N° d'ordre : 16/2011-M/IN

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEURE ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENNE
FACULTE D'ELECTRONIQUE ET DINFORMATIQUE

MEMOIRE

Présenté pour l'obtention du diplôme de MAGISTER
En : Informatique
Spécialité : Systèmes Informatiques

Par **: HACHEMI Asma**

Sujet

COMPOSITION DE PATRONS LOGICIELS

Contributions à la Documentation Agile et à l'Analyse Automatique des Relations entre Patrons

Soutenu publiquement, le 08/12/2011, devant le jury composé de :

M^me Malika BOUKALA	Professeur à l'USTHB	Présidente
M^r Mohamed AHMED-NACER	Professeur à l'USTHB	Directeur de mémoire
M^me Zaia ALI MAZIGHI	Professeur à l'USTHB	Examinatrice
M^elle Latifa MAHDAOUI	MCA à l'USTHB	Examinatrice
M^elle Assia HACHICHI	MCB à l'USTHB	Invitée

Remerciements

Dieu soit loué, le tout puissant qui m'a guidée pour l'accomplissement de ce modeste travail. Je n'aurai jamais su me faire guider sans le guidage de Dieu.

Je tiens ensuite à exprimer mes remerciements à tous ceux qui m'ont aidée dans cette phase de post graduation. Parmi eux …

… Les membres du jury, pour avoir bien voulu faire l'évaluation de mon travail.

… Pr Mohamed AHMED-NACER mon directeur de thèse qui m'a offert une grande liberté de travail. J'ai apprécié sa vision scientifique ainsi que ses suggestions.

… Maman et Papa ainsi que mon frère et ma sœur pour leur soutien et amour. Je les aime de tout mon cœur.

Mes derniers remerciements sont pour chaque personne ayant prié ma réussite. Qu'elle reçoive à travers ces quelques lignes toute l'expression de ma sincère reconnaissance.

Asma HACHEMI

Table des matières

Liste des figures

Liste des tables

CHAPITRE :

INTRODUCTION GENERALE

Ce chapitre introduit de manière concise le travail présenté dans ce mémoire. Il commence par donner le contexte de notre travail **(cf. § 1)**. Ensuite, les objectifs que nous nous sommes fixés sont présentés **(cf. § 2).** Troisièmement, un aperçu de nos principales contributions est donné **(cf. § 3)**. Enfin, le plan de ce présent mémoire est exposé **(cf. § 4)**.

1. Contexte du présent travail

La dynamique du marché génère plus d'exigences sur le produit logiciel développé, notamment en termes de délais et de besoins changeants. Les méthodes Agiles [Manifeste01] sont des procédés de développement logiciel plus pragmatiques, qui offrent une formidable voie pour gérer des cycles de développement rapides, et traiter le changement avec efficacité [Pikkarainen05] [Highsmith04]. L'essor de ces méthodes souligne leur intérêt. Ces dernières ont gagné beaucoup de partisans et beaucoup d'entreprises les ont adoptées [Nerur07]. Les projets utilisant les méthodes Agiles sont en train de rapporter des améliorations dans le temps et le coût, par rapport à ceux utilisant des méthodes classiques [Charvat03]. Ainsi, l'importance des méthodes Agiles impose une attention particulière pour ces dernières, et des travaux spéciaux deviennent un impératif.

Par ailleurs, la documentation de logiciels pérennise des informations pertinentes couvrants tous les aspects du logiciel [Cook94] [Forward02]. Elle est partie prenante dans un procédé de développement, et est un composant clé dans la qualité du logiciel [Card87] [Lientz81] [Rombach87]. Donc, on ne peut s'en passer d'elle ni tolérer la dégradation de sa qualité, d'où l'importance de la documentation de logiciels.

Pourtant, cette documentation a des particularités dans les projets Agiles [Grosjean07b] : la quantité et les types de documents diffèrent d'un projet à un autre, et le besoin en documentation n'est pas connu à priori [Cockburn05]. Aussi, le logiciel est livré à chaque fin d'itération et ceci inclut la documentation qui l'accompagne. De plus, toute documentation rédigée doit évoluer dans le temps et être synchronisée avec le logiciel. Conséquemment, un soin particulier doit s'appliquer sur cette documentation dans les projets Agiles [Ambler09] [Aguiar09] [Grosjean07b]. Dans ce domaine, des problèmes récurrents restent posés. Certains sont inhérents à la documentation de logiciels [Komulainen06] [Lethbridge03] [Ruping03] [Kylmäkoski03] [Boehm03], et donc réapparaissent dans la documentation Agile, d'autres sont imposés par les contraintes des projets Agiles.

Un outil très puissant, qui peut offrir des solutions aux problèmes récurrents sont les patrons logiciels [Beck87] [Coplien94] [Ambler98] [Fowler02] [Alast03] [Fernandez07] [Vora09]. Un patron logiciel est un modèle représentant une solution prouvée, qui résout un problème récurrent dans un contexte donné. Il capture une connaissance dont il est avéré de rapporter des résultats efficaces. Il peut être instancié dans plusieurs situations, et peut être enrichi si nécessaire par des informations détaillées supplémentaires. Il est avantageux donc d'explorer des patrons, à la recherche d'une solution pour le problème d'exhaustivité de la documentation [Boehm03] dans les projets Agiles.

Le défi du travail présenté dans ce mémoire est de pénétrer dans deux domaines peu explorés mais très importants, à savoir la documentation dans les procédés Agiles et les patrons logiciels. L'idée derrière ce travail est avant tout de porter un focus sur des lacunes qui demeurent existantes dans ces domaines, et qu'il est intéressant d'adresser dans des travaux de recherche. Ainsi, ce travail n'est qu'un point de départ qui peut se prolonger vers plusieurs perspectives, dont quelques unes sont présentées à la fin de ce mémoire.

2. Problématiques

Les documents nombreux et volumineux produits lors d'un projet sont difficiles à gérer et à exploiter. Or, peu de documentation comme trop de documentation dégrade l'utilité de cette dernière. Il est donc indispensable d'avoir une quantité optimale de documents, qui permet de couvrir les besoins en documentation tout en étant facile à gérer et à exploiter [Boehm03]. Dés lors, ce problème d'exhaustivité de la documentation doit être résolu dans les projets Agiles, notamment que les méthodes Agiles ne présentent aucun procédé concret de documentation.

Ainsi, notre première problématique était d'explorer des catalogues de patrons logiciels à la recherche d'une solution au problème d'exhaustivité de la documentation Agile. Lors de cette exploration, le seul catalogue trouvé apte à fournir des patrons dans le sujet est le catalogue de Ruping [Ruping03] ; puisque c'est un catalogue spécialisé dans la documentation Agile, et qui fournit un grand nombre de patrons traitants des problèmes différents dans ce domaine. Donc, le catalogue de Ruping offre plus de chance de repérer un patron adéquat au problème en question.

Seulement, aucun patron parmi ceux de Ruping ne traite le problème d'exhaustivité de la documentation Agile dans sa totalité, il nous a fallut par conséquent composer une solution à partir de plusieurs patrons. Lorsque les patrons coopèrent pour résoudre un problème complexe, ils expriment pleinement leurs forces. Du coup, il est nécessaire de savoir quels patrons peuvent fonctionner ensemble et les patrons dépendants, pour pouvoir les composer et former une solution.

A partir de ce moment, nous avons observé une autre problématique et l'avons considérée comme seconde objectif de notre travail. Il s'agit de la composition de patrons, qui permet de résoudre des problèmes complexes non résolus par des patrons seuls. La composition permet aux patrons d'être de plus en plus utilisés, et donc donne un meilleur profit des connaissances prouvées procurées par les patrons. La composition de patrons se base principalement sur les relations entre patrons. Ces dernières sont généralement explicites dans les patrons ; le cas contraire, elles doivent être analysées et ressorties pour pouvoir servir la composition de patrons.

Plusieurs chercheurs ont abordé les relations entre patrons [Zimmer94] [Meszaros97] [Beck97] [Henney99] [Volter00], mais très peu ont tenté de faire ressortir ces relations dans la cas où elles ne sont pas explicites, ou lorsqu'elles concernent différents catalogues. Or, la composition de patrons sera mieux servi par une approche automatique, capable de gérer un nombre quelconque de patrons appartenant à des collections différentes. Kubo et al. sont les premiers chercheurs à présenter une approche automatique d'analyse des relations entre patrons de différents catalogues, pour permettre de les composer, et les seuls à proposer une approche qui peut traiter différents types de patrons logiciels [Kubo05] [Washizaki07].

Toutefois, cette méthode ne reconnait pas tous les types de relations, et ne peut traiter certains formalismes de patrons (ceux qui ne possèdent pas un élément nécessaire à la procédure d'analyse). Nous nous sommes donc fixés comme objectif l'amélioration de la

méthode de Kubo et al. sur deux plans : le premier pour reconnaitre plus de relations entre patrons (les relations Uses et Refines), et le second pour analyser plus de formalismes de patrons (les formalismes n'ayants pas une rubrique de contexte final).

3. Contributions

Notre première contribution concerne la proposition d'une solution au problème d'exhaustivité de la documentation Agile.

Nous avons ainsi conçu une solution à ce problème via la composition de patrons de Ruping. Pour ce faire, nous avons sélectionné trois patrons du catalogue de Ruping où chacun résout une partie du problème ; nous avons ensuite composé ces trois patrons en se basant sur les relations entre eux (particulièrement la relation Resulting-Starting qui se base sur les contextes initiaux et finaux des patrons), pour former une solution complète au problème.

Notre seconde contribution concerne un apport à la composition de patrons logiciels, et plus précisément l'amélioration d'une méthode d'analyse automatique de relations entre patrons sur deux volets ciblés :

Premièrement, nous avons proposé une analyse des relations Uses et Refines, en faisant appel à un auxiliaire qui est l'analyse de l'Inclusion. Cet auxiliaire nous permet de savoir si un élément d'un patron inclut un autre, et du coup nous permet d'analyser si un patrons utilise ou spécialise un autre.

Deuxièmement, nous avons permis l'analyse automatique des patrons dont les formalismes ne possèdent pas une rubrique de contexte final. En effet, nous avons proposé d'exploiter la rubrique solution pour représenter l'élément contexte final du patron, et avons montré que cette utilisation n'altère pas la signification des différentes relations analysées.

Enfin, nous avons implémenté le noyau de l'analyseur automatique des relations entre patrons, qui étant donné deux patrons sous le modèle de Kubo et al., calcule toutes les relations primaires entre ces patrons, et adopte la relation la plus puissante comme la relation à considérer entre ces patrons. Ce noyau nous a permis de faire des tests et de découvrir des relations entre patrons de même ou de différents catalogues, qui n'étaient pas citées par les auteurs des patrons.

4. Organisation du mémoire

Pour structurer ce mémoire, nous avons opté pour l'organisation suivante :

Chapitre I : Introduit les concepts qui seront exploités dans notre travail. Il aborde les patrons logiciels, donne une introduction au domaine de la documentation de logiciels et introduit la documentation Agile avec ses spécificités.

Chapitre II : Détaille dans un premier lieu le problème d'exhaustivité de la documentation dans les projets Agiles, et l'outil utilisé pour sa résolution. Ensuite la composition de patrons est abordée, et une méthode d'analyse automatique des relations entre patrons est choisie pour ce travail et est analysée.

Chapitre III : Expose les solutions aux problèmes traités par ce travail. Il présente dans la première partie une issue pour solutionner le problème d'exhaustivité de la documentation, en présentant entre autres les patrons utilisés dans solution. La seconde partie porte sur la solution du problème de composition automatique de patrons.

Chapitre Conclusion Générale : Dans ce chapitre final, un bilan des apports de notre travail est dressé et quelques perspectives qui pourront êtres poursuivies sont énumérées.

CHAPITRE I :

PATRONS ET DOCUMENTATION DE LOGICIELS : ETAT DE L'ART

Ce chapitre est consacré à introduire les concepts qui seront exploités dans notre travail. Il aborde dans un premier temps les patrons logiciels **(cf. § I.1)**. Dans un second lieu, une introduction au domaine de la documentation de logiciels est donnée **(cf. § I.2).** Troisièmement, la documentation Agile est introduite avec ses spécificités **(cf. § I.3)**. Enfin, une synthèse du chapitre est donnée en conclusion **(cf. § I.4)**.

I.1. Patrons logiciels

I.1.1. Préambule

Le développement logiciel ne cesse de gagner des niveaux de complexité sans précédents, et la concurrence rude impose une réaction rapide aux besoins du marché avec des systèmes sophistiqués et flexibles. La communauté du génie logiciel s'est rendu compte qu'une application est un produit manufacturé complexe, dont la réalisation doit s'intégrer dans une démarche méthodologique (le procédé de développement), et que pour améliorer la qualité du produit développé il est important de maîtriser son procédé de développement, dont la problématique s'apparente à celle du logiciel lui-même **[Montangero99] [Osterweil87] [Estublier05]**. Ainsi, tout outil aidant à réduire la complexité du monde de l'ingénierie des logiciels est une bénédiction.

Dans ce contexte, les patrons logiciels (software patterns) ont émergé comme un excellent outil. Il s'agit d'un concept destiné à résoudre les problèmes récurrents du domaine du génie logiciel, en apportant des solutions prouvées.

L'origine des patrons remonte aux années 70 avec les travaux de l'architecte Christopher Alexander, qui remarque que la phase de conception en architecture laisse apparaître des problèmes récurrents. Il cherche alors à résoudre l'ensemble de ces problèmes et établi une collection de 253 patrons **[Alexander77]**, qui couvrent tous les aspects de la construction.

La première fois où des patrons sont apparus dans le monde du génie logiciel fut en 1987, avec le fameux papier de Kent Beck et W. Cuningham intitulé Using Pattern Languages for Object-Oriented Programs **[Beck87]**. Ce travail décrit cinq patrons, discutant des problèmes de conception de fenêtres en Smalltalk.

La communauté logicielle a suivi l'exemple de Beck et Cuningham, et des patrons ont commencé à apparaitre et à investir plusieurs domaines. Parmi les premiers travaux fut celui des patrons organisationnels de Coplien **[Coplien94]**. Mais le travail le plus célèbre dans le monde des patrons logiciels, est bien l'ouvrage publié en 1995 par le Gang of Four sur les patrons de conception (Design Patterns). Cette équipe qui regroupe Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides intitule son livre Design Patterns : Elements of Reusable Object-Oriented Software. Celui-ci présente 23 patrons de conception qui font aujourd'hui référence dans le monde de l'informatique. Un patron de conception est une solution à un problème récurrent dans la conception d'applications orientées objet. Il décrit alors la solution prouvée pour résoudre ce problème d'architecture de logiciel, indépendamment des langages de programmation utilisés.

Une fois familier avec les patrons de conception, on est capable d'inférer ce qu'est un patron de procédé à partir de son nom. Le terme patron de procédé (Process Pattern) a été introduit la première fois par Coplien **[Coplien94]** dans le sens d'un patron qui capture des connaissances de procédés. Alors que les patrons de conception représentent une issue aux problèmes communs et récurrents dans la phase de conception des logiciels, les patrons de procédés

montrent comment accomplir efficacement les travaux quelque soit la phase du développement logiciel. Les patrons de conception s'intéressent au produit de la conception (le résultat) et non pas au processus de conception qui est du domaine des patrons de procédé. Ainsi, un patron de procédé représente une approche structurée d'un procédé dont il est prouvé de rapporter des résultats efficaces. Parmi les premiers patrons qui ont émergé dans ce domaine, il y a les patrons de procédés d'Ambler **[Ambler98] [Ambler98a] [Ambler99]**.

La communauté du logiciel s'est rendu compte de la puissance des patrons, et d'autres types de patrons ont continué à émerger, couvrant des aspects très divers touchant de prés ou de loin au développement logiciel, tels que les patrons de logiciels d'entreprises **[Fowler02]**, les patrons de workflow **[Alast03]**, les patrons de sécurité informatique **[Fernandez07],** les patrons d'applications web **[Vora09]**… Et surement, la communauté du logiciel ne cessera de voir fleurir des patrons.

L'utilisation des patrons offre de nombreux avantages :

Tout d'abord, les patrons permettent de mettre en avant les bonnes pratiques, en répondant à un problème via une solution prouvée et validée par des experts. Ainsi on gagne en rapidité, en qualité et on diminue également les coûts.

De plus, l'abstraction du problème et de la solution dans un patron le rend réutilisable et capable de s'appliquer dans divers cas de figure, puisque il pourra être instancié dans plusieurs situations et pourra être enrichi si nécessaire par des informations détaillées supplémentaires.

Aussi, les patrons (notamment les patrons de conception) étant largement documentés et connus d'un grand nombre de spécialistes, ils permettent également de faciliter la communication (si un développeur annonce que sur un point du projet il utilise le patron de conception *Observer* par exemple, il est compris des informaticiens sans pour autant rentrer dans les détails de la conception).

Enfin, de même que la qualité d'un code source peut facilement être jugée par un regard rapide aux normes de codage qui ont été suivies, la qualité d'une solution peut maintenant être estimée d'après les patrons utilisés.

I.1.2. Modélisation des patrons

Dans cet axe de recherche, deux problèmes sont distingués : la proposition d'un formalisme pour représenter les patrons logiciels ; et le développement d'un langage de description de patrons logiciels.

Un formalisme est une structure syntaxique qui comporte un ensemble de rubriques, chaque rubrique représente une information particulière du patron. Les formalismes de description de patrons sont généralement tous basés sur le triplet (problème, contexte, solution) pour exprimer respectivement un certain problème récurrent, le contexte d'application du patron et

une solution pour ce problème. Ces rubriques représentent une structuration du contenu des patrons.

Le besoin d'un langage formel pour modéliser les patrons logiciels émerge de plus en plus. Un tel langage devrait préciser la sémantique des concepts qu'il véhicule et fournir les notations graphiques. La notation graphique permet d'exprimer visuellement une solution, ce qui facilite la comparaison et l'évaluation de cette dernière. L'aspect formel de la notation limite les ambiguïtés et les incompréhensions. Un tel langage devrait être basé sur un méta modèle qui décrit de manière précise tous les éléments de modélisation du patron et la sémantique de ces éléments.

I.1.2.1. Formalisme d'Ambler

Le formalisme d'Ambler **[Ambler98]** comporte les six rubriques suivantes:

- **Nom** : nom qui décrit brièvement le patron
- **Type** : indique si le patron décrit est de type Tâche, Phase ou Etape (puisque Ambler a sa propre collection de patrons de procédé de différents types, Tâche, Phase ou Etape).
- **Forces** : buts à atteindre et contraintes à respecter en appliquant ce patron
- **Contexte initial** : conditions nécessaires pour déclencher l'application du patron
- **Solution** : décrit en détail comment exécuter les étapes ou les actions du patron pour résoudre le problème
- **Contexte résultant** : conditions qui doivent être satisfaites après l'application de ce patron

La figure suivante montre un patron sous le formalisme d'Ambler :

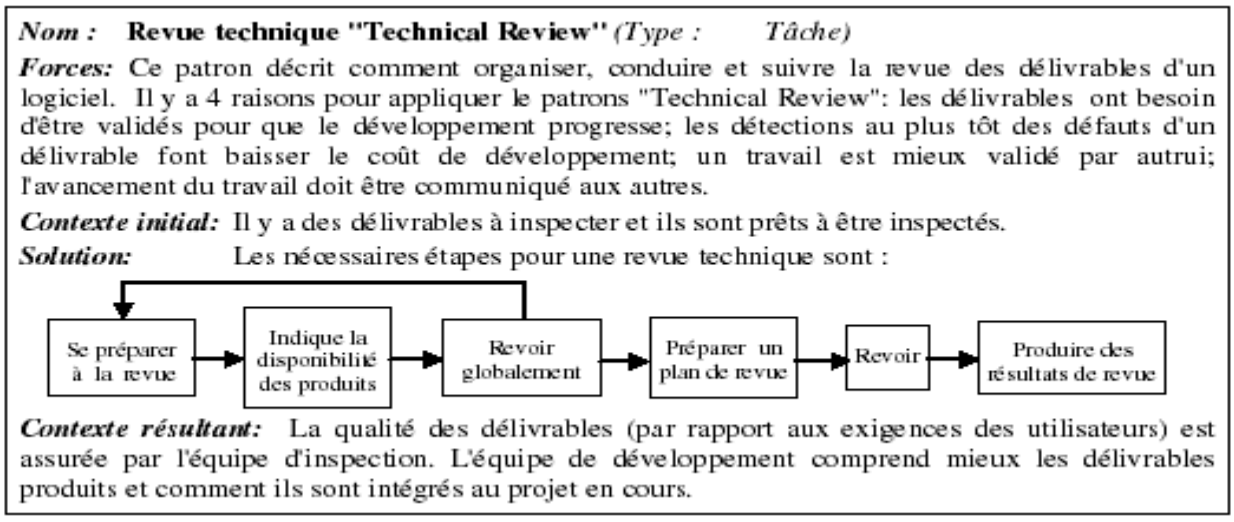


Figure I.1. Un patron sous le formalisme d'Ambler **[Ambler98]**

I.1.2.2. Formalisme d'Iida

Le formalisme d'Iida **[Iida99]** comporte les six rubriques suivantes :

- **Problem**: le problème à résoudre en utilisant le patron.
- **Forces**: restrictions et facteurs additionnels justifiant l'utilité du patron.

- **Context**: décrit la situation dans laquelle le patron est applicable.
- **Resulting Context**: décrit la situation à établir en appliquant le patron.
- **Description**: description de la solution en langage naturel.
- **Remarks**: commentaires incluant des conseils, des justifications, des patrons en relation avec le sujet traité …

La figure suivante montre un patron sous le formalisme d'Iida :

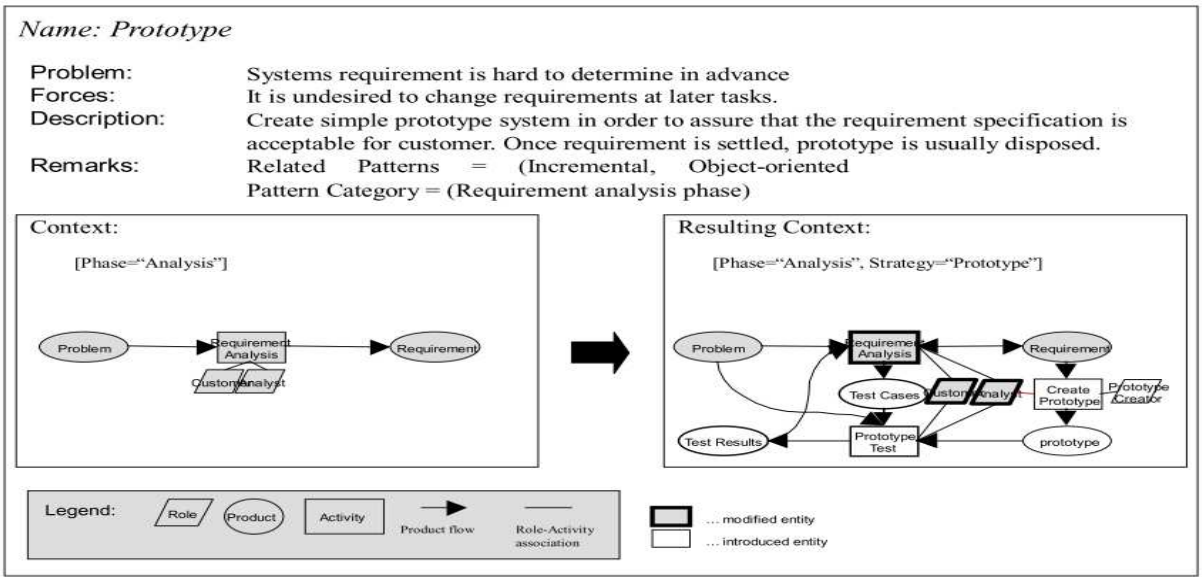


Figure I.2. Un patron sous le formalisme d'Iida **[Iida99]**

I.1.2.3. Formalisme de XDP

Dans **[XmlPatterns00]** un formalisme de patrons est proposé incluant les rubriques suivantes:

- **Name** : nom du patron.
- **Abstract** : une brève description du patron qui ne dépasse pas quelques phrases.
- **Problem** : description du problème à résoudre.
- **Context** : contexte dans lequel apparait le problème, car certains problèmes apparaissent uniquement dans des contextes bien précis.
- **Forces** : description des éléments qui influent sur le problème. Ce sont les contraintes qui affectent la solution et que cette dernière tente de régler.
- **Solution** : la manière de résoudre le problème.
- **Examples** : exemple(s) montrant comment appliquer le patron.
- **Discussion** : explication pourquoi la solution fonctionne et pourquoi elle ne peux fonctionner dans certains cas.
- **Related Patterns** : liste de patrons liés.
- **Known Uses** : description de cas réels où le patron est appliqué.

I.1.2.4. Formalisme de RHODES

Dans le projet RHODES **[Coulette00]**, Coulette et al. ont proposé un formalisme pour représenter les patrons de procédé **[Tran01]** contenant six rubriques :

- **Nom** : nom qui identifie le problème de conception de procédé.
- **Intention** : description du problème adressé par le patron
- **Contexte**: situations où on peut utiliser le patron.
- **Solution semi-formelle** : décrit en formalisme UML étendu le procédé à exécuter pour résoudre le problème.
- **Solution formelle**: solution exécutable en PBOOL+ **[Crégut97]** qui peut être réutilisée dans plusieurs contextes.
- **Application** : remarques, guidages dans l'application du patron.

La figure suivante montre l'exemple d'un patron dans le formalisme de RHODES :

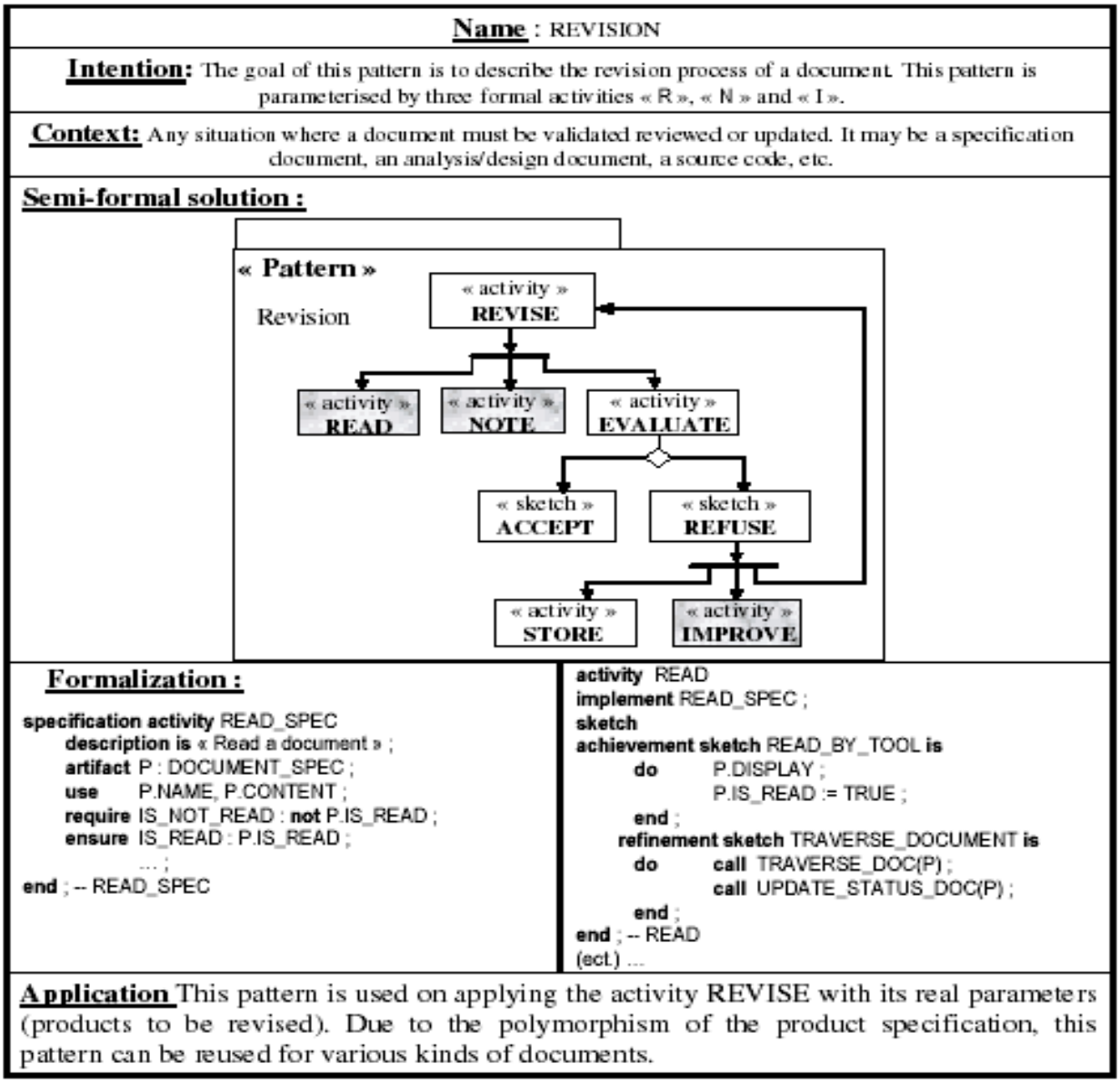


Figure I.3. Un patron sous le formalisme de RHODES **[Tran01]**

I.1.2.5. Formalisme de PROMENADE

Le formalisme proposé dans PROMENADE **[Ribo00] [Ribo02]** est composé des onze rubriques suivantes :

• **Nom** : identifie le problème
• **Auteur** : nom de l'auteur du patron
• **Synonymes** : autres noms du patron s'ils existent
• **Mot clés** : pour décrire en bref le patron
• **Intention** : description du problème adressé par le patron
• **Paramètres** : paramètres du patron qui facilitent son adaptation
• **Contexte initial** : conditions nécessaires pour déclencher l'application du patron
• **Contexte résultant** : conditions qui doivent être satisfaites après la fin de l'application de ce patron
• **Applicabilité** : contraintes du le patron
• **Participants** : rôles intervenants dans la solution de ce patron
• **Procédé** : décrit en détail les étapes ou les actions du patron pour résoudre le problème.

La figure suivante montre un patron dans le formalisme de PROMENADE :

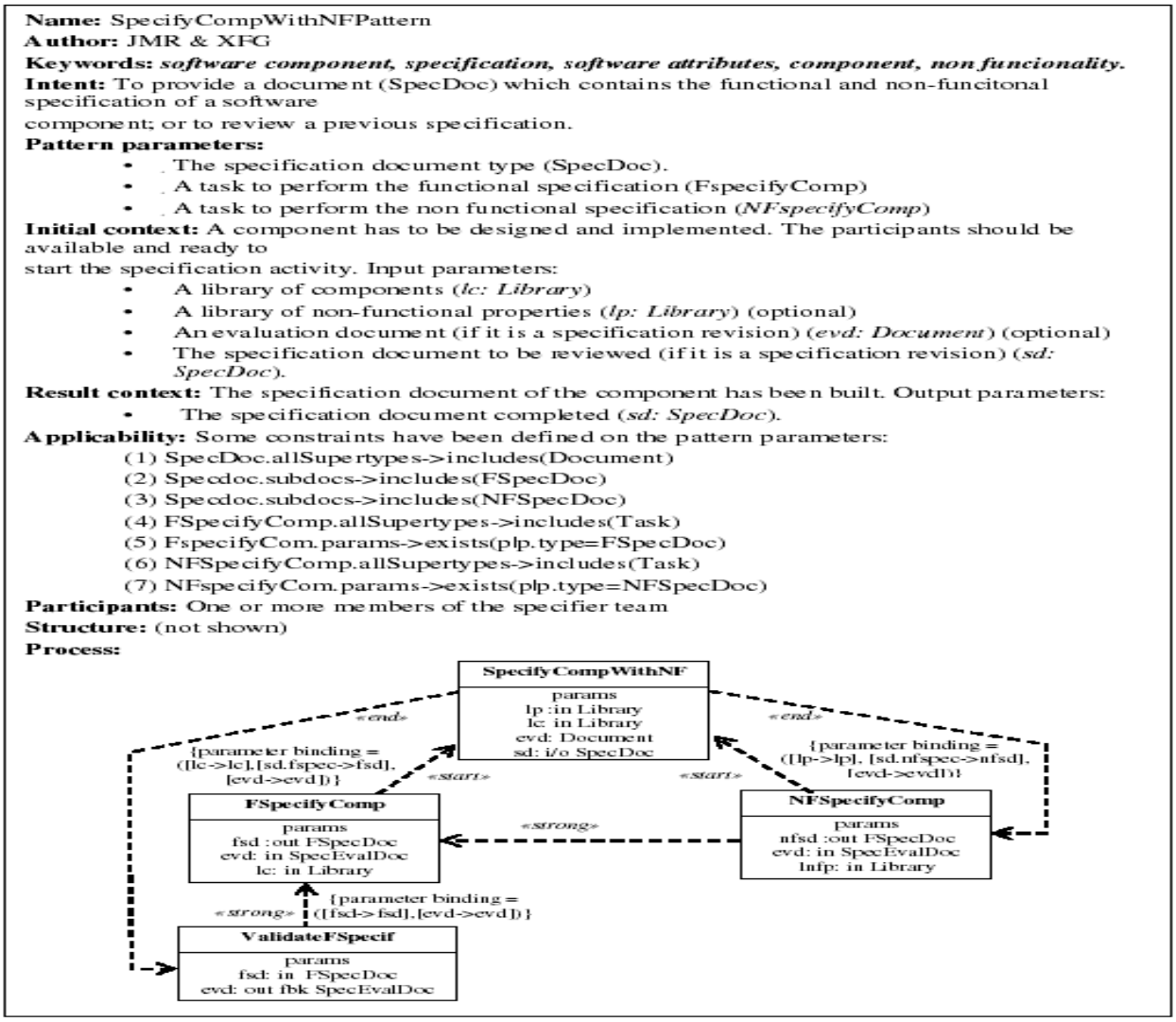


Figure I.4. Un patron sous le formalisme de PROMENADE **[Ribo02]**

I.1.2.6. Formalisme P-Sigma

Le formalisme P-Sigma **[Conte01]** a pour objectifs :

- d'exprimer une sémantique commune à la majorité des formalismes proposés dans la littérature.
- d'expliciter l'interface de sélection des patrons afin de faciliter leur réutilisation.
- de proposer des relations inter-patrons afin d'améliorer l'organisation des ensembles de patrons.

Le formalisme P-Sigma est constitué de trois grandes parties qui couvrent les 13 rubriques :

1. Interface :
   - **Identifiant:** définit le couple problème/solution à partir duquel le patron pourra être référencé.
   - **Classification:** définit la fonction du patron par un ensemble de mots-clés du domaine.
   - **Contexte :** décrit la pré condition pour appliquer le patron.
   - **Problème:** définit le problème résolu par le patron.
   - **Force :** définit les apports attendus de l'application du patron. Cette rubrique apporte une valeur complémentaire au problème.
2. Réalisation :
   - **Solution Modèle :** décrit la solution en terme des produits attendus après l'application du patron.
   - **Solution Démarche :** indique la solution du problème en terme de processus à suivre.
   - **Cas d'application :** décrit des exemples d'imitation du patron.
   - **Conséquence d'application :** présente les limites et les bénéfices de l'application du patron.
3. Relation :

Cette partie est composée de quatre rubriques correspondant aux quatre types de relations possibles entre les patrons : **utilise, raffine, requiert** et **alternative.**
Chaque relation est exprimée par une rubrique donnant l'ensemble des patrons qui sont liés au patron en question par la relation concernée.

La figure suivante montre une partie d'un patron sous le formalisme de P-Sigma :

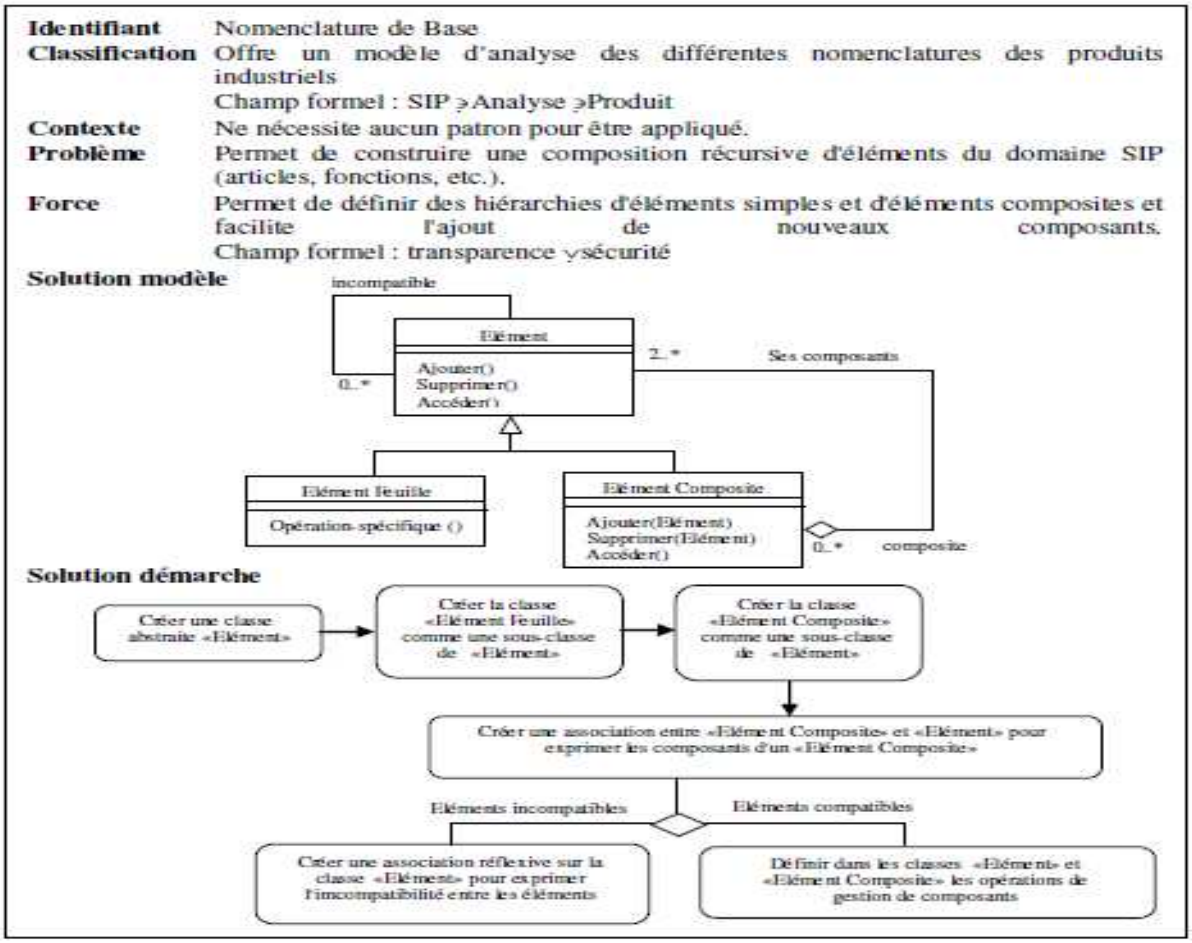


Figure I.5. Une partie d'un patron sous le formalisme de P-Sigma **[Conte01]**

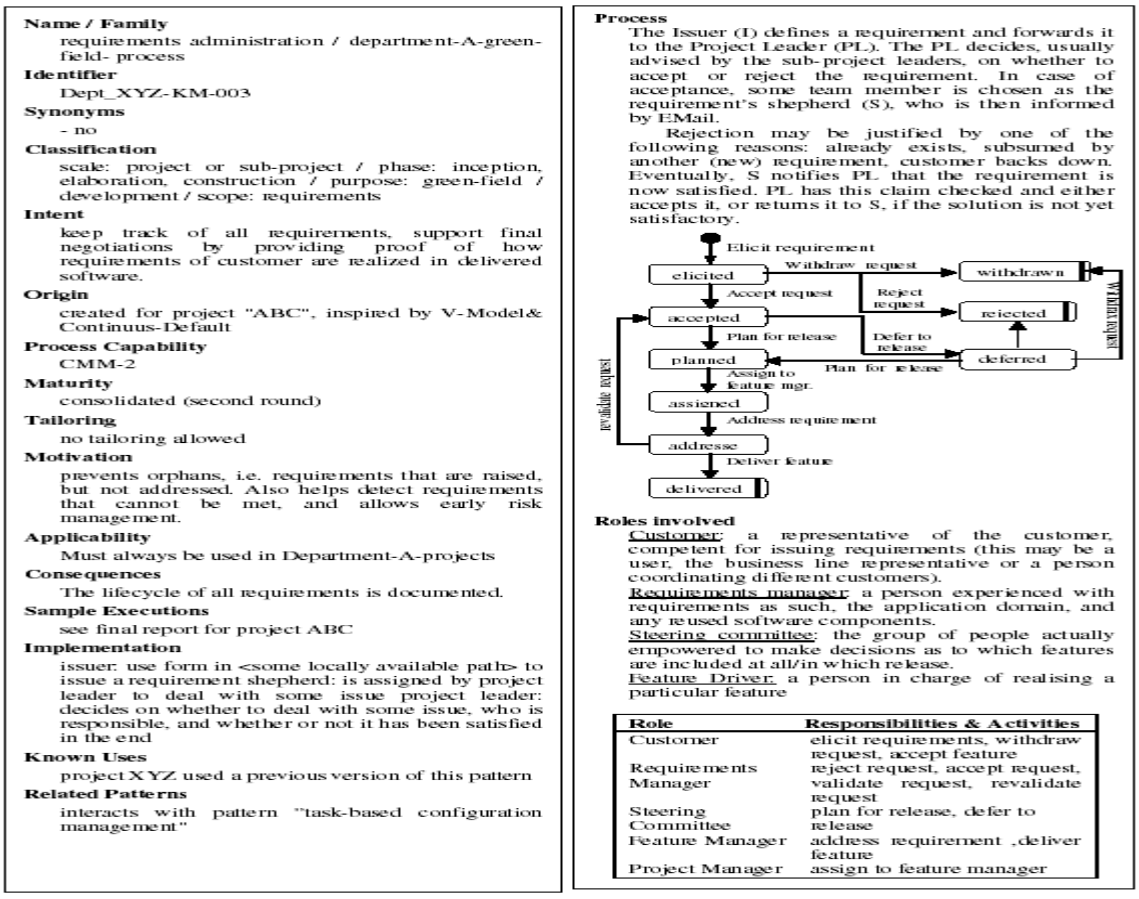
I.1.2.7. Formalisme de Störrle

Storrle **[Storrle01]** a adapté le formalisme de patron de conception du Gang of Four aux patrons de procédé comme suit :

- **Nom :** nom qui décrit brièvement le patron. Le nom du patron est souvent le nom de la tâche à faire ou le produit créé par les activités de ce patron.
- **Identité :** identité unique pour identifier ce patron même sous les changements de nom.
- **Synonymes :** autres noms du patron s'ils existent.
- **Origine :** nom de la source ou de l'auteur du patron.
- **Classification :** classifie le patron selon les aspects suivants : *échelle* déterminé par la taille d'organisation qui vise à appliquer ce patron; *phase de développement* où ce patron peut être appliqué; *but* c'est le type de projet qui peut appliquer ce patron; *portée* le domaine adressé par le patron.
- **Intention :** les justifications, les bénéfices du patron et ses domaines d'application.
- **Capacité du processus :** (Process capability) niveau optimal de capacité du processus auquel ce patron peut être appliqué.
- **Maturité :** niveau de maturité du patron, par exemple : expérimental, consolidé, stable, etc.

- **Adaptation :** description des situations où l'adaptation de ce patron peut être nécessaire.
- **Motivation :** un scénario qui illustre le problème traité dans ce patron, son applicabilité et son objectif.
- **Applicabilité :** pré requis pour appliquer le patron. Contexte où le patron peut être appliqué.
- **Processus :** activités à suivre pour résoudre le problème. Un langage informel ou formel est utilisé pour décrire le procédé.
- **Participants :** rôles intervenants dans les activités exécutées par ce patron.
- **Conséquences :** discussion des avantages et des inconvénients de l'utilisation du patron. Idéalement, les conséquences devraient être quantifiables par métriques.
- **Exemple d'exécution :** descriptions des exécutions concrètes de ce patron, y compris un détail des artefacts utilisés et créés par ce patron.
- **Implémentation :** discussion des problèmes concernant la mise en oeuvre de ce patron.
- **Usages connus :** référence à des projets où ce patron a été appliqué.
- **Patrons reliés :** autres patrons de procédé dans le même catalogue qui peuvent être soit utilisés conjointement avec ce patron, soit des patrons alternatifs, soit des patrons requis ou conséquences de ce patron.

La figure suivante montre un exemple illustrant le formalisme de Störrle.



Name / Family
requirements administration / department-A-green-field- process
Identifier
Dept_XYZ-KM-003
Synonyms
- no
Classification
scale: project or sub-project / phase: inception, elaboration, construction / purpose: green-field / development / scope: requirements
Intent
keep track of all requirements, support final negotiations by providing proof of how requirements of customer are realized in delivered software.
Origin
created for project "ABC", inspired by V-Model& Continuus-Default
Process Capability
CMM-2
Maturity
consolidated (second round)
Tailoring
no tailoring allowed
Motivation
prevents orphans, i.e. requirements that are raised, but not addressed. Also helps detect requirements that cannot be met, and allows early risk management.
Applicability
Must always be used in Department-A-projects
Consequences
The lifecycle of all requirements is documented.
Sample Executions
see final report for project ABC
Implementation
issuer: use form in <some locally available path> to issue a requirement shepherd: is assigned by project leader to deal with some issue project leader: decides on whether to deal with some issue, who is responsible, and whether or not it has been satisfied in the end
Known Uses
project XYZ used a previous version of this pattern
Related Patterns
interacts with pattern "task-based configuration management"

Process
The Issuer (I) defines a requirement and forwards it to the Project Leader (PL). The PL decides, usually advised by the sub-project leaders, on whether to accept or reject the requirement. In case of acceptance, some team member is chosen as the requirement's shepherd (S), who is then informed by EMail.
Rejection may be justified by one of the following reasons: already exists, subsumed by another (new) requirement, customer backs down. Eventually, S notifies PL that the requirement is now satisfied. PL has this claim checked and either accepts it, or returns it to S, if the solution is not yet satisfactory.

Elicit requirement
elicited
Withdraw request
withdrawn
Accept request
Reject request
rejected
accepted
Plan for release
Defer to release
planned
deferred
Plan for release
Assign to feature mgr.
assigned
Address requirement
addressed
Deliver feature
delivered
invalidate request
withdraw request

Roles involved
Customer: a representative of the customer, competent for issuing requirements (this may be a user, the business line representative or a person coordinating different customers).
Requirements manager: a person experienced with requirements as such, the application domain, and any reused software components.
Steering committee: the group of people actually empowered to make decisions as to which features are included at all/in which release.
Feature Driver: a person in charge of realising a particular feature

Role	Responsibilities & Activities
Customer	elicit requirements, withdraw request, accept feature
Requirements Manager	reject request, accept request, validate request, revalidate request
Steering Committee	plan for release, defer to release
Feature Manager	address requirement ,deliver feature
Project Manager	assign to feature manager

Figure I.6. Un patron sous le formalisme de Störrle **[Storrle03]**

I.1.2.8. Formalisme de Gnatz

Gnatz et al. **[Gnatz02] [Gnatz03]** ont proposé un formalisme composé des rubriques suivantes:

- **Nom** : nom qui décrit brièvement le patron
- **Aussi connu comme** : autres noms du patron
- **Auteur:** nom de l'auteur du patron
- **Intention:** sommaire concis de l'intention et du raisonnement du patron.
- **Problème:** problème que le patron vise à résoudre et éventuellement un exemple réel pour illustrer ce problème.
- **Contexte initial:** la situation où le patron peut être appliqué.
- **Solution:** suggère des activités à suivre pour résoudre le problème. La solution décrit également les liens entre les contextes initiaux et les contextes résultant des activités exécutées. La solution peut être décrite informellement ou formellement.
- **Activités réalisées** : les noms des activités dont ce patron fournit un schéma de réalisation.
- **Contexte résultant:** l'état résultant des produits après l'application du patron.
- **Discussions:** les apports du patron et les problèmes générés en appliquant ce patron.
- **Usages connus:** des exemples d'utilisation du patron dans des projets réels. Ces exemples illustrent l'acceptation et l'utilité du patron, et peuvent fournir les guidages, les astuces et les techniques utiles pour appliquer le patron. Ils mentionnent également des échecs de l'application du patron.
- **Patrons reliés**: les références aux patrons qui résolvent des problèmes semblables, et les références aux patrons qui raffinent ce patron.

