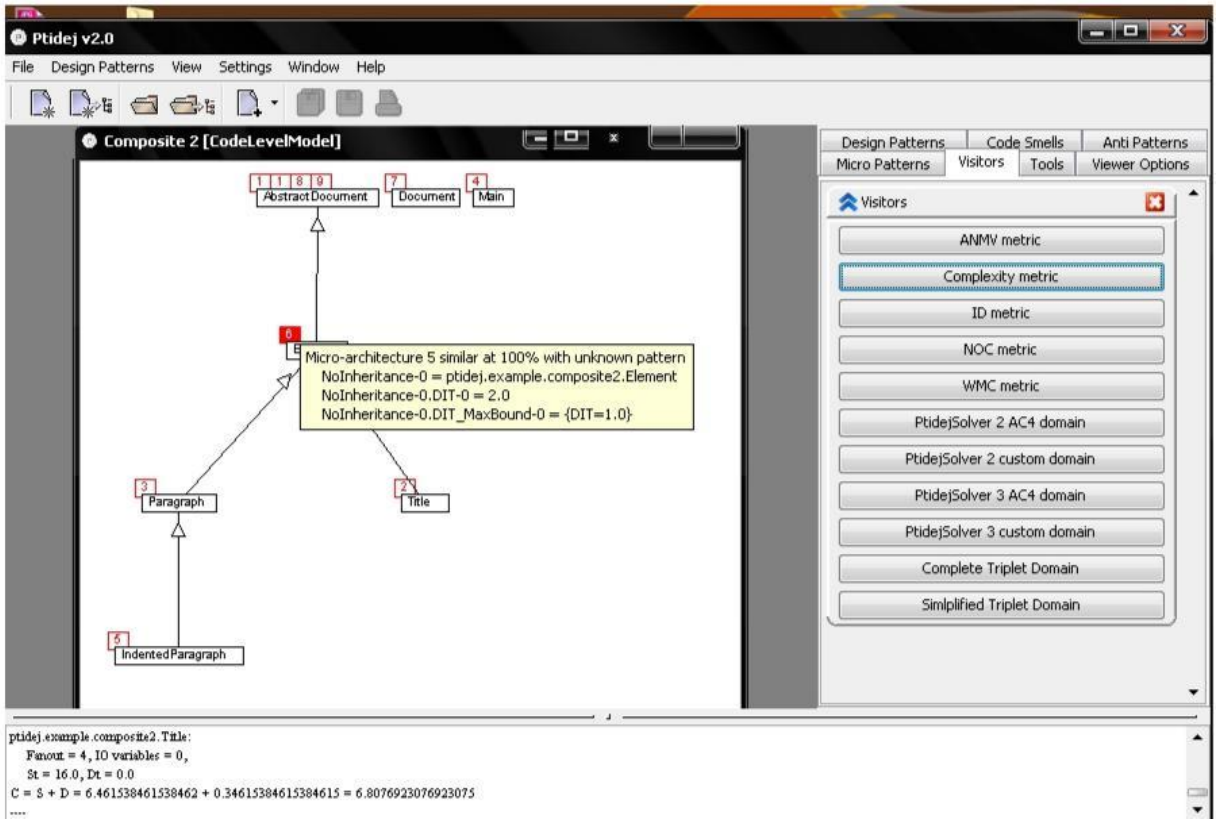


Figure II.3 :Copie d'écran de l'interface Swing de Ptidej

Il est possible de charger un projet Ptidej, afficher sa modélisation et lancer des analyses dessus.

2.Fonctionnalités de la solution : PEPS, Ptidej for EcliPSe

Avant de présenter l'architecture du plugin, un tour des fonctionnalités déjà implémentées s'impose. Les fonctionnalités assurées par le plugin sont réparties entre deux projets, « Peps » et « Peps Extension Starter Pack ».

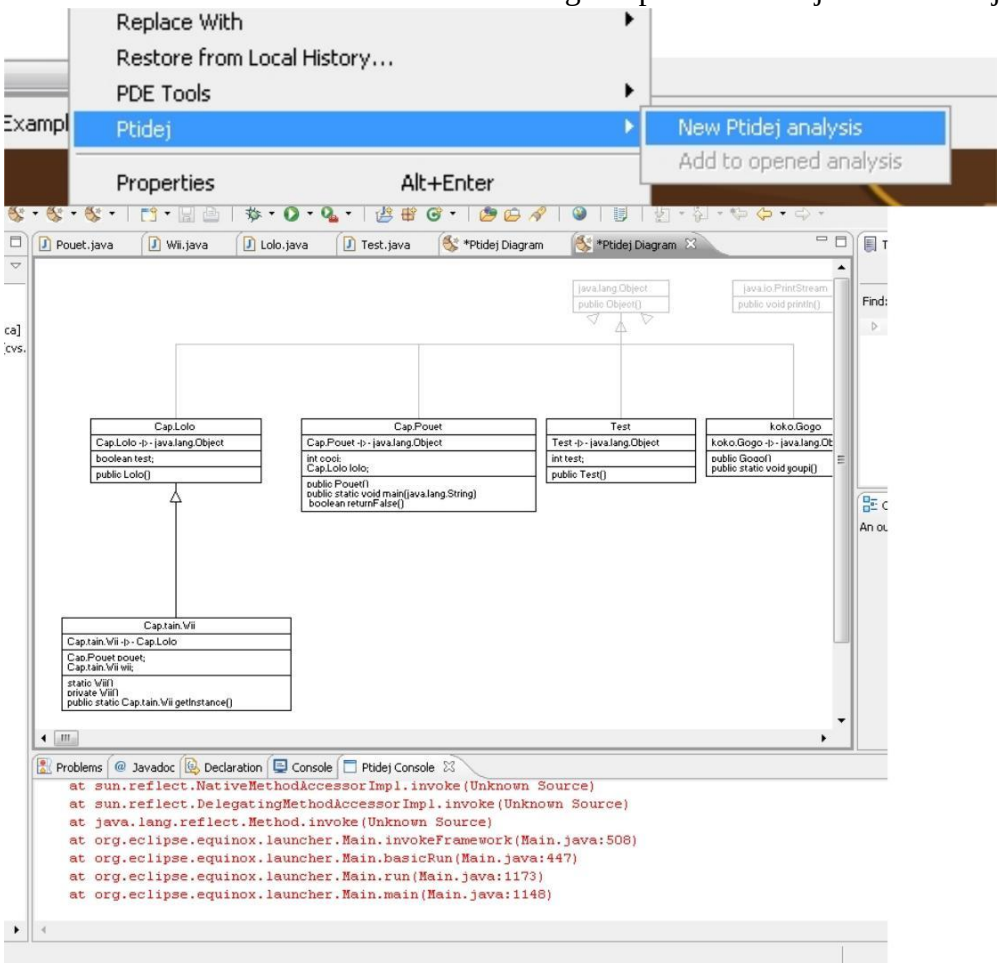
Peps

Ce projet est le cœur du plugin. Il assure la gestion des analyses, de leur modèle PADL et de leur canvas. Lui sont donc affectées la création d'une analyse (projet Ptidej) et la gestion de son cycle de vie, dont le chargement et la sauvegarde. Il n'est pas responsable de l'implémentation des outils de lasuite Ptidej, mais de la gestion de leur cycle de vie et de leur mise en relation avec le modèle. Pour cela il gère un mécanisme d'extension qui encadre la création d'outils (couche d'abstraction).

Créer un nouveau projet Ptidej

La création d'un projet Ptidej (nouvelle analyse) se fait via la vue Package Explorer d'Eclipse. Sélectionnez un ou plusieurs éléments du navigateur, projet, package, classe, ou quelconque combinaison de ces trois éléments, faites un clic droit dessus (menu contextuel). Un nouveau menu apparaîtra, « Ptidej », et vous pourrez sélectionner « New Ptidej Analysis ». Un nouvel éditeur s'ouvrira avec un superbe schéma représentant votre projet et vos classes. Un bouton dans la toolbar effectue la même fonctionnalité : faites votre sélection comme précédemment et cliquez sur le bouton « New Ptidej Analysis ». Il y a un bug sur le bouton, il faut que la vue sélectionnée soit le Package Explorer pour que cela marche.

Résumé : clic droit sur un élément du Package Explorer > Ptidej > New Ptidej Analysis.



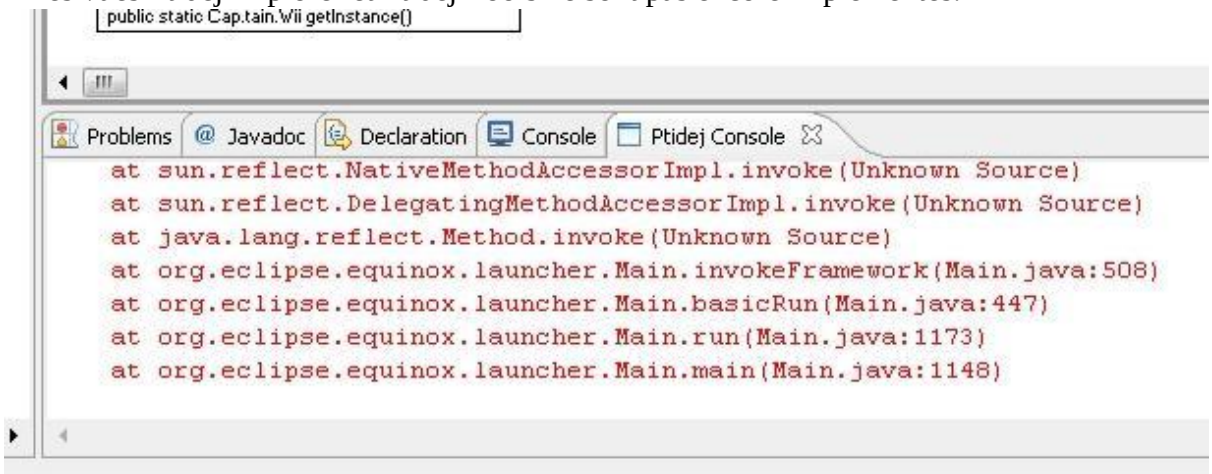
Deux vues : Editeur et Console

Vous constaterez qu'il existe deux vues :

Une vue Editeur qui s'ouvre à chaque fois qu'un projet est chargé et qui l'affiche, c'est la vue centrale du plugin, visible dans la capture précédente.

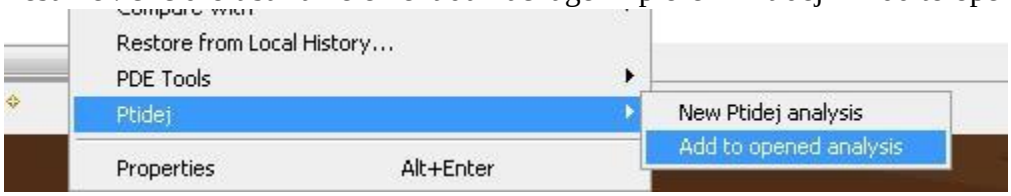
Une vue Console, dans laquelle s'affichent les messages en provenance du plugin. Cette vue, si elle n'est pas affichée par défaut, peut être ouverte en allant dans Window > Show View >Ptidej > Ptidej Console.

Les vues Ptidej Explorer et Ptidej Tools ne sont pas encore implémentés.



Ajouter des classes à un projet déjà ouvert

Il est possible, une fois une analyse créée, d'ajouter des classes ou des projets à un projet Ptidej déjà ouvert. Pour cela, sélectionnez un élément du Package Explorer que vous voulez rajouter à votreanalyse, sélectionnez « Add to opened analysis ». Un bouton dans la toolbar effectue la même fonctionnalité : faites votre sélection comme précédemment et cliquez sur le bouton déroulant « New Ptidej Analysis » pour faire apparaître un menu, vous y verrez « Add to opened analysis ». Il y a un bug sur le bouton, il faut que la vue sélectionnée soit le Package Explorer pour que cela marche.
Résumé : clic droit sur un élément du Package Explorer > Ptidej > Add to opened analysis.

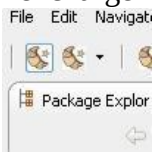


Sauvegarder un projet

Une sauvegarde très basique a été implémentée. Une fois qu'un projet est ouvert, sélectionnez File>Save As, choisissez le chemin de votre fichier et lancez la sauvegarde.

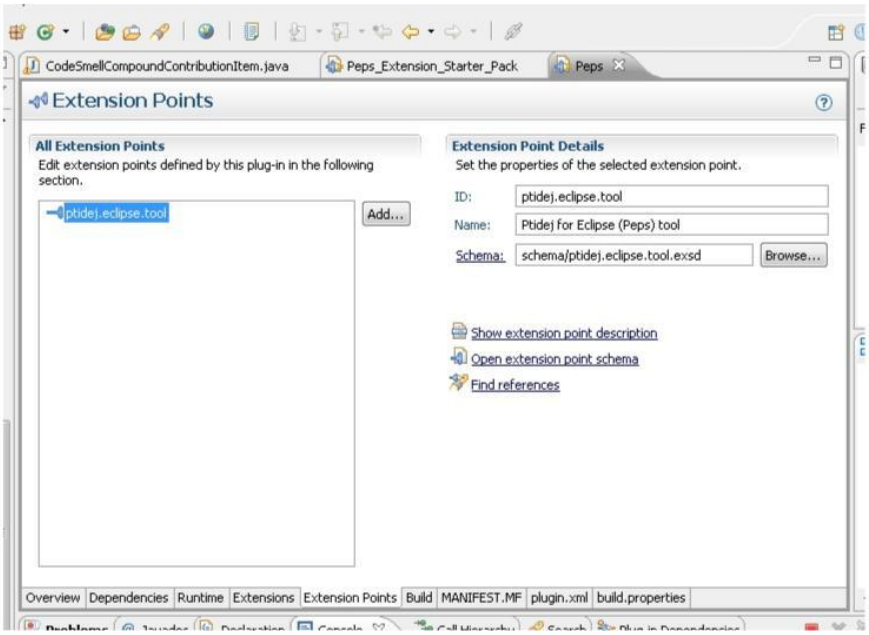
Charger un projet

Pour charger un projet Ptidej : Cliquez sur le bouton « Load Ptidej Analysis » dans la toolbar, sélectionnez votre fichier et validez.
Le chargement est très basique, aucun test n'est fait sur le fichier donné en argument!



Mécanisme d'extension

Comme il peut exister un nombre indéterminé d'outils Ptidej, dont le développement peut être échelonné dans le temps et dont l'existence n'est pas bloquante à l'évolution du coeur du plugin, un mécanisme d'extension a été mis en place. Celui-ci, basé sur le système d'extension d'Eclipse, fournit une mini-API pour créer un outil, l'enregistrer de manière transparente au sein du plugin et accéder au modèle et au canvas PADL.
Le projet Peps fournit donc un point d'extension « ptidej.eclipse.tool » comme indiqué dans la capture ci-dessous. Voir « Etendre le plugin » pour plus d'informations.

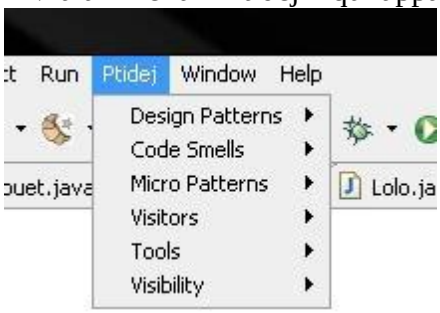


Peps Extension Starter Pack

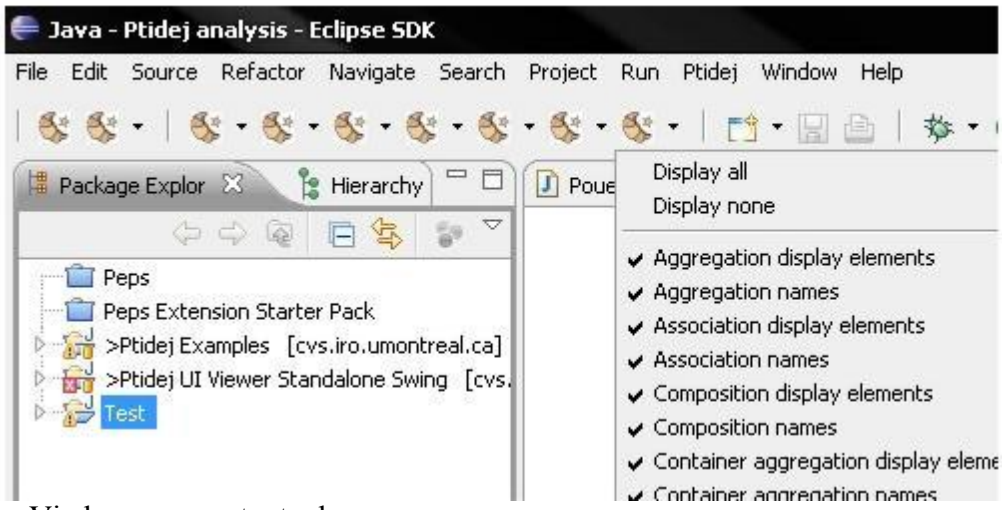
Comme indiqué ci-dessus, le projet « Peps » n'implémente aucun outil d'analyse mais il fournit un point d'extension qui sert à en créer. Un premier ensemble d'outils a été implémenté en utilisant cette interface. Sont déjà disponibles six catégories d'outils Ptidej, implémentées dans le projet « Peps Extension Starter Pack ».

Ces fonctionnalités sont activées uniquement quand l'éditeur ouvert est un éditeur Ptidej (PtidejEditorPart). Une fois cette condition remplie, elles sont accessibles de trois manières :

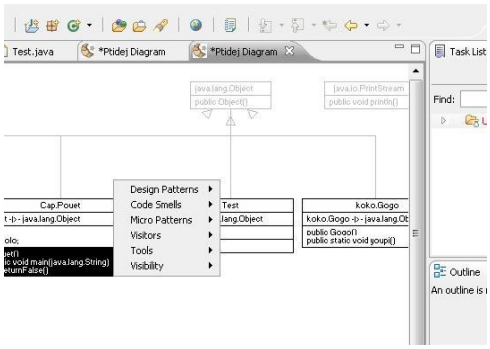
Via un menu « Ptidej » qui apparaît en haut (méthode par défaut).



● Via une toolbar Ptidej.

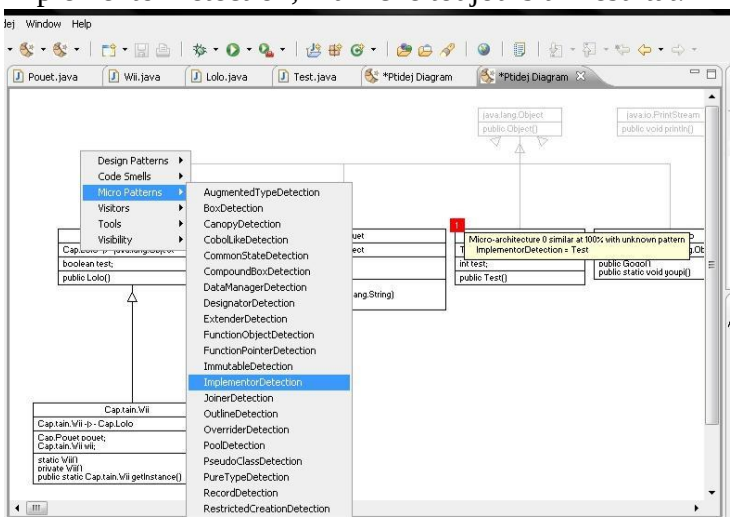


● Via le menu contextuel.



Micro Patterns

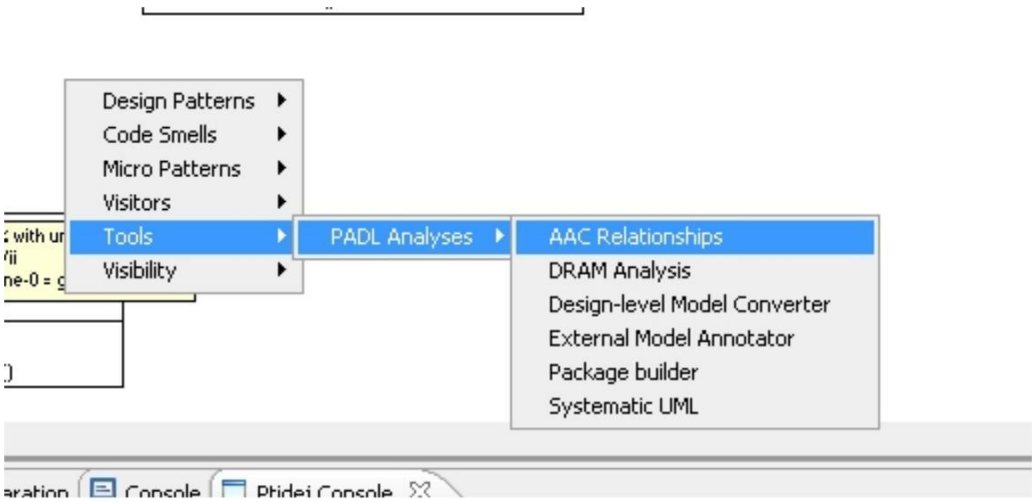
De même la détection de micro-patterns a été mise en place : Ptidej > Micro Patterns.
Les résultats s'affichent graphiquement par des petites notes numérotées rouges. Essayez ImplementorDetection, il affiche toujours un résultat.

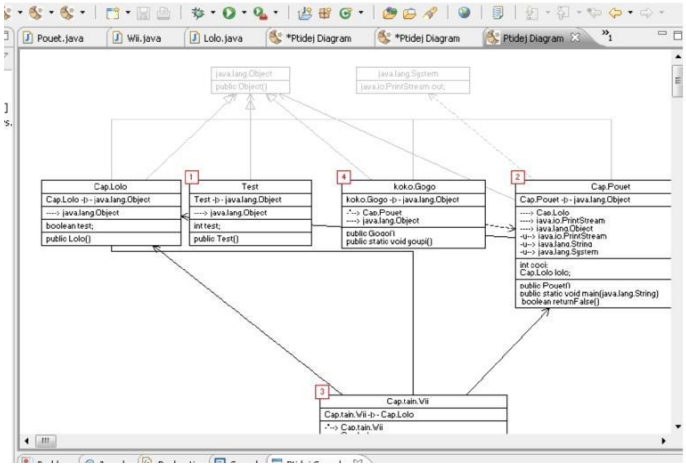


Outils d'analyse PADL

La suite d'outil Ptidej a été créée pour la détection de designs patterns et la détection de défauts de conception. Une première étape est de construire un modèle comportant les

relations de composition et d'agrégation entre classes. Cette étape correspond à la fonction disponible sous Ptidej > Tools > PADL Analyses > AAC Relationship.

