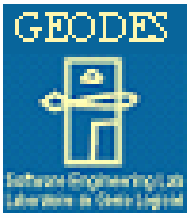
Rapport IFT3051
Été 2005

Qualité des programmes et patrons de conception

Auteurs :
Sébastien Boisclair
Vinh Thinh Le
Joseph Vong

Responsables :
Yann-Gaël Guéhéneuc
Stéfan Monnier

Superviseur :
Duke Huynh D.L.



lundi, 1er août 2005

Plan

Analyse des logiciels

Détection de patrons de conception et Méthodologie

Résultats et évolution

Conclusion .....page 34

Appréciation personnelle.....page 35

Bibliographie, Références .....page 38

Annexe A .....page 39

Contexte

Qu'est-ce qui différencie un bon logiciel d'un mauvais ? Les réponses à cette question sont multiples. Ceci peut varier de la portabilité, à la robustesse en passant par la maintenance, la fiabilité, la stabilité, l'évolutivité. Tous ces critères de jugement reviennent à un seul mot : la qualité!

Ceci nous amène à une autre question : Pouvons-nous mesurer la qualité ? La réponse à cette question n'est malheureusement pas définie. En effet, il n'existe pas de métrique ou de règle indiscutable pouvant réaliser cette tâche à ce jour. Il est donc difficile pour les programmeurs d'écrire un code tout en affirmant que celui-ci serait d'une bonne qualité. Ces raisons ont motivé la nécessité de développer des méthodologies et des concepts pour concevoir un bon code : les patrons de conception ont alors fait leur apparition pour répondre à ce besoin. Ces petites 'merveilles' définissent une structure normalisée à appliquer lors d'un problème typique rencontré par la plupart des programmeurs.

Cependant, il s'avère que l'abus d'utilisation des patrons de conception est également néfaste à la qualité du code. La qualité du logiciel est donc grandement affectée, mais n'est pas proportionnelle à l'utilisation de ceux-ci.

La présence de patrons de conception est une bonne indication sur la qualité d'un logiciel. Cependant, la recherche manuelle de ceux-ci dans un programme est une tâche relativement fastidieuse et ardue.

Des outils « magiques » de rétro conception ont donc été développés pour remédier à cela. Est-ce une réussite?

Objectifs

Le travail consiste en un premier temps à trouver un outil pouvant nous aider à l'analyse de grands programmes. Une comparaison des différents outils de rétro conceptions disponibles est donc nécessaire afin de déterminer le plus efficace des outils de rétro conception parmi une liste déterminée par le superviseur du projet. Pour vérifier l'exactitude de ces comparaisons, il est impératif de se baser sur la rétro conception d'une application. Dr Java a été désigné comme application à être analysée. Nous considérons que ce programme est grand et bien écrit ce qui suppose qu'il contient une bonne quantité de patrons de conception.

La recherche de l'évolution des patrons de conception, pour constater les changements au fil du temps, sur différentes versions de Dr Java, serait un atout à la compréhension des implémentations de patrons. Cette dernière tâche est complémentaire à l'analyse du programme.

Les objectifs de ce projet sont les suivants :

- Comparer des outils de rétro conception;
- Analyser différentes versions de Dr Java et identifier les patrons de conception utilisés lors de la conception et l'implantation du programme;
- Corréler l'introduction, la suppression ou la modification de patrons de conception avec les demandes de changements.

Les résultats de la comparaison des outils de rétro conception ont été mis sous forme de grilles d'évaluation qui se trouvent en annexe A. Les résultats de la recherche des patrons de conception sont présentés sous format de microarchitectures décrites en XML.

Cheminement global du projet

Lecture du livre de référence

Design Pattern, le livre de référence, a été utilisé pour se familiariser aux patrons de conception et leurs différentes implémentations possibles dans un logiciel. Étant une référence, le livre a été consulté tout au long de la recherche de patrons.

Les 3 programmes : InfoGlue, Junit et Dr Java ont été évalués afin de sélectionner celui utilisé pour l'évaluation des différents outils :

L'analyse d'InfoGlue

InfoGlue, une application choisie aléatoirement pour classer et départager tous les logiciels. Ce programme a été utilisé comme baromètre pour évaluer la capacité et le potentiel de réingénierie des différents outils. Seize logiciels ont été testés en environ deux semaines et ceux possédant le plus de potentiels ont été sélectionnés pour une évaluation plus poussée. En effet, il était plus simple de vérifier la pertinence des diagrammes de classes à l'aide d'un programme contenant un nombre de classes restreints. Ainsi, il est possible de distinguer l'omission de certaines classes, par exemple. Les outils suivants furent éliminés à cette étape : MagicDraw, MetaMill, ObjectIF et VisualUML, Poséidon 3.0 et 3.1, Objectering UML modeler et PowerDesigner

L'analyse de JUnit

L'analyse de JUnit a été similaire à l'étape d'InfoGlue. En effet, la distinction notable, entre ces deux analyses, est que nous nous sommes beaucoup plus penchés sur la détection de patrons de conception et des possibilités que pouvaient offrir les logiciels restants. InfoGlue contenait qu'une vingtaine de classes alors, il était peu probable qu'il contienne des patrons de conception.

L'analyse de JUnit a durée environ deux semaines. Il faut noter, qu'à ce point de l'évaluation des outils, une consultation plus poussée fût nécessaire, car certains logiciels pouvant contenir des fonctionnalités uniques et non évidentes à première vue. Une grille d'évaluation, voir l'annexe, regroupant les principaux critères, a été définie afin de comparer les différents outils. Deux niveaux de critères ont été élaborés : les critères primordiaux et les critères secondaires. Les critères primordiaux regroupent ce qui est ré-ingénierie et les critères secondaires regroupent plutôt ce qui est technique comme la documentation ou les interfaces usagers. Il faut également noter que plus un outil obtient de *oui*, plus celui-ci est efficace. À cette étape, les outils suivants ont été éliminés : Fubaja , Entreprise Architecte, Ess Model et Ideogramic uml.

L'analyse de Dr Java :

L'analyse de Dr Java a été effectuée à l'aide des outils suivants : Describe, Omondo, Code Logic et Eclipse. Afin de vérifier la véracité du potentiel détecté, une analyse poussée a été effectuée. Deux versions ont été analysées en détail selon des méthodologies systématiques afin d'y révéler des patrons de conception. Cette étape a durée un mois et demi et le seul outil éliminé à ce point est Structural Analysis IBM.

La rédaction des microarchitectures en format XML a été effectuée lors de l'analyse de Dr Java. Les résultats de l'analyse d'InfoGlue et de JUnit n'ont pas été définis sous ce format, car cela n'aurait pas été pertinent.

Corrélation des versions de Dr Java

Cette tâche a été effectuée durant les deux dernières semaines. Une comparaison des patrons de conception des plus anciennes versions avec la version la plus récente a été réalisée en observant l'évolution de ces patrons au fil du temps. Quatre modifications ont été relevées.

Voici un tableau récapitulatif comprenant le descriptif des tâches, le temps consacré et les résultats obtenus :

Description des tâches	Temps (en semaines)	Résultats
Lecture du livre de référence	1	Meilleure compréhension des intentions et des implémentations possibles des patrons
Analyse d'InfoGlue	2	Élimination des outils
Analyse de JUnit	2	Élimination des outils
Analyse complète de 2 versions de Dr Java	6	Détection à l'aide des outils d'une vingtaine de microarchitectures
Évolution des versions de Dr Java	2	Quatre patrons ont évolués au cours des versions

Tableau 1 - résumer de la répartition du temps au long du projet.

Description des outils

Entreprise 6.1.7

Describe est l'outil, parmi tous ceux qui ont été testés, le plus complet et celui offrant le plus de potentiel. En effet, en observant la grille d'évaluation de Describe en annexe, il est clair que cet outil a un énorme potentiel, puisque la quantité de « Oui » coché est en majorité écrasante par rapport au nombre de « Non ». Faut-il rappeler que cet élément est un indicateur du potentiel de l'outil ?

Pour ce qui est de la réingénierie en tant que telle, Describe génère une série de métas modèles à partir des classes de Dr Java. Il suffit de sélectionner le nombre de classes désirés, venant du ou des packages voulus pour générer le diagramme de notre choix. Donc, en théorie, Describe pourrait rétro concevoir n'importe quel diagramme. Il faut préciser que seuls les diagrammes de classe et de séquence ont été testés, car les autres n'avaient pas d'intérêt pour le projet. Précisons aussi que le diagramme de séquence n'affiche que les acteurs et les entités avec leurs lignes de vie, il omet les interactions.

Côté réingénierie du diagramme de classe, celui-ci est assez précis et rétro conçoit les agrégations comme illustrée ci-dessous. Il est intéressant de noter que Describe est le seul outil offrant cette fonctionnalité. Il est même possible d'y voir directement des patrons de conception grâce à la précision du digramme (*state* dans ce cas-ci).

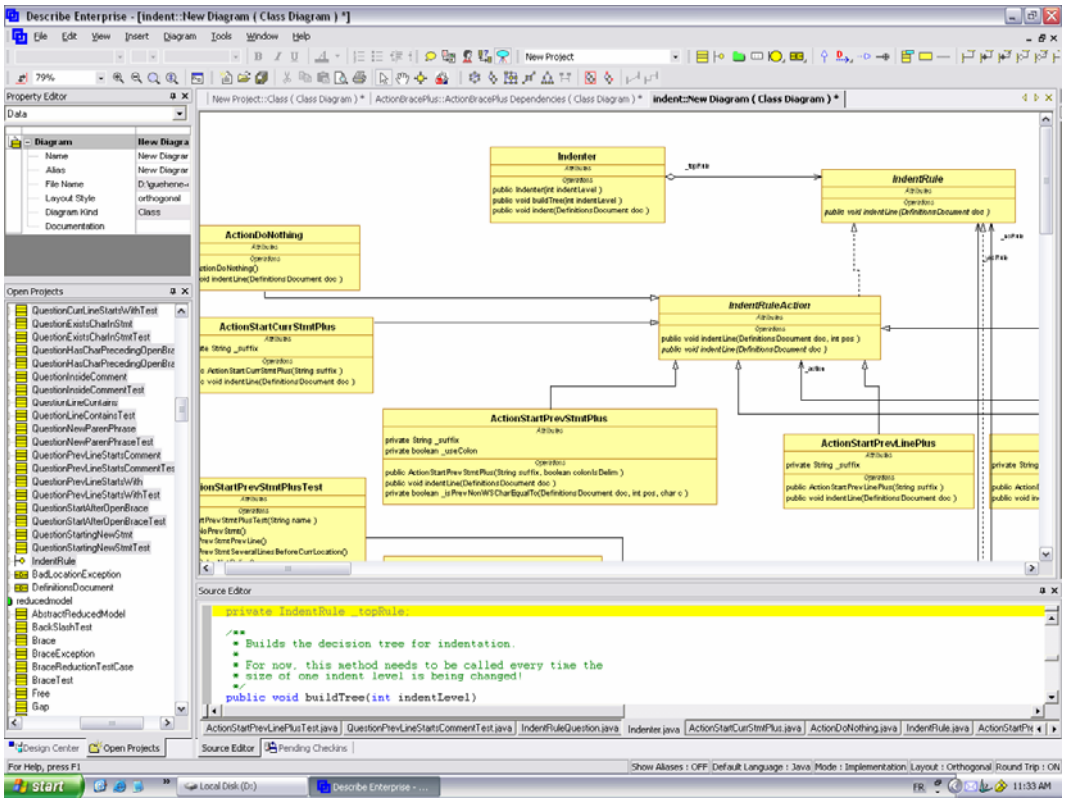


Figure 1- Représentation d'une architecture : State

Pour ce qui est de la navigation de Describe, elle est très conviviale et offre des fonctionnalités uniques et plaisantes pour l'usager. En effet, l'interactivité de la navigation s'active lorsqu'une association est sélectionnée. Celle-ci devient bleue et une bulle apparaît pour indiquer son type. Une fenêtre de propriété permet de connaître les moindres détails comme la classe qui est à l'extrémité de l'association.