La figure suivante montre un patron sous le formalisme de Gnatz :

Name	BPM Task Analysis with Activity Diagrams
Intent	Development of: - a precise and unambiguous documentation of a BPM - documentation of BPM on different levels of abstraction cover not only all details but also provide an overview of relevant business processes. - documentation of BPM such that it can be understood by business experts as well as software developers - ensured adequacy of BPM (validated model)
Problem	Business experts are available but a precise documentation of as-is and to-be business processes being relevant for the system to be developed does not exist.
Solution	In order to achieve a precise and unambiguous documentation of business processes use UML activity diagrams with its formal syntax to describe business processes. In order to achieve a documentation of different layers of abstraction apply the principle of stepwise refinement: – Start with the definition of major tasks and refine them iteratively. Ensure adequacy of the business process model by reviewing each iteration of the model with (third party) experts. Involve business and software architecture experts being fluent with activity diagrams. – After having identified major tasks and user classes assign user classes as actors to tasks. Refine task characteristics of major tasks, and define a first version of the tasks' task chains by decomposing it into sub-tasks, and defining their causal dependencies. **Consider alternative chains:** – Review this first model involving persons representing the identified user classes in the review. – Perform the refinement steps of tasks iteratively and review each iteration (apply pattern Refinement of Activity Diagrams).
Realized Activities	Business Process Modeling
Initial Context	Initial Customer Specification with arbitrary modeling concepts and notations
Result Context	Business Process Model with UML activity diagrams as notation for task chains and a pre- and post-condition style specification of tasks.
Pros and Cons Pros:	– UML Activity Diagrams provide a standardized, concise and unambiguous notation to document business processes (Task Chains). The applied modeling concepts are widely used by business experts (e.g. similarity with Event Driven Process Chains) [24] as well as software developers (e.g. similarity with Petri nets). – This precise way of description supports detailed and precise review.

Figure I.7. Un patron sous le formalisme de Gnatz **[Gnatz03]**

I.1.2.9. Formalisme de Demeyer

Dans leur ouvrage **[Demeyer03]** Demeyer et al. ont présenté le formalisme suivant constitué de sept rubriques :

- **Name** : nom du patron. Cette rubrique inclut une sous rubrique **Intent** qui capture l'essentiel du patron.
- **Problem** : le sujet traité par le patron sous forme de question. Cette rubrique inclut les contraintes influençant le problème, et parfois elle décrit le contexte explicitement.
- **Solution** : la solution au problème posé.
- **Tradeoffs** : avantages et inconvénients de l'application du patron.
- **Rationale** : justifications de la solution.
- **Known uses** : liste quelques utilisations documentées du patron.
- **Related patterns** : liste les patrons alternatifs ou complémentaires.

La figure suivante montre l'exemple d'un patron dans le formalisme de Demeyer :

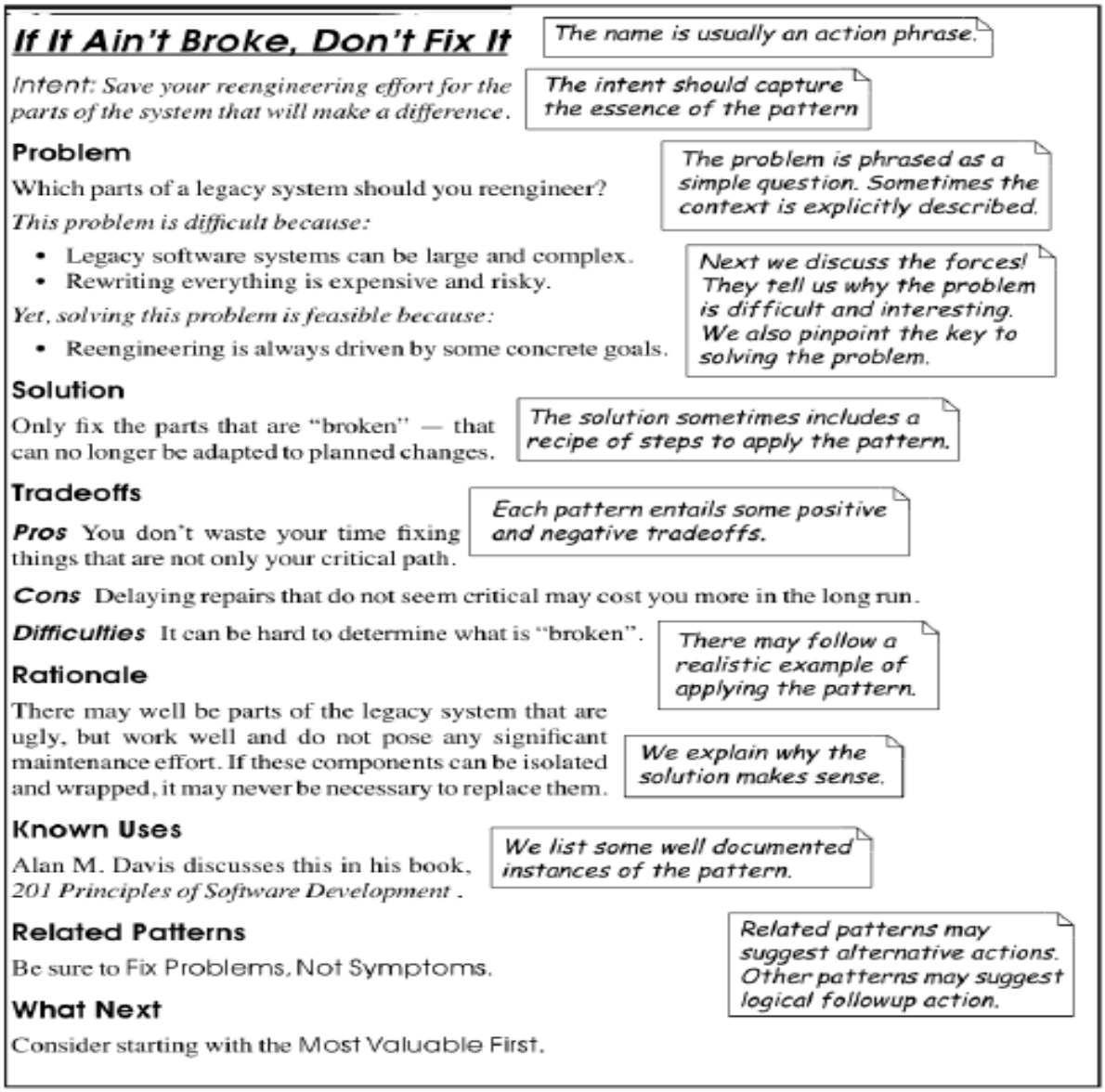


Figure I.8. Un patron sous le formalisme de Demeyer **[Demeyer03]**

I.1.2.10. Formalisme de Crumlish

Le formalisme proposé par Crumlish dans **[Crumlish09],** est constitué des rubriques suivantes en plus du nom du patron :

- **What** : le problème traité par le patron.
- **When** : contexte dans lequel le problème est pose.
- **How** : comment résoudre le problème (la solution).
- **Why** : les contraintes influençant la solution.
- **Related patterns** : les patrons lies.
- **As seen on** : exemples d'application du patron.
- **Sources / Similar patterns in other libraries** : cas particulier des patrons lies, ceux qui sont similaires au patron en question ou le patron source ayant inspiré le patron en question.

I.1.2.11. Langage PROPEL

PROPEL **[Dittmann02] [Hagen04a] [Hagen04b]** est un langage de description de patron de procédé (Process Pattern Description Language PPDL) basé sur UML. Ce langage permet de représenter les procédés de manière semi-formelle.

Un procédé dans PROPEL est modélisé comme un graphe d'activité UML. Il décrit le déroulement d'un ensemble d'activités et le flux d'objets entre ces activités. Un rôle (Role) est responsable de l'exécution du procédé.

Dans PROPEL, un patron de procédé (ProcessPattern) propose un procédé prouvé (Process), pour résoudre un problème de développement logiciel récurrent (Problem), dans un contexte spécifique (Context).
Un patron de procédé est un composant de l'agrégat ProcessPatternCatalog, c'est-à-dire qu'un patron de procédé est toujours assigné à un catalogue de patrons par l'association catalogue (Catalog).

Un contexte définit les conditions qui doivent être vérifiées avant et après l'application du patron, il est décrit par un ensemble d'objets ObjectFlowState. Un patron ainsi qu'un problème sont caractérisés par un contexte d'entrée et un contexte de sortie.
Le procédé proposé par un patron contient plusieurs activités. A chaque activité correspond un problème (ActivityProblemMapping).

Les extensions de la sémantique et de la syntaxe d'UML faites par PROPEL pour décrire les patrons de procédé, sont organisées en trois paquetages : Process Pattern, Relationship et ProcessPattern Graph.

- Le paquetage **ProcessPattern** décrit les relations entre les concepts de patron de procédé, catalogue de patrons, procédé, problème et contexte. Il est montré dans la figure suivante :

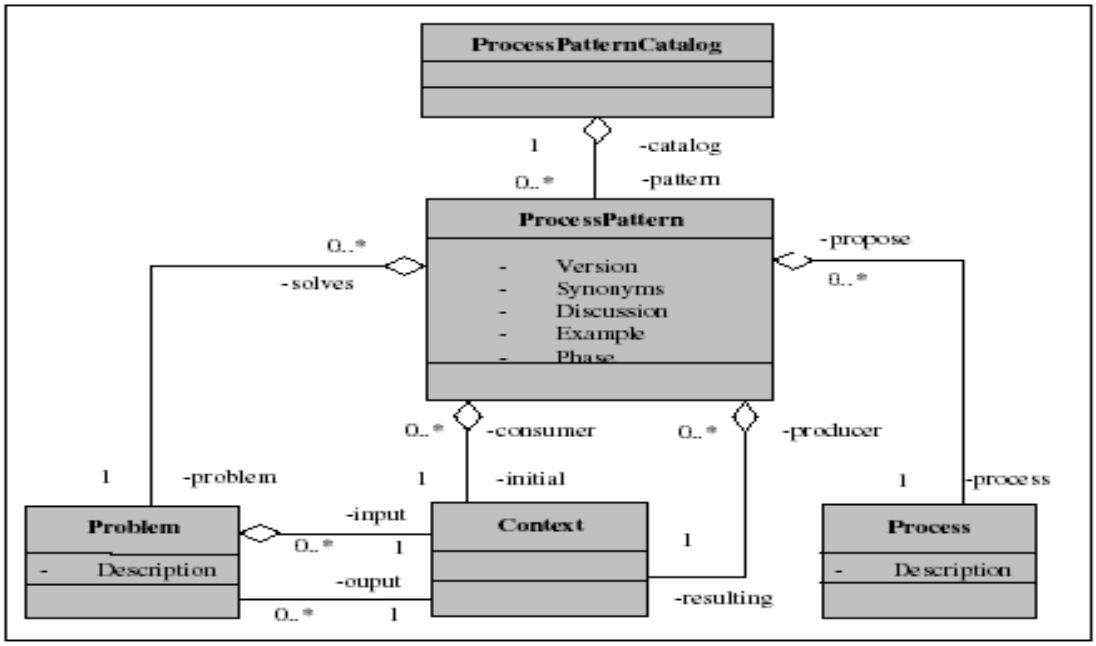


Figure I.10. Le paquetage ProcessPattern de PROPEL **[Hagen04b]**

- Le paquetage **Relationship** décrit les relations entre des ProcessPatterns.

- Le paquetage **ProcessPattern Graph** fournit les trois types suivants de diagrammes pour décrire les différents aspects d'un patron de procédé :
  Diagramme de Problème : représente le problème que le patron vise à résoudre, son entrée, sa sortie. **Diagramme de Procédé** : représente une démarche à suivre pour résoudre le problème du patron. **Diagramme d'Usage** : indique les patrons reliés au patron représenté. Ce diagramme permet de représenter les inter-relations entre patrons.
  D'après Hagent et al. **[Hagen04c]**, le choix des patrons est généralement fait via deux étapes :
  - Si un utilisateur fait face à un problème, d'abord il recherche ce problème en utilisant le diagramme de Problèmes. Ce dernier indique (peut être) plusieurs patrons qui résolvent ce problème.
  - Dans la deuxième étape, l'utilisateur peut examiner et choisir un de ces patrons. Il peut alors utiliser le diagramme d'Usage pour savoir quels sont les autres patrons qui devraient être appliqués avant ou après ce patron, ou quels sont les variantes ou les raffinements disponibles de ce patron.

  Enfin, pour exploiter le patron choisi, utiliser le diagramme de procédé.

I.1.2.12. Le méta-modèle UML-PP

Le méta-modèle UML for Process Patterns (UML-PP) **[TRAN07]** basé sur UML, permet d'une part de décrire les éléments principaux des procédés logiciels, et d'autre part de représenter les procédés basés sur la réutilisation de patrons de procédé.

UML-PP permet de décrire un langage de description de procédé (LDP) qui répond aux objectifs suivants :

- Le LDP devra prendre en compte les langages de modélisation existants, en particulier ceux qui sont standardisés, pour faciliter sa diffusion et son alignement avec les autres LDP.
- Le LDP devra être suffisamment formel pour faciliter son support par des outils.
- Le LDP devra être aussi simple et compréhensible que possible, pour permettre aux concepteurs de modéliser les procédés aisément et rapidement.

UML-PP est organisé en trois paquetages :

Le paquetage ProcessStructure décrit les éléments constituant un procédé ; ce paquetage contient les concepts qui définissent les éléments de procédé (*ProcessElement*) et les relations entre eux (*ProcessRelation*). Un procédé est décrit à l'aide de trois types d'éléments principaux : les tâches à réaliser (*Task*), les rôles participants (*Role*) et les produits manipulés *(Product*). Les relations entre éléments de procédés définies dans ce paquetage sont : Aggregation, Refinement, Task Parameter, Product Impact, Task Performance, Task Precedence, Product Responsibility.

Le paquetage ProcessPattern décrit la structure d'un patron de procédé (problème, contexte, solution).

Le paquetage PatternRelationship décrit les relations d'application et les relations d'organisation de patrons de procédé. Pour exprimer l'application on propose les relations *ProcessPatternBinding* et *ProcessPatternApplying* qui supportent respectivement la définition d'un élément de procédé et la réorganisation / enrichissement d'un groupe d'éléments de procédé. Pour exprimer l'organisation on propose les relations : *PatternUse (*existe entre deux patrons si le problème d'un patron *component* est un sous-problème de celui de l'autre patron *composite), PatternRefinement* (existe entre deux patrons si le problème d'un patron *refinedPattern* est un raffinement de celui d'un autre *orginalPattern*) et *PatternVariation* (associe deux patrons (*variant1* et *variant2*) s'ils résolvent un même problème en utilisant différentes solutions).

La figure suivante montre un extrait d'UMP-PP :

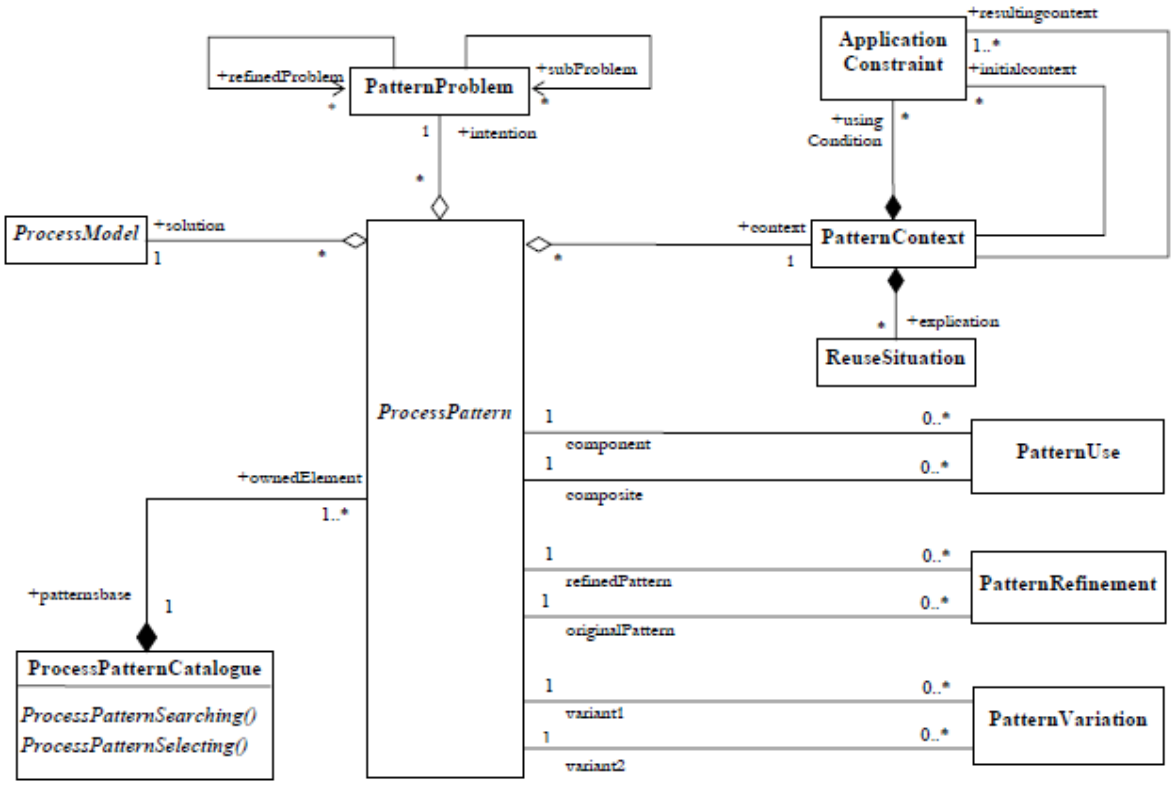


***Figure I.11. Extrait du méta-modèle UML-PP* [TRAN07]**

I.1.3. Identification des patrons

L'identification de patrons constitue un très important axe de recherche sur les patrons logiciels, surtout dans l'industrie. Elle consiste d'abord à identifier les sources de connaissances susceptibles de contenir des éléments (des problèmes) récurrents, ensuite identifier ces éléments (les problèmes que les patrons vont résoudre) pour enfin spécifier les solutions offertes par les patrons **[Gzara00]**.

Les travaux dans cet axe de recherche tentent de proposer des ensembles de patrons extraits d'expériences réelles et prouvées, pour fournir à leurs exploitants des solutions efficaces.
Pour représenter de tels ensembles, on emploi souvent le terme *catalogue* de patrons, qui désigne un groupement de patrons répondants à une problématique commune. D'autres termes sont aussi utilisés pour désigner de tels ensembles comme *collection, famille, système* ou *langage*.

Quelques collections de patrons logiciels sont données dans ce qui suit.

I.1.3.1. Le catalogue de patrons de Coplien

Le catalogue de Coplien **[Coplien94]** est une collection de 38 patrons, basée sur son expérience dans les procédés de développement et l'analyse organisationnelle. Ces patrons peuvent être utilisés pour construire une organisation, et de guider son procédé de

développement logiciel. Mais aussi, pour améliorer les procédés de développement logiciels au sein d'une organisation qui présente des faiblesses.

Les patrons de Coplien ont le réputation d'être des patrons d'organisation, malgré qu'ils offrent beaucoup de connaissance de procédés, et qu'ils s'intéressent à différents aspects organisationnels tels que : les procédés de développement dans une organisation, les techniques émergentes de la communauté logicielle, les techniques de gestion, la structure approprié à une organisation.

Le premier travail qui s'est intéressé aux classement des patrons de Coplien est **[Kaul97]**, qui a proposé une taxonomie selon trois dimensions :

- Les *patrons de structure* décrivent les structures techniques et humaines d'une organisation pour gérer le développement.
- Les *patrons de procédé* quant à eux décrivent les démarches, les flux d'information et les communications liées au développement.
- Les *patrons de comportement* décrivent la délégation des rôles dans une organisation.

I.1.3.2. La collection de patrons I-SPI

La collection I-SPI (**I**nitiating **S**oftware **P**rocess **I**mprovement) **[Appleton97]** est un ensemble de patrons s'intéressants à l'amélioration des procédés logiciels. Ces patrons représentent les meilleures pratiques observées par l'auteur de la collection, à travers les divers projets d'amélioration auxquels il a eu part.

L'amélioration d'un procédé affecte tout son écosystème. Ainsi, l'auteur a regroupé les patrons de sa collection en deux catégories :

- **Organization Patterns** : regroupe les patrons traitant l'aspect organisationnel du projet (Local Heroes, Center PEG, PIT also Practices, Dedicated Improvement Processors, Improvement Action Teams).
- **Process and Communication Patterns** : regroupe les patrons ayants trait à la communication ou au procédé d'amélioration proprement dit (Process is Product *(process)*, Virtual Forum *(communication)*, Process follows Practice *(process)*, Improvement follows Process *(process)*, Improvement follows Spiral *(process)*).

I.1.3.3. Le langage de patrons cOOherentBPR

Le langage de patrons cOOherentBPR **[Beedle97]** est une collection de 30 patrons, qui touchent à différentes valeurs organisationnelles dans des projets de réingénierie de procédés métiers.
Particulièrement, ces patrons s'intéressent à la réingénierie dans les entreprises dont l'architecture est orientée objet ; d'où l'appellation du langage de patrons cOOherentBPR (OO pour Object Oriented, et BPR pour Business Process Reengeneering).

Les patrons de cette collection sont les suivants : *Leader, BusinessArchitect, Sub-Architect, Business Architecture Team, EnterpriseArchitecture, ScenariosDefineBusinessProcesses, CoreProcesses, CaseOfAction, VisionStatement, LearningOrganization, EncourageProductiveValues, ProcessOwners, StrategicDirectionInitiative, BusinessArchitectureClientTeam, OrganizationFollowsProcesses, FlatReportingStructure, CoachCoach, CompressTheProcess, CaseApplication, CaseWorker, CaseTeam, CaseManager, ApplicationFollowsProcess, ReifyTheProcess, SharedBusinessObjects, MultipleProcessVersions, IntegrateCustomersAndSuppliers, ProcessWithMinimalChecksAndControls, ProcessWithMinimalReconciliation, ModelOffice.*

I.1.3.4. Le langage de patrons OOSP

Dans **[Ambler98] [Ambler98a] [Ambler99]** est introduit le procédé de développement orienté objet OOSP (Object Oriented Software Process), qui combine expériences réelles du terrain et théories prouvées dans le domaine de développement logiciel orienté objet. Ce procédé se compose de quatre phases séquentielles et chaque phase de plusieurs étapes itératives, comme le montre la figure suivante :

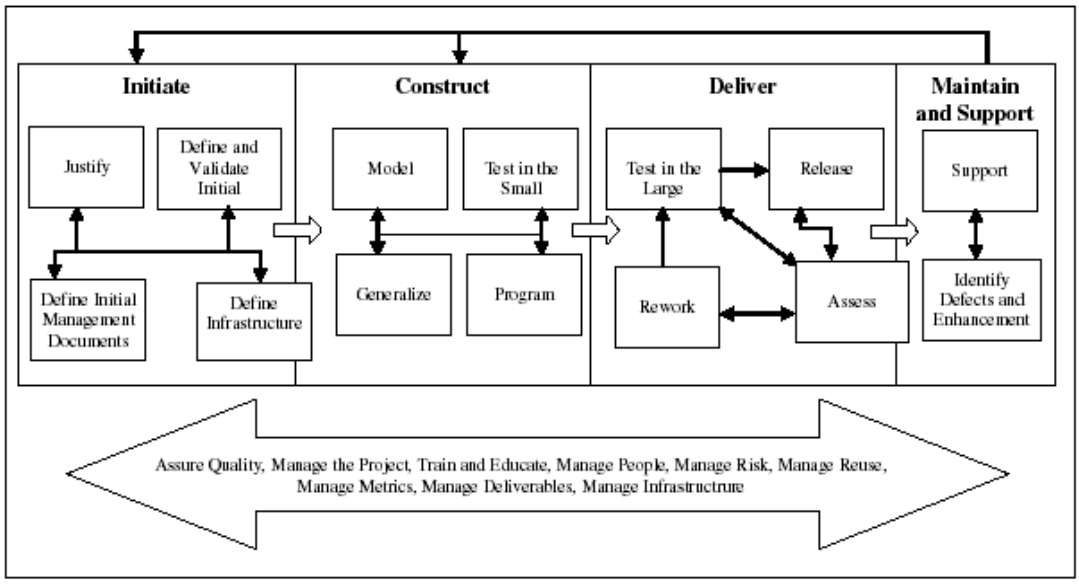


Figure I.12. Procédé OOSP **[Ambler98]**

Ce procédé est représenté comme une collection de patrons de procédé couvrant l'ensemble du cycle de développement des logiciels. Ambler distingue trois types de patrons :

- **Patrons de tâche** : Ce type de patron de procédé décrit les éléments pour accomplir une tâche spécifique.
- **Patrons d'étape** : Ce type de patron de procédé décrit les pas qui sont souvent exécutés itérativement, d'une étape de projet. Un patron d'étape peut être composé de plusieurs patrons de tâche.

- **Patrons de phase :** Ce type de patron de procédé décrit les interactions entre les patrons d'étape pendant une phase de projet. Un patron de phase est un ensemble de patrons d'étape.

Dans ses deux livres, Ambler identifie quatre patrons de phase et quatorze patrons d'étapes correspondants :

- Patron de phase **Initiate** : il comprend les quatre patrons d'étape *Define and Validate Initial Requirements, Define Initial Management Documents, Justify* et *Define Infrastructure*.
- Patron de phase **Construct** : il comprend les patrons d'étape M*odel, Program, Test In The Small* et *Generalize*.
- Patron de phase **Deliver** : il comprend les patrons d'étape *Test In The Large, Rework, Release* et *Assess*.
- Patron de phase **Maintain and Support** : il comprend les patrons d'étape *Support* et *Identify Defects and Enhancements*.

Pour chaque patron d'étape, Ambler mentionne des patrons de tâche à appliquer dans l'étape. En outre, plusieurs patrons de tâches qui doivent être appliqués tout au long du développement sont également décrits.
Alors que les patrons de phase et les patrons d'étape sont définis et représentés dans un formalisme structuré, les patrons de tâches d'Ambler ne sont pas identifiés et décrits clairement.

I.1.3.5. Le langage de patrons de Bergner

Bergner et al. **[Bergner98]** ont proposé un langage de patrons pour le domaine du développement logiciel à base de composants. Ce langage est constitué de patrons classés en 4 catégories selon le niveau de détails apportés par ces patrons :

- **Project Patterns** : Ce sont les patrons qui donnent des guidages tout au long du processus de développement. Ils décrivent les ordres de réalisation des phases pour créer les artefacts principaux (MainResult). Citons dans ce groupe les patrons *Topdown, BottomUp, ArchitectureDriven, Roundtrip, Iterative*.
- **Inter-Result Patterns** : Un *Inter-Result Pattern* suggère une solution qui produit les artefacts dans au moins deux différents artefacts principaux du développement. Les patrons suivants sont des Inter Result Patterns: *TunePerformance, ReEngineering, ReverseEngineering, PeriodicalBuild, TechnicalFoundation, Combined Experimental Prototyping, Component Assessment, Component Update, Component Innovation.*
- **Main Result Patterns**: Ces patrons concernent la manière d'élaborer les artefacts principaux du développement. Ce groupe est composé des patrons suivants : *Customer-Driven AnalysisMarket Driven Analysis, Market-Driven Analysis, Experimental Prototyping, Explorative Prototyping, Design-Driven Evaluation, Component-Driven Evaluation.*

- **Subresult Patterns :** Ce type de patrons fournit les conseils pour la création d'un artefact (subresult) dans une phase de développement. Parmi les exemples de ces patrons on peut citer le patron *Adaptation by Wrapping***,** et le patron *Adaptation by Reimplementation.*

I.1.3.6. Le catalogue XDP

Le catalogue Xml Design Patterns (XDP) **[XmlPatterns00]** comporte 28 patrons, et vise à développer des documents XML efficaces.

Le catalogue a comme but d'applique le concept de patron à la conception de structures XML. Comme l'utilisation du XML s'est répandue, et comme les patrons sont un excellent moyen pour passer la connaissance des experts vers les débutants, ces patrons sont devenu plus importants que jamais.

I.1.3.7. Le catalogue RTDP

Le catalogue Real-Time Design Patterns (RTDP) **[Douglass02a]** s'intéresse aux systèmes embarqués et à temps réel (Real-Time and Embedded systems RTE).

Les patrons de ce catalogue gravitent autour de six principales problématiques inhérentes aux RTE, à savoir :

- Identification des décisions stratégiques de larges envergure, qui affectent les éléments logiciels.
- Coordination et organisation des composants et systèmes.
- Gestion de la mémoire et des ressources.
- Définition de la manière de distribuer des objets à travers multiples systèmes.
- Construction d'architectures sécurisées et fiables.
- Mappage des sous systèmes et composants au matériel.

I.1.3.8. Le catalogue de Ruping

La problématique de ce catalogue **[Ruping03]** est la suivante : « Un projet peut il réellement suivre les principes Agiles tout en continuant à produire de la bonne documentation ? ». Ce catalogue est constitué d'un ensemble de patrons, qui offrent des solutions à des problèmes récurrents affrontés dans le domaine de la documentation.

Les solutions de ces patrons observées lors de projets précédents peuvent être utilisées comme guidage (guidelines) pour la documentation de projets futurs.
Les patrons de Ruping sont organisés dans les cinq chapitres suivants, qui traitent chacun un aspect particulier de la documentation des projets. Ces chapitres sont :

- *Finding the Right Topics*: ce chapitre présente quelques guidages sur le choix des documents qu'un projet requiert (quels documents sont nécessaires ? quels sujets sont à traiter ? quel est le niveau de détails à apporter ? quels documents sont à éviter ?).
- *Structuring Individual Documents*: ce chapitre offre des suggestions pour améliorer la lisibilité des documents du projet (comment structurer un document ? comment s'assurer que le lecteur retrouve facilement l'information recherchée ?).
- *Layout and Typography*: ce chapitre aborde l'amélioration de l'apparence des documents (comment la présentation du document favorise la compréhension rapide de son contenu ? comment aboutir à cette présentation avec des outils standards ?).
- *Infrastructure and Technical Organisation*: ce chapitre aborde la gestion d'un projet de documentation sur les plans organisationnel (comment avoir une vue d'ensemble sur la documentation projet ? quel est le format adéquat pour la documentation ?) et technique (comment traiter et sauvegarder les documents ? comment s'assurer de la disponibilité des documents ? quelle démarche suivre pour faciliter la maintenance des documents, et quels sont les outils nécessaires ?).
- *Management and Quality Assurance*: ce denier chapitre traite des problèmes de gestion de projet qui affectent la documentation, en mettant l'accent sur les rôles des intervenants et l'importance de leur feedback (comment éviter la bureaucratie dans un procédé de documentation ?).

Le catalogue de Ruping contient, pour chacun de ces cinq chapitres, un rapport d'expérience montrant comment les patrons du chapitre ont été appliqués dans différents projets.

I.1.3.9. Le catalogue OORP

Le catalogue de patrons de Demeyer et al. **[Demeyer03]** intitulé Object-Oriented Reengineering Patterns (OORP), est une collection de patrons s'intéressant à la réingénierie (Reengineering) et à la rétro ingénierie (Reverse engineering) de logiciels orientés objet. L'ingénierie et la rétro ingénierie sont montrés dans la figure suivante :

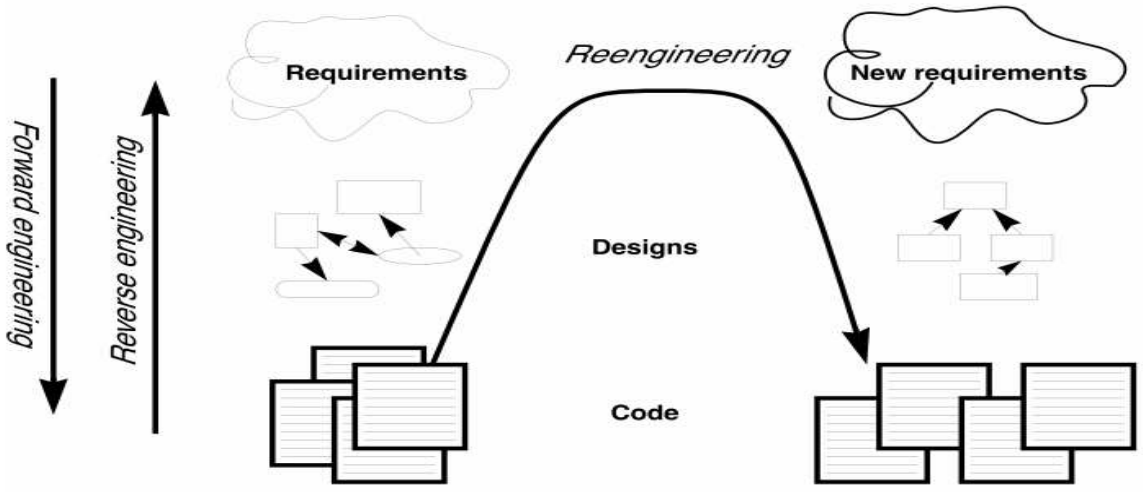


Figure I.13. Retro ingénierie et Réingénierie **[Demeyer03]**

Le catalogue OORP est constitué de deux partie, l'une pour les patrons de Retro ingénierie et l'autre pour les patrons de Réingénierie. Les patrons sont rassemblés dans des groupes (clusters) montrés dans la figure I.14, et chaque groupe traite une phase particulière du procédé :

- Phase de réingénierie (traitée par les groupes *Tests: your life insurance!, Migration strategies, Detecting duplicated code, Redistribute responsibilities, Transform conditionals to polymorphism*).
- Phase de retro ingénierie (traitée par les groupes *Setting direction, First contact, Initial understanding, Detailed model capture*).

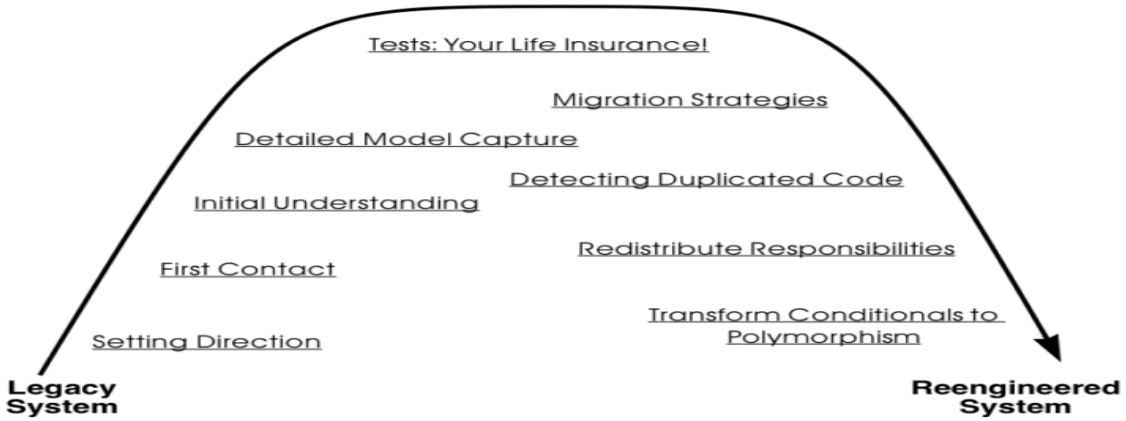


Figure I.14. Carte des différents groupes de patrons **[Demeyer03]**

I.1.3.10. Le catalogue de Tidwell

La collection de patrons de Tidwell **[Tidwell05]** est un langage de patrons proposant des solutions aux problèmes de conception d'interfaces homme machine, dans des applications interactives ou web.

Les patrons de ce catalogue abordent les concepts clés de la conception d'interfaces interactives ou web, à savoir : l'architecture des information ; la navigation ; la mise en page ; la gestion des cartes, graphes, et tables ; les formulaires ; les éditeurs graphiques ; les couleurs, la typographie, et le « look-and-feel ».

I.1.3.11. La collection de patrons YDPL

La collection de patron Yahoo ! Design Patterns Library **[YDPL]** est une librairie qui propose 59 patrons, destinés à la conception d'interfaces utilisateurs (user interfaces) notamment les interfaces web.

I.1.3.12. Catalogue de Staudt

Le catalogue de Staudt et al. **[Staudt10]** est constitué de patrons de haut niveau, traitants les problèmes communs de gestion des exceptions. Ces patrons s'intéressent à l'utilisation du mécanisme de gestion des exceptions dans différents contextes de procédés. L'objectif de ces

patrons est de permettre aux concepteurs de procédés de réfléchir en terme de ces patrons et distinguer quand est ce qu'ils sont utiles dans un procédé.

Le catalogue de Staudt est organisé en trois catégories de patrons qui sont les suivantes :

- *Trying other alternatives* : Cette catégorie s'intéresse au choix d'une alternative parmi d'autres, lorsqu'aucune condition n'est explicite dans le procédé pour faire ce choix. Dans un tel cas, une alternative est essayée et si elle s'avère non adéquate, il faut utiliser le mécanisme de gestion des exceptions pour essayer une autre alternative. Cette catégorie comporte les patrons *Ordered Alternatives* et *Unordered Alternatives*.

- *Inserting behavior* : Lors de la résolution d'un problème rencontré en cours d'exécution d'un traitement, il est très important de choisir l'instant (timming) de la résolution du problème par rapport à l'instant de son indentification. Ainsi, le problème est soit tout de suite résolu, soit sa résolution est différée jusqu'à la fin de l'exécution du traitement en cours. Un autre point important est la nature des activités de résolution du problème : soit ces activités sont conçues spécialement pour résoudre le problème, soit elles représentent un retour arrière pour réexécuter un traitement déjà réalisé.
  Se sont ces comportements qui sont traités par les patrons de cette catégorie, qui sont *Immediate Fixing, Defferred Fixing, Retry* et *Exception Driven Rework*.

- *Canceling behavior* : Cette catégorie de patrons s'intéresse au cas où une action en cours de traitement ne doit pas être achevée pour une raison quelconque. Cette catégorie comporte les patrons *Reject* et *Compensate*.

I.1.4. Organisation des patrons

Si un patron est une solution récurrente à un problème dans un contexte donné, un *système* de patrons est une collection de solutions qui coopèrent pour résoudre un problème complexe. Organisés en de tels systèmes, les patrons peuvent exprimer pleinement leurs forces **[Coplien96]**. Ainsi, un système de patron est une collection de patrons couvrant un même domaine, dont l'application est assurée par des inter relations qui permettent de les combiner.

Du coup, les relations entre patrons sont à la base de la composition de ces derniers, pour permettre de résoudre des problèmes complexes qui ne sont pas traités par un patron tout seul. Ainsi, il est nécessaire de savoir quels patrons peuvent fonctionner ensemble et les patrons dépendants, pour pouvoir former une solution. Donc, les contextes des patrons doivent être exprimés avec plus d'exactitude, et les relations entre patrons doivent être définies plus précisément **[Hagen02]**.

Partant de cela, plusieurs chercheurs ont abordé les relations entre patrons : certains ont travaillé sur la définition de ces relations avec plus ou moins de notations graphiques et de

langages pour les représenter, d'autres travaux plus évolués ont tenté de faire ressortir ces relations dans la cas où elles ne sont pas explicites, ou lorsqu'elles concernent différents catalogues. Quelques exemples de travaux sont cités dans ce qui suit.

Travaux de définition de relations entre patrons :

I.1.4.1. Relations de Zimmer

Walter ZIMMER **[Zimmer94]** est le premier à définir les relations entre patrons de conception orientée objet, en utilisant pour ce faire les patrons du Gang of Four **[GoF95]**. Selon Zimmer, trois relations peuvent exister entre les patrons :

- Un patron peut utiliser un autre.
- Un patron peut être combiné avec un autre.
- Un patron peut être similaire à un autre.

I.1.4.2. Relations de Stal

STAL et al. **[Stal96]** décrivent trois types de relations, très proches de celles de Zimmer, à savoir :

- Un patron peut être une variante d'un autre.
- Un patron peut utiliser un autre.
- Deux patrons peuvent être utilisés en combinaison pour résoudre un problème.

I.1.4.3. Relations de Meszaros

MESZAROS et al. **[Meszaros97]** identifient dans leur travail cinq relations pouvant exister entre patrons, qui sont les suivantes :

- Un patron peut utiliser un autre.
- Un patron peut être utilisé par un autre.
- Un patron généralise un autre.
- Un patron spécialise un autre.
- Un patron fournit une alternative à un autre patron.

I.1.4.4. Relation de Beck

Beck présente dans son travail **[Beck97]** une relation entre patrons intitulée *Conflicts*. Cette relation n'est autre que la relation Alternative, lorsqu'il s'agit de deux patrons abordant le même problème mais dont la solution de l'un est l'opposée de la solution de l'autre.

Pour illustrer cette relation, Beck donne l'exemple suivant de deux patrons proposants comment un programmeur Smalltalk peut accéder aux variables d'un objet. La solution du patron *Direct Variable Access* préconise un accès directe aux variables, car c'est plus facile à lire et à compiler. En opposé, la solution du patron *Indirect Variable Access* préconise d'accéder indirectement aux variables via les accesseurs, car ainsi on peut changer les variables sans changer le code qui les utilise.

I.1.4.5. Relations de Henney

HENNEY **[Henney99]** dit « un riche ensemble de relations entre patrons est le sang qui permet à un langage de patrons de devenir vivant ». Il définit les relations suivantes entre patrons :

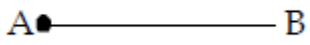
- Solution is context : cette relation signifie que la solution d'un patron est une part du contexte d'un autre. Cela inclut le cas commun où un patron fournit un support à un autre.
- Specialization : un patron résout un problème plus spécifique qu'un autre.
- Pattern pair : deux patrons résolvent le même problème mais avec des rubriques Forces complémentaires.
- Inverted patterns : deux patrons proposent des solutions à des problèmes similaires, mais avec des structures (solutions) presque inverses.

I.1.4.6. Relations de Volter

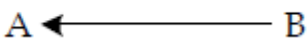
Volter dans son travail **[Volter00]** indique qu'un langage de patron tire sa force principalement de la façon avec laquelle ses patrons sont inter liés. Concernant ces liens, Volter définit trois types de relations, et propose pour chacune un symbole pour pouvoir exprimer graphiquement un ensemble de patrons liés.

Les relations sont les suivantes :

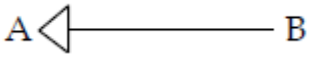
- A provides context for B : cette relation signifie qu'une fois le patron A appliqué, il est possible d'appliquer le patron B ensuite. Cette relation est symbolisée comme suit :



- B is required to make A work : pour pouvoir appliquer le patron A, on doit appliquer le patron B, car le patron A ne peut fonctionner sans le B. cependant, le patron B peut être exploité dans un autre contexte (c'est un patron à part entière, indépendant du A). Habituellement, le A est un patron de plus haut niveau que le B, et ce dernier permet de résoudre un aspect particulier du patron A. Cette relation est symbolisée comme suit :



- B is a specialization of A : cette relation signifie que B est une forme particulière de A. B fournit une adaptation de A pour certaines circonstances. Cette relation est symbolisée comme suit :



I.1.4.7. Relations de P-Sigma

Pour tirer pleinement profit des patrons de procédés, l'équipe Sigma a définit quatre types de relations entre les patrons **[Rieu99] [Conte01]**, permettant de les organiser :

- **Alternative** : Un patron A est une alternative d'un patron B, si A résout le même problème que B mais propose une solution différente. Les deux patrons ont le même contexte, la même classification mais des forces différentes.
- **Raffine** : un patron A raffine un patron B, si le problème posé par A est une spécialisation de celui posé par B. B peut résoudre les problèmes posés par A. La force et le contexte de A sont enrichis par rapport à ceux de B.
- **Requiert** : un patron A requiert un patron B, si l'application de B est un pré requis à l'application de A. Ainsi, la solution modèle du patron B fait partie du contexte du patron A et/ou le patron B paraît dans le contexte du patron A.
- **Utilise** : un patron A utilise un patron B, si une partie des problèmes posés par A peuvent être résolus en partie ou complètement par B. Ainsi, la solution démarche de A est exprimée en utilisant le patron B.

I.1.4.8. Approche de Deneckère

Deneckère **[Deneckere01a]** utilise le concept de *Carte de processus* pour organiser les patrons d'un catalogue. Une carte est un graphe, les noeuds sont les **intentions** (buts à atteindre) et les arcs sont les différentes **stratégies** (façons d'atteindre les buts).

Le concept de **Section** est un élément clef de la carte de processus. Il représente un patron par un triplet <intention source, intention cible, stratégie>. Une section correspond à la façon d'atteindre une intention cible en partant d'une intention source, en suivant une certaine stratégie.

La structure d'une carte est décrite par la figure suivante :

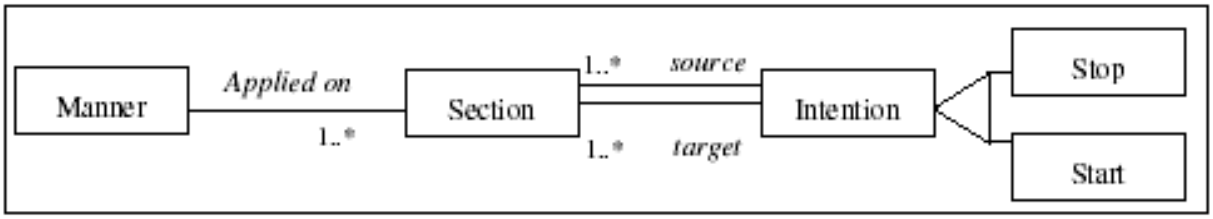


Figure I. 15. Structure d'une carte de processus **[Deneckere01a]**

Chaque carte contient deux intentions spécifiques appelées **Démarrer** et **Arrêter**. Si l'intention cible de la section correspond à un autre verbe (autre que démarrer et arrêter), cela signifie que cela mènera à l'exécution d'un patron. Une navigation dans la carte représente un chemin particulier commençant toujours par l'intention Démarrer et terminant par l'intention Arrêter.

Chaque patron est donc représenté dans la carte par une section spécifique. Autrement dit, l'application d'une stratégie permet de partir d'une intention source pour atteindre une intention cible. Le patron à appliquer sera différent selon la stratégie utilisée. Il y aura donc autant de patrons applicables, entre deux intentions, que de stratégies (et donc de sections dans la carte). La manière de naviguer dans une carte de processus est la même quelle que soit

la carte : on part du nœud *Démarrer*, choisit une stratégie afin d'atteindre un autre nœud et ainsi de suite, jusqu'à atteindre le nœud *Arrêter* qui permet de terminer l'exécution de la carte. Cette technique est un moyen d'organiser les patrons sous forme de sections dans la carte de façon à les retrouver et à les utiliser de la meilleure manière possible.

I.1.4.9. Approche de Gnatz

Gnatz et al. **[Gnatz03]** ont identifié les deux relations suivantes:

- **Alternative**: cette relation existe entre deux patrons qui résolvent le même problème.
- **Raffine**: cette relation peut être déduite entre un patron supérieur, et des patrons qui réalisent des activités contenues dans le patron supérieur.