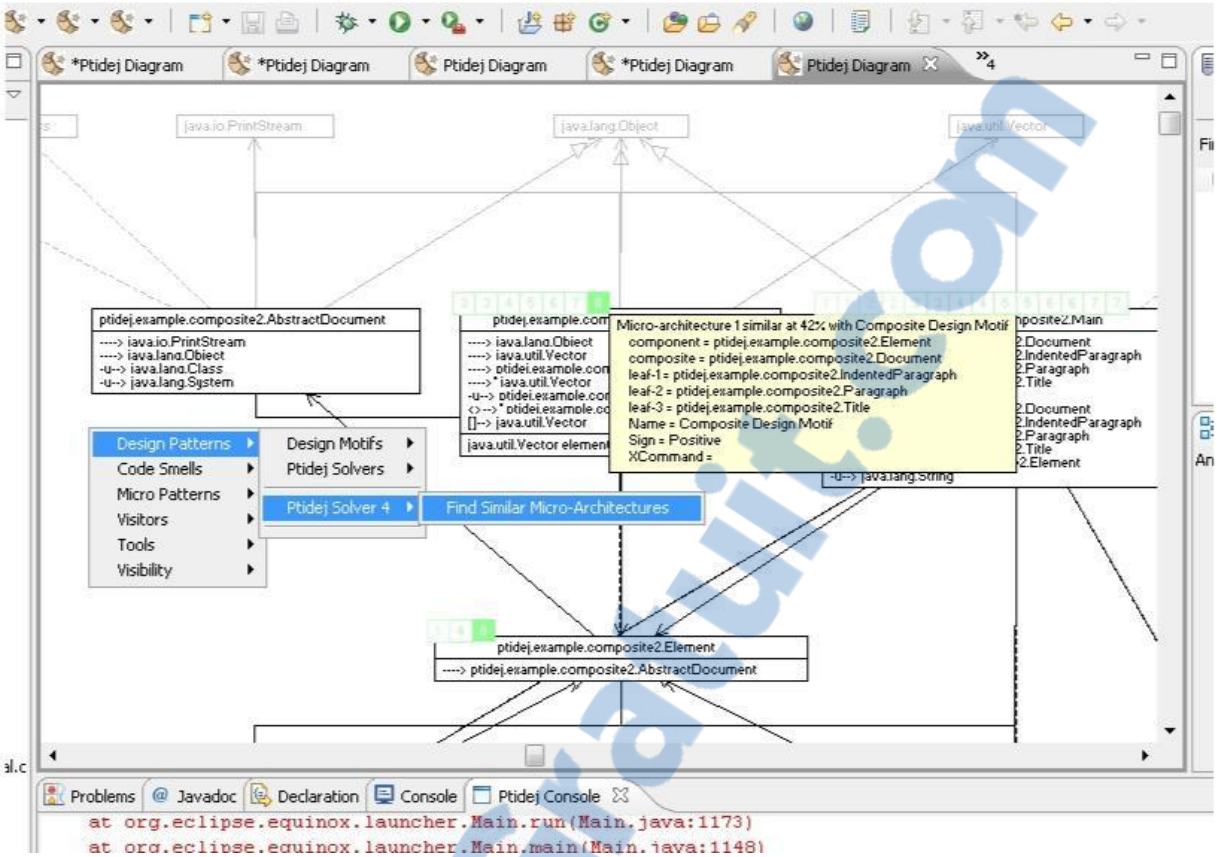




Design Patterns

Une fois la fonction AAC Relationship lancée, vous pouvez lancer la détection de design pattern.

Pour cela allez dans la rubrique du même nom : Ptidej > Design Patterns. Sélectionnez un motif et un Ptidej Solver puis lancez l'analyse. Les résultats s'affichent graphiquement, par des carrés verts situés sur les classes concernées.
Un bon exemple est de rechercher le design pattern « Composite » dans le projet Ptidej Examples, package ptidej.exemple.composite2.



Architecture générale

Peps est un plugin de Ptidej pour Eclipse, il est donc dépendent techniquement de ces deux ensembles.
Le projet en lui-même est composé de plusieurs sous-projets :

Peps, le coeur du plugin. Il fait un lien entre Eclipse et Ptidej sur lequel il est possible de rajouter des outils via son point d'extension.

Peps Extensions, abstraction qui représente tous les outils qui se greffent au plugin. Il est dépendant de Peps qu'il étend, mais aussi d'Eclipse et de Ptidej. Il peut y avoir autant d'extension que l'on souhaite.
Un premier ensemble d'extensions a été créé en même temps que le projet Peps, Peps Extension Starter Pack.

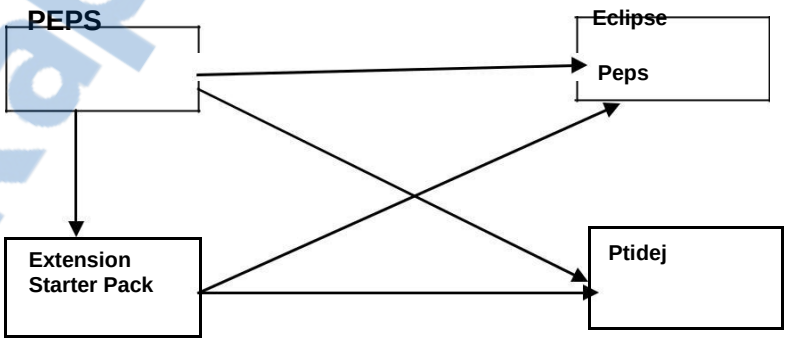


Figure V.4 : Architecture du projet « Peps »

Le projet est décomposable en deux sous-parties :

Un ensemble de classe indépendantes d'Eclipse. C'est une sur couche de Ptidej. Ces classes ont été créées pour simplifier la gestion de divers aspects de Ptidej, comme la gestion du modèle, la gestion du canvas ou encore la gestion du chargement et de l'enregistrement. Ces fonctionnalités étant indépendantes d'Eclipse, elles ont été logiquement factorisées. Il est même possible de séparer ces classes et les mettre dans un autre projet qui ne serait dépendant que de Ptidej.

Un ensemble dépendant d'Eclipse. Ce sont ces classes qui intègrent réellement Ptidej dans Eclipse.

Pour différencier ces classes, les dépendances sont indiquées dans les noms des packages :

Les packages dont le nom commence par ptidej.eclipse regroupent les classes dépendantes de Ptidej et d'Eclipse. Ex : ptidej.eclipse.ui.part

Les packages comportant ptidej SANS eclipse indiquent les package indépendants d'Eclipse,mais uniquement de Ptidej. Ex : ptidej.manager.canvas.

3.Création d' une extension pour Peps

Cette partie décrit comment créer une extension pour Peps, un nouvel outil, et l'intégrer dans l'interface d'Eclipse, pas à pas.

1. Créez votre projet plug-in

Il faut bien commencer par quelque chose. New > Project ... > Plug-in development > Plug-in project.

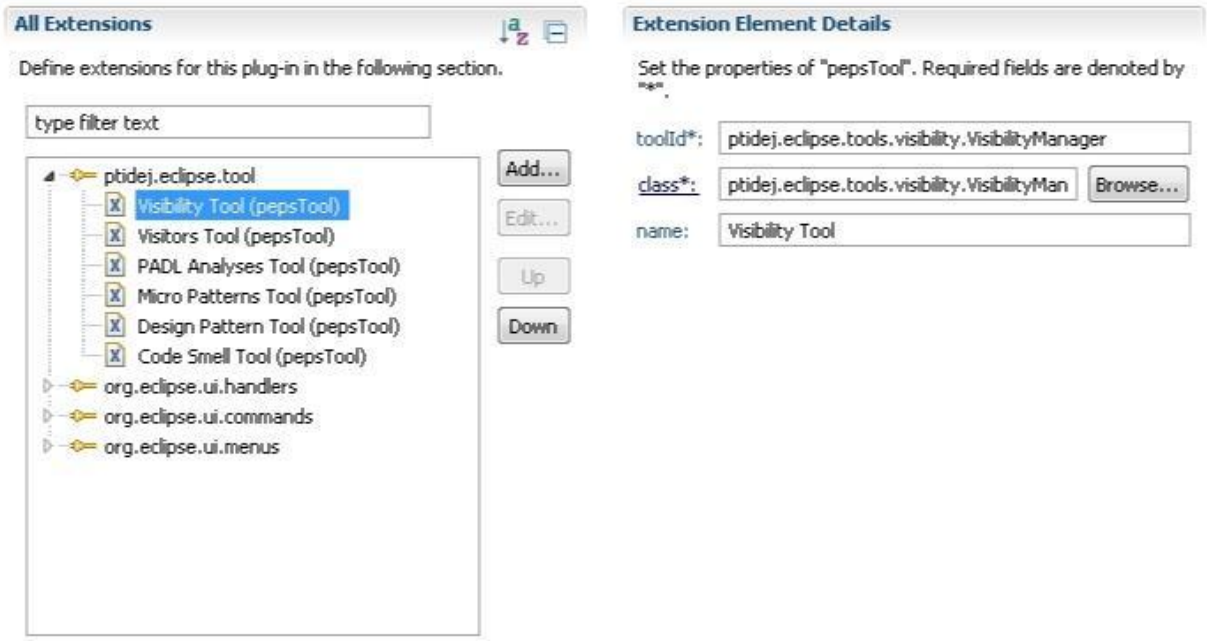
2. Établissez la dépendance avec Peps

Dans votre projet, cliquez sur plugin.xml. Eclipse vous ouvrira un éditeur contenant des formulaires.
Choisissez l'onglet Dependencies en bas de l'éditeur. C'est ici que sont déclarées les dépendances de votre plugin.

De plus si pendant l'exécution vous tombez sur une erreur ClassDefNotFound, c'est souvent parce que la dépendance est déclarée dans le build path et non dans plugin.xml. Ou alors parce que l'un des projets dont vous dépendez n'est pas un plugin, et qu'il n'est donc pas exporté en même temps que votre
projet : dans ce cas vérifiez que les projets dont vous êtes dépendant sont aussi des plugin sinon convertissez-les.

3. Sélectionnez le point d'extension

Désormais le point d'extension « ptidej.eclipse.tool » est disponible. Pour le trouver, toujours dans plugin.xml allez dans l'onglet Extensions. Cliquez sur « Add... » et s'afficheront tous les points d'extension auxquels vous avez acces. Sélectionnez ptidej.eclipse.tool.
Pour déclarer votre outil, faites un clic droit sur « ptidej.eclipse.tool » qui devrait s'être ajouté À votre liste de dépendances et choisissez l'option : New > pepsTool. Un nouvel élément apparaît.



Les champs à remplir sont :

● toolId : l'identifiant de votre outil. Veillez à ce qu'il soit unique sinon il y a risque de conflit avec une autre extension. Cet identifiant sert à récupérer une instance de votre implémentation de IPtidejTool.
● class : la classe qui implémente IPtidejTool.
● name : un nom pour votre outil.
Maintenant que vous avez déclaré votre extension, il vous reste à la réaliser.

4. Partie métier :

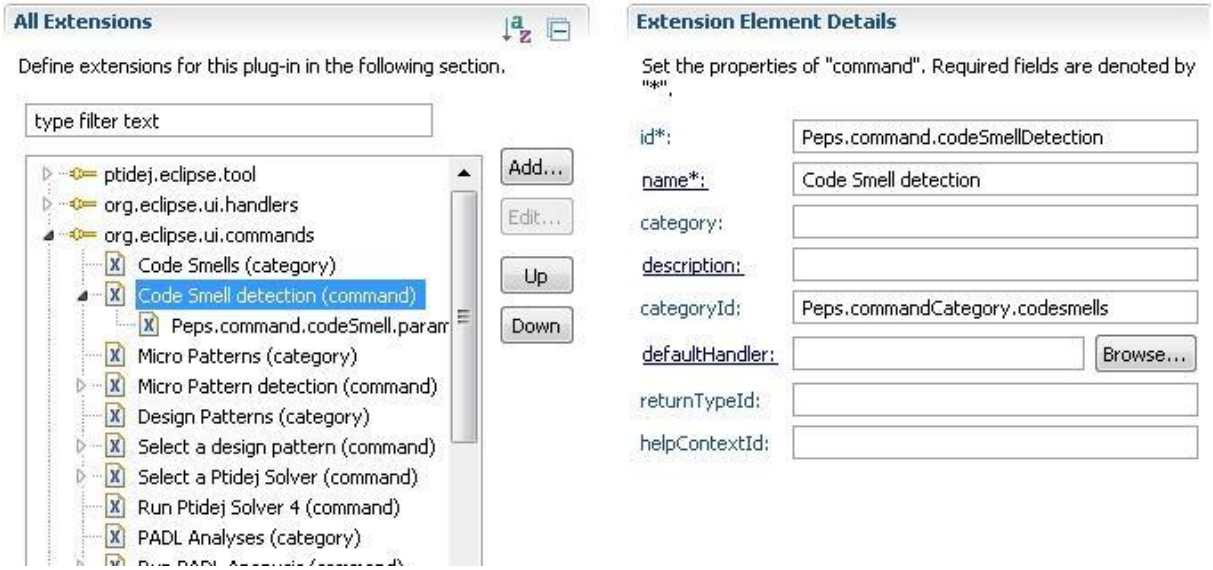
Implémentation de IPtidejTool
Cette interface est destinée à accueillir la partie métier de votre extension. Comprenez par là qu'il ne devrait pas y avoir de code propre à Eclipse dans cette implémentation.
C'est dans cette classe que vous devriez implémenter les actions de votre outil, de sorte que les handlers n'aient qu'à les appeler.
Il faut savoir qu'il n'y a qu'une instance de cette classe par PtidejEditorPart, par couple ModelManager/ CanvasManager. Une fois que votre classe est instanciée, l'éditeur appelle directement setManagers().

5. Déclarez vos actions

Dans Eclipse, les actions sont la définition abstraite de vos boutons. Il peut y avoir plusieurs boutons,qui font appel à une action, et plusieurs implémentations d'une action selon le contexte. Par exemple pour l'action « Display All » il y a un bouton dans le menu Ptidej, dans la toolbar et dans le menu contextuel (clic droit sur un editeur Ptidej), mais ces boutons exécutent tous la même action. De plus il est possible que l'action aie plusieurs implémentations (handler), selon le contexte dans lequel elle est appelée (« copier » dans un éditeur de texte n'a pas la même implémentation de « copier » dans un éditeur graphique).
Pour définir votre action, retournez dans la fenêtre Extensions de plugin.xml, faites « Add... » et choisissez le point d'extension « org.eclipse.ui.commands ». Définissez d'abord une

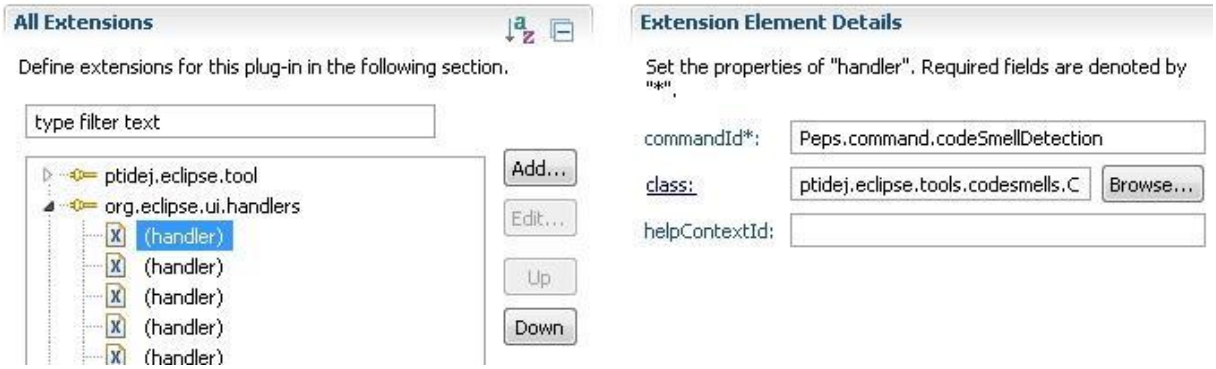
Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

catégorie de commande (New > category) puis définissez l'action en tant que telle (New > action).



6. Déclarez vos handler

Dans Eclipse, un Handler est une classe qui implémente une action. Quand vous cliquez sur un bouton, c'est cette classe qui est exécutée.
Cette fois-ci ajouter le point d'extension org.eclipse.ui.handlers et choisissez New > handler. Faites le lien entre l'action que vous avez définit précédemment et le handler en question.



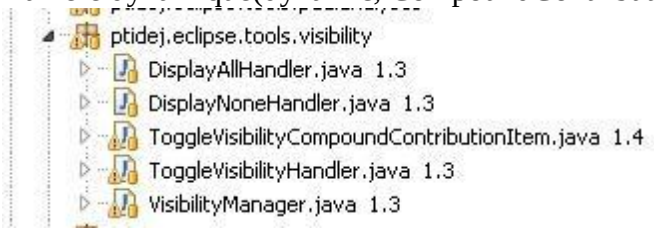
Un handler étend la classe AbstractHandler. Si votre action a un état (coché, décoché), c'est votre Handler qui met à jour l'état en question, pour cela implémentez en plus la classe IElementUpdater. Pour récupérer une instance de votre IPtidejTool initialisée (avec le ModelManager et leCanvasManager de l'éditeur), faites appel à IPtidejTool PtidejEditorPart.getTool(String), où le String est l'identifiant de votre outil (déclaré au 1. normalement).

Notez aussi que vous êtes aussi responsable du rafraichissement de la fenêtre : effectuez vos opérations puis indiquez à l'éditeur que vous souhaitez qu'il soit rafraichi (dans le cas où vous modifiez son modèle ou son canvas). Voir PtidejEditorPart.refresh().
Pour plus d'information sur les handlers, voyez dans la documentation d'Eclipse, et pour un exemple d'implémentation voyez le projet Peps Extension Starter Pack.

7. Déclarez vos boutons et vos menus

Il ne vous reste plus qu'à déclarer les boutons qui actionneront vos actions (et en cascade les handlers qui sont leur implémentation concrète).
Le point d'extension est org.eclipse.ui.menus. Pour plus d'information sur ce point d'extension voyez dans la documentation d'Eclipse, et pour un exemple d'implémentation voyez le projet Peps Extension Starter Pack.
Peps fournit déjà trois menus où insérer vos boutons :

● menu:Peps.menu.ptidej qui est le menu « Ptidej » qui ne s'affiche que quand l'éditeur ouvert est un PtdejEditorPart.
● popup:peps.ui.parts.PtidejEditorPart?after=additions qui est le menu contextuel de PtidejEditorPart (clic droit sur l'éditeur).
● toolbar:Peps.toolbar.ptidej , toolbar associée à l'éditeur Ptidej, ne s'affiche que quand celui-ci est ouvert.
● toolbar:Peps.toolbar.ptidej.new toolbar associée au plugin Peps, elle est tout le temps visible. Si vous placez votre bouton dedans, sachez qu'il est possible que l'éditeur ouvert ne soit pas de type PtidejEditorPart.
Les boutons peuvent être déclarés de manière statique (plugin.xml) ou de manière dynamique(dynamic, CompoundContributionItem).



II. Argo uml :

1.Introduction

Dans l'introduction, nous listons les trois clés pour notre choix. ArgoUML différent aux autres : i) il nous donne une idée sur l'utilisation de la psycology cognitives , ii) basé en open standards; iii) 100% pure Java; and iv) projet open source .

Cognitive Psychology
ArgoUML comme particularité a été inspire par trois théories : i) reflection-in-action, ii) opportunistic design iii) et comprehension et solution de problem .

Open Standards
UML lui meme est open standard. ArgoUML tout long a éssaiyé d'utiliser open standards pour tous ses interfaces.
La clé avantage de open standards il permet facilement inter-working entre les applications, et la possibilité
De supprimer d'une application des autres qui son intéressants.

Il est très flexible cela veux dire que certains logiciels commerciales n'utilise pas les standards.
Les clients ne son pas stupide et hésite d'acheter un non-standard logiciel, a cause de danger de lock-in.
Le mouvement open source évite le contrôle oligopolistique des logiciels.

Open Source

L'ArgoUML c'est un projet *open source* . Signifie que personne peut avoir une copie code source et le
change, utilisé pour nouveaux buts. La seul (major) obligation est de passer le code au même chemin aux autres .
Il existe trois façons pour obtenir argouml
1. Run ArgoUML directement de Web Site en utilisant Java Web Start. C'est la plus facile.
2.télécharger le code binaire .c'estr plus juste pur utiliser argouml régulièrement .
3. téléchargeable code source en utilisant CVS. Cette option pour voir les fonctionnements d'ArgoUML, ou Site: [http:/-/-argouml.tigris.org].

2. Caractéristiques

ArgoUML est un outil d'aide à la conception orientée objet.

- Une application multi-plateforme :

  ArgoUML est entièrement codé en Java 1.2 et utilise les classes de base de Java (Java Foundation Classes). Ceci permet à ArgoUML de fonctionner sur pratiquement n'importe quelle plateforme munie d'une machine virtuelle Java.

- Standard UML

  ArgoUML est conforme avec la norme UML 1.3 définie par l'OMG. Le code pour la représentation interne d'un modèle UML est complètement produit suivant les spécifications de l'OMG. Pour se faire, une bibliothèque spéciale de metamodel (NSUML) a été développée par la société Novosofts sous licence GPL. Ceci rend ArgoUML extrêmement flexible pour s'ajuster aux nouvelles normes UML à venir. Cependant quelques caractéristiques avancées d'UML ne sont pas encore disponibles dans les diagrammes. Notamment, il n'existe pas encore de diagramme de séquence.

- Supporte XMI

  XMI est un format d'échange XML entre les outils UML. ArgoUML utilise ce standard de sauvegarde pour faciliter l'échange de données avec d'autres applications. Ceci permet de convertir des données Rational Rose vers ArgoUML. Malheureusement, cette conversion ne peut se faire sur les données graphiques de Rational Rose.

- Edition de modèle UML

  ArgoUML utilise l'outil GEF (Graph Editing Framework) de l'University de Californie Irvine (UCI) pour éditer les diagrammes UML. Les digrammes suivants sont supportés :

  - Diagramme de classes
  - Diagramme d'état
  - Diagramme d'activité
  - Diagramme de cas d'utilisation
  - Diagramme de collaboration
  - Diagramme de déploiement
  - Diagramme de séquence (disponible dans un future proche)

- Supporte OCL

  OCL (Object Constraint Language) est un langage logique de prédicat. ArgoUML est entièrement conforme à la syntax et aux types d'OCL et s'appuit sur le compilateur OCL développé par l'Université Technique de Dresden. Par ailleurs, il est possible de générer du code Java à partir de contraintes OCL.