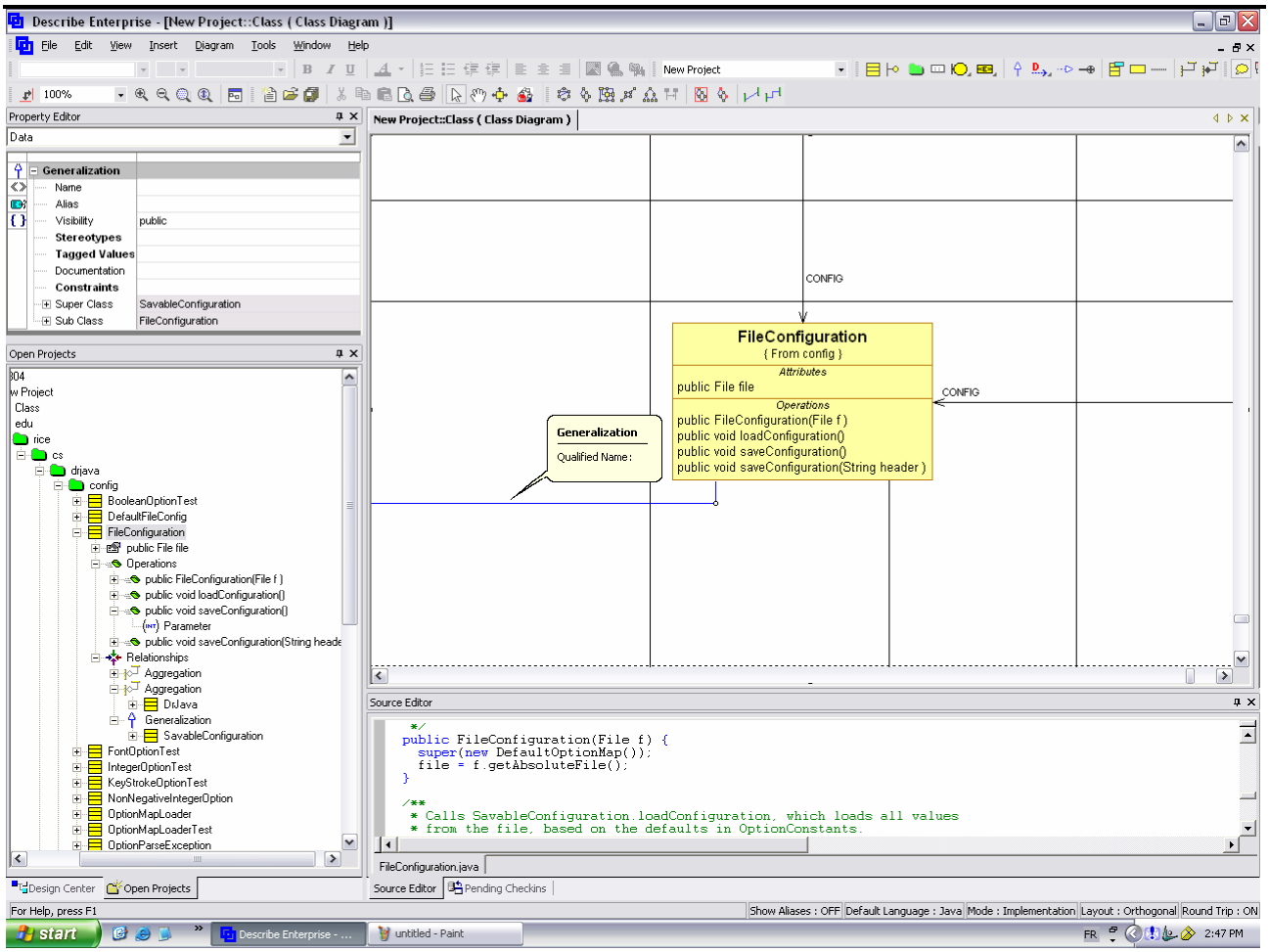


Figure 2 - Navigation de Describe

Un éditeur de texte est également présent et permet de naviguer dans le code source, mais celui-ci ne présente aucune fonctionnalité. Par contre, il existe une synchronisation automatique avec les diagrammes. Une modification du code source a pour effet une mise à jour immédiate du

De plus, il est possible d'effectuer des relations inter packages en sélectionnant les métas modèles des différents packages c'est-à-dire, sélectionner deux classes d'un package et trois autres d'un autre par exemple. La possibilité d'avoir toutes les classes sur un seul et même diagramme permet une plus grande latitude et liberté. Tous ces petits points peuvent sembler être que des petites fonctionnalités anodines, mais simplifient la tâche pour rechercher la présence des patrons de conception.

Java 1.5 n'est pas tout à fait compatible avec la rétro conception de Describe. En effet, Describe indique néanmoins qu'il ne reconnaît pas cette classe en la déclarant datatype.

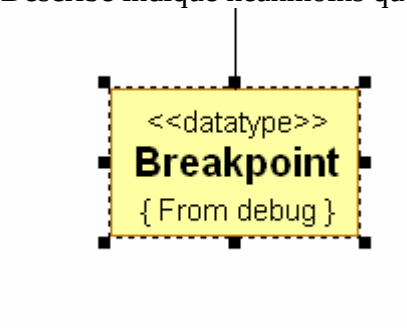


Figure 3 – datatype de Describe

Seul véritable inconvénient du diagramme de classe étant l'absence des classes internes. Néanmoins, il est possible de les identifier grâce à l'éditeur de texte. Describe est autonome en tant que tel et il n'a nul besoin d'un autre outil pour compléter ses lacunes.

Il faut dire qu'Eclipse comblait les lacunes de l'éditeur de texte de Describe. En effet, les fonctions *Call hierarchie* et *Type hierarchie* permettent de compléter à la main ce que l'outil aurait pu négliger. Rappelons que l'éditeur de texte de Describe n'a aucune fonctionnalité avancée. Cependant, Describe offre également une version *plug-in* avec Eclipse. Toutefois, cette version n'a pas été testée car lors de l'installation initiale des problèmes d'installation étaient survenus. A ce point du projet, ces fonctionnalités ne représentaient aucune valeur à notre esprit. Cette version mérite d'être davantage étudiée. En conclusion, Describe est supérieur à tous les programmes étudiés et il est un des seuls outils que nous aurions pu utiliser sans l'aide d'un autre outil pour combler ses faiblesses.

Omondo est un petit logiciel permettant de rétro concevoir avec justesse le code source qu'on lui donne. Ce programme est un plug-in à ecplise. Il est très facile d'utilisation et permet la synchronisation du code source avec le diagramme dans le cas où le code est sans erreur. Une des fonctionnalités intéressantes de ce logiciel est la génération du diagramme de package, comme nous pouvons voir un aperçu à la figure 4.

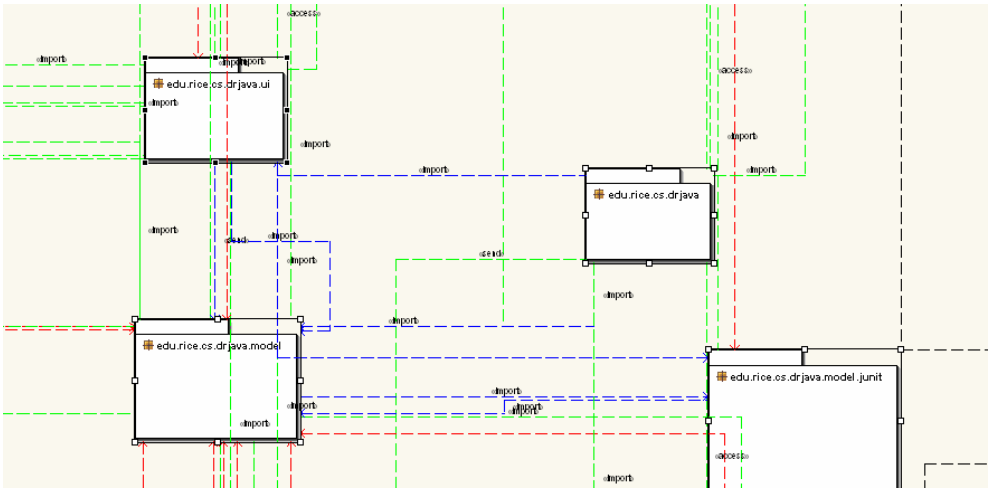


Figure 4 – Diagramme de package d'Omondo

Ce qui est le plus amusant sur ce diagramme, c'est que plus que la représentation du package est volumineuse, plus le nombre de classes à l'intérieur est imposant.

Regardons maintenant une vue typique d'Omondo. Il est très agréable de voir que les ingénieurs ont pensé d'illustrer de façon visuelle les types de méthodes et de classes, de façon esthétique et intuitive.

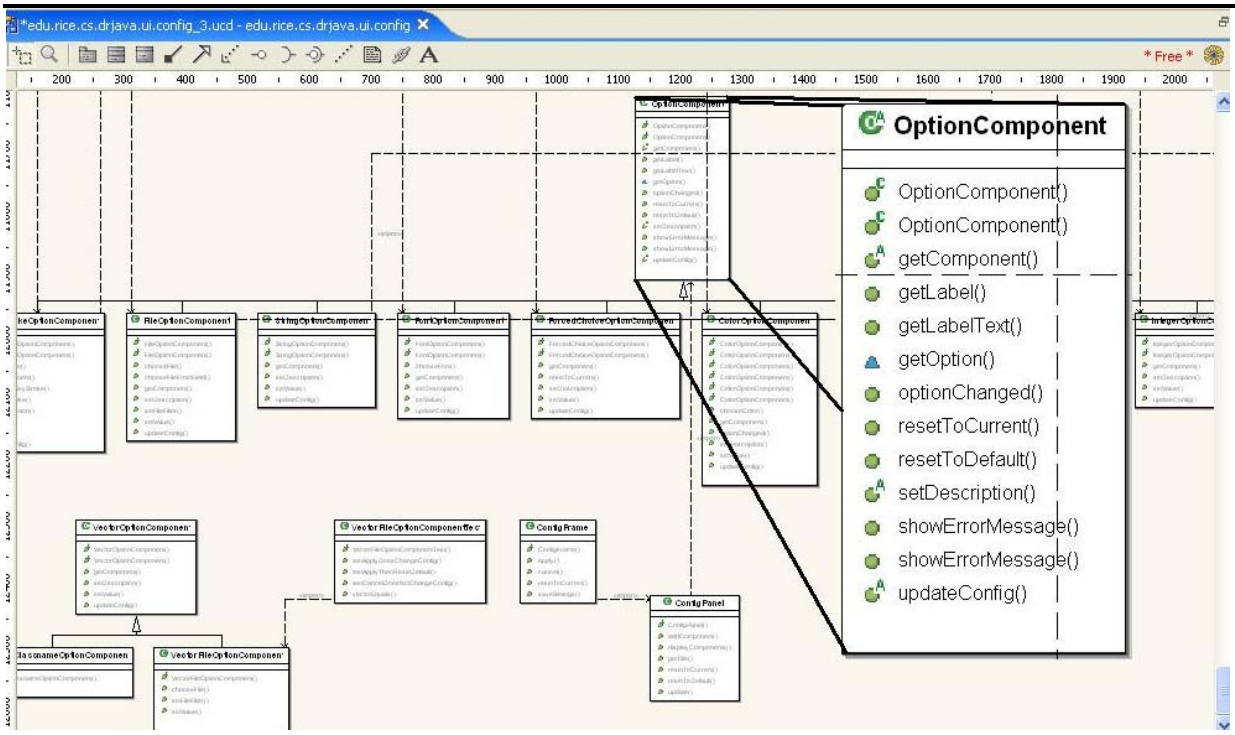


Figure 5 – Grossissement de la classe OptionComponent de Dr Java

Comme on peut le voir sur l'agrandissement de la classe *OptionComponent* de la figure 5, on comprend intuitivement que le type de la classe est *Abstract* grâce au petit dessin, pour les méthodes standards elles sont représentées par des points verts et les méthodes *template* de Java 1.5 sont représentées par de petits triangles bleus.

Plusieurs fonctionnalités ne sont pas présentes dans la version gratuite testée. La version complète est théoriquement 50 fois plus rapide que la version de base, dû au fait que la rétro conception ne se fait de la même manière dans les 2 cas. De plus, elle supporte la réingénierie du diagramme de séquence.

