Titre: Title:	Améliorer la testabilité et les tests des systèmes à l'aide des patrons
Auteurs: Authors:	Aminata Sabane
Date:	2010
Type:	Rapport / Report
Référence: Citation:	Sabane, Aminata (2010). Améliorer la testabilité et les tests des systèmes à l'aide des patrons. Rapport technique. EPM-RT-2010-08.

EPM–RT–2010-08

AMÉLIORER LA TESTABILITÉ ET LES TESTS DES SYSTÈMES À L'AIDE DES PATRONS

Aminata Sabané
Département de Génie informatique et génie logiciel
École Polytechnique de Montréal

Septembre 2010

EPM-RT-2010-08

Améliorer la testabilité et les tests des systèmes à l'aide des patrons

Aminata Sabané
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

Septembre 2010

Dépôt légal :
Bibliothèque nationale du Québec, 2010
Bibliothèque nationale du Canada, 2010

EPM-RT-2010-08
Améliorer la testabilité et les tests des systèmes à l'aide des patrons
par : Aminata Sabané
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Améliorer la testabilité et les tests des systèmes à l'aide des patrons

par

Aminata SABANÉ

Une proposition de recherche en vue de
continuer le programme de Ph.D.

au

Département de génie informatique et génie logiciel

31 août 2010

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Résumé

Département de génie informatique et génie logiciel

Ph.D.

par Aminata SABANÉ

Les activités de test sont très complexes et coûteuses mais elles restent cependant le moyen le plus rependu pour assurer la fiabilité des systèmes et augmenter notre confiance en eux. De nombreux travaux ont pour objectif la réduction de ces coûts et de cette complexité à travers diverses approches : critères de test plus efficaces, automatisation des activités de tests, augmentation de la testabilité. Cette dernière approche est intensivement explorée car pour de nombreux chercheurs, la testabilité peut permettre de réduire signicativement les coûts de test surtout si elle est évaluée et exploitée tôt dans le cycle de développement du logiciel notamment à la phase de conception. C'est dans ce domaine très actif des conceptions testables que s'inscrit notre projet de recherche. Ce projet vise à contribuer à réduire les coûts de test et en augmenter l'efficacité en exploitant les microarchitectures, telles que les patrons de conception et les anti-patrons. Ces microarchitectures existent largement dans les systèmes orientés objets et sont reconnus comme ayant un impact sur plusieurs attributs de qualité. Notre objectif est donc, à travers ce projet de recherche et la méthodologie proposée dans le présent document, d'étudier l'impact des microarchitectures sur la testabilité des systèmes, d'exploiter les résultats de cette étude pour améliorer la testabilité des conceptions mais aussi exploiter directement les microarchitectures dans la phase de test.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Présentation du sujet

1.1 Contexte

Les logiciels sont aujourd'hui indispensables dans presque tous les domaines de notre société. Leur fiabilité est donc un réel enjeu. En effet, la défaillance des logiciels peut provoquer des pertes financières énormes mais également risquer des vies humaines dans les systèmes critiques tels que les systèmes avioniques, de santé.
Les tests sont le moyen le plus répandu et adopté pour essayer de garantir la fiabilité des logiciels. Ils ne permettent certes pas de certifier la fiabilité absolue ou l'absence de fautes dans un logiciel mais ils permettent d'augmenter notre confiance dans le logiciel et constitue un critère d'assurance qualité ou d'acceptabilité [1].
Les tests constituent donc une activité importante et essentielle dans le cycle de vie de tout logiciel. Ils interviennent à différentes étapes de ce cycle et font intervenir différents niveaux d'abstraction. Malheureusement, les activités de tests sont très complexes et coûteuse financièrement mais aussi en temps. En effet, la marge attribuée aux tests est évaluée à plus de 50% [1] du temps et du coût total du développement du logiciel. Il est donc nécessaire de trouver des approches et des techniques pour réduire cette complexité et ces coûts. Répondre à cette nécessité permettra également de réduire le coût global de développement du logiciel mais aussi de répondre au niveau de fiabilité requis par les systèmes aujourd'hui.
C'est dans ce cadre que s'inscrit notre projet de recherche qui vise à exploiter les microarchitectures, telles que les patrons de conception et les anti-patrons, pour contribuer à réduire les coûts de test et en augmenter l'efficacité.

1.2 Problématique

Les tests présentent énormément de défis. De nombreux chercheurs depuis des décennies tentent de trouver des approches afin d'en réduire la complexité, le coût mais aussi d'en augmenter l'efficacité en termes de taux de couverture ou de détection de fautes.
Plusieurs techniques de test, de niveaux de test, de critères de couvertures, de méthodes de génération de cas de test et aussi d'approches d'automatisation des activités de test ont été proposées ; chacune ayant un but spécifique mais toutes poursuivant le même objectif global : réduire l'effort et les coûts des tests et augmenter leur efficacité. Ainsi donc, on a des tests à différents niveaux d'abstraction : tests unitaires, tests d'intégration, tests systèmes ou encore tests de régression. Certaines techniques permettent de tester les fonctionnalités du système (test boîte noire) et d'autres l'implantation de celles-ci (test boîte blanche), et d'autres encore évaluent et analysent l'attribut de qualité qu'est la testabilité du système.
Le standard IEEE définit la testabilité comme le degré avec lequel un système ou un composant facilite l'établissement de critères de test et la performance des tests à déterminer si ces critères ont été couverts. Une autre définition présente la testabilité comme un facteur de qualité des systèmes logiciels ou de leurs architectures relatif à la facilité de tester une partie du logiciel [2]. Pour de nombreux chercheurs, la testabilité est un attribut du logiciel dont l'exploitation peut permettre d'améliorer significativement les tests et réduire les coûts qui y sont liés [3–5]. C'est donc un domaine qui fait l'objet de nombreuses études qui vont de l'évaluation de la testabilité des systèmes [4, 6, 7] à son amélioration [8–10], en passant par l'analyse des facteurs qui l'influencent [3, 5].
La testabilité d'un système peut être évaluée tant au niveau de la conception qu'au niveau du code. Pour de nombreux chercheurs [3, 5] l'information sur la testabilité revêt toute son importance en début du cycle de développement ; particulièrement à la phase de conception. En effet, durant la conception, cette information peut être utilisée pour améliorer la testabilité afin de rendre le logiciel avenir plus facile à tester et donc améliorer sa fiabilité. Le but est donc d'intégrer des moyens de réduction des coûts et de la complexité des tests tôt dans le cycle de développement des systèmes : "Other things being equal, a more testable system will reduce the time and cost needed to meet reliability and more generally, quality goals" [3].
Les conceptions orientés objets contiennent généralement des microarchitectures. Ces microarchitectures sont formées par des groupes de classes ayant une intention et une organisation spécifiques. Les patrons de conception forment avec les anti-patrons les microarchitectures les plus populaires dans les conceptions orientés objets ; ils y sont largement présents, intentionnellement ou non.
Les patrons de conception ont depuis leur introduction au milieu des années 90 suscité un grand engouement dans la recherche de conceptions de qualité. Ils sont définis comme