- Exportation vers une base de données

  Les données textes et les diagrammes de cas d'utilisations peuvent être maintenant stockés dans une base de données mySQL. Etant donné,

qu'ArgoUML implémente JDBC, l'exportation des données vers n'importe quelle base de données compatible JDBC devient facile.

- Plusieurs formats d'exportation de diagramme

  Les diagrammes peuvent maintenant être exportés vers un format GIF, PostScript, eps, PGML ou svg. Le format standard de sauvegarde des diagrammes reste le PGML, mais il sera changé lors de la prochaine norme en svg (Scalable Vector Graphics) suivant les spécifications du consortium W3C.

- Génération de code Java

  ArgoUML permet la génération de classes Java à partir des diagrammes de classes.

- Design Critics

  Design Critics est un thread de control qui s'exécute en tâche de fond. Il analyse le travail de l'utilisateur, avertit des erreurs de syntaxes et propose des améliorations. Design Critics n'interrompt jamais l'utilisateur, mais envois toutes ses remarques dans la « To Do » liste.

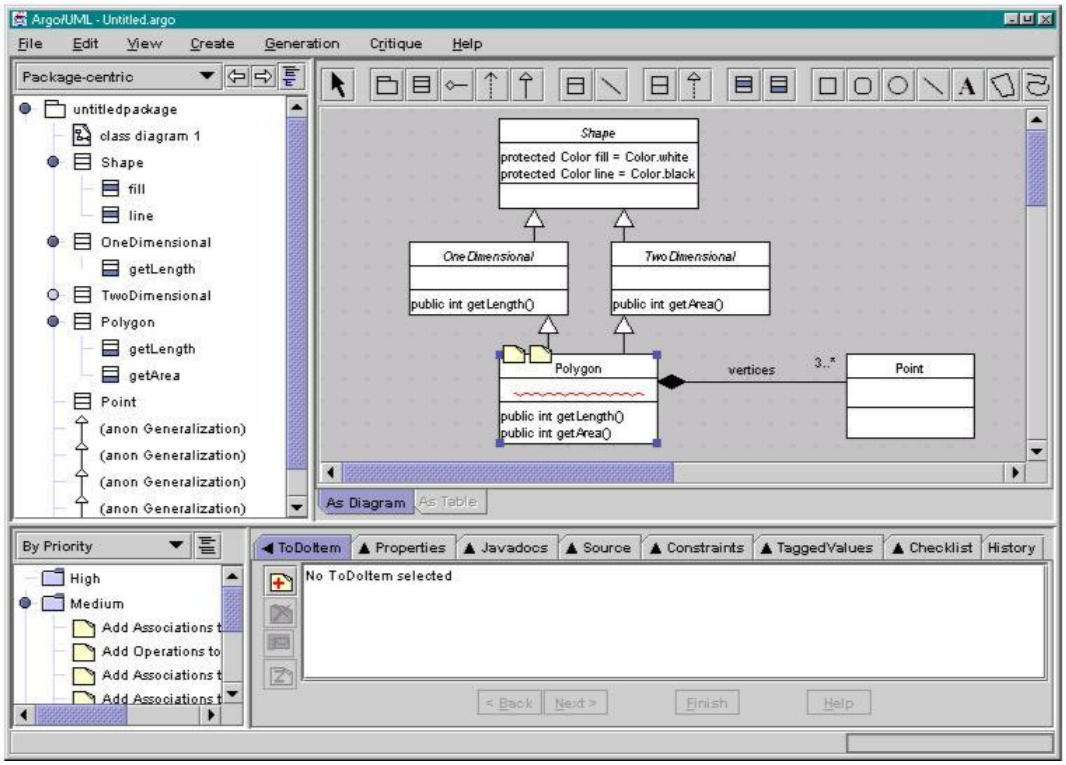
- La « To Do » liste

  Une des difficultés des concepteurs est de garder le fil conducteur de leur conception. C'est à dire le séquencement des tâches à réaliser. Grâce à la « To Do » liste, ArgoUML facilite le travail des concepteurs en leurs rappelant les tâches élémentaires qui reste à accomplir et les éventuelles erreurs trouvées par le Design Critics. Ceci permet alors au concepteur de se concentrer d'avantage sur des questions de conceptions et non sur des problèmes de syntaxes. Par ailleurs, le concepteur peut ajouter des notes personnelles dans la « To Do » liste pour rappel. Les entrées de la « To Do » liste peuvent être ensuite classées par ordre de priorité. Dans certains cas, un wizard est disponible pour la résolution des problèmes.

- La Checklist

  Les Checklists sont largement répandues lors des réunions d'étude de conception, en partie, parce qu'elles rappellent aux concepteurs, les détails de conception à vérifier et permet d'éviter des erreurs communes de conception. ArgoUML fournit aussi une Checklist qui répertorie les questions essentielles à se poser. Comme par exemple, le nom de la classe est-il approprié ? etc. Par ailleurs, les checklists sont évidemment spécifiques aux éléments (classe, attribut, association,...) dont elles se rapportent .

2.1 L'interface Utilisateur

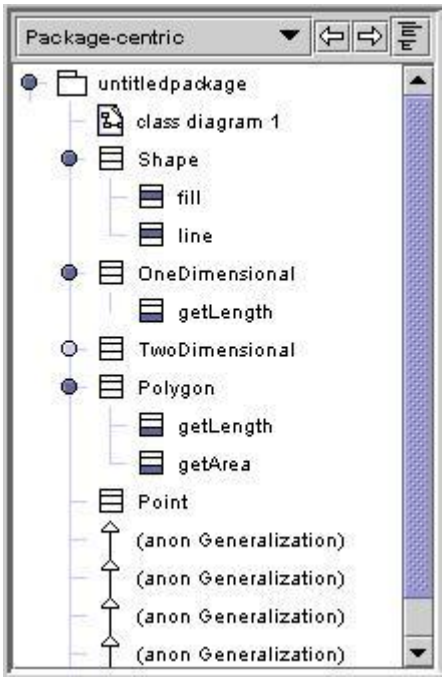


La fenêtre principale d'ArgoUML est composé de 4 sous-fenêtres :

- Une fenêtre de navigation (en haut à gauche)
- Une fenêtre d'édition (en haut à droite)
- Une fenêtre « To Do » (en bas à gauche)
- Une fenêtre détails (en bas à droite)

Nous allons étudier en détails ces différentes fenêtres dans les parties suivantes. Notez que dans la figure ci-dessus, on remarquera qu'ArgoUML permet tout de même l'édition d'héritage multiple alors que java interdit l'héritage multiple.

Fenêtre de navigation

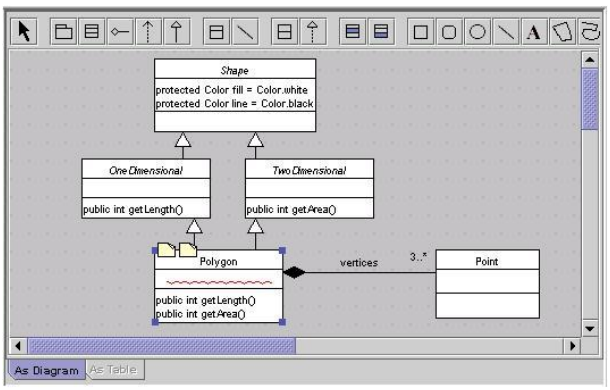


La fenêtre de navigation nous montre tous les éléments de notre conception (classe, attribut, association,...). Cette fenêtre est similaire à n'importe quel autre fenêtre de navigation, tel que l'Explorateur Windows ou Rational Rose.

Un simple clic sur un objet de cette fenêtre, sélectionne l'objet dans la fenêtre d'édition et force l'affichage ses propriétés dans la fenêtre de détails.

Un double-clic sur un objet permet d'effectuer des modifications de celui-ci dans la fenêtre d'édition. Bien que cette fenêtre de navigation soit similaire à beaucoup d'autres applications, ArgoUML fournit un système de filtre. En effet, grâce à la liste déroulante située en haut de cette fenêtre de navigation, il est facile d'obtenir une vue mettant en évidence l'héritage des classes ou les transitions entre les états. D'autres vues sont fournies par ArgoUML et il nous est possible de définir une vue personnalisée par le biais de la fenêtre « Navigational Perspective Configuration ».

Fenêtre d'édition



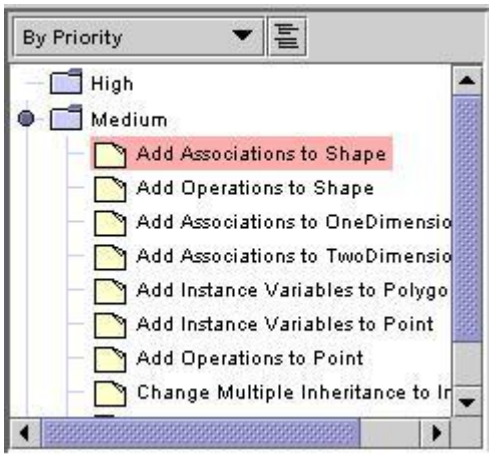
La fenêtre d'édition est l'espace de travail principal. Elle est utilisée principalement pour l'édition de diagrammes.

Cependant, il est possible d'utiliser cette fenêtre pour éditer une table qui liste le contenu des diagrammes.

Plusieurs onglets sont disponibles en bas de la fenêtre indiquant les différentes façons pour visualiser ou éditer un objet.

ArgoUML inclus par défaut l'onglet «As Diagram», et le téléchargement de fichier jar optionnel permet d'avoir les onglets «As Table» et «As Metrics».

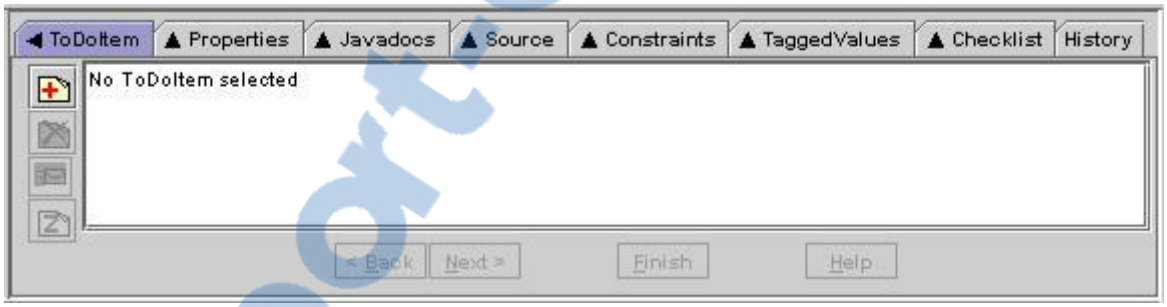
s Fenêtre « To Do »



La fenêtre « To Do » list est une aide mémoire sur les tâches restant à faire. Ces tâches peuvent être classées par ordre de priorité ou suivant différent critères. Les items de cette liste peuvent être des mémos personnels entrées par le concepteur, mais la majorité d'entres-eux est générée par le thread de contrôle : Design Critics. En effet, le Design Critics analyse continuellement le travail du concepteur à la recherche d'éventuels omissions ou problèmes. Lorsqu'un problème est trouvé, le thread de contrôle crée un item dans la « To Do » liste. Bien entendu, la résolution d'un problème répertorié induit la suppression automatique de l'item de la « To Do » liste.

Lors de la sélection d'un item de la « To Do » liste, les informations relatives au problème sont affichées dans la fenêtre de détails sous l'onglet « ToDoItem ».

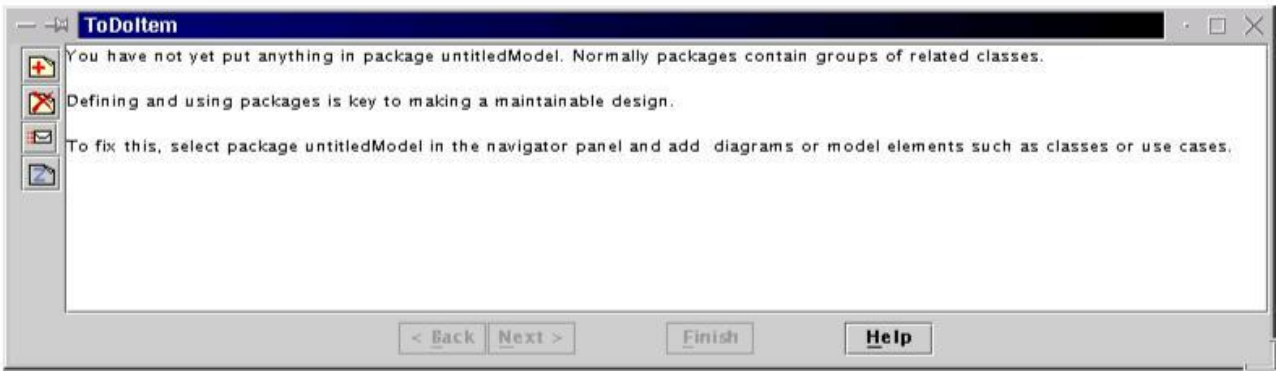
FENETRE DETAILS



La fenêtre de détails permet l'édition d'informations concernant les objets selectionnés. La plupart des onglets possèdent une flèche orienté indiquant les objets auxquels l'onglet se rapporte. Par exemple, l'onglet « ToDoItem » possède une flèche orienté gauche indiquant que le contenu de l'onglet se rapporte à la fenêtre « To Do » liste. Les autres onglets ont une flèche orientée haut se référant aux objets de la fenêtre d'édition ou de navigation.L'onglet « History » ne possède pas de flèche orientée, sa présence dans la fenêtre détails est justifié par une facilité d'accès.

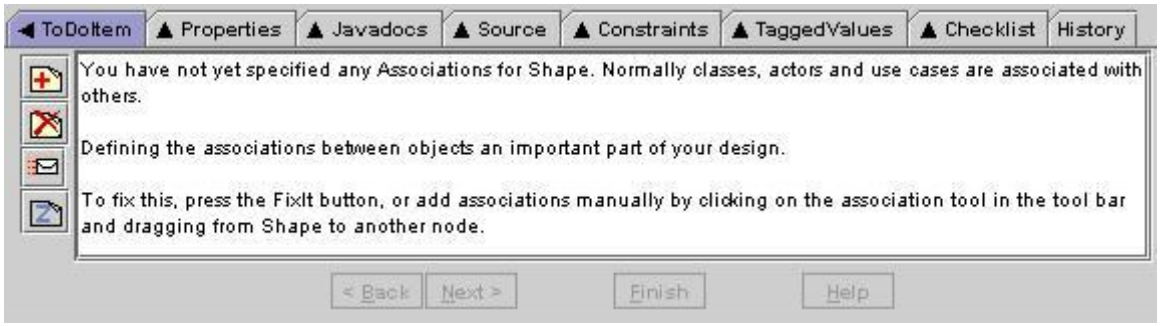
Chaque onglet peut-être extrait de la fenêtre détails par un double-clic.

Rapport-Gratuit.com



Cependant, les fenêtres extraites ne sont pas mises à jour lors d'une modification de l'onglet correspondant dans la fenêtre détails.

Onglet « ToDoItem »



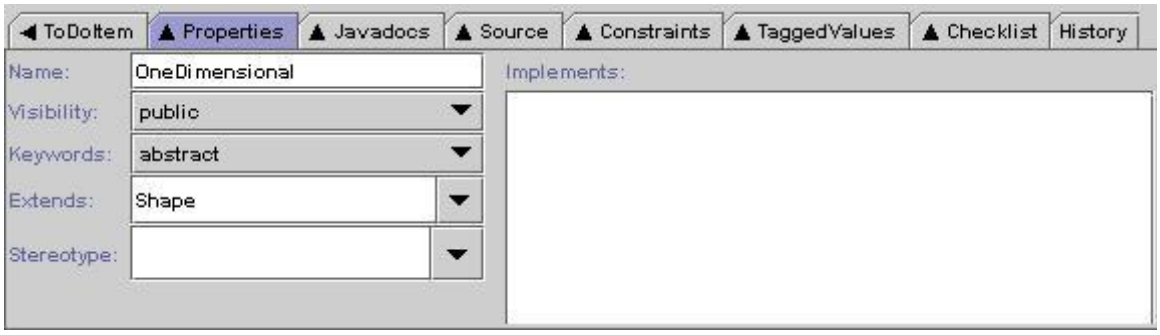
L'onglet « ToDoItem » donne la description des items de la fenêtre « To Do ». La description est faite en trois courts paragraphes expliquant le problème, l'importance du problème et comment le résoudre.

La barre d'outils situé à gauche de l'onglet permet :

- D'ajouter un mémo personnel dans la « ToDo » liste
- Supprimer l'item « To Do » courant
- Envoyer un mail à l'auteur de la critique de l'item courant (le mail est envoyé à Tigris dans le cas d'un item généré par le Design Critics)
- Désactivé (Snooze) l'item durant un certain temps avant d'être réactivé

Certain item peut-être résolu par Wizard, dans ce cas les boutons "next" et "finish" deviennent actifs.

Onglet « Properties »

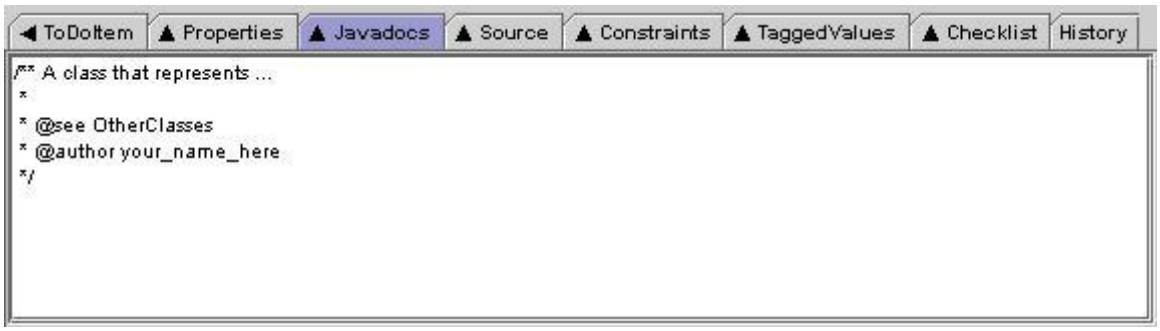


Cette onglet donne les propriétés des éléments selectionnés. Son contenu varie suivant le type des éléments sélectionnés.

Ces types sont définis comme suit :

- General
  - Diagram
  - Package
- Class Diagram
  - Package, Class, Interface, Instance
  - Dependency, Association, Generalization, Realization, Link
- Use Case Diagram
  - Actor, Use Case
  - Association, Generalization
- State Diagram
  - State, Pseudostate
  - Transition

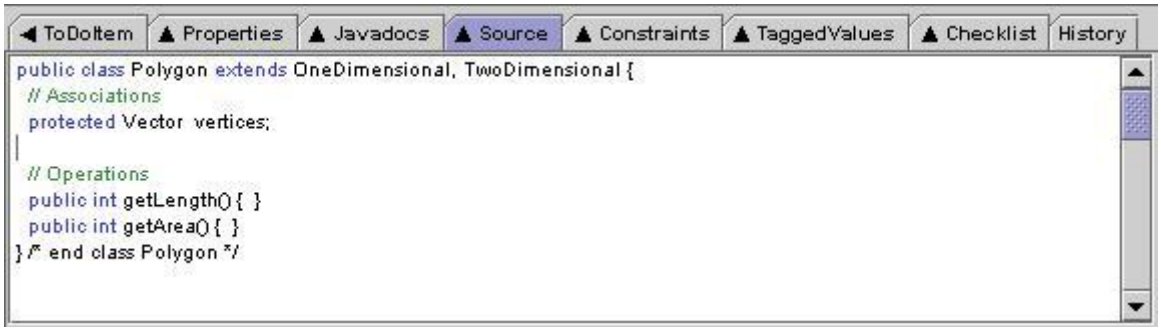
Onglet « Javadocs »



Comme son nom l'indique, l'onglet « Javadocs » permet la saisie de documentation java sur l'élément sélectionné.

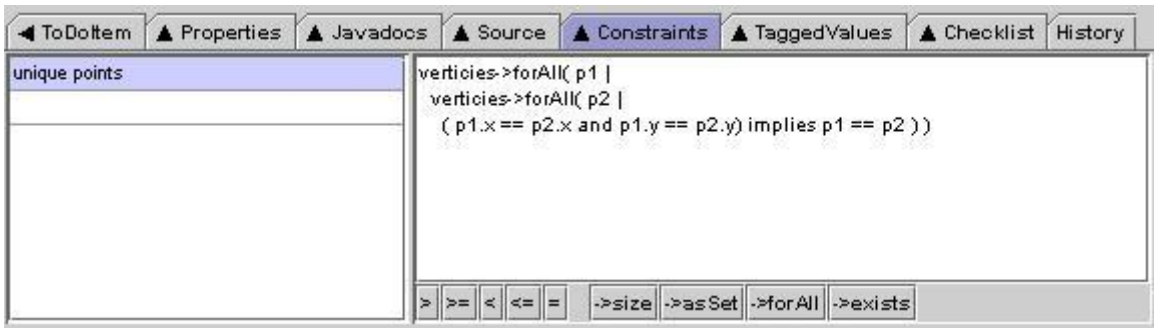
Onglet « Source »

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES



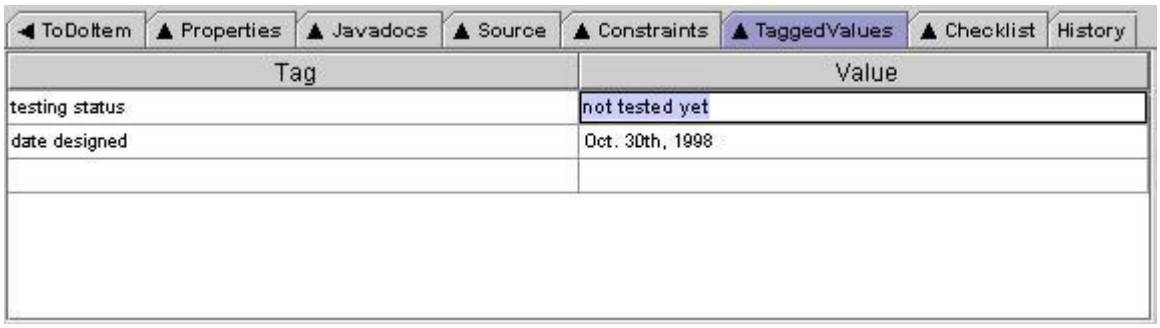
L'onglet « Source » permet de voir le code java généré par ArgoUML. Pour le moment, toute modification effectuée sur de code n'est malheureusement pas répercutée sur le modèle UML.