La version complète est gratuite pour les universités. Ainsi, nous recommandons fortement à l'université de faire la demande pour ce logiciel. Dans notre cas, nous n'avons pas fait la demande, car d'une part nous ne savions pas si le logiciel serait retenu et d'autre part parce que plusieurs informations relevant de l'administrateur réseau devaient être fournies. Il faut aussi dire que les fonctionnalités de bases nous ont satisfaites.

Quoi dire de plus, il est gratuit, rapide et esthétique. Le seul petit problème, c'est qu'il superpose des liens entre les classes et dessine que : les associations, les dépendances et les héritages. Même si tous les liens ne sont pas dessinés, la majorité du temps, cet outil aide grandement à la détection de patrons de conception.

Code Logic permet l'analyse des programmes sans erreur et dessine bien les diagrammes. En effet, les dessins sont représentés sous une forme standard et permettent donc l'analyse simple de celui-ci. De plus, il contient les options nécessaires à une bonne analyse.

Par contre, la rétro conception de programme ne dessine pas les liens d'agrégation dans les diagrammes de classes. Du même fait, nous devons utiliser Eclipse pour compléter la recherche de patrons.

En plus, ce programme contient un point fort qu'aucun autre programme ne contient ; il permet le dessin de beaux diagrammes de séquence. L'utilisation de cette fonctionnalité n'a malheureusement pas été testée du au fait qu'elle n'était pas requise au long du projet, mais cela n'empêche que ceci est un aspect non négligeable de cet outil.

Cependant, cet outil ne supporte pas Java 1.5. Il génère une fenêtre d'erreur à chaque fois qu'il ne reconnaît pas la syntaxe du code. (Donc de 50 à 100 fenêtres pour un programme comme DrJava). De plus, lorsqu'il rencontre une erreur de syntaxe, il ignore complètement la classe et du même coup, la rétro conception sur celle-ci ne ce fait pas.

Fubaja

Fubaja était un outil qui semblait avoir un potentiel énorme dû à sa capacité de détection des patrons de conception. D'ailleurs, il est le seul outil testé offrant cette fonction. Pour ce qui est de la rétro conception, il l'effectue que sur un package à la fois. Donc, la rétro conception ne se fait pas sur des répertoires de façon récursive.

De plus, seuls les héritages sont rétro conçus, les appels de fonction et les agrégations sont ignorés. La fonction de détection des associations ne détecte aucun lien supplémentaire. Il existe une fonction de détection des patrons de conception qui ne semble pas être à point. En effet, seuls les *Templates method* et les *Singletons* sont détectés. Les expériences ont pu montrer que la capacité de détection des patrons de conception de l'outil n'est pas assez poussée pour détecter les autres patrons plus complexes. Pour arriver à cette conclusion, des fichiers contenant certaines microarchitectures ont été fournies à l'outil et celui-ci ne détectait aucune d'entre elles.

En éliminant cette fonctionnalité, Fubaja devient un programme plus qu'ordinaire avec une documentation majoritairement allemande de surcroît.

MagicDraw

MagicDraw n'offre aucune fonctionnalité qui permet de le distinguer parmi les outils testés. La navigation et le programme en tant que tels sont assez standards. Ressemblant de beaucoup à Rational Rose avec des fonctionnalités moins développées, MagicDraw effectue une tâche similaire à Describe mais sans agrégation et en omettant certaines classes lors de la rétro conception. En effet, pour vérifier la qualité et la pertinence des diagrammes de classes, une comparaison a été effectuée entre les autres diagrammes de classes et le code source.

VisualUML 4.21

Ressemblant beaucoup à MagicDraw, VisualUML offre des fonctionnalités similaires. Par contre, il est encore moins précis et est esthétiquement beaucoup moins convivial.

ObjectIF 5.0

Le diagramme de classe omettait certaines classes, telles que *Test* dans *JUnit*, ne contenait aucune agrégation et n'était pas lisible. En effet, la disposition du diagramme est faite de façon très compacte, les relations se chevauchant les unes par rapport aux autres. Seul point positif, les cardinalités des relations sont affichées. De plus, l'outil requiert un temps démesuré pour générer un environnement de travail. Ce logiciel à beaucoup trop d'inconvénients par rapport à ses avantages.

Metamills 4.1

Éliminer Metamills fut un autre choix facile, car le diagramme de classe généré est illisible. Les relations empiètent sur des classes et d'autres relations de manière très peu esthétique et il n'y a aucun réalignement automatique possible. Esthétiquement, le diagramme de classe correspond à un rectangle très mince en largeur et ayant des dizaines de pages en longueur. De plus, les classes sont disposées en ordre alphabétique ce qui rend les relations encore plus chaotiques.

Poséidon 3.0 et 3.1 (ajout jdk 1.5)

À la lecture de la documentation, ce logiciel semblait être extraordinaire, mais ce ne fût pas le cas. Tout d'abord, il n'est pas capable d'analyser tout le code de Dr. Java même avec la version 3.1 qui supporte le Java 1.5. Il trébuche sur des erreurs de fin de lignes incomprises jusqu'à ce jour. De plus, les architectures générées sont complètement affreuses. Il met toutes les classes sous forme de lignes/colonnes et ajoute les liens parmi les classes et tout ceci devient illisible pour un programme de moyenne envergure relativement petit comme Dr Java. Il est impossible de voir un patron de conception de cette façon.

Entreprise Architecte

Prometteur, ce logiciel est rapide et facile d'utilisation. On peut traiter différents langages de programmation. Les classes sont relativement bien disposées, mais lorsque le nombre de classes devient trop important la représentation devient encore une fois illisible, car il superpose toutes les classes. Il est à noter que c'est l'un des seuls logiciels qui est capable de terminer la rétro conception sur les programmes d'environ 1000 classes. Il a donc été éliminé pour 2 raisons. Premièrement, il ne supporte pas le java 1.5. Deuxièmement, les diagrammes de la rétro conception doivent être retravaillés s'il y a beaucoup de classes. De plus, il n'apporte pas vraiment d'avantage par rapport à Omondo si l'on considère que les fonctionnalités de bases.

PowerDesigner

Comme *Entreprise Architecte,* ce logiciel est rapide, facile d'utilisation et positionne correctement les classes pour les héritages particulièrement. De plus, comme notre but était d'analyser le pouvoir de rétro conception des outils, les autres fonctionnalités ont été seulement survolées. Ainsi, encore une fois ce logiciel ne montre pas de nets avantages face à Omondo au niveau de la rétro conception.

Objectering UML modeler

L'installation de ce logiciel n'est pas chose facile. De plus, sa convivialité est très discutable. Il est très ardu de comprendre le fonctionnement de ce logiciel sans lire le manuel d'instructions. Des messages d'erreurs s'affichent, mais rien ne nous indique la provenance de la source. La rétro conception est partielle et l'ensemble du logiciel n'est pas convivial. Le logiciel a été éliminé dès le début du processus de sélection.

EssModel_binary_22

À première vue, ce programme ne semblait pas fiable du fait de sa très petite taille (moins de 1mb). Cependant, il ne faut pas se fier aux apparences. Comme on le dit si bien : « il fait ce qu'il faut comme il le faut ». Facile à utiliser, il dessine sans erreur toutes les classes et les héritages.
Par contre, nous avons décidé de l'éliminer pour une seule raison soit l'insuffisance de ressources. En effet, il ne permet pas de faire aucune modification sur les diagrammes, il ne dessine pas les agrégations et toute la rétro conception se fait sur la même fenêtre, ce qui devient lourd pour un programme de grande taille.

Ideogramic UML Desktop Edition

Ce programme est très bien. Il fait des diagrammes de rétro conception justes et est facile à manipuler. Cependant, suite à l'analyse de plusieurs programmes, nous avons découvert qu'il n'arrive pas toujours à réaliser la rétro conception de certains programmes du fait qu'il contient lui même des « bugs ». Il est impératif qu'un outil puisse faire la rétro conception de n'importe quelle application pour qu'il puisse être considéré. Il n'était donc pas difficile d'en juger de celui-ci.

IBM Structural Analysis

Ce grand programme est sans aucun doute un de ceux que nous avons hésité d'éliminer. Il permet la rétro conception d'un programme avec une précision rare. Il comprend toutes les options, les composantes et les statistiques nécessaires à une bonne analyse.
La raison pour laquelle nous avons décidé de l'éliminer est qu'il ne représente pas ses diagrammes et ses liens de la façon à laquelle nous sommes habitués (format UML). Ils sont représentés sous une forme totalement différente ce qui ne permet pas l'analyse facile du programme. Il est déjà une tâche difficile de trouver les patrons de conception si en plus il faut décortiquer la manière que l'outil schématise les architectures !

Conclusion de l'analyse des outils

Ces six outils peuvent être regroupés en 3 niveaux :
Excellent : Describe et Omondo
Passable : CodeLogic, Fubaja, MagicDraw, VisualUML et Structural Analysis IBM
À éviter : ObjectIF, MetalMills, Ideogramic uml et ESS Model

Les Méthodologies

Il est important de dire que la recherche de patron de conception ne se fait pas simplement par intuition. Déjà l'équipe précédente de ift3051 en génie logiciel, supervisé par Yann-Gaël Guéhéneuc, avait exploité plusieurs méthodes. Il nous fallait donc innover sur les méthodologies à utiliser. Il faut dire que la méthodologie opportuniste n'est rien de plus qu'un regroupement des quatre premières méthodes selon l'opportunité. Voici donc les méthodes de recherche qui ont été exploitées et utilisées tout au long de notre projet :

Les méthodologies par vue globale, par héritage, opportuniste ainsi que par relations des patrons ont été nommées par notre équipe de travail tandis que la méthodologie naïve est une méthodologie existante dans le domaine du génie logiciel.

Recherche par vue globale

Outil: logiciel de rétroconception ayant la possibilité de générer un diagramme de classe précis.

Étapes :

1. Générer par l'outil de rétroconception le diagramme de classes
2. Naviguer dans le code en espérant trouver des microarchitectures similaires à un patron de conception

Description détaillée :

Tout d'abord, à l'aide d'un outil qui est capable de rétro concevoir un programme, on génère le diagramme de classes.

Ensuite, il faut manipuler le diagramme afin de retrouver une microarchitecture conforme aux patrons de conception normalisés. Il faut normalement, repositionner les classes sous forme standardisée. Ceci veut dire que les héritages sont positionnés verticalement et les agrégations de façon horizontale. De cette manière, il est bien plus évident de lire le diagramme de classes et cela accélère la détection des patrons de conception. Commencer par regarder les classes abstraites ou interfaces peut aider à tout moment dans la détection des patrons de conception.

Efficacité :

Limitée par les outils de modélisation, puisque jusqu'à maintenant peu réussissent d'une part à rétro concevoir les diagrammes de classes et d'autre part aucun outil testé détecte des patrons de conception automatiquement.

Recherche naïve

Outil nécessaire : Très bons logiciels de recherche (eclipse)

Étapes :

1. Rechercher parmi toutes les classes, les noms liés aux patrons de conception
2. Graviter autour des classes qui portent un nom significatif et vérifier si un patron de conception lui est associé

Description détaillée :

Tout d'abord, à l'aide d'un outil de recherche textuelle, rechercher les noms relatifs aux patrons de conception parmi tous les fichiers du programme.

Ensuite à l'aide d'un outil qui est capable de rétro concevoir un programme, on génère le diagramme de classes où l'on dessine à la main les liens possibles autour de la classe ayant un nom significatif.

Finalement, il faut manipuler le diagramme afin de retrouver une microarchitecture conforme aux noms de patrons de conception recherchés. Il faut normalement repositionner les classes sous forme standardisée. Ceci veut dire que les héritages sont positionnés verticalement et les agrégations de façon horizontale. De cette manière, il est bien plus évident de lire le diagramme de classes et ceci accélère la détection des patrons de conception. Commencer par regarder les classes abstraites ou interfaces peut aider à tout moment dans la détection des patrons de conception.

Efficacité :

Comme son nom l'indique, cette méthode est naïve. Nul besoin d'avoir de grandes connaissances des patrons de conception afin de trouver des patrons. Cette méthode est limitée par l'expérience et le bon vouloir du programmeur.

Recherche par héritage

La méthodologie par héritage est la méthodologie qui a été la plus exploitée tout au cours du projet. Efficace, cette méthodologie peut s'appliquer sur l'analyse de n'importe quel programme dans le but de trouver des patrons de conception. Par contre, il faut une compréhension plus approfondie et une connaissance globale de tous les patrons de conception de la part de l'utilisateur pour améliorer les recherches. En effet, un utilisateur qui posséderait une faible compréhension des patrons de conception ou une vue restreinte de ceux-ci résulterait à de faibles découvertes.