L'approche de Gnatz ne fournit pas de spécification formelle et explicite de ces relations, mais propose une structure nommée **ProcessPatternActivityMap [Gnatz02]**. Cette structure représente un catalogue de patrons en reliant les patrons, avec les activité correspondantes aux problèmes qu'ils adressent et aux solutions qu'ils proposent.

Le ProcessPatternActivityMap est un graphe orienté avec deux sortes de noeuds: **Activity** et **Process Pattern**.
Un noeud Process Pattern a des arcs vers toutes les activités exécutées par ce patron, ce qui représente la relation **Execute**. Ce même nœud ProcessPattern à un arc vers l'activité qu'il réalise, et cela représente la relation **Realize**.

Ainsi, deux patrons réalisant la même activité sont liés par la relation **Alternative**. Et un patron réalisant une activité exécuté par un au autre patron est relié à ce dernier par la relation **Raffine**.

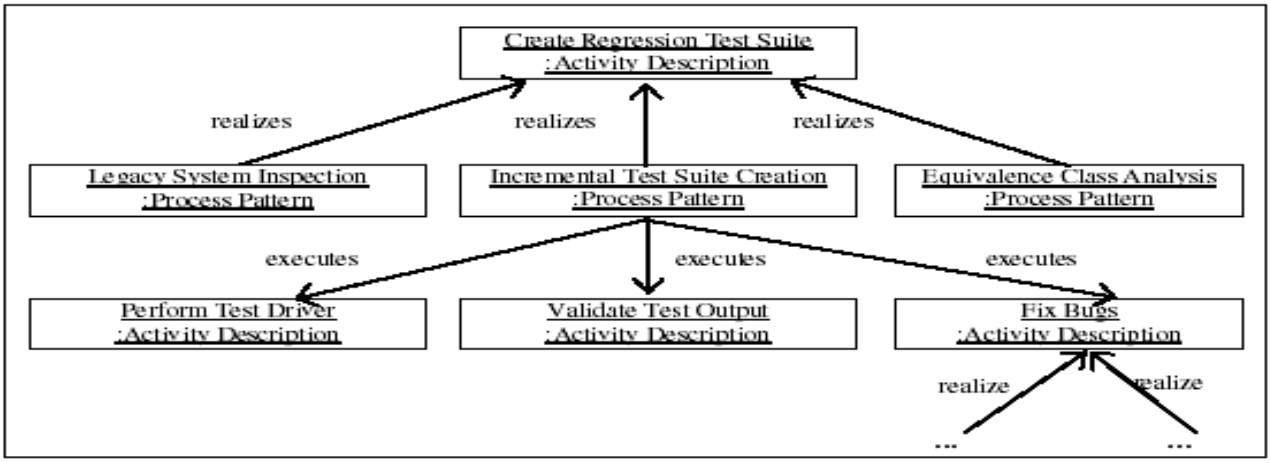


Figure I.16. ProcessPatternActivity Map **[Gnatz03]**

I.1.4.10. Relations de PROPEL

Les relations entre des patrons sont définies dans le langage PROPEL **[Hagen04a]** via le paquetage Relationship (cf. I.1.2.11). Un patron est donc associé à un autre patron par l'intermédiaire d'une ProcessPatternRelatonship. Il y a 4 types de relations entre patrons :

- **Sequence** : existe si un ou plusieurs patrons prédécesseurs (*predecessor*) produisent tous les objets et événements dont un patron successeur (*successor*) a besoin pour son exécution.
- **Use** : existe si un patron (*component*) décrit un sous procédé (c'est-à-dire une partie de la solution) d'un autre patron *(composite)*.
- **Processvariance** : associe deux patrons variant1 et variant2 s'ils résolvent un même problème en utilisant différentes solutions.
- **Refinement** : associe deux patrons (*superpattern* et *subpattern*) si les problèmes de ces patrons sont associés par la relation RefineProblem.
    - **RefineProblem** : associe deux problèmes (*superProblem* et *subProblem*) si le *subProblem* raffine le *superProblem*.

PROPEL fournit également des notations pour représenter les quatre relations : Sequence, Use, Processvariance et Refinement. Ces notations sont montrées dans les figures suivantes :

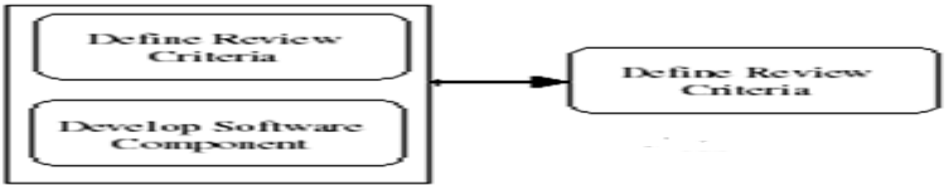


Figure I.17. Notation de la relation Séquence **[Hagen04a]**

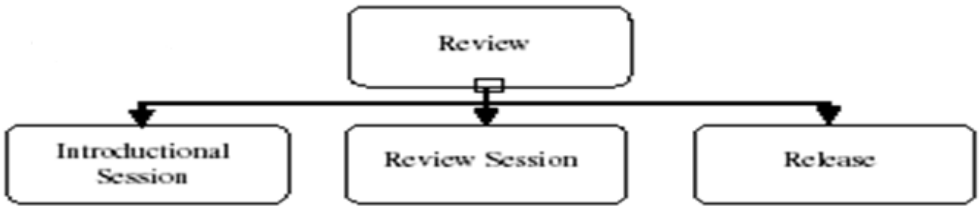


Figure I. 8. Notation de la relation Use **[Hagen04a]**

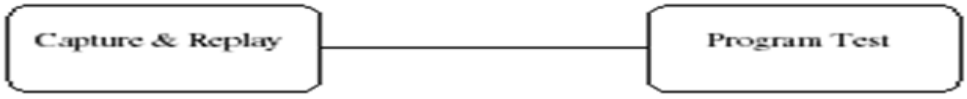


Figure I.19. Notation de la relation Process-variance **[Hagen04a]**

Figure I.20. Notation de la relation Refinement **[Hagen04a]**

Travaux d'analyse de relations entre patrons :

I.1.4.11. Approche de Prabhakar

Prabhakar et al. **[Prabhakar10]** proposent un modèle graphique DDTM (Design Decision Topology Model), pour analyser les relation entre patrons de conception de manière automatique. Ces relations sont assimilées aux relations entre les graphes représentants les patrons.

En effet, chaque patron est considéré comme une topologie d'un ensemble de *décisions de conception* (Design Decisions DD), cette topologie capture la sémantique du patron. Une décision de conception est une stratégie appliquée pour résoudre une partie du problème traité par le patron. Les décisions de conception sont définies dans les travaux **[Bass03]** et **[Kruchten04]**.
Les propriétés des graphes sont appliquées dans la méthode de Prabhakar pour analyser quatre types de relations entre patrons, à savoir :

- ***Is duplicate of*** : cette relation signifie que deux patrons fournissent la même solution au même problème, et est analysée à travers la propriété d'équivalence des graphes.

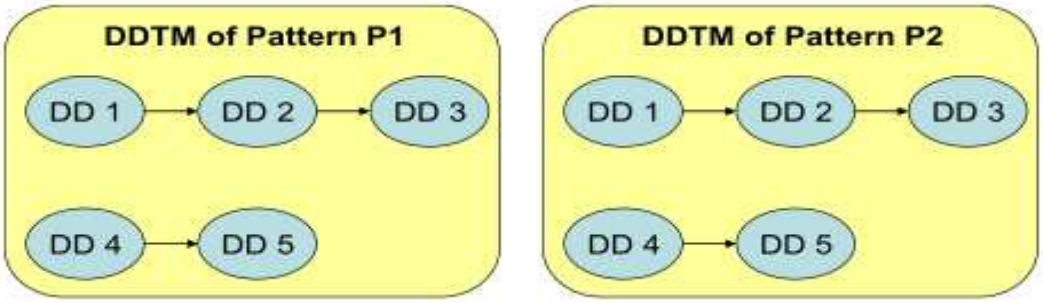


Figure I.21. Exemple de graphes de patron en relation Is duplicate of **[Prabhakar10]**

- ***Is an alternative to*** : cette relation signifie que deux patrons fournissent différentes solutions au même problème, et est identifiée si les ensembles de nœuds de départ des deux graphes (représentants les deux patrons) sont identiques mais le reste des graphes est différent.

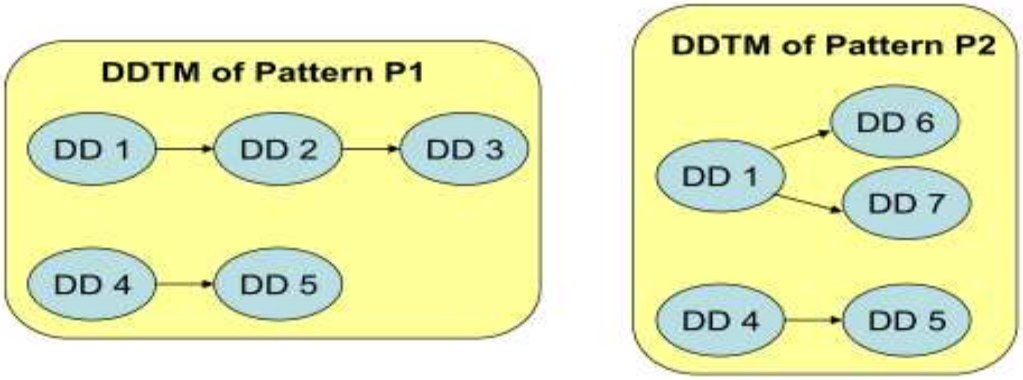


Figure I.22. Exemple de graphes en relation Is an alternative to **[Prabhakar10]**

- ***Uses*** : P1 uses P2 signifie qu' un sous problème de P1 est similaire à P2, et cette relation est identifiées lorsque le graphe de P2 est un sous graphe de P1.

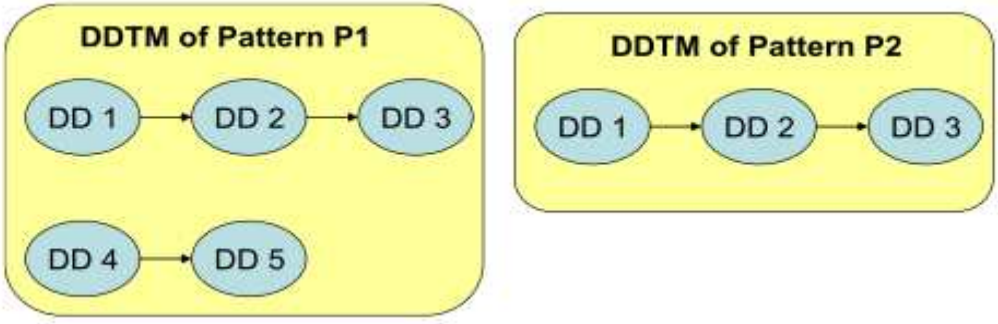


Figure I.23. Exemple de graphes de patron en relation Uses **[Prabhakar10]**

- ***Sub sums*** : P1 refines P2 signifie que P1 et P2 traitent le même problème mais P1 fournit plus de détails que P2, et cette relation est identifiée lorsque les ensembles de nœuds de départ des deux graphes sont équivalents et le graphe de P2 est un sous graphe de P1.

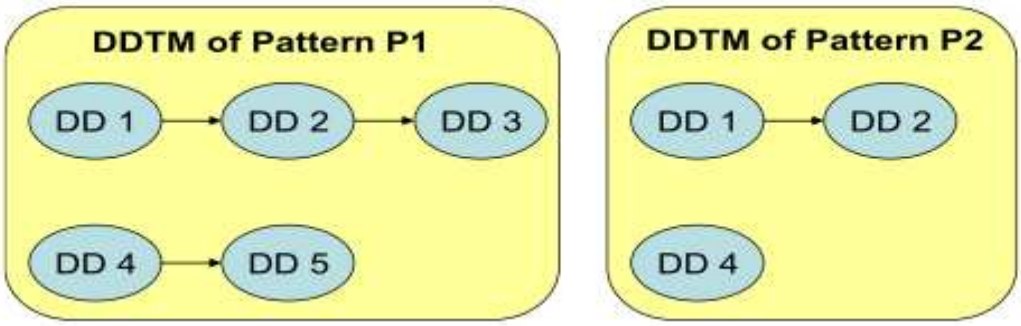


Figure I.24. Exemple de graphes de patron en relation Sub sums **[Prabhakar10]**

I.1.4.12. Approche de KUBO

KUBO et al. **[Kubo05]** sont les premiers à aborder les relation entre les patrons de différentes natures et de différents catalogues, pour permettre de les combiner. Ce travail présente une approche automatique d'analyse des relations entre patrons.

Un patron dans cette approche est considéré comme une transition allant d'un contexte initial vers un contexte final, la transition est étiquetée par les contraintes du patron (forces). Dans un premier temps, trois types de relations étaient considérés dans ce travail :

- *Starting-Starting* : lorsque les contextes initiaux de deux patrons sont identiques, ces deux patrons fournissent des solutions différentes au même problème. La figure suivante montre la représentation graphique de cette relation :

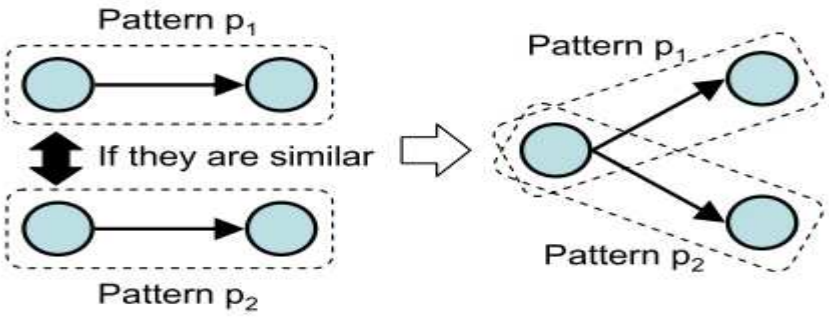


Figure I.25. Représentation graphique de la relation Starting-Starting **[Kubo05b]**

- *Resulting-Resulting* : lorsque les contextes finaux de deux patrons sont similaires, ces deux patrons fournissent le même résultat une fois appliqués. La figure suivante montre la représentation graphique de cette relation :

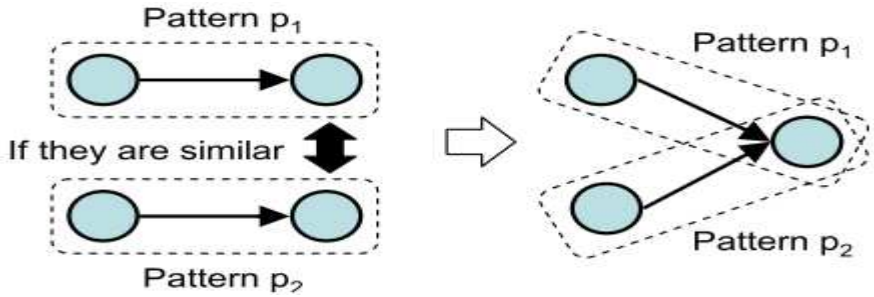


Figure I.26. Représentation graphique de la relation Resulting-Resulting **[Kubo05b]**

- *Resulting-Starting* : quand le nœud de départ du patron P2 et le nœud final du patron P1 sont identiques, on peut appliquer le patron P2 après P1. La figure suivante montre la représentation graphique de cette relation :

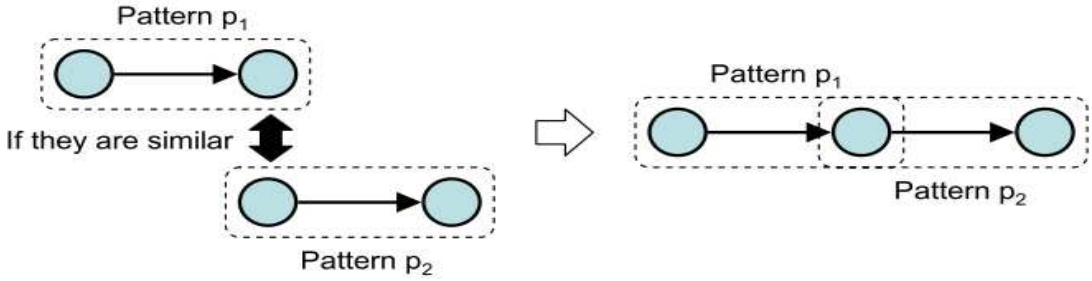


Figure I.27. Représentation graphique de la relation Resulting-Starting **[Kubo05b]**

Plus tard, ces trois relations ont été rebaptisées *SimilarProblem*, *SimilarResult* et *Continuous* respectivement dans **[Washizaki07]**, et quatre autres relations ont été considérées dans **[Washizaki07]** pour enrichir la méthode originale de Kubo et al., en se basant toujours sur le modèle de patrons présenté dans le travail original.
Ces relations sont les suivantes :

- *Same* : deux patrons sont liés par cette relation si les deux sont similaires dans leurs contextes initiaux et dans leurs contextes finaux.

- *SubInStarting* : deux patrons sont liés par cette relation si les contextes initiale et finale d'un patron sont similaire au contexte initial d'un autre patron.
- *SubInResulting* : deux patrons sont liés par cette relation si les contextes initiale et finale d'un patron sont similaire au contexte final d'un autre patron.
- *SimilarForce* : deux patrons sont liés par cette relation si les contraintes des deux patrons (forces) sont les similaires.

I.1.5. Réutilisation des patrons

Une approche globale pour le développement de patrons logiciels nécessite :

- Une ingénierie *pour la réutilisation :* a pour but de produire les patrons à réutiliser et consiste à identifier, spécifier et organiser les patrons logiciels.
- Une ingénierie *par la réutilisation :* a pour but de réutiliser les patrons d'une façon systématique. Cette ingénierie facilite la recherche, la sélection, l'adaptation et l'intégration de patrons pour modéliser des solutions.

Ainsi, il faut d'abord produire des patrons à réutiliser (ingénierie pour la réutilisation), ensuite réutiliser ces patrons (ingénierie par la réutilisation) pour modéliser les solutions.

Sont donnés dans ce qui suit quelques travaux de réutilisation de patrons.

I.1.5.1. PROMENADE

L'ingénierie de réutilisation dans PROMENADE **[Ribo00] [Ribo02]** s'intéresse aux patrons de procédé , et est assurée par deux activités : la Récolte (Harvesting) et la Réutilisation (Reuse).
La **Récolte** est une transformation de modèles de procédé exécutables en patrons de procédé réutilisables. A l'opposé, la **Réutilisation** est une transformation de patrons de procédé en modèles de procédé exécutables.

Sachant qu'un patron de procédé dans PROMENADE est un modèle de procédé paramétrable, les deux activités (récolte et réutilisation) peuvent être effectuées en utilisant les mécanismes suivants :

Abstraction : ce mécanisme permet d'éliminer les détails spécifiques d'un modèle de procédé pour obtenir un modèle plus générique.

Adaptation : ce mécanisme a pour but de rendre un modèle de procédé plus spécifique pour être réutilisé dans une situation particulière. Les mécanismes d'adaptation impliquent les modifications de modèles suivantes :

- Addition de nouveaux éléments au modèle de procédé pour rendre le modèle de procédé plus spécifique : *spécialisation* et *instanciation.*

- Suppression d'éléments (inutiles) du modèle de procédé pour limiter le modèle aux éléments choisis et à leur contexte : *projection.*
- Substitution d'éléments du modèle de procédé : *re-dénomination* et *substitution sémantique.*

Composition : ce mécanisme combine un ensemble de modèles de procédé afin d'en créer un nouveau. PROMENADE distingue 3 mécanismes de composition :

- *Grouping :* ce mécanisme regroupe un ensemble de modèles de procédé sans ajouter de sémantique supplémentaire spécifique.
- *Combination :* ce mécanisme regroupe un ensemble de modèles de procédé en définissant plusieurs relations de précédence entre les tâches des modèles composants.
- *Inclusion :* ce mécanisme incorpore la fonctionnalité d'un modèle de procédé à un autre modèle de procédé. Le modèle incorporé devient une sous-tâche d'une tâche complexe du modèle destinataire, avec éventuellement certains comportements supplémentaires.

Liaison tardive : ce mécanisme permet de retarder le choix d'une partie spécifique d'un modèle de procédé jusqu' au moment de l'exécution.

I.1.5.2. Méthode de TRAN

TRAN a proposé **[TRAN07]** pour l'ingénierie de réutilisation une méthode de modélisation de procédés à base de patrons de procédé. Cette méthode comporte d'une part des opérateurs de réutilisation des patrons de procédé, et d'une autre part une démarche de modélisation de procédés à base de patrons décrite sous forme d'un méta-procédé appelé PATPRO.

Donc les opérateurs permettent de manipuler les patrons de procédé dans le but de les réutiliser, alors que le méta-procédé décrit quand et comment réutiliser les patrons.
Le méta procédé définit un ensemble de méta-tâches pour modéliser les procédés qui, en s'exécutant, peuvent utiliser les opérateurs de réutilisation.

Les opérateurs de réutilisation
Les opérateurs de base pour manipuler les patrons de procédé sont classés en trois catégories montrés dans la figure suivante : *opérateurs de recherche*, *opérateurs d'adaptation* et *opérateurs d'imitation.*

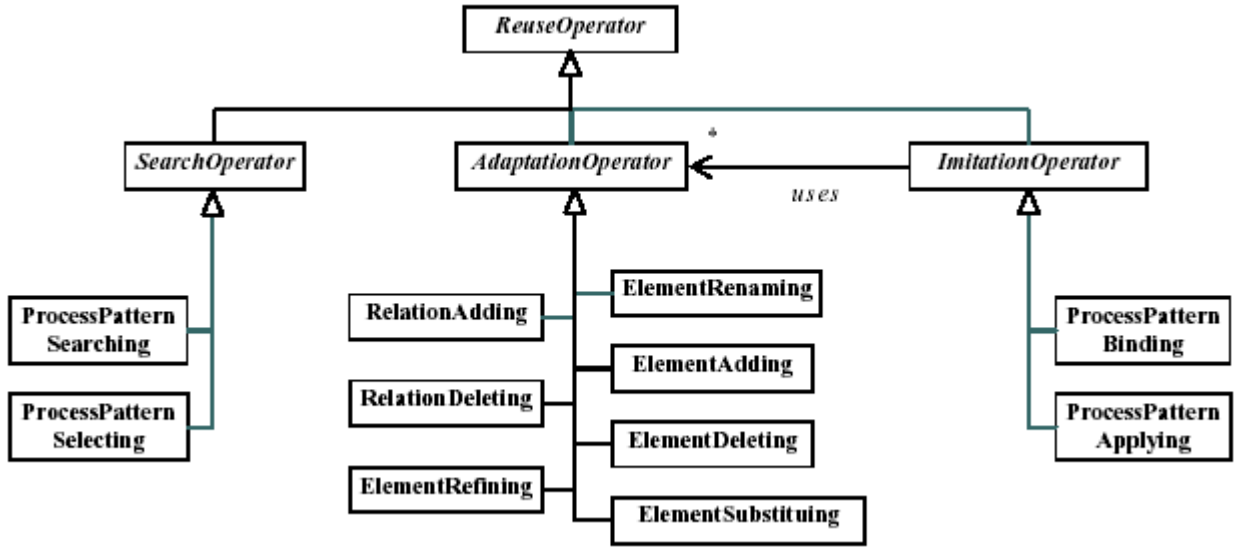


Figure I.28. Opérateurs de réutilisation des patrons de procédé **[TRAN07]**

Opérateurs de recherche : permettent à partir d'un ensemble de patrons de procédé, d'identifier et de sélectionner les patrons les plus adaptés à un problème donné, de modélisation de procédés. On propose deux opérateurs dans cette catégorie : *ProcessPatternSearching* et *ProcessPatternSelecting.*

Opérateurs d'adaptation : un patron est rarement appliqué sans modification. En général, l'application nécessite une adaptation de la solution du patron à un contexte spécifique. Cela peut être réalisé grâce aux opérateurs d'adaptation, qui permettent de manipuler les modèles de procédé capturés dans les solutions de patrons. Dans cette catégorie, on propose les opérateurs suivants : *ElementRenaming*, *ElementAdding*, *ElementDeleting*, *RelationAdding*, *RelationDeleting*, *ElementSubstituing* et *ElementRefining*.

Opérateurs d'imitation : permettent soit de générer un modèle à partir de la solution d'un patron (opérateur *ProcessPatternBinding*), soit d'appliquer la solution d'un patron à un modèle pour l'enrichir ou le restructurer (opérateur *ProcessPatternApplying*).

Le méta procédé PATPRO

PATPRO permet la construction ou l'amélioration de modèles de procédé, en guidant les concepteurs dans la réutilisation de patrons de procédé. PATPRO se présente principalement sous la forme des quatre méta-tâches suivantes :

- **Analyser les Besoins** : cette méta-tâche analyse les caractéristiques du procédé à modéliser, et identifie les besoins de modélisation.
- **Modéliser un Procédé** : cette méta-tâche élabore le modèle d'un procédé à partir de sa spécification en réutilisant éventuellement des patrons. C'est la tâche centrale de la modélisation.
- **Simuler un Procédé** : cette méta-tâche simule l'exécution d'un modèle de procédé par l'intermédiaire d'un moteur d'exécution de procédés.

- **Évaluer un Procédé** : cette méta-tâche évalue un modèle de procédé, en le comparant avec des critères prédéfinis, pour identifier éventuellement de nouveaux besoins.

I.1.6. Synthèse

Descendants des patrons d'architecture de Christopher Alexander, les patrons logiciels sont un excellent outil destiné à résoudre des problèmes récurrents, en apportant des solutions prouvées.

L'utilisation de patrons apporte beaucoup d'avantages à leurs utilisateurs, en leur permettant de tirer profit d'une connaissance prouvée, de la réutiliser autant de fois que nécessaires et de l'adapter à différents contextes. Des lors, les chercheurs se sont penchés sur différents aspects des patrons logiciels.

Les travaux portants sur les formalismes de patrons sont globalement équivalents. Ils diffèrent par le degré de détails plus ou moins élevé des rubriques, mais ils intègrent tous le triplet {problème, contexte, solution}. Pourtant, l'absence d'un formalisme commun pour les patrons est préjudiciable, car il est difficile d'interpréter et de comparer les éléments des patrons dans différents formalismes.

Parmi les langages de description de patrons, PROPEL et UML-PP sont dédiés aux patrons de procédé.
Les avantages de PROPEL sont la description de la structure interne d'un patron et la définition des relations entre patrons. Un autre avantage de PROPEL est la définition explicite du concept de catalogue des patrons.
L'avantage majeur de UML-PP est qu'il favorise la réutilisation de patrons, étant un méta modèle pour modéliser les procédés par réutilisation de patrons. De plus UML-PP fournit les paquetages nécessaires pour décrire les patrons et leur organisation.

Concernant les recherches portants sur l'identification des patrons logiciels, les travaux de Coplient, I-SPI et cOOherentBPR sont consacrés à identifier des patrons concernant le niveau organisationnel des organismes de développement logiciel.

Les travaux OOSP et de Bergner sont des patrons de procédé généraux, couvrants l'ensemble du cycle de développement du logiciel.

La collection XDP regroupe des patrons s'intéressants à la conception de structures XML. Quand à RTDP, ce travail identifie des patrons dans le domaine des systèmes embarqués et à temps réel.

Le catalogue de Ruping est l'unique collection de patrons dans la littérature, dédiée au domaine de la documentation Agile. Ce catalogue a l'avantage d'offrir des procédés qui restent valables même dans des environnements classiques.

A la différence des patrons qui abordent des procédés démarrant de zéro, les patrons OORP s'intéressent aux procédés qui démarre à partir d'un software hérité, et traitent ainsi des problèmes de réingénierie et de retro ingénierie de logiciels orientés objet.

Le travail de Tidwell et YDPL sont des catalogues de patrons orientés vers la conception d'interfaces homme machine.

Les patrons de Staudt sont des patrons de haut niveau, qui peuvent être exploités dans n'importe quel procédé métier pour gérer des exceptions, et c'est ce qui fait leur valeur.

Quand à l'organisation des patrons logiciels, Les travaux de Zimmer, Stal, Meszaros, Beck, Henney, Volter, P-Sigma et PROPEL utilisent l'approche de définition explicite des relations entre des patrons. Leurs relations sont de façon générale équivalentes, malgré quelques petites différences. Ces travaux couvrent les relations primaires entre patrons. Toutefois, ces relations présentent l'inconvénient de ne pouvoir être utilisées, que si elles sont citées de manière explicite dans des patrons. Aucun de ces travaux ne permet malheureusement de ressortir les relations, si elles ne sont pas explicites.

Deneckere et Gnatz abordent les relations entre patrons de manière implicite, et se focalisent sur la structuration des patrons de procédé en graphe, en fournissant la manière de navigation dans ces graphes pour choisir les patrons nécessaires.

Les travaux de Kubo et Prabhakar sont par contre parmi les rares travaux qui permettent de ressortir des relations non explicites entre patrons. De plus, ils présentent l'avantage d'aborder les inter-relations à travers des patrons de différents catalogues. Le travail de Prabhakar se limite malheureusement aux patrons de conception, alors que le travail de Kubo et al. se distingue d'avantage par sa capacité de gérer des patrons de différentes natures.

En ce qui concerne les travaux de réutilisation de patrons, PROMENADE considère les patrons de procédés comme des modèles de procédé paramétrables, et fournit un ensemble de mécanismes bien définis pour manipuler et appliquer ces patrons ; mais il manque un guidage méthodologique pour une réutilisation systématique de patrons.

En fin, l'approche de Tran comporte des opérateurs de réutilisation des patrons de procédé, et un méta-procédé pour modéliser les procédés à base de patrons. Cependant, un modèle de procédé peut comporter des informations incomplètes qui seront raffinées ou complétées ultérieurement. Dans le méta-procédé de Tran, ce raffinage est effectué avant l'exécution de l'instance. Pourtant, pour être plus flexible et efficace, ce méta-procédé devra dans l'avenir prendre en compte l'adaptation dynamique des modèles de procédé **[TRAN07]**.

I.2. Documentation de logiciels

I.2.1. Définition et intérêts de la documentation de logiciels

Un procédé logiciel est une méthode de développement ou de production de logiciels **[Tyrrell00]**, dont le but est « aider l'ingénieur à faire un bon travail » au lieu d'être « évité à l'ingénieur de le faire mal ». Le principal artefact du procédé de développement logiciel est le code source, mais ce dernier n'existe jamais seul, il évolue dans un environnement constitué de différents autres artefacts produits ou requis lors du développement, de l'exploitation ou de la maintenance du logiciel.

Destinés à plusieurs types d'utilisation et offrants multiples intérêts, ces différents artefacts pérennisent des informations complémentaires couvrants tous les aspects du logiciel, et représentent la documentation de ce dernier. Cette documentation sert entre autres comme média de communication entre les différents intervenants du développement (exemple : Rapports d'avancement des travaux…), comme outil de vérification (exemple : Spécification des tests, Rapports de revue technique…), comme répertoire d'information pour les équipes de développement et les équipes de maintenance (exemple : Rapports d'analyse des besoins, Documents de conception, Descriptions techniques du système…), comme support pour les gérants dans la planification des travaux et budget de développement (exemple : Plan du projet, Stratégie des livrables…) et comme outil d'aide à l'exploitation du logiciel par les utilisateurs (exemple : Guides des utilisateurs, Documentation opératoire…).

Donc la documentation du logiciel englobe tout document généré pendant le développement logiciel, et non seulement les documents des utilisateurs (comme le croient certains) **[Cook94]**. Elle réfère tout artefact dont le but est de communiquer des informations concernant le logiciel auquel l'artefact appartient **[Forward02]**.

Ainsi, la documentation du logiciel est partie prenante dans un projet logiciel, et on ne peut s'en passer d'elle ni tolérer la dégradation de sa qualité. Elle joue un rôle essentiel dans la communication entre développeurs, managers et utilisateurs **[Garceau90]** ; permet de répéter les succès et d'éviter de refaire les erreurs ou de réinventer le roue ; permet aux compagnies de mesurer leurs performances courantes, d'identifier les zones de faiblesses et initialiser des action d'amélioration **[Coleman98]** ; pour ces raisons elle est imposée par les modèles de qualité (CMM, Six Sigma, Total Quality Management …). Toutefois, malgré que chacun soit d'accord que la documentation du logiciel est importante, certains ne réalisent pas qu'elle représente un acteur *critique* dans la qualité du logiciel **[Cook94],** du fait elle demeure un aspect souvent négligé des pratiques de développement **[Nasution09].**

En effet, depuis longtemps, des données empiriques ont montré que la documentation est un composant clé dans la qualité du logiciel **[Card87] [Lientz81] [Rombach87]**. Ces études révèlent qu'une documentation dépassée, incomplète ou de qualité médiocre est une cause majeure des erreurs dans le développement et la maintenance du logiciel **[Cook94]**.

Une solution à ces problèmes est l'amélioration du procédé de documentation **[Cook94]**. En réalité, le procédé de documentation est ni parallèle ni indépendant du procédé de développement logiciel, mais par contre contenu dans ce dernier. Du coup, toute amélioration dans le procédé de développement de documentation contribue à l'amélioration du procédé de développement logiciel, alors que toute lacune influe négativement sur ce dernier. Ce qui fait que l'idée d'intérêt du procédé de documentation, malgré qu'elle soit ancienne demeure un sujet d'actualité **[Nasution09]**. S'ajoute à cela que des problèmes restent posés, dans le domaine de la documentation de logiciels, qui nécessitent des améliorations et des solutions.

Dés lors, nombreux chercheurs se sont penchés sur le thème de la documentation du logiciel pour apporter des nouveautés, des solutions et des améliorations. Sont cités à titre non exhaustif quelques travaux dans ce domaine :

Des normes ISO ont abordé la documentation du logiciel de manière générale **[ISO04] [ISO05]**, puis d'autres normes ISO sont venues traiter des aspects plus précis **[ISO08] [ISO09a]**, et des projets de normes continuent à émerger **[ISO09b] [ISO09c] [ISO10]** pour apporter de plus en plus de guidages concernant cet élément critique des projets logiciels, à savoir la documentation.

Similairement, des guidages concernant la documentation de logiciels sont donnés par des standards et projets de standards de l'IEEE, couvrants des aspects divers **[IEEE01] [IEEE04] [IEEE08] [IEEE09a] [IEEE09b]**.

D'autres travaux de recherche s'intéressent à des problèmes de plus en plus accrus des procédés de documentation de logiciels tels que **[Sánchez09]**, **[Zdun09]**, **[Ramos08]**, **[Holdaway09]**, **[Bartell09]**, **[Nord09]**.

Certains chercheurs se penchent sur la structuration et catégorisation des documents, pour pérenniser de la connaissance projet et assurer un accès efficace aux grandes quantités d'informations, tels que **[Heinrich09]**, **[Kokkoniemi09]**.

D'autres se penchent sur la synchronisation de la documentation avec le code source, pour qu'elle reflète exactement l'état de ce dernier **[Reiss06]**, **[Zhang08]**, **[Díaz09]**, **[Ratanotayanon09]**, **[Ben09]**, ou bien la synchronisation des différents documents entre eux pour préserver la cohérence des documents **[McMillan09]**, **[Ratanotayanon09]**, **[Correia09]**.

La présentation du contenu des documents pour les optimiser est abordée par des travaux tels que **[Tilley09]**, **[Dzidek08]**, **[Torchiano10]**.
Des recherches sur les supports de documentation continuent à se faire comme **[Gotel09]**, **[Ben09]**. Alors que l'exploitation de la documentation pour les phases Test **[Connolly09]** et Maintenance **[Hyland08]**, **[Dzidek08]**, **[Torchiano10],** ainsi que la génération automatique de documents **[Carter09]**, **[Maalej09]**, **[Heinrich09]** font aussi l'objet de nombreux travaux.

I.2.2. Exemple d'une méthode de documentation classique (méthode de Sommerville)

Selon Sommerville **[Sommerville01]** la documentation du logiciel peut être groupée en deux catégories: *Documentation du procédé* produite pour permettre la gestion du développement du logiciel, et qui capture le procédé de développement et de maintenance. *Documentation du produit* produite pour être utilisée une fois le logiciel opérationnel, mais qui contribue aussi à la gestion du développement.

Documentation du procédé : constituée principalement des *Plans* du projet, *Rapports* d'utilisation des ressources, *Standards* ou méthodes utilisés par le procédé, *Working papers* (utilisés pour communiquer entre les membres de l'équipe, conserver les idées de ces derniers telles que les stratégies d'implémentation, les problèmes identifiés, les décisions de conception… et être des versions provisoires de la documentation du produit). *Memos et email* de la communication journalière entre membres de l'équipe.

Documentation du produit : destinée aux utilisateurs du logiciel. Elle décrit le logiciel produit, a une longue durée de vie et doit évoluer au pas avec le logiciel qu'elle décrit. Cette documentation se classe en 2 sous catégories : documentation d'*utilisateurs* et documentation du *logiciel*.

> La documentation d'utilisateurs regroupe les documents destinés aux utilisateurs finaux qui vont exploiter le logiciel dans l'exécution de leurs taches journalières ; et les documents destinés aux administrateurs et décideurs qui vont gérer le logiciel. La documentation d'utilisateurs doit contenir au moins les 5 types de documents suivants, montrés dans la figure qui suit, pour satisfaire les besoins des différents niveaux d'expertise des utilisateurs :

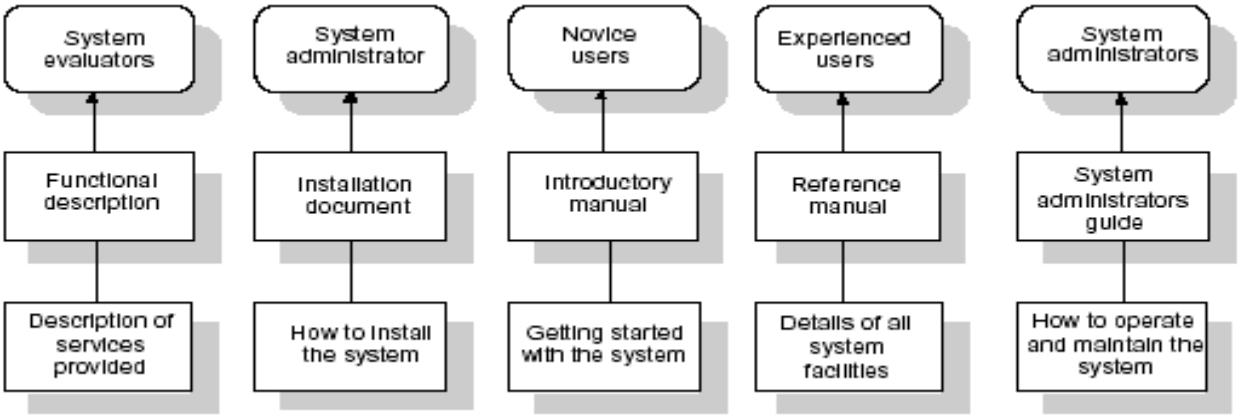


Figure I.29. Différents types de documents d'utilisateurs **[Sommerville01]**

- Description Fonctionnelle : décrit brièvement les services fournis par le logiciel. Utilisée avec le Manuel d'Introduction pour décider si le logiciel est approprié.

- Document d'Installation : détaille l'installation du logiciel ainsi que la configuration matérielle et logicielle requise.

- Manuel d'Introduction : simple introduction au logiciel décrivant comment les utilisateurs finaux pourront l'exploiter. Contient aussi les solutions aux erreurs les plus commises par les utilisateurs.

- Manuel de Référence : décrit les facilités offertes par le logiciel, en plus d'un listing des messages d'erreurs et la manière de résoudre ces dernières.

- Guide de l'Administrateur : ce document est fournit avec des logiciels particuliers tels que ceux de commandes et contrôle des systèmes; il décrit les messages générés lors de l'interaction du logiciel avec le système contrôlé et comment réagir à ces messages.

  La documentation du logiciel regroupe les 7 documents suivants :

- Document d'Analyse des besoins.

- Document d'Architecture du logiciel.

- Pour chaque programme dans le logiciel, un document d'Architecture du programme.

- Pour chaque composant dans le logiciel, un document de Description de ses fonctions et interfaces.

- Listings des codes sources avec commentaires et justifications des choix de codage.

- Document de Validation de chaque programme par rapport aux besoins.

- Guide de maintenance décrivant les problèmes connus et les parties du logiciel dépendantes d'un hardware ou d'un software particulier.

Enfin, pour aboutir à un document de qualité Sommerville **[Sommerville01]** recommande le modèle de procédé de préparation de documentation, montré dans la figure suivante, constitué de 3 étapes majeures : Création, Raffinage et Production du document.

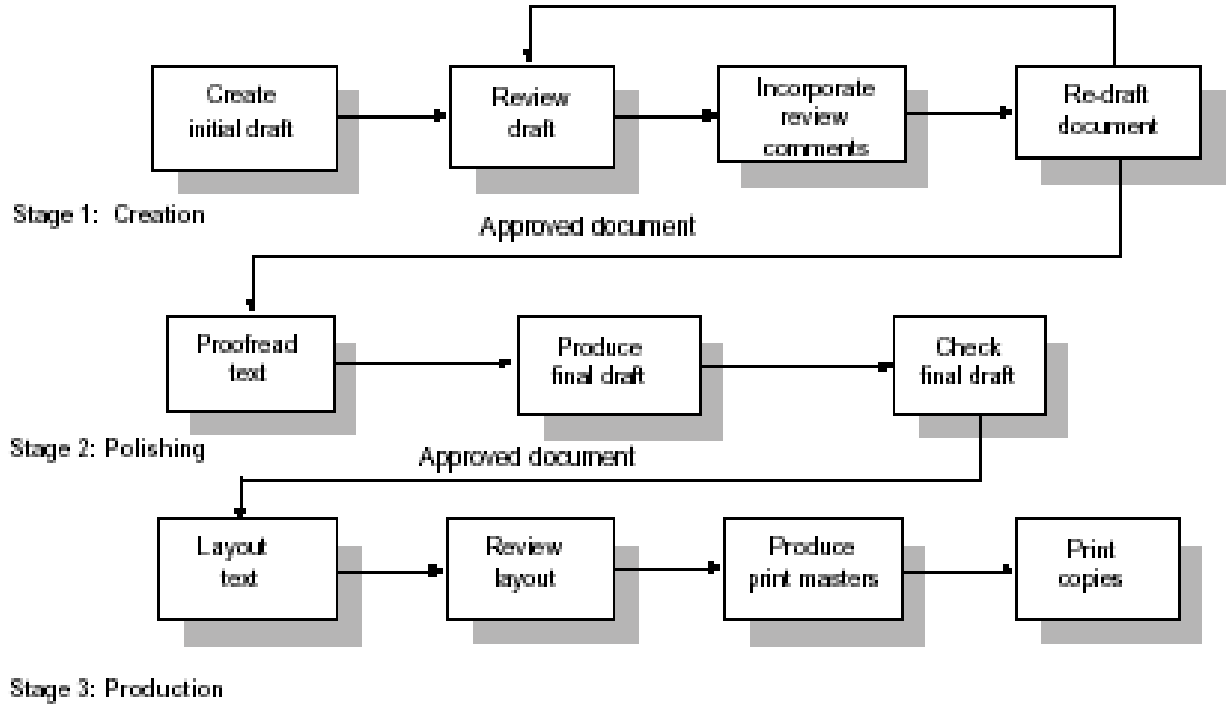


Figure I.30. Etapes de préparation d'un document **[Sommerville01]**

I.2.3. Contraintes et inconvénients de la documentation classique

Les méthodes de documentation classiques (telle que la méthode de Sommerville) sont complètes du point de vu de couverture de tous les besoins en matière de documentation ; mais sont lourdes à manipuler et consommatrices de temps et d'effort.

Dans ce qui suit, des points négatifs de la documentation classique sont donnés :

- Dans la majorité des projets, une méthode de documentation classique nécessite une équipe dédiée pour pouvoir être mise en place. Ainsi, une grande proportion du cout du procédé de développement est induite par la production de documentation.

- Les documents générés par ces méthodes sont nombreux et volumineux. Les erreurs et les incohérences de la documentation sont ainsi multiples, ce qui induit l'utilisateur du logiciel en erreur et par conséquent provoque tous les coûts et perturbations associés. Ambler a bien exprimé cette idée **[Ambler02]** « que préférez vous, une documentation de 2000 pages ayant probablement un nombre significatif d'erreurs, ou 20 pages de vue d'ensemble du logiciel ? ».

- La mise à jour de la documentation en même temps que la mise à jour du code est souvent négligée dans les méthodes classiques au profit du respect des délais, et la mise à jour de la documentation après plusieurs modifications du code devient de plus en plus difficile **[Sommerville01]**. En effet, une étude a montré que dans 68% des cas,

la documentation est toujours ou presque toujours en retard par rapport au code qu'elle représente **[Forward02]**.

- Beaucoup de documents sont requis dans un projet classique juste parce qu'un décideur (stockholder) veut se montrer dans une position de contrôle, justifier sa participation dans le projet ou tout simplement parce qu'il ne connaît pas un meilleur moyen (plus souple que la documentation) pour obtenir des explications **[Ambler09]**.

- Le procédé ou la norme appliqué lors d'un projet classique suggère la création d'un ensemble standard de documents ; sans vérifier pour chaque document sa valeur ajoutée et son utilité par rapport au projet en cours.

- Certains documents sont exigés dans une méthode classique juste pour le but de communiquer avec les différentes équipes du projet. Or, la documentation n'est pas le seul moyen de communiquer, d'autre outils doivent être privilégiés (les outils de travail collaboratif, les réunions face à face, les conférences vidéo ou téléphoniques…) **[Ambler09]**.

- Généralement dans les projets classiques, les décideurs ne sont pas conscients du coût total de propriété de la documentation (Total Cost of Ownership TCO) ; donc ils ne pensent pas à la valeur du document par rapport à l'effort fournit pour le produire et surtout le maintenir.

- Enfin, la documentation classique ne convient pas aux procédés logiciels qui favorisent le changement et livrent très rapidement (méthodes incrémentales et itératives). En effet, ces méthodes livre la documentation à chaque fin d'itération, cette documentation doit être synchronisée avec le logiciel. De plus, ces méthodes traitent de nombreux changements qui doivent être reflétés sur la documentation. Donc la documentation classique doit être optimisée pour convenir à ce type de méthodes **[Nasution09]**.

I.2.4. Synthèse

La documentation de logiciels est incontournable et est aussi importante que le logiciel lui-même. Elle permet de rendre l'information disponible et de pérenniser la connaissance pour des utilisations futures. Toutefois, trop de documents est aussi néfaste que peu de documents, puisque l'utilisateur se perd dans cette abondance et l'information recherchée devient de plus en plus difficile à retrouver.

Les approches de documentation classiques sont complètes mais lourdes, et souvent des documents sont créés pour des causes non pertinentes. Cela influe négativement sur la documentation produite : documents non cohérents, difficilement mis à jour et difficilement exploitables.

Il serait meilleur de trouver une approche qui permet d'avoir moins de documents tout en couvrant tous les besoins en matière de documentation, et qui rend facile la création et la gestion ainsi que l'exploitation de ces documents. Il serai intéressant aussi que cette approche puisse s'adapter aux méthodes incrémentales et itératives, qui sont actuellement en pleine expansion.

I.3. Documentation Agile

I.3.1. Méthodes Agiles

I.3.1.1. Définition

Une méthode Agile est une approche de développement de logiciels itérative et incrémentale, menée dans un esprit collaboratif avec juste ce qu'il faut comme formalisme. Elle génère un produit de haute qualité, tout en prenant en compte l'évolution des besoins des clients **[Rota07]**.