Onglet « Constraints»



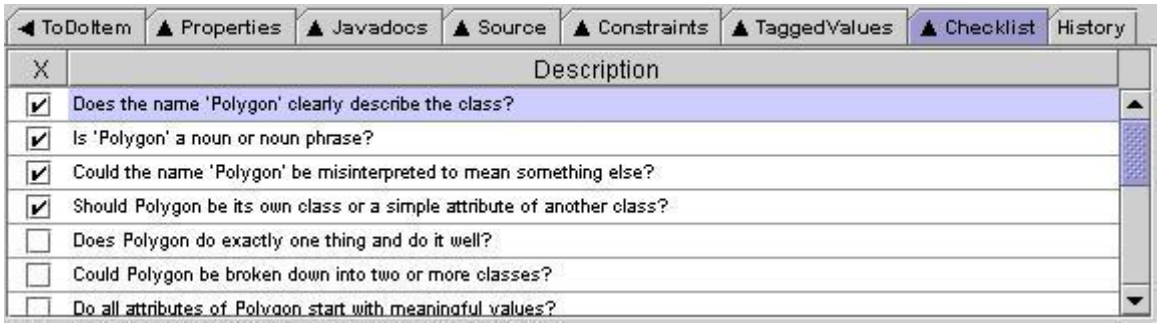
L'onglet « Constraints » permet d'éditer des contraintes sur l'élément sélectionné à l'aide du langage OCL (Object Constraint Language). OCL est un langage logique de prédicat. Malheureusement, cette version d'ArgoUML ne tient pas compte des contraintes saisies.

Onglet « TaggedValues »



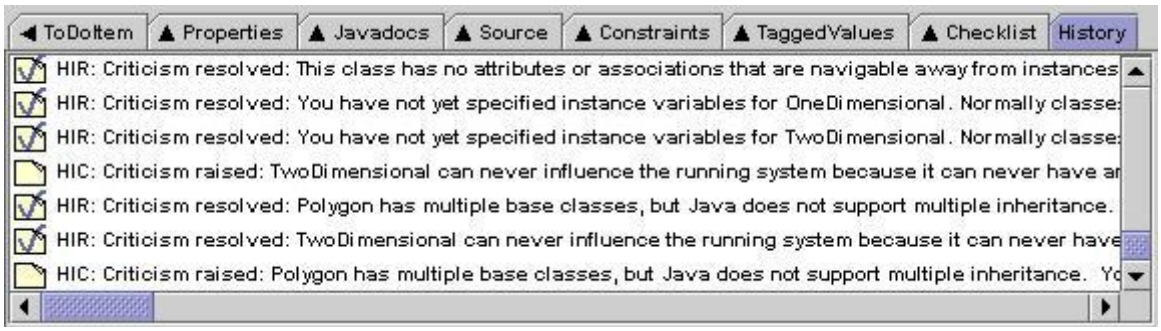
L'onglet « TaggedValues » permet d'ajouter des tags sur les objets sélectionnés. On peut par exemple, saisir le statut de l'objet : testé ou non, la date de création, etc. Cependant, les valeurs de cet onglet n'est pas pris en compte par ArgoUML.

Onglet « Checklist »

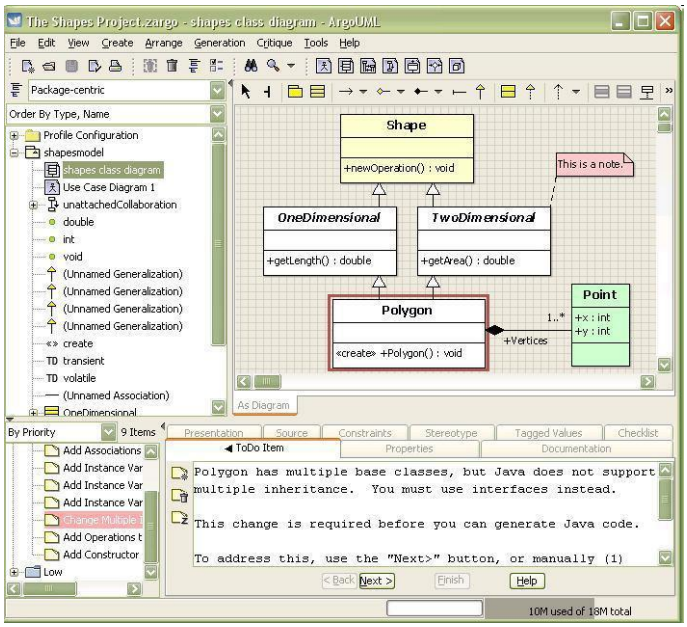


L'onglet « Checklist » liste des questions essentielles à se poser lors de la conception. Ces questions sont naturellement spécifiques aux objets sélectionnés.

Onglet « History »



L'onglet « History » liste par ordre chronologique les critiques générés par le Design Critics et la manière dont elles ont été résolues.



Le Menu Bar

menu bar donne l'accée de tous main futures d' ArgoUML. Comme il est conventional, menu options t et menu items that invoke a dialog box are followed by an ellipsis (…).

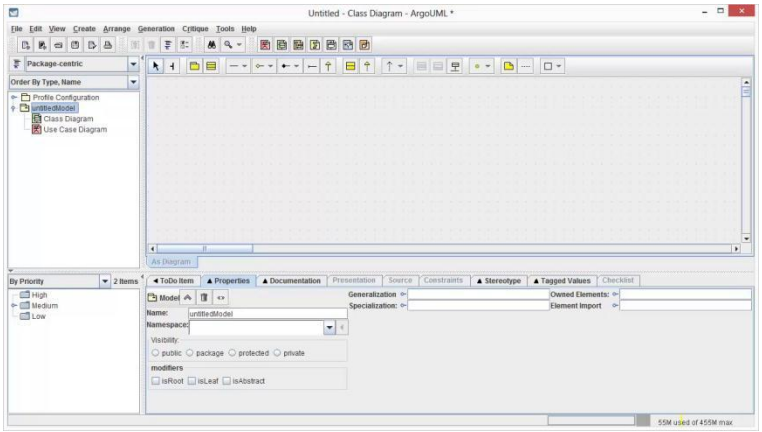
• *FileFile* menu. Pour créee un nouveau project, save et open projects, import sources from ,

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

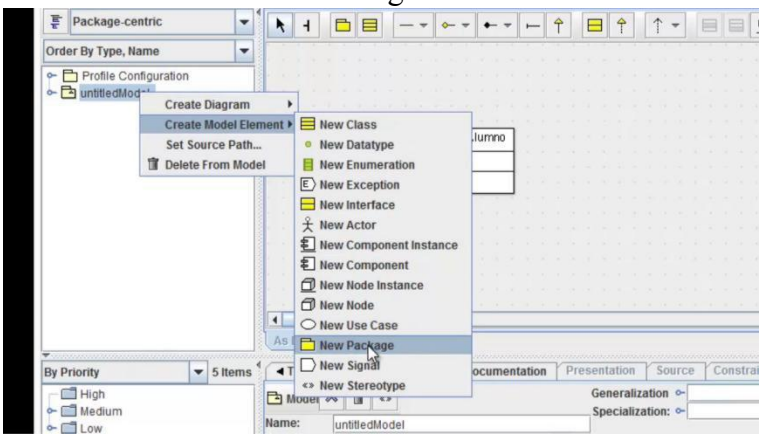
load and save le model a et de database, print le model, sauvgarder le graphes de model,sauvgarder la configuration du model et quiter ArgoUML
• *EditEdit* menu. Select un ou plusieurs UML items de diagramme , défaire et refaire réalisation, suprimer
items de diagrams ou de vrai model, vider ce qu'il est inutile et changer l'environnement.
• *ViewView* menu. Les changements entre les diagrams, trouver les artifacts de model, zoom le diagram,
Selectionner les diagrammes dereprésentation particilieieres a particular diagram representation
Dans details menu, ajuster les grilles, voir les buttons de selection, et faire des changements entre UML and Java notation.
• *Create DiagramCreate Diagram* menu. Permet de créer n'imorte qu'elle de septs types UML diagram
(class, use case, state, activity, collaboration, deployment and sequence) supportées par ArgoUML.

3. Intégration de l'outil Argo uml a netbeans pour la retro-ingénierie

Vue général d'Argo uml

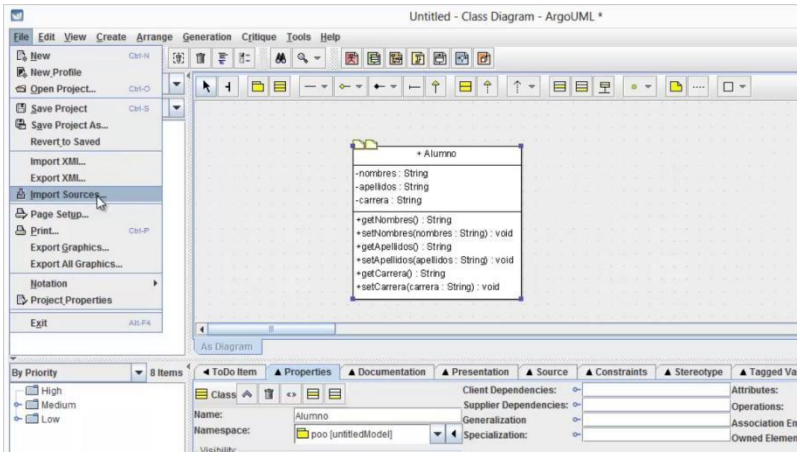


Interface d'argo uml

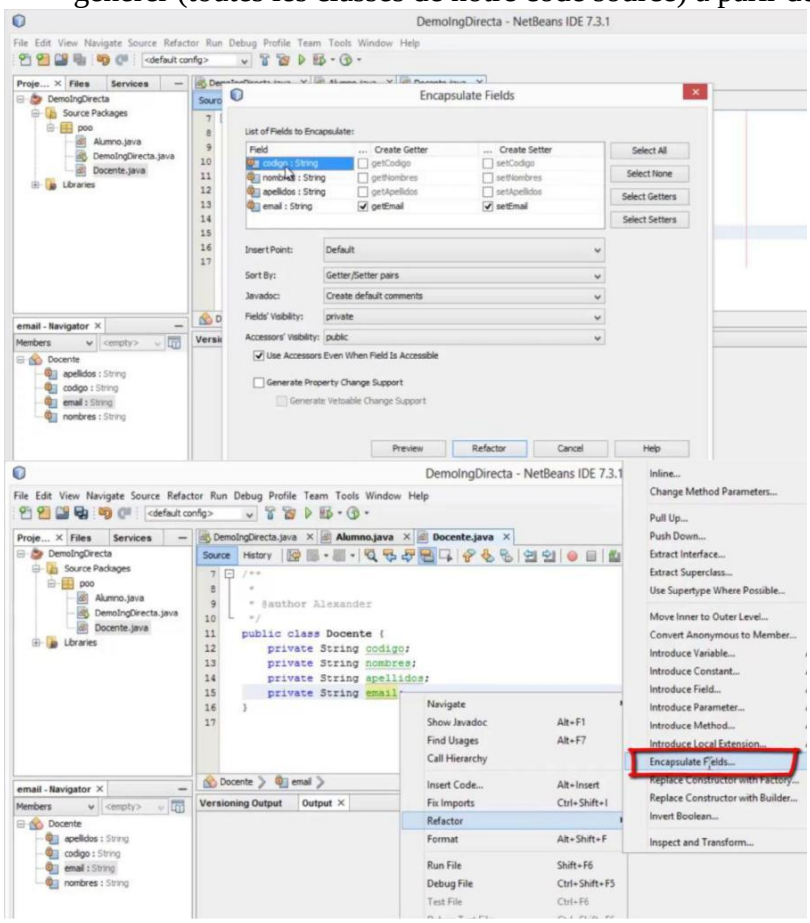


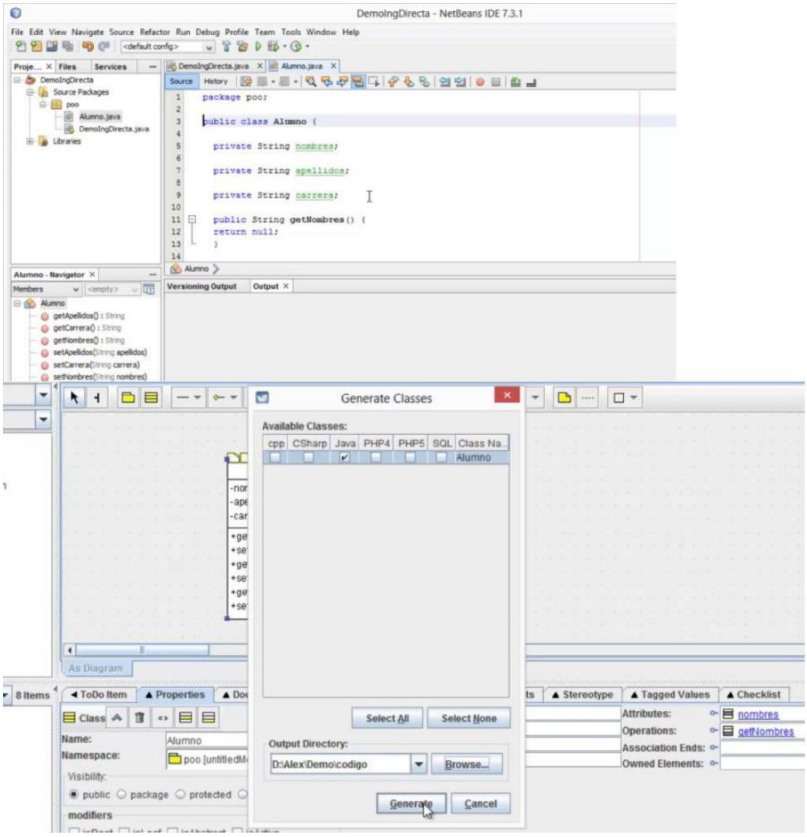
Création d'un nouveau package de notre code source(petit progmamme ecrie en java sous netbeans)

Création de notre classe sous argo uml et implantation de ses déclarations et methodes

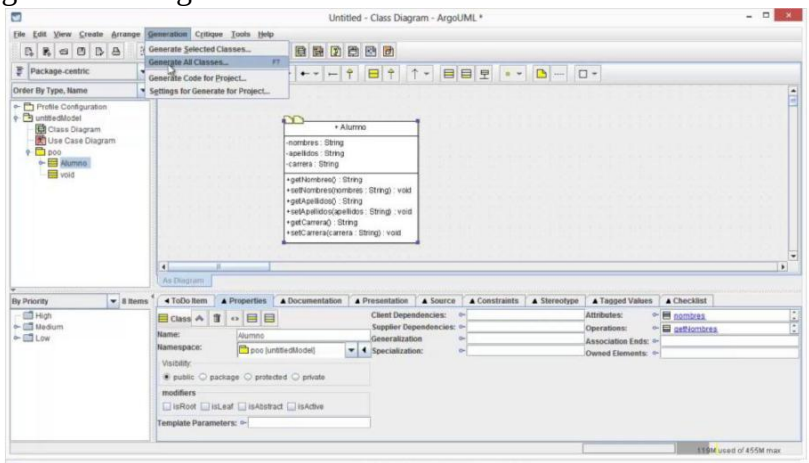


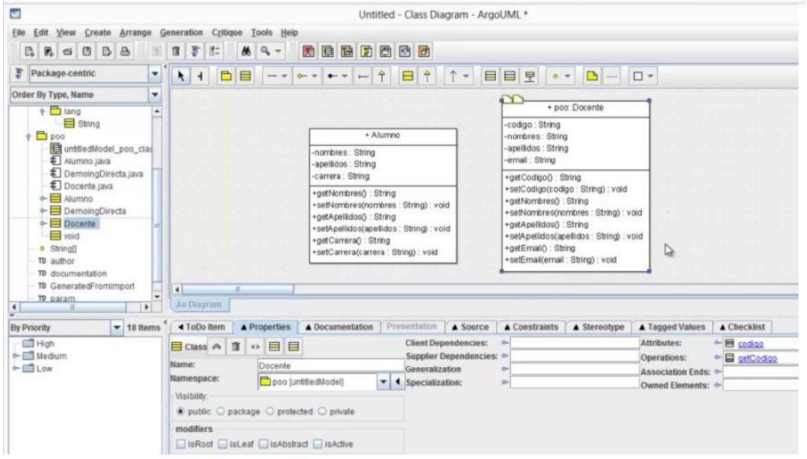
generer (toutes les classes de notre code source) a parir de netbeans .





generaration de toutes les classe
ici nous cochons « java » après « generate »
generation « generate all classes »





Transformation de code souce en uml

III.Entreprise Architect

1. représentation entreprise architect

Enterprise Architect est un logiciel de modélisation et de conception **UML**, **édité** par la société australienne Sparx Systems. Couvrant, par ses fonctionnalités, l'ensemble des étapes du cycle de conception d'application, il est l'un des logiciels de conception et de modélisation les plus reconnus.

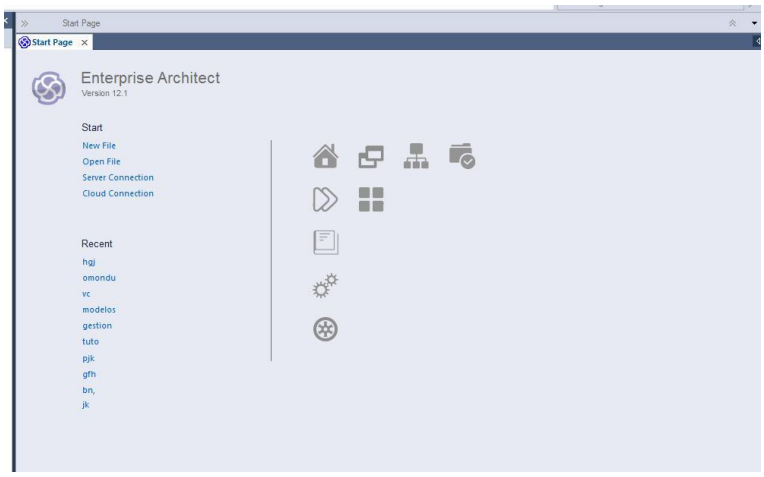


Figure II.1 :start menu

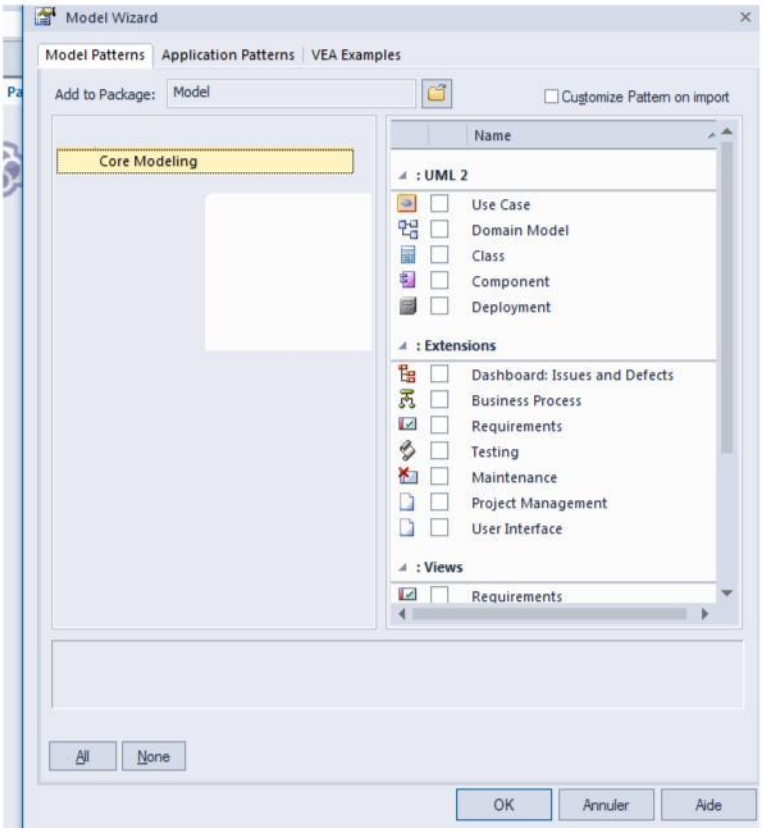
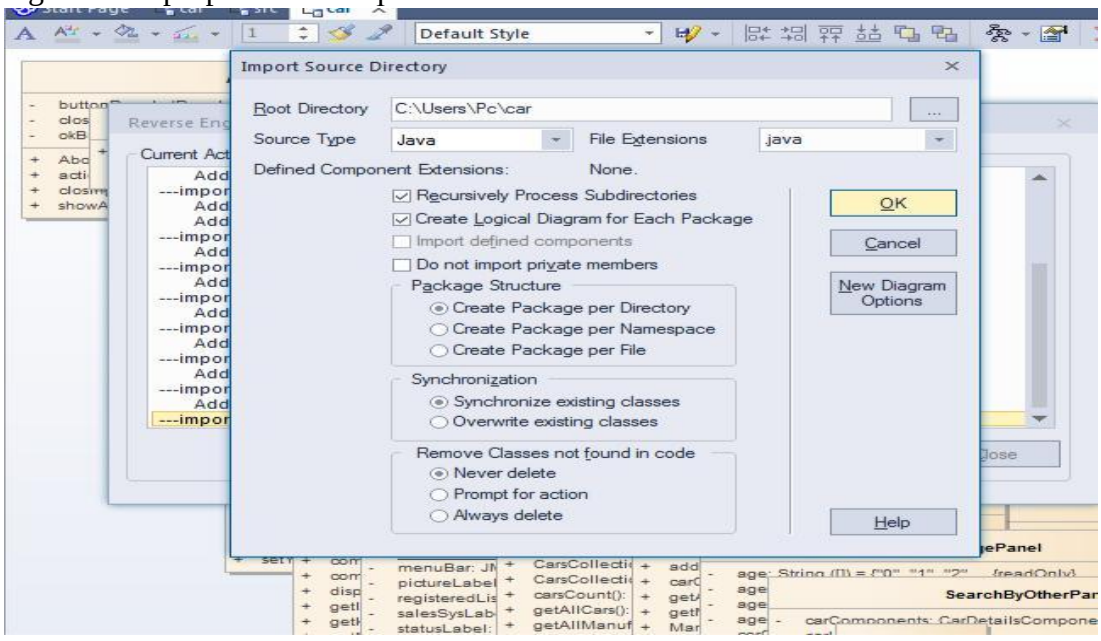