des solutions réutilisables aux problèmes de conception récurrents [11]. Ces microarchitectures font l'objet d'un grand nombre d'études en génie logiciel. Plusieurs chercheurs affirment qu'ils ont un impact positif sur plusieurs attributs de qualité tels que la maintenabilité, la réutilisabilité [11, 12]. Cependant d'autres études [13**?** , 14] viennent nuancer ces allégations et appellent à utiliser les patrons de conception avec justesse et prudence.
À l'opposé des patrons de conception, les anti-patrons sont définis comme des solutions pauvres aux problèmes de conception récurrents [15]. Ils sont largement décrits dans la littérature comme ayant un impact négatif sur la qualité des systèmes [15–17].
Les nombreuses études effectuées sur ces microarchitectures et leurs impacts sur les logiciels montrent leur importance et l'influence qu'elles ont sur les systèmes. Les conclusions de ces études qui parfois viennent contredire les idées préconçues montrent la nécessité de telles études. Elles sont la base de choix et de prises de décisions rationnelles afin d'améliorer la qualité des systèmes.
Malheureusement, il existe, à notre connaissance, peu d'études de ce genre sur la question de l'impact de ces microarchitectures sur les tests en général et sur la testabilité en particulier. Nous nous proposons donc dans le premier volet de notre travail de recherche de mettre en évidence l'impact, s'il existe, des microarchitectures sur la testabilité des logiciels. Cette étude servira de base pour suggérer des directives et mettre en place des mécanismes afin d'améliorer la testabilité des systèmes et contribuer à l'objectif de réduire les coûts et la complexité des tests mais aussi augmenter leur efficacité.
Si l'étude de la testabilité des microarchitectures révèle qu'elles ont un impact sur la testabilité des logiciels, nous nous proposons d'explorer les moyens de les exploiter directement pour les tests afin réduire les coûts de test et en augmenter l'efficacité. En effet, comme mentionné auparavant, le test est l'une des méthodes les plus utilisées pour améliorer la fiabilité des logiciels. Il vise à détecter les fautes du programme et également s'assurer du fonctionnement adéquat de celui ci. On ne peut malheureusement pas tester tous les cas possibles ni être sûr d'avoir détecté toutes les fautes. Cela conduit à demander ce qu'il faut tester et quand est ce qu'il faut arrêter les tests. Ces questions sont l'objet d'un grand nombre de travaux de recherche allant de la définition des critères de couverture, à la génération des cas de tests significatifs en passant par l'automatisation de cette génération. Les microarchitectures en particulier les patrons de conception véhiculent des intentions. Il est donc important de s'assurer que cette intention est préservée quand ils sont implémentés. De plus, la structure parfois complexe de certains patrons de conception peut être source de fautes. Ces considérations nous emmènent à vouloir étudier le gain que pourrait apporter les tests des patrons de conception. Ce second volet de notre projet de recherche nous permettra également d'explorer l'exploitation des patrons de conception dans la génération de cas de tests efficaces (permettent un grand taux de couverture ou la détection d'un grand nombre de fautes).

Remarque : Dans la suite de cette proposition, nous utiliserons généralement les patrons de conception pour illustrer nos questions/approches sans perte de généralité vis-à-vis des autres patrons. Nous choisissons les patrons de conception à cause de leur grande popularité dans les systèmes orientés objet.

1.3 Définition de concepts clés

Nous présentons dans cette section les définitions que nous adoptons pour quelques concepts clés dans le cadre de ce document.
Le coût global de test : ce facteur évalue l'effort de test nécessaire à la vérification d'un critère de test donné. Cette mesure est relative à la taille de l'ensemble de cas de test, à la difficulté de trouver des données de test pour vérifier le critère ainsi qu'à la difficulté d'évaluer la valeur d'oracle [4].
La testabilité : c'est un facteur de qualité, défini comme la facilité à tester un logiciel. Cette facilité est à la fois une propriété intrinsèque du schéma de conception (et donc une caractéristique propre au produit étudié) et une propriété relative à la stratégie de test adoptée pour vérifier un critère de test particulier. La testabilité se dérive en trois attributs :

- le coût global de test : le coût global pour obtenir des cas de test vérifiant un critère de test donné ;
- la contrôlabilité : la facilité pour générer des données de test efficaces (propriété intrinsèque au logiciel sous test) ;
- l'observabilité : la facilité avec laquelle il est possible de vérifier la pertinence des résultats obtenus après l'exécution des cas de test [4].

Les patrons de conception : offrent des solutions génériques à des problèmes récurrents de conception [11].
Les motifs de conception : solutions idéales qui décrivent les rôles joués par les classes qui implémentent les patrons de conception [18].
Les anti-patrons : Solutions à des problèmes récurrents qui entrainent des conséquences négatives [15].

1.4 Objectifs

L'objectif principal de notre travail de recherche est d'exploiter les microarchitectures, en particulier celles décrites par les patrons de conception et les anti-patrons, pour réduire les coûts de test mais aussi augmenter leur efficacité. Cet objectif s'inscrit dans celui plus globale de la recherche d'une meilleure fiabilité des systèmes.