Les méthodes Agiles sont des procédés de développement de logiciels qui se veulent plus pragmatiques que les méthodes traditionnelles. En impliquant au maximum le client, ces méthodes permettent une grande réactivité à ses demandes, et visent la satisfaction réelle de son besoin et non les termes du contrat de développement. Les méthodes Agiles sont comme leur nom l'indique flexibles et ajustables **[Armour04],** et un de leur points forts est leur capacité de traiter le changement avec efficacité.

Les méthodes Agiles visent à réduire le cycle de développement du logiciel en développant une version minimale, puis en intégrant les fonctionnalités par un processus itératif basé sur une écoute client et des tests tout au long du cycle de développement. Le changement est une chose qu'il faut gérer au lieu d'éviter **[Karlesky08]**. Dans un environnement Agile, le changement qui rend le logiciel meilleur même en fin d'itération, est bienvenu et est encouragé **[Berry07]**.

Selon Boehm **[Boehm03]**, un procédé est perçu comme Agile s'il présente tous les attributs requis pour l'agilité, à savoir :

- *Incrémental* : les développeurs ne livrent pas tout le logiciel (toutes les fonctionnalités) en une seule fois.
- *Itératif* : le développement se fait à travers plusieurs cycles.
- *Auto organisé* : c'est l'équipe de développement qui détermine la meilleure façon de planifier et gérer le travail demandé.
- *Emergent* : la structure du travail est retrouvée en cours du projet, au lieu d'être prédéterminée.

La notion de méthode Agile a été officialisée en 2001 par un document *Agile Manifesto* **[Manifeste01]**. Les méthodes Agiles étaient antérieures au Manifeste Agile, qui n'est que la formalisation consensuelle de ces méthodes par leurs auteurs, du fait qu'elles avaient (les méthodes) des valeurs communes, une structure commune et une base de pratiques communes. Grâce aux méthodes Agiles, le client est pilote à part entière de son projet et obtient très vite une première mise en production de son logiciel, ce qui lui permet d'associer les utilisateurs dès le début du projet. Le Manifeste Agile **[Manifeste01]** distingue 4 valeurs fondamentales et 12 principes.

Valeurs :

1. Individus et interactions plutôt que processus et outils.
2. Développement logiciel plutôt que documentation exhaustive.
3. Collaboration avec le client plutôt que négociation contractuelle.
4. Ouverture au changement plutôt que suivi d'un plan rigide.

Principes :

1. « Notre première priorité est de satisfaire le client, en livrant tôt et régulièrement des logiciels utiles ».
2. « Le changement est bienvenu, même tardivement dans le développement. Les processus agiles exploitent le changement comme un avantage compétitif pour le client ».
3. « Livrer fréquemment une application fonctionnelle toutes les deux semaines à deux mois, avec une tendance pour la période la plus courte ».
4. « Les gens du métier et les développeurs doivent collaborer quotidiennement au projet ».
5. « Bâtissez le projet autour de personnes motivées. Donnez leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail ».
6. « La méthode la plus efficace pour transmettre l'information est une conversation en face à face ».
7. « Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet ».
8. « Les processus agiles promeuvent un rythme de développement soutenable. Commanditaires, développeurs et utilisateurs doivent pouvoir maintenir le rythme indéfiniment ».
9. « Une attention continue à l'excellence technique et à la qualité de la conception améliore l'agilité ».
10. « La simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle ».
11. « Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto-organisent ».
12. « À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens ».

I.3.1.2. Intérêts

La dynamique du marché, les nouvelles technologies ainsi que les besoins changeants des clients génèrent plus d'exigences sur le produit développé. Des versions du produit doivent être développées et gérées sur des courtes itérations, tout en répondant au changement externe rapide et en maintenant un haut niveau de qualité. Les méthodes Agiles offrent une formidable voie pour gérer des cycles de développement rapides **[Pikkarainen05]**.

L'essor des méthodes Agiles souligne l'intérêt des approches incrémentales. Ces dernières ont gagné beaucoup de partisans, et beaucoup d'entreprises ont commencé à les inclure dans leurs procédés de développement logiciel **[Nerur07]**. Les projets utilisant les méthodes Agiles sont entrain de rapporter des améliorations dans le temps et le coût, par rapport à ceux utilisant des méthodes classiques **[Charvat03]**. Généralement, le feedback en provenance de ces projets est positif **[ITM08]**.

Shine Technology a conduit une enquête, pour évaluer l'intérêt porté par les organisations de développement logiciel sur les méthodes Agiles **[Shine03]**. 131 réponses ont été reçues de différents coins du monde, dans lesquelles la réactivité au changement était identifiée comme une caractéristique positive des méthodes Agiles.

Une enquête (Agile Adoption Rate Survey) menée avec 642 participants **[AARS08]** a révèlé entre autres que : 69% des participants à l'enquête indiquent que leurs organismes adoptent une ou plusieurs méthodes Agiles ; et que 15% des organisations qui n'ont pas encore adopté une méthode Agile, prévoient de le faire dans l'année prochaine (ie 2009).
L'enquête évalue l'efficacité des méthodes Agiles par rapport aux méthodes traditionnelles comme suit :

% Réponses que l'efficacité des méthodes Agiles par rapport aux méthodes classiques est :	Assez élevée ou très élevée	Assez basse, très basse ou inchangée
Productivité	82%	18%
Qualité	77%	23%
Satisfaction client	78%	22%

Table I.1. Efficacité des méthodes Agiles vs les méthodes classiques

En 2009, 125 personnes s'intéressants à l'Agilité ont participé au State of IT Union Survey **[ITUnion09]**. 76% des répondants ont affirmé que l'Agilité était entrée dans leurs entreprises.

Les méthodes Agiles offrent un retour sur investissement (ROI) rapide, avec un investissement inférieur et un gain supérieur par rapport aux méthodes classiques, comme le montre la figure I.31.

L'Agilité modifie donc la façon de concevoir des produits et d'envisager un projet informatique, notamment en termes d'estimation, de planification et de suivi. La gestion de projets Agiles a pour objectif de développer le logiciel en se basant sur les exigences du client, de manière itérative, simple et basée sur l'expérience de l'équipe **[Highsmith04]**. Les pratiques intégrables et ajustables, ainsi que la remise au premier plan du facteur humain sont la vraie richesse des Méthodes Agiles **[Grosjean07].**

L'Agilité c'est l'efficience au futur immédiat **[Vickoff07]**; elle présente une approche meilleure étant réactive au changement, que les méthodes traditionnelles qui sont prédictives ; car il ne s'agit plus seulement de déterminer les besoins du client au début du projet, mais de savoir distinguer quand ces derniers évoluent.

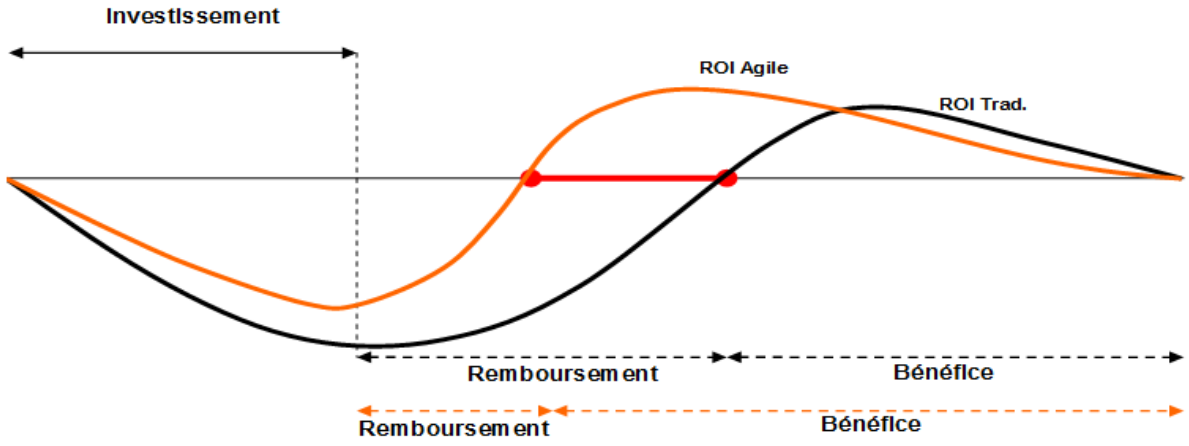


Figure I.31. ROI des méthodes Agiles et des méthodes traditionnelles **[Oldani06]**

I.3.1.3. Lacunes

Un des principaux écueils qui peuvent être rencontrés quand on décide de mettre en place ce type de méthodes, est que la frontière est parfois ténue entre l'application d'une méthode Agile, et le « n'importe quoi » qui s'installe quand on laisse travailler une équipe sans méthode prédictive et figée (une méthode traditionnelle).

Beaucoup de gens n'y voient dans l'agilité qu'une structuration des plus mauvaises pratiques du monde de l'entreprise :

- La réactivité, c'est lorsque le client change sans arrêt d'avis, et qu'on s'efforce de le suivre comme une girouette.
- La mise en place d'une équipe réduite, c'est quand on n'a pas assez de développeurs et qu'on leur donne plus de travail à faire que ce dont ils sont capables.
- Les cycles de développement ultra-rapides, c'est quand on n'a pas le temps de tester correctement les nouvelles versions, et qu'on est obligé de faire des versions de correction en urgence quand le client découvre des bugs.
- Sans parler des développeurs incapables d'écrire la moindre documentation, des chefs de projet qui ne veulent pas faire de planning, et des outils de suivi inexistants...

Les méthodes Agiles ou plus simplement les processus itératifs ne sont pas synonymes d'anarchie, où chacun fait les choses à sa sauce dans son coin. Mettre en place une méthode Agile, nécessite que tous les acteurs soient impliqués dans le processus, connaissent les avantages et acceptent de jouer le jeu. Toutefois, certaines personnes n'ont pas la flexibilité d'esprit nécessaire pour y arriver, et ont besoin de méthodes plus directives.

I.3.2. Définition et intérêt de la documentation Agile

La documentation Agile est comme son nom l'indique une documentation souple, qui convient aux procédés Agiles.

Etre Agile c'est penser communication, feedback, réactivité et adaptation plutôt que lourdeur, lenteur et « bureaucratie ». Donc produire une documentation Agile c'est documenter de façon intelligente, appropriée et précise en s'appuyant sur ce dont on a réellement besoin **[Grosjean07b]**.

Ainsi, une méthode de documentation qui convient aux projets Agiles suggère de produire juste ce qu'il faut comme documents, au bon moment et pour une audience ciblée **[Aguiar09]** ; ce qui résulte en une documentation qui est elle-même Agile.

L'interrogation qui se pose autour de la documentation Agile concerne son intérêt, puisque le manifeste agile favorise un logiciel qui marche à la documentation.

« Agilité rime souvent avec absence de documentation » : voilà une idée reçue bien nuisible qui doit être combattue avec force, car mener un projet en utilisant les méthodes Agiles n'a jamais signifié ne produire aucune documentation **[Grosjean07b]**. A l'origine de cette confusion une mauvaise interprétation de l'une des quatre valeurs du manifeste Agile **[Manifeste01]** « *un logiciel fonctionnel plutôt qu'une documentation complète* ».

Privilégier un logiciel qui marche plutôt que la documentation est un principe Agile de base, car il est plus efficace d'avoir un code qui fonctionne et qui peut être objectivement évalué, que d'avoir une documentation volumineuse et non à jour qui le décrit **[Forward02]**. Mais aujourd'hui, peu de projets peuvent se passer d'une documentation précise et adaptée **[Grosjean07b]**, et les connaisseurs de l'industrie du logiciel savent bien que les arguments disant que la documentation est superflue sont faux. D'ailleurs, l'industrie est pleine d'exemples de codes hérités sans documentation, que personne ne peut comprendre **[Forward02]**.

Linda Rising **[Rising03]** dit à ce propos: « rien dans le manifeste Agile ne déclare qu'aucune documentation ne doit être rédigée, mais à partir du moment où la majorité des développeurs ont une réticence de toute formes de rédaction (en dehors du code source), ils ont adopté ce principe Agile (*un logiciel fonctionnel plutôt qu'une documentation complète*) et ont proclamé au monde que la documentation est démodée ».

En effet, un document représente un enregistrement permanent de l'information **[Ambler07]**. Le manque de documentation peut conduire à des problèmes tels que dépenser beaucoup de temps à répondre aux mêmes questions, rencontrer les mêmes problèmes sans se rappeler la solution, ne pas s'échanger de connaissances entre les équipes, perdre une importante connaissance avec le départ des membres de l'équipe **[Holz03]** ; et peut conduire à la perte de la mémoire de l'entreprise **[Turk02]**.

I.3.3. Spécificités de la documentation Agile

La documentation dans les projets Agiles se base sur le principe « *Traveling Light* » **[Ambler02b]**, qui signifie créer juste le minimum de documents que requiert le bon fonctionnement du projet et du logiciel développé.

Highsmith a évoqué cette idée bien avant Ambler, et pour l'expliquer il a fait une analogie avec les randonnées **[Highsmith99]** : « Imaginez traverser un désert avec insuffisamment d'eau, vous partez trop léger ; ou imaginez traverser le même désert avec un sac de 100 livres, là vous partez trop lourd. Maintenant imaginez la construction d'une application d'e-commerce sans fournir aucune documentation décrivant comment la faire fonctionner, votre projet échouera car vous partez trop léger. Maintenant imaginez la construction de la même application avec des milliers de pages de documentation que vous devez mettre à jour et valider à chaque changement porté sur l'application, vous échouerez encore car vous partez trop lourd. »

Du coup, dans les projets Agiles les spécificités suivantes s'imposent sur la documentation **[Ambler09]** :

- Les documents Agiles sont assez simples et contiennent juste suffisamment d'information pour remplir leurs objectifs.

- Un document Agile traite un seul sujet bien précis. Si l'objectif du document n'est pas clair ou est contestable, alors il faudra repenser la création de ce document.

- Un document Agile décrit les *bonnes informations*, celles qui sont critiques ou non évidentes. Par exemple, une documentation indiquant que la colonne Name de la table Customer représente le nom du client, ne fournit pas une information de valeur. Par contre une documentation indiquant que la table Customer ne contient pas d'informations sur les clients habitant au Yukon (Canada), car ces informations sont sauvegardées dans un fichier ASCII plat pour des raisons réglementaires, est une *bonne information*.

- Un document Agile a un lecteur ciblé pour lequel il facilite le travail. Ainsi, il faut œuvrer à capturer et communiquer l'information à ce lecteur en utilisant le meilleur moyen.

- Un document Agile n'a pas besoin d'être parfait, il a juste besoin d'être assez bien en étant précis et cohérent. Une fois qu'un document Agile a atteint cet objectif, tout effort supplémentaire investit en lui est un effort inutile.

- Une règle empirique dans les projets Agiles est de ne produire aucun document jusqu'a en avoir réellement besoin, et ne documenter qu'une fois l'information est stable, pour éviter de produire un document qui s'avèrera plus tard inutile ou qui devra être changé.

- Dans un projet Agile, ne documenter que si c'est le seul moyen de communiquer. La documentation doit rester le dernier recourt au lieu d'être le choix préférable.

- Une mise à jour de documents ne doit être opérée dans un projet Agile que si elle s'avère absolument nécessaire « Update only when it hurts », sinon préserver l'effort si le document n'étant pas à jour ne pose pas un problème (certains documents peuvent être utiles même s'ils sont pas à jour **[Forward02]**).

- Il faut faire collaborer le développeur et le rédacteur technique pour rédiger de la documentation Agile, car les développeurs maitrisent le sujet rédigé (ils ont la connaissance) alors que les rédacteurs techniques ont la compétence de rédaction.

- Il faut faire un compromis sur le niveau d'expérience des développeurs impliqués dans la rédaction d'une documentation Agile ; car un développeur non expérimenté rédige beaucoup d'informations superflues, par contre un développeur très expérimenté rédige très peu d'informations (étant très occupé par d'autres taches ou trouvant beaucoup d'informations évidentes alors qu'elles ne le sont pas).

I.3.4. Difficultés de la documentation Agile

Le challenge dans les méthodes Agiles est que la quantité et les types de documents diffèrent d'un projet à un autre (cela dépend du type du logiciel développé, de sa durée de vie prévue, des exigences du client en matière de documentation, et du planning prévu pour le projet) **[Cockburn05]**, et que le besoin en documentation n'est pas connu avec exactitude à priori.

Aussi, le logiciel est livré à chaque fin d'itération et ceci inclut la documentation qui l'accompagne. De plus, toute documentation rédigée doit évoluer dans le temps et être synchronisée avec le logiciel.

En réalité la plus grande difficulté de la documentation Agile, est qu'il n'existe pas un procédé complet pour le développement de documents dans les projets Agiles. Les auteurs des méthodes Agiles n'ont pas été très précis à propos de la documentation, ils se sont contentés

de donner des idées atour du sujet, sans donner de procédures concrètes pour produire de la documentation Agile. Par exepmle :

- Cockburn dans son ouvrage *Agile Software Development* **[Cockburn01]** introduit la famille des méthodes Crystal, utilisées par des projets de différentes tailles et de différentes criticités. Les méthodes Crystal requièrent la création d'une documentation, mais laissent chaque projet décider de quoi sera constituée cette documentation.
- Ambler dans son livre *Agile Modeling* **[Ambler02b]** inclut un chapitre entièrement dédié à la documentation. Pour parler de l'approche de documentation Agile Ambler utilise la métaphore « Traveling Light », qui signifie créer juste le minimum de documentation permettant au projet d'avancer.
- Highsmith dans son livre *Agile Software Developpement Ecosystems* **[Highsmith02]** avertit de ne pas produire la documentation juste pour documenter, mais plus tôt de produire une documentation modérée qui facilite la communication, améliore le transfert de connaissances et préserve l'historique.
- L'idée véhiculée par la méthode XP **[XP]** concernant la documentation est la suivante : « *agile models are paintings, not photographs* » ne pas tout décrire et avec détails (comme dans une photographie), par contre ne parler que d'idées importantes avec juste ce qu'il faut comme informations (comme dans une peinture).
- La méthode FDD **[FDD]** contient cinq procédés, chacun est constitué de plusieurs taches obligatoires ou optionnelles. Certaines taches ont pour objectif de produire des documents, ce qui sous entend que la méthode préconise un ensemble de documents obligatoires ou optionnels à produire. Toutefois, les documents générés par la méthode se limitent à ceux requis pour l'avancement du projet en cours, et aucune règle n'est imposée pour les autres types de documents.
- Dans la méthode RAD **[RAD]** on préconise une multitude de dossiers à fournir lors d'un projet, ainsi que les documents qui les constituent ; mais sans préciser quels sont les documents obligatoires et ceux optionnels. Ceci rappelle les méthodes de documentation classiques qui recommandent un maximum de documents (pour couvrir tous les besoins du projet) mais qui sont difficiles à gérer et à exploiter.

De plus, des problèmes récurrents restent posés dans la documentation Agile, qui sont inhérents à la documentation de logiciels et dont quelques uns sont cités dans ce qui suit :

- Avec des documents nombreux et volumineux la recherche d'une information précise est compliquée, de plus la probabilité que cette information ne soit pas à jour est forte **[Komulainen06] [Lethbridge03]**. Il est nécessaire de pouvoir aboutir à une documentation concise et à jour.
- On rencontre souvent dans les projets des documents mal structurés, et ne couvrants pas toutes les informations requises pour un thème particulier **[Kylmakoski03] [Lethbridge03] [Ruping03]**. Il faudra non seulement accumuler toutes les informations concernant un thème particulier, mais aussi les structurer de manière adéquate dans un document.

- Le coût élevé de la documentation est causé principalement par l'effort consommé par cette activité lorsqu'elle est gérée par un procédé anarchique **[Komulainen06] [Kylmakoski03] [Lethbridge03]**. Ce cout doit être réduit en intervenant principalement sur le procédé de documentation.
- La gestion des versions et l'archivage de la documentation **[Ruping03]**. Il s'agit d'une activité qui doit absolument être gérée dans un procédé de documentation.
- Les documents nombreux et volumineux produits lors d'un projet (documentation exhaustive) causent un problème, qui est la difficulté de gérer et d'exploiter ces documents **[Boehm03]**. Il est indispensable dans un projet Agile d'avoir une quantité de documents optimale, qui permet de couvrir les besoins en documentation tout en étant facile à gérer et à exploiter.

I.3.5. Synthèse

Les méthodes Agiles sont de plus en plus adoptées et la documentation Agile est par conséquent de plus en plus requise. La valeur du manifeste Agile qui préconise « un logiciel fonctionnel plus tôt qu'une documentation complète » ne signifie pas absence de documentation. C'est souvent une mauvaise interprétation de cette valeur qui motive certaines équipes à ne pas documenter, mais aussi un manque de volonté (voir de compétence) de certaines équipes souvent soumises à un timing toujours plus serré.

La documentation Agile est donc une documentation qui est elle-même Agile et qui convient aux projets Agiles. Du fait, des moyens qui permettent de créer et gérer la documentation Agile s'avèrent nécessaires. Cependant, quelques problèmes restent posés dans le domaine de la documentation Agile, et des solutions doivent venir en aide pour mettre efficacement en œuvre cette approche de documentation.

I.4. Conclusion

Ce premier chapitre a donné un aperçu sur les patrons logiciels, ainsi que sur la documentation de logiciels, dans les projets classiques comme dans les projets Agiles.

L'état de l'art sur les patrons logiciels présenté dans ce chapitre n'est pas exhaustif, il expose les travaux les plus riches s'intéressant aux patrons logiciels. A travers cet état de l'art, on peut constater que les patrons représentent une issue prometteuse.

Un patron représente une approche structurée d'une solution, dont il est prouvé de rapporter des résultats efficaces. Il permet la description d'une solution comme un élément réutilisable correspondant à un problème spécifique, et capable d'être adapté pour plusieurs situations.

Toutefois, l'approche *Patrons* reste encore peu exploitée en pratique, et les travaux dans ce domaine sont encore modestes. De plus, la plupart des travaux existants sont de nature académique et ne visent pas une application (rapide) dans l'industrie.

Les patrons logiciels seront explorés dans les chapitres qui suivent, où une contribution sera apportée à la composition de patrons qui permet à ces derniers d'exprimer pleinement leur intérêts.

Un aspect important des projets logiciels est montré dans ce chapitre qui est la documentation de logiciels. Avec la multitude d'intérêts qu'offre cette documentation, on a tendance à créer un maximum de documents dans un projet pour tirer un maximum de profit. Sauf que l'expérience a montré qu'un maximum de documents est aussi néfaste pour un projet qu'un manque de documents, vu que cette multitude de documentation influe négativement sur la qualité de cette dernière et sur la facilité de son exploitation.

Ainsi, l'intérêt est porté sur la documentation Agile, pour palier aux lacunes de la documentation classique mais aussi pour prendre en charge les contraintes des projets Agiles. Un projet Agile préfère une documentation légère, de bonne qualité (précise, à jour, concise et bien structurée) et adaptée aux besoins particuliers des projets Agiles.

Cependant, des problèmes restent posés dans le domaine de la documentation Agile, entre autres le problème d'exhaustivité de la documentation Agile qui sera abordé dans les chapitres qui suivent, où une solution sera conçue pour le résoudre.

CHAPITRE II :

EXHAUSTIVITE DE LA DOCUMENTATION AGILE ET COMPOSITION DE PATRONS

Ce chapitre détaille dans un premier lieu le problème d'exhaustivité de la documentation dans les projets Agiles**,** et l'outil choisi pour sa résolution **(cf. § II.1)**. Ensuite la composition de patrons est abordée, et une méthode de traitement automatique des relations entre patrons est choisie pour ce travail et est analysée **(cf. § II.2)**. Troisièmement, les principaux problèmes qui seront résolus dans ce travail sont résumés **(cf. § II.3)**. Enfin, une conclusion récapitulant le chapitre est donnée **(cf. § II.4)**.

II.1. Problème d'Exhaustivité de la Documentation dans les projets Agiles

II.1.1. Explication du problème

II.1.1.1. Définition du problème et exemples de documentation optimale

Un problème important qui revient dans les projets Agiles est le problème *d'exhaustivité de la documentation* (Completeness of documentation) **[Boehm03]**. Comme la documentation est d'une utilité immense pour un projet logiciel (cf. I.2.1 et I.3.2), on a tendance à croire que cette utilité croit avec la quantité de documents. Ainsi, les approches de documentation classiques (non Agiles) définissent une multitude de documents à créer, couvrants tous les aspects du projet. Malheureusement, ces approches engendrent un autre problème dû au grand nombre de documents générés, sans que ces derniers ne soient cohérents et à jour. Ceci complique la recherche d'un renseignement dans ces documents et les rend d'une utilité médiocre pour leurs exploitants (cf. I.2.3) ; de plus que ça ne convient pas aux projets Agiles (cf. I.3.1 et I.3.4).

Les études sur la quantité de documentation produite dans les projets ont montré que peu de documentation ou trop de documentation dégrade l'utilité de cette dernière, comme le montre la figure suivante ; alors qu'un nombre étudié (optimal) de documents permet de tirer pleinement profit de ces derniers **[Ruping03]**.

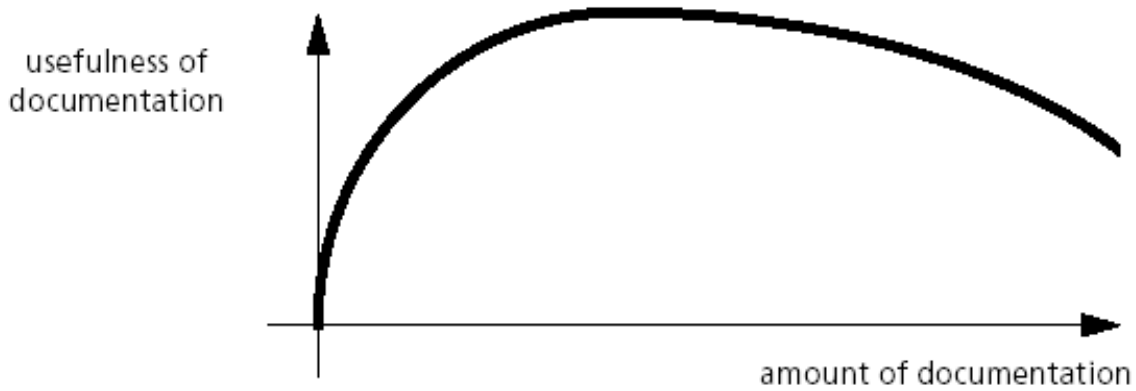


Figure II.1. Utilité de la documentation en fonction de sa quantité **[Ruping03]**

C'est cette quantité optimale de documentation (maximale pour couvrir tous les besoins et minimale pour être gérée et exploitée avec aisance) qu'il est intéressant d'avoir dans un projet Agile, et c'est cette quantité qui est ciblée par le problème d'exhaustivité de la documentation dans les projets Agiles.

Sont donnés dans ce qui suit quelques exemples de documentation optimale dans divers projets. Ces exemples montrent combien cette documentation diffère d'un projet à un autre. Par conséquent, la résolution du problème d'exhaustivité de la documentation ne consiste pas en la proposition d'une liste standard de documents à produire, qui soit valable

pour tous les projets ; mais plus tôt de proposer un moyen de décider d'une documentation optimale propre à chaque projet.

Exemples de projets nécessitants un minimum de documents [Ruping03] :

- Projet Paracelsus *(exemple avec un minimum de documents)* : Dans ce petit projet la tache était claire depuis le début : le client développait un framework, et avait besoin de certains composants de transformation de données pour les intégrer dans son framework. Une collaboration étroite entre l'équipe et le client devenait naturelle, et ces derniers ont décidé dés le début du projet qu'une quantité minimale de documentation allait être suffisante.

  Le document de spécification des besoins contenait essentiellement quelques notes prises durant les workshops, où le client expliquait ce que les composants sont supposés faire. Simultanément avec les phases de conception et de codage, l'équipe produisait un document de conception et un document des concepts d'utilisation (usage concepts). Le papier de conception documentait les principales idées de conception des composants, et était mis à disposition du client comme entrée d'un workshop, où la compatibilité entre la conception des composants et la conception du framework était vérifiée avant le début de la phase de codage. Le papier des concepts d'utilisation fournit des informations sur l'utilisation des composants, par exemple l'appelle des composants, le paramétrage de ces derniers … ; ce papier sera principalement utilisé par le client lors de l'intégration des composants à son framework.

- *Projet AirView (exemple avec un document de spécification des besoins très réduit) :* L'objectif du projet était le développement d'une nouvelle interface graphique pour les utilisateurs. Donc une importante tache était de spécifier à quoi cette interface devait ressembler.

  Toutefois, l'équipe et le client avaient décidé que le document de spécification des besoins ne devait pas inclure une description détaillée de l'interface (les détails visuels de l'interface étaient intentionnellement abandonnés). Par contre, le document devait définir les cas d'utilisation que l'interface aller satisfaire. Pour décrire la géométrie de l'interface, l'équipe avait opté pour un prototype qui avait été remis au client pour revus. Le prototype pouvait être adapté rapidement et s'est avéré meilleur qu'un document.

Exemples de projets nécessitants plus de documents [Ruping03] :

- *Projet FlexiCar (exemple avec un document de conception très détaillé):* Lors du démarrage du projet, le client avait une idée très précise sur l'optimisation du système de gestion de son procédé métier (fabrication de voitures). L'équipe était petite à cette époque, et avait produit un document qui résumait les besoins du client et donnait une esquisse de l'architecture du nouveau système.

Plus tard, un autre document s'était avéré nécessaire, et avait été produit pour décrire l'architecture du nouveau système avec plus de détails, et avait été raffiné à mesure que le projet progressait. Ce document servait deux objectifs, d'une part il était utilisé pour communiquer les principes de l'architecture à toute l'équipe (à un certain stade cette dernière comptait plus de 50 personnes et ne pouvait plus compter sur la communication verbale) ; d'une autre part, le document aller être utilisé dans le futur par le client pour la maintenance du système.

- *Projet Extricate (exemple avec plusieurs documents d'analyse des besoins) :* Ce projet soulevait deux challenges, premièrement il faisait participer plusieurs intervenants à savoir : l'équipe de développent, des ingénieurs logiciels du client et des experts métier du client aussi ; deuxièmement, comme il s'agissait d'un projet de réingénierie l'équipe devait d'abord se familiariser avec l'ancien système ainsi que son domaine d'application (métier), les fonctionnalités du système n'allaient pas changer et l'objectif était juste d'avoir un système plus flexible, mais l'équipe devait extraire à partir de l'ancien système des propriétés métier qui passaient inaperçues lors des discussions avec le client.

  Un document de spécifications des besoins s'avérait utile puisqu'il permettait de représenter ce que l'équipe avait comprit, et permettait au client de clarifier les malentendus. Ce document subissait plusieurs mises à jour après les discussions avec le client, et servait comme source fiable d'informations. Un autre document était particulièrement important, celui de la stratégie de migration qui nécessitait une étude à part entière. Ce document révélait les dépendances dans la migration des différents sous systèmes, et montrait le compromis entre la qualité du nouveau modèle de données et la complexité du procédé de migration.

- *Projet Persistor (exemple avec un maximum de documents) :* La documentation jouait un rôle important dans ce projet, parce que ce dernier impliquait plusieurs personnes de différents sites géographiquement éloignés, et parce que les contributions des différentes personnes devaient s'intégrer étroitement.

  Dés le lancement du projet l'équipe avait produit un document assez court spécifiant la couche d'accès aux données du framework développé. Ce document était validé par le client et servait comme base pour la conception. Avec l'avancement du projet, des besoins supplémentaires étaient apparus mais qui n'avaient jamais été documentés.

  Après un certain temps cela avait conduit à des conflits sur les fonctionnalités à implémenter et celles qui avaient été refusées, ce qui avait créé une forte tension entre les différents participants du projet. Ainsi, il avait été décidé de documenter les nouveaux besoins.

Un autre grand problème que le projet avait rencontré, était la formation du personnel des différents sites du client à l'intégration du nouveau framework dans leurs applications. Pour résoudre ce problème, l'équipe avait décider de mixer documentation et workshops ; ainsi un document de concepts d'utilisation (usage concepts) avait été produit expliquant comment configurer le framework, comment appeler ses méthodes d'interface et les consignes générales à suivre pour son utilisation. Le document avait été distribué sur tous les sites, avant de lancer les workshops qui complétaient les informations de ce document.

II.1.1.2. Division du problème

La résolution du problème d'exhaustivité de la documentation Agile, a pour objectif d'assurer la disponibilité de tous les documents nécessaires à un projet Agile tout en évitant la lourdeur, pour satisfaire les contraintes Agiles. Ainsi, pour appréhender le problème d'exhaustivité de la documentation, on peut le décomposer en les trois sous problèmes suivants :

- Comment fournir une documentation qui sera bien appropriée à ses destinataires. L'objectif de ce sous problème est de fournir une documentation dont l'exploitation sera facile (puisque appropriée à ses destinataires) ; mais aussi de limiter le nombre de documents à ceux ayants un destinataire bien précis. Ce sous problème correspond à la spécificité de la documentation Agile relatant qu'un document doit avoir un lecteur ciblé (cf. I.3.3).

- Comment s'épargner des besoins non nécessaires et des besoins dupliqués en matière de documentation. L'objectif de ce sous problème est de proposer une documentation personnalisée et adéquate pour chaque projet. Ceci reflète l'une des difficultés de la documentation Agile, à savoir le besoin en documentation qui change d'un projet à un autre et qui n'est pas connu avec exactitude à priori (cf. I.3.4).

- Comment fournir un contenu pertinent par document. L'objectif de ce sous problème est de limiter le contenu des documents aux informations utiles, ce qui donnera une quantité d'informations inferieure, et par conséquent l'exploitation de ces documents sera facile. Ce sous problème correspond aux spécificités de la documentation Agile (cf. I.3.3).

Tout cela pour s'assurer que la gestion des documents du projet sera facile, et que leurs exploitants trouveront aisément les renseignements recherchés.

II.1.2. Outil pour la résolution du problème

Un outil très puissant qui peut offrir une solution à ce problème, sont les patrons logiciels. Un patron est un modèle représentant une solution prouvée qui résout un problème récurrent. Il capture de la connaissance dont il est avéré de rapporter des résultats efficaces. Il peut être instancié dans plusieurs situations, et peut être enrichi si nécessaire par des informations détaillées supplémentaires.

L'utilisation de cet outil offre donc plusieurs profits. Principalement, cela permet de répondre au problème grâce à une solution prouvée, ainsi on gagne en qualité et en efficacité. De plus, les solutions dans les patrons sont abstraites, et donc peuvent être appropriées pour plusieurs situations conformes au contexte du patron. Aussi, on gagne en rapidité de conception de la solution lorsqu'on se base sur des patrons, par rapport à la conception d'une solution à partir de zéro.

Il est avantageux donc d'explorer des catalogues de patrons à la recherche d'une solution pour le problème d'exhaustivité de la documentation Agile.

II.1.3. Choix du catalogue de patrons

Lors de l'exploration des catalogues de patrons, le seul catalogue trouvé susceptible de fournir des patrons dans le sujet de la documentation Agile est le catalogue de Ruping **[Ruping03]**.

Les patrons de ce catalogue ont été observés à travers divers projets, et ont été revus lors de plusieurs workshops [**Ruping98a]** [**Ruping98b]** [**Ruping99a]** [**Ruping99b]**. Ces patrons gravitent autour de la question « Un projet peut il réellement suivre les principes Agiles tout en continuant à produire de la bonne documentation ? ».

Ainsi, le catalogue de Ruping est jugé approprié puisque c'est un catalogue spécialisé dans la documentation Agile, qui fournit un grand nombre de patrons traitants des aspects différents de cette documentation et qui est recommandé par plusieurs spécialistes (à l'instar de Scott Ambler du domaine de la documentation Agile, et Linda Rising du domaine des patrons).

Les patrons dans l'ouvrage de Ruping **[Ruping03]** adoptent le formalisme qui suit :

- *Problem* : chaque patron débute avec un bref énoncé du problème sous forme d'une question.
- *Forces* : contraintes influençant les solutions possibles au problème.
- *Solution* : apporte une résolution à la question posée dans la rubrique *Problem*.
- *Discussion* : donne des informations supplémentaires sur le patron, et décrit les relations de ce dernier avec d'autres patrons (principalement du même ouvrage).

Ce formalisme est avantageux puisqu'il présente les points forts suivants :

- La simplicité de ce formalisme constitue son principal point fort. Structuré en quatre rubriques seulement, il facilite l'exploitation de patrons ; mais aussi la tache des auteurs qui l'adoptent pour rédiger leurs patrons.

- Ce formalisme profite d'une signification évidente des rubriques, à la différence de certains formalismes qui proposent des rubriques dont le sens n'est pas clair, ce qui peut induire en erreur l'utilisateur du patron.

- Un aperçu de chaque patrons est donné à travers le premier paragraphe de la rubrique Solution et le contenu de la rubrique Problem. Cet aperçu permet de juger si le patron est approprié ou pas, sans avoir à consulter le patron dans sa totalité.

- Un point fort de ce formalisme est la présence de la rubrique Forces, qui est le cœur de chaque patron **[Buschmann07]**. Le problème d'un patron est toujours accompagné par un nombre de contraintes sur la solution à proposer. Ainsi, la rubrique Forces isole ces contraintes, indique pourquoi le problème traité est réellement un problème, pourquoi il est important ou difficile et pourquoi il nécessite une solution bien particulière.

- La description de la solution en langage naturel est un avantage car elle permet à l'auteur du patron d'exprimer toute information voulue, et ne demande à l'utilisateur du patron aucune connaissance supplémentaire pour la compréhension. Par exemple, l'utilisation des langages Little-Jil **[Wise06]** et BPMN **[BPMN10]** en plus du langage UML pour exprimer la solution des patrons de **[Staudt10],** est du au fait que chacun de ces langage est incapable tout seul d'exprimer la solution ; et l'exploitation de ces patrons va se limiter aux connaisseurs de ces langages, ou à ceux prêts à les apprendre.

- Enfin, les noms des patrons de Ruping sont évocateurs permettant d'avoir une idée sur l'objectif du patron. Cette manière de faire joint l'idée disant qu'un patron est mémorable si son nom évoque une image claire véhiculant l'essence de la solution **[Meszaros97]**.

Toutefois, on peut reprocher au formalisme de Ruping les lacunes suivantes :

- La rubrique Contexte joue un rôle important dans un patron en définissant la situation dans laquelle il est applicable. A défaut d'une rubrique Contexte précise, le patron devient un « all things to all people » avec des perspectives différentes et contradictoires **[Buschmann03]**. Malheureusement, dans le formalisme de Ruping le contexte d'application n'est pas donné de manière explicite.

- Malgré que la rubrique *Discussion* de ce formalisme donne les relations du patron avec d'autres patrons (principalement du même ouvrage), ces informations demeurent vagues (non explicites) et l'utilisateur du patron doit lui-même faire l'effort de découvrir avec précision les types de relations existantes entre les patrons en question.

- Enfin, la solution des patrons de Ruping est exprimée exclusivement en langage naturel. Cela peut produire des expressions ambigües voire inexactes, de plus qu'il est difficile d'exploiter la solution avec des outils (automatiques) de conception de solution. Ainsi, le besoin d'une représentation formelle supplémentaire émerge pour préciser la sémantique de la solution, et exprimer cette dernière de façon visuelle.

Le catalogue de Ruping sera donc exploré à la recherche de patrons, pour concevoir une solution au problème d'exhaustivité de la documentation Agile. Dans le cas où aucun patron n'est capable de résoudre le problème, on peut recourir à la composition de plusieurs patrons.

II.1.4. Une autre problématique observée

Alors que nous travaillons sur la résolution du problème su cité, nous avons observé l'utilité de la composition de patrons qui permet de résoudre des problèmes complexes non résolus par des patrons seuls, qui permet aux patrons d'être de plus en plus utilisés et qui donne un meilleur profit des connaissances prouvées procurées par les patrons.

La composition de patrons se base principalement sur les relations entre patrons. Ces dernières sont généralement explicites dans les patrons ; le cas contraire, elles doivent être analysées et ressorties pour pouvoir servir la composition de patrons, d'où la seconde problématique de notre travail présentée dans ce qui suit.

II.2. Problème de Composition de Patrons

II.2.1. Explication du problème

II.2.1.1. Définition et intérêt de la composition de patrons

Un patron résout un problème, mais lorsqu'on aborde un thème particulier dans la réalité jamais un problème n'est seul, facile à cadrer et indépendant. On est souvent confronté à un ensemble de problèmes liés, dont la résolution de l'un influence la résolution d'un autre. Ainsi on est amené à utiliser plusieurs patrons qui sont liés entre eux, où chacun résout une partie du problème, pour pouvoir résoudre le problème confronté dans sa totalité. Cette utilisation de plusieurs patrons participants pour résoudre un problème, en se basant sur les relations entre patrons, est la *Composition de Patrons*.

Les relations entre patrons sont à la base de la composition de ces derniers. Le but de ces relations est d'indiquer quels patrons peuvent fonctionner ensemble et de quelle manière, pour pouvoir former une solution.

Une fois ces relations obtenues, il ne restera qu'à réutiliser les patrons pour aboutir à une solution composée.

Beaucoup de patrons peuvent être composés et les relations entre eux sont citées, tels que :

- Le patron *Distinguish Identities* utilise les patrons *Component Proxy*, *Component Home* et *Managed Resource* **[Volter00]**.
- Le patron *Review* utilise trois autres patrons, à savoir le patron *Introductional Session*, le patron *Review Session* et le patron *Release* **[Hagen04a]**.
- Le patron *Role Attribute* pattern est utilisé dans le patron *Generic Element* **[XmlPatterns00]**.
- Le patron *Capture Vocabulary Centrally* et le patron *Capture Vocabulary Participatorily* spécialisent le patron *Capture A Common Vocabulary* **[Hagen04a]**.

- Selon **[Noble98]**, le patron *Iterator* **[GoF95]**, le patron *Type-safe Session* **[Pryce97]** ainsi que le patron *Accumulator* **[Yelland96]** tous raffinent le patron abstrait *Curried Object* **[Noble97]**.

- Le patron *Collection Element* **[XmlPatterns00]** specialize le patron *Container Element* **[XmlPatterns00]**.

II.2.1.2. Difficultés de la composition de patrons

La composition de patrons consiste d'abord à découvrir les relations entre patrons, pour pouvoir les organiser en solution complexe, ensuite réutiliser chacun de ces patron comme s'il s'agissait d'une simple exploitation d'un patron.

La partie la plus difficile de la composition de patrons est l'analyse des relations entre patrons. Ces relations sont difficiles à discerner si elles ne sont pas données explicitement dans chaque patron (dans une rubrique dédiée à expliciter les liens entre le patron en question et d'autres patrons). S'ajoute à cela que même les patrons contenant une rubrique pour les relations, se contentent de donner les liens entre le patron en question et des patrons du même catalogue ; sont très rares ceux indiquant des relations avec des patrons d'autres catalogues.

Par exemple, sur 170 collections de patrons examinées dans une étude **[Henninger07]**, un seul catalogue liste des références vers des patrons dans d'autres collections **[Snow06]**. Cela est du entre autres au fait que chaque catalogue définit ses propres relations, différentes de celles des autres catalogues. Mais aussi à cause de la non existence d'un moyen permettant de connaitre tous les patrons existants.

Conséquemment pour concevoir une solution complexe, il appartient au concepteur d'explorer les différentes collections de patrons à la recherches de patrons susceptibles de résoudre partiellement le problème, et de découvrir les relations existantes entre ces patrons pour pouvoir les combiner. Evidement ce procédé est onéreux, et il n'est pas évident d'aboutir facilement à de bons résultats ; ce qui décourage les concepteurs et les empêche d'exploiter des patrons.

Dans ce contexte le besoin a émergé d'avoir une méthode automatique, qui peut traiter un nombre quelconque de patrons, et qui analyse les relations entre patrons même si ces derniers sont hétérogènes (exprimés dans des formalismes différents) ou appartiennent à des catalogues différents.

La méthode de **[Kubo05]** et al. est la première approche automatique d'analyse des relations entre patrons, qui aborde les relation entre patrons de différents catalogues, et c'est la seule approche qui peut traiter des patrons logiciels de différentes natures. L'approche de Kubo et al. est présentée dans ce qui suit.

II.2.2. Méthode de Kubo et al.

II.2.2.1. Terminologie

Kubo et al. appellent document de patron (*pattern document*) le document dans lequel un patron est décrit. Dans un document de patron, les entêtes (*headings*) et les corps (*bodies*) apparaissent alternativement, l'entête représente le nom d'une rubrique alors que le corps représente le contenu de cette dernière. Une *section* est une combinaison d'une entête et d'un corps.

Notons H un ensemble d'entêtes, et B un ensemble de corps. Ainsi, une section s est défini comme suit :

$$s = (h,b),\ h \in H,\ b \in B.$$

Pour un ensemble de sections S, un document de patron d est défini comme suit :

$$d = \{s_1,s_2, \ldots, s_n\},\ s_1,s_2, \ldots, s_n \in S.$$

L'ensemble des entêtes qui constituent le patron représente la forme de ce dernier (*pattern form*). Ainsi, une forme de patron f est défini comme suit :

$$f = \{h_1,h_2, \ldots, h_m\},\ h_1,h_2, \ldots, h_m \in H.$$

La table suivante montre quelques formes de patrons:

Formes de patrons	***Ensemble d'entêtes***
Forme du GoF **[GoF95]**	f_{GoF} = { Pattern Name, Classification, Also Known As, Motivation, Applicability, Structure, Particiants, Collaborations, Consequences, Implementation, Sample Code, Known Uses, Related Pattern }
Forme de PoSA **[Stal96]**	f_{PoSA} = { Name, Also Known As, Examples, Context, Problem, Solution, Structure, Dynamics, Consequences, Implementation, Sample Code, Known Uses, Related Pattern }

Table II.1. Exemple de deux formes de patrons

II.2.2.2 Procédure d'analyse des relations

La procédure d'analyse de Kubo et al. est présentée dans la figure II.2. Elle cible les documents de patrons décrits en HTML, et les traite de la manière suivante :

Premièrement, les documents de patrons sont analysés et les sections sont extraites dans le bloc *HTML Analyser*. Ensuite, la forme du document de patron est jugée dans le bloc *Pattern Form Judgment*. Troisièmement, le patron (représenté dans le modèle de Kubo et al.) est obtenu dans le bloc *Pattern Extraction*, à partir des sections extraites et selon la forme jugée précédemment. Enfin, les relations entre patrons sont analysée dans le bloc *Relation Analysis*.