Figure II.2 : propriété d'entreprise Architect



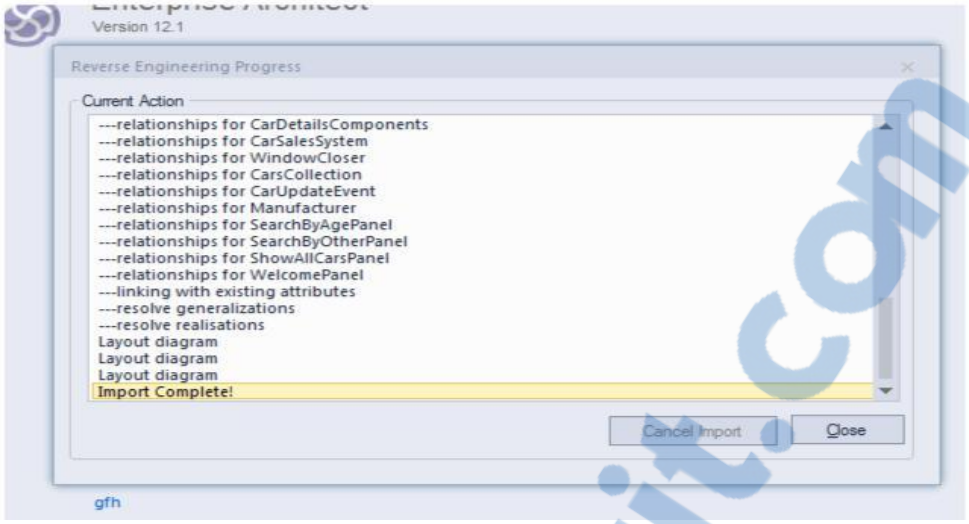
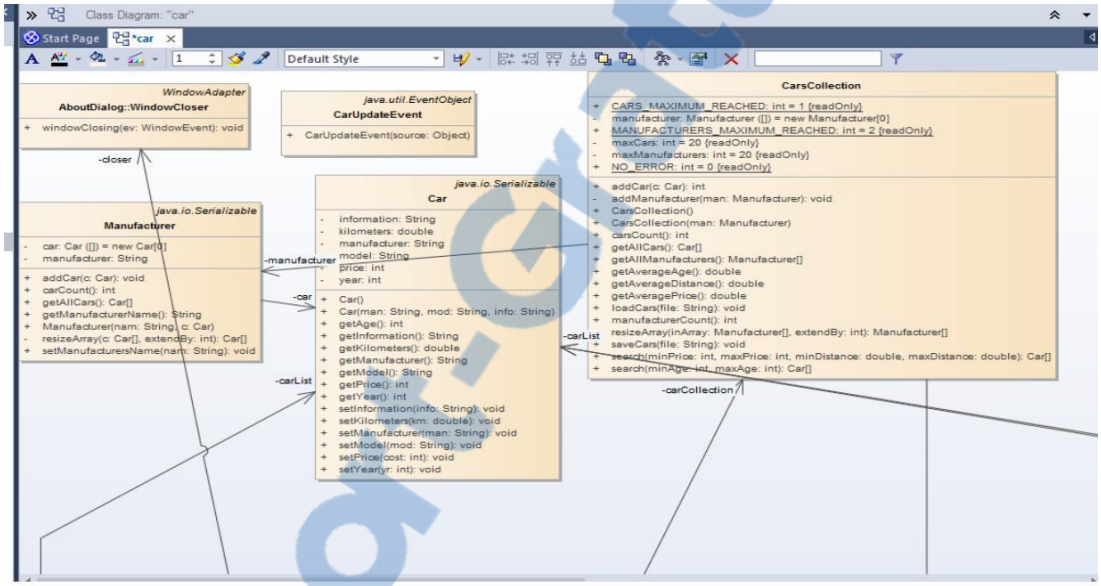
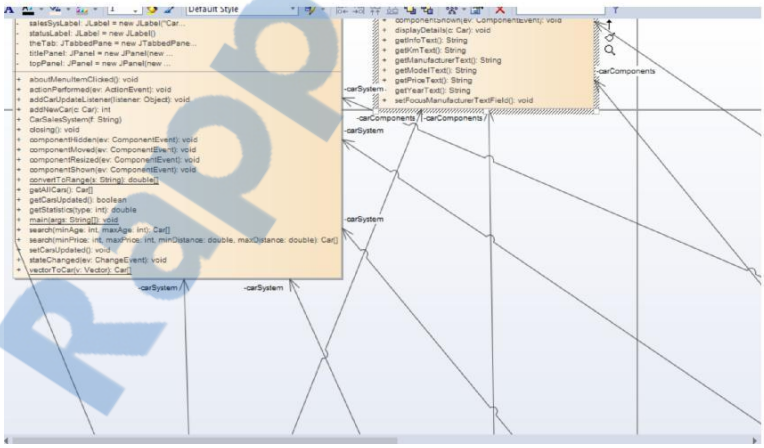


Figure II.3 : importation de code source





Rapport-Gratuit.com

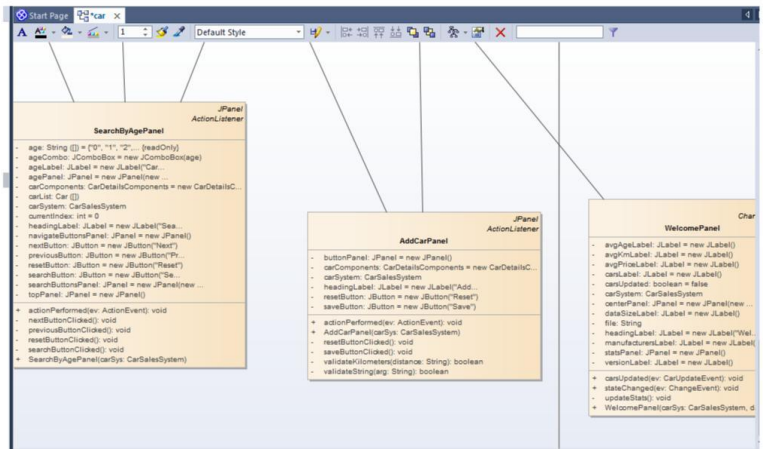


Figure II.4 : extraction de diagramme de classe.

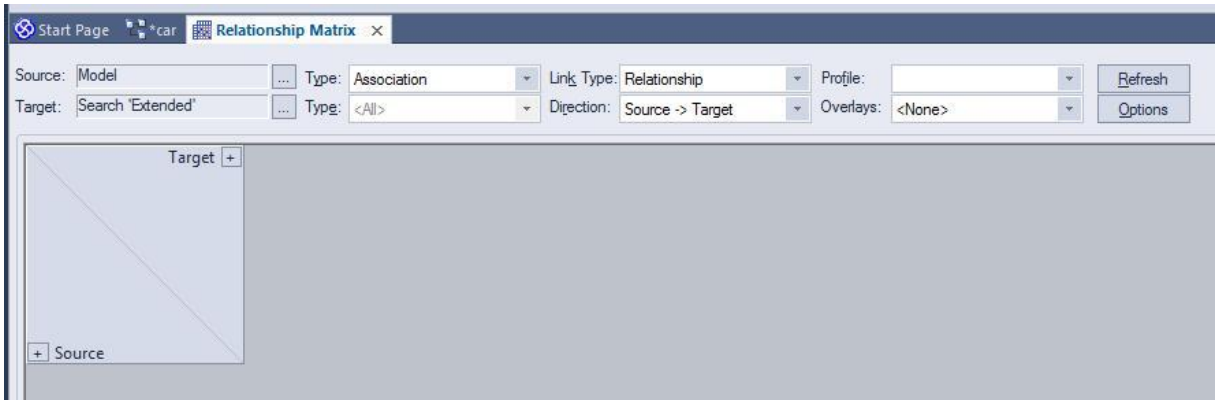
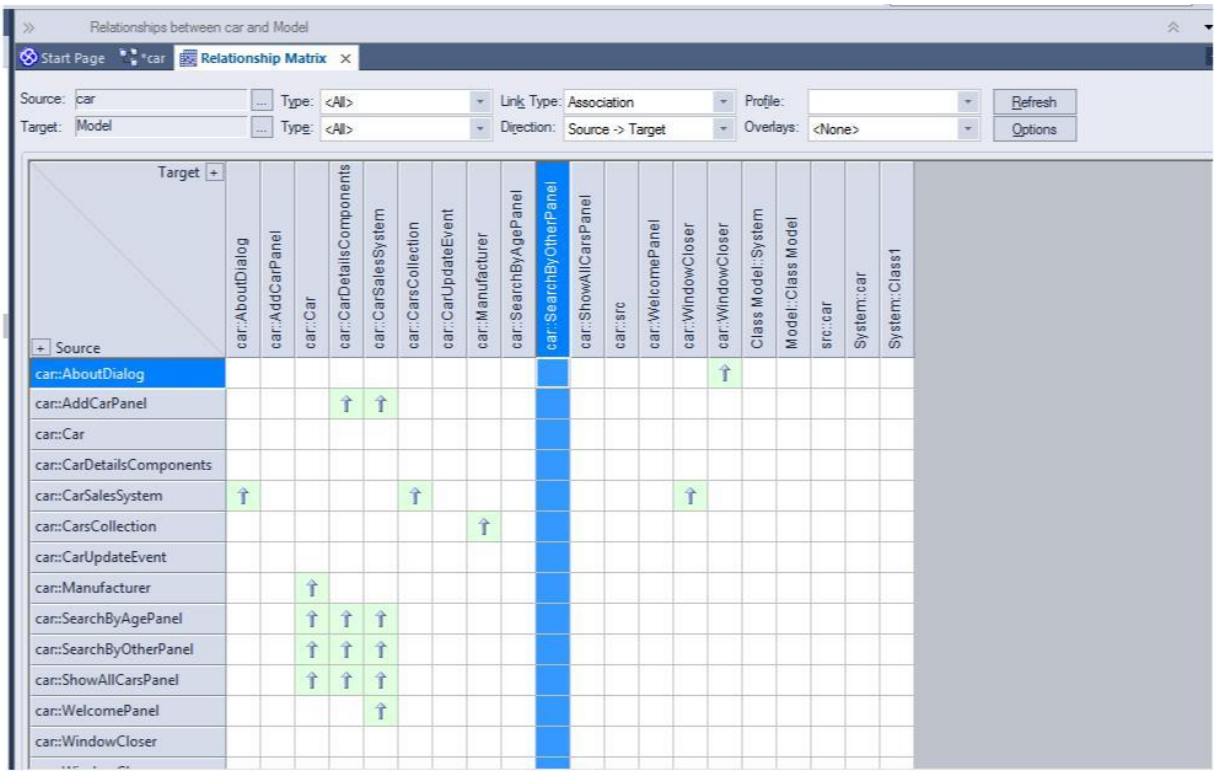


Figure II.5 :matrice de dépendance



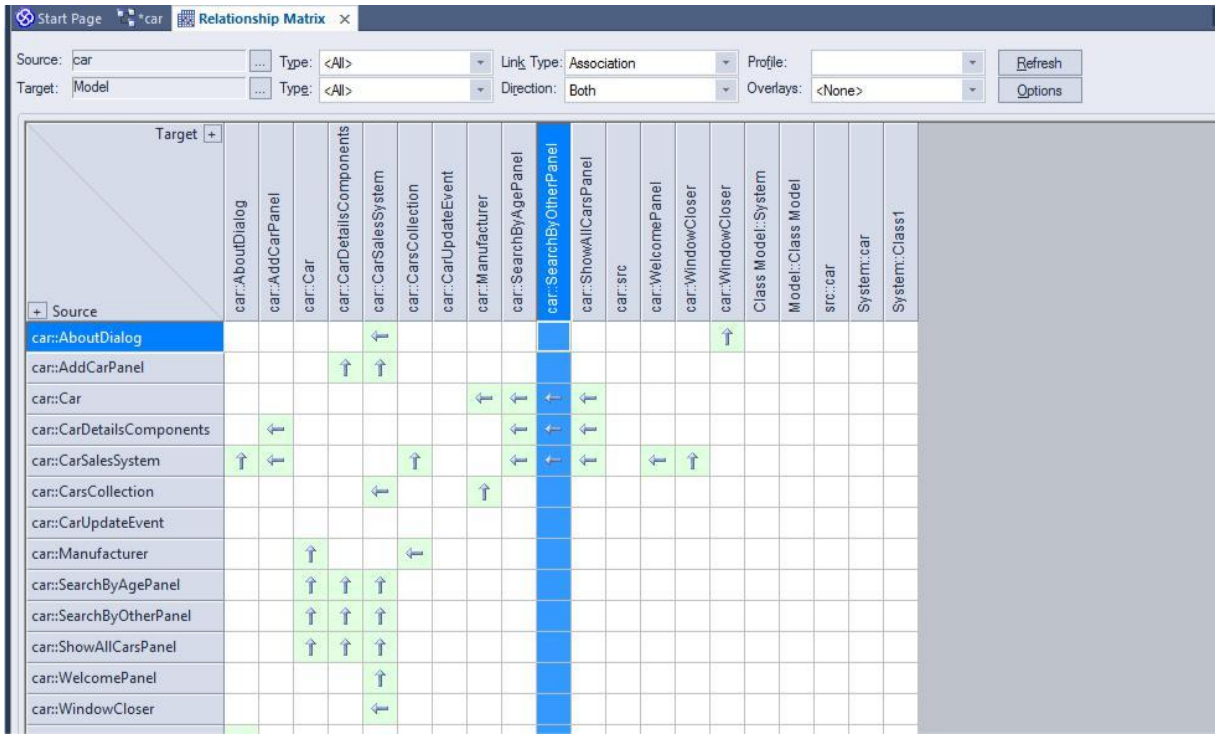


Figure II.6 : matrice des liens entre les classe de system

Conclusion

La rétro-ingénierie logicielle est le principe d'analyser un programme sans ses sources, pour en comprendre le fonctionnement interne. Deux approches complémentaires sont en général utilisées, l'analyse statique et l'analyse dynamique. Dans le premier cas il s'agit de reconstituer, partiellement ou totalement, le code source du logiciel à partir du programme exécutable ou au moins de le traduire du langage machine au langage assembleur. Pour le second cas, il s'agit d'étudier le programme directement pendant son exécution à l'aide d'un débogueur.

59

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

1. Introduction

La modification des systèmes est une tâche à la fois difficile et porteuse de conséquence sur la suite de l'évolution de ces systèmes. Les effets des changements subis par le système doivent donc être pris en considération. La motivation de notre travail est d'améliorer la maintenance des systèmes orientés objet, et d'intervenir plus précisément dans la tâche de l'analyse de l'impact de changement[40].
Nous visons principalement la réduction de l'effort ainsi que le coût de la maintenance. La réduction de cet effort peut être accomplie par la réduction du temps entre la proposition du changement, son implantation et sa réalisation, tout en assurant la qualité du système. L'effort peut être aussi réduit si on peut prédire le comportement du système face à d'éventuels changements. Notre travail se situe beaucoup plus dans cet axe de recherche. Plus l'analyse et la prédiction d'impact de changement est systématique, plus la réduction d'effort est à notre avis optimale. Ainsi les bonnes décisions peuvent être prises avant d'initier les changements. En identifiant l'impact potentiel d'une modification, on réduit le risque de s'embarquer dans des changements coûteux et imprévisibles. Plus un changement affecte de classes, plus le coût de sa réalisation n'est élevé. L'analyse des impacts de changement permettra ainsi d'évaluer le coût d'un changement et de faire un compromis entre les différents changements suggérés.

2. Définition de l'impact de changement

Un impact, dans son sens le plus large est l'effet ou l'influence d'une chose sur une autre. Dans notre étude, l'impact est la conséquence d'un changement. L'analyse d'impact est l'étude de la portée de cet impact.

L'analyse d'impact du changement est l'estimation des conséquences d'un changement sur le reste du système. Elle a été pratiquée de différentes manières pendant des années.
Pfleeger et Bohner [78] définissent l'analyse d'impact du changement comme :
"*the evaluation of the many risks associated with the change, including effects on resources, effort, and schedule estimates*". Arnold et Bohner [78] définissent l'analyse d'impact par : "*impact analysis is the activity of identifying what to modify to accomplish a change or of identifying the potential consequences of a change*".

Le développement des logiciels consomme beaucoup de temps et de ressources. Mais, les coûts de développement sont de loin inférieurs aux coûts de maintenance. En fait, ces coûts de maintenance représentent un souci majeur, même pour systèmes conçus avec des techniques avancées telle que l'orienté objet[41]. Dans les systèmes, un flot continu de changements doit être pris en compte sinon ces systèmes risquent de devenir désuets ou même d'être abandonnés. Il devient donc nécessaire que les systèmes objets développés soient évolutifs et capables d'absorber facilement tout changement. Une analyse d'impact permet en général d'évaluer les conséquences d'une modification avant de l'implémenter et d'estimer ainsi les coûts et risques associés à la modification.

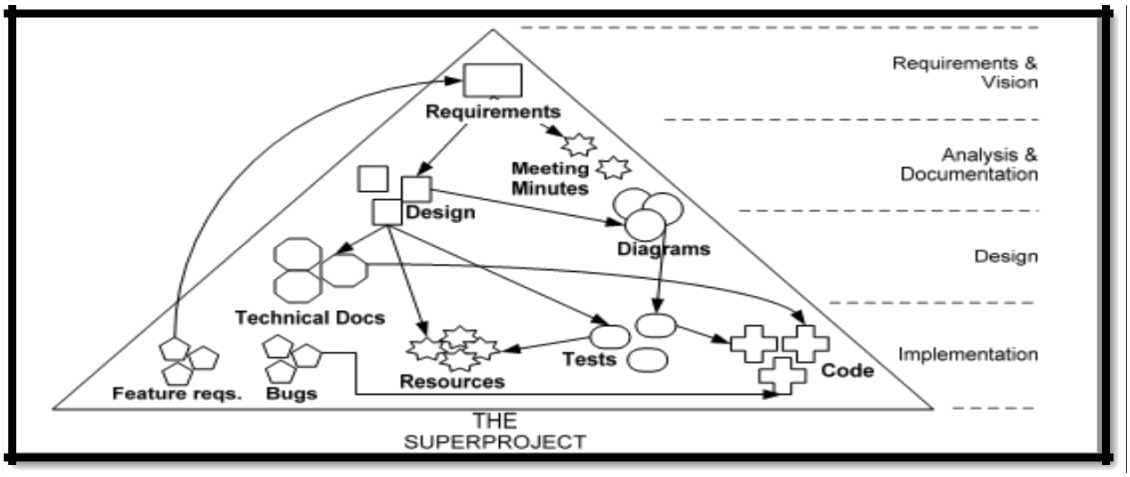


Figure III-1 :L'analyse de l'impact basée sur la traçabilité entre les artefacts [42]

3. Intérêts d'une analyse d'impact de changement

L'identification des impacts possibles avant d'effectuer le changement, nous permettra de réduire les risques d'entreprendre des changements coûteux. En effet, plus les problèmes liés à un changement sont découverts tard, plus leur coût augmente. Généralement, plusieurs changements sont proposés pour résoudre un même problème et satisfaire le même besoin. L'évaluation de l'impact de chaque changement permet de choisir le changement adéquat.
Selon Michelle L. Lee [78], l'expérience a montré qu'effectuer des changements dans un code sans comprendre leurs implications peut conduire à une mauvaise estimation de l'effort à produire, à prolonger les délais de livraison, à dégrader la conception d'un logiciel, à entamer la fiabilité du produit et enfin peut amener au retrait prématuré du logiciel[43]. Elle propose dans ses travaux une démarche constituée en 6 étapes :

- Convertir le changement proposé en des spécifications de changement,
- Extraire les informations à partir du code source et les convertir en une représentation Interne,
- Calculer l'impact de changement pour le changement proposé (refaire les trois premières étapes pour d'autres changements),
- Donner une estimation des ressources basée sur des considérations telles que la taille et la complexité du système.
- Analyser le coût et les bénéfices des requêtes de changements
- et enfin, le responsable de la maintenance du projet devrait aviser ses superviseurs de l'implication de la requête de changement. Ces derniers décident d'autoriser ou d'annuler le changement.

Pour la gestion de projet logiciel, l'analyse d'impact permet une meilleure estimation des Ressources, de l'effort et du temps associés à une activité de maintenance.
En identifiant l'impact potentiel d'une modification, on réduit le risque de s'embarquer dans des changements coûteux et imprévisibles.
Pour un programmeur, l'analyse d'impact peut avoir plusieurs objectifs dont l'évaluation de la complexité d'un code et la conduite des tests de régression. Si chaque modification d'une partie d'un code a un impact sur un grand nombre d'autres parties; cela indique une forte dépendance entre ces parties, un fort couplage et par conséquent une grande complexité.

4. Impact de changement orienté objet

4.1. Objectifs

La conception d'un système peut être décrite comme un ensemble d'artefacts logiciels (classes) qui sont en interaction[44]. Les liens d'inter-classes sont assumées avoir une plus grande influence sur la maintenabilité que les liens d'intra-classe **[**81**].** Ainsi, nous nous concentrons sur comment les divers liens entre des classes influencent en fait L'impact de changement. Quand un changement à un système est considéré, il est nécessaire d'identifier les composants du système qui seront impactés suite à ce changement. Cela permet de s'assurer que le système continue toujours à s'exécuter correctement après que le changement soit mis en œuvre. Notre intérêt est concentré sur comment le système réagit à ce changement et à d'autres changements en général. Notons qu'un système absorbe facilement un changement si le nombre de composants impactés est petit[45].

4.2. Modèle Conceptuel

Un système est vu comme un ensemble de classes connectées par différents liens. Une classe est définie comme un groupe de méthodes qui servent comme interface publique ou pour des opérations internes, et une section de variables qui définissent l'état des instances de la classe.

4.2.1 Changements

Nous définissons un changement à un système comme un changement qui peut s'appliquer à un composant. Un composant se réfère à une classe, une méthode, ou bien une variable. Comme exemples de changement, on peut avoir la suppression d'une variable, le changement de la portée d'une méthode, de "public" à "protected", ou le déplacement du lien entre une classe et son parent[51].
Un changement donné est caractérisé par une transformation du code quelque part dans le système. Si le système est recompilé avec succès, alors il n'y a aucun impact. Sinon, nous sommes face à un impact, c'est-à-dire, des modifications du code qui doivent être faites ailleurs dans le système pour obtenir un code syntaxiquement correct qui se recompilera.