Pour réduire les coûts de test, nous choisissons d'étudier l'impact des microarchitectures sur la testabilité du système et mettre en évidence les facteurs qui l'occasionnent. Cela nous permettra ainsi d'élaborer des recommandations et des techniques qui permettront de réduire au besoin cet impact et d'avoir des conceptions plus testables.
Si cette première étude révèle un impact des microarchitectures sur la testabilité des systèmes, nous nous proposons de les exploiter pour augmenter l'efficacité des tests. Nous explorerons dans cette phase les moyens de (1) réduire les cas de test nécessaires pour tester un système selon un critère de couverture à l'aide des patrons de conception, (2) de garantir la préservation de l'intention des patrons de conception par des tests, (3) et également de mettre à disposition des cas de tests génériques préconçus pour les patrons de conception.
Les résultats de l'étude de ce projet de recherche devraient être utiles aux concepteurs, aux gestionnaires de projet, aux développeurs et aux testeurs. En effet, les résultats de l'étude de l'impact des microarchitectures sur la testabilité des systèmes de la testabilité permettra de guider les concepteurs dans leurs choix de conception en prenant en compte les répercussions sur les tests. Ils pourront ainsi sur cette base décider d'utiliser ou non un patron de conception, quelle variante de ce patron préférer, quelles interactions éviter avec ce patron, quelles compositions privilégier. Ces résultats permettront également aux gestionnaires de projet de mieux planifier et organiser les efforts de tests. Les développeurs pourront justifier sur des bases rationnelles leurs refactorings et bénéficier également de recommandations et de techniques pour les y aider. Enfin, les résultats du second volet pourront faciliter la tâche aux testeurs dans la génération de cas de tests et dans l'atteinte des objectifs de tests en termes de couverture et de fiabilité.

1.5 Illustration de l'impact d'un patron de conception sur la testabilité du système

Le patron de conception Visiteur appartient à la catégorie des patrons de conception comportementaux définis par [11]. Ce patron vise à séparer un algorithme d'une structure de données. Le Visiteur permet d'ajouter une nouvelle opération sans modifier les classes des objets sur lesquels s'appliquent cette opération [11]. Il permet ainsi d'étendre fonctionnellement une hiérarchie de données sans modifier celle-ci. La structure de ce patron est décrite par la figure 1.1.
Le diagramme de classes de la figure (cf. Figure 1.2 [1]) décrit une structure de données représentant les composants d'un programme dans un système de compilation. Chaque

1. http ://pcaboche.developpez.com/article/design-patterns/programmation-modulaire/