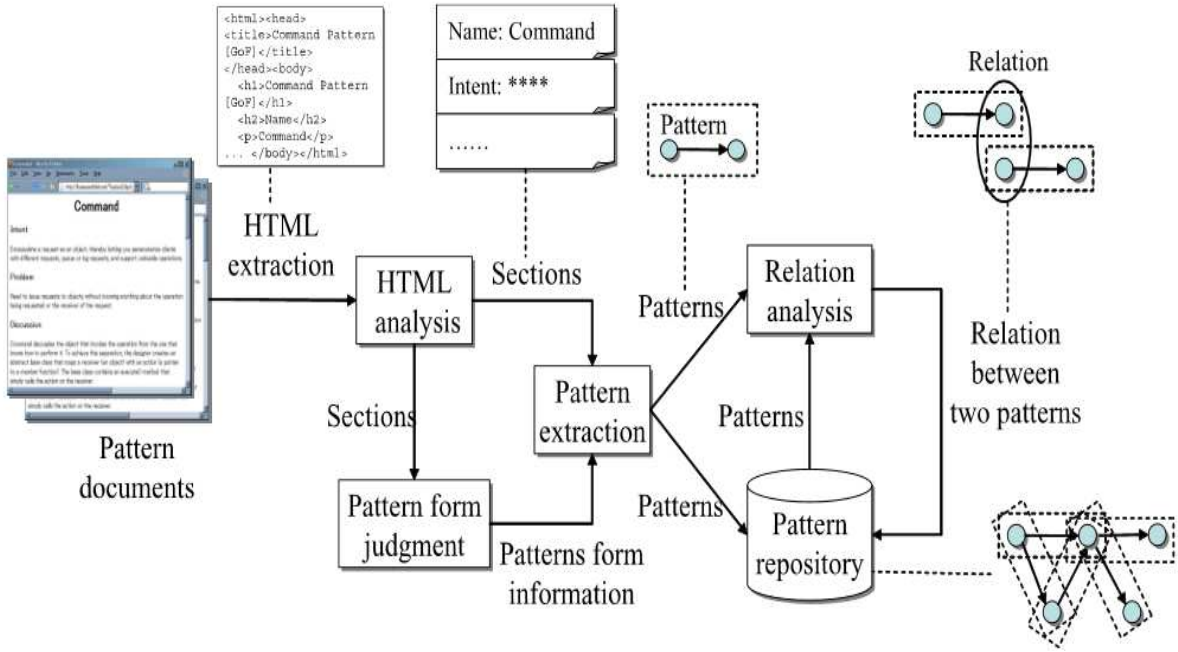


Figure II.2. Procédure d'analyse **[Washizaki05]**

II.2.2.3 Modèle de patrons

Kubo et al. représentent l'application d'un patron par une transition de contextes, qui part d'un contexte initial vers un contexte final. De plus, les contraintes du patron (*forces*) sont incluses dans la représentation du patron, car deux patrons qui différent seulement par les contraintes doivent être considérés comme deux patrons différents.

Ainsi, un patron est modélisé par un graphe orienté qui illustre le flux de l'application du patron. Le modèle est montré dans la figure II.3. Formellement, pour un ensemble de contextes $C = \{c_1, c_2, \ldots, c_n\}$, et Λ un ensemble de contraintes, un patron p est défini comme suit :

$$p = (c_i, c_j, \lambda_k),\ c_i, c_j \in C,\ i \neq j,\ \lambda_k \in \Lambda.$$

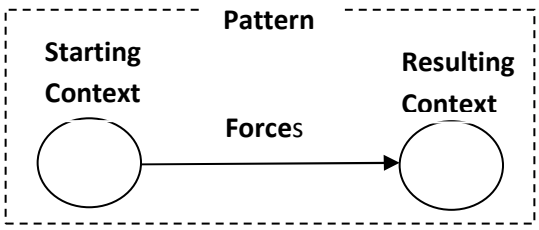


Figure II.3. Modèle de patrons

Un PRG (*Pattern Relation Graph*) montré dans la figure II.4 est un graphe illustrant un système (ensemble) de transitions de contextes. Il est défini comme suit, avec P un ensemble de patrons :

$$PRG = (C, P, \Lambda).$$

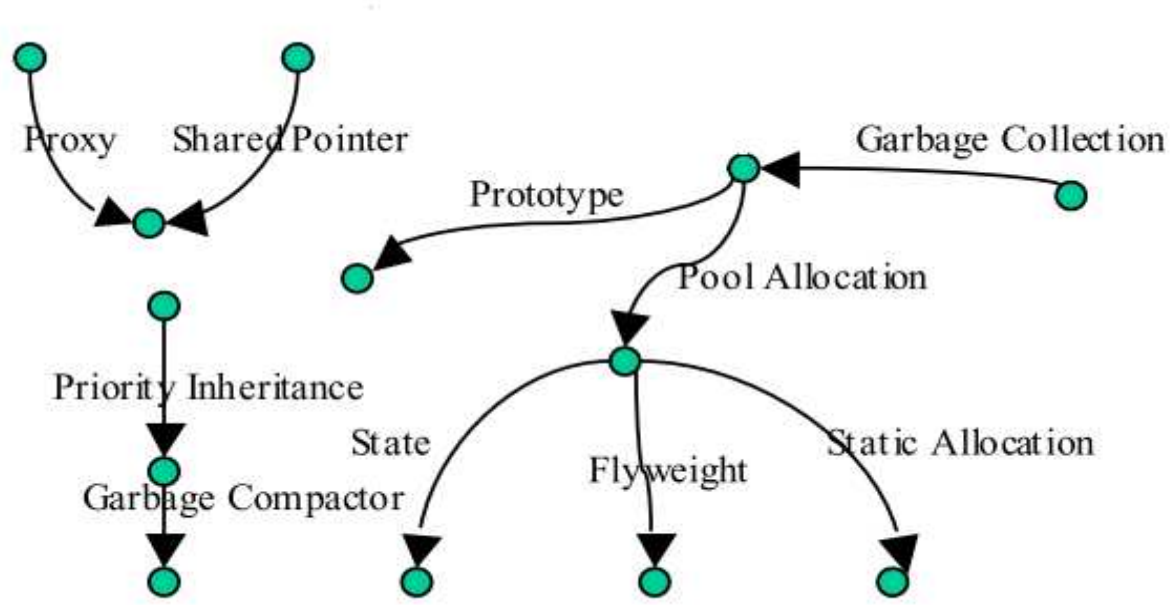


Figure II.4. PRG résultant de l'analyse des patrons [Douglass02] et [GoF95] **[Kubo05b]**

II.2.2.4 HTML Document Analysis

Dans ce bloc, un analyseur HTML est utilisé pour analyser la structure d'un document de patron et obtenir les sections. Comme HTML est un langage balisé, les entêtes et les corps des sections sont clairement séparés. Ainsi le document peut être automatiquement analysé en ayant l'ensemble des balises marquant les entêtes, et l'ensemble des balises marquant les corps des sections.

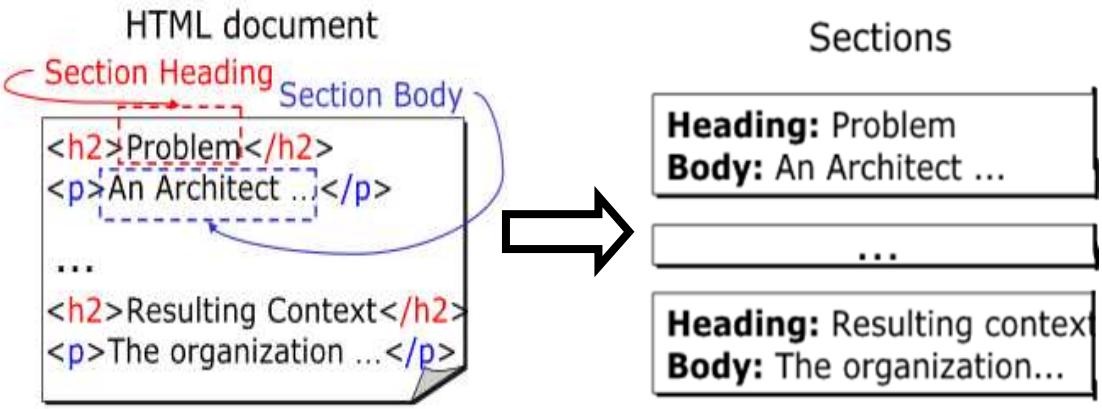


Figure II.5. Obtention des sections à partir du document de patron **[Kubo05b]**

Cet analyseur HTML est une machine d'états finis, ayant trois états : *empty*, *heading* et *body*. Initialement l'analyseur est dans l'état *empty*, et à la rencontre d'une balise il change d'état. A l'état *heading*, la partie du document rencontrée est considérée comme entête. Similairement, à l'état *body*, la partie du document rencontrée est considérée comme corps. A l'état *empty*

l'analyseur détruit la partie du document rencontrée. Par ce processus, les sections sont extraites du document HTML.

II.2.2.5 Pattern Form Judgment

Dans leur technique, Kubo et al. ont conçu une mesure qui exprime l'adaptabilité (*adaptability*) entre le document de patron et une forme de patrons.

On définie *h(d)* l'ensemble d'entêtes du document de patron *d*, et *ad(d, f)* l'adaptabilité de la forme du document de patron *d* pour une forme de patrons *f*. Alors :

$$ad(d,f) = \frac{|f \cap h(d)|}{|f \cup h(d)|}$$

A mesure que *ad(d, f)* est grande, la forme du document de patron *d* est proche de la forme de patrons *f*.

Pour gérer le cas des entêtes ayants le même radical telles que *Intent* et *Intention*, une procédure de *Stemming* (utilisant l'algorithme de Paice **[Paice90]**) est appliquée préalablement pour réduire tous les mots à leurs radicaux.

II.2.2.6 Pattern Extraction

Dans ce bloc, les sections extraites précédemment sont converties en patron représenté dans le modèle de Kubo et al., en utilisant une table de correspondance. Chaque section est soit correspondante à un élément du modèle de patron, soit elle est détruite. Ainsi un patron est obtenu.

Par exemple, la section *Consequences* du GoF correspond au contexte final du modèle de Kubo et al., alors que la section *Motivation* du même formalisme correspond au contexte initial.

Sections	***Elément correspondant du modèle de patrons***
Pattern Name	-
Intent	Contexte Initial
Motivation	Contexte Initial
Applicability	Forces
Solution	-
Consequences	Contexte Final
Related Patterns	-

Table II.2. Table de correspondance des sections au modèle de Kubo et al.

II.2.2.7 Relation Analysis

Pour analyser les liens entre patrons, les relations entre les différentes parties des patrons (*partial documents*) sont analysées.

Comme les techniques d'analyse de similarité entre documents sont matures, les relations entre les parties des patrons sont obtenues en calculant la similarité entre ces dernières. Donc, la similarité indique la force de la relation entre les patrons.

Le degré de similarité est calculé en utilisant un vecteur de poids basé sur la méthode *TF-IDF* **[Salton73]**.

Soit *P* un ensemble de patrons contenant *N* patrons. La procédure de Stemming **[Paice90]**, ainsi que la suppression des mots vides (*stop word removal*) **[Salton83]** sont appliquées aux mots des patrons de *P*. La liste des mots (*terms*) *T* est obtenue via ce traitement. Comme chaque patron est constitué de trois partial documents (contexte initial, forces et contexte final), donc le nombre total des partial documents dans *P* est *3N*.

Soit *tf(s,t)* la fréquence du terme *t* dans le partial document *s*. Et soit *df (t)* le nombre des partial documents contenant le terme *t*. Ainsi, le poids du terme *t* dans le partial document *s* est :

$$w(s,t) = tf(s,t)\left(\log_2 \frac{3N}{df(t)} + 1\right)$$

Ainsi, le vecteur des poids du partial document *s* est conçu comme suit :

$$tv(s,T) = (w(s,t_1), w(s,t_2), \ldots, w(s,t_m)) \text{ où } t_1, t_2, \ldots, t_m \in T$$

Enfin, la similarité entre deux partial documents s_1 et s_2 est calculée comme suit :

$$sim(s_1,s_2) = \frac{tv(s1,T) \cdot tv(s2,T)}{|tv(s1,T)|\,|tv(s2,T)|}$$

On calcule la similarité cosinus entre les paires de patrons, et on extrait les paires dont la similarité est supérieure à un seuil prédéfinie, pour considérer les patrons de ces paires comme des patrons liés. Ces patrons vont se partager leurs nœuds communs, ce qui donnera enfin un *PRG* comme celui montré dans la Figure II.4.

II.2.3. Analyse de la méthode de Kubo et al.

II.2.3.1 Significations des relations entre patrons

Les travaux de Kubo et al. **[Kubo05] [Washizaki07]** (cf. § I.1.4.12) permettent d'analyser les sept types suivants de relations entre deux patrons : *Starting-Starting*, *Resulting-Starting*, *Resulting-Resulting*, *Same*, *SubInStarting*, *SubInResultin*, *SimilarForce*. Les significations de ces relations sont les suivantes :

- Lorsque deux patrons partagent des contextes initiaux similaires, cela signifie que ces patrons traitent le même problème. Donc la relation *Starting-Starting* est synonyme aux relations : *Variant* **[Stal96],** *Alternative* **[Meszaros97],** *Alternative* **[Rieu99]**

[Conte01], *Alternative* **[Gnatz03],** *Processvariance* **[Hagen04a],** *Is an alternative to* **[Prabhakar10]**, *PatternVariation* **[Tran07]**, *Conflicts* **[Beck97]**.

- Lorsque deux patrons partagent des contextes finaux similaires, cela signifie que ces patrons aboutissent au même résultat. La relation *Resulting-Resulting* n'est définie dans aucun travail parmi ceux cités dans la section I.1.

- Lorsque le contexte final d'un patron et le contexte initial d'un autre sont similaires, cela signifie que ce dernier (le patron dont on s'intéresse au contexte initial) s'appliquera après le premier patron. Cela signifie aussi que l'application du premier patron est nécessaire pour le second. Donc la relation *Resulting-Starting* est synonyme aux relations : *Solution is context* **[Henney99],** *Requiert* **[Rieu99][Conte01],** *Sequence* **[Hagen04a]**, Requiert **[Dyson97]**, *Provides Context* **[Volter00]**.

- Lorsque deux patrons partagent des contextes initiaux similaires et des contextes finaux similaires, cela signifie que ces patrons traitent le même problème et fournissent le même résultat. Donc la relation *Same* est synonyme à la relation : *Is duplicate of* **[Prabhakar10].**

- Lorsque le contexte initial et le contexte final d'un patron sont similaires au contexte initial d'un autre patron, cela signifie que le premier patron peut être un sous patron du contexte initial du second patron. La relation *SubInStarting* n'est définie dans aucun travail parmi ceux cités dans la section I.1.

- Lorsque le contexte initial et le contexte final d'un patron sont similaires au contexte final d'un autre patron, cela signifie que le premier patron peut être un sous patron du contexte final du second patron. La relation *SubInResulting* n'est définie dans aucun travail parmi ceux cités dans la section I.1.

- Lorsque deux patrons partagent les même forces, cela signifie que ces patrons gèrent les mêmes contraintes. La relation *SimilarForce* n'est définie dans aucun travail parmi ceux cités dans la section I.1.

II.2.3.2 Avantages de la méthode

La méthode de Kubo et al. est une approche très intéressante à adopter lors d'analyses de relations entre patrons, puisque elle possède beaucoup d'avantages. Principalement :

- C'est une méthode qui n'est pas très contraignante, et qui peut être implémentée.

- Elle est valable pour tous les patrons logiciels de différentes natures.

- Elle permet un traitement automatique qui s'avère de plus en plus nécessaire, vu le nombre croissant de patrons.

- C'est une méthode valable pour tous les formalismes de patrons, puisqu'elle définie un appariement des rubriques de (presque) tout formalisme vers son propre modèle de patrons.

- Comme la méthode est indépendante des formalismes, elle permet de combiner des patrons de différents catalogues.

- La méthode permet aussi de ressortir des relations entre patrons d'une même collection, qui ne sont pas citées par les auteurs de la collection.

- La méthode se base sur des algorithmes de calcul de similarité. Ces algorithmes relèvent du domaine de la Recherche d'Information qui est en évolution continue, donc les performance de la méthode peuvent être en amélioration continue.

II.2.3.3 Inconvénients de la méthode

Cependant, la méthode de Kubo et al. présente les points faibles suivants, mais qui ne dégradent guère l'intérêt de la méthode :

- Le principal inconvénient de cette méthode, est le fait qu'elle se base sur les trois éléments (*Starting Context, Forces, Resulting Context*) constituant le modèle de patrons de Kubo et al.. Ainsi, les patrons ne contenant pas l'un de ces éléments ne pourront être représentés. Plus précisément, on voit ce problème dans le bloc *Pattern Extraction* lorsque la forme du patron à traiter ne contient pas l'une des rubriques nécessaires à la table de correspondance. De plus si des patrons ne peuvent être représentés, les relations abordées par le travail de Kubo et al. ne pourront être analysées, car leur analyse se basent sur le modèle de patrons.

- Un autre désavantage de la méthode de Kubo et al. est qu'elle analyse des relations peu connues dans le domaine des patrons, telles que *Resulting-Resulting, SubInStarting, SubInResulting, SimilarForce* ; alors qu'elle ne permet pas de reconnaitre des relations primaires telles que *Uses (Utilise)* et *Refines (Spécialise)*.

- Un autre inconvénient concerne le bloc *HTML Analysis*, où il faut définir pour chaque catalogue de patrons les deux ensembles de balises (ensemble des balises d'entêtes et ensemble des balises des corps). Cela est du au fait que les balises ne sont pas des mots clés du HTML, donc elles différent d'un catalogue à un autre. Aussi, le bloc *HTML Analysis* se limite au traitement des patrons exprimés en HTML, et ne peut traiter des patrons exprimés autrement.

- Ce qu'on peut aussi reprocher à la méthode de Kubo et al. est le défaut inhérent à la représentation vectorielle, utilisée pour le calcul de similarité entre les éléments des patrons. Ce défaut est que la représentation vectorielle suppose l'indépendance entre les termes, chose qui n'est pas toujours vraie car deux termes différents peuvent être synonymes. Donc la méthode ne permet pas de reconnaitre des termes synonymes.

II.3. Objectifs de notre travail

II.3.1. Objectif concernant le problème d'exhaustivité de la documentation Agile

II.3.1.1. Exploitation des patrons de Ruping pour proposer une solution au problème

Le premier objectif de ce travail concerne la résolution du problème d'exhaustivité de documentation dans les projets Agiles. Comme dit précédemment, la seule collection de patrons qui aborde ce genre de problèmes est le catalogue de Ruping **[Ruping03]**. Pour cela, ce catalogue est retenu pour trouver des patrons adéquats.

Dans le cas où aucun patron n'est capable de traiter le problème dans sa totalité, nous procéderons à la composition de plusieurs patrons où chacun résoudra une partie du problème.

Ainsi, le choix des patrons adéquats, de même que la proposition d'une solution à notre problème seront présentés dans le chapitre suivant.

II.3.2. Objectifs concernant le problème de composition de patrons

Comme Kubo et al. présentent une méthode très avantageuse au service de la composition de patrons (cf. § II.2.3.2), mais qui souffre de quelques lacunes (cf. § II.2.3.3) ; le second objectif de notre travail est d'enrichir cette méthode, du point de vue des relations analysées et du point de vue des formalismes de patrons traités.

II.3.2.1. Analyse des relations Uses et Refines

Plusieurs sortes de relations entre patrons sont proposées dans la littérature (cf. § I.1). Certaines ont des définitions précises, alors que d'autres ont des définitions plus ou moins ambigües. De plus, certaines définitions des relations sont basées sur d'autres relations considérées comme basiques (relations primaires), qui sont présentes très souvent entre les patrons **[Noble98]**.

Les appellations de ces relations primaires diffèrent d'un travail à un autre, mais leurs sémantiques sont équivalentes à travers tous les travaux à quelques détails prés.

Le tableau suivant récapitule les différentes relations citées dans la littérature :

Sémantique des Relations	***Uses***	***Refines***	***Same***	***Rsulting - Starting***	***Starting - Starting***	***Autres relations***				
[Zimmer94]	Uses					Can be combined with	Is similar to			
[Stal96]	Uses				Variant			Combi-naison		

[Meszaros97]	-Uses -Is used by	-Specialize -Generalize			Alternative					
[Beck97]					Conflicts					
[Dyson97]				Requiert						
[Henney99]		Speciali-zation		Solution is context					Pattern pair	Inverted pattern
[Rieu99] [Conte01]	Utilise	Raffine		Requiert	Alternative					
[Volter00]	Is rquired to make a pattern X work	Is a specialization		Provides context						
[Gnatz03]	Raffine				Alternative					
[Hagen04a]	Use	-Refine-problem -Refinement		Sequence	Process-variance					
[Tran07]	Pattern Use	Pattern Refinement			Pattern Variation					
[Prabhakar 10]	Uses	Subsums	Is duplicate of		Is an alternative to					

Table II.3. Types de relations entre patrons à travers différents travaux.

La majorité des relations citées dans la littérature peuvent être analysées par la méthode de Kubo et al. **[Kubo05] [Washizaki07]** sauf les relations *Uses* et *Refines*, qui sont très importantes dans le domaine de l'organisation des patrons et qui sont considérées comme des relations primaires **[Noble98]**.

Nous nous intéressons dans ce travail aux relations primaires, qui sont les plus fréquentes dans la communauté des patrons et qui sont définies dans la majorité des travaux connus. Ces relations sont : *Starting-Starting*, *Resulting-Starting*, *Same*, *Uses* et *Refines*. Ainsi, une amélioration sera proposée dans le chapitre suivant pour ajouter *Uses* et *Refines* aux relations analysées par la méthode de Kubo et al..

II.3.2.2. Traitement des patrons manquants d'une rubrique de contexte final

Comme expliqué précédemment, pour pouvoir analyser les relations entre patrons il faut d'abord pouvoir représenter ces derniers via le modèle de Kubo et al.. Le manque d'une rubrique nécessaire à la représentation de patrons pose problème, car cette dernière ne peut être faite. Plus précisément, ce problème est perçu dans le bloc Pattern Extraction lorsque la forme d'un patron ne contient pas une des rubriques nécessaires à la table de correspondance :

- Le problème se pose rarement avec le **Contexte Initial**, car la quasi-totalité des formalismes de patrons contiennent une rubrique de contexte initial (la majorité des patrons logiciels incluent les rubriques *Context* (starting context), *Forces* et *Solution* **[Stal96]**), ce qui permet de représenter l'élément Starting Context du modèle de patrons de Kubo et al.. Ainsi, le manque du contexte initial ne sera pas traité dans ce travail.

- Si le patron ne contient pas une rubrique correspondante à **Forces**, il ne pourra pas être représenté. Comme la majorité des formalismes de patrons contiennent une rubrique synonymes à Forces **[Stal96]**, ce problème se pose rarement et ne sera donc pas abordé dans le cadre de ce travail.

- Si le patron ne contient pas une rubrique synonyme au **Contexte Final**, il ne pourra être représenté et donc analysé. Cette absence est très fréquente, puisque beaucoup de catalogues de patrons sont exposés dans des formalismes n'exprimant pas le contexte final **[Stal96]**, à l'instar des formalismes utilisés dans : **[Ruping03]**, **[XmlPatterns00], [Lammi], [Crumlish09], [Tidwell05], [YDPL], [FUIDP], [Demeyer03], [Staudt10], [Volter00]**. Du coup, ce manque doit être pallié.

Ainsi, une solution sera proposée dans le chapitre suivant pour pallier à l'absence de la rubrique Contexte Final, et pouvoir traiter ce genre de patrons.

II.4. Conclusion

Le présent chapitre a détaillé les problèmes traités par ce travail, à savoir : le problème d'exhaustivité de la documentation dans les procédés Agiles, qui sera résolu via des patrons ; ainsi que le problème de composition de patrons où une méthode d'analyse automatique des relations entre patrons sera améliorée sur deux plans, l'un pour analyser plus de types de relations et l'autre pour gérer plus de formalismes de patrons.

Le problème d'exhaustivité de la documentation Agile est exposé en détails ; un catalogue de patron est choisi et analysé (catalogue de Ruping) pour exploiter ses patrons dans la résolution de ce problème.

Pour la composition de patrons, l'analyse des relations entre patrons est abordée via une méthode automatique, qui est retenue (méthode de Kubo et al.) en vue d'amélioration sur deux volets :

Pour le premier volet d'amélioration de la méthode, il s'agit de permettre à la méthode d'analyser les relations *Uses* et *Refines*, qui sont des relations primaires se manifestants très souvent entre les patrons.

Pour le second volet de l'amélioration, il s'agit d'enrichir la méthode de Kubo et al. pour pouvoir traiter des patrons dont le formalisme n'exprime pas le contexte final de l'application du patron.

Nos contributions pour résoudre ces problèmes feront l'objet du chapitre suivant.

CHAPITRE III :

CONTRIBUTIONS A LA DOCUMENTATION AGILE ET A L'ANALYSE AUTOMATIQUE DES RELATIONS ENTRE PATRONS

Le présent chapitre est consacré à exposer les solutions aux problèmes traités par ce travail. Il présente dans la première partie une issue pour solutionner le problème d'exhaustivité de la documentation Agile **(cf. § III.1),** en présentant entre autres les patrons utilisés dans solution. La seconde partie porte sur la résolution du problème de composition automatique de patrons. Deux principales améliorations à la méthode de Kubo et al. sont données, permettant d'analyser les relations Uses et Refines, et d'analyser les relations sur des patrons n'ayants pas une rubrique de contexte final **(cf. § III.2)**. Enfin, une conclusion récapitulant le chapitre est donnée **(cf. § III.3)**.

III.1. Solution au problème d'Exhaustivité de la Documentation Agile

III.1.1. Choix des patrons

Comme expliqué dans le chapitre précédent, nous avons retenu le catalogue de Ruping **[Ruping03]** pour chercher des patrons adéquats à la résolution du problème d'exhaustivité de documentation dans les projets Agiles. La composition de patrons s'impose dans notre cas, car aucun patron ne traite le problème en question dans sa totalité. Nous somme donc dans l'obligation de composer une solution à partir de plusieurs patrons.

Nous jugeons les patrons *Target Readers*, *Focused Information* et *Individual Documentation Requirements* intéressants à composer pour résoudre le problème d'exhaustivité de la documentation dans les projets Agiles. Nous présentons les raisons de notre choix dans ce qui suit :

III.1.1.1. Target Readers

Le problème traité par ce patron est *comment s'assurer que les documents produits lors d'un projet seront appréciés*. Il correspond au sous problème d'exhaustivité de la documentation relatant qu'il faut fournir une documentation, qui sera bien appropriée à ses destinataires. Ce patron sera de ce fait utilisé pour résoudre cette partie du problème.

La solution que propose ce patron est que chaque document doit avoir des lecteurs ciblés, et être adapté à ces lecteurs pour leur être utile.

Le tableau suivant présente les intérêts de ce patron qui contribuent à la résolution du problème d'exhaustivité de la documentation, ainsi que les lacunes qui empêchent ce patron de résoudre seul ce même problème :

Intérêts	Lacunes
Ce patron aide au choix des documents pertinents, et cela en ne produisant que des documents ayants des lecteurs cibles. En plus, il aide à optimiser le contenu des documents en se limitant à un contenu approprié aux lecteurs ciblés, ce qui facilite à ces derniers la recherche d'un renseignement.	Un même lecteur peut être adressé (ciblé) par plusieurs documents, voir beaucoup de documents. L'objectif de ce patron se limité à l'adéquation du contenu d'un document à son lecteur ciblé, sans avoir aucune visibilité sur les autres documents qui peuvent traiter le même contenu. Du coup, avec ce patron seul, rien n'empêche de produire des documents superflus, voir dupliqués dans le projet. Ce patron offre des informations appropriées au lecteur cible, mais pas forcément utiles. Ainsi, en se limitant à ce patron on peut se

	retrouver avec un contenu excessif par document.

Table III.1. Intérêts et lacunes du patron Target Readers

Dès lors, ce qui manque pour résoudre le problème d'exhaustivité de la documentation dans sa totalité après utilisation de ce patron, est d'avoir : une visibilité par rapport à tous les documents du projet et l'utilité du contenu de chaque document.

III.1.1.2. Focused Information

Le problème traité par ce patron est *comment fournir un document utile*. Il correspond au sous problème d'exhaustivité de la documentation relatant qu'il faut fournir un contenu pertinent par document. Ce patron sera de ce fait utilisé pour résoudre cette partie du problème.

La solution que propose ce patron est de se focaliser sur un sujet particulier, qui soit clair et bien identifié, et de se limiter à fournir dans un document juste les informations ayants rapport à ce sujet.

Le tableau suivant présente les intérêts de ce patron pour contribuent à la résolution du problème d'exhaustivité de la documentation, ainsi que les lacunes qui empêchent ce patron de résoudre seul ce même problème :

Intérêts	Lacunes
Ce patron résout la partie du problème qui consiste à avoir un contenu pertinent pour aboutir à un minimum d'informations (juste les plus judicieuses), ce qui rend l'exploitation du contenu plus facile.	Le patron se limite à assurer l'utilité d'un document, sans se soucier de la partie du problème qui consiste à choisir les documents à rédiger, ces derniers peuvent traiter le même contenu. Ainsi avec ce patron seul, on a pas de visibilité globale sur les documents du projet, ce qui peut laisser passer des documents superflus. Le patron permet d'avoir un contenu pertinent par document, mais n'adopte pas la meilleure façon d'adresser le lecteur de ce document.

Table III.2. Intérêts et lacunes du patron Focused Information

Par conséquents, ce qui manque pour résoudre le problème d'exhaustivité de la documentation dans sa totalité après utilisation de ce patron, est d'avoir : une vue sur tous les documents du projet et la convenance des documents à leurs destinataires.

III.1.1.3. Individual Documentation Requirements

Le problème traité par ce patron est *comment s'épargner des besoins non nécessaires en termes de documents*. Il correspond au sous problème d'exhaustivité de la documentation, relatant qu'il faut s'épargner des besoins non nécessaires et des besoins dupliqués en matière de documentation. Ce patron sera de ce fait utilisé pour résoudre cette partie du problème.

La solution que propose ce patron est de désigner pour chaque projet son besoin particulier en documentation.

Le tableau suivant présente les intérêts de ce patron pour contribuent à la résolution du problème d'exhaustivité de la documentation, ainsi que les lacunes qui empêchent ce patron de résoudre seul ce même problème :

Intérêts	Lacunes
Ce patron permet de choisir la quantité globale de documentation appropriée pour chaque projet, ce qui permet de s'épargner des besoins non nécessaires et offre une vue globale sur tous les documents du projet.	Ce patron ne cible pas les lecteurs (des documents du projet) pour les adresser de manière adéquate. Il se contente de lister des documents pertinents par projet, sans se préoccuper de donner un contenu pertinent par document.

Table III.3. Intérêts et lacunes du patron Individual Documentation Requirements

Ainsi, ce qui manque pour résoudre le problème d'exhaustivité de la documentation dans sa totalité après utilisation de ce patron, est d'avoir : une pertinence du contenu de chaque document et l'adéquation des documents aux lecteurs.

III.1.2. Composition des patrons de Ruping pour résoudre le problème

Les trois patrons choisis pour résoudre le problème d'exhaustivité de la documentation Agile ne sont pas explicitement liés entre eux, car aucune relation n'est citée par l'auteur dans les rubriques *Discussion* de ces patrons. Nous devons ainsi nous même découvrir les relations existantes, pour pouvoir composer les patrons.

Pour faciliter la compréhension de la composition à laquelle nous allons procéder, nous donnons dans ce qui suit des représentations graphiques des patrons composants et de la solution composée.

III.1.2.1. Les différentes solutions des patrons sous une forme graphique

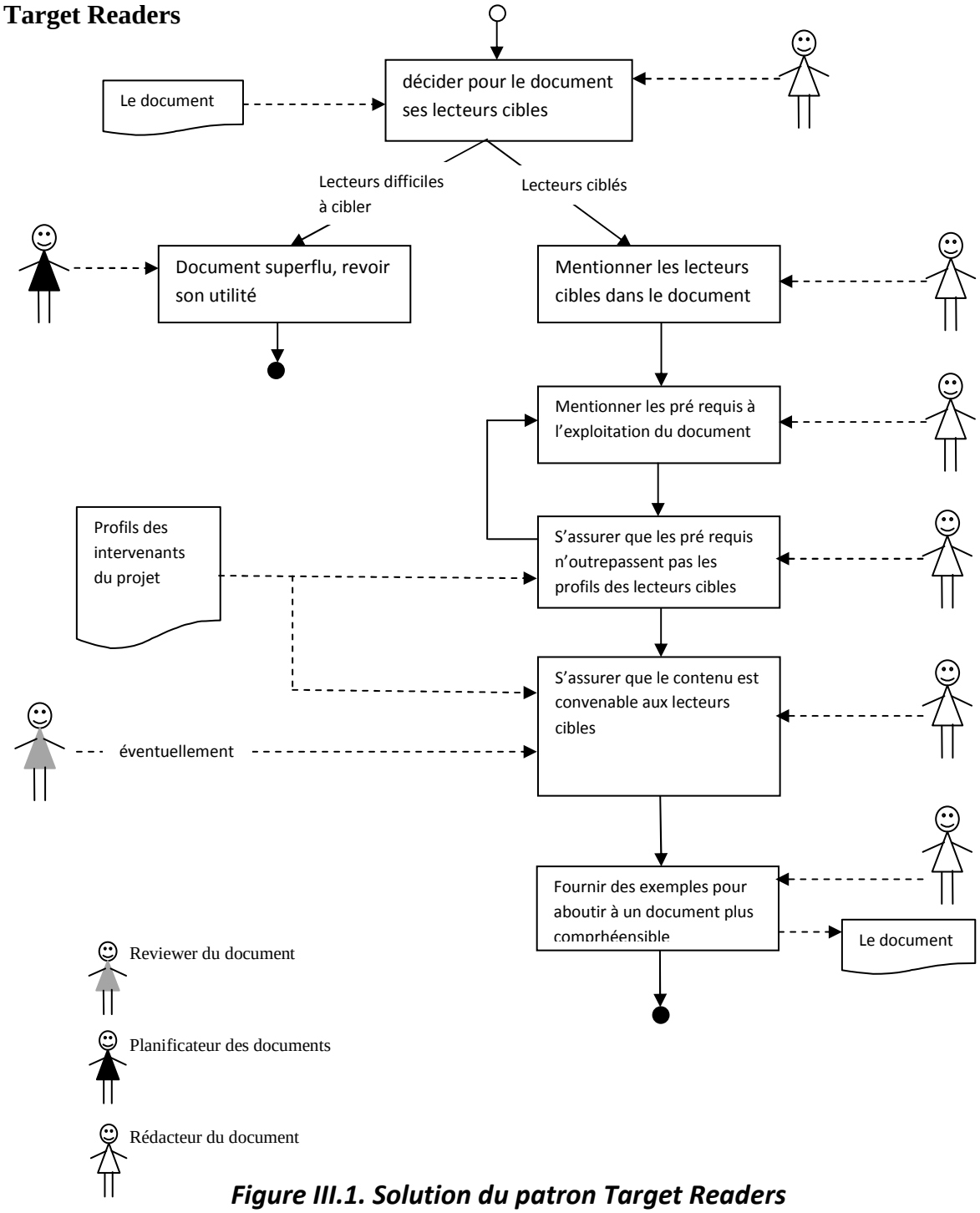


Figure III.1. Solution du patron Target Readers

Focused Information

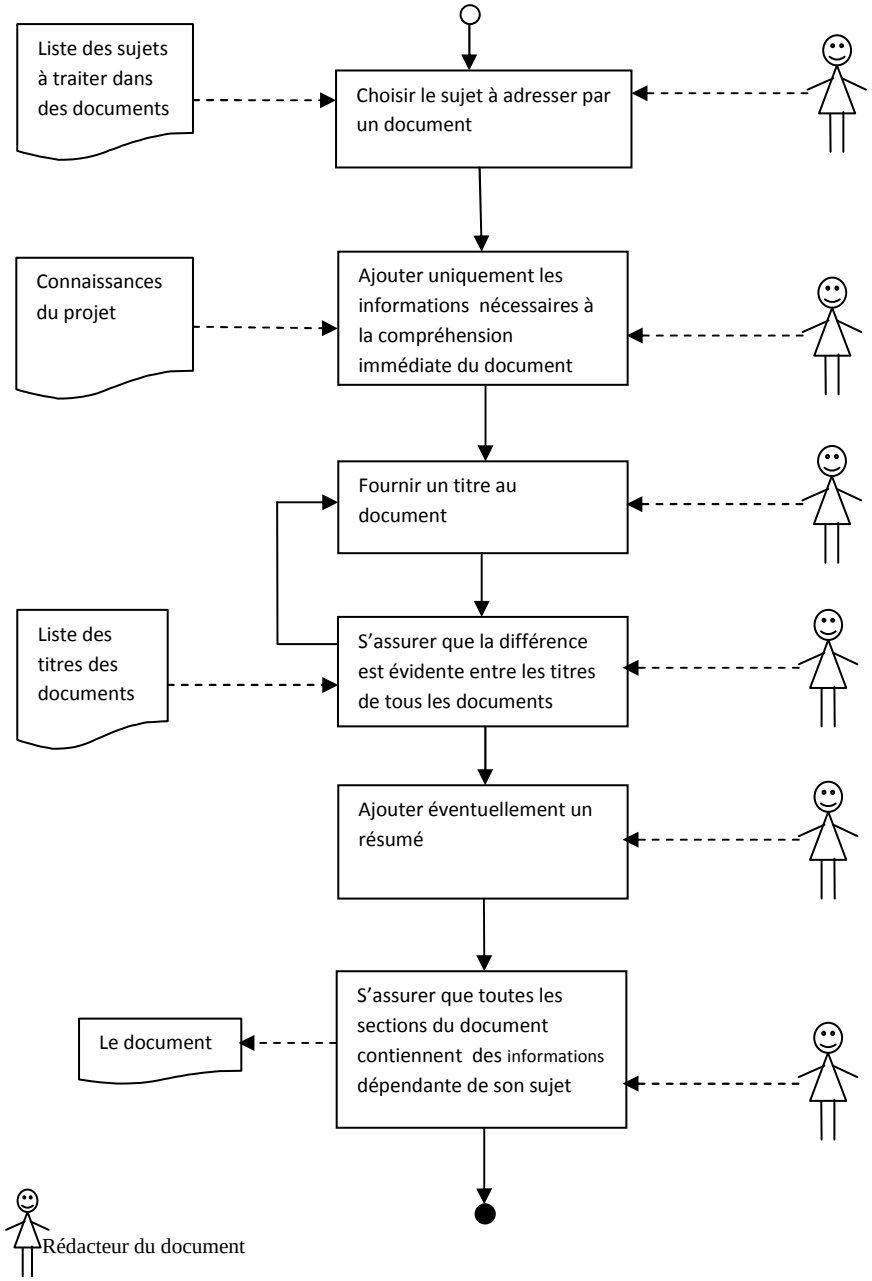
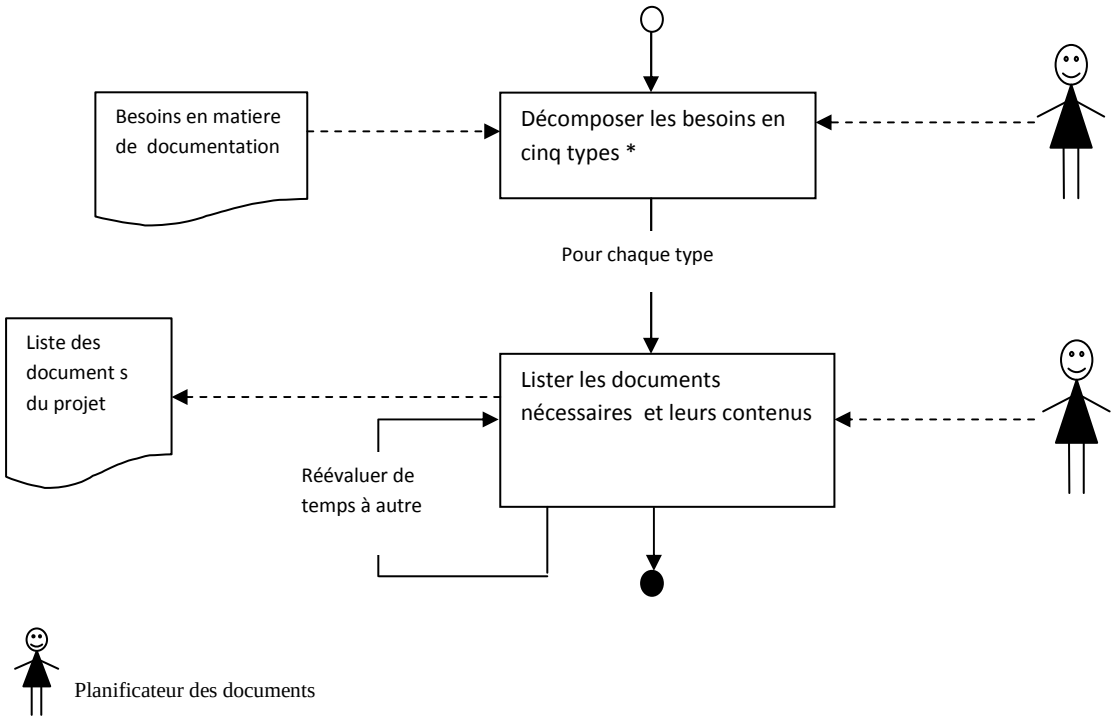


Figure III.2. Solution du patron Focused Information

Individual Documentation Requirements



* Les types de besoins: des stockholders, de communication, des futures étapes du projet, de suivi du projet, de description des idées.

Figure III.3. Solution du patron Individual Documentation Requirements

III.1.2.2. La composition proprement dite

Pour concevoir notre solution au problème d'exhaustivité de la documentation Agile, nous avons réutilisé les trois patrons choisis tels quels, sans leur appliquer aucune modification.

Nous avons procédé à la composition de ces trois patrons en se basant sur la relation existante entre eux, à savoir la relation *Resulting-Starting*. Cette relation est fondée sur la similarité des contextes initial et final des deux patrons qu'elle lie. Les contextes des patrons auxquels nous nous intéressons sont les suivants :

Le contexte initial du patron **Individual Documentation Requirements** est l'existence de différents besoins de documentation. Son contexte final est la décision d'une liste de documents à réaliser pour le projet.

Le contexte initial du patron **Focused Information** est l'existence de thèmes à adresser par des documents. Son contexte final est la production d'un document avec un contenu focalisé sur un thème.

Le contexte initial du patron **Target Readers** est le besoin de convenance d'un document à un type de lecteurs. Son contexte final est la production d'un document bien approprié pour un type de lecteurs.

Ainsi, nous avons opéré comme suit :

- Le patron **Individual Documentation Requirements** est utilisé en premier lieu, pour permettre de décider d'une liste de documents pertinents pour le projet, à partir des différents besoins en matière de documentation. Il est normale dans un procédé de documentation de commencer par choisir les documents à produire.

- En effet, la liste de documents résultante de l'application de ce patron représente l'ensemble des thèmes qui devront être adressés par des documents. Et c'est exactement la pré condition nécessaire à l'application du patron **Focused Information**, à savoir avoir un sujet à traiter par un document.

- Le patron **Focused Information** s'avère nécessaire, car il permet de produire pour un sujet donné un document focalisé dessus. Donc, pour chaque thème (pour chaque élément de la liste), on applique ce patron qui résulte en un document avec un contenu focalisé.

- Jusqu'ici, le problème d'exhaustivité de la documentation n'est pas résolu dans sa totalité, il va devoir aboutir à des documents appréciés par les lecteurs. Ainsi, pour chaque document issu de l'application du patron précédent, on applique le patron **Target Readers.** Ce dernier reçoit en entrée un document et résulte en le même document transformé de manière à être bien appropriée à ses destinataires.

La figure **III.4** montre la solution composée via les trois patrons de Ruping, pour la résolution du problème d'exhaustivité de la documentation Agile.

Apres application de ces trois patrons, on aboutit à une quantité optimale de documents qui permet de couvrir les besoins en documentation, tout en étant une quantité facile à gérer et à exploiter.

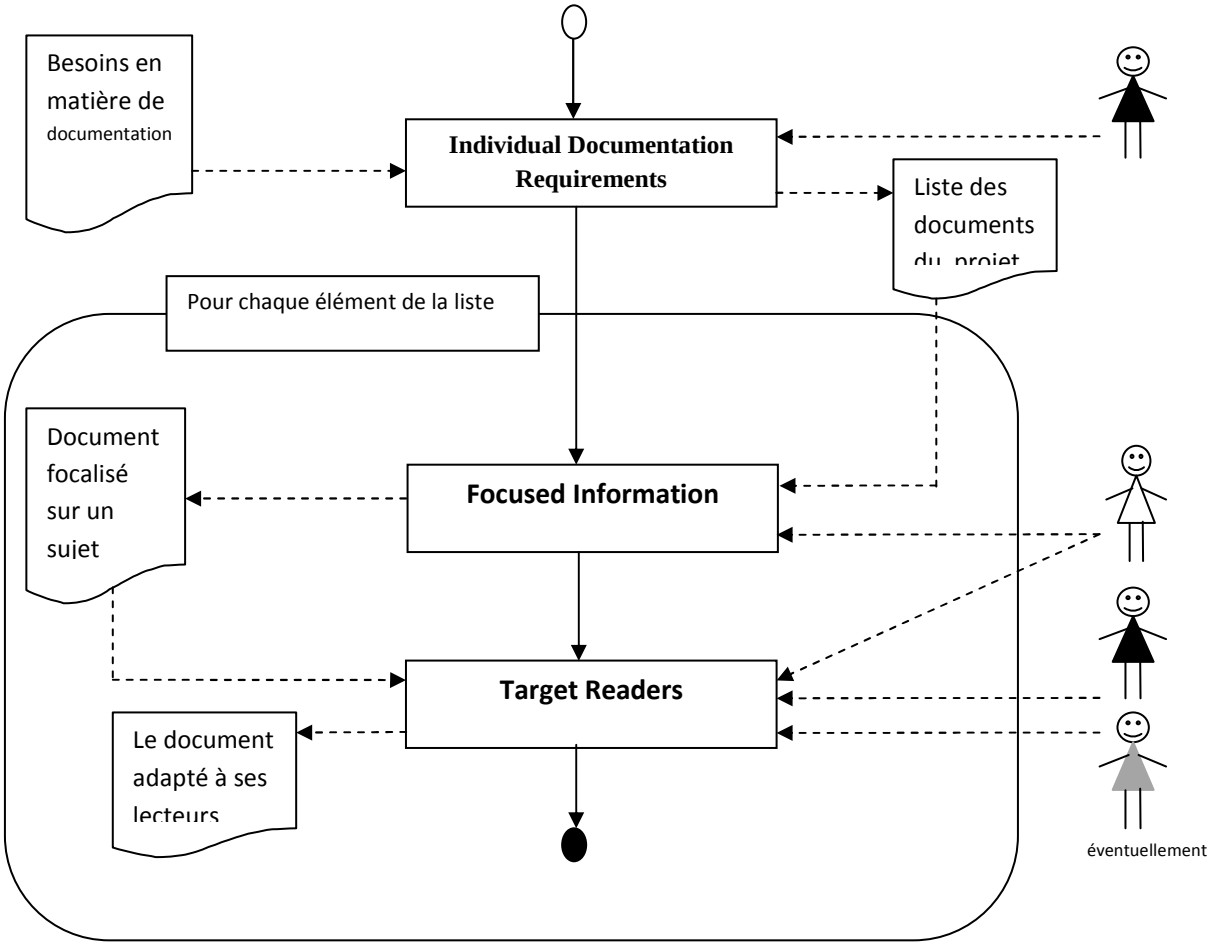


Figure III.4. Composition des trois patrons

III.2. Solution au problème de Composition de Patrons

Cette section présente notre contribution à la résolution du problème de composition de patrons logiciels. Pour ce faire, nous avons retenu une méthode d'analyse automatique des relations entre patrons dans le but de l'améliorer sur deux plans, l'un pour analyser plus de types de relations et l'autre pour gérer plus de formalismes de patrons.

Dans la suite de ce chapitre, nous allons donner des résultats de calculs de relations entre patrons. afin d'obtenir ces résultats, nous avons utilisé notre prototype qui analyse les relations entre patrons, et que nous décrivons en détails dans **III.2.3**.

III.2.1. Analyse des relations Uses et Refines

L'analyse des relations entre patrons dans la méthode de Kubo et al., se base sur le calcul de similarité entre les différentes rubriques des patrons (éléments de patrons). Toutefois, cette approche de similarité est nécessaire mais insuffisante pour analyser les relations *Uses* et *Refines*.

Nous proposons pour analyser ces deux relations, de se baser sur la vérification de certaines conditions, parmi lesquelles se trouve l'*Inclusion*. Ainsi, nous commençons par proposer dans ce qui suit une définition et une méthode d'analyse de l'*Inclusion*.