La table 1 consigne les principaux changements aux systèmes orientés objets, au niveau conception.

Composant	***Description du Changement***
Variable	Changement de type de variable
	Changement de portée de variable Ajout de variable Suppression de variable
Méthode	Changement de type de retour de méthode
	Changement d'implémentation de méthode
	Changement de signature de méthode
	Changement de portée de méthode
	Ajout de méthode
	Suppression de méthode
Classe	Changement de structure d'héritage de classe Ajout de classe Suppression de classe

Tableau III-1 : Principaux changements au niveau conceptuel

4.2.2 Liens

Une fois qu'un composant donné est soumis à un changement, une partie spécifique peut être affectée, dans le cas où elle est liée au composant changé via un lien. Ces liens sont parmi les quatre types suivants :

S (association) : une classe fait référence aux variables d'une autre classe,
G (agrégation) : la définition d'une classe implique des objets d'une autre classe,
H (héritage) : une classe hérite les particularités définies dans une autre classe parente),
I (invocation) : les méthodes d'une classe invoquent des méthodes définies dans une autre classe.

Nous considérons aussi une notation spéciale généralement utilisée dans l'algèbre booléenne :

L'absence d'un opérateur entre 2 liens signifie une ***intersection***.

L'opérateur "+" signifie une ***union***.

L'opérateur "~" avant un lien signifie la ***négation***, *c'est-à-dire l'*ensemble des classes non associées par ce lien spécial, par exemple, **~G** signifie l'ensemble des classes qui ne sont pas liées à la classe indiquée par le lien d'agrégation. Les liens sont indépendants les uns des autres et nous pouvons trouver n'importe quel nombre et type de liens entre deux classes. Un changement d'une classe peut aussi avoir un impact dans la même classe.

Le pseudo-lien **L** (local) est introduit pour exprimer ceci.

Ainsi, pour un changement donné ch_i dans la classe cl_j , l'ensemble des classes impactées est exprimé par une expression booléenne dans laquelle les variables représentent les liens. Par exemple, la formule d'impact pour un changement hypothétique peut être donnée par : Impact (cl_j , ch_i) = **S~H+G** signifie que les classes qui sont en association (S) avec la classe changée cl_j et non dérivées (~H) de cette classe, ou les classes qui sont en agrégation (G) avec cl_j : sont impactées.

La Table II-2 suivante présente un exemple de changement pour chaque type de composant avec son expression d'impact :

Composant	***Description de changement***	***Expression d'impact*** **Impact (cl_j , ch_i)**
Variable	*Changement de type*	S+L
Méthode	*Changement de portée de"public" à "protected"*	I~H*
Classe	*Suppression*	H+G+S+I

Tableau III.2 : Exemples de Changements

Les considérations sémantiques relatives à la transformation du code ne sont pas considérées dans ce cas car elles ne peuvent pas être inférées du code source seulement. Puisque l'intérêt est centré seulement sur l'impact syntaxique d'un

changement, les mesures appropriées que nous devons appliquer sont basées sur l'impact qui dépend seulement de la nature statique du code source[52].

Enfin, notons que pour certains changements, on sait qu'il y a un impact (impact certain), tandis que dans d'autres cas, il est clair qu'il n'y a aucun impact (impact nul). Encore, dans quelques cas, on ne sait pas s'il y a un impact ou pas (impact incertain) avant que le morceau du code correspondant ne soit attentivement examiné. Par exemple, considérons le changement dans le type de retour d'une méthode abstraite. Les classes dérivées peuvent ou ne peuvent pas définir la méthode. Si la méthode n'est pas définie dans une classe dérivée, il n'y a aucun impact. Mais, si la méthode est définie dans une classe dérivée, alors il y a un impact dans la définition de cette méthode[54].

4.3.3. Firewall de classe

Pour avoir un certain degré de confiance dans un programme modifié, nous devons re-tester toutes les classes du firewall, car il constitue la portée maximale des conséquences d'une modification. Aucun effet de bord ne peut échapper au firewall. Une définition équivalente du firewall, consiste à dire qu'il s'agit de toutes les classes qui doivent être re-testées après qu'une classe ait été modifiée. Pour la modification de plusieurs classes, le firewall est la réunion des firewalls de chaque classe prise individuellement[55].

5. Adaptation du modèle à Java

Le modèle défini au niveau conceptuel (Table II.1) a été déjà adapté en C++. **[**82**]** ,le travail de [89] vise les systèmes logiciels codés en Java, une opération d'adaptation de ce modèle à ce langage a été nécessaire. A cet effet [89] a examiné l'adaptation déjà faite dans **[**88**]** et [87]. Il a remarqué qu'il y a certains changements qui sont communs aux deux langages, dans le sens où ils peuvent être appliqués aux deux langages. A titre d'exemple, le changement de type de variable, changement de signature de méthode, changement de la structure d'héritage d'une classe, etc.

Par contre, il y a d'autres changements qui sont propres au langage C++, principalement les concepts de "virtual" (méthode virtuelle ou classe virtuelle) et " friendship" (classe amie). La Table II.4 présente quelques exemples de ces changements. Les résultats de l'étude d'adaptation du modèle en question à C++ sont présenté :

Concept	***Description de changement***
Virtual (virtuelle)	virtuelle → non virtuelle
	Pure virtuelle → virtuelle
Friendship (Amitié)	Suppression de classe amie
	Ajout de classe amie
….	**….**

Tableau III.3 Exemples de changements propres à C++

Le concept de "virtual" est introduit en C++ pour gérer les appels dynamiques. Le rôle joué par ce concept en C++ est assuré en Java par le biais de sa machine virtuelle. Le fait de précéder la signature d'une méthode par le mot clé "virtual" en C++ ou "abstract" en Java signifie que cette méthode doit être surchargée "overiding" dans ses classes dérivées. Une erreur lors de la compilation est donnée si une méthode virtuelle pure (en C++) ou abstraite

(en Java) n'est pas redéfinie dans les classes dérivées

Conclusion

Dans ce chapitre, nous avons présenté l'analyse d'impact de changement ainsi que ses intérêts, le modèle d'impact de changement.

1. Introduction et contribution

Ce chapitre est dédié à notre contribution qui consiste à l'étude d'impact de changement pour les systèmes orientés objets à l'aide d'une implémentation d'un outil , en tenant compte de la taille importante des systèmes OO, la complexité de l'opération de l'extraction des constituants du modèle conceptuel du code source (les différents composants du système), la détection de l'impact de changement et sa propagation dans le système.

2. Approche proposé

Afin de vérifier empiriquement ce modèle d'impact raffiné en JAVA , nous pensons qu'il faut effectuer plusieurs expérimentations sur divers systèmes (codés en JAVA) en procédant de la manière suivante [44], [79] : 1.Extraction des constituants du modèle conceptuel du code source 2.Représentation graphique de la structure du système (pour visualiser les interactions entre classes) 3.Alimentation des matrices des liens 4.porter des changements concrets sur le système (code)

Explication de l'étude expérimentale :

.

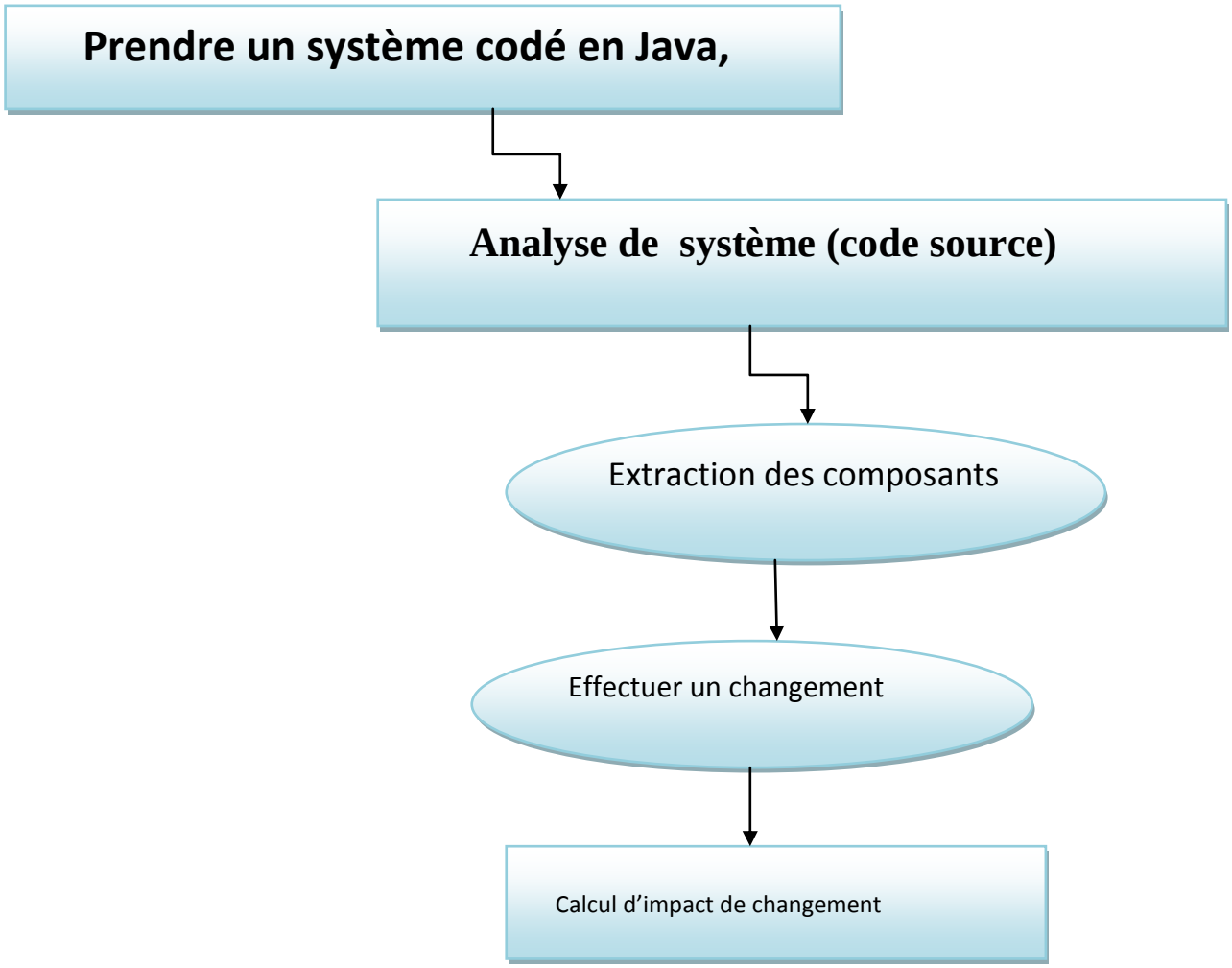


Figure V.1: Schéma globale de l'approche proposé

2.1. Extraction des composants :

L'objectif consiste à extraire les différents composants de system source :

Agrégation G examine les classes qui sont en agrégation avec le système

Association S examine les classes qui sont en association avec le système

Héritage H examine les classes qui sont en héritage avec le système

Invocation I examine les classes qui sont en invocation avec le système

2.2. Matrice des liens:

La matrice des liens montre les différentes relations entre classes, elle est alimentée lors de la phase d'extraction par AG, AH, AS, AI qui renseignent chaque lien localisé entre deux classes.

2.3. Présentation graphique:

Dans l'objectif de mettre en évidence les liens entre les différents composants du système cette représentation graphique vient faciliter la compréhension du système et illustrer la dépendance entre les différentes classes en visualisant les liens (Héritage, Association, Agrégation et Invocation) qui existent entre eux.

2.4. Table d'impact selon le modèle:

En se basant sur la matrice des liens calculée et l'expression booléenne générée par le modèle d'impact raffiné , nous obtenons la table d'impact selon le modèle.

3. Réalisation :

3.1. Outil de développement

Nous avons implémenté un outil sous l'environnement jbuilder.x

3.1.1.jbuilder.x

Nous avons implémenté un outils sous l'environnement jbuilder.x pour deux raisons :

A.JBuilder est un environnement de développement intégré pour Java, permettant le RAD, et édité par Borland. L'application est elle-même développée en grande partie en Java.

B.JBuilder apporte certaines fonctionnalités spécifiques, disposant notamment d'une JVM propre, permettant notamment l'exécution de code Java pas à pas.

Selon les éditions, il ne permet que la réalisation d'applications clientes (J2SE) ou également serveur (J2EE). Des modules supplémentaires, pour les applications mobiles, en particulier pour les téléphones mobiles et les PDA, ou pour les services Web, sont également disponibles.

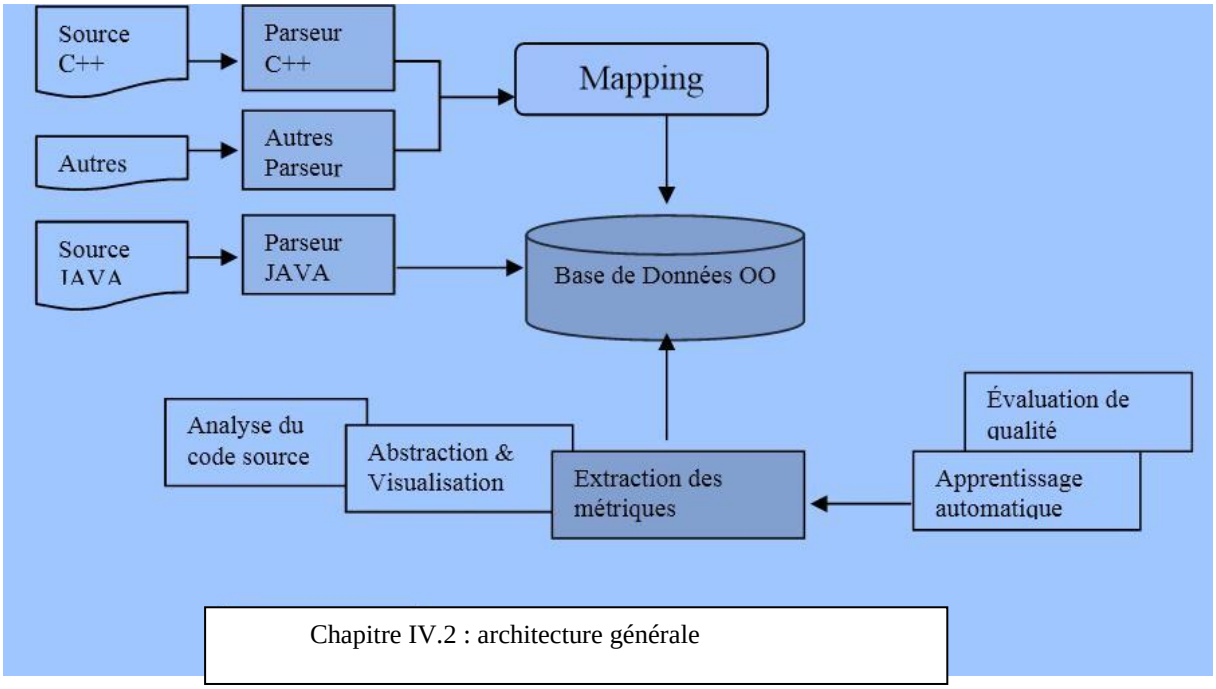
Certaines versions intègrent des solutions de conception UML ou des outils de débogage et de test.

Rapport-Gratuit.com

JBuilder est disponible sur Windows, Linux, Solaris et Mac OS X.

La version 2008 de JBuilder n'est pas éditée par Borland mais par Codegear, la dernière version éditée par Borland est JBuilder 2007. Codegear a été acheté par l'éditeur Embarcadero en mai 2008.

Pour l'analyse syntaxique de code source JAVA en entré, nous avons utilisé une machine virtuelle qui nous a permet d'installer jbuilder.10 parceque il est compatible sauf avec windows xp et à partir de code source avoir son arbre syntaxique[82]



Chapitre IV.2 : architecture générale

3.2 : implémentation de l'outil :

```
package uml_fin;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class Cadre1_AboutBox extends JDialog implements ActionListener {

  JPanel panel1 = new JPanel();
  JPanel panel2 = new JPanel();
  JPanel insetsPanel1 = new JPanel();
  JPanel insetsPanel2 = new JPanel();
  JPanel insetsPanel3 = new JPanel();
  JButton button1 = new JButton();
  JLabel imageLabel = new JLabel();
  JLabel label1 = new JLabel();
  JLabel label2 = new JLabel();
  JLabel label3 = new JLabel();
  JLabel label4 = new JLabel();
  ImageIcon image1 = new ImageIcon();
  BorderLayout borderLayout1 = new BorderLayout();
  BorderLayout borderLayout2 = new BorderLayout();
  FlowLayout flowLayout1 = new FlowLayout();
  GridLayout gridLayout1 = new GridLayout();
```

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

```
String product = "";
String version = "1.0";
String copyright = "Copyright (c) 2009";
String comments = "";
public Cadre1_AboutBox(Frame parent) {
  super(parent);
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}
//Initialiser le composant
private void jbInit() throws Exception  {
  image1 = new ImageIcon(uml_fin.Cadre1.class.getResource("about.png"));
  imageLabel.setIcon(image1);
  this.setTitle("A propos");
  panel1.setLayout(borderLayout1);
  panel2.setLayout(borderLayout2);
  insetsPanel1.setLayout(flowLayout1);
  insetsPanel2.setLayout(flowLayout1);
  insetsPanel2.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
  gridLayout1.setRows(4);
  gridLayout1.setColumns(1);
  label1.setText(product);
  label2.setText(version);
  label3.setText(copyright);
  label4.setText(comments);
  insetsPanel3.setLayout(gridLayout1);
  insetsPanel3.setBorder(BorderFactory.createEmptyBorder(10, 60, 10, 10));
  button1.setText("Ok");
  button1.addActionListener(this);
  insetsPanel2.add(imageLabel, null);
  panel2.add(insetsPanel2, BorderLayout.WEST);
  this.getContentPane().add(panel1, null);
  insetsPanel3.add(label1, null);
  insetsPanel3.add(label2, null);
  insetsPanel3.add(label3, null);
  insetsPanel3.add(label4, null);
  panel2.add(insetsPanel3, BorderLayout.CENTER);
  insetsPanel1.add(button1, null);
  panel1.add(insetsPanel1, BorderLayout.SOUTH);
  panel1.add(panel2, BorderLayout.NORTH);
  setResizable(true);
}
//Redéfini, ainsi nous pouvons sortir quand la fenêtre est fermée
protected void processWindowEvent(WindowEvent e) {
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    cancel();
  }
  super.processWindowEvent(e);
}
//Fermer le dialogue
```

```
  void cancel() {
    dispose();
  }
  //Fermer le dialogue sur un événement bouton
  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == button1) {
      cancel();
    }
  }
}
```

Figure V.3 : fenêtre principal

Plusieurs systèmes ont été considérés dans notre étude .
Afin de dérouler cette démarche nous considérons seulement les résultats du système le plus important du point de vue taille (nombre de classes), à savoir le système de “gestion de compagnie aérienne” qui comporte un total de 10 classes

Après exécution de cet outil implémenté sous jbuilder

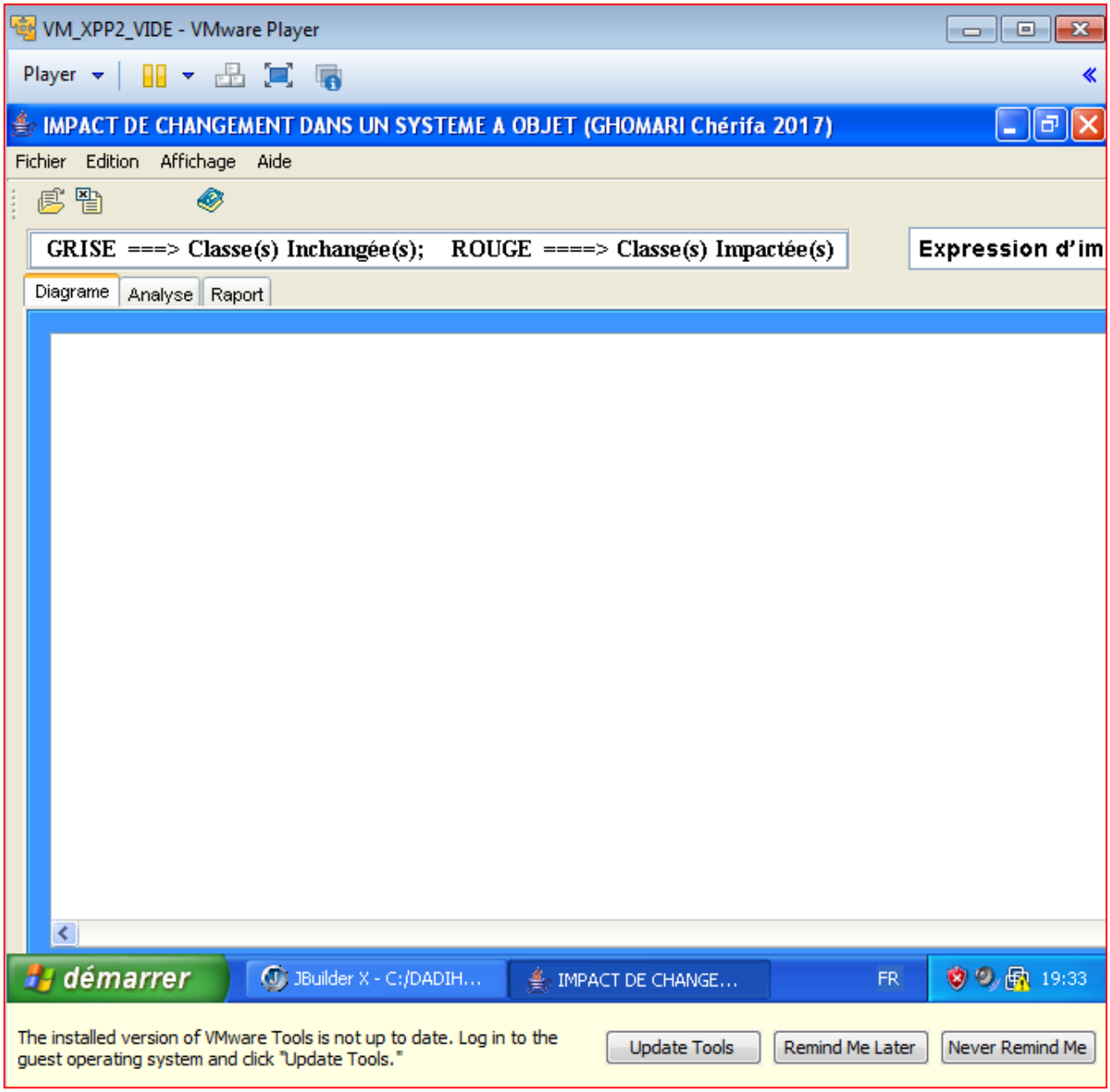


Figure IV.4.interface de l'outil

on vas importer un système parmi les systèmes :