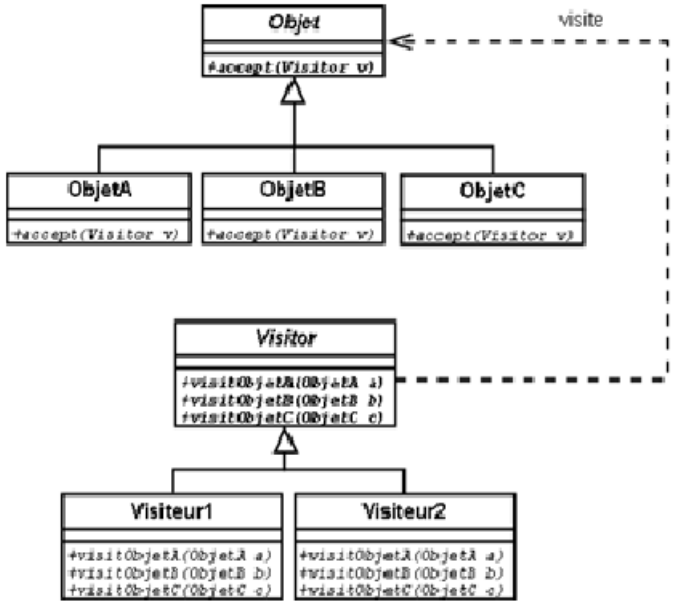

FIGURE 1.1 – Structure du patron de conception Visiteur

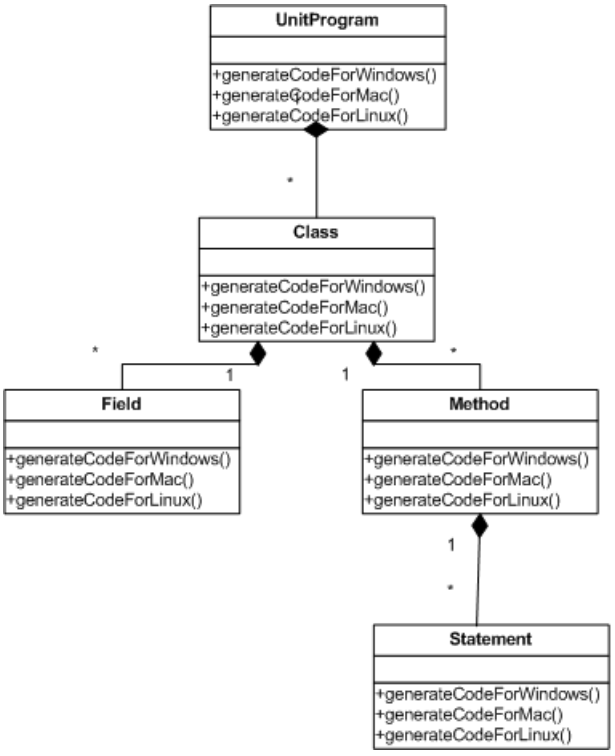

FIGURE 1.2 – Diagramme de classes du système 1

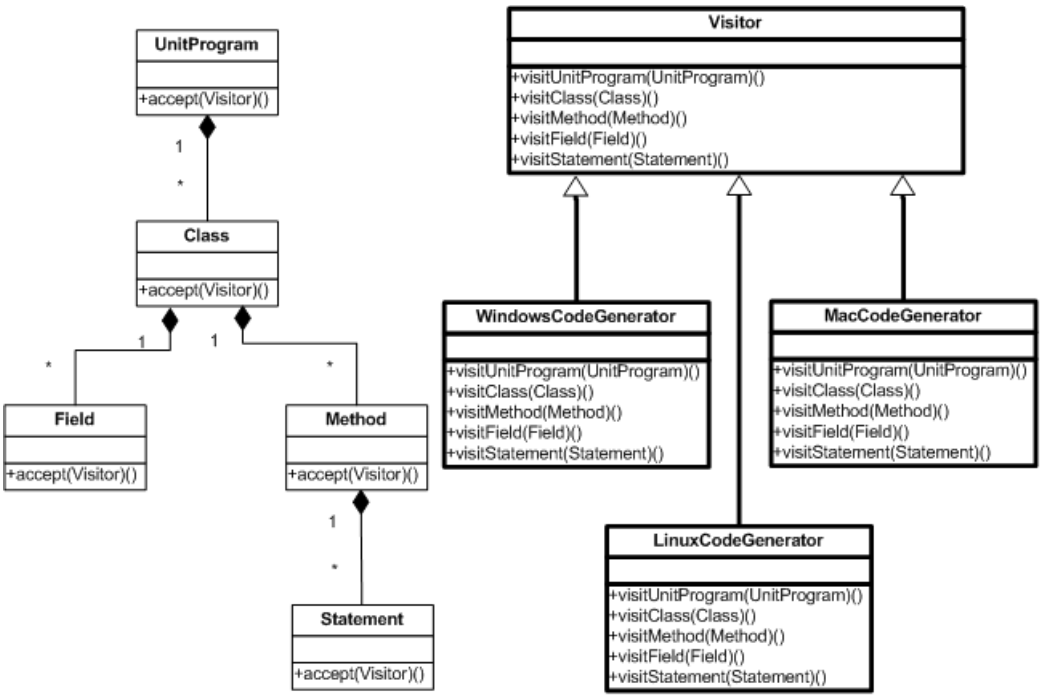

FIGURE 1.3 – Diagramme de classes du système 2

nœud de cette structure effectue un certain nombre d'opérations parmi lesquelles l'opération de génération de code. Mais chaque nœud effectue l'opération de façon spécifique. De plus, cette opération est faite selon le système d'exploitation sur lequel s'exécute le programme. Ainsi, on pourrait avoir des opérations telles que *generateCodeForWindows*, *generateCodeForLinux*, *generateCodeForMac* qui doivent être exécutées par chacun des objets de la structure de données. Le système 1 (cf. Figure 1.2) conçoit cette spécification en mettant chacune de ces opération dans chacune des classes de la structure de données. Cette conception rend difficile l'évolution ou la maintenance du système car l'ajout ou la modification d'une opération entraîne le changement de toutes les classes de la structure. De plus, le code sera plus difficile à comprendre ainsi.

Le diagramme de classes de la figure 1.3 représente le même système mais conçu cette fois avec le patron de conception Visiteur. Ce patron sépare les opérations de la structure de données. Pour cela, on crée une classe Visiteur par opération. Cette classe implémente cette opération pour chaque nœud de la structure de données. Les nœuds sont donc allégés des différentes opérations et n'implémentent que la méthode *accept(Visitor)*. Ainsi, lorsqu'on veut ajouter une nouvelle opération, il suffit d'ajouter au diagramme une nouvelle classe Visiteur sans modifier les classes de la structure de données.

Le patron de conception Visiteur facilite ainsi l'extensibilité et la maintenabilité du système et le rend plus compréhensible et lisible. Cependant, le système 2 semble plus complexe pour les tests d'intégration parce que le patron de conception Visiteur utilise

l'héritage et introduit un fort polymorphisme dans le système. L'héritage et le polymorphisme sont deux caractéristiques de l'orienté objet qui complexifient les tests d'intégration en conduisant à de nombreux cas de tests possibles entrainant ainsi des coûts de test élevés.
Ainsi donc, le patron de conception Visiteur pourrait impacter négativement la testabilité du système.

1.6 Questions de recherche

Comme mentionné dans la section 1.4, nous nous proposons pour atteindre notre objectif de recherche :
– d'étudier l'impact des microarchitectures sur la testabilité ;
– de générer des cas de tests pour les tests des patrons de conception.

1.6.1 L'étude de la testabilité des patrons de conception

Dans ce premier volet de notre projet de recherche, nous serons amenées à explorer différentes pistes que nous discutons dans la suite :

1. l'étude de la testabilité d'un patron de conception en isolation ;
2. l'étude de l'impact d'un ou plusieurs patrons de conception sur la testabilité du système.

Nous ne nous intéresserons pas dans le cadre de ce travail à l'étude de la testabilité de la composition de plusieurs patrons de conception en isolation qui sera plutôt implicite dans le point 2.

1.6.1.1 RQ1 : Quel est le niveau de testabilité des patrons de conception ?

Les patrons de conception peuvent être vus comme des éléments intermédiaires entre les classes et les systèmes. Ils sont d'une part comme les classes des éléments constitutifs des systèmes et d'autre part comme des systèmes isolés, car constituées eux-mêmes de classes en interaction. Dans cette question, les patrons de conception sont considérés comme des systèmes isolés.
L'enjeu de cette question est d'utiliser les méthodes existantes d'évaluation de la testabilité pour évaluer la testabilité des patrons de conception. Cette évaluation permettra de déterminer le degré relatif de testabilité de ces patrons de conception, de les classer et d'identifier les facteurs qui influencent cette testabilité. Elle servira de base aux concepteurs pour des choix plus éclairés quant à l'utilisation de ces patrons de conception.

Supposons que le patron de conception visiteur a une faible testabilité. Il est plausible d'imaginer que la présence du visiteur dans le système affectera négativement la testabilité de celui-ci ; ce qui signifie que ce système sera plus difficile à tester. Cette observation nous amène à ne suggérer l'utilisation de ce patron que lorsque cela est vraiment nécessaire. Et s'il s'avère que nous devions l'utiliser, ce fait nous permettra d'organiser l'effort de test en prenant en compte la difficulté occasionnée par cette utilisation.
L'étude de la testabilité du patron de conception en isolation peut également nous permettre de mettre en exergue les facteurs qui l'impactent. Cela nous permettra de proposer des variantes du patron qui auront une meilleure testabilité ou d'autres mécanismes pour améliorer cette testabilité. Les résultats de cette étude seront également utiles pour analyser l'impact des patrons de conception sur la testabilité du système.

1.6.1.2 RQ2 : Les patrons de conception ont-ils un impact sur la testabilité du système ?

Les patrons de conception étant des composants du système, il est utile d'évaluer leur impact sur la testabilité globale du système. Même si l'étude précédente nous montre que la testabilité d'un patron de conception P est élevée, cela ne nous garanti pas qu'elle n'aura pas un impact négatif sur le système où ses interactions avec les autres parties du système peut résulter à dégrader la testabilité de l'ensemble. Cette étude nous amènera donc à savoir dans quel sens, la présence des patrons de conception impactent la testabilité du système, quelles sont les interactions et également les compositions de patrons de conception qui influencent cet impact.
Nous savons par exemple que le patron de conception Abstract Factory est souvent utilisé avec le patron de conception Method Factory ou le patron de conception Prototype. On pourrait à travers cette étude mettre en évidence par exemple que la composition Abstract Factory et Method Factory est moins testable que la composition Abstract Factory et Prototype.

1.6.2 Génération de cas de tests pour les patrons de conception

Pour atteindre les objectifs du second volet de notre projet de recherche, nous nous proposons de répondre aux questions développées ci-dessous.