III.2.1.1. Analyse de l'Inclusion

L'analyse de l'*Inclusion* est un auxiliaire permettant d'analyser les relations *Uses* et *Refines*. L'*Inclusion* signifie lorsqu'elle existe entre deux textes, que non seulement ces deux textes sont similaires mais en plus, que l'un contient l'autre.

Nous proposons ici une analyse possible de l'*Inclusion* qui se procède de la manière suivante : Soient deux éléments T1 et T2, où chacun est un texte constitué d'un ou plusieurs termes. T1 *est Inclus dans* T2 signifie que les deux conditions suivantes sont vérifiées.

- T1 similaire à T2 : T2 doit contenir les termes de T1. Cette condition a pour but de s'assurer que le contenu de T1 est traité par T2.
- T2 plus large que T1 : T2 doit contenir plus de termes que T1. Cette condition a pour but de s'assurer que T1 et T2 ne soient pas égaux, ce qui signifie que T2 contient T1 en plus d'un autre contenu.

Une fois deux éléments de patrons sont similaires, la valeur de l'*Inclusion* entre eux est : le rapport de la différence de taille des deux éléments, à la taille du plus grand élément. Si ce rapport est supérieur ou égale au seuil, l'*Inclusion* entre les deux éléments est considérée vraie.

Pour le calcul de la taille d'un élément, nous considérons tous les termes de l'élément après suppression des Stopwords.

Quand à la technique de calcul de similarité choisie dans notre travail, nous reprenons celle utilisée par Kubo et al. qui est la similarité cosinus, expliquée dans II.2.2.7.

Pour illustrer notre idée d'analyse de l'*Inclusion*, nous donnons dans ce qui suit des exemples sur six éléments de patrons. Le calcule de l'*Inclusion* entre ces différents éléments de patrons commence par le calcul de similarité entre eux, le calcul des tailles des éléments, ensuite nous comparons ces tailles pour vérifier enfin l'Inclusion proprement dite.

- Par exemple, l'élément de patron (appelé élément 0) *"A clear separation is needed between what is metadata and what is data that forms the body of the document. This affects ease of authoring and processing of the document because the context of the data is clear"*, est inclus dans l'élément suivant (appelé élément 1) *"A clear separation is needed between what is metadata and what is data that forms the body of the document. This affects ease of authoring and processing of the document because the context of the data is implied. The metadata needed for an element needs to structured in such a way that attributes are difficult to use"*.

Elément	**Nombre de termes**
0	16
1	22
Couple d'éléments	**Comparaison de tailles**
0 et 1	L'élément 0 n'est pas plus large que l'élément 1
1 et 0	Taux de supériorité de l'élément 1 par rapport à 0 est de 27%
Couple d'éléments	**Résultats de l'Inclusion**
0 et 1	Similarité des éléments 0 et 1 = 0,789
	L'élément 0 inclut l'élément 1 = False
	L'élément 1 inclut l'élément 0 = True

Table III.4. Calcul de l'Inclusion entre les éléments 0 et 1

Ces deux éléments sont similaires, en plus le second est plus large que le premier. Donc l'élément 0 est inclus dans l'élément 1.

- Un autre exemple concernant les deux éléments : *"Provide an interface for creating families of related or dependent objects without specifying their concrete classes"* (appelé element 2) et *"Providing an interface to create objects"* (appelé element 3).

Elément	**Nombre de termes**
2	9
3	4
Couple d'éléments	**Comparaison de tailles**
2 et 3	Taux de supériorité de l'élément 2 par rapport à 3 est de 55%
3 et 2	L'élément 3 n'est pas plus large que l'élément 2

Couple d'éléments	Résultats de l'Inclusion
2 et 3	Similarité des éléments 2 et 3 = 0,537
	L'élément 2 inclut l'élément 3 = True
	L'élément 3 inclut l'élément 2 = False

Table III.5. Calcul de l'Inclusion entre les éléments 2 et 3

Ces deux éléments sont similaires et le premier élément est plus large que le second. Ainsi, le premier élément inclut le second.

- Encore un exemple sur les deux éléments : "*Totalitarian control is viewed by most development teams as a draconian measure. The right information must flow through the right roles. You need to support information flow across analysis, design, and implementation. Managers have some accountability. Developers should have ultimate accountability, and have the authority and control of the product; these are often process issues*" (appelé element 4) et "*Totalitarian control is viewed by most development teams as a draconian measure. The right information must flow through the right roles*" (appelé element 5).

Elément	Nombre de termes
4	26
5	10
Couple d'éléments	**Comparaison de tailles**
4 et 5	Taux de supériorité de l'élément 4 par rapport à 5 est de 61%
5 et 4	L'élément 5 n'est pas plus large que l'élément 4
Couple d'éléments	**Résultats de l'Inclusion**
4 et 5	Similarité des éléments 4 et 5 = 0,646
	L'élément 4 inclut l'élément 5 = True
	L'élément 5 inclut l'élément 4 = False

Table III.6. Calcul de l'Inclusion entre les éléments 4 et 5

Ces deux éléments sont similaires et le premier élément est plus large que le second. Ainsi, le premier élément inclut le second.

III.2.1.2. Analyse de la relation Uses

En se basant sur l'analyse de l'*Inclusion* qui vient d'être présentée, nous proposons d'analyser la relation *Uses* de la manière suivante :

Soient P1 et P2 deux patrons exprimés dans le modèle de Kubo et al. (cf. § II.2.2.3). Chaque élément de ces patrons (Starting Context, Forces, Resulting Context) est un texte, donc l'*Inclusion* peut être analysée entre les différents éléments de ces deux patrons.

Ainsi, P1 *Uses* P2 signifie que les deux conditions suivantes sont vérifiées.

- L'élément Contexte Initial de P2 *est inclus dans* l'élément Contexte Initial de P1 : cette condition est nécessaire pour signifier que le problème traité par P2 est un sous problème de celui traité par P1.

- L'élément Contexte Final de P2 *est inclus dans* l'élément Contexte Final de P1 : cette condition a pour but de vérifier que le résultat produit par l'application du patron P2 est un sous ensemble du résultat produit par l'application du patron P1.

Voici un exemple de calcul de la relation *Uses* :

- Le patron *Code Ownership* (appelé P2) **[Coplien94]** qui s'intéresse à l'organisation d'un développement logiciel, préconise que chaque module du code doit être possédé par un développeur qui veillera sur lui. Le patron *Review* (appelé P1) **[Coplien94]** présente les activités d'une revue. Il est naturel que lorsque un développeur veille sur un module de code, il effectue entre autres une revue et il utilisera pour cela le patron *Review*.

  Cette relation *P2UsesP1* est citée explicitement par l'auteur du patron *Review* : *"It will also solve potential problems that have been pointed out for Code Ownership"*.

  La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

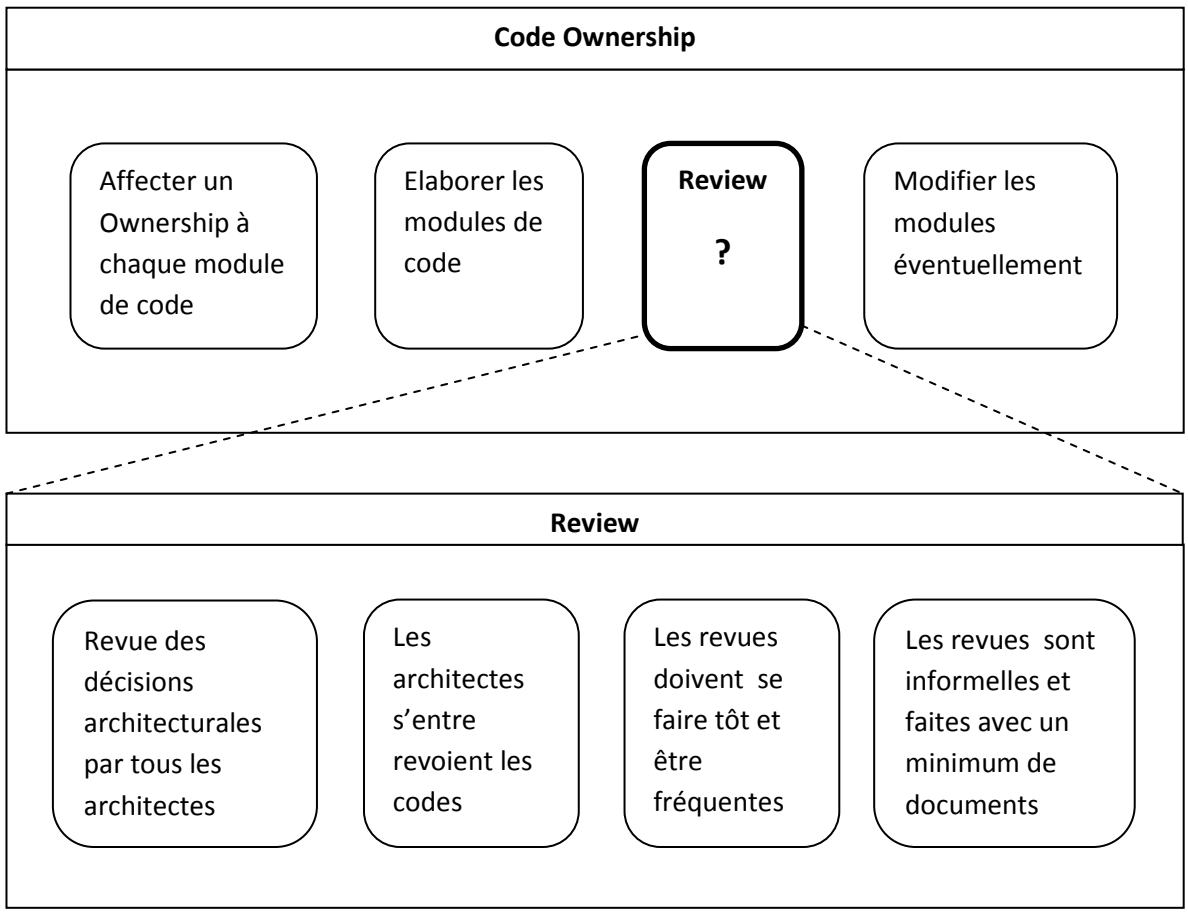


Figure III.5. La relation Uses entre les patrons Code Ownership et Review

Donc, nous considérons cette relation (citée par l'auteur) comme une relation correcte, et nous appliquons notre méthode pour retrouver le même résultat. Notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,095
	CI1 inclut CI2 = False
	CI2 inclut CI1 = True
Contexte final (CF) 1, Contexte final 2	Similarité = 0,240
	CF1 inclut CF2 = False
	CF2 inclut CF1 = True
Forces (F) 1, Forces 2	Similarité = 0,127
	F1 inclut F2 = False
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,040
Contexte final 2, Contexte initial 1	Similarité = 0

Table III.7. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

A partir de ces résultats, nous calculons toutes les relations auxquelles nous nous intéressons entre ces deux patrons. Pour les relations Same, Starting-Starting et Resulting-Starting nous reprenons la manière de Kubo et al. pour les calculer ; par contre pour la relation Refines, nous expliquons son analyse dans la suite de ce chapitre. Les résultats sont les suivants :

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0,167
P2 Refines P1	0
Same	0,167
Starting-Starting	0,095

Resulting-Starting (P1P2)	0
Resulting-Starting (P2P1)	0

Table III.8. Résultats des calculs de relations entre le couple de patron

Enfin et comme dans la méthode de Kubo et al., la relation la plus puissante est la relation à considérer pour les deux patrons. Pour cet exemple il s'agit bien de la relation *Uses*.

NB : Cette manière de procéder pour le calcul des relations entre patrons, sera reprise pour le reste des exemples de ce présent chapitre.

III.2.1.3. Analyse de la relation Refines

En se basant sur l'analyse de l'*Inclusion* (cf. § III.2.1.1), nous proposons d'analyser la relation *Refines* de la manière suivante :

Soient P1 et P2 deux patrons exprimés dans le modèle de Kubo et al. (cf. § II.2.2.3). Chaque élément de ces patrons (Starting Context, Forces, Resulting Context) est un texte, donc l'*Inclusion* peut être analysée entre ces différents éléments. Ainsi, P2 *Refines* P1 signifie que les trois conditions suivantes sont vérifiées.

- L'élément Contexte Initial de P2 *est similaire à* l'élément Contexte Initial de P1 : cette condition a pour but de vérifier que les patrons P1 et P2 traitent le même problème.

- L'élément Contexte Initial de P1 *n'Inclut pas* l'élément Contexte Initial de P2 : cette condition a pour but d'assurer que le patrons P1 (le patron raffiné) ne traite un problème plus large que celui traité par P2.

- L'élément Forces de P1 *est Inclus dans* l'élément Forces de P2 : cette condition signifie que les contraintes imposées dans le patron P1 représentent un sous ensemble des contraintes du patron P2.

Voici un exemple de calcul de la relation *Refines* :

- Le patron *Head-Body* **[XmlPatterns00]** (appelé P1) qui s'intéresse à la structuration de documents XML, aborde la séparation de l'entête d'un document XML (les métadonnées décrivant le contenu du document) de son corps (les données du document XML). *Head-Body* est une spécialisation du patron *Separate Metadata and Data* **[XmlPatterns00]** (appelé P2) préconisant que de manière générale, lorsqu'il y a « un contenu » et « des données sur ce contenu », alors ces deux types doivent être séparés.

  Cette relation *P1RefinesP2* est citée explicitement par l'auteur des patrons : "*Head-Body* is a specialization of *Separate Metadata and Data*".

  La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

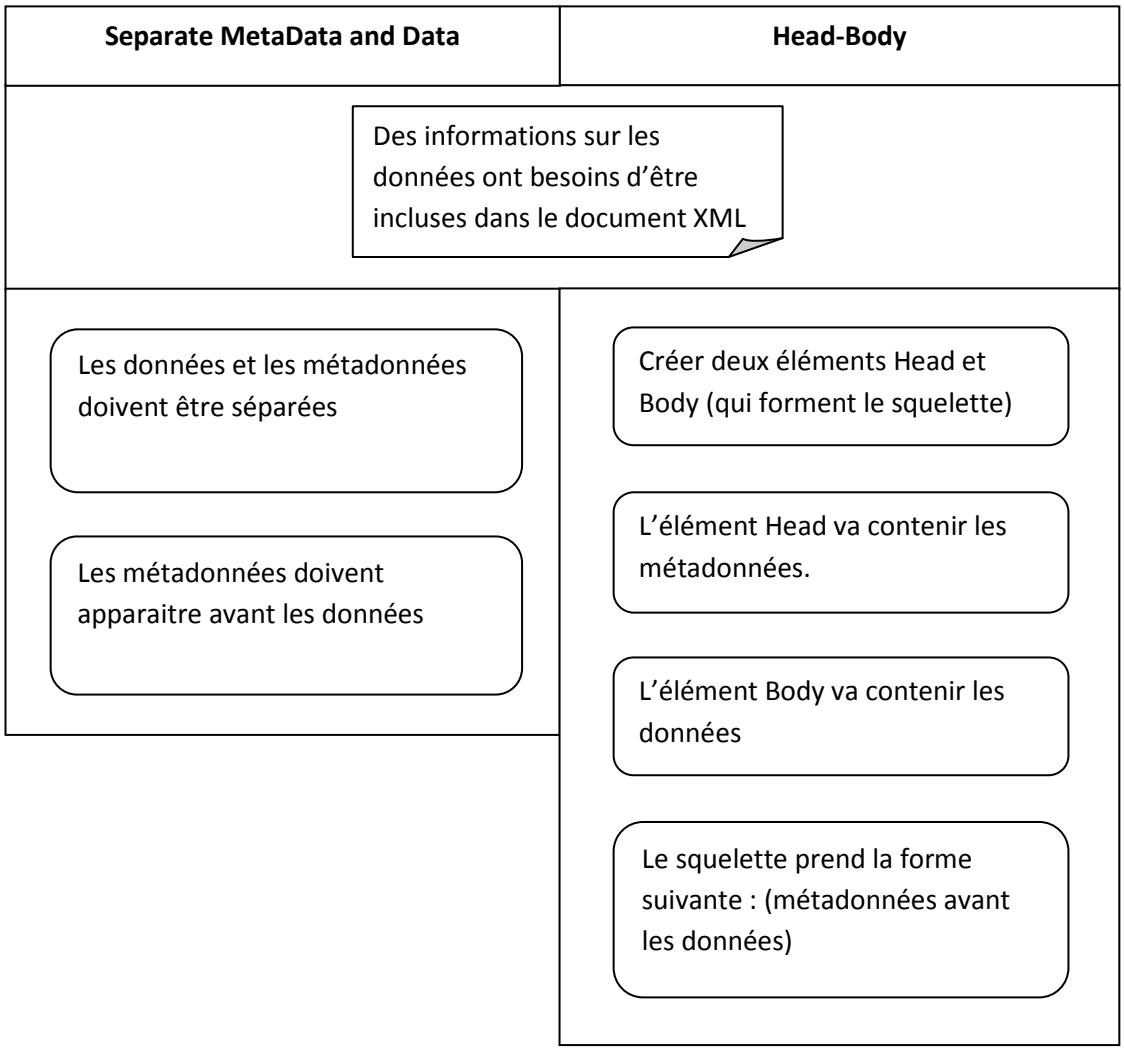


Figure III.6. La relation Refines entre les patrons Head-Body et Separate Metadata and Data

Donc, on considère cette relation (citée par l'auteur) comme une relation correcte, et nous appliquons notre méthode qui donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,342
	CI1 inclut CI2 = False
	CI2 inclut CI1 = False
Contexte final (CF) 1, Contexte final 2	Similarité = 0,040
	CF1 inclut CF2 = False
	CF2 inclut CF1 = True
Forces (F) 1, Forces 2	Similarité = 0,746

	F1 inclut F2 = True
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,056
Contexte final 2, Contexte initial 1	Similarité = 0.070

Table III.9. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0,544
P2 Uses P1	0
P2 Refines P1	0
Same	0
Starting-Starting	0,342
Resulting-Starting (P1P2)	0
Resulting-Starting (P2P1)	0,070

Table III.10. Résultats des calculs de relations entre le couple de patron

Ainsi, pour cet exemple il s'agit de la relation *Refines*.

III.2.1.4. Expérimentation

Les patrons cités dans ce qui suit sont inter liés par les relations *Uses* ou *Refines*, sans que cela ne soit explicitement mentionné par les auteurs des patrons ; et la méthode de Kubo et al. n'est pas capable d'analyser ces relations. Avec notre proposition, il est maintenant possible de détecter ces relations.

- Pour les deux patrons *Multiple Process Versions* **[Beedle97]** (appelé P1) et *Organization Follows Market* **[Coplien94]** (appelé P2), notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,063
	CI1 inclut CI2 = False
	CI2 inclut CI1 = False

Contexte final (CF) 1, Contexte final 2	Similarité = 0
	CF1 inclut CF2 = False
	CF2 inclut CF1 = True
Forces (F) 1, Forces 2	Similarité = 0,112
	F1 inclut F2 = True
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,061
Contexte final 2, Contexte initial 1	Similarité = 0

Table III.11. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0,088
P2 Uses P1	0
P2 Refines P1	0
Same	0
Starting-Starting	0,063
Resulting-Starting (P1P2)	0
Resulting-Starting (P2P1)	0

Table III.12. Résultats des calculs de relations entre le couple de patron

Les contextes initiaux de ces deux patrons sont similaires. Le contexte initial du patron P2 n'est pas inclus dans le contexte initial du patrons P1. L'élément Forces du patron P2 est inclus dans Forces du patron P1. Ainsi, pour cet exemple la relation à considérer pour les deux patrons est **P1 *Refines* P2**. En effet, cette relation est correcte pour les raisons suivantes :

Les deux patrons s'appliquent dans la cas où *les besoins des différents segments du marché (ou des clients) sont similaires mais parfois conflictuels* (ie le même produit ne peut convenir à tous).

Le patron P2 a pour but d'aboutir à un produit adaptée à un segment particulier du marché (ou à un client). Pour cela le patron s'intéresse à un noyau englobant

seulement ce qui est commun pour tous les clients (sans indiquer s'il faudra personnaliser ce noyau ou le livrer tel quel aux différents clients).

Le patron P1 qui a le même but offre une solution plus détaillée, et propose de cibler les éléments pouvant être communs et de les réutiliser pour faire différentes versions du produit.

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

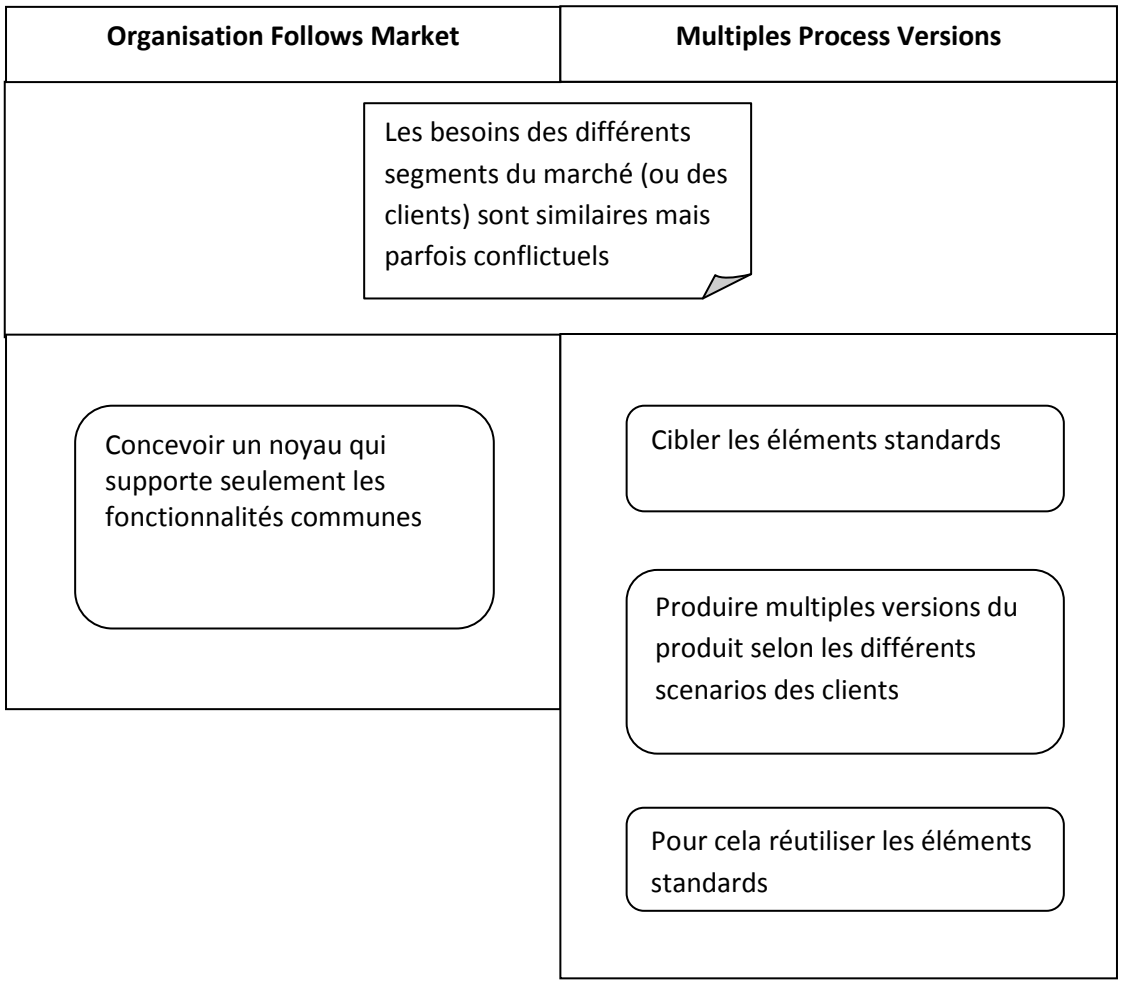


Figure III.7. La relation Refines entre les patrons Multiple Process Versions et Organization Follows Market

- Pour les deux patrons *Process Follows Practice* **[Appleton97]** (appelé P1) et *Scenarios Define Business Processes* **[Beedle97]** (appelé P2), notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,132
	CI1 inclut CI2 = True

	CI2 inclut CI1 = False
Contexte final (CF) 1, Contexte final 2	Similarité = 0,042
	CF1 inclut CF2 = True
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0,164
	F1 inclut F2 = True
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,113
Contexte final 2, Contexte initial 1	Similarité = 0,130

Table III.13. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0,148
P2 Uses P1	0
P2 Refines P1	0
Same	0
Starting-Starting	0,132
Resulting-Starting (P1P2)	0,113
Resulting-Starting (P2P1)	0,130

Table III.14. Résultats des calculs de relations entre le couple de patron

Les contextes initiaux de ces deux patrons sont similaires. Le contexte initial du patron P2 n'est pas inclus dans le contexte initial du patrons P1. L'élément Forces du patron P2 est inclus dans Forces du patron P1. Ainsi, pour cet exemple la relation à considérer pour les deux patrons est **P1 *Refines* P2**. En effet, cette relation est correcte pour les raisons suivantes :

Les deux patrons s'appliquent pour bien documenter un procédé et son évolution.

Le patron P2 adresse le meilleure moyen de documenter le changement d'un procédé. Le patron P1 raffine ce patron et montre comment améliorer un procédé, tout en

assurant en même temps que la documentation représente avec exactitude le travail en cours.

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

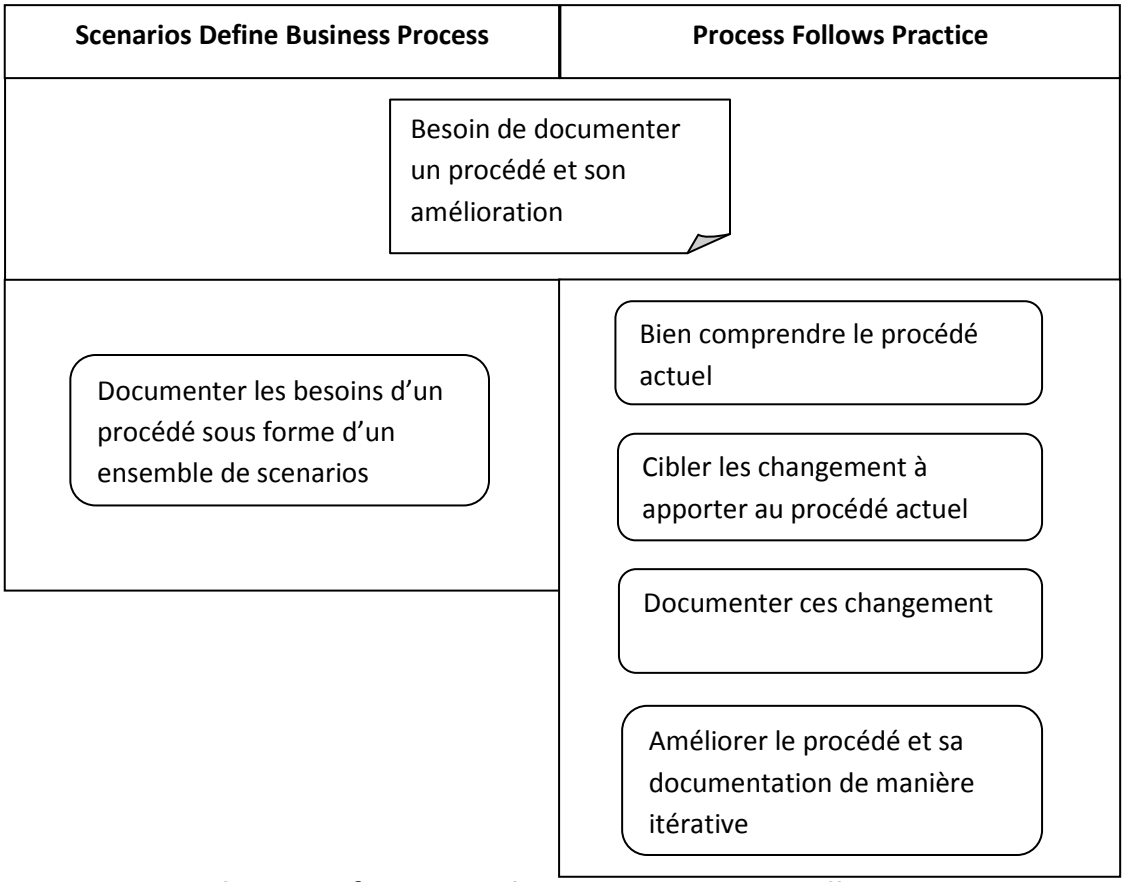


Figure III.8. La relation Refines entre les patrons Process Follows Practice et Scenarios Define Business Processes

- Pour les patrons *Prototype* **[Coplien94]** (appelé P2) et *Application Design is Bounded By Test Design* **[Coplien94]** (appelé P1), notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,122
	CI1 inclut CI2 = False
	CI2 inclut CI1 = True
Contexte final (CF) 1, Contexte final 2	Similarité = 0,205
	CF1 inclut CF2 = False
	CF2 inclut CF1 = True

Forces (F) 1, Forces 2	Similarité = 0,170
	F1 inclut F2 = True
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,099
Contexte final 2, Contexte initial 1	Similarité = 0,038

Table III.15. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0,164
P2 Refines P1	0
Same	0,164
Starting-Starting	0,122
Resulting-Starting (P1P2)	0,099
Resulting-Starting (P2P1)	0

Table III.16. Résultats des calculs de relations entre le couple de patron

Le contexte Initial du patron *Application Design is Bounded By Test Design* est inclus dans le Contexte Initial de *Prototype.* Aussi, le contexte résultant (final) du patron *Application Design is Bounded By Test Design* est inclus dans le contexte résultant du patron *Prototype*. Donc pour cet exemple, la relation à considérer pour les deux patrons est **P2 *Uses* P1**. En effet, cette relation est correcte pour les raisons suivantes :

Le patrons P2 a pour but de valider les besoins du client, pour pouvoir préparer les phases suivantes du projet logiciel. Il préconise dans sa solution l'utilisation de tests pour valider ces besoins acquis au préalable. Pour ce faire il peut utiliser le patron P1 qui montre comment préparer et procéder aux tests.

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

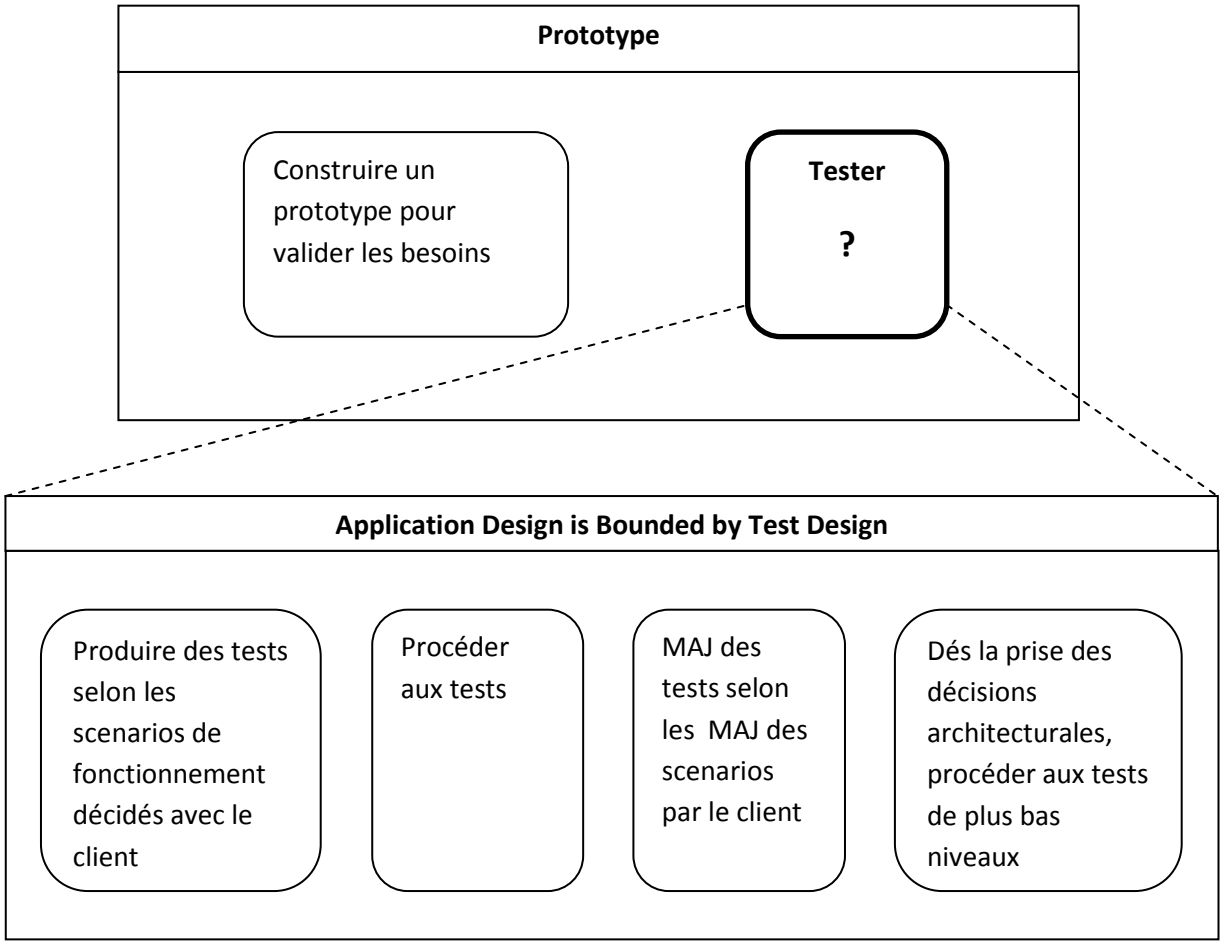


Figure III.9. La relation Uses entre les patrons Prototype et Application Design is Bounded By Test Design

- Pour les deux patrons *Engage Customer* **[Coplien94]** (appelé P1) et *Group Validation* **[Coplien94]** (appelé P2), notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,082
	CI1 inclut CI2 = True
	CI2 inclut CI1 = False
Contexte final (CF) 1, Contexte final 2	Similarité = 0,099
	CF1 inclut CF2 = True
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0,057

	F1 inclut F2 = False
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0
Contexte final 2, Contexte initial 1	Similarité = 0,036

Table III.17. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0,091
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0
Same	0,091
Starting-Starting	0,082
Resulting-Starting (P1P2)	0
Resulting-Starting (P2P1)	0

Table III.18. Résultats des calculs de relations entre le couple de patron

Le contexte Initial de *Group Validation* est inclus dans le Contexte Initial de *Engage Customer*. Aussi, le contexte résultant du patron *Group Validation* est inclus dans le contexte résultant du patron *Engage Customer*. Donc pour cet exemple, la relation à considérer pour les deux patrons est **P1 *Uses* P2**. En effet, cette relation est correcte pour les raisons suivantes :

Le patron P1 s'intéresse à maintenir la satisfaction client, en lui fournissant un produit de qualité et en le faisant participer au projet. Pour assurer une bonne qualité du produit, le patron peut utiliser P2 qui en effet aborde ce problème.

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

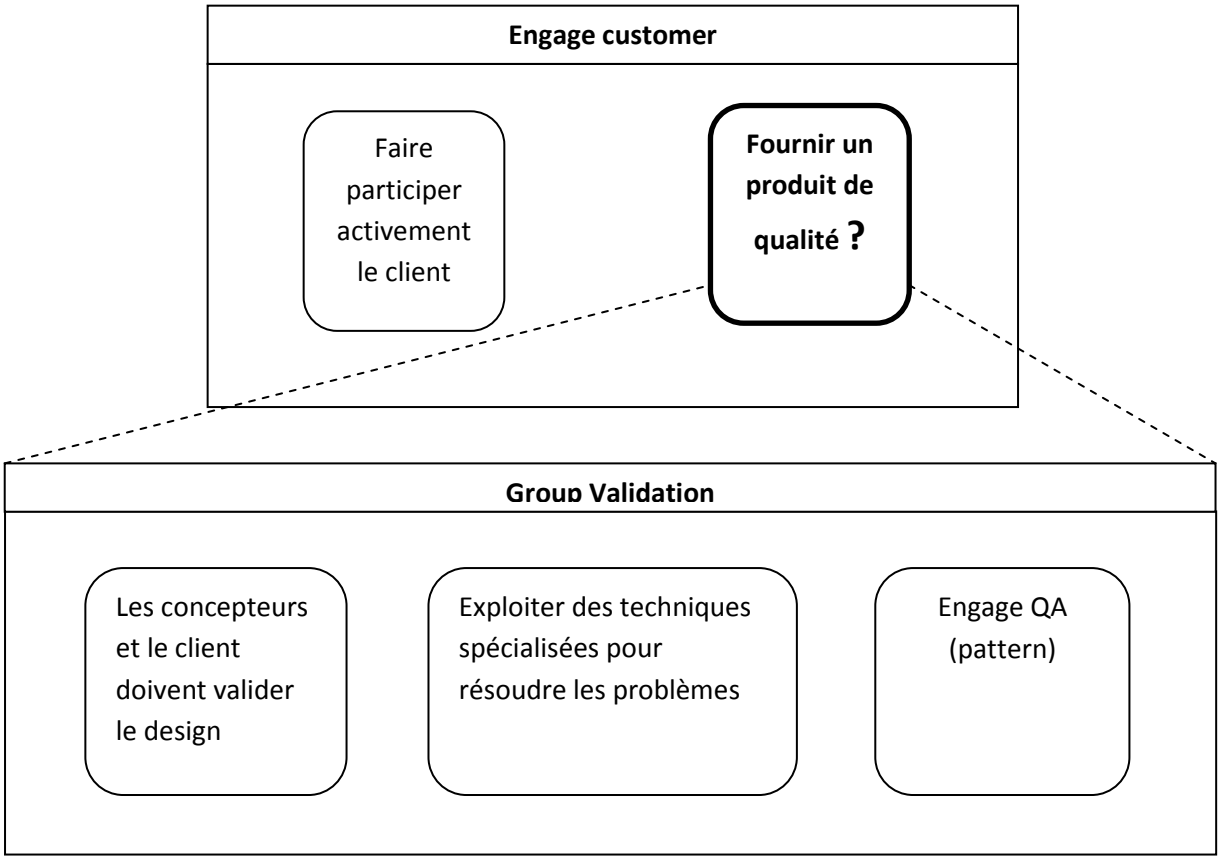


Figure III.10. La relation Uses entre les patrons Engage Customer et Group Validation

III.2.2. Traitement des patrons manquants d'une rubrique de contexte final

III.2.2.1. Notre proposition

Nous proposons de pallier à l'absence d'une rubrique dédiée au contexte final (Resulting Context), par l'utilisation de la rubrique Solution qui inclut le contexte résultant de l'application du patron. Dans ce qui suit, nous donnons des exemples de rubriques Solution incluant le contexte final du patron :

- Le patron *Write Tests to Enable Evolution* **[Demeyer03]** a pour but d'instaurer une politique de test, pour optimiser la réingénierie d'un code hérité. Le contexte résultant de ce patron est que les tests une fois automatisés, seront de plus en plus pratiqués par les développeurs ; que la persistance des tests informe les développeurs sur l'objectif de la réingénierie ; et que l'indépendance des tests favorise leur crédibilité. Ce contexte est cité dans la Solution du patron :*"By minimizing the effort needed to run tests, developers will hesitate less to use the tests. Stored tests document the way the system is expected to work. Independence minimizes distrust in the tests"*.

- Le patron *Compare Code Mechanically* **[Demeyer03]** a pour but de détecter la duplication de portions de code. Le contexte résultant de ce patron est que certaines

duplications ne pourront être détectées, lorsqu'il s'agit de variables renommées dans le code. Ce contexte est cité dans la rubrique Solution du patron : *"This approach may fail to identify some instances of duplicated code due to renaming of variables"*.

- Le patron *Focused Information* **[Ruping03]** a pour but de fournir un contenu pertinent par document. Le contexte résultant de l'application de ce patron est la réduction mais pas la suppression totale d'informations redondantes. Ce contexte est cité dans la rubrique Solution du patron : *"The consequence of this pattern is that you avoid redundant information to some degree, but not entirely. Small overlaps between documents are fine as long as they are necessary to make documents self-contained"*.

- Le patron *Structured Information* **[Ruping03]** a pour but de structurer le contenu d'un document, de façon à ce que le contenu soit facilement accessible. Le contexte résultant de l'application de ce patron est l'aboutissement à un document bien structuré. Ce contexte est cité dans la rubrique Solution du patron : *"This pattern allows you to create a well-structured and evenly-balanced document"*.

- Le patron *Component* **[Volter00]** a pour but de séparer les besoins fonctionnels des besoins techniques d'une application, pour favoriser la réutilisation. Le contexte résultant de ce patron est que l'application produite via ce patron sera constituée de composants faiblement couplés, ce qui favorise leurs réutilisations. Ce contexte est cité dans la rubrique Solution du patron : *"An application is made up of a set of loosely coupled components"*.

- Le patron *Declare Before First Use* **[XmlPatterns00]** a pour but d'assurer dans un document XML que la déclaration d'un élément soit positionné avant la référence à ce dernier. Le contexte résultant de l'application de ce patron est l'augmentation de la probabilité de traitement du document en un seul parcours. Ce contexte est cité dans la rubrique Solution du patron : *"This gives the processing software a better chance of doing a single pass traversal of the document"*.

- Le patron *Fill-in-the-Blanks* **[Tidwell05]** a pour but d'agencer les champs de saisie dans une IHM, de façon à former une phrase. Le contexte résultant de l'application de ce patron est la nécessité de redéposer les éléments de l'IHM dés que la langue de cette dernière change, car la sémantique change avec l'ordre des éléments. Ce contexte est cité dans la rubrique Solution : *"There's a big "gotch" in this pattern. You may have to rearrange the UI to make it work in a different language"*.

- Le patron *Visual Framework* **[Tidwell05]** a pour but de fournir une même charte graphique pour plusieurs IHM ou pages web. Le contexte résultant de l'application de ce patron est la séparation des éléments de style des éléments du contenu. Ce contexte est cité dans la rubrique Solution : *"Implementation of a visual framework should force you to separate stylistic aspects of the UI from the content"*.

- **…**

III.2.2.2. Analyse de notre proposition

Notre proposition se base sur la rubrique Solution qui est présente dans tous les formalismes ; par conséquent, notre proposition a l'avantage d'être valable pour tous les formalismes de patrons.

Nous donnons à titre d'exemple quelques catalogues dont les patrons ne peuvent être analysés par la méthode de Kubo et al. :

[XmlPatterns00],

[Lammi],

[Crumlish09],

[Tidwell05],

[YDPL],

[FUIDP],

…

Les patrons de ces catalogues possèdent les rubriques Starting Context, Forces, mais ne possèdent pas une rubrique Resulting Contexte. Heureusement, notre proposition permet de les traiter.

Aussi, notre proposition est avantageuse car l'utilisation de la rubrique Solution n'altère pas la signification des différentes relations, soit parce que la rubrique Solution n'est utilisée que pour le besoin de représentation du patron, soit parce que la sémantique de la relation demeure inchangée lors de l'utilisation de la Solution à la place du contexte final. Les explications pour chaque type de relation sont données dans ce qui suit.

La relation *Starting-Starting* :

Pour analyser la relation *Starting-Starting* on se base sur l'élément Starting Context (cf. II.2.3.1). L'élément Resulting Context n'est utilisé que pour le besoin de représentation du patron. Ainsi, l'utilisation de la rubrique Solution à la place de la rubrique Resulting Context n'affecte nullement du point de vue sémantique, l'analyse de la relation *Starting-Starting*.

La relation *Refines* :

Pour analyser la relation *Refines* on se base sur les éléments Starting Context et Forces (cf. III.2.1.3) ; l'élément Resulting Context n'est utilisé que pour le besoin de représentation du patron. Ainsi, l'utilisation de la rubrique Solution à la place de l'élément Resulting Context n'affecte nullement du point de vue sémantique, l'analyse de la relation *Refines*.

La relation *Same* :

Il est vrai que lorsque deux patrons partagent des contextes initiaux similaires et des solutions similaires, alors cela signifie que ces deux patrons traitent le même problème et fournissent le même résultat.

Donc, on peut utiliser la rubrique Solution à la place de la rubrique Resulting Context pour analyser la relation *Same* (cf. II.2.3.1).

Nous donnons dans ce qui suit des exemples de patrons liés par la relation *Same*, dont le lien est cité par les auteurs des patrons. Nous considérons ces relations comme correctes, et nous appliquons notre méthode sur ces patrons.

- Les patrons *Ratings* **[Lammi]** et *Rating an Object* **[YDPL]** ont pour but de donner à l'utilisateur un moyen d'exprimer son opinion dans une interface, sans interrompre ses activités en cours.

  Ces patrons sont liés par la relation *Same*, citée explicitement par l'auteur de *Ratings* : "*This pattern is based on the Rating an Object pattern from Yahoo! Design Pattern Library*".

  La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

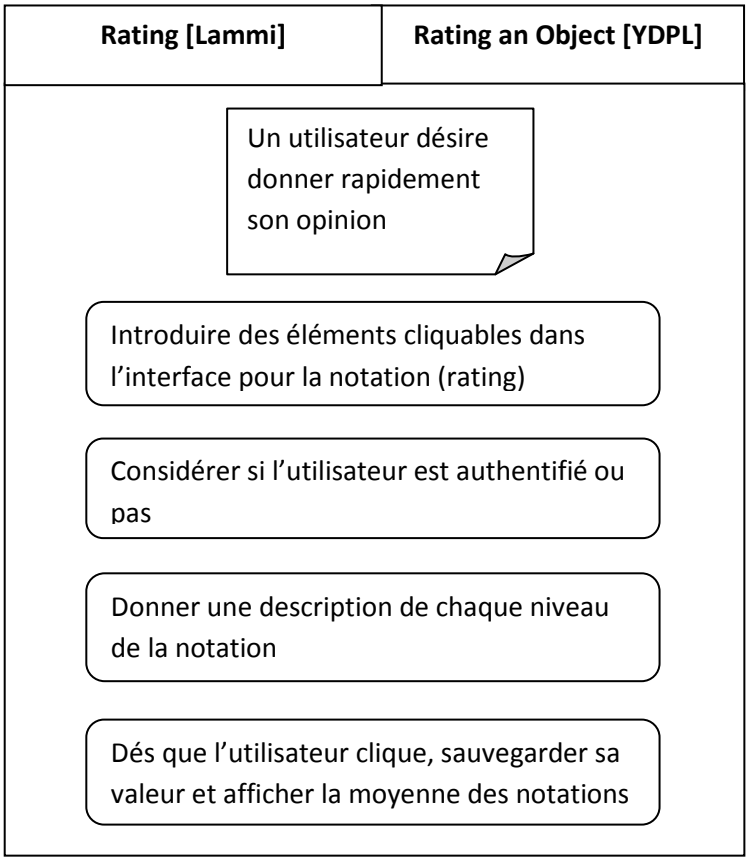


Figure III.11. La relation Same entre les patrons Ratings et Rating an Object

Notre expérimentation donne ce qui suit :

Éléments comparés	Résultats
Contexte initial (CI) 1, Contexte initial 2	Similarité = 1
	CI1 inclut CI2 = False
	CI2 inclut CI1 = False
Contexte final (CF) 1, Contexte final 2	Similarité = 1
	CF1 inclut CF2 = False
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0,178
	F1 inclut F2 = True
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,176
Contexte final 2, Contexte initial 1	Similarité = 0,176

Table III.19. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	Valeur de la relation
P1 Uses P2	0
P1 Refines P2	0,589
P2 Uses P1	0
P2 Refines P1	0
Same	1
Starting-Starting	1
Resulting-Starting (P1P2)	0,176
Resulting-Starting (P2P1)	0,176

Table III.20. Résultats des calculs de relations entre le couple de patron

Donc, pour cet exemple il s'agit bien de la relation *Same*.