Cette méthodologie peut être résumée comme suit :

- Étapes
    1. Rechercher les classes abstraites et les interfaces
    2. Graviter autour de ces classes afin de trouver des liens entre d'autres classes qui pourraient être intéressants
    3. Lorsque la structure nous donne l'idée d'un ou de plusieurs patrons de conceptions, en discuter en groupe
    4. Accepter ou rejeter les idées formées à l'aide de preuve concluante
    5. Passer à d'autres classes abstraites après satisfaction des recherches autour de cette classe abstraite ou interface

Ci-dessous, la procédure à effectuer pour ce type de recherche est expliquée avec un petit exemple de découverte:

1. Il faut tout d'abord rechercher les classes abstraites et les interfaces à l'aide d'outils de rétroconception. À partir de ceux-ci, dessiner tout l'arbre de dérivation défini plus précisément par les mots réservés *extend* et *implement* en java.

Voici un exemple d'arbre de dérivation généré par l'outil de rétroconception Code Logic.

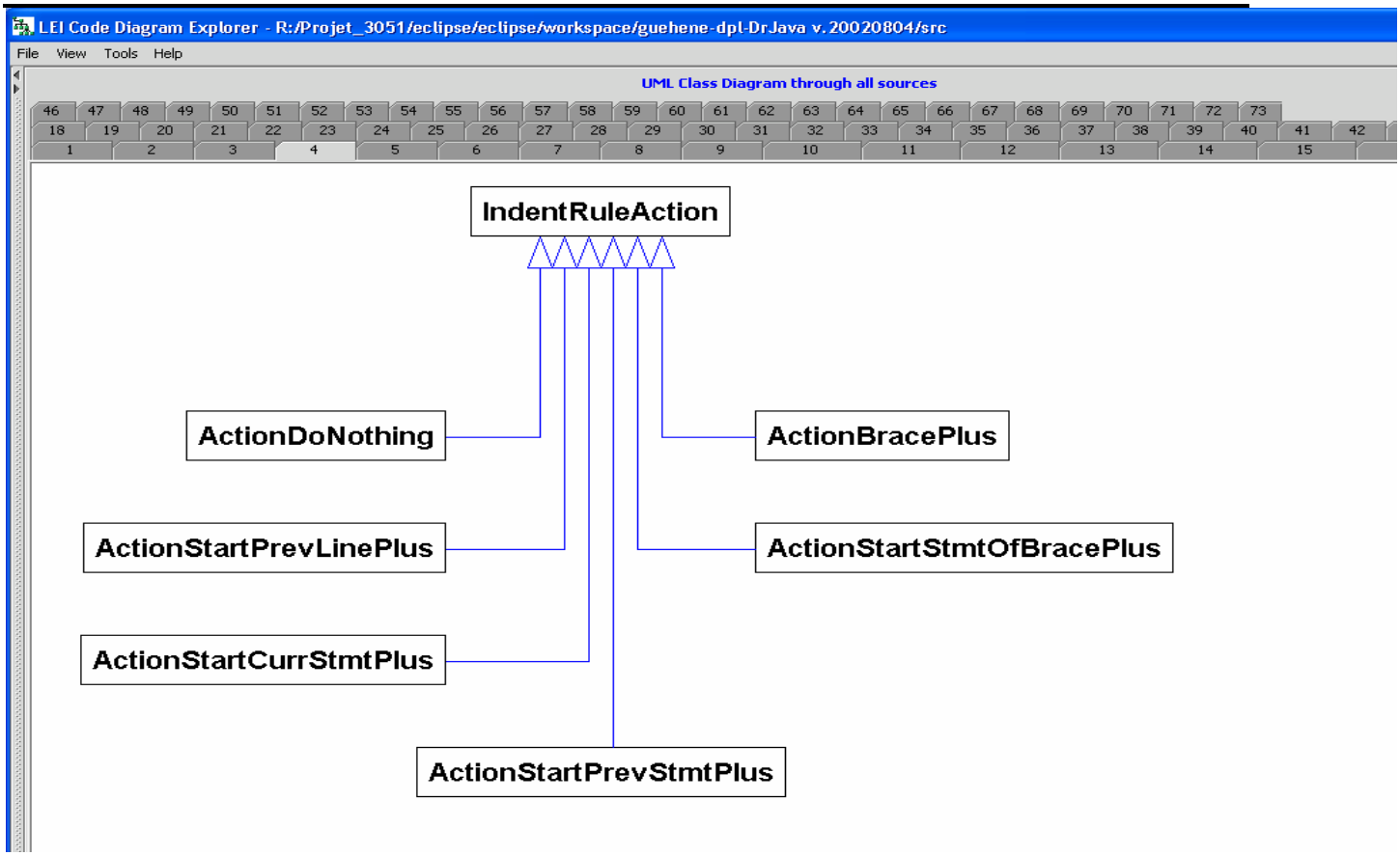


Figure 6 – Détection de patron de conception avec Code Logic

2. Ensuite, majoritairement à l'aide d'Eclipse, il faut graviter autour de cet arbre de dérivation dans le but de trouver des liens d'agrégation ou d'appel de fonction pouvant être contenus dans des patrons de conceptions. Bien évidemment, cette étape est basée sur le jugement de l'utilisateur. Une bonne compréhension et une connaissance globale de tous les patrons de conception sont donc un atout majeur. Elle permet d'avoir une meilleure intuition sur l'importance d'un lien et de ses utilités dans un patron.

3. Lorsque la structure du diagramme de classe inspire la possibilité de certains patrons de conception, des preuves concluantes doivent être élaborées pour confirmer ou infirmer la présence de ces patrons. Il est à noter qu'il est parfois difficile de déterminer à 100% la présence de patron de conception. N'oublions pas que ceux-ci ne sont pas tous implémentés de la même manière. Ils peuvent donc varier d'un programme à un autre.

4. Finalement, lorsque la recherche de patron autour de cet arbre de dérivation semble ne plus générer de résultats, il est temps de revenir à la l'étape 1 en axant ses recherches sur un autre héritage.

Voici un diagramme complet généré par Describe, suivant l'idée de ce type de méthodologie. Notons ici la découverte d'un patron de conception (*state*) grâce aux étapes 2 et 3.

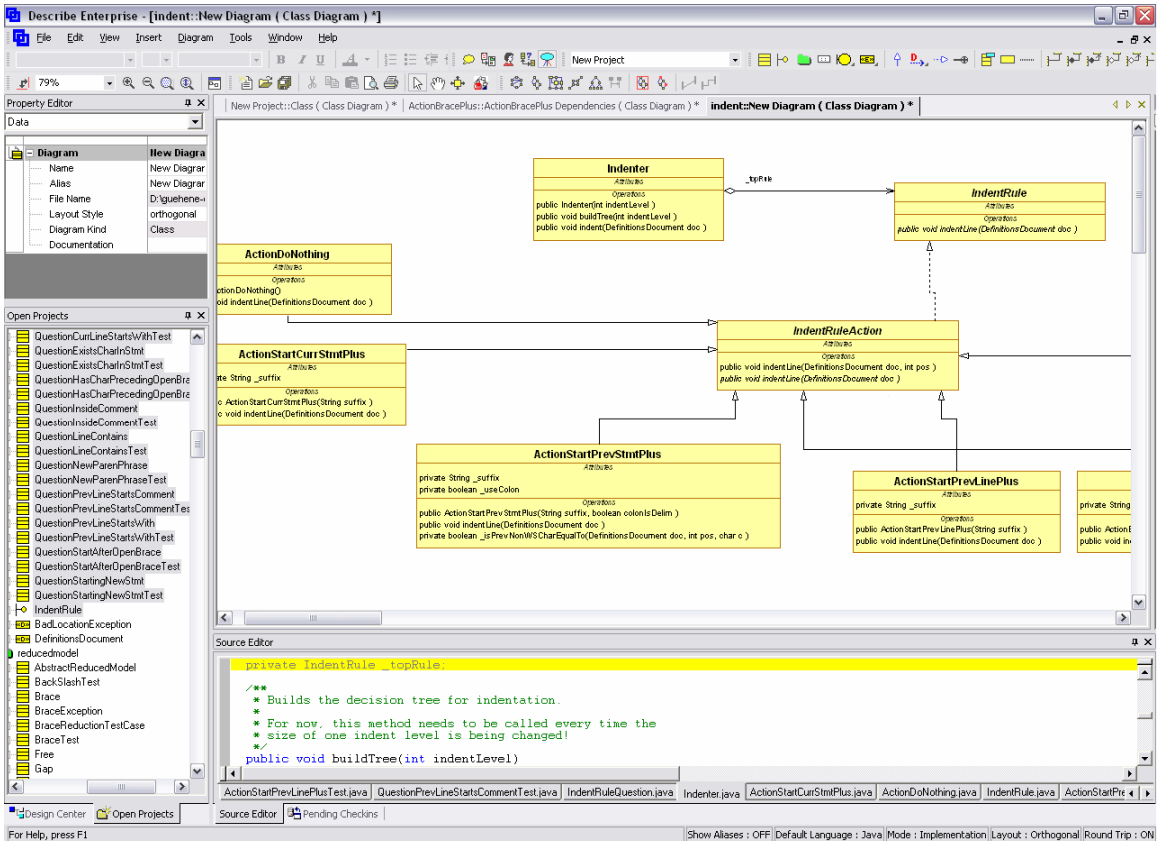


Figure 7 – Détection de patron de conception avec Describe : State

L'application de cette méthodologie lors de l'analyse de Dr Java nous a permis de trouver des patrons de conception majoritairement dans la classe *behavioral*. Cependant, il serait trop tôt pour affirmer que ce type de recherche favorise les découvertes dans cette classe de patron de conception. Il faudrait l'analyse de plus d'un programme (en appliquant cette méthode) pour pouvoir confirmer que cette méthodologie est plus efficace pour certains types de patron.

Recherche par relations des patrons

Cette méthodologie peut se résumer comme suit :

- Étapes
    1. Sélectionner un patron de conception trouvé précédemment et qui n'a pas été marqué;
    2. Identifier un lien potentiel entre le patron sélectionné et des classes avoisinantes, basé sur le schéma des relations du livre de référence;
    3. Sélectionner un autre lien potentiel jusqu'à l'épuisement de tous les liens potentiels existants;
    4. Marquer le patron et retourner à l'étape 1 s'il reste des patrons non marqués.

Cette méthode complète les autres méthodologies, car c'est à partir des patrons trouvés à l'aide de ces méthodologies que nous pouvons effectuer la nouvelle recherche. Le nombre de patrons identifiés auparavant influencera nécessairement sur la quantité de recherche par relations qu'il est possible d'effectuer.

Un lien potentiel peut être identifié, soit à l'aide d'un des outils de rétro conception, ou en examinant les appels de fonction du code source de Dr Java pour y faire correspondre une relation. Pour restreindre les patrons envisageables, le schéma des relations du livre de référence dicte en quelque sorte les patrons qu'il est concevable de retrouver à partir d'un certain patron.