1.6.2.1 RQ3 : Le test de l'intention des patrons de conception peut-il améliorer la fiabilité du système ?

L'enjeu de cette question est d'étudier l'impact de tests explicites de l'intention des patrons de conception sur la qualité finale des tests. Ces tests des patrons de conception viseront essentiellement la préservation de l'intention du patron de conception. En effet, les patrons de conception véhiculent des intentions et c'est au regard de ces intentions, qu'ils sont choisis. De plus, de par les intentions qu'ils véhiculent, les patrons de conception participent à la compréhension des programmes [19]. Il est donc important que l'intention perçue au travers de la présence de ces patrons de conception soit effective surtout qu'il est possible que des maintenances soient faites à la base de ces intentions. Par exemple, le Singleton est un patron qui permet d'assurer qu'il n'y aura qu'une et une seule instance de la classe l'implémentant dans tout le système. S'assurer que cette intention est préservée de façon explicite et systématique pourra nous éviter d'avoir plusieurs instances et éviter des conséquences fâcheuses.
Un autre aspect qui motive cette question est le résultat de l'étude empirique de Aversano et al. [14]. Cette étude révèle que les patrons de conception qui participent à des fonctionnalités importantes des systèmes sont enclins aux changements et favorisent également les changements des classes avec lesquels ils interagissent. Il est donc important de s'assurer que l'intention de ces patrons de conception soit bien implémentée et aussi qu'elle soit préservée au cours des changements.

1.6.2.2 RQ4 : Tester le patron de conception comme étant un sous système peut-il réduire le coût des tests (réduire le nombre de cas de tests) ?

Soit un système orienté objet A contenant des patrons de conception. Et un système B qui est le système A dans lequel les patrons de conception sont vus comme des entités élémentaires. L'intérêt de cette question est de savoir si le système B tel que réécrit permet d'améliorer l'efficacité des tests d'intégration (réduire le nombre de cas de tests) ? Les patrons de conception ayant été testés auparavant. On vise ici à étudier les pistes qui permettraient de réduire le nombre de cas de tests nécessaires pour tester et les patrons de conception et le système B relativement au système A en considérant un critère et un taux de couvertures donnés. La réécriture des patrons de conception comme des entités élémentaires a pour but de réduire les interactions entres classes et pourrait résulter à une réduction de la complexité des tests d'intégration du système.

1.6.2.3 RQ5 : Les cas de tests pour les patrons de conception peuvent-ils être assez génériques pour être utilisés dans plusieurs systèmes ?

Cette question vise à réduire les coûts de tests des patrons de conception proposés dans les deux premières questions de cette seconde partie. En effet, l'intention encapsulée par un patron de conception est la même quelque soit l'implémentation de ce celui-ci et quelque soit le système dans lequel il se trouve. Partant de ce constat, peut on générer des cas de test pour les tests des patrons selon les requis de test identifiés dans la résolution de la question RQ3. qui pourraient servir à tester l'intention dans différents systèmes dans lesquels ces patrons sont implémentés ? Au niveau structurel, certaines variantes comme celles proposées par [11] sont très populaires ; pour ces variantes, y a-t-il un ensemble de cas de test commun aux tests du même patron de conception implémentée dans différents systèmes ? Cette étude permettrait ainsi de mettre à disposition des cas de tests réutilisables d'un système à un autre pour tester l'intention mais aussi la structure du patron de conception. Cela réduirait significativement l'effort de test tout en facilitant la mise en œuvre de ces tests.

1.6.2.4 Plan du document

La suite du présent document est structurée comme décrit ci-dessous :
Le chapitre 2 donne un aperçu de quelques travaux relatifs à notre projet de recherche.
Le chapitre 3 décrit la méthodologie que nous proposons pour résoudre nos différentes questions de recherche.
Le chapitre 4 introduit le projet sur lequel nous travaillons actuellement et qui consiste en l'analyse de l'évolution des microarchitectures.
Et enfin le chapitre 5 qui conclut ce document présente également notre planning prévisionnel.

Chapitre 2

État de l'art

Dans ce chapitre, nous décrivons quelques travaux relatifs à notre projet de recherche. La première section décrira des travaux sur la testabilité des conceptions des systèmes orientés objets. La deuxième section présentera des travaux sur les patrons de conception et les anti-patrons. Enfin, nous décrirons dans la section 3, d'autres contributions relatives à notre projet de recherche. Nous discuterons les travaux présentés à la fin de chaque section.

2.1 La testabilité, mesures et facteurs

On trouve dans la littérature de nombreux travaux sur la conception pour la testabilité (Design for Testability) dans le cadre des systèmes orientés objet. C'est un domaine de recherche devenu très actif en raison du fait que les flots de contrôle dans les systèmes orientés objets ne sont pas hiérarchiques mais plutôt diffus et repartis dans toute l'architecture [21]. De plus, certaines caractéristiques propres à l'approche orientée objet tels que l'héritage et le polymorphisme complexifient les tests [3, 5].
Ces travaux concernent l'évaluation de la testabilité [4–6, 20], l'étude des facteurs qui influencent cet attribut de qualité [3, 5, 10, 21] et aussi les approches permettant de l'améliorer [8, 9, 21] et ainsi réduire les coûts et la complexité des tests. La testabilité a été définie et évaluée selon différents points de vue. En effet, comme le mentionnent Jungmayr [10] et Baudry et al. [4], la testabilité doit être définie en fonction d'un contexte donné. Ainsi, elle sera perçue différemment au niveau des tests unitaires, d'intégration ou des tests systèmes ou même selon le critère de test retenu.
Binder [3] a défini la testabilité comme la facilité de détecter les fautes d'un logiciel. Il déclare que la testabilité permet de réduire le coût total des tests dans un processus de développement basé sur la fiabilité mais aussi d'augmenter la fiabilité du système

dans un processus à budget de test limité. Binder propose un "fishbone" de la testabilité qui vise à décrire toutes les facettes de cet attribut de qualité. Ce fishbone est basé sur six facteurs dont la combinaison détermine la testabilité du système : la représentation, l'implémentation, la construction des tests, les suites de test, l'environnement de test et le processus de développement. L'auteur détaille chacun des facteurs et énumère leurs attributs qui influencent la testabilité et dans quel sens. Ainsi, concernant la représentation du système, les exigences doivent être objectives et les spécifications complètes si on veut contribuer à garantir une bonne testabilité. Binder a également, au niveau de l'implémentation, fournit des métriques permettant d'évaluer la testabilité structurelle du système. Ces métriques sont relatives à la complexité du système et aux concepts orientés objets que sont l'héritage, le polymorphisme et l'encapsulation.