- Les patrons *Navigation Tabs* **[Lammi]** et *Navigation Tabs* **[YDPL]** s'intéressent au cas où un utilisateur navigue à travers un site, pour atteindre un contenu (ou une fonctionnalité) en ayant une indication sur l'emplacement de ce dernier.

Ces patrons sont liés par la relation *Same*, citée explicitement par l'auteur de *Navigation Tabs* **[Lammi]** : "*The pattern is based on Navigation Tabs pattern from Yahoo! Design Pattern Library*".

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

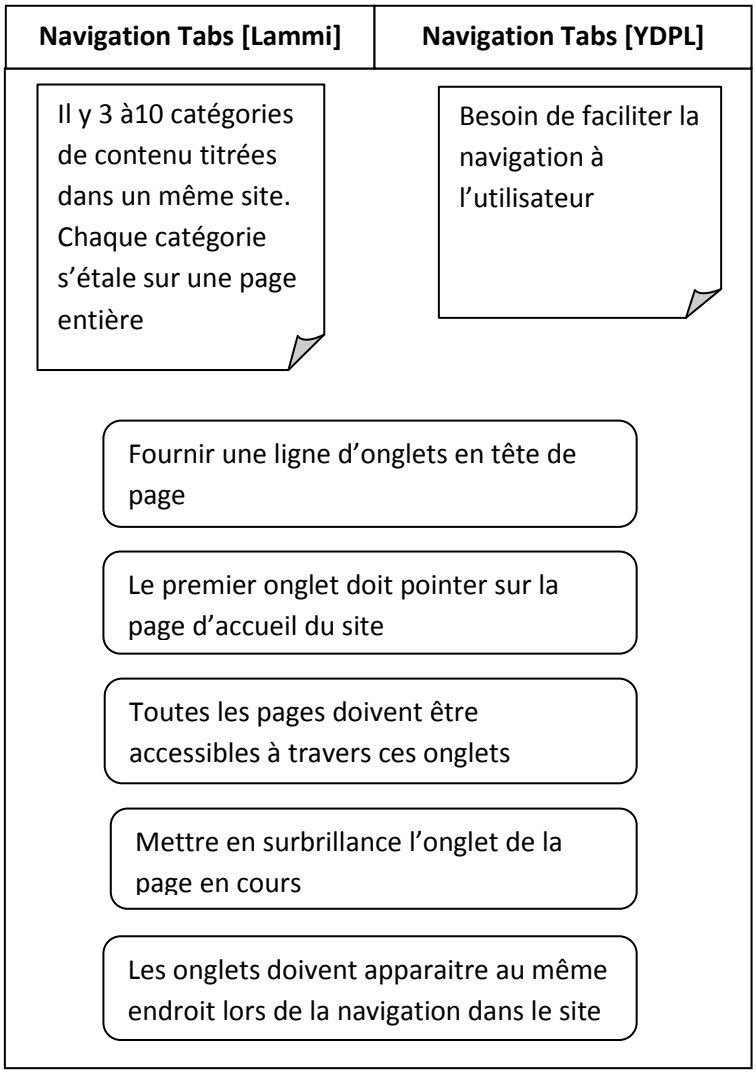


Figure III.12. La relation Same entre les patrons Navigation Tabs [Lammi] et Navigation Tabs [YDPL]

Notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,934
	CI1 inclut CI2 = False
	CI2 inclut CI1 = False

Contexte final (CF) 1, Contexte final 2	Similarité = 0,956
	CF1 inclut CF2 = True
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 1
	F1 inclut F2 = False
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,136
Contexte final 2, Contexte initial 1	Similarité = 0,135

Table III.21. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0
Same	0,945
Starting-Starting	0,934
Resulting-Starting (P1P2)	0,136
Resulting-Starting (P2P1)	0,135

Table III.22. Résultats des calculs de relations entre le couple de patron

Ainsi, pour cet exemple il s'agit bien de la relation *Same*.

La relation *Resulting-Starting* :

Il est vrai que lorsque la solution d'un patron et le contexte initial d'un autre sont similaires, cela signifie que le dernier patron (celui dont on s'intéresse au Contexte Initial) s'appliquera après le premier patron (celui dont on s'intéresse à la Solution), puisque la solution du premier patron fournit les prés conditions nécessaires à l'application du second.

Donc, on peut utiliser la rubrique Solution à la place de la rubrique Resulting Context pour analyser la relation *Resulting-Starting* (cf. II.2.3.1).

Nous donnons dans ce qui suit des exemples de patrons liés par la relation *Resulting-Starting*, dont le lien est cité par les auteurs des patrons. Nous considérons ces relations comme correctes, et nous appliquons notre méthode sur ces patrons.

- Le patron *Use XML* **[XmlPatterns00]** (appelé P2) détermine quand l'utilisation de la technologie XML est appropriée. Le patron *Referenced Note* **[XmlPatterns00]** (appelé P1) s'intéresse à l'utilisation des notes de références en XML.
  Ces deux patrons sont liés par la relation *Resulting-Starting*, citée explicitement par l'auteur du patron *Use XML* : " *All other patterns in this language depend on first using this pattern*".

  La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

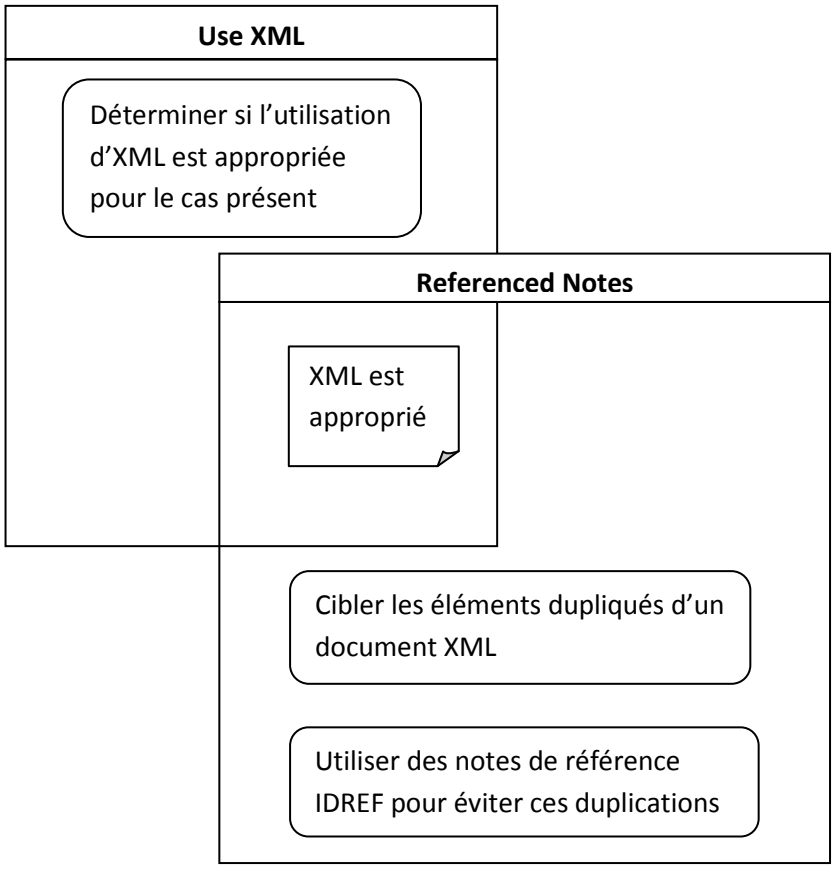


Figure III.13. La relation Resulting-Starting entre Use XML et Referenced Note

Notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,010
	CI1 inclut CI2 = False
	CI2 inclut CI1 = True

Contexte final (CF) 1, Contexte final 2	Similarité = 0
	CF1 inclut CF2 = True
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0
	F1 inclut F2 = False
	F2 inclut F1 = True
Contexte final 1, Contexte initial 2	Similarité = 0
Contexte final 2, Contexte initial 1	Similarité = 0,062

Table III.23. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0
Same	0
Starting-Starting	0
Resulting-Starting (P1P2)	0
Resulting-Starting (P2P1)	0,062

Table III.24. Résultats des calculs de relations entre le couple de patron

Donc, pour cet exemple il s'agit de la relation *Resulting-Starting*.

- Le patron *Titled Sections* **[Tidwell05]** (appelé P1) permet de scinder un contenu en sections. Le patron *Closable Panels* **[Tidwell05]** (appelé P2) s'intéresse à la présentation d'interfaces, ayants un contenu excessif mais divisé en sections.

  Ces deux patrons sont liés par la relation *Resulting-Starting*, citée explicitement par l'auteur du patron *Closable Panels* : "*Use when : The content is divisible into clearly-named sections, as with Titled Sections*".

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

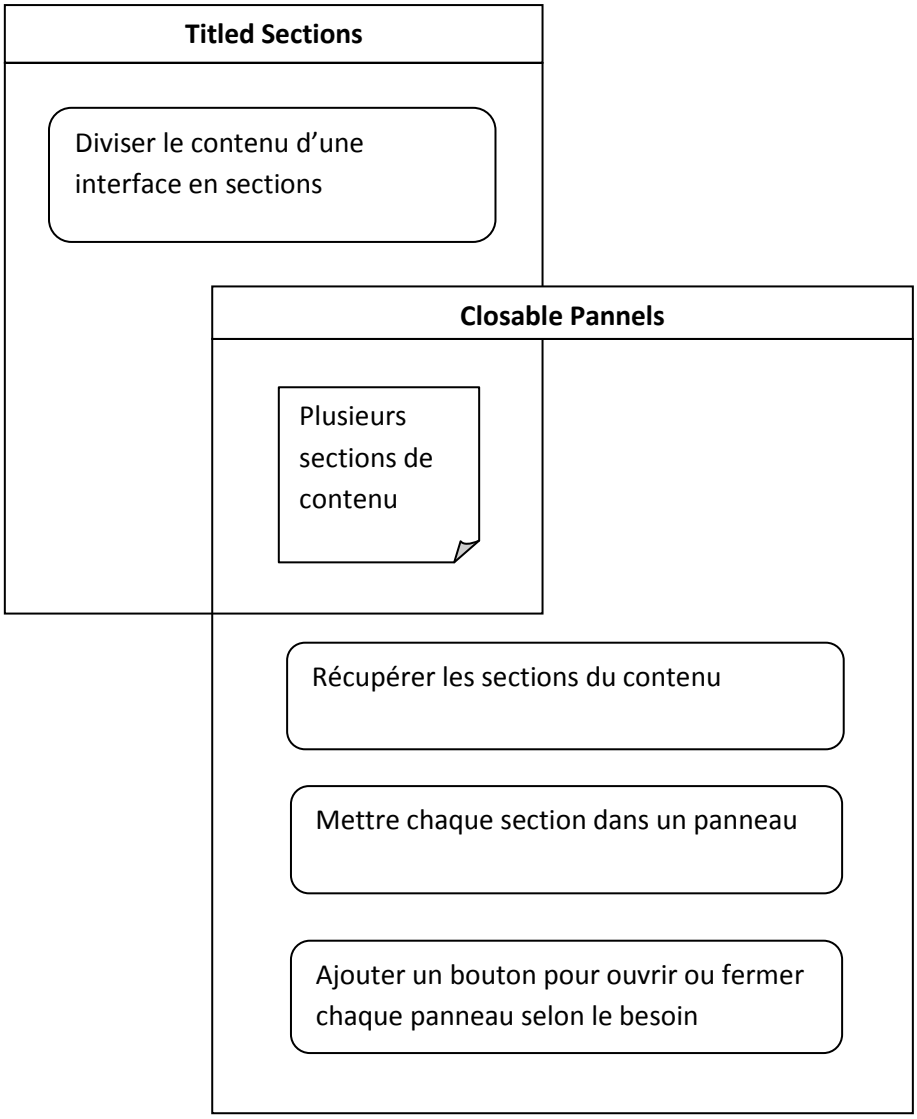


Figure III.14. La relation Resulting-Starting entre les patrons Titled Sections et Closable Panels

Notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,156
	CI1 inclut CI2 = False
	CI2 inclut CI1 = True
Contexte final (CF) 1, Contexte final 2	Similarité = 0,119
	CF1 inclut CF2 = True
	CF2 inclut CF1 = False

Forces (F) 1, Forces 2	Similarité = 0,097
	F1 inclut F2 = False
	F2 inclut F1 = True
Contexte final 1, Contexte initial 2	Similarité = 0,248
Contexte final 2, Contexte initial 1	Similarité = 0,060

Table III.25. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0,127
Same	0,137
Starting-Starting	0,156
Resulting-Starting (P1P2)	0,248
Resulting-Starting (P2P1)	0

Table III.26. Résultats des calculs de relations entre le couple de patron

Ainsi, pour cet exemple il s'agit bien de la relation *Resulting-Starting*.

La relation *Uses* :

Il est vrai que lorsque le contexte initial et la solution d'un patron sont inclus respectivement dans le contexte initial et la solution d'un autre patron, alors cela signifie que le second patron utilise le premier. Donc, on peut utiliser la rubrique Solution à la place de la rubrique Resulting Context pour analyser la relation *Uses* (cf. III.2.1.2).

Nous donnons dans ce qui suit des exemples de patrons liés par la relation *Uses*, dont le lien est cité par les auteurs des patrons. Nous considérons ces relations comme correctes, et nous appliquons notre méthode sur ces patrons.

- Le patron *Extras On Demand* **[Tidwell05]** (appelé P1) gère les pages ayants beaucoup de contenu, où une partie de ce dernier est plus importante que les autres. Le patron *Closable Panels* **[Tidwell05]** (appelé P2) s'intéresse aux interfaces ayants un contenu excessif mais divisé en sections.

Ces deux patrons sont liés par la relation *Uses*, citée explicitement par l'auteur du patron *Extras On Demand* : " *Closable Panels pattern is one way to do this*".

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

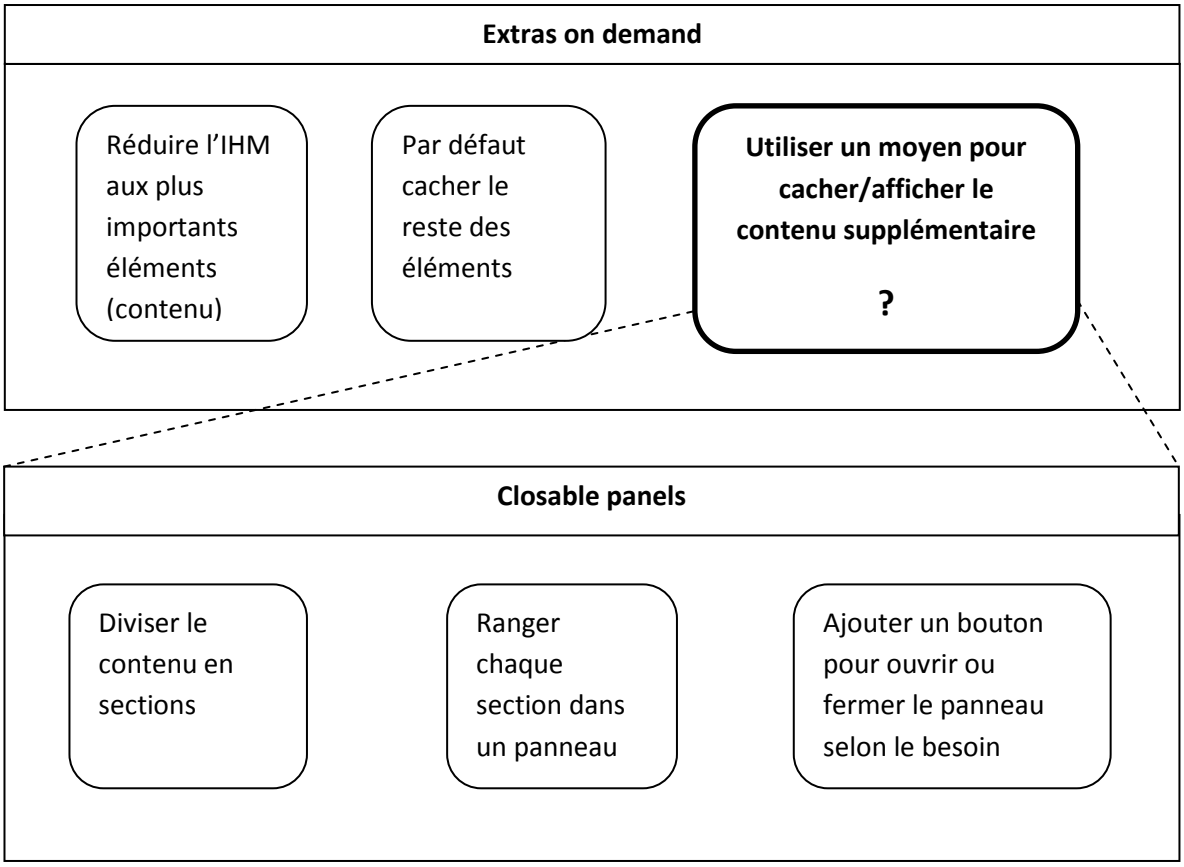


Figure III.15. La relation Uses entre les patrons Extras On Demand et Closable Panels

Notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,216
	CI1 inclut CI2 = True
	CI2 inclut CI1 = False
Contexte final (CF) 1, Contexte final 2	Similarité = 0,223
	CF1 inclut CF2 = True
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0,130
	F1 inclut F2 = True

	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,073
Contexte final 2, Contexte initial 1	Similarité = 0,114

Table III.27. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0,220
P1 Refines P2	0,173
P2 Uses P1	0
P2 Refines P1	0
Same	0,220
Starting-Starting	0,216
Resulting-Starting (P1P2)	0,073
Resulting-Starting (P2P1)	0,114

Table III.28. Résultats des calculs de relations entre le couple de patron

Ainsi, pour cet exemple il s'agit bien de la relation *Uses*.

- Le patron *Action Panel* **[Tidwell05]** (appelé P2) s'intéresse à la présentation d'une interface, ayant une multitude d'actions organisées en catégories. Le patron *Closable Panels* **[Tidwell05]** (appelé P1) s'intéresse aux interfaces avec un contenu excessif mais divisé en sections.

  Ces deux patrons sont liés par la relation *Uses*, citée explicitement par l'auteur du patron *Action Panel* : "*you need to decide how to structure the actions you need to present. Here are some ways you could do it : Closable Panels …*".

  La figure III.16 donne un synopsis des deux patrons, et du lien entre eux.

  Donc, on considère cette relation (citée par l'auteur) comme une relation correcte, et nous appliquons notre méthode pour retrouver le même résultat. Notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,066

	CI1 inclut CI2 = False
	CI2 inclut CI1 = True
Contexte final (CF) 1, Contexte final 2	Similarité = 0,155
	CF1 inclut CF2 = False
	CF2 inclut CF1 = True
Forces (F) 1, Forces 2	Similarité = 0,143
	F1 inclut F2 = False
	F2 inclut F1 = True
Contexte final 1, Contexte initial 2	Similarité = 0,058
Contexte final 2, Contexte initial 1	Similarité = 0,071

Table III.29. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0,110
P2 Refines P1	0,104
Same	0,110
Starting-Starting	0,066
Resulting-Starting (P1P2)	0
Resulting-Starting (P2P1)	0,071

Table III.30. Résultats des calculs de relations entre le couple de patron

Donc, pour cet exemple il s'agit de la relation *Uses*.

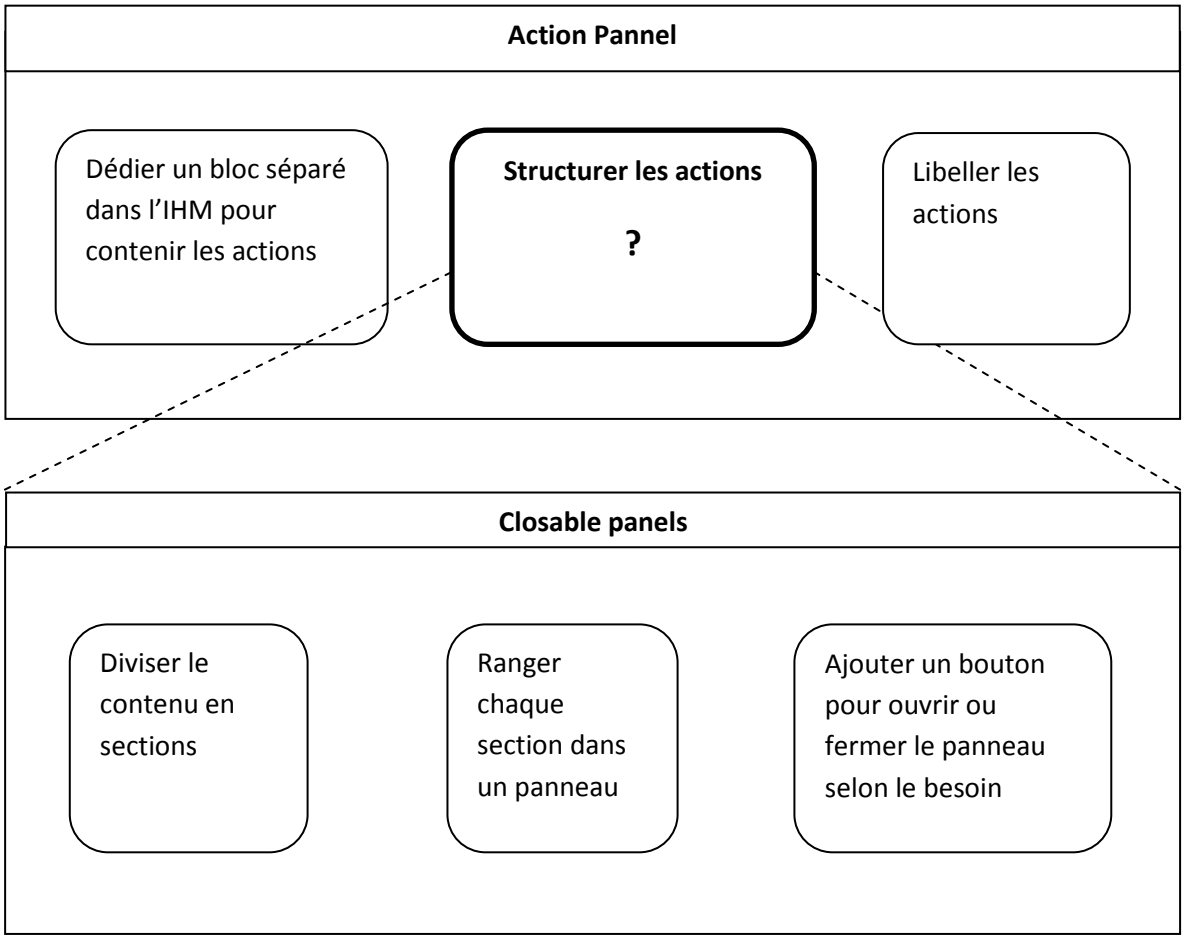


Figure III.16. La relation Uses entre les patrons Action Panel et Closable Panels

Le seul inconvénient quand à l'utilisation de la rubrique Solution pour représenter le contexte final, est que cette dernière peut être exprimée exclusivement de manière non textuelle (dans un langage de modélisation par exemple). Dans ce cas, notre méthode ne peut être appliquée, car elle se base sur des techniques de traitement de textes.

III.2.2.3. Expérimentation

Nous nous intéressons dans la suite de ce chapitre à des exemples portants sur les relations *Uses*, *Same* et *Resulting-Starting*, vu que pour les autres relations (*Starting-Starting*, *Refines*) la rubrique Solution n'est utilisée que pour le besoin de représentation des patrons.

Nous donnons dans ce qui suit quelques relations entre patrons, qui ne sont pas citées par les auteurs mais qui sont détectées par notre méthode. Ces relations sont correctes pour les raisons que nous allons donner pour chaque couple de patrons :

- Pour les patrons *Auto Complete* **[YDPL]** et *Autocomplete* **[Lammi]**, notre expérimentation donne ce qui suit :

Éléments comparés	Résultats
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,161
	CI1 inclut CI2 = True
	CI2 inclut CI1 = False
Contexte final (CF) 1, Contexte final 2	Similarité = 0,965
	CF1 inclut CF2 = False
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0,430
	F1 inclut F2 = False
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,395
Contexte final 2, Contexte initial 1	Similarité = 0,080

Table III.31. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	Valeur de la relation
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0
Same	0,563
Starting-Starting	0,161
Resulting-Starting (P1P2)	0,395
Resulting-Starting (P2P1)	0,080

Table III.32. Résultats des calculs de relations entre le couple de patron

Dans cet exemple la relation à considérer pour les deux patrons est *Same*. En effet, cette relation est correcte pour les raisons suivantes :

Les deux patrons fournissent une fonctionnalité d'auto saisie, dans les champs de texte d'une IHM.

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

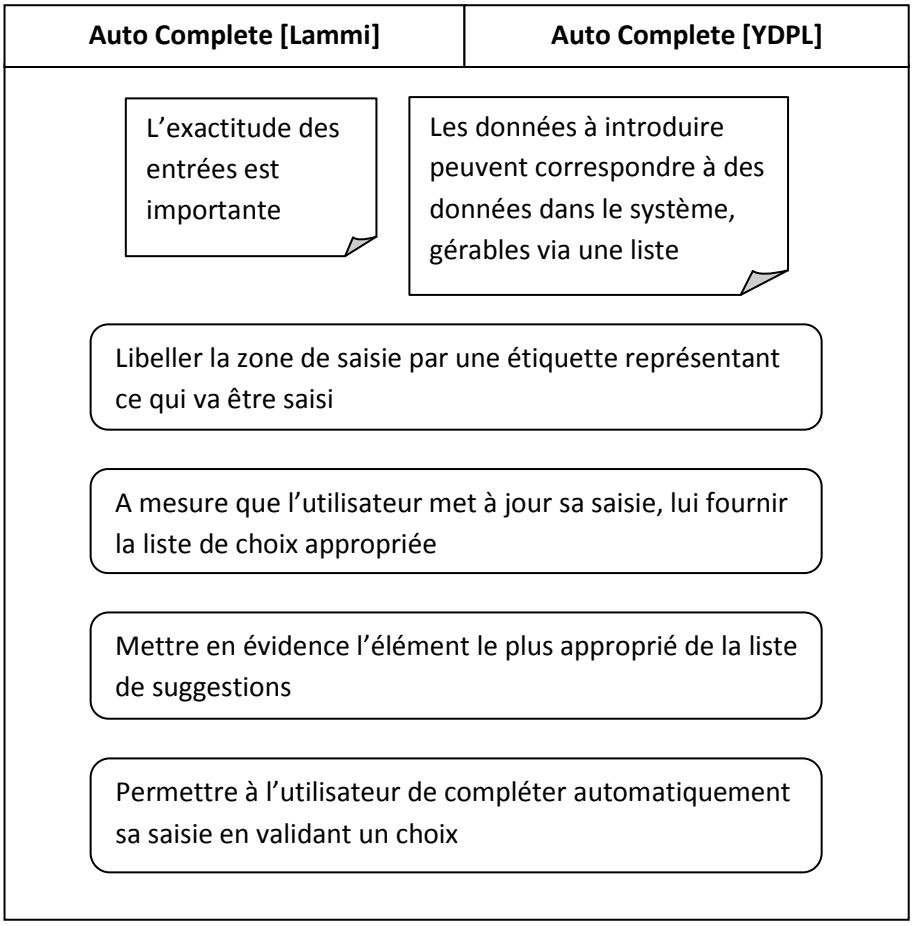


Figure III.17. La relation Same entre les patrons Auto Complete [YDPL] et Autocomplete [Lammi]

Les contextes initiaux de ces deux patrons sont similaires, les deux patrons s'appliquent dans *le cas d'une IHM où l'utilisateur doit entrer une donnée qui doit être écrite correctement, ou qui peut être choisie dans un ensemble restreint.*

Les solutions de ces deux patrons sont similaires, et donnent le même résultat en permettant *la proposition d'une liste des choix au fur et à mesure que l'utilisateur saisie son entrée, et compléter automatiquement la saisie de ce dernier dés qu'il valide un choix.*

- Pour les patrons *Drag and Drop Modules* **[YDPL]** et *Drag and Drop Modules* **[Lammi]**, notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,526
	CI1 inclut CI2 = False
	CI2 inclut CI1 = False

Contexte final (CF) 1, Contexte final 2	Similarité = 0,979
	CF1 inclut CF2 = False
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0,175
	F1 inclut F2 = False
	F2 inclut F1 = True
Contexte final 1, Contexte initial 2	Similarité = 0,052
Contexte final 2, Contexte initial 1	Similarité = 0,083

Table III.33. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0,351
Same	0,752
Starting-Starting	0,526
Resulting-Starting (P1P2)	0
Resulting-Starting (P2P1)	0,083

Table III.34. Résultats des calculs de relations entre le couple de patron

Pour cet exemple la relation à considérer pour les deux patrons est ***Same***. En effet, cette relation est correcte pour les raisons suivantes :

Les deux patrons traitent la possibilité d'un utilisateur de personnaliser une page web, directement avec sa souris. La figure III.18 donne un synopsis des deux patrons, et du lien entre eux.

Les contextes initiaux de ces deux patrons sont similaires, les deux patrons s'applique dans la cas où *on conçoit une page web personnalisable par l'utilisateur, et on veut éviter à ce dernier d'aller dans une autre page pour personnaliser sa page en cours.*

Les solutions de ces deux patrons sont similaires, et donnent le même résultats en permettant *la gestion des acteurs de l'interface (contenu, curseur, emplacement original, emplacement cible …), impliqués lors d'un comportement particulier de Glisser-Déposer (génération de la page, début de l'opération Glisser …).*

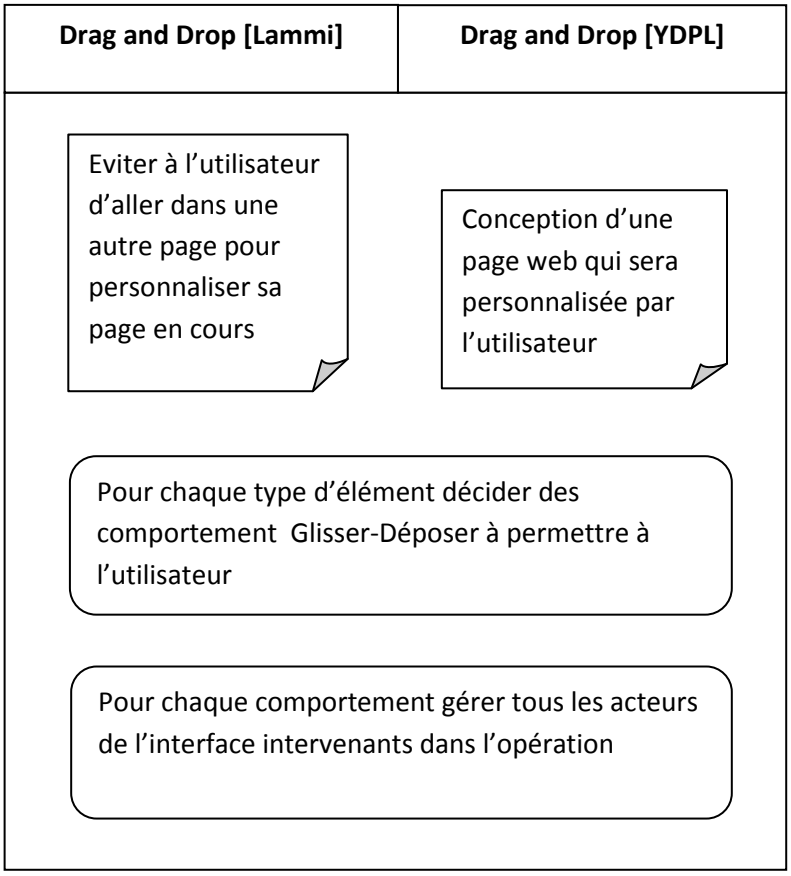


Figure III.18. La relation Same entre les patrons Drag and Drop Modules [YDPL] et Drag and Drop Modules [Lammi]

- Pour les patrons *Deep Background* **[Tidwell05]** (appelé P2) et *Constrained Resize* **[Tidwell05]** (appelé P1), notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,013
	CI1 inclut CI2 = True
	CI2 inclut CI1 = False
Contexte final (CF) 1, Contexte final 2	Similarité = 0,094
	CF1 inclut CF2 = True

	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0,008
	F1 inclut F2 = True
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,004
Contexte final 2, Contexte initial 1	Similarité = 0,062

Table III.35. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0
Same	0
Starting-Starting	0
Resulting-Starting (P1P2)	0
Resulting-Starting (P2P1)	0,062

Table III.36. Résultats des calculs de relations entre le couple de patron

Pour cet exemple la relation à considérer pour les deux patrons est *Resulting-Starting*. En effet, cette relation est correcte pour les raisons suivantes :

La solution du patron *Deep Background* qui s'intéresse à l'utilisation d'une image comme arrière plan, produit ce qui suit : *l'utilisation d'un fond (en arrière plan) pour mettre l'accent sur le contenu de l'avant plan, en prenant en considération que le fond doit s'adapter au redimensionnement subi par l'IHM*. Ce résultat est similaire au contexte initial du patron *Constrained Resize,* qui est *la gestion des redimensionnements d'une IHM*.

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

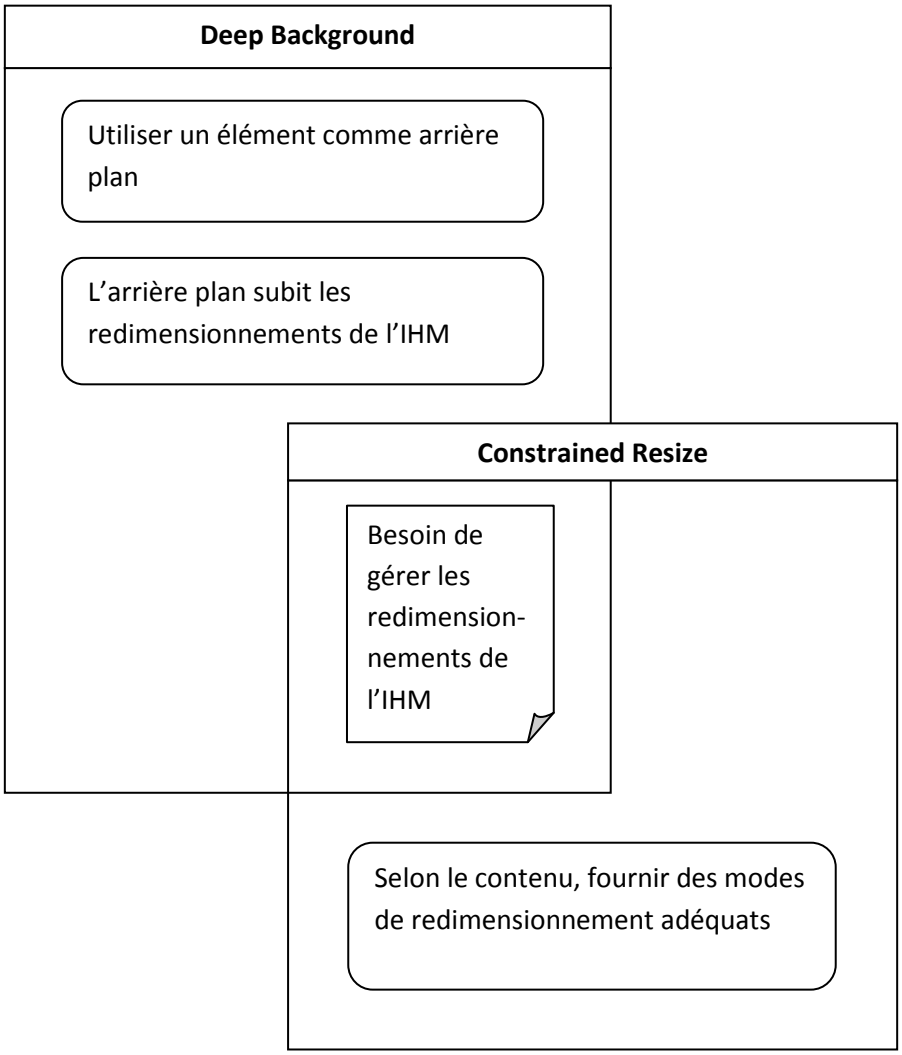


Figure III.19. La relation Resulting-Starting entre les patrons Deep Background et Constrained Resize

- Pour les patrons *Sortable Table* **[Tidwell05]** et *Row Striping* **[Tidwell05]**, notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0
	CI1 inclut CI2 = False
	CI2 inclut CI1 = True
Contexte final (CF) 1, Contexte final 2	Similarité = 0,059
	CF1 inclut CF2 = False
	CF2 inclut CF1 = True

Forces (F) 1, Forces 2	Similarité = 0,096
	F1 inclut F2 = False
	F2 inclut F1 = True
Contexte final 1, Contexte initial 2	Similarité = 0
Contexte final 2, Contexte initial 1	Similarité = 0,091

Table III.37. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0
Same	0
Starting-Starting	0
Resulting-Starting (P1P2)	0
Resulting-Starting (P2P1)	0,091

Table III.38. Résultats des calculs de relations entre le couple de patron

Pour cet exemple la relation à considérer pour les deux patrons est ***Resulting-Starting***. En effet, cette relation est correcte pour les raisons suivantes :

Le patron *Sortable Table* préconise l'utilisation de tables, pour la présentation de résultats dans une interface. La solution de ce patron produit ce qui suit : *présentation d'un contenu dans une table, en prenant en compte que le contenu doit être exposé de façon claire*. Ce résultat est similaire au contexte initial du patron *Row Striping*, qui est *le besoin de séparation des lignes d'une table pour la clarté du contenu*.

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

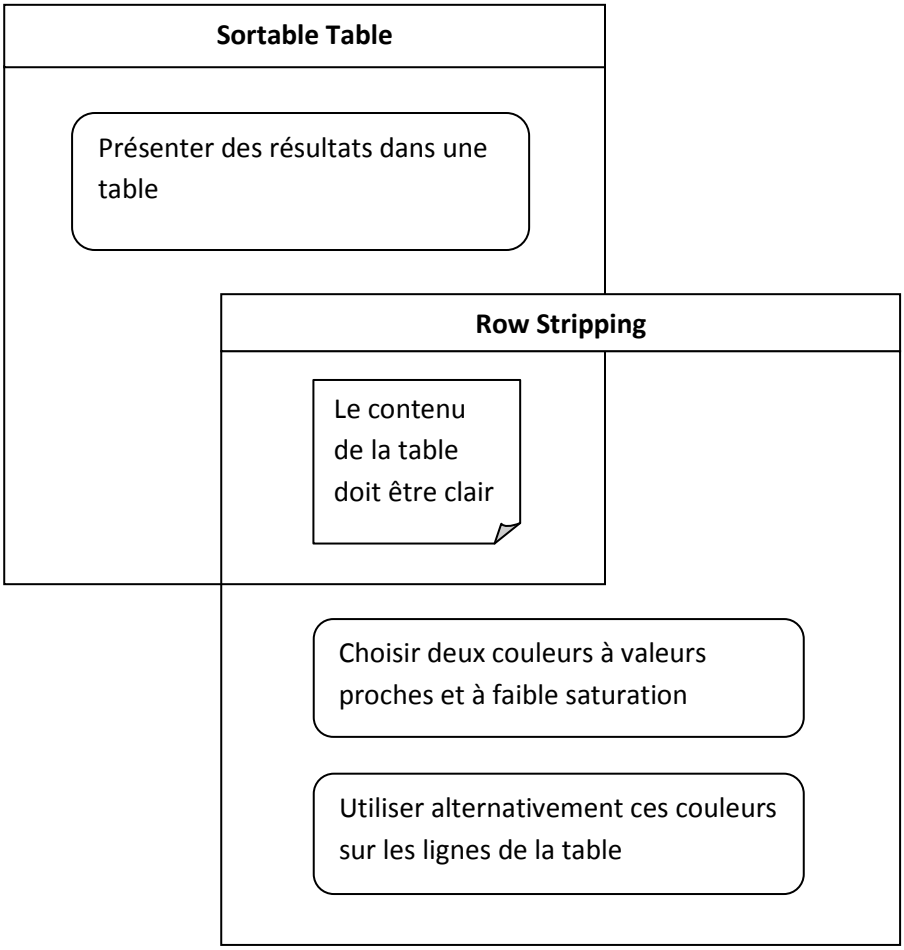


Figure III.20. La relation Resulting-Starting entre Sortable Table et Row Striping

- Pour les patrons *Wizard* **[Tidwell05]** (appelé P1) et *Good Defaults* **[Tidwell05]** (appelé P2), notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,143
	CI1 inclut CI2 = True
	CI2 inclut CI1 = False
Contexte final (CF) 1, Contexte final 2	Similarité = 0,155
	CF1 inclut CF2 = True
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0,108
	F1 inclut F2 = False
	F2 inclut F1 = True

Contexte final 1, Contexte initial 2	Similarité = 0,079
Contexte final 2, Contexte initial 1	Similarité = 0,142

Table III.39. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0,149
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0,126
Same	0,149
Starting-Starting	0,143
Resulting-Starting (P1P2)	0,079
Resulting-Starting (P2P1)	0,142

Table III.40. Résultats des calculs de relations entre le couple de patron

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

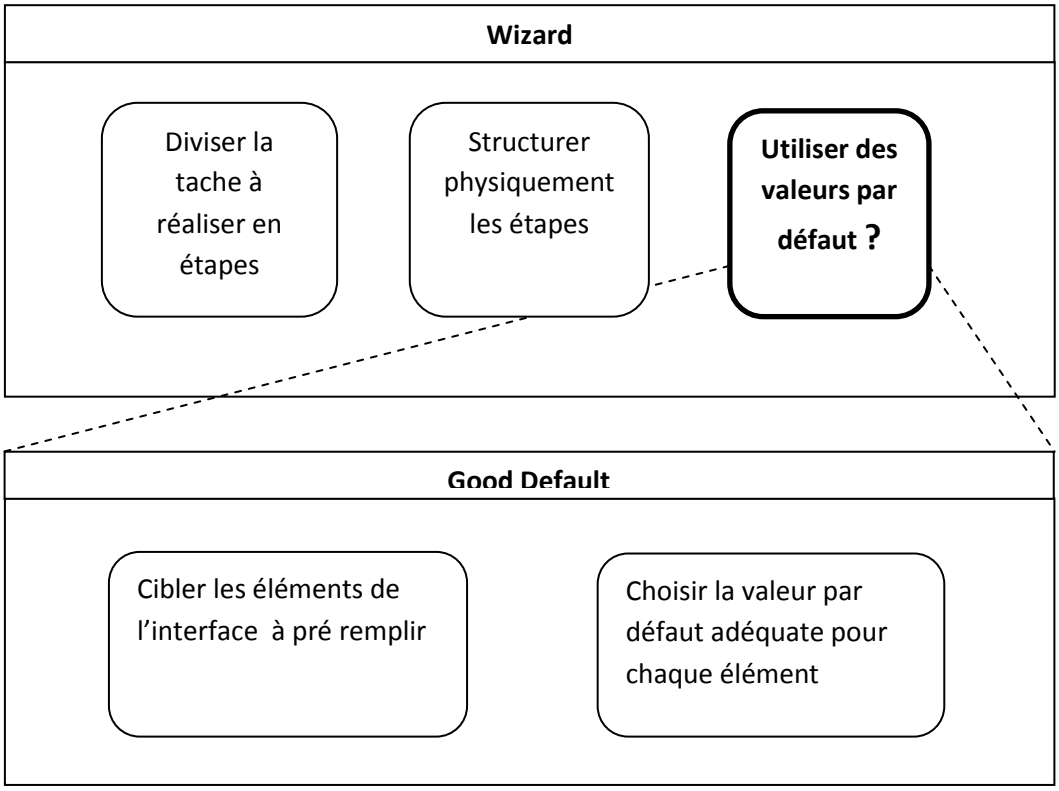


Figure III.21. La relation Uses entre les patrons Wizard et Good Defaults

Le Contexte Initial de *Good Defaults* est inclus dans le Contexte Initial de *Wizard*. Aussi, la solution du patron *Good Defaults* est incluse dans la solution du patron *Wizard*. Donc pour cet exemple, la relation à considérer entre les deux patrons est *P1UsesP2*. En effet, cette relation est correcte pour les raisons suivantes :

Le patron P1 s'applique pour guider l'utilisateur à effectuer une tache étape par étape, à travers plusieurs champs de l'interface. Pour certains champs l'information à saisir est évidente, donc on peut anticiper et utiliser des valeurs par défaut. Ainsi, le patron P1 peut utiliser le patron P2, qui permet de pré renseigner des champs d'une interface avec des valeurs appropriées.

- Pour les patrons *Center Stage* **[Tidwell05]** (appelé P1) et *Titled Sections* **[Tidwell05]** (appelé P2), notre expérimentation donne ce qui suit :

Éléments comparés	**Résultats**
Contexte initial (CI) 1, Contexte initial 2	Similarité = 0,064
	CI1 inclut CI2 = True
	CI2 inclut CI1 = False
Contexte final (CF) 1, Contexte final 2	Similarité = 0,166
	CF1 inclut CF2 = True
	CF2 inclut CF1 = False
Forces (F) 1, Forces 2	Similarité = 0,049
	F1 inclut F2 = False
	F2 inclut F1 = False
Contexte final 1, Contexte initial 2	Similarité = 0,113
Contexte final 2, Contexte initial 1	Similarité = 0,031

Table III.41. Résultats des Similarités et Inclusions entre les éléments du couple de patrons

Type de relation	**Valeur de la relation**
P1 Uses P2	0,115
P1 Refines P2	0
P2 Uses P1	0
P2 Refines P1	0

Same	0,115
Starting-Starting	0,064
Resulting-Starting (P1P2)	0,113
Resulting-Starting (P2P1)	0

Table III.42. Résultats des calculs de relations entre le couple de patron

Le Contexte Initial de *Titled Sections* est inclus dans le Contexte Initial de *Center Stage*. Aussi, la solution du patron *Titled Sections* est incluse dans la solution du patron *Center Stage*. Donc pour cet exemple, la relation à considérer pour les deux patrons est *P1UsesP2*. En effet, cette relation est correcte pour les raisons suivantes :

Le patron P1 s'intéresse à la mise en évidence d'une partie du contenu de l'interface, par rapport à plusieurs autres qui vont être placées tout autour de la partie importante. Pour atteindre son but, le patron P1 peut utiliser le patron P2 qui permet d'avoir des sections de contenu séparées, avec chacune un titre évident ; avant d'établir une organisation visuelle des différentes sections.