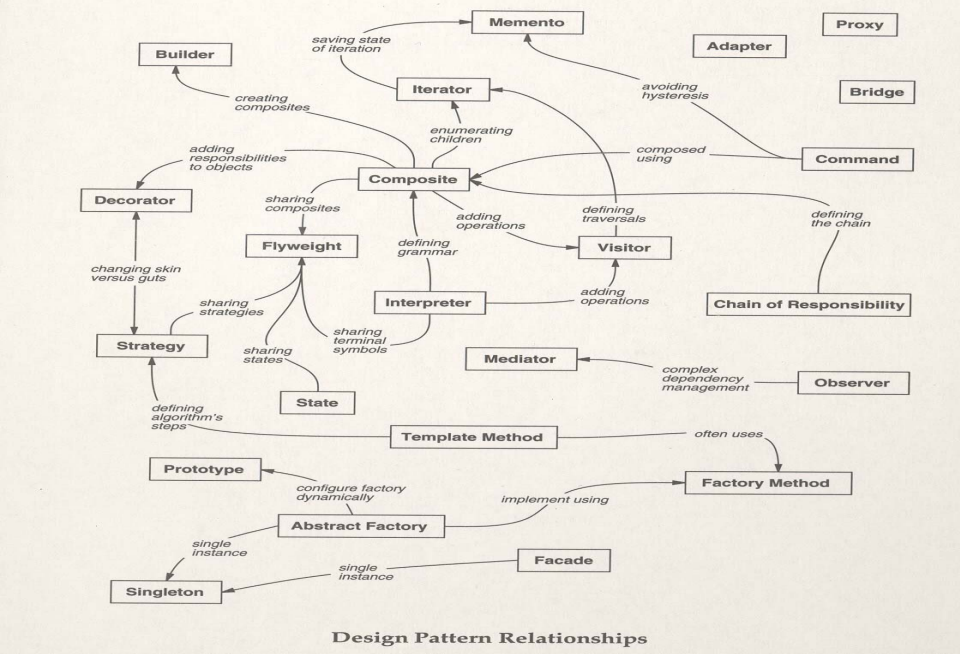


Figure 8 – Diagramme des relations des patrons de conception tiré du livre de référence

Il est intéressant de noter également, qu'il existe une section *Related Pattern*, venant du livre de référence. Cette section justifie les relations définies dans le schéma en expliquant en détails la raison d'être entre la relation de deux patrons. De plus, cette section fournie des liens supplémentaires qui ne sont pas représentés dans le schéma. Pour illustrer cette affirmation, prenons un exemple : un *Bridge* sur le schéma n'a aucun lien avec d'autres patrons. Cependant, dans la section *Related Pattern* décrit un lien possible avec un *Abstract Factory*. Donc, le schéma et la description sont nécessaires pour déterminer des relations possibles.

Il faut également ajouter que ce n'est pas nécessairement parce qu'il n'y a pas de liens sur le schéma des relations qu'il faut rejeter la possibilité que ce patron soit en relation avec le patron sélectionné. Effectivement, lesrelations étant implantées par les développeurs, ceux-ci peuvent ne pas tenir compte de ceux-ci en écrivant et en ajoutant des fonctionnalités à Dr Java. De plus, le schéma n'est qu'une référence.

Cette méthodologie est efficace et elle a surtout été utilisée avec les *Template method*. Les patrons suivants ont été identifiés par cette méthode :

- *2 States*
- *2 Strategy*
- *1 Mediator*
- *1 Bridge*

Par contre, il faut spécifier que cette méthode à également ces limites. En effet, plus les patrons trouvés précédemment ont de liens, plus il est probable de trouver d'autres patrons connexes. En effet, un composite est en relation avec huit autres patrons alors que d'autres patrons ont une interaction beaucoup moindre. Donc, l'identité et la quantité de patrons trouvés préalablement feront varier les résultats de la méthodologie par relations.

Enfin, la méthodologie par relations est celle qui nécessite une meilleure compréhension parmi toutes les méthodologies utilisées. En effet, les trois autres méthodologies peuvent plus ou moins être appliquées et données des résultats sans pour autant avoir une compréhension des patrons de conception.

Méthodologie opportuniste

La méthode opportuniste consiste en un regroupement des méthodologies citées précédemment, c'est-à-dire :

- Par vue globale
- Naïve
- Par héritage
- Par relations des patrons

Effectivement, chacune de ces méthodes ont leurs points forts et faibles. Pour minimiser les points faibles et maximiser la probabilité d'identifier un patron de conception, un mélange de ces méthodes est requis. En effet, les méthodes se complètent les unes par rapport aux autres et il est donc intéressant d'utiliser plusieurs techniques à un moment opportun pour tenter de trouver la présence d'un patron.

Prenons un exemple pour illustrer cette méthode, cela permettra également de montrer comment nous avons procédé pour identifier nos patrons de conception à ce point.

Pour identifier le Mediator de la version 20020804, la recherche a débuté à partir de la découverte d'un *Template method* (*Breakpoint* et *DocumentDebugAction*) : donc utilisation de la méthodologie par héritage.

Ensuite, la méthodologie par relations des patrons a été employée pour tenter d'identifier un patron supplémentaire. Il faut noter que les outils de rétro conception ne représentaient pas de façon convenable cette partie du diagramme de classe puisque ces classes utilisaient des fonctionnalités de Java 1.5. Donc, à l'aide du code source, nous tentions d'identifier un patron avec peu de succès.

À ce point, les noms *DebugManager*, *Breakpoint* et *Step* semblaient être en relations par leurs noms et leurs fonctionnalités dans une application type telle que Dr Java. Nous étions certains qu'une microarchitecture se cachait derrière cela. En examinant les différentes implémentations possibles des différents patrons du livre de référence, il devient clair que le patron en question est un *Mediator*. En effet, la structure n'est pas standard, mais des variances d'implémentation sont possibles et nous étions dans un de ces cas.

Donc, trois des quatre méthodologies ont été utilisées et ont chacune données certains indices sur l'identité du patron recherché ou tout au moins, en permettant d'éliminer certaines possibilités quant à l'identité du patron.

Synthèse des patrons identifiés

Le résultat de l'analyse des versions de Dr Java a été synthétisé dans des tableaux pour que visuellement, il soit facile de constater quelles microarchitectures de patrons ont été identifiées. Les résultats détaillés sous format XML ont été remis au professeur et ils sont également disponibles sur le site web du projet, http://www-etud.iro.umontreal.ca/~lethinhv/

20020703	20020804
Factory method	Factory method
Adapter	Adapter
Bridge	Bridge
Proxy	Proxy
	Mediator
State (3)	State (3)
Strategy (2)	Strategy (2)
Template method (7)	Template method (9)

Tableau 2 – Résultat de la recherche de patron de conception

Ceci étant dit, 20020703 et 20020804 sont les deux versions les plus anciennes qu'il était possible d'analyser. Seulement un mois sépare la date de sortie de ces deux versions. Suite à une analyse poussée, il a pu être déterminé qu'un *Meditor* et deux *Template method* ont été ajoutés dans l'intervalle d'un mois.

En regroupant, les patrons de conceptions retrouvés au cours du projet avec ceux retrouvés avec l'équipe précédente, les patrons suivants sont obtenus :

20020619-20020703-20020804	Nombre de patrons
Factory method	1
Singleton	8
Adapter	2
Bridge	1
Proxy	1
Command	2
Iterator	1
Mediator	1
Momento	1
State	3
Strategy	3
Template method	9

Tableau 3 – Regroupement de 3 versions analysées

Leur version étant 20020619, ces trois versions se succèdent et sont les trois versions les plus anciennes de Dr Java. Notons que leurs patrons sont en italique. Comme il a été mentionné plutôt, l'équipe précédente avait opté pour la méthodologie naïve. Le fait qu'il y a eu une spécialisation de notre part envers la méthodologie par héritage et par relations résulte en une divergence des patrons identifiés. Seules deux découvertes se chevauchent (*Proxy* et un *Strategy*). Ceci confirme donc qu'une variation des méthodologies permet de compléter une recherche de patrons de conception, puisqu'il faut le dire, leurs résultats ont été consultés qu'en fin de projet.

Évolution des patrons des patrons de conception dans DrJava

Regardons les changements notés des patrons de conception dans les 2 premières versions analysées par rapport à la dernière version. Les versions analysées sont : 2002-06-19 & 2002-08-04 et la version partiellement analysée est 2004-03-26

Voici les résultats :

- ***Mediator* (3901):**
  - La première version : pas de patron;
  - Dans la version intermédiaire, le c*oncreteMediator* et le *Mediator* étaient dans la même classe;
  - Dans la dernière version, *DebugManager* s'est divisé en deux : *Debugger* et *JPDADebugger*

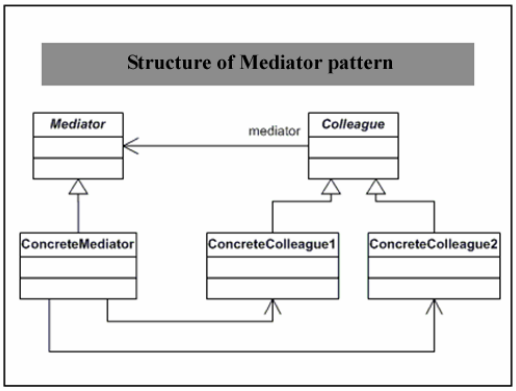
- ***State* (2085) :** Ajout de 3 *concreteState*
- ***Strategy* (2070) :** Ajout de 2 *concreteStrategy*
- ***State* (2079) :** Ajout d'un *concreteState*

Note : le numéro entre parenthèses représente le numéro d'identification unique dans les fichiers XML.

Regardons plus en détail l'évolution du Médiateur et du State (2085)

Évolution du Médiateur

Voici le premier exemple de l'évolution d'un patron de conception dans Dr Java à travers le temps.



Qu'est-ce qu'un médiateur ?

Définis un objet qui encapsule les modalités d'interaction de divers objets
Le médiateur favorise un couplage faible en dispensant les objets d'avoir à faire référence explicitement les uns aux autres.

Figure 9 – Modèle du patron de conception Médiateur

Dans notre analyse, il est clair que dans la première version de Dr Java, toutes les fonctionnalités du mode de débuggeur n'étaient pas finies d'être écrites d'où le manque du médiateur.

Dans la version intermédiaire, on constate la première évolution. Il est possible de voir qu'une nouvelle fonctionnalité s'ajoute au débuggeur. L'implémentation du débuggeur avec *breakpoint* et *step* y est maintenant présente. Il est difficile de dire si cette option est fonctionnelle dans cette version ou simplement une trace du début de l'instauration de cette nouvelle fonctionnalité.

Dans la dernière version, partiellement analysée, on voit que la fonctionnalité de débogueur est maintenant sous forme standard d'un médiateur.

Regardons plus en détail à l'aide d'Eclipse

Il est facile de le constater avec ce diagramme que dans la version 2002-06-19, il n'y a aucune trace de médiateur.

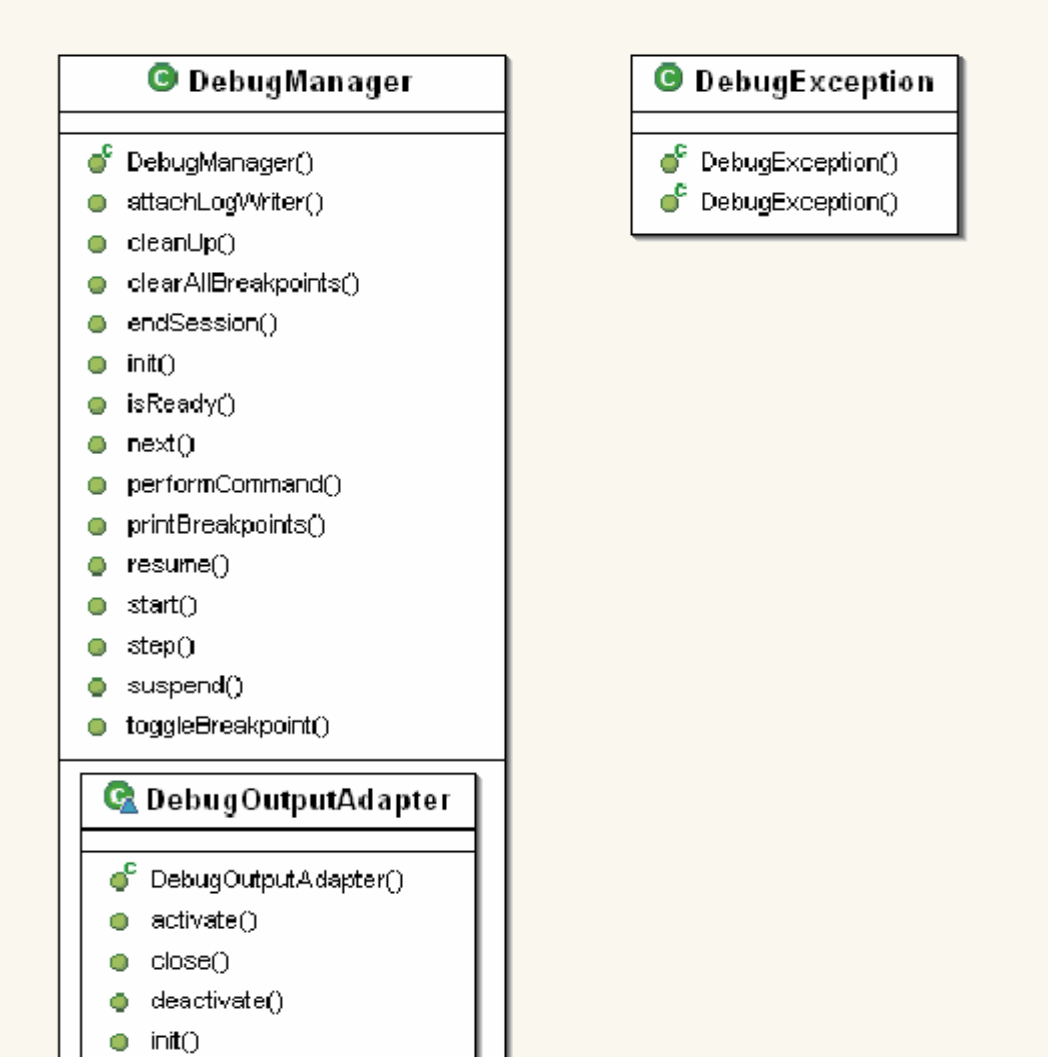


Figure 10 – Médiateur manquant dans la première version de Dr Java

Dans la version intermédiaire, il est plus difficile de constater qu'un médiateur est présent. Le *concreteMediator* et le *Mediator* sont dans la même classe.