Jungmayr [10] a défini une approche d'évaluation de la testabilité basée sur les dépendances entre composants. Un composant peut être un module, une classe, une interface, un "package" ou un sous système. L'auteur se place ainsi dans la perspective des tests d'intégration. La dépendance d'un composant A vis-à-vis d'un composant B signifie que le composant A a besoin du composant B pour compiler ou bien fonctionner. Selon l'auteur, la testabilité du système décroit lorsque le nombre de dépendances croît. L'auteur a définit un ensemble de métriques qui permet d'évaluer la testabilité du système à travers les dépendances qui existent entre les composants. Il a également défini la notion de dépendances critiques pour les tests. Une dépendance critique aux tests est une dépendance qui a un impact élevé sur la testabilité globale du système relativement aux autres dépendances. L'auteur fournit, à l'aide des métriques définies, une technique pour identifier ces dépendances et des pistes de refactoring pour les supprimer. Les études de cas sur cette approche montrent que l'analyse des dépendances critiques peut considérablement aider à détecter les mauvaises décisions de conception qui affectent la testabilité et à ainsi les corriger dès la conception.

Baudry et al. [4, 21] se sont quant à eux intéressé aux interactions entre les classes. Ils ont proposé une approche permettant de détecter les problèmes de testabilité dans un diagramme de classes UML. Cette approche est basée sur la notion d'anti-patrons de testabilité qui correspondent à des configurations indésirables dans le diagramme des classes susceptibles d'augmenter l'effort de test. Deux anti-patrons ont été identifiés : les interactions entre classes et l'auto-utilisation d'une classe. Les auteurs ont proposé un modèle dérivé du diagramme de classes qui permet la détection des anti-patrons de testabilité et le calcul de leur complexité. La testabilité globale du système est alors évaluée comme le nombre et la complexité des anti-patrons de testabilité existant dans le diagramme. Pour améliorer la testabilité du système au niveau conceptuel, les auteurs ont suggéré l'ajout de stéréotypes pour aider les programmeurs à préserver l'intention des concepteurs et à éviter les interactions entre classes inutiles.

Baudry et al. [2] ont également proposé une analyse de la testabilité en appliquant leur

approche d'évaluation définie dans [21] aux patrons de conception définis dans [11]. Ils ont ainsi déterminé théoriquement en considérant le pire des cas, la sévérité de testabilité de chaque patron. Ils ont par la suite proposé l'application automatique de stéréotypes aux patrons de conception sous forme de contraintes dans un méta niveau d'UML afin d'en améliorer la testabilité. Cette technique a été expérimentée avec un patron de conception, la Fabrique Abstraite.

Mouchawrab et al. [5] ont proposé un "framework" générique d'évaluation de la testabilité au niveau conceptuel pour différentes activités de tests mais aussi à partir de différents artefacts conceptuels. Les auteurs ont identifié pour cela un certain nombre d'attributs pouvant affecter la testabilité. Ces attributs ont été classés suivant les activités ou sous activités qu'ils influencent. Les auteurs ont ensuite défini pour chaque attribut sa relation de cause à effet avec la testabilité. Ces descriptions ont été traduites en hypothèses qui pourraient être testées par la suite, répondant ainsi à l'un des principaux objectifs des auteurs : fournir une base théorique pour la recherche empirique dans le domaine de la testabilité. Leur second objectif était de fournir une plateforme générique et un modèle pour évaluer efficacement la testabilité au niveau conceptuel et aider à son amélioration. Les auteurs, ont pour cela, défini pour chacun des attributs un ensemble de mesures permettant de l'évaluer. Ils ont précisé également les modèles UML permettant de déterminer chaque mesure. Enfin, les auteurs ont donné des directives quant à l'utilisation du framework et comment il pourrait aider dans l'amélioration de la testabilité des conceptions.

Discussion

Les travaux décrits ci-dessus donnent un aperçu de la diversité de points de vue concernant la testabilité, son évaluation et les facteurs qui l'influencent. Cependant, il n'existe à notre connaissance que peu d'études sur l'impact que pourraient avoir les microarchitectures en particulier les patrons de conception et les anti-patrons sur la testabilité du système (cf section 2.3).

L'étude de Baudry et al [2] concernant l'évaluation de la testabilité des patrons de conception est un moyen de contourner la complexité à évaluer la testabilité de toute la conception. Les auteurs ont ainsi utilisé les patrons de conception dans une perspective de décomposition de la conception en vue d'analyses et d'améliorations locales de la testabilité. Cette théorie n'est malheureusement pas supportée par des expérimentations. De plus, cette étude ne fait pas cas de l'impact des patrons de conception sur la testabilité globale du système qui les utilise. Les patrons de conception ainsi que les anti-patrons, comme le montrent les travaux présentés dans la section 2, impactent de nombreux attributs de qualité et constituent des facteurs plus visibles et plus faciles à contrôler à priori. En effet, la plupart des facteurs identifiés dans les différents travaux

sont difficiles à appréhender avant la fin de la conception. Nous nous proposons dans le cadre de notre recherche d'explorer l'impact des microarchitectures sur la testabilité des systèmes et fournir ainsi des recommandations qui peuvent guider à priori les concepteurs à une conception testable.
Dans le cadre de notre projet de recherche, nous nous baserons également sur les approches d'évaluation existantes sur la conception telles que celles décrites ci-dessus. La méthode d'évaluation de Binder évalue la testabilité au niveau de l'implémentation. Elle n'est donc pas appropriée à notre étude qui se penche sur l'évaluation de la testabilité au niveau conceptuel. Ce travail donne néanmoins une vue panoramique de la testabilité et des facteurs qui peuvent l'influencer à différents stades du développement ou des tests. Les travaux de Jungmayr, Baudry et al. et Mouchawrab et al. seraient donc de bons candidats pour l'évaluation de la testabilité des patrons de conception et des systèmes au niveau conceptuel. Le framework de Mouchawrab et al. semble être un cadre adaptable et extensible pour diverses évaluations de la testabilité selon le contexte choisi et les artefacts de conception disponibles. Une étude plus approfondie de ces approches nous permettra de déterminer celles qui seront appropriées à l'évaluation de la testabilité des microarchitectures et des systèmes les contenant.

2.2 Les microarchictectures et leur impact sur les attributs de qualité

Il existe de nombreux travaux sur les patrons de conception et les anti-patrons. Ces travaux traitent de différentes facettes de ces microarchitectures : leurs définitions plus ou moins formelle [11, 15, 22, 23], leurs impacts sur la qualité des systèmes [14, 17, 24] ou encore des méthodes pour les détecter [18, 25, 26]. Cette riche littérature sur ces microarchitectures montre leur importance dans la conception des systèmes mais aussi leur influence sur la qualité des systèmes. Nous allons dans cette section présenter quelques-uns de ces travaux, en particulier ceux qui étudient l'impact de ces microarchitectures sur les attributs de qualité.