La figure suivante donne un synopsis des deux patrons, et du lien entre eux :

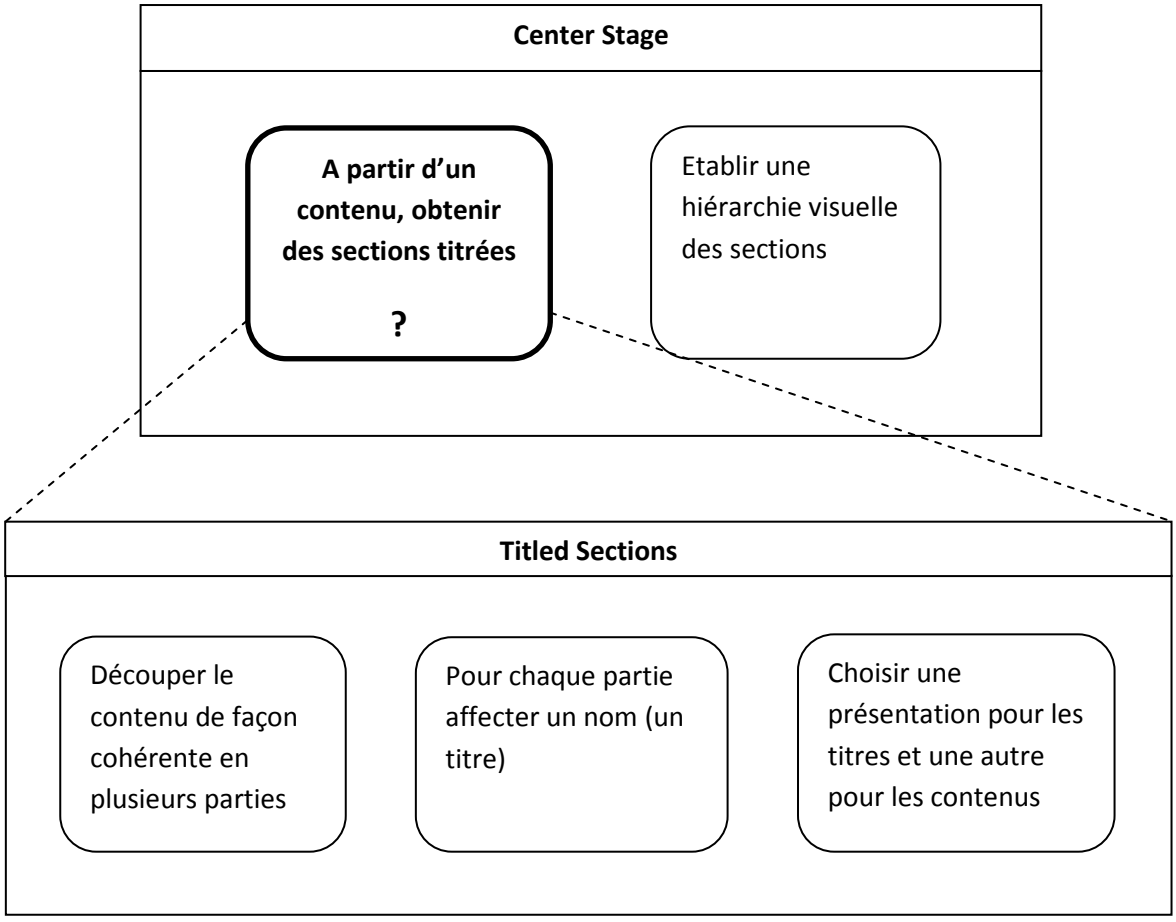


Figure III.22. La relation Uses entre les patrons Center Stage et Titled Sections

III.2.3. Notre prototype

Dans les sections III.2.1 et III.2.2 de ce chapitre, nous avons présenté des exemples qui démontrent le fonctionnement de notre approche. Pour obtenir ces exemples nous étions dans l'obligation de réaliser des tests ; et pour ce faire nous avons développé un prototype que nous décrivons en détails dans ce qui suit.

III.2.3.1 Architecture de notre prototype

La méthode de Kubo et al. est une bonne méthode d'analyse automatique des relations entre patrons (cf. § II.2.3.2). Elle est conçue de façon modulaire, ce qui la rend facile à améliorer. Du coup, notre méthode d'analyse reprend celle de Kubo et al., et lui incorpore les différentes améliorations que nous avons proposées dans ce chapitre.

Les modifications que nous apportons vont affecter seulement quelques blocs de traitement. Les blocs *HTML Document Analysis* et *Pattern form Judgment* vont rester inchangés. Par contre, les blocs *Pattern Extraction* et *Relation Analysis* vont être adaptés pour supporter nos propositions, comme nous l'expliquerons dans ce qui suit.

III.2.3.1.1. Le bloc *Pattern Extraction*

Tables de correspondance

Nous faisons subir aux tables de correspondance utilisées par la méthode de Kubo et al. dans le bloc *Pattern Extraction* un petit ajout. Il s'agit d'indiquer avec une marque spéciale dans chaque table de correspondance, la rubrique Solution qui sera utilisée dans le cas d'absence d'une rubrique de Contexte Final.

L'exemple suivant montre une table de correspondance, avec la rubrique Solution marquée par un astérisque.

Sections	***Elément correspondant du modèle de patrons***
Pattern Name	-
Intent	Contexte Initial
Motivation	Contexte Initial
Applicability	Forces
Solution	*
Consequences	Contexte Final
Related Patterns	-

Table III.43. Notre table de correspondance entre sections et éléments du modèle de patron

Traitement

Les sections extraites à partir d'un patron sont converties dans le bloc *Pattern Extraction*, via une table de correspondance. La conversion abouti en un patron représenté

dans le modèle de Kubo et al. (cf. § II.2.2.3). Ce dernier est constitué des éléments Contexte Initial, Forces et Contexte Final.

Ainsi, pour représenter un patron donné dans le modèle de Kubo et al., plus précisément pour représenter l'élément Contexte Final, nous procédons comme suit :

- S'il existe parmi les sections extraites une section correspondante au Contexte Final, alors nous l'utilisons dans la représentation du patron.
- Sinon, nous utilisons la section marquée avec le symbole spécial (l'astérisque dans l'exemple précédent), qui n'est en réalité autre que la section Solution.

III.2.3.1.2. Le bloc *Relation Analysis*

On calcule la similarité et l'inclusion entre les éléments des couples de patrons, dans le bloc *Relation Analysis*. Ces calculs permettent de distinguer les paires de patrons qui vont être considérées comme des patrons liés.

Pour chaque paire de patrons P1 P2, nous analysons toutes les relations et nous adoptons la plus significative ; les relations sont calculées entre les patrons P1 et P2, et entre P2 et P1.

Est donné dans la figure suivante un synopsis du fonctionnement de notre bloc *Relation Analysis* :

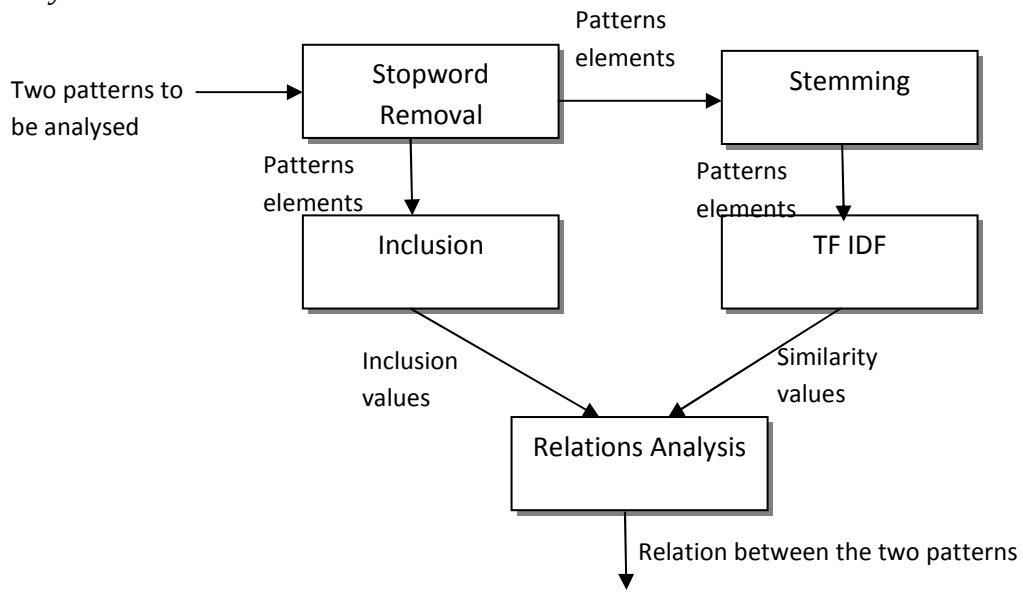


Figure III.23. Procédure interne de notre bloc Relation Analysis

III.2.3.2 Mise en œuvre de notre prototype

L'objectif de cette section est maintenant de décrire l'implémentation de notre prototype d'analyse des relations entre patrons, qui supporte la méthode de Kubo et al. ainsi que les améliorations que nous lui avons apporté.

Nous avons développé ce prototype en *C#* sous l'environnement *Microsoft Visual Studio 2008* version 9.0 (*Microsoft .NET framework* version 3.5 SP1), sous le système d'exploitation Microsoft Windows Vista Édition Familiale Premium Version 6.0.6001 Service Pack 1 Build 6001.
Le choix d'un langage de programmation objet est naturel, car il facilite l'implémentation du prototype qui est élaboré lui aussi selon l'approche orientée objet.

En ce qui concerne le Stemmer exploité, notre travail réutilise le code de Christopher O'Neill **[O'Neill00]** qui implémente le Stemmer de Paice **[Paice90].**
Pour la liste des Stopwords, nous avons utilisée celle de Gerard Salton et Chris Buckley de Cornell University **[SaltonSW]**, qui comporte 571 mots.

A ce stade, notre prototype n'implémente pas tous les blocs de la méthode de Kubo et al. (à savoir : *HTML Document Analysis*, *Pattern form Judgment*, *Pattern Extraction* et *Relation Analysis*) faute de temps. Nous nous sommes contentés d'implémenter le noyau de cette méthode, à savoir le bloc ***Relation Analysis***.

III.3. Conclusion

Le présent chapitre avait pour objectif de proposer des solutions aux problèmes abordés dans ce travail ; à savoir le problème d'exhaustivité de la documentation dans les procédés Agiles et le problème de composition de patrons.

Pour résoudre le problème d'exhaustivité de la documentation, des patrons étaient choisis mais chacun d'eux ne résout qu'une partie du problème. Donc, il a fallu exploiter les relations existantes entre ces patrons, pour pouvoir les composer en une solution complexe dans le but de résoudre le problème dans sa totalité.

En ce qui concerne le problème de composition de patrons, deux principales améliorations à la méthode de Kubo et al. sont apportées.

La première amélioration permet à la méthode d'analyser les relations Uses et Refines entre patrons, en se basant sur notre proposition d'analyse de l'*Inclusion*.

La seconde amélioration permet d'analyser les relations sur des patrons n'ayant pas une rubrique de contexte final, et cela en exploitant la rubrique Solution. Il est montré aussi dans ce chapitre que l'utilisation de la rubrique Solution n'altère pas la signification des relations analysées.

Enfin, ce chapitre a montré notre implémentation du noyau de la méthode d'analyse, qui incorpore toutes nos améliorations présentées dans ce travail. Ce chapitre a aussi montré quelques exemples à valeur ajoutée sur des relations entre patrons.

Des extensions de notre travail restent possibles, quelques unes feront l'objet du chapitre suivant.

CHAPITRE :

CONCLUSION GENERALE

Dans ce chapitre final, nous dressons un bilan des apports de notre travail **(cf. 1 et 2)**, et énumérons quelques perspectives qui pourront êtres poursuivies **(cf. 3)**.

1. Bilan des matières abordées

Les méthodes Agiles sont des procédés de développement logiciel qui offrent une formidable voie pour gérer des cycles de développement rapides, et traiter le changement avec efficacité. Nous avons présenté les particularités ainsi que les intérêts des méthodes Agiles dans le chapitre I. L'importance des méthodes Agile impose une attention particulière pour ces méthodes, et des travaux spéciaux deviennent un impératif.

La documentation de logiciels que nous avons présentée dans la chapitre I, est partie prenante dans un projet et est un composant clé dans la qualité du logiciel. Cette documentation a des particularités dans les projets Agiles où des problèmes récurrents restent posés ; certains sont inhérents à la documentation de logiciels, d'autres sont imposés par les contraintes des projets Agiles. Parmi ces problèmes, celui de l'exhaustivité de la documentation. Nous nous sommes intéressés à ce problème et l'avons exposé en détail dans le chapitre II de ce mémoire.

Un outil très puissant, qui peut offrir une solution à ce problème sont les patrons logiciels. Nous avons donné une introduction et un riche état de l'art sur les patrons dans le chapitre I. Il est avantageux donc d'explorer des catalogues de patrons, à la recherche d'une solution pour notre problème. Le seul catalogue trouvé apte à fournir des patrons dans le sujet de la documentation Agile est celui de Ruping. Nous avons abordé le choix de ce catalogue dans la première partie du chapitre II. Seulement, aucun patron parmi ceux de Ruping ne traite notre problème dans sa totalité, il a fallu par conséquent composer une solution à partir de plusieurs patrons.

La composition de patrons se base principalement sur les relations entre patrons. Ces dernières sont généralement explicites dans les patrons ; le cas contraire, elles doivent être analysées et ressorties pour pouvoir servir la composition de patrons. Nous avons consacré la deuxième partie du chapitre II à l'étude de la composition et de l'analyse des relations entre patrons.

Très peu de chercheurs ont tenté de faire ressortir les relations entre patrons dans le cas où elles ne sont pas explicites, ou lorsqu'elles concernent différents catalogues. Kubo et al. sont les premiers à présenter une approche automatique d'analyse des relations entre patrons de différents catalogues, et les seuls à proposer une approche qui peut traiter différents types de patrons logiciels. Nous avons exposé la méthode de Kubo et al. en détaille, et avons analysé ses lacunes dans le chapitre II.

Dans le cadre de ce travail nous avions deux principaux objectifs, concernant le problème d'exhaustivité de la documentation et concernant la composition de patrons, que nous allons rappeler dans la partie Contributions.

2. Contributions

Les contributions présentées dans ce mémoire s'inscrivent dans un champ encore peu exploré de la recherche, et le travail présenté constitue entre autres un pas vers la mise en œuvre d'approches automatiques pour le développement de systèmes de réutilisation de patrons. Cependant, certains points abordés restent soit à étudier soit à compléter, nous citons quelques uns dans la partie Perspectives de ce chapitre. Principalement, nous avons ciblé deux points comme problématique de notre travail, et nos deux objectifs étaient les suivants :

- Conception d'une solution au problème d'exhaustivité de la documentation dans les procédés Agiles, et cela via la composition des patrons de Ruping.

- Amélioration de la méthode de Kubo et al. sur deux plans : le premier pour reconnaitre les relations Uses et Refines, et le second pour analyser les patrons n'ayants pas une rubrique de contexte final.

A l'issue de ce travail, nous pouvons résumer nos contributions dans les deux principaux résultats suivants :

- Le premier résultat concerne la proposition d'une solution au problème d'exhaustivité de la documentation dans les procédés Agiles, via la composition de patrons de Ruping. Nous avons d'abord ciblé trois patrons Target Readers, Focused Information et Individual Document Requirements du catalogue de Ruping, où chacun résout une partie du problème. Une fois ensemble, ces trois patrons résolvent notre problème dans sa totalité.

  Pour pouvoir composer ces patrons, nous avons analysé les relations implicites existantes entre eux. La composition a eu lieu en reprenant les patrons sans modification, et en se basant sur les relations Resulting-Starting existantes entre les patrons choisis. La solution composée fait l'objet de la première partie du chapitre III.

- Le second résultat concerne l'amélioration de la méthode de Kubo et al. pour pouvoir analyser les relations Uses et Refines. Nous avons proposé un moyen d'analyser ces deux relations, en se basant sur l'idée de Kubo et al. et en introduisant l'analyse de l'inclusion. Cette dernière nous permet de savoir si un élément de patron inclut un autre, et du coup si deux patrons sont liés par la relation Uses ou Refines.

  Nous avons aussi amélioré de la méthode de Kubo et al., en lui donnant la possibilité d'analyser les relations entre des patrons qui ne possèdent pas des rubriques de contexte final. Pour ce faire, nous avons proposé l'exploitation de la rubrique solution et avons montré que la sémantique des relations analysées demeure inchangée.

  Aussi, nous avons implémenté le noyau de la méthode d'analyse, à savoir le bloc Patterns Analysis. Toutes les améliorations apportées à la méthode de Kubo et al. font l'objet de la deuxième partie du chapitre III.

3. Perspectives

Le travail présenté dans ce mémoire peut se prolonger vers plusieurs perspectives ; nous présentons quelques unes dans ce qui suit.

Concernant la documentation Agile :

Notre solution composée peut faire l'objet de la rédaction d'un patron, vu qu'elle résout un problème récurent si important et qui n'est abordé par aucun patron. Une fois le patron rédigé, il sera amélioré à chaque fois que l'opportunité se présente. Le formalisme avantageux de Ruping pourra être utilisé pour exprimer le nouveau patron composé. Pour que ce formalisme soit plus efficace encore, ses lacunes devront être remédiées.

Concernant notre méthode d'analyse :

La première perspective concernant notre méthode d'analyse est de finaliser son implémentation, pour pouvoir lui intégrer par la suite d'autres améliorations telles que la procédure de composition de patrons proprement dite, que nous pouvons considérer comme une forme particulière de la réutilisation de patrons.

Notre méthode peut être enrichi pour pouvoir traiter des patrons où les éléments nécessaires à la représentation (Contexte Initial, Forces, Contexte Final) ne sont pas présents.

Notre méthode d'analyse peut être améliorée en adoptant une méthode de calcul de similarité plus performante, qui prend en considération les synonymes.

Il est possible d'enrichir notre méthode pour faire de la recherche de patrons (pattern retreival), qui étant donné un problème renvoie la liste de tous les patrons qui le traitent.

Au terme de ce travail, nous estimons important de mentionner que des lacunes demeurent existantes dans le domaines des patrons logiciels, qu'il est intéressant d'adresser dans des travaux de recherche :

- **Formalisation insuffisante des patrons :** Les structures des patrons diffèrent d'un formalisme à un autre bien que très proches. Les formalismes existants ne font pas l'objet d'un consensus, et chaque formalisme définit des concepts différents. Cette diversité nuit d'une certaine manière au partage et à l'échange de patrons. Aussi, cela ne permet pas une application directe (éventuellement automatique) des patrons, ce qui est un frein évident à la réutilisation des solutions offertes par ces derniers.

- **Réutilisation manuelle de patrons :** Il y en a encore très peu de travaux sur l'aspect méthodologique de la réutilisation de patrons, pour pouvoir appliquer ces derniers de façon automatique ou tout au moins assistée. Aussi, la finalité des patrons est de pouvoir être utilisés non seulement pour des fins de description mais aussi de production. Donc pour les rendre productifs, la réutilisation des patrons devra être enrichie par l'exécutabilité de ces derniers (notamment pour les patrons de procédés)

et par la possibilité de modification de la solution pendant l'exécution (les solutions ont souvent tendance à évoluer pendant leurs exécutions). Malheureusement, de manière générale ce besoin critique d'éxécutabilité n'est pas encore satisfait.

- **Sous exploitation des patrons dans l'amélioration de solutions existantes** : Ce type d'application (amélioration de solutions existantes) est encore négligé. Les différentes connaissances capturées dans les patrons, doivent servir également pour restructurer ou enrichir des solutions déjà en application dans le but de les améliorer.

- **Manque de bases riches de patrons :** Pour appliquer efficacement les patrons dans des projets réels, on a besoin d'accéder à des patrons de différents niveaux d'abstraction et dans différents domaines d'application. Une typologie des problèmes et des contextes est nécessaire pour aider à organiser les patrons en bases de patrons logiciels. Il est également nécessaire de développer des outils de recherche et de sélection de patrons.

- **Manque de feedback de l'industrie :** Comme la majorité des travaux sur les patrons logiciels sont restés confinés dans les laboratoires de recherche, et très peu ont été appliqué réellement sur le terrain, il y a eu donc peu de retour d'expérience à partir du monde industriel, et du fait les patrons logiciels ont été privés de l'évolution qu'ils méritent.

Bibliographie

[AARS08] Agile Adoption Rate Survey ; 2008. Available at http://www.ambysoft.com/surveys/agileFebruary2008.html

[Aguiar09] A. Aguiar ; Tutorial on Agile Documentation with Wikis; WikiSym09, Florida, U.S.A ; 2009.

[Alast03] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P.Barros ; Workflow Patterns ; Distributed and Parallel Databases, 14(3), pages 5-51 ; July 2003.

[Alexander77] C. Alexander ; A pattern language: towns, buildings, constructions ; Oxford university press ; 1977.

[Alexander79] C. Alexander ; The Timeless Way of Building ; Oxford University Press ; 1979.

[Ambler02] S. Ambler ; All the Right Questions ; Software Development ; December 2002.

[Ambler02b] S.W. Ambler ; Agile Modeling – Effective Practices for eXtreme Programming and the Unified Process; John Wiley & Sons ; 2002.

[Ambler09] S. Ambler ; Agile/Lean Documentation : Strategies for Agile Software Development ; Ambysoft Inc. ; 2009. Available at http://www.agilemodeling.com/essays/agileDocumentation.htm

[Ambler98] S.W. Ambler ; Process Patterns: Building Large-Scale Systems Using Object Technology; New York: SIGS Books Cambridge University Press; 1998.

[Ambler98a] S.W. Ambler ; An Introduction to Process Patterns ; AmbySoft Inc. White Paper; 1998.

[Ambler07] S.W. Ambler ; Agile Model Driven Development (AMDD); Xootic magazine ; February 2007.

[Ambler99] S.W. Ambler ; More Process Patterns: Delivering Large-Scale Systems Using Object Technology ; New York: SIGS Books Cambridge University Press; 1999.

[Appleton97] B. Appleton ; Patterns for Conducting Process Improvement ; PLoP-97; 1997.

[Armour04] P.G. Armour ; The Laws of Software Process-A New Model for the Production and Management of Software; CRC Press LLC ; 2004

[Bartell09] A.L. Bartell et K.A Brown ; Secrets to managing a large documentation project virtually—process, technology, and group ethos: Lessons learned ; IEEE International Professional Communication Conference, IPCC 2009; 2009.

[Bass03] L. Bass, P. Clements and R. Kazman ; Software Architecture in Practice; Second Edition. AddisonWesley ; 2003.

[Ben09] Y. Ben-Chaim, E. Farchi, O. Raz ; An effective method for keeping design artifacts up-to-date ; IEEE CONFERENCES ICSE Workshop on Wikis for Software Engineering WIKIS4SE '09; 2009.

[Beck87] K. Beck ; Using Pattern Languages for Object-Oriented Programs ; Submitted to the OOPSLA-87 workshop on the Specification and Design for Object-Oriented Programming ; 1987.

[Beck97] K. Beck ; Smalltalk Best Practice Patterns ; Prentice-Hall ; 1997.

[Beedle97] A. Michael, B. OherentBPR ; A pattern language to build agile organizations ; PLoP-97 ; 1997.

[Bergner98] K. Bergner, A. Rausc et M. Sihling ; A Component Methodology based on Process Patterns; In Proceedings of the Annual Conference on the Pattern Languages of Programs PLOP'98, Monticello, Illinois ; 1998.

[BPEL07] Web Services Business Process Execution Language Version 2.0 ; Apr. 2007. Available at : http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[BPMN10] Business Process Modeling Notation (BPMN) Version 1.2 ; OMG ; 2010. Available at : http://www.omg.org/spec/BPMN/1.2

[Berry07] T. Berry, A. Gentle ; Writing End-User Documentation in an Agile Development Environment ; CIDM Best Practices Newsletter ; June 2007.

[Boehm03] B. Boehm et R. Turner ; Balancing Agility and Discipline: A Guide for the Perplexed ; USA: Addison-Wesley Publishers ; 2003.

[Buschmann03] F. Buschmann et K. Henney ; Beyond the Gang-of-Four ; Tutorial at the ACM OOPSLA 2003 Conference, Anaheim, CA ; Octobre 2003.

[Buschmann07] F. Buschmann, K. Henney et D.C. Schmidt ; PATTERN ORIENTED SOFTWARE ARCHITECTURE On Patterns and Pattern Languages Volume 5 , John Wiley & Sons Ltd ; 2007.

[Card87] N. D. Card, E. F. McGarry, and G. T. Page ; Evaluating software engineering technologies ; IEEE Transactions on Software Engineering, Vol. Se-13,No. 7; 1987.

[Carter09] L.R. Carter, A. Karatsolis ; Lessons from trying to develop a robust documentation exemplar; SIGDOC '09: Proceedings of the 27th ACM international conference on Design of communication ; Octobre 2009.

[Charvat03] J. Charvat ; Project Management Methodologies—Selecting, Implementing, and Supporting Methodologies and Processes for Projects ; John Wiley & Sons, Inc. 2003.

[Cook94] C. Cook ; DOCUMENTATION IS IMPORTANT ; Marcello Visconti Nov1994 crosstalk ;1994.

[Coleman98] G. Coleman et R. Verbruggen ; A Quality Software Process for Rapid Application Development ; Software Quality Journal, 7(2), pp. 107-122 ; 1998.

[Correia09] F. Correia, H. Ferreira, N. Flores ; Patterns for Consistent Software Documentation; Ademar Aguiar Pattern Languages of Programs Conference 2009 Chicago, USA; 2009.

[Cockburn01] A. Cockburn ; Agile Software Development ; Addison-Wesley ; 2001.

[Cockburn05] A. Cockburn ; Crystal Clear: A Human-Powered Methodology for Small Teams ; Addison-Wesley ; 2005.

[Connolly09] D. Connolly, F. Keenan, F. McCaffery ; Developing acceptance tests from existing documentation using annotations: An experiment ; IEEE CONFERENCES ICSE Workshop on Automation of Software Test, AST '09 ; 2009.

[Crégut97] X. Crégut, B. Coulette ; PBOOL: an Object-Oriented Language for Definition and Reuse of Enactable Processes ; Revue internationale Software Concepts and Tools, volume 18, numéro 2 ; Novembre 1997.

[Conte01] A. Conte, M. Fredj, JP. Giraudin et D. Rieu; P-Sigma : un formalisme pour une représentation unifiée de patrons ; Inforsid, Genève ; 2001.

[Coplien94] J.O. Coplien ; A Development Process Generative Pattern Language ; In Proceedings of the Annual Conference on the Pattern Languages of Programs PLoP'94; 1994.

[Coplien96] Coplien ; Software Patterns ; SIGS Book & Multimedia ; 1996.

[Coulette00] B. Coulette, X. Crégut, T. B. T. Dong et D. T Tran ; RHODES, a Process Component Centered Software Engineering Environment ; ICEIS2000; 2000.

[Crumlish09] C. Crumlish, E. Malone ; Designing Social Interfaces Principles, Patterns, & Practices for Improving the User Experience ; O'Reilly Media ; 2009.

[Demeyer03] S. Demeyer, S. Ducasse, O. Nierstrasz ; Object-Oriented Reengineering Patterns ; by Elsevier Science (USA); 2003.

[Deneckere01a] R. Deneckere, C. Souveyet; Organising and selecting patterns in pattern languages with Process Maps; OOIS'01 Canada; 2001.

[Deneckere01b] R. Deneckere ; Approche d'extension de méthodes fondée sur l'utilisation de composants génériques ; PhD thesis, University of Paris1-Sorbonne ; 2001.

[Dittmann02] T. Dittmann, V. Gruhn, M. Hagen ; Improved Support for the definition and usage of process patterns ; In SDPP02 ; 2002.

[Diaz09] O. Díaz, I. F. Anfurrutia, J. Kortabitarte ; Using DITA for documenting software product lines ; DocEng '09: Proceedings of the 9th ACM symposium on Document engineering ; Septembre 2009.

[Douglass02] B. Douglass ; Design patterns for real-time systems : Resource patterns ; http://www.informit.com/articles/article.aspx?p=30188 ; 2002.

[Douglass02a] B. Douglass ; Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems ; Addison-Wesley Professional ; 2002.

[Dzidek08] W.J. Dzidek, E. Arisholm, L.C. Briand ; A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance ; Software Engineering, IEEE Transactions on Volume: 34 , Issue: 3 ; 2008.

[Dyson97] P. Dyson ; Patterns for Abstract Design ; PhD thesis, University of Essex ; 1997.

[EPF06] Eclipse Process Framework; available at http://www.eclipse.org/epf/.

[Estublier05] J. ESTUBLIER ; Software are Processes Too ; In Invited Paper at Software Process Workshop, China ; 2005.

[FDD] Feature Driven Development; Nebulon Pty. Ltd. ; available at : http://www.featuredrivendevelopment.com

[Fernandez07] E. Fernandez, J. Pelaez, M. Larrondo-Petrie ; Attack Patterns: A New Forensic and Design Tool ; Advances in Digital Forensics III, International Federation for Information Processing, Volume 242/2007, pp. 345-357 ; eds. P. Craiger and S. Shenoi (Boston: Springer); 2007.

[Fowler02] M. Fowler ; Patterns of Enterprise Application Architecture ; Addison Wesley ; 2002.

[Forward02] A. Forward, T.C. Lethbridge ; The Relevance of Software Documentation, Tools and Technologies : A Survey ; ACM symposium in document engineering; 2002.

[FUIDP] Fluid Open Source Design Pattern Library - Open Source UI Design Patterns ; available at : http://www.uidesignpatterns.org/designPatterns

[Garceau90] L. Garceau, E. Jancura ; Documentation and Training as a Systems Development Tool ; The CPA Journal, 60(11), pp. 84-89 ; 1990.

[Gnatz02] M. Gnatz, F. Marschall, G. Popp , A. Rausch, W. Schwerin ; Common Meta-Model for a Living Software Development Processes; The 1st Workshop on Software Development Process Patterns (SDPP'02); 2002.

[Gnatz03] M. Gnatz, F. Marschall, G. Popp , A. Rausch, W. Schwerin; The Living Software Development Process; Journal Software Quality Professional, Volume 5, Issue 3 ; Join 2003.

[GoF95] E. Gamma, R. Helm, R. Johnson, J. Vlissides ; Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley ; 1995.

[Gotel09] G. Olly; M. Stephen ; More than Just "Lost in Translation"; Software, IEEE Volume: 26 , Issue: 2 ; 2009.

[Grosjean07] J. C. GROSJEAN ; Quality street Méthodes AGILES: une belle définition Disponible sur: http://www.qualitystreet.fr/2007/11/20/methodes-agiles-un-belle-definition/ ; 2007.

[Grosjean07b] J. C. GROSJEAN ; Quality street Documentation Agile : Juste ce qu'il faut. Disponible sur : http://www.qualitystreet.fr/2007/08/28/documentation-agile-juste-ce-qu-il-faut/ ; 2007.

[Gzara00] L. Gzara ; Les patterns pour l'ingénierie des Systèmes d'Information Produit; Doctorat de l'INPG, spécialité Génie Industriel ; 2000.

[Hagen02] M. Hagen ; Support for the definition and usage of process patterns; Position Paper, Focus Group "What makes Pattern Languages work well?", EuroPlop02 ; 2002.

[Hagen04a] M. Hagen, V. Gruhn ; Process Patterns - a Means to Describe Processes in a Flexible Way; International Workshop on Software Process Simulation and Modeling ProSim'04, Edinburgh, United Kingdom; 2004.

[Hagen04b] M. Hagen, V. Gruhn ; Towards flexible Software Processes by using Process Patterns; 8th IASTED International Conference on Software Engineering and Applications (SEA2004), Cambridge, USA ; 2004.

[Hagen04c] M. Hagen, V. Gruhn; Process Patterns - Flexible Modelling of Software Processes; available at http://ebus.informatik.unileipzig.de/papers/paperuploads; 2004.

[Henney99] K. Henney; Patterns Inside Out ; Talk presented at Application Development, London ; 1999.

[Henninger07] S. Henninger, V. Corrêa ; Software Pattern Communities: Current Practices and Challenges ; University of Nebraska–Lincoln, Computer Science and Engineering Technical Report ; 2007.

[Heinrich09] M. Heinrich, A. Boehm-Peters, M. Knechtel ; A platform to automatically generate and incorporate documents into an ontology-based content repository ; DocEng '09: Proceedings of the 9th ACM symposium on Document engineering ACM ; 2009.

[Highsmith99] J. A. Highsmith ; Adaptive Software Development : A Collaborative Approach to Managing Complex Systems ; Dorset House ; December 1999.

[Highsmith04] J. Highsmith ; Agile Project Management ; Creating innovative products: Addison-Wesley ; 2004.

[Highsmith02] J. Highsmith ; Agile Software Development Ecosystems ; Addison-Wesley, 2002.

[Holz03] H. Holz, F. Maurer ; Knowledge Management Support for Distributed Agile Software Processes ; Chicago, USA ; 2003.
[Holdaway09] J. Holdaway ; Technical reviews: Framing the best of the best practices; IEEE International Professional Communication Conference, IPCC 2009 ; 2009.

[Hyland08] D. Hyland-Wood, D. Carrington, S. Kaplan ; Towards a software maintenance methodology using Semantic Web techniques and paradigmatic documentation modelling; IEEE Software, IET Volume: 2 , Issue: 4 ; 2008.

[IEEE04] IEEE Std 1558-2004 ; IEEE Standard for Software Documentation for Rail Equipment and Systems.

[IEEE09a] IEEE Draft Standard: Systems and software engineering - Requirements for acquirers and suppliers of user documentation ; Dec 2009.

[IEEE01] IEEE Std 1063-2001; IEEE Standard for Software User Documentation.

[IEEE08] IEEE Std 829-2008; IEEE Standard for Software and System Test Documentation.

[IEEE09b] Draft Standard Software and systems engineering -- Content of life-cycle information products (documentation) ; Dec 2009.

[Iida99] H. Iida et al. ; Pattern-Oriented Approach to Software Process Evolution; proceedings of International Workshop on the Principles of Software Evolution IWPSE; Fukuoka Software Research Park, Fukuoka City, Japan ; Juillet 1999.

[ISO04] ISO/IEC 18019:2004 ; Ingénierie du logiciel et du système -- Lignes directrices pour la conception et la préparation de la documentation de l'utilisateur de logiciels d'application.

[ISO05] ISO/IEC TR 9294:2005 ; Technologies de l'information -- Lignes directrices pour la gestion de la documentation technique du logiciel.

[ISO09a] ISO/IEC 26513:2009 ; Ingénierie des systèmes et du logiciel -- Exigences pour testeurs et vérificateurs de documentation utilisateur.

[ISO09b] ISO/IEC WD 26511 Current stage date: 2009-05-23 ; Ingénierie du logiciel et des systèmes -- Exigences pour les managers de la documentation de l'utilisateur.

[ISO09c] ISO/IEC CD 15289 Current stage date: 2009-12-18 ; Ingénierie des systèmes et du logiciel -- Contenu des systèmes et produits d'information du processus de cycle de vie du logiciel (documentation).

[ISO08] ISO/IEC 26514:2008 ; Ingénierie du logiciel et des systèmes — Exigences pour les concepteurs et les développeurs de la documentation de l'utilisateur.

[ISO10] ISO/IEC FCD 26512 Current stage date: 2010-02-15; Ingénierie du logiciel et des systèmes -- Exigences pour les acheteurs et les fournisseurs de la documentation de l'utilisateur.

[ITM08] L. R. Vijayasarathy, D. Turk ; Agile software development : a survey of early adopters ; Journal of Information Technology Management Volume XIX, Number 2, 2008.

[ITUnion09] State of the IT Union Survey ; Juillet 2009. Available at : http://www.ambysoft.com/surveys/stateOfITUnion200907.html

[Karlesky08] M. Karlesky, M.V. Voord ; Agile Project Management (or, Burning Your Gantt Charts) Embedded Systems Conference Boston (Boston, Massachusetts) ESC 247-267; Octobre 2008.

[Kaul97] U. Kauls; Taxonomy for organisational patterns ; Désponible sur http://www.bell-labs.com/cgi-user/OrgPatterns/OrgPatterns?UrvashiKaulsTaxonomy; 1997.

[Kokkoniemi09] J.K. Kokkoniemi, L. Harjumaa ; From Software Documents to Experience Knowledge Based Artifacts; HICSS '09. 42nd Hawaii International Conference on System Sciences, IEEE CONFERENCES ; 2009.

[Komulainen06] T. Kynkäänniemi, K. Komulainen ; Agile documentation in Mobile-D projects ; Information Technology for European Advancement ; 2006.

[Kruchten04] P. Kruchten ; An ontology of architectural design decisions in software intensive systems ; In 2nd Groningen Workshop on Software Variability, pages 54-61 ; Décembre 2004.

[Kubo05] A. Kubo, H. Washizaki, A. Takasu, Y. Fukazawa ; Analyzing Relations among Software Patterns based on Document Similarity; Proc. IEEE Internationale Conference on Information Technology : Coding and Computing (ITCC 2005) ; 2005.

[Kubo05b] A. Kubo, H. Washizaki, A. Takasu, Y. Fukazawa; EXTRACTING RELATIONS AMONG EMBEDDED SOFTWARE DESIGN PATTERNS; Transactions of the Society for Design and Process Science, Journal of integrated design and process science vol9 no3 pp 39-52; 2005.

[Kylmäkoski03] R. Kylmäkoski ; Ed. Efficient Authoring of Software Documentation Using RaPiD7 ; Proceedings of the 25th International Conference on Software Engineering. Portland, Oregon, IEEE Computer Society ; 2003.

[Lammi] J. Lammi, M. Varjokallio, J. Hocksell ; A user interface design pattern library; http://www.patternry.com.

[Lethbridge03] T.C Lethbridge, J. Singer, et al ; How Software Engineers Use Documentation: The State of the Practice ; IEEE Software**:** 35-39. ; 2003.

[Lientz81] Lientz, P. Bennet, Swanson, E. Burton ; Problems in applications software maintenance ; Communications of the ACM; Novembre 1981.

[Maalej09] W. Maalej, H.J Happel ; From work to word: How do software developers describe their work? ; MSR '09. 6th IEEE International Working Conference on Mining Software Repositories ; 2009.

[Manifeste01] Agile Manifesto ; available at http://agilemanifesto.org/ ;2001.

[McMillan09] C. McMillan, D. Poshyvanyk, M. Revelle ; Combining textual and structural analysis of software artifacts for traceability link recovery ; TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering Publisher: IEEE Computer Society; Mai 2009.

[Meszaros97] G. Meszaros, J. Doble ; A Pattern Language for Pattern Writing, PLoP ; 1997.

[Montangero99] C. Montangero, J.C Derniame, B.A. Kaba, B. Warboys ; The software process: Modelling and technology ; LNCS GmbH. Vol. 1500 ; 1999.

[Nasution09] M.F.F. Nasution, H.R Weistroffer ; Documentation in Systems Development: A Significant Criterion for Project Success; IEEE CONFERENCES HICSS '09. 42nd Hawaii International Conference on System Sciences ; 2009.

[Nerur07] S. Nerur, V. Balijepally ; Theoretical reflections on agile development methodologies ; Commun. ACM, 50(3):79– 83 ; 2007.

[Noble97] J. Noble ; Arguments and results ; In PLOP Proceedings ; 1997.

[Noble98] J. Noble; Classifying Relationships Between Object-Oriented Design Patterns; Australian Software Engineering Conference, Adelaide, South Australia, November 09-13; 1998.

[Nord09] R.L Nord, P.C Clements, D. Emery, R. Hilliard ; Reviewing architecture documents using question sets; WICSA/ECSA'09 Joint Working IEEE/IFIP Conference on Software Architecture,& European Conference on Software Architecture ; 2009.

[Oldani06] G. Oldani ; Agilité ; Disponible sur le site: http://social.hortis.ch/wp-content/uploads/2006/10/Agilit%C3%A9_synth%C3%A8se_v2.doc ; 2006.

[O'Neill00] C. O'Neill ; The Lancaster Stemmer ; 2001. Available at : http://www.comp.lancs.ac.uk/computing/research/stemming/paice/paicejava.htm

[Osterweil87] L.J. OSTERWEIL ; Software Processes Are Software Too ; In Proceedings of the 9th International Conference on Software Engineering (ICSE), pages 2–13, Monterey, California, USA; Mars 1987.

[Paice90] C.D. Paice; Another Stemmer ; SIGIR Forum, vol24, no 3, pp56-61; 1990.

[Pikkarainen05] M. Pikkarainen, U. Passoja ; An Approach for Assessing Suitability of Agile Solutions: A Case Study ; 6th International conference of eXtreme Programming and agile process in software engineering, Sheffield University, UK ; 18-23 Join 2005.

[Prabhakar10] T.V Prabhakar, K. Kumar ; Design Decision Topology Model for Pattern Relationship Analysis; Asian PLOP 2010 Tokyo ; 16-17 Mars 2010.

[Pryce97] N. Pryce ; Type-safe session ; In EuroPLOP Proceedings ; 1997.

[RAD] Jean-Pierre Vickoff ; Site historique de la méthode RAD ; available at www.rad.fr

[Ramos08] R.A Ramos, J. Castro, J. Araujo, A.M Diniz Moreira, F. Alencar, R. Penteado; IDEAS01: Refactoring to Requirements Documents: An approach Aspect Oriented; Latin America Transactions, IEEE (Revista IEEE America Latina) Volume: 6 , Issue: 3 ; 2008.

[Ratanotayanon09] S. Ratanotayanon, S. Elliott Sim, D.J. Raycraft ; Cross-artifact traceability using lightweight links ; TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering Publisher: IEEE Computer Society ; Mai 2009.

[Reiss06] S.P. Reiss ; Incremental Maintenance of Software Artifacts ; Proceedings of the 21st IEEE International Conference on Software Maintenance ; 2006.

[Ribo00] J.M Ribó, X. Franch ; PROMENADE, a PML intended to enhance standardization, expressiveness and modularity in SPM ; Research Report LSI-00-34-R, Dept. LSI, Politechnical University of Catalonia; 2000.

[Ribo02] J.M Ribó, X. Franch ; Supporting Process Reuse in PROMENADE; Research Report LSI-02-14-R, Dept, LSI, Politechnical University of Catalonia; 2002.

[Rieu99] D. Rieu, J.P. Giraudin, S. Marcel, C. Front-Conte ; Des opérations et des relations pour les patrons de conception ; Inforsid'99 ; 1999.

[Rising03] L. Rising ; Foreword of A Ruping's book (Agile Documentation, A Pattern Guide to Producing Lightweight Documents for Software Projects) ; West Sussex, Jonh Wiley & Sons Ltd ; 2003.

[Rombach87] H.D Rombach, V.R Basili ; Quantitative assessment of maintenance: an industrial case study ; IEEE Proceedings of Conference on Software Maintenance, City Press; 1987.

[Rota07] V. M. Rota ; Gestion de projet : Vers les méthodes agiles ; EYROLLS ; 11/2007.

[Ruping03] A. Ruping ; Agile Documentation, A Pattern Guide to Producing Lightweight Documents for Software Projects ; West Sussex, Jonh Wiley & Sons Ltd ; 2003.

[Ruping98a] A. Rüping, J. Coldewey, P. Dyson ; The Structure and Layout of Technical Documents ; In Proceedings of the 3rd European Conference on Pattern Languages of Programming and Computing 1998, Universitätsverlag Konstanz ; 1998.

[Ruping98b] A. Rüping, J. Coldewey, P. Dyson ; Writing and Reviewing Technical Documents', in Proceedings of the 3rd European Conference on Pattern Languages of Programming and Computing ,Universitätsverlag Konstanz, 1998.

[Ruping99a] A. Rüping, P. Dyson, M. Devos ; Project Documentation Management ; In Proceedings of the 4th European Conference on Pattern Languages of Programming and Computing , Universitätsverlag Konstanz, 1999.

[Ruping99b] A. Rüping, P. Dyson, M. Devos ; Typography and Desktop Publishing; In Proceedings of the 4th European Conference on Pattern Languages of Programming and Computing, Universitätsverlag Konstanz; 1999.

[Salton83] G. Salton, M.J. McGill ; Introduction to Modern Information Retrivial ; McGraw-Hill Inc ; 1983.

[Salton73] G. Salton, C.S. Yang ; On The Specification of Term Values in Automatique Indexing ; Journal of Documentation, vol. 29, pp 351-372; 1973.

[SaltonSW] Gerard Salton and Chris Buckley; Stopword list ; available at : http://www.lextek.com/manuals/onix/stopwords2.html

[Sanchez09] I. Sánchez-Rosado, P. Rodríguez-Soria, B. Martín-Herrera, J.J. Cuadrado-Gallego, J. Martínez-Herráiz and A. González ; Assessing the Documentation Development Effort in Software Projects ; Lecture Notes in Computer Science, Software Process and Product Measurement ; Springer; 2009.

[Shine03] Shine Technologies ; Agile Methodologies Survey Results ; Disponible sur le site webe: http://www.shinetech.com/download/attachments/98/ShineTechAgileSurvey2003-01-17.pdf ; 2003.

[Snow06] K. Snow, M. Marks, D. Hong, T. Dennis ; Web Patterns Project ; Disponible sur le sitehttp://harbinger.sims.berkeley.edu/ui_designpatterns/webpatterns2/webpatterns/home.php, U.C. Berkeley School of Information ; 2006.

[Sommerville01] I. Sommerville ; Software Documentation ; Lancaster University ; 2001.

[Stal96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, M. Stal; Pattern-oriented software architecture - A System of Patterns, Volume 1; Wiley and Sons ; 1996.

[Staudt10] B. Staudt Lerner, S. Christov, L.J. Osterweil, R. Bendraou, U. Kannengiesser, et A. Wise ; Exception Handling Patterns for Process Modeling ; IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 36 ;2010.

[Storrle01] H. Storrle ; Describing Process Patterns with UML ; In Proceedings of the European Workshop in Software Process Technology EWSPT'01, Germany; 2001.

[Storrle03] H. Storrle ; Process Modeling and Simulation; ICSE-03, USA; 2003.

[Tidwell05] J. Tidwell ; Designing Interfaces - Patterns for Effective Interaction Design, O'Reilly Media ; 2005.

[Tilley09] S.Tilley ; Documenting software systems with views VI: lessons learned from 15 years of research & practice ; SIGDOC '09: Proceedings of the 27th ACM international conference on Design of communication ; Octobre 2009.

[Torchiano10] M. Torchiano, F. Ricca, P. Tonella ; Empirical comparison of graphical and annotation-based re-documentation approaches; Software, IET Volume: 4 , Issue: 1 ; 2010.

[Tran01] D.T Tran ; Formalisation et mise en oeuvre de la notion de composant de procédés logiciels ; Thèse doctorale, N°1818, Institut National Polytechnique de Toulouse ; 2001.

[TRAN07] H.N. TRAN ; Modélisation de procédés logiciels à base de patrons réutilisables; Thèse doctorale, l'Université Toulouse II- Le Mirail ; 2007.

[Turk02] D. Turk, R. France, et B. Rumpe ; Limitations of Agile Software Processes ; Proceedings of the 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering ; Mai 2002.

[Tyrrell00] S. Tyrrell; ACM Crossroads; The Many Dimensions of the Software Process; available at : http://www.acm.org/crossroads/xrds6-4/ ; 2000.

[Vickoff07] J.P.Vickoff ; AGILE l'Entreprise et ses Projets ; QI Editions ; 2007.

[Volter00] M. Volter ; Server-side components - a pattern language ; In proceedings of EuroPLoP'2000 ; 2000.

[Vora09] P. Vora ; Web application design patterns ; Morgan Kaufmann, Elsevier ;2009.

[Washizaki05] H. Washizaki, A. Kubo, A. Takasu et Y. Fukazawa ; Relation analysis among patterns of software development process ; PROFES 2005, LNCS 3547, pp 299-313 ; 2005.

[Washizaki07] . Washizaki, A. Kubo, A. Takasu et Y. Fukazawa ; Extracting Relations among Security Patterns ; 1st International Workshop on Software Patterns and Quality SPAQu07; 2007.

[Wise06] A. Wise ; Little-JIL 1.5 Language Report ; Technical report ; Univ. of Massachusetts ; 2006.

[XmlPatterns00] Develop effective XML documents using structural design patterns ; 2000 ; available at : http://www.xmlpatterns.com/

[XP] Don Wells ; Extreme Programming : A gentle introduction ; Last modified September 2009 ; available at www.extremprogramming.org

[YDPL] Yahoo! Design Pattern Library ; available at : http://developer.yahoo.com/ypatterns/

[Yelland96] P.M. Yelland ; Creating host compliance in a portable framework : A study in the use of existing design patterns ; In OOPSLA Proceedings ;1996.

[Zdun09] U. Zdun ; Guest Editor's Introduction: Capturing Design Knowledge ; Software IEEE Volume : 26 , Issue:2 ; Mars-Avrile 2009.

[Zhang08] Y. Zhang, R. Witte, J. Rilling, V. Haarslev ; Ontological approach for the semantic recovery of traceability links between software artefacts ; IEEE Software, IET Volume: 2 , Issue: 3 ; 2008.

[Zimmer94] W. Zimmer ; Relationships between design patterns ; In Pattern Languages of Program Design, Addison-Wesley ;1994.