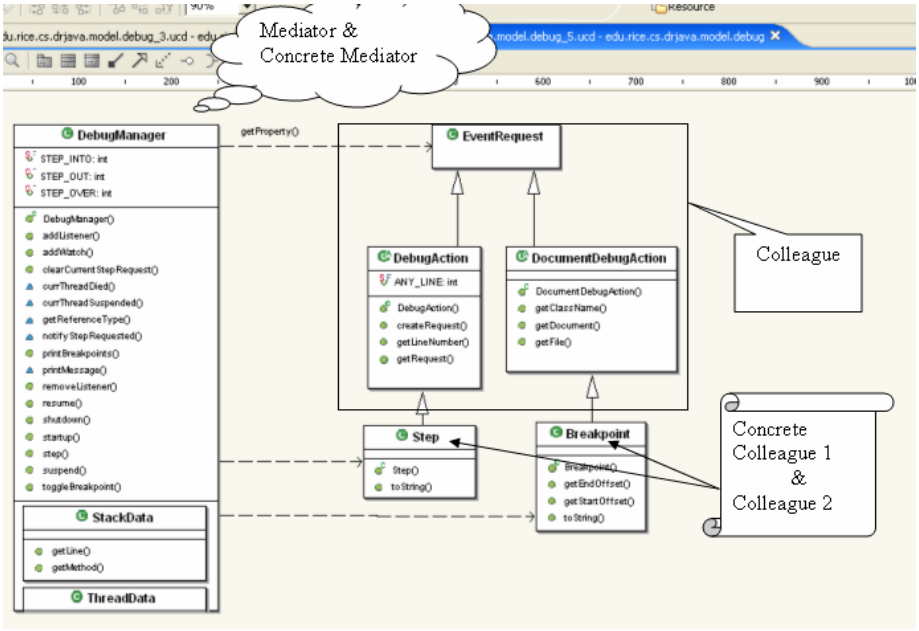


Figure 11 – Diagramme de classes du médiateur dans la version intermédiaire

Le médiateur dans la dernière version de Dr Java, la classe *DebugManager* se scinde en deux, ainsi le *Mediator* devient *Debugger*, une interface, et le *ConcreteMediator* devient *JPDADebugger*, une classe usuelle.

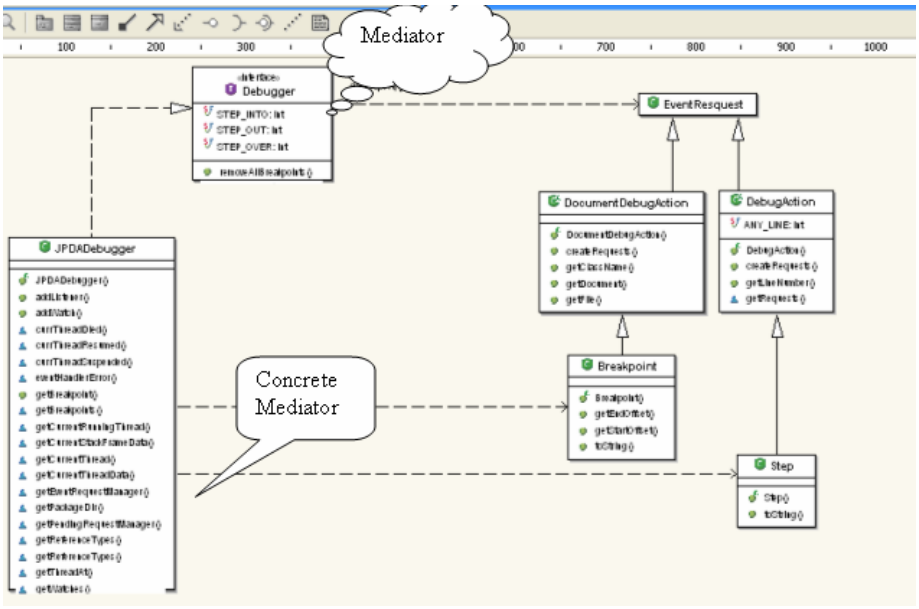
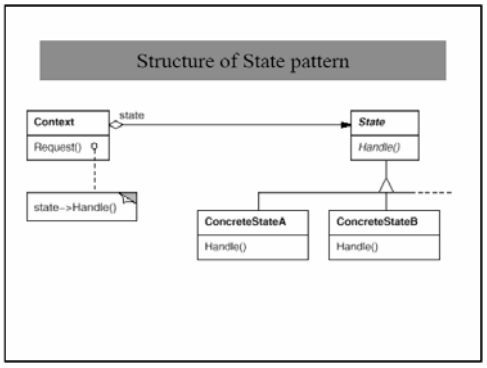


Figure 12 – Diagramme de classes du médiateur dans la dernière version

Évolution du State 2085

Voici un deuxième exemple d'évolution de patron de conception noté lors de l'analyse des différentes versions du programme de Dr Java.



Un *state* facilite l'implémentation d'une classe qui aurait la capacité de changer de comportement en changeant d'état.

Voici un schéma représentant la structure générale de ce patron de conception : (livre de référence)

Figure 13- Modèle du patron de conception State

Considérons le succès de la recherche de ce type de patron de conception dans une première version de Dr Java. Le dessin ci-dessous montre ce patron. *IndenRuleAction* joue le rôle du *State* et les classe *Action…* représentent tous ses états possibles (*Concrete State*).

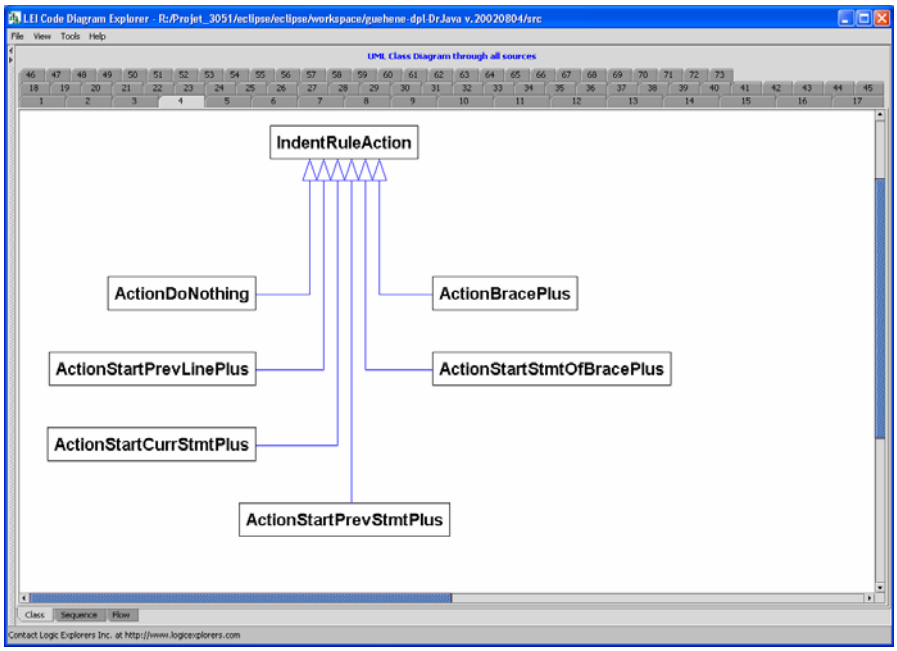


Figure 14- Vue du state à l'aide de Code Logic

Notez l'ajout de trois nouveaux états (*Concrete State*) dans une version plus récente. (Elles sont encadrées en rouge avec un fond gris)

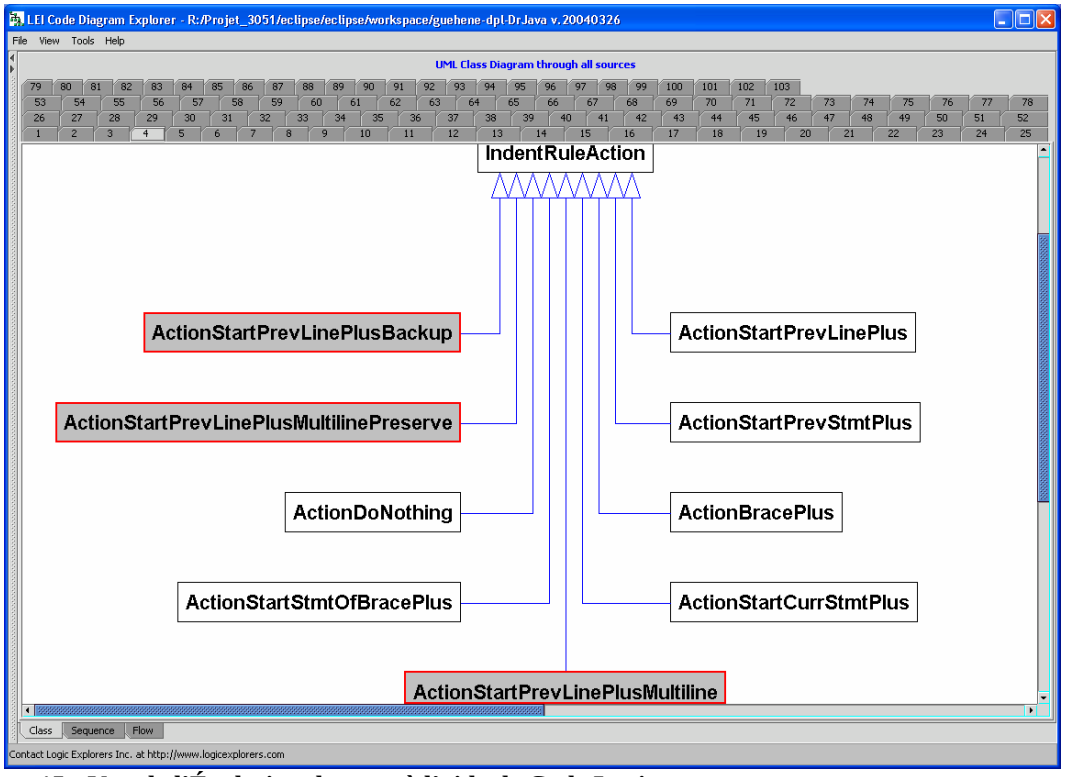


Figure 15- Vue de l'Évolution du state à l'aide de Code Logic

Cette évolution du patron de conception signifie le succès de l'utilisation de celui-ci. En effet, l'utilisation d'un *State* a permis au programmeur l'ajout facile de trois nouveaux comportements de la classe *IndentRuleAction* en définissant trois nouvelles classes héritant de celle-ci.

Nous pouvons en déduire qu'à cette version, un nouvel aspect du programme a vu le jour. L'explication de ceci est fort simple; en se concentrant sur les noms des nouvelles classes, il est facile de constater que deux d'entre elles possèdent le sous mot *Multiline*, qu'aucune classe antérieure ne possède. Du même coup, l'hypothèse qu'un nouvel aspect du programme permettant le support de lignes multiples vient d'apparaître dans cette version nous vient à l'esprit. Les programmeurs ont donc, sous l'utilisation du *State*, implanté ces trois nouvelles classes pour pouvoir supporter cet aspect ou fonctionnalité. Bien sûr, tout ceci n'est qu'une déduction rapide. Il nous faudrait exécuter le programme ou au moins, regarder dans le commentaire pour pouvoir appuyer notre hypothèse.