2.2.1 Description des patrons de conception et des anti-patrons

Gamma et al. [11] peuvent être perçus comme à l'origine de la popularité des patrons de conception. Ils ont défini selon le modèle de la figure 2.1 23 patrons de conception qui font partis des plus utilisés. Ce modèle fournit des informations utiles pour la compréhension de ces patrons, le contexte de leur utilisation, leurs structures (appelés motifs de conception) et aussi des exemples d'implémentation. Le motif d'un patron de

FIGURE 2.1 – Template de description d'un patron de conception

conception est la partie la plus visible du patron et qu'on retrouve dans les conceptions : c'est l'organisation des classes qui représentent ce patron de conception. Plus formellement, Guéhéneuc et Antoniol [18] définissent les motifs de conception comme des solutions idéales qui décrivent les rôles joués par les classes qui implémentent les patrons de conception. Gamma et al. ont regroupé les 23 patrons de conception dans 3 catégories. Les patrons de conception de création facilitent la création d'objets. Les patrons de conception comportementaux défini les interactions entre objets et la distribution des responsabilités. Les patrons de conception structuraux aident quant à eux dans la composition d'objets.

Les anti-patrons sont avec les patrons de conception, les microarchitectures les plus connus dans les systèmes orientés objets. Les anti-patrons, contrairement aux patrons de conception sont considérés comme des solutions de conception pauvres [15] ayant, un impact négatif sur les attributs de qualité du système et l'évolution des systèmes [15].

Brown et al. [15] ont effectué avec les anti-patrons un travail similaire à celui de Gamma et al. concernant la description des patrons de conception. Ils ont ainsi répertorié et décrit textuellement 40 anti-patrons selon le modèle décrit à la figure 2.2. Pour chacun d'eux, ils précisent entre autre le contexte et les causes qui entrainent à cet anti-patron, une solution de refactoring qui permet de le supprimer, les symptômes pour le reconnaître, ses caractéristiques générales et les conséquences de sa présence.

Discussion

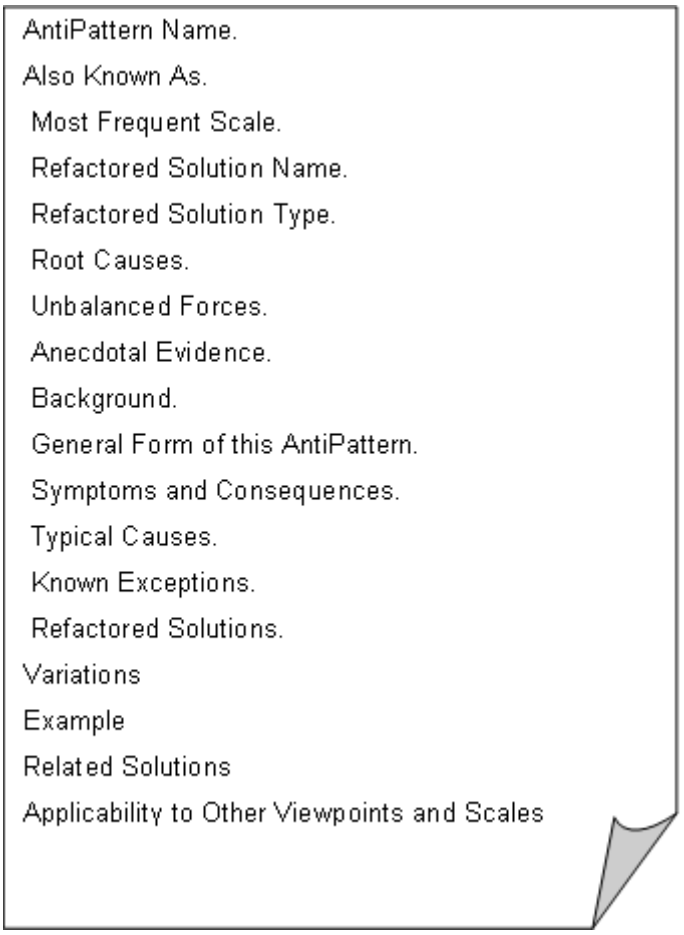

FIGURE 2.2 – Template de description d'un anti-patron

Les livres de Gamma et al. et Brown et al. sont des références pour appréhender les patrons et en avoir une bonne connaissance. Nous nous servirons d'eux comme tels dans le cadre de notre travail de recherche.
Les patrons de conception définis par Gamma et al. nous servirons de base dans notre étude et nous nous référerons généralement aux motifs qu'ils présentent comme variantes principales. Nous utiliserons également les définitions de l'intention qu'ils fournissent pour élaborer les requis de tests des intentions des différents patrons de conception. De même, nous nous appuierons sur l'ensemble défini par Brown et al. pour définir notre sous-ensemble d'anti-patrons à étudier.

2.2.2 Impact des patrons de conception et des anti-patrons sur les attributs de qualité

Bieman et al. [13] ont examiné à travers 3 systèmes industriels et 2 systèmes open source, les prédispositions des patrons de conception aux changements. Ils ont ainsi découvert dans 4 des 5 systèmes étudiés que les classes qui jouent un rôle dans des motifs de