Nom du système	**Taille (nombre de classes)**
Animalerie	**04**
Plantes	**04**
Location	**06**
Gestion de bibliothèque	**07**
Automobile	**08**
Gestion de compagnie aérienne	**10**

Figure v.3 : exemple de code source
*

Après execution de l'outil implémenté les différente liens entre les classe sont localisé et on établit la représentation graphique a l'aide de cet outil :

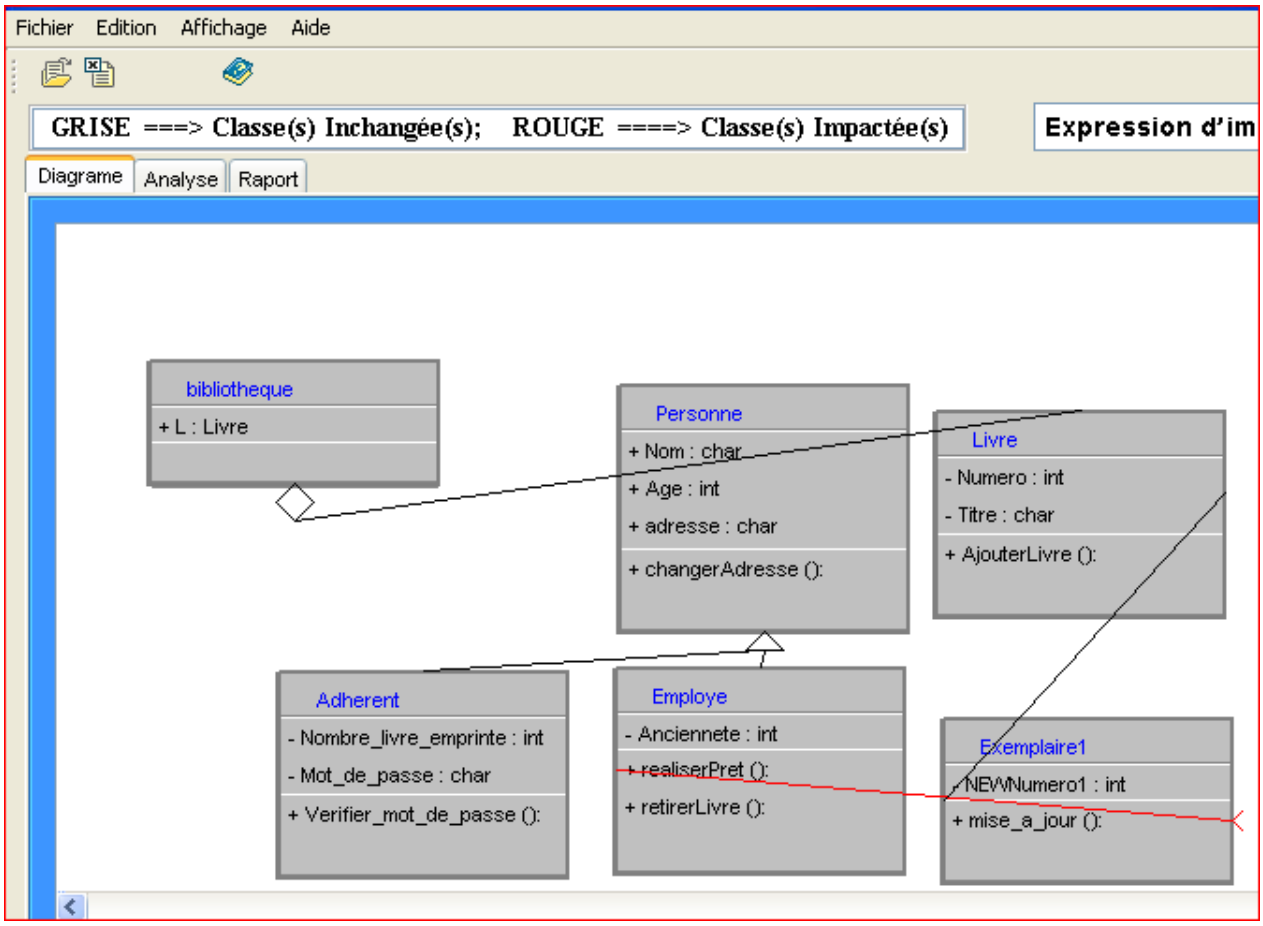


Figure IV.5 Visualisation de code source exporté

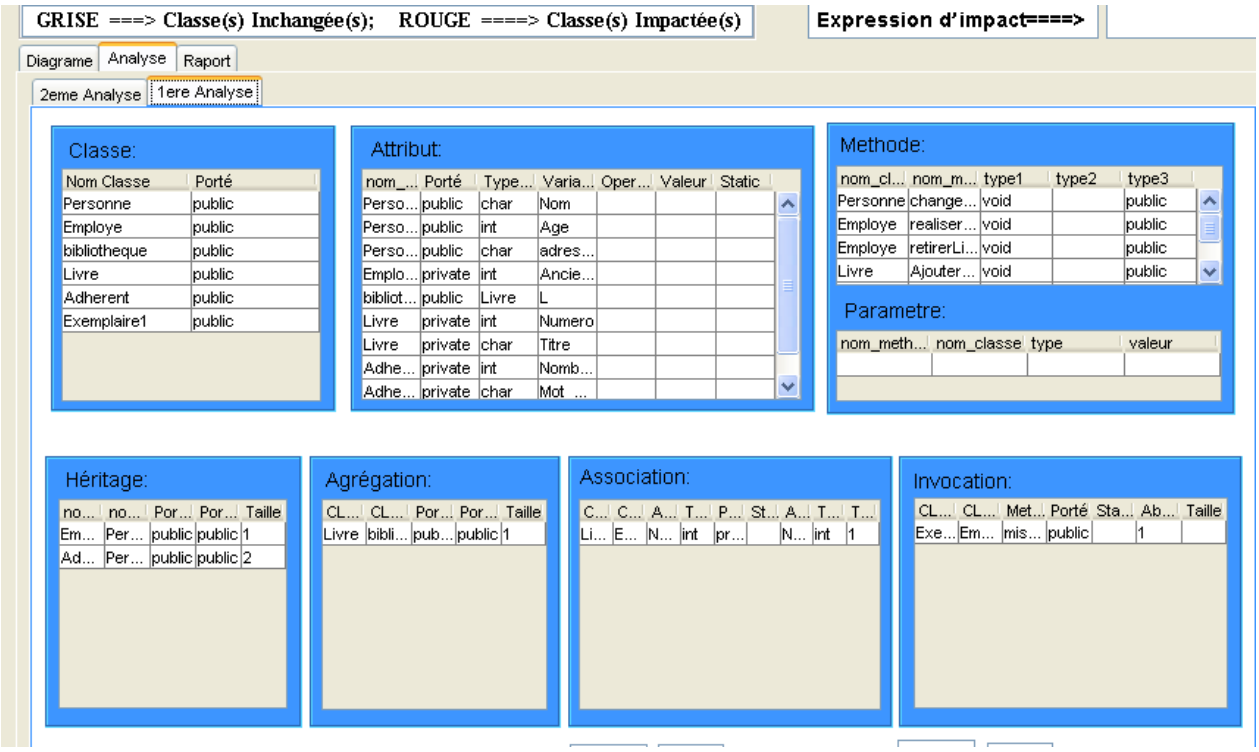


Figure IV.6.slancemment d'analyse et extraction des composants système

Une fois la phase d'extraction est terminée, on effectue une phase d'analyse de ses composants , pour mieux le présenter

Phase d'impact de changement :

Dans cette étape nous effectuons des changements sur le code source,relencer les agents pour détecter l'impact du changement .Les changements choisis dans notre étude :

a) **Portée de méthode** : changement de la porté de la méthode getName de « public » a « private » dans la classe GraphNode.
b) **Type de variable** :changement de type de la portée CLASS-TYPE de « int » a « string » dans la classe GraphNode.
c) **Suppression de la classe** : suppression de la classe GraphNode.
d) **Suppression d'une variable** : suppression du porté INTARFACE-TYPE dans la classe GraphNode.

Les changement sont porté concrètement sur le système code

l'imapct deduit par le modele prends toujours lavleur maximaledu changemment.Si on prendsle changemment de type variable ,selon le modele de l'impact toutes les classes en assotiations avec la classe changée et la classe elle-même seront impactées tandis que notre système détecte seulement les classes qui font référence à la variable modifiée.

Conclusion

Au terme de ce chapitre, nous avons présenté une approche de validation d'un modèle d'impact de changement à base d'un outil, nous avons détaillé l'architecture du système. Aussi nous avons présenté l'étude d'impact de changement réelle sur le système.

Conclusion générale et perspectives

Réduire le coût de la maintenance nécessite de se procurer les moyens nécessaires pour l'accomplir. L'analyse de l'impact du changement est l'une des techniques qui a connu de l'importance dans le domaine de la maintenance ces dernières années. Le but de cette technique est de permettre aux gens responsables de la maintenance d'évaluer le coût du changement à priori, i.e., avant d'entamer l'implémentation du changement. Cette évaluation se fait de deux façons : technique et gestion. La première, évaluation technique, se base sur la compréhension du logiciel en modification et sur l'identification du changement, son impact et ses effets de propagation sur tout le système. Suite à cette analyse technique, l'équipe de maintenance est en mesure de conclure sur la faisabilité technique du changement et proposer des solutions optimales qui ne compromettent pas les fonctionnalités du système existant. La deuxième, évaluation du point de vue gestion (ressources humaines et financières), est établie habituellement par les gestionnaires. Ces derniers peuvent choisir parmi les solutions proposées celle qui satisfait les besoins ou décider de réétudier le changement pour des solutions plus sûres.

L'analyse d'impact du changement a été étudiée de différentes manières par différents utilisateurs, en se basant sur un ensemble de concepts. Notre approche d'aide à la décision pour la gestion des projets de maintenance logicielle utilise l'analyse d'impact de changement dans les programmes à objets à base des métriques qui mesurent une propriété architecturale des logiciels qui est le couplage.

En premier lieu, nous avons procéder à l'extraction de certaines métriques relatives à la propriété architecturale que l'on veut évaluer ; il s'agit du couplage. Nous avons développé un outil sous la plate-forme ECLIPSE qui permet d'analyser un programme orienté objet Java et d'extraire les métriques nécessaires. Cette étape s'est avérée importante pour mener notre expérience.

Notre approche aide le manager de maintenance à choisir une solution parmi plusieurs en tenant en compte de deux aspect : la nature de changement et sa propagation dans le reste du système.

L'intérêt est de :

-réduire le temps entre la proposition du changement, son implantation et sa réalisation.

-Choisi la solution optimale.

-Et par conséquent la réduction du coût de la maintenance.

-réduire le temps entre la proposition du changement, son implantation et sa réalisation.

-Choisi la solution optimale.

-Et par conséquent la réduction du coût de la maintenance.

Rapport-Gratuit.com

En résumé, nos contributions dans ce travail sont :

-Le développement d'un outil qui sur la plateforme Eclipse qui permet d'analyser et d'extraire les métriques de couplage des programmes à objets développés en Java.

-L'aide qui peut apporter notre outil pour le manager de maintenance dans son choix de la solution optimale.

-Ce mémoire a permis d'ouvrir de nouveaux horizons de recherche. Parmi ces axes, trois nous ont paru intéressants. Le premier axe de recherche est d'étendre cette approche et permettre de la généraliser pour supporter d'autres langages de programmation. Néanmoins, il s'avère que cette généralisation est plus compliquée.

La deuxième voie de recherche est celle de la finalisation de l'outil que nous avons développé pour englober toutes les métriques de conception orientée objet nécessaires pour l'évaluation des différentes caractéristiques de qualité de logiciel. Cette extension s'avère importante et bénéfique, car avec Eclipse tous les produits sont des plug-ins que nous pouvons intégrer à n'importe quel projet.

Enfin, bien que les résultats obtenus soient encourageants, il reste à généraliser l'approche présentée dans ce mémoire afin de renforcer la pertinence des résultats. En effet, mener d'autres expériences avec notre approche serait d'un grand apport au domaine de la maintenance et en particulier au domaine de l'analyse d'impact du changement. L'utilisation d'une solution comme un ensemble de changements donnera un plus significatif à la proche, Cela contribuera largement à la détection des facteurs qui augmentent le coût de la maintenance, et à procurer aux développeurs les moyens nécessaires pour concevoir des logiciels qui sont moins coûteux.

Biliographie

[1] Martin Flower est auteur de plusieurs ouvrages dans le domaine de la conception logicielle et de l'approche Agile.

[2]Robert John ALLEN. A formal approach to software architecture.
Thèse de Doctorat, Carnegie Mellon University, Pittsburgh, PA, USA, 1997.
Chair-David Garlan,
[3]*IEEE*.Chik*Software,*fskyndJ. *7(1):13*I.Cross.Reverse*–17,* engi*Jan*eering*1990*and design recovery: A taxonomy.

[4] E. Eilam. Reversing: Secrets of Reverse Engineering. Publishing, Inc. ISBN-10: 0-
[5] J. Ka-Yee Ng. Identification of Behavioral and Creational Design Patterns through Dynamic*7645-7481*Analysis-*7*..Proceedings*2005*. of the 3rd International Workshop on Program
Comprehension through Dynamic Analysis (PCODA), pages 34-42, October 2007.

[6] T. Systa. Static and dynamic reverse engineering techniques For Java Software Systems. Ph.D Thesis, University of Tampere, Dept. of Computer and Information Sciences, Report A-2000-4, 2000.

[7] Bieman J. M., Kang B.-K., « Cohesion and reuse in an object-oriented system », Proc. of the *Symp. on Software reusability,SSR '95, ACM Press, USA, p. 259-262, 1995.*

[8]L.J. Arthur. Software Evolution: The Software Maintenance Challenge. John Wiley & sons, 1998

[9] ObjectManagementGroup,CorbaComponentModel,
http://www.omg.org, 1999
[10] W. De Pauw, E. Jensen, N. Mitchell, G. Sevitsky, J. Vlissides, J. Yang. Visualizing the Execution of Java Programs. Revised Lectures on Software Visualization, International *Seminar. pp. 151–162, 2001.*

[11] Guillemet A., Haïk G., Meurisse T., Briot J-P., and Lhuillier M,. Une approche à base de composants pour la conception d'agents. Journées Francophones pour l'Intelligence Artificielle Distribuées et les Systèmes Multi-Agents (JFIADSMA'99), Ile de la Réunion, Nov. 1999, France.Edition Hermès.

[12] Drogoul A., De la Simulation Multi-Agents à la Résolution Collective de Problèmes. Thèse de
[13]Doctorat,Peschanski Université F.,Comet:Architecture Paris6, Réflexive 1993. à Base de Composants pour la Construction d'Applications Concurrentes et Réparties. LMO'2K, 26-28 Janv. 2000, Mont St-Hilaire, Canada.
Edition Hermès.

Rapport-gratuit.com
LE MEMOIRE ET THESE GRATUITES

[14] S. A. Bohner, R. S. Arnold, An Introduction to Software Change Impact Analysis.

Software Change Impact Analysis IEEE Computer society, Press Los Alamitos,

[15]Grady Booch, Object-Oriented Analysis and Design with Applications. Second Edition, 1996.
Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.

[16] Evans H., DickMan P., Zones, Contracts and Absorbing Change : An Approach to Software Evolution, Proceedings of OOPSLA'99, ACM Press, Oct. 1999.

[17] Shaw M., Deline R., Klein D. V., Ross T. L., Toung D. M., Zelesnik G., Abstractions for Software Architecture and Tools to Support Them, IEEE Trans. On Software Engineering, vol. SE-21(N.4) avril 1995, p. 314-335.

[18]L.C. Briand, J. Daly, J. Wuest, A Unified Framework for Coupling Measurement in
Object-Oriented Systems. IEEE Transactions on Software Engineering, 1999.
[19] Briand L. C., Daly J. W., and Wüst J. K., A Unified Framewoek for
Coupling Measurement in
Object-Oriented Systems, IEEE Trans. On Software Engineering, vol.
[20]SEGuerraoui-25(N.1)R.January/February,WhatObject-OrientedDistributed Programming
may be – and What it does not have to Be. White Paper. HP Labs. 1999
1999, p. 91-121.
[21] J. BOSCH.
Design and Use of Software Architectures : Adopting and Evolving a Product Line Approach. Pearson Education (Addison-Wesley & ACM Press), 2000.

[22] Don BOX. Essential COM.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. Foreword By-Booch,,
[23]GradyLionelandBRIAND,ForewordPremDEVANBUBy-Kindel,,etWalcelioCharlie,MELO.
An investigation into coupling measures for C++. In Proc. of the Int. Conf. on
Software Engineering, pages 412–421. ACM, 1997.

[23] A. Hamou-Lhadj, T. C. Lethbridge. A Survey of Trace Exploration Tools and
Techniques. Proceedings of the 2004 conference of the Centre for
[24] Holger BÄR.
Famix Advanced c++language Studies plug-in1.0.Rapport on technique, University of Berne, September 1999.
Collaborative[25]G.Canf research,oraand Mp.42Di-55,PentaOctober.New04Frontiers-07,2004,
ofMarkham,Reverse Ontario,Engineering Canada.International
Conference on Software Engineering, 2007 Future of Software
[26]A JosephframeworkP.CAVANOetJamesforA. MCCALLthe. measurement of
Engineering, pp. 326-
software quality.
341, May 2007.
[27] L. Audibert. UML2. Éd. 2007-2008.

[27] Djalel CHEFROUR et Françoise ANDRÉ. Aceel :
modèle de composants auto-adaptatifs.
In Systèmes à composants adaptables et extensibles, Grenoble, France,
[28]Grenoble,Siddhartha CHIBFrance,etEdward2002GREENBERG.Understanding the metropolis-hastings algorithm.The American Statistician, 49(4) :327–335, 1995.

[29] E.Gamma Richard Helm, Ralph Johnson, John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software.1995.

[30] Shyam R. CHIDAMBER et Chris F. KEMERER.
Towards a metrics suite for object oriented design.
In OOPSLA ’91 : Conference proceedings on Object-oriented programming systems, languages, and applications, pages 197–211, New York, NY, USA, 1991.
ACM Press.

[31] K. Kowalczykiewicz and D. Weiss. Traceability: Taming uncontrolled change in software development. In Proceedings of the IV KKIO Conference, Tarnowo Podgrne,

Poland, pages 33-42, 2002.
[31] Paul C. CLEMENTS, Rick KAZMAN et Mark KLEIN.
Evaluating software architectures : methods and case studies.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[32] Jacob COHEN.A coefficient of agreement for nominal scales.
Educational and Psychological Measurement, 20(1) :37–46, April 1960.
[33] Bernard COULANGE. Software Reuse. Springer-Verlag, 1998.

[34] C. DARWIN. On the Origin of Species A facsimile of the first edition.
Harvard University Press, July 1975.
[35] Remco C. de BOER et Hans van VLIET.
Architectural knowledge discovery with latent semantic analysis : Constructing a reading guide for software product audits. J. Syst. Softw., 81(9) :1456–1469, 2008.

[36] Serge DEMEYER, Stéphane DUCASSE et Oscar Marius NIERSTRASZ.
Object-oriented reengineering patterns.
Morgan Kaufmann, 2003.
[37] Serge DEMEYER, Stéphane DUCASSE et Er TICHELAAR.
Why unified is not universal ? uml shortcomings for coping with round-trip engineering.
In Proceedings UML’99 (The Second International Conference on The Unified
[38]ModelingSergeDEMEYER,Language),SanderTICHELAAR et Patrick STEYAERT.
Famix 2.0 - the famoos information exchange model.
Rapportvolumetechnique,1723,Universitypages 630ofBerne,–645August.Springer1999.-Verlag, 1999.

[39] D. DOVAL, S. MANCORIDIS et B. S. MITCHELL.
Automatic clustering of software systems using a genetic algorithm. In STEP ’99 : Proceedings of the
Software Technology and Engineering Practice, page 73, Washington, DC, USA, 1999.
IEEE Computer Society. 212 BIBLIOGRAPHIE

[40] J. DRÉO, A. PÉTROWSKI, P. SIARRY et É. D. TAILLARD.
Métaheuristiques pour l'optimisation difficile . Eyrolles, septembre 2003.

[41] Emanuel FALKENAUER. Genetic Algorithms and Grouping Problems.
[42]John.-M.WileyFAVRE, &H. CEVANTES,Sons,IncF.,DUCLOS,NewYork,R.SANLAVILLENY,USA,etJ.ESTUBLIER.1998.
Issues in reengineering the architecture of component-based software.
In In Proceedings of the WCRE 2001 Discussion forum on Software Architecture Recovery and
[43]ModelingP.FINNIGAN,(SWARM.HOLTet2001AL..The.CWI,software2001bookshelf. .
IBM systems Journal, 36(4) :564–593, novembre 1997.

[44] International Organization for STANDARDIZATION.
ISO 9126-1 Software Engineering - Product Quality - Part 1 : Quality Model.
International Organization for Standardization, 2001.

[45] H. GALL et R. KLOSCH.
Finding objects in procedural programs : an alternative approach.
In WCRE '95 : Proceedings of the Second Working Conference on Reverse Engineering, page 208, Washington, DC, USA, 1995. IEEE Computer Society.

[46] M. Lindvall, Measurement of Change: stable and Change-prone constructs in a

commercial C++ System. In Proceedings of the 6 th International Software Metrics Symposium, Pages 40-49, Nov.1999.
[47] H.Lounis, W.L.Melo, Identifying and Measuring Coupling on Modular Systems. 8 th

International Conferenceon Software Technology (ICST97)- 1997.
[48] Joseph P. Loyall and Susan A. Mathisen, Using Dependence Analysis to Support the Software Maintenance Process. Confer ence on Software Maintenance, September 1993, Pages 282-291.

[49] David GARLAN et Mary SHAW.
An introduction to software architecture. Rapport technique, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[50]S. L. Pfleeger and Shawn A. Bohner, A Framework for Software Maintenance Metrics. In proceedings of the Conference on Software Engineering, May 1990, Pages 320-327.