Conclusion

Lors de la première partie du projet, il a été décevant de constater que les outils miraculeux et efficaces ne sont pas au rendez-vous. En effet, nos attentes étaient bien plus élevées face à ces logiciels. Tout au long du projet, plusieurs outils ont été analysés et rejetés. De nombreux programmes étaient limités par des restrictions de licences et n'ont pas pu être pleinement testés. Néanmoins, l'avantage que nous apportent les outils de réingénierie, en modélisant les classes, est non négligeable. Cela nous permet d'éliminer incontestablement de nombreuses recherches à travers plusieurs lignes de code. Le raccourci visuel que nous apportent ces outils aide fortement à une première lecture d'un programme.

Durant la deuxième partie du projet, il est intéressant de noter qu'une vingtaine de patrons de conception ont été détectés. La majorité de ces patrons de conception ont été détectés par la méthodologie héritage et relations des patrons. Cependant, garder l'œil ouvert et bien maîtriser les patrons de conception permet de maximiser la recherche. Il faut dire que la majorité des patrons de conception détient une classe de type abstrait et/ou interface. Cerner les classes de ce type aide grandement au décèlement des patrons.
Les fichiers XML qui contiennent l'ensemble de nos résultats permettront à l'équipe de développement de Ptidej de calibrer leur outil de rétro conception. En effet, ces résultats permettront à l'équipe de Ptidej de vérifier l'exactitude des patrons détectés par Ptidej en se basant sur un ensemble de patrons déjà connus.

Remerciements

Nous voudrions remercier Yann-Gaël Guéhéneuc , Duke Huynh D.L. et Naouel Moha pour leur grande disponibilité et l'intérêt apporté à toutes nos questions.

Appréciation personnelle et conseil

Avant de débuter ce projet de 3051, j'avais une vague idée de ce qu'était un patron de conception. Maintenant que le projet est terminé, je me demande comment j'aurais pu graduer sans avoir appris tant de choses sur ceux-ci.

Grâce à cet apprentissage, sous forme de projet réalisé en équipe, je crois fortement que ma vue sur un programme ne sera plus jamais la même. En effet, ce projet nous a tous aidé à voir les programmes d'un autre point de vu. L'analyse et même la conception de ceux-ci seront une tâche beaucoup moins ardue que nous le croirons au début du projet. J'affirme donc avec certitude qu'un programmeur en orienté objet se doit de posséder le bagage de connaissances des patrons de conception.

Malgré toutes les difficultés rencontrées lors du projet, j'ai très apprécié faire sa réalisation. Ce travail n'était pas que sur les patrons de conception, mais aussi sur le travail en équipe. Il est parfois difficile de s'entendre sur quelques points en équipe. Plusieurs chamailleries ont failli éclater tout au long du projet. Mais ceci reflète tout aussi bien ce qui nous attend sur le marché du travail. Le projet nous a donc aussi donné de l'expérience non négligeable sur ce point.

À la prochaine équipe de 3051 Patrons de conception :

S'il y a une prochaine équipe de projet sur les patrons de conceptions, voici quelques conseils que je vous laisse :

- Une bonne lecture du livre de référence donne un atout face à la recherche de patron de conception, car une connaissance et une compréhension globale de ceux-ci sont primordiales pour la recherche.

- Il n'existe pas d'outil magique qui peut remplacer une bonne compréhension du chercheur combiné avec des bonnes techniques de recherche.

- N'hésitez pas à poser des questions lorsque vous n'êtes pas sûre de certains points qui pourraient être importants. Il est normal que vous ne connaissiez rien sur la recherche,… Si oui, il n'y aurait aucune valeur à vous évaluer sur cela.

- Ne soyer pas toujours trop sérieux lors de la réalisation du projet. Vous êtes là pour être évalué, mais aussi pour passer du bon temps tout en apprenant.

Bonne chance et gros merci à tout le monde impliqué tout au long du projet.

Joseph

Les outils de rétro conception détectant automatiquement et parfaitement des patrons de conception n'ont pas encore vu le jour. Alors, un projet de détection des patrons est encore d'actualité et permet un cheminement personnel incroyable. Une expérience unique est acquise tout au long du projet.

Pour ce qui est de la détection de patron de conception, quelques points essentiels doivent être prisent en compte pour maximiser l'identification des patrons de conception. Tout d'abord, un outil de rétro conception offre une vue d'ensemble du diagramme de classe de l'application à analyser. Il faut toutefois ne pas se fier aveuglément à leur résultat. Une re-vérification des résultats s'impose toujours.

Ensuite, bien que des méthodologies permettent un encadrement accru lors de la recherche de patrons, l'instinct demeure le facteur le plus déterminant. Des indices peuvent parfois indiquer la présence des patrons, mais comment savoir qu'il faille stopper les recherches à un certain endroit ou continuer, l'instinct le permet. Celui-ci doit reposer sur des connaissances solides et non sur un je ne sais quoi aléatoire.

Dans le même ordre d'idée, la détection de microarchitectures vient avec l'expérience et rien ne peut la remplacer. Lorsqu'un même type de patrons est décelé, l'intention et la raison d'être de ce patron comprise. De plus, il ne faut pas négliger qu'une multitude d'implémentations existe.

Enfin, pour ce type de projet, une équipe de deux suffit. Une structure trop grande n'apporte que des ennuis et souvent, un seul avis complémentaire est nécessaire pour vérifier la véracité d'une hypothèse.

Vinh Thinh Le

Depuis mon premier cours de génie logiciel à l'Université de Montréal, j'étais perplexe à l'égard de cette discipline. Je croyais que cette science était réservée uniquement au très gros projet. Cependant, je me suis rendu compte assez rapidement que cela n'était pas le cas

Lors de la réalisation de notre projet, nous avons constaté que Dr Java est un petit programme, mais très bien structurée. Les ingénieurs qui ont travaillés à la réalisation de ce progiciel ont sans aucun doute pensé longuement au principe de base. Effectivement, on peut voir qu'un découpage précis du chemin à parcourir permet une bonne identification et une meilleure optimisation des ressources, des délais, des charges de travail et des coûts.

La détection des patrons de conception, n'est pas chose facile. Le départ de notre projet fut laborieux, voire même chaotique. Nous nous sommes promenés au travers de plusieurs logiciels : Dr Java, Junit, InfoGlue. Initialement, nous avons eu de la difficulté à faire fonctionner les logiciels de réingénierie et nous avons rapidement constaté qu'il n'en existait pas une grande panoplie, donc les choix de logiciels étaient assez restreints.

Ainsi, afin d'accroître notre efficacité, il a été naturel que nous nous interrogions individuellement et collectivement sur nos méthodes de conduite de projet. Le renouvellement des progiciels et des petits logiciels fut l'occasion favorable à une telle réflexion.

Ce projet nous a permis de constater que bien que la détection de patrons de conception était une chose difficile, celle-ci l'est encore davantage lorsqu'un programme est mal écrit et mal commenté. Mon appréhension face aux outils de réingénierie était bien plus élevée. Toutefois, nous sommes encore qu'au début de l'évolution du génie logiciel et la méthode « crayon papier » demeure la meilleure méthode pour apprendre. Il est évident que si les outils nous avaient fourni les patrons, nous n'aurions jamais autant appris.

Ce projet est une étape importante dans le cheminement d'un bachelier en informatique, la bonne compréhension des patrons de conception, sera un atout majeur, à plusieurs niveaux dans la pour suite de ma carrière dans le domaine de l'informatique.

Dans ce projet, le travail d'équipe a été une très grande source de motivation qui nous a permis, à tous, de repousser nos propres limites. Ce projet a été pour moi une expérience très enrichissante dans laquelle j'ai appris à apprivoiser le génie logiciel. Je suis heureux d'avoir participé à ce projet.

Mille mercis à la super équipe d'encadrement qui a su nous appuyer tout au long du projet. Les questions avec vous ont toujours étés les bienvenues et cela a fait toute une différence.

Sébastien

Bibliographie, Références

Livre :

[1] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for objectoriented design. Technical report E53-315, MIT Sloan School of Management, December 1993.
[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1st edition, 1994. isbn: 0-201-63361-2.
[3] Peter Wendorff. Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. In Pedro Sousa and Jurgen Ebert, editors, proceedings of 5th Conference on Software Maintenance and Reengineering, pages 77–84. IEEE Computer Society Press, March 2001.Design Patterns: Elements of Reusable Object Oriented Programming Software
ISBN: 0201633612

Internet :

http://www.dofactory.com/Patterns/Patterns.aspx#list

ANNEXE A – Grille des outils testés

Nom du logiciel : Describe Entreprise 6.1.7

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X X X			
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.	X			
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement du diagramme de classe	X			Circulaire, arbre, symétrique….
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)	X			
nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats	X			En théorie, dans la version complète
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	
Génère un API à partir du diagramme de classe	X			
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)	X		1 2 3 4 5	
Supporte Java 1.5		X		Met Data type
Supporte UML 2.0	X			
Limitations du programme	X			30 jours, pas export
Langage de programmation supporté (autre que java)	X			C++, C#, VB
Qualité de la documentation			1 2 3 4 5	
Indépendance sur différentes plateformes				
Offre les options de Undo / Redo récursif		X		Pas undo/redo
Facilité d'installation	X			
Autres				
Bugs notables				
Commentaires	Catalogue de patrons de conceptions et possibilité de les appliquer Agrégation			

Nom du logiciel : Omondo

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X X X			
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.	X			
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement du diagramme de classe	X			
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)		X		Fonctionne meme s'il y a des petites erreurs
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats	X			Limiter par la version gratuite
Convivialité et facilité d'utilisation du logiciel	X		1 2 3 4 5	
Génère un API à partir du diagramme de classe		X		
Esthétique des diagrammes et de rétro conception	X		1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)	X		1 2 3 4 5	
Supporte Java 1.5	X			
Supporte UML 2.0	X			
Limitations du programme	X			Sur les prints et sauvegarde
Langage de programmation supporté (autre que java)		X		
Qualité de la documentation			1 2 3 4 5	
Indépendance sur différentes plateformes				
Offre les options de Undo / Redo recursif		X		Dans le code oui mais pas sur les diagrammes
Facilité d'installation	X			
Autres				
Bugs notables	Print mais car on ne peut pas dans la version gratuites			
Commentaires	Excellent plug'in pour éclipse			

Nom du logiciel : CodeLogic

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X	 X X X		Dessine de très beaux diagrammes de classe et de séquences facile à comprendre.
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.		X		
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions		X	1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement automatique du diagramme de classe		X		Ne bouge pas (static)
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)		X		
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats	X			PHG, Visio.net, XMI,…
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	
Génère un API à partir du diagramme de classe		X		
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	Aucune modification possible
Supporte Java 1.5				
Supporte UML 2.0	X			
Limitations du programme	X			Trial Version
Langage de programmation supporté (autre que java)	X			C#
Qualité de la documentation			1 2 3 4 5	
Indépendance sur différentes plateformes		X		
Offre les options de Undo / Redo recursif		X		
Facilité d'installation		X		Licence requis à chaque fois
Autres				
Bugs notables				
Commentaires	Excellent programme qui fait tout ce que l'on a besoin a première vu. Malheureusement le trial nous limite a moins d'options que le programme peut offrir.			

Nom du logiciel : Fujaba

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X 	 X X		Génère un diagramme activité et classe à partir du code
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.				
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	Possède un jar qui détecte des design pattern
Critères secondaires				
Possibilité de faire un réalignement automatique du diagramme de classe	X			
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)				
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats	X			svg, pdf, jpeg, tiff, png
Convivialité et facilité d'utilisation du logiciel	X		1 2 3 4 5	
Génère un API à partir du diagramme de classe		X		
Esthétique des diagrammes et de rétro conception	X		1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	
Supporte Java 1.5				
Supporte UML 2.0	X			Probablement
Limitations du programme		X		
Langage de programmation supporté (autre que java)		X		
Qualité de la documentation		X	1 2 3 4 5	Inexistance, sauf quelques pages sur le site
Indépendance sur différentes plateformes				
Offre les options de Undo / Redo recursif		X		Aucun retour en arrière possible
Facilité d'installation	X			
Autres				
Bugs notables	Plugins supplémentaires ne marche pas			
Commentaires	Peut compiler, run, créer des jar. Appliquer des patrons conceptions aussi.			

Nom du logiciel : MagicDraw 9.5 demo version

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X X X			
Possibilité de rétro conception	X			Créer sur un nouveau diagramme ou courant
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.				
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement du diagramme de classe	X			Circulaire, orthogonal, hierachique, tree, organic
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)		X		
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		Fichier java ou jar. On peut ajouter les répertoires de façon récursive
Possibilité de sauvegarder les diagrammes en différents formats	X			Différents types images
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	Difficile de retrouver une classe que l'on sélectionne sur le menu sur le diagramme de classe
Génère un API à partir du diagramme de classe	X			
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	Beaucoup de chevauchements des associations.
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	
Supporte Java 1.5	X			Mais recommandé pour 1.4.2
Supporte UML 2.0	X			
Limitations du programme	X			Impossible de sauvegarder au dessus de 20 classes
Langage de programmation supporté (autre que java)	X			C#, C++
Qualité de la documentation			1 2 3 4 5	
Indépendance sur différentes plateformes				
Offre les options de Undo / Redo recursif	X			
Facilité d'installation	X			
Autres				
Bugs notables				
Commentaires	-Possède un zoom control -affiche des messages erreurs s'il y en a lors de la rétro -option de CVS - option pour vérifier la syntaxe - option pour appliquer des design pattern lorsque l'on crée des diagramme de classe.			

Nom du logiciel : Visual UML 4 version démo

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X X X			
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.				
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement automatique du diagramme de classe		X		Manuel
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)				
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats	X			XML, jpeg, tiff etc
Convivialité et facilité d'utilisation du logiciel	X		1 2 3 4 5	
Génère un API à partir du diagramme de classe	X			
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	
Supporte Java 1.5				
Supporte UML 2.0	X			
Limitations du programme	X			Le mot démo est écrit sur les diagrammes
Langage de programmation supporté (autre que java)	X			C++,C#, VB, access, cache CDL, possèede une vraiment longue liste
Qualité de la documentation			1 2 3 4 5	Un help
Indépendance sur différentes plateformes				
Offre les options de Undo / Redo recursif	X			
Facilité d'installation	X			
Autres				
Bugs notables				
Commentaires	Zoom control, possède du potentiel			

Nom du logiciel : ObjectIF Personal Edition

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X	 X X		
Possibilité de rétro conception				
Pertinence du résultat de la rétro conception	X		1 2 3 4 5	
Possibilité de définir les associations pertinentes.				
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement automatique du diagramme de classe		X		
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)				
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		Accepte également les .jar
Possibilité de sauvegarder les diagrammes en différents formats		X		
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	
Génère un API à partir du diagramme de classe	X			En .txt
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	
Supporte Java 1.5		X		
Supporte UML 2.0	X			
Limitations du programme	X			Pas export
Langage de programmation supporté (autre que java)	X			C++, C#, VB
Qualité de la documentation	X		1 2 3 4 5	
Indépendance sur différentes plateformes		X		
Offre les options de Undo / Redo recursif		X		Aucun redo
Facilité d'installation	X			
Autres				
Bugs notables				
Commentaires	-Créer un environnement de travail prend une éternité. -Peut fonctionner seul, avec Eclipse ou JBuilder pour Java. -Pas Zoom Control. -Offre peut de fonctionnalité.			

Nom du logiciel : Metamill 4.1

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X X	 X		
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.				
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement automatique du diagramme de classe		X		Manuel
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)				
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats	X			
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	
Génère un API à partir du diagramme de classe	X			Avec dictionnaire
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	
Supporte Java 1.5				
Supporte UML 2.0				
Limitations du programme	X			30 jours
Langage de programmation supporté (autre que java)	X			C++, C#
Qualité de la documentation			1 2 3 4 5	Inexistante
Indépendance sur différentes plateformes		X		
Offre les options de Undo / Redo recursif	X			
Facilité d'installation	X			
Autres				
Bugs notables				
Commentaires	On peut exporter en format XML			

Nom du logiciel : Poséidon for UML Pro

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X X X			
Possibilité de rétro conception	X			Import : jar , mdl, java
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.	X			
Vitesse pour générer le diagramme de classe	X		1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement du diagramme de classe	X			
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)		X		Ignorer les erreurs
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats	X			En image sinon UMLDOC
Convivialité et facilité d'utilisation du logiciel	X		1 2 3 4 5	
Génère un API à partir du diagramme de classe	X			UmlDoc
Esthétique des diagrammes et de rétro conception	X		1 2 3 4 5	Mal dispose
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)	X		1 2 3 4 5	
Supporte Java 1.5	X			Derniere version
Supporte UML 2.0	X			
Limitations du programme		X		30 jours
Langage de programmation supporté (autre que java)		X		
Qualité de la documentation	X		1 2 3 4 5	
Indépendance sur différentes plateformes	X			Mac, linux
Offre les options de Undo / Redo recursif	X			
Facilité d'installation	X			
Autres				
Bugs notables	ERREUR de Fin de ligne incomprise ☹			
Commentaires				

Nom du logiciel : Enterprise Architect v5.0

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X X X			
Possibilité de rétro conception	X			Projet -> Source -> import
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.	X			
Vitesse pour générer le diagramme de classe	X		1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement du diagramme de classe	X			
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)		X		
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes	X			
Possibilité de sauvegarder les diagrammes en différents formats	X			
Convivialité et facilité d'utilisation du logiciel	X		1 2 3 4 5	
Génère un API à partir du diagramme de classe	X			
Esthétique des diagrammes et de rétro conception	X		1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)	X		1 2 3 4 5	
Supporte Java 1.5		X		
Supporte UML 2.0	X			
Limitations du programme		X		30 jours
Langage de programmation supporté (autre que java)	X			C, php,…
Qualité de la documentation	X		1 2 3 4 5	
Indépendance sur différentes plateformes	X			
Offre les options de Undo / Redo recursif		X		
Facilité d'installation	X			
Autres				
Bugs notables				
Commentaires				

Nom du logiciel : PowerDesigner

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X X X			
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.	X			
Vitesse pour générer le diagramme de classe	X		1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement du diagramme de classe	X			
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)		X		
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats		X		Print sur 135 pas très bien
Convivialité et facilité d'utilisation du logiciel	X		1 2 3 4 5	Peut d'option pour la rétro
Génère un API à partir du diagramme de classe		X		
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	
Supporte Java 1.5		X		
Supporte UML 2.0		X		1.3 Export only
Limitations du programme		X		
Langage de programmation supporté (autre que java)	X			Bases de données
Qualité de la documentation			1 2 3 4 5	Rien sur la rétro et documentation faible
Indépendance sur différentes plateformes		X		
Offre les options de Undo / Redo recursif	X			
Facilité d'installation	X			
Autres				
Bugs notables				
Commentaires	Bien plus pour les bases de données			

Nom du logiciel : Objecteering / UML Modeler

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X	 X X X		
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.	X			
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement du diagramme de classe	X			
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)		X		
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats		X		Print bien laid sans option
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	
Génère un API à partir du diagramme de classe				
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	
Supporte Java 1.5	X			e
Supporte UML 2.0	X			
Limitations du programme		X		
Langage de programmation supporté (autre que java)		X		
Qualité de la documentation			1 2 3 4 5	
Indépendance sur différentes plateformes	X			
Offre les options de Undo / Redo recursif	X			
Facilité d'installation		X		2/3 non réussi a installé
Autres				
Bugs notables	Erreur sans message dans la rétro conception			
Commentaires				

Nom du logiciel : ESS- Model

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	X	X X X X		Le programme est de moins de 1mb. Il est donc compréhensible qu'il génère le stricte nécessaire.
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.		X		
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions		X	1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement automatique du diagramme de classe		X		
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)		X		
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats	X			.wmf et .png
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	
Génère un API à partir du diagramme de classe	X			
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)		X	1 2 3 4 5	
Supporte Java 1.5		X		
Supporte UML 2.0		X		
Limitations du programme		X		
Langage de programmation supporté (autre que java)	X			Kilix
Qualité de la documentation			1 2 3 4 5	
Indépendance sur différentes plateformes		X		
Offre les options de Undo / Redo recursif		X		
Facilité d'installation	X			Extraction simple
Autres				
Bugs notables				
Commentaires	Petit programme "cute" et facile d'utilisation mais loin d'être compétent pour notre travail.			

Nom du logiciel : Ideogramic UML

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	 X X X X	X		
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.		X		
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions		X	1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement automatique du diagramme de classe	X			Pas fameux.
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)		X		
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes		X		
Possibilité de sauvegarder les diagrammes en différents formats	X			
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	
Génère un API à partir du diagramme de classe		X		
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)		X	1 2 3 4 5	Le trial ne permet que les diagramme de classe
Supporte Java 1.5		X		
Supporte UML 2.0	X			
Limitations du programme	X			Trial (ne comporte pas tout les options)
Langage de programmation supporté (autre que java)	X			C++
Qualité de la documentation			1 2 3 4 5	
Indépendance sur différentes plateformes		X		
Offre les options de Undo / Redo recursif	X			Très bien
Facilité d'installation	X			
Autres				
Bugs notables	Le Html générer est sans image.Lors de la retro-conception de certain programme, l'application « plante ». (i.e JUnit)			
Commentaires	Permet le dessin de diagramme avec un environnement agrable.			

Nom du logiciel : Structural Analysis for Java

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration	X	X X X X		Tres complet avec tout sorte d'option disponible. Diagramme représenter sous une forme non standard.
Possibilité de rétro conception	X			
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.	X			Excellent
Vitesse pour générer le diagramme de classe			1 2 3 4 5	Dépend des l'option de retro-conception.
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement automatique du diagramme de classe		X		Mais reste bien aligné
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)	X			
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes	X			.class
Possibilité de sauvegarder les diagrammes en différents formats	X			
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	Complexe
Génère un API à partir du diagramme de classe				
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	
Supporte Java 1.5		X		
Supporte UML 2.0	X			
Limitations du programme		X		
Langage de programmation supporté (autre que java)		X		
Qualité de la documentation			1 2 3 4 5	
Indépendance sur différentes plateformes	X			Linux, Win, Solaris
Offre les options de Undo / Redo recursif		X		
Facilité d'installation	X			
Autres				
Bugs notables	- Impossibilité de quitter l'application (parfois)--- (la seul option devient le ctrl-alt-del) - La recherche se fait sur le lecteur disquette par default (agaçant au début de la retro-conception)			
Commentaires	Tres bon programme. Complet et facile d'utilitariste. La seule chose qui est négatif est que les représentations ne sont pas sous la forme habituelle.			

Nom du logiciel :

Critères primordiaux	Oui	Non	Évaluation	Commentaires
Diagrammes UML supportés - Diagramme de séquence - Diagramme de classes - Diagramme d'activités - Diagramme d'états - Diagramme de collaboration				
Possibilité de rétro conception				
Pertinence du résultat de la rétro conception			1 2 3 4 5	
Possibilité de définir les associations pertinentes.				
Vitesse pour générer le diagramme de classe			1 2 3 4 5	
Pertinence du résultat des patrons de conceptions			1 2 3 4 5	
Critères secondaires				
Possibilité de faire un réalignement du diagramme de classe				
Obligation d'avoir un code sans erreur pour générer un diagramme de classe (compilable)				
Nécessité d'avoir certains types de ficher (autre que .java) pour générer les diagrammes				
Possibilité de sauvegarder les diagrammes en différents formats				
Convivialité et facilité d'utilisation du logiciel			1 2 3 4 5	
Génère un API à partir du diagramme de classe				
Esthétique des diagrammes et de rétro conception			1 2 3 4 5	
Synchronisation d'une modification à un diagramme sur les autres diagrammes (refractoring)			1 2 3 4 5	
Supporte Java 1.5				
Supporte UML 2.0				
Limitations du programme				
Langage de programmation supporté (autre que java)				
Qualité de la documentation			1 2 3 4 5	
Indépendance sur différentes plateformes				
Offre les options de Undo / Redo recursif				
Facilité d'installation				
Autres				
Bugs notables				
Commentaires				