[51] Robert B. GRADY et Deborah L. CASWELL.
Software metrics : establishing a company-wide program. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.

[52] V. GRANVILLE, M. KRIVÁNEK et J. P. RASSON.
Simulated annealing : A proof of convergence.
IEEE Trans. Pattern Anal. Mach. Intell., 16(6) :652–656, 1994.
[53] Anna GRIMÁN, María A. PÉREZ, Luis Eduardo MENDOZA et Francisca LOSAVIO.
Feature analysis for architectural evaluation methods. *Journal of Systems and Software, 79(6) :871–888, 2006.*

[54] Anna GRIMÁN, María A. PÉREZ, Luis Eduardo MENDOZA et Edumilis Maria MÉNDEZ.
A method proposal for architectural reliability evaluation.
[55]In Object Jorge Management CARDOSO, GROUP José CORDEIRO et Joaquim FILIPE, réds., ICEIS
Corba components specification, version 3.0, omg tc document formal/2002-06-65.
Rapport(1),pagestechnique,564Object–568,Management2007. GROUP, 2002.

[56] Lars GRUNSKE. Early quality prediction of component-based systems - a ge neric framework.

Journal of Systems and Software, 80(5) :678–686, 2007.

[57]
Leila Chikhi, D partement d'informatique et de r echerch opérationnelle, Faculté des arts et des sciences "estimation d'impact de changement dans les systems à objets"
.Mémoire

[5]
F. Kemmere, in Proceedings of the Software

[59] Mark HARMAN, Robert M. HIERONS et Mark PROCTOR.
A new representation and crossover operator for search-based optimization of
Software odularization.
In GECCO '02 : Proceedings of the Genetic and Evolutionary Computation
[60] Mark HARMAN, Stephen SWIFT Kiarash MAHDAVI.
Conference, pages
An empirical study of the robustness of two module clustering fitness
functions.1351–1358, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.
In GECCO '05 : Proceedings of the 2005 conference on Genetic and
[61] D. R. HARRIS, H. B. REUBENSTEIN et A. S. YEH.
Reverseevolutionaryengineeringcomputation,tothearchitectural level.
In pages Proc. of 1029 ICSE, pages 1036, 186 New –195. ACM, York, Inc NY, .1995 USA, . 2005. ACM.
[62] G.T. HEINEMANN et W.T. COUNCILL. Component-based
software engineering. Addison-Wesley, 2001.

[63] Sallie M. HENRY et Dennis G. KAFURA.
Software structure metrics based on information flow.

IEEE Trans. Software Eng., 7(5) :510–518, 1981.

[64] B. Roy, « The optimisation formulation: criticism and overstepping », problem
The
Jounal of the Operational Research Society, 1982.
[65] A.Schärlig, D cider sur plusieurs crit res, panorama de l'aide à la d cision multicrit re
, Collection Diriger l'entreprise, Presses Polytechniques et Universitaires Romandes, Lausanne, Suisse, 1985.

[66] L.Y Maystre, J. Pictet, et J. Simos «Méthodes multicritères Electre », Presses Polytechniques et Universitaires Romandes, Lausanne, Suisse, 1994.

[67] P.Vincke, « L'aide multicritère à la décision », Éditions de l'Université de Bruxelles,

[68] S.Ben Mena, « Introduction aux méthodes multicritères d'aide à la décision », Biotechnol Agron.Soc.Environ, 2000.

[69] R. KAZMAN, L. O'BRIEN et C. VERHOEF.Architecture reconstruction guidelines.

Rapport technique, Software engineering institute, Carnegie-Mellon university, 2001.

[70]W.Jaziri «élisation et gestion des d'optimisation sur -contraint: Application à l'aide à la décision pour la gestion du risque de ruissellement », Thèse de l'Institut National.

[71] Rick KAZMAN, Gregory D. ABOWD, Leonard J. BASS et Paul C. CLEMENTS.
Scenario-based analysis of software architecture.
IEEE Software, 13(6) :47–55, 1996.

[72] Rick KAZMAN et S. Jeromy CARRIÈRE.
Playing detective : Reconstructing software architecture from available evidence.
Automated Software Engg., 6(2) :107–138, 1999.
[73] S. KIRKPATRICK, C. D. GELATT et M. P. VECCHI.
Optimization by simulated annealing.
Science, 220 :671–680, 1983.

[74] Tilley, Scott, "A Reverse Engineering Environment Framework"TECHNICAL REPORT (CMU/SEI-98-TR-005). Pittsburgh,

PA: SoftwareEngineering Institute, Carnegie Mellon University, 1998.

[75] Laurant AUDIBERT "UML2, de l'apprentissage à la pratique " Edition Ellipses 2009.

[76] Pierre PARREND "Gestion de projet informatique" Sciences U Lyon 2005.

[77] Glass, R. L. and Noiseux, Ronald A., Software maintenance Guidebook, Prentice-Hall, Inc., 1981.

[78] B. Lientz, E. Swanson, et G. Tompkins. "Characteristics of application software". Communications of ACM, 21(6):466-471, Juin 1978.

[79] Computer Society Press, Standards Collection – Software Engineering, The Institute of Electrical and Electronics Engineers, Inc., 1993.

[80] Carma McClure, The Three R's of Software Automation : Re-ngineering, Repository, and Reusability, Prentice Hall, 1992.

[81] Martin, Roger J. et Osborne, Wilma M., "Guidance on Software Maintenance", Computer Science and technology, . U.S. Department of Commerce National Bureau of Standards, NBS Special Publication 500-106, 1985.

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

[82] Tilley, Scott, "A Reverse Engineering Environment Framework"TECHNICAL REPORT (CMU/SEI-98-TR-005). Pittsburgh, PA: SoftwareEngineering Institute, Carnegie Mellon University, 1998.

[83] S. R. Chidamber and C. "A Metrics Suite for Object Oriented Design" in IEEE Transactions on Software Engineering, Vol. 20, No. 6,pages 476-493, June 1994.

[84] Chikofsky Elliot J. and Cross, James H, Reverse Engineering and Design Recovery : A Taxonomy. IEEE Software, Vol. 7, No.1, Pages13-17, 1990.

[85] Shari Lawrence Pfleeger and Shawn A. Bohner, "A Framework for Software Maintenance Metrics," in IEEE Transactions on Software Engineering, May 1990, pp. 320 -327.

[86] Michelle L. Lee, “Change Impact Analysis of Object Oriented Software”,these de Doctorat, George Mason University, Virginia, USA, 1998.

[87] H.D. Rombach, “Design Measurement: Some Lessons Learned”, in proceedings of IEEE Software, Vol. 7, No. 2, pages 17-25, 1990.

[88] M. A. Chaumun, H. Kabaili, R. K. Keller and F. Lustman, “A Change Impact Model for Changeability Assessment in Object-Oriented Software Systems”,

[89] Mustapha Kamel ABDI, Analyse et Pr diction d’impact de changementdans syst me à
objets, Th se de doctorat d’ tat en Informatique, Universit Es -S nia d’Oran, Avril 2007.

1.Notions de Base

L'objectif de cette annexe est de présenter le contexte de l'analyse d'impact et les concepts fondamentaux.

1.1Concepts de l'approche orientée objet

L'application de la technologie orientée objet produit des systèmes logiciels bien structurés, avec des architectures compréhensibles qui sont faciles à tester, à maintenir et à étendre [64]. Les concepts véhiculés par l'approche orientée objet sont importants à considérer dans une problématique comme la notre. Un système orienté objet est composé d'*objets* et de *classes.* Un objet est caractérisé par un état et un comportement. L'approche orientée objet est basée sur plusieurs concepts dont l'encapsulation, l'héritage et le polymorphisme. L'encapsulation est une manière de séparer l'implémentation d'une classe de sa spécification. C'est le fait de cacher ou masquer tous les détails d'un objet qui ne contribuent pas essentiellement aux caractéristiques de l'objet [15]. L'utilisateur connaît ce que l'objet fait (son comportement) mais pas comment il est fait (son implémentation).

1.2Notion de dépendance

Une dépendance dans un système logiciel est une relation directe entre les entités dans le système X→ Y tel que le programmeur qui modifie X doit être concerné par les effets de bord (side effects) possibles dans Y [70].

Wilde et Huitt [70] ont classifié les dépendances en : (i) dépendances de données entre deux variables, (ii) dépendances d'appels entre deux modules, (iii) dépendances fonctionnelles entre un module et les variables qu'il calcule, et (iv) dépendances de définition entre une variable et son type. Loyall [53] répartit les dépendances en deux types : (i) dépendance inter-procédures, à titre d'exemple, une procédure Ma est dépendante d'une autre Mb, si et seulement si au moins une instruction (statement) de Ma dépend d'une autre instruction de Mb, et (ii) dépendances intra-procédure en tenant compte des dépendances entre les différentes instructions à l'intérieur d'une procédure. Lee, dans sa thèse, se base sur la classification des dépendances en orienté objet de Wilde et Huitt et présente cinq catégories de dépendances : (i) dépendances classe à classe, (ii) dépendances classe à méthode, (iii) dépendances classe à variable, (iv) dépendances méthode à variable, et (v) dépendances méthode à méthode [48].

1.2.1Relations directes entre classes

Une relation directe entre deux classes se manifeste par l'existence d'au moins une des relations de dépendances suivantes : l'agrégation, l'utilisation ou l'héritage.

Agrégation : Représente la relation de tout/partie. Selon Li et Offut [50], c'est une relation de contenance. À titre d'exemple, on dit qu'une voiture a des portes ou un avion possède des ailes. On dit que la classe Class_A est une agrégation de la classe Class_B si :

Unattribut ou plus de Class_A est une instance de Class_B.

Untableau d'instances de Class_B est un attribut de Class_A.

Unattribut de Class_A est une référence vers une instance de Class_B.

Unattribut de Class_A est un tableau de références vers des instances de Class_B.

Les deux premiers types correspondent à l'agrégation automatique et les deux derniers correspondent à l'agrégation dynamique [47].

Utilisation : On dit que Class_A utilise Class_B, si Class_A envoie des messages à Class_B. Class_A a besoin de fonctionnalités offertes par Class_B, appelée client de Class_B, et Class_B offre ses services à Class_A, appelée serveur de Class_A. Ci-dessous des exemples d'utilisation :

Class_A invoque une méthode de Class_B.

Le type de retour d'une méthode de Class_A est une instance de Class_B.

Une instance de Class_B est passée en paramètre dans une méthode de Class_A. Une variable de type Class_B est modifiée ou utilisée par une méthode de Class_A.

Un attribut de Class_B est modifié ou utilisé dans une méthode de Class_A.

Class_A invoque une méthode de Class_B. Le type de retour d'une méthode de Class_A est une instance de Class_B. Une instance de Class_B est passée en paramètre dans une méthode de Class_A. Une variable de type Class_B est modifiée ou utilisée par une méthode de Class_A. Un attribut de Class_B est modifié ou utilisé dans une méthode de Class_A.

Héritage : Le concept d'héritage correspond à la création de nouvelles classes d'objets en se basant sur les classes existantes soit par réutilisation ou extension des fonctionnalités de ces dernières. Lee et Barby présentent trois schémas d'héritage dépendant des caractéristiques de la classe dérivée par rapport à la classe parente [9, 48].

- Strict inheritance est le schéma le plus simple de l'héritage. La classe drivée garde le comportement exact hérité de sa classe parente. Les propriétés héritées ne peuvent être modifiées (overriden), et la classe dérivée peut être raffinée seulement par l'ajout de nouvelles propriétés.
- Subtyping est le plus courant type d'héritage. En plus des propriétés de l'héritage strict, il permet la redéfinition des propriétés héritées quand les opérations de la classe parente ne sont pas appropriées pour la sous-classe (donner une nouvelle implémentation à une opération héritée pour répondre aux besoins propres de la sous-classe).
- Lorsque la classe dérivée n'est pas une spécialisation de la classe de base et représente une abstraction complètement nouvelle (qui base une partie de son comportement sur une partie d'une autre classe), ce type d'héritage est appelé Subclassing.

1.2.2Relations indirectes

Rapport-Gratuit.com

Une classe peut dépendre indirectement d'une autre classe via une séquence successive de dépendances intermédiaires. Une classe Class_A a une relation indirecte avec une classe Class_B notée par Class_A **R**+ Class_B, s'il existe une chaîne :

Class_B1, Class_B2, Class_B3…… Class_Bn, avec n >=2,

tel que **Class**_A**R** Class_B1, Class_B1 R Class_B2…… Class_Bn **R Class_B**.

1.2.4Graphe de relation

Un graphe de relation permet de capturer toutes les relations de dépendances entre les différentes classes d'une application orientée objet afin de mieux comprendre les interactions et la structure de cette application. Ce graphe est utilisé par différents auteurs dans leurs travaux de recherche avec des appellations différentes. Pour Kung et al. Il s'agit "d'Object Relation Diagram" [47] et pour Lee, c'est "Object Oriented System Dependency Graph" [48].

En général, un graphe de relation est un graphe orienté G= (N, D) où N est l'ensemble des nœuds représentant les classes du programme N = {Class_C1, Class_C2, Class_C3, …Class_Cn}, et D l'ensemble des relations de dépendances entre ces classes D= {U, G, H}. Comme cité précédemment, ces relations consistent en l'héritage (H), l'utilisation (U) et l'agrégation (G). La figure ci-dessous est un exemple de ce flux de relations de dépendances dans une application orientée objet.

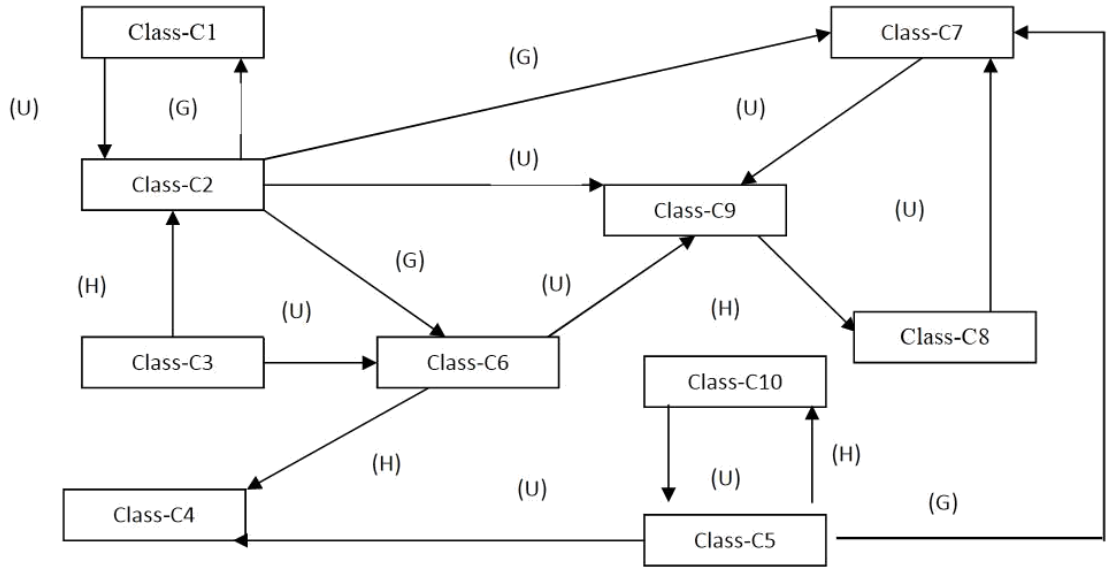


Figure 1. Graphe de relations entre les classes

1.3Notion d'impact

Un impact représente l'effet ou la conséquence d'une chose sur une autre. Arnold et Bohner considèrent l'impact comme : "a part determined to be affected, and therefore worthy of inspection"[14]. Dans les systèmes orientés objets, l'unité de base est la classe et non la fonction comme dans le paradigme procédural. On dit qu'une classe Class_A a un impact sur une autre Class_B, si un changement dans Class_A altère directement ou indirectement le comportement de Class_B.

La notion d'impact est étroitement liée à la notion de dépendances. Ainsi, si Class_B dépend de Class_A directement ou indirectement, alors un changement dans Class_A impacte Class_B. Si une classe est dépendante de beaucoup d'autres classes, alors les changements dans les autres classes peuvent impacter la classe dépendante. Ce qui rend sa réutilisation difficile, et sa séparation de son environnement ne peut se faire sans altérations supplémentaires.

régression.

1.4.Effet de propagation "Ripple effect"

Bohner définit l'effet de propagation par : " effect caused by making a small change to a system which impacts many other parts of a system"[14]. L'effet de propagation d'un changement apporté au code source d'un système logiciel est défini par Turver comme "the consequential effects on other parts of the system resulting from that change. These effects can be classified into a number of categories such as logical effects, performance effects, or understanding effects" [67].

L'effet de propagation peut se présenter pendant les phases initiales du cycle de vie d'un logiciel orienté objet (conception et spécification). Par exemple, un changement dans les spécifications du système impacte les autres phases du cycle de vie du logiciel dont le développement, le test et la maintenance. Cette propagation se manifeste dans les plannings, les délais de livraisons, le budget, les ressources humaines, etc.

Par ailleurs, un changement dans la logique de programmation d'une classe peut affecter les fonctions, les performances ou le comportement des autres classes auxquelles elle est liée. Michelle Lee propose deux types d'effet de propagation [48] :

-Effet de propagation direct (ripple effet direct) : se présente lorsque le changement d'une variable impacte directement la définition d'une autre variable.

-Effet de propagation indirect (ripple effect indirect) : se produit quand la variable affectée à son tour impacte d'autres variables.

La stabilité du logiciel réside dans sa résistance aux conséquences des changements. Selon Turver, l'analyse de la stabilité diffère de l'analyse d'impact parce qu'elle considère la somme des effets de propagations possibles plutôt qu'un effet de propagation particulier causé par un changement [67].

Rapport-gratuit.com
LE NUMERO 1 MONDIAL DU MÉMOIRES

1.5.Firewall

Effet de propagation direct (ripple effet direct) : se présente lorsque le changement d'une variable impacte directement la définition d'une autre variable. Effet de propagation indirect (ripple effect indirect) : se produit quand la variable affectée à son tour impacte d'autres variables.

La stabilité du logiciel réside dans sa résistance aux conséquences des changements. Selon Turver, l'analyse de la stabilité diffère de l'analyse d'impact parce qu'elle considère la somme des effets de propagations possibles plutôt qu'un effet de propagation particulier causé par un changement [67].

Le concept de Firewall dans le contexte des applications orientées objets a été initié par Kung et al. [46]. Le firewall de la classe Class_C noté par CFW (Class_C) correspond à l'ensemble des classes qui peuvent être affectées par un changement dans Class_C. Autrement dit, toutes les classes sur lesquelles Class_C peut avoir un impact. Le firewall représente la portée maximale d'un changement. Il contient toutes les classes possiblement mais pas nécessairement affectées par le changement. En vue de s'assurer de la validité des modifications apportées au système, il faut retester toutes les classes se trouvant dans le firewall.

1.5Conclusion

Dans cette annexe, nous avons présenté des concepts de base utilisés lors de l'analyse d'impact du changement. Puisqu'on s'intéresse aux applications développées avec le paradigme orienté objet, quelques notions de base ont été citées, dont la classe qui représente l'unité de base.

Un changement peut se propager et atteindre les autres parties de l'application; il s'agit de l'effet de propagation (ripple effect). Modifier une classe peut avoir des effets sur les classes qui lui sont reliées (i) directement à travers les relations directes ou (ii) indirectement via la succession de relations directes entre les paires de classes.

L'analyse d'impact du changement, permet d'estimer les conséquences du changement sur le reste du système et de fournir aux gestionnaires les informations utiles pour planifier et implémenter le changement.

Résumé

Les systèmes logiciels sont devenus très importants dans notre vie quotidienne. Ils contrôlent les machines de payement, les instruments médicales, les systèmes de télécommunication et plusieurs d'autres systèmes. A cause de leur intérêt et leur vaste domaine d'application, tout comportement non désiré ou erreur de fonctionnement peut engendrer des pertes d'argent, arrêts de services et des risques de vies. La fiabilité de logiciels devient donc un facteur suprême. D'autre part, l'évolution de besoins et le changement continu de contextes de fonctionnement nécessitent la mise à jour et la maintenance courante de ces systèmes.

Dans le cycle de vie de développement des applications logicielles, l'étape de maintenance est l'étape la plus critique, et elle consomme environ 90% de l'ensemble des ressources utilisées [5]. La tache principale de la maintenance est de comprendre l'application en analysant sa structure et l'organisation de ses composants afin de fixer les problèmes et/ou d'ajouter des nouvelles caractéristiques [5].

Mots –clés : Analyse d'impact de changement, maintenance, rétro-ingénierie, matrice des liens.

Abstract

Software systems have become very important in our daily lives. They Control payment machines, medical instruments, telecommunication systems and several other systems. Because of their interest and Scope, any unwanted behavior or operational Resulting in loss of money, service stoppages and life-threatening risks. The reliability of Software becomes a supreme factor. On the other hand, the evolution of needs and Continuous change in operating contexts require updating and Maintenance of these systems.

In the development lifecycle of software applications, the Maintenance is the most critical step, and it consumes about 90% of all Resources used [5]. The main task of maintenance is to understand By analyzing its structure and the organization of its components in order to

Keywords: Impact analysis of change, maintenance, reverse engineering, link matrix.

ملخص

أصبحت أنظمة البرمجيات مهمة جدا في حياتنا اليومية بأنهاتسيطر على الآلات، الأدوات الطبية، أنظمة الاتصالات السلكية واللاسلكية والعديد من النظم الأخرى بسبب اهتمامها ونطاقها، أي سلوك غير مرغوب فيه
مما يؤدي إلى فقدان المال، ووقف الخدمات والمخاطر التي تهدد الحياة من ناحية أخرى، تطور الاحتياجات و يصبح البرنامج عاملا أعلى يتطلب التغيير المستمر في سياقات التشغيل تحديث وصيانة هذه الأنظمة في دورة حياة تطوير تطبيقات البرمجيات، الصيانة هي الخطوة الأكثر أهمية.
المهمة الرئيسية للصيانة هي الفهم من خلال تحليل هيكلها وتنظيم مكوناتها .[ويستهلك حوالي 90٪ من جميع الموارد المستخدمة]

الكلمات الرئيسية: تحليل تأثير التغيير، الصيانة، الهندسة العكسية، مصفوفة الارتباط

Chapitre I :

Maintenance des logiciels

Chapitre II:

Rétro-ingénieurie des logiciels

Chapitre III :

Impact de changemment

Chapitre IV :

Conception et réalisation