conception changent plus fréquemment que les autres. Contrairement donc aux idées préconçues, l'utilisation des patrons de conception ne garanti pas l'adaptabilité.
Vokac [24] a analysé à travers une étude de cas les maintenances correctives hebdomadaires d'un logiciel commercial sur une période de 3 ans. Le but de cette étude était de confirmer ou d'infirmer la présomption selon laquelle les patrons de conception sont moins sujets aux fautes et permettraient de réduire le nombre de défaut du système. À partir des "snapshots" hebdomadaires de ce logiciel, Vokac a détecté automatiquement les occurrences des motifs de ces 5 patrons de conception dans le logiciel et a comparé le taux de défauts détectés dans les classes participant à ces occurrences à celui des autres classes du système. Les résultats de cette étude de cas montrent une grande variation du taux de défauts d'un patron de conception à un autre : de 63% à 154 % en moyenne. L'auteur a expliqué ces variations par la taille et la complexité des classes participant à l'implémentation des différents patrons de conception. Ainsi, il a découvert que les patrons de conception Observateur et Singleton sont plus sujets aux défauts car s'appuyant sur de grandes classes. L'auteur conclut donc que l'utilisation des patrons de conception ne garantit pas une réduction des fautes et que certains patrons de conception sont sujets aux fautes et ce même s'ils sont bien implémentés.
L'étude d'Aversano et al. [14] vient apporter une autre précision aux changements fréquents que subissent les motifs de conception. En effet, après l'analyse de trois systèmes open source, les auteurs ont conclu que les motifs de conception qui participent à des fonctionnalités importantes du système changent plus fréquemment. La prédisposition aux changements ne dépend donc pas du type du motif de conception mais du rôle auquel il participe dans le système. Aversano et son équipe indiquent aussi les types de changement qui affectent ces motifs de conception : changement d'implémentation, ajout de sous classes ou changement des interfaces. Ce dernier changement crée de nombreux co-changements dans les classes clientes. Les auteurs appellent à d'autres études pour confirmer ces résultats et recommandent une utilisation prudente des patrons de conception.
Di Penta et al. [27] ont effectué une étude empirique similaire à celle de Bieman et al. [13] afin d'identifier la relation entre les rôles dans les patrons de conception et leurs prédispositions aux changements. Ils ont pour cela analysé l'évolution des motifs de conception à travers 3 logiciels open source de tailles et de domaines différents : JHotDraw, Eclipse-JDT, and Xerces. Leur but était de déterminer la prédisposition aux changements des classes jouant différents rôles dans des motifs de conception et le type de changements qui affectaient ces classes. La plupart des résultats de l'étude rejoigne ceux de Bieman et al. [13] et concorde avec les suppositions faites au sujet des différents motifs de conception : certains rôles dans les motifs de conception favorisent des changements fréquents chez les classes les implémentant. Par exemple, les auteurs ont observé que les classes qui jouent le rôle de l'adaptateur sont plus prédisposées aux changements que les classes

adaptées dans le motif Adaptateur.

Khomh et Guéhéneuc [28] ont mené une étude empirique afin de déterminer l'impact des patrons de conception sur un certain nombre d'attributs de qualité. Ils ont pour cela fait un sondage auprès de développeurs et mainteneurs expérimentés. L'analyse des résultats de ce sondage a montré que contrairement aux idées préconçues, les patrons de conception n'amélioraient pas toujours les attributs de qualité. En effet, certains attributs de qualité dont la compréhensibilité, la reusabilité et l'extensibilité pouvaient être négativement influencés par l'utilisation de certains patrons de conception. Ces auteurs ont donc eux aussi appelé à la prudence dans l'utilisation des patrons de conception aux vus de ces résultats.

Wei et Raed [16] ont réalisé une étude empirique qui met en évidence la relation entre les anti-patrons et la prédisposition d'une classe aux fautes durant l'évolution du système. Les auteurs ont analysé trois versions d'Eclipse et ont montré que les classes participant aux anti-patrons God Class, Shotgun Surgery, and Long Method ont une plus grande prédisposition aux fautes que les autres classes. Ils suggèrent d'utiliser cette connaissance durant le développement pour détecter les classes susceptibles de contenir des fautes car c'est dans cette phase que la probabilité d'introduction aux fautes est plus grande. Ils invitent néanmoins à d'autres études pour soutenir leurs résultats.

Khomh et al. [17] ont quant à eux exploré l'impact des anti-patrons sur les prédispositions aux changements des classes qui les contiennent. Ils ont pour cela analysé les relations entre 29 anti-patrons et les changements des classes dans plusieurs versions d'Azureus et d'Eclipse. Les résultats de cette étude empirique montrent que les anti-patrons ont un impact négatif sur la prédisposition aux changements des classes qui les contiennent. Les auteurs conseillent donc d'éviter d'avoir un grand nombre d'anti-patrons dans le système au risque d'avoir un système peu adaptable.

Discussion

Les travaux décrits dans cette section montrent le grand intérêt porté aux patrons de conception et aux anti-patrons et surtout leur impact sur les attributs de qualité et sur l'évolution des systèmes.

Bien que certains travaux aient montré les avantages de l'utilisation des patrons de conception sur les attributs de qualité des systèmes, les nombreuses déclarations allant dans ce sens proviennent surtout d'intuitions ou d'idées préconçues. Ces intuitions ont, comme le démontrent quelques travaux ici, été parfois infirmées. Ainsi, les patrons de conception peuvent impacter négativement certains attributs de qualité. Ce fait montre l'importance des études empiriques pour soutenir les hypothèses ou les relations de cause à effet. De plus, ce type d'études permet de mettre en évidence les causes à la base de

certains phénomènes, permettant ainsi d'y apporter des solutions adéquates. Finalement, ces études sont également importantes car elles donnent des bases fiables pour prendre des décisions rationnelles et justifiées quant à l'utilisation ou non des patrons de conception ou encore au refactoring ou non des anti-patrons. Comme nous l'avons déjà mentionné, la testabilité est un attribut de qualité qui est aujourd'hui intensivement étudié en vue de contribuer à réduire les coûts et la complexité des tests. Ces études qui mettent à profit des expérimentations et des cas d'étude pour établir des relations entre les patrons de conception et les attributs de qualité nous inspirent et nous motivent à explorer l'impact des patrons de conception et des anti-patrons et la testabilité.
De plus, plusieurs travaux mentionnés ci-dessus mettent en évidence les prédispositions aux fautes et aux changements des classes participant aux motifs de conception ou aux anti-patrons. Ces résultats renforcent l'objectif de notre étude à nous pencher sur les liens entre ces microarchitectures et les tests. Ils incitent également à mettre l'accent sur ces classes au moment des tests afin de réduire les défauts qu'ils pourraient contenir de même que leur propagation. Ces conclusions constituent aussi un motif pour explorer les pistes qui permettraient de supporter et faciliter les tests des patrons de conception.

2.3 Les tests et les microarchitectures

Izurieta et al. [29] ont examiné les conséquences qu'entrainent les "grime buildup" sur la testabilité de la conception. Les auteurs ont défini les grime buildup comme du code ajouté au cours de l'évolution du système dans les classes participant à un motif de conception mais qui ne contribuent pas à la mission de ce motif. Les grime buildup sont également à l'origine de relations entre les classes du motif de conception et des classes ne jouant aucun rôle dans le motif de conception. Les auteurs ont réalisé dans [30] une étude qui a permis de mettre en évidence la construction des grime buildup dans les motifs de conception au cours de l'évolution du système. Le but de l'étude de [29] était de déterminer l'impact de ses grime buildup sur le nombre total de requis de test nécessaires pour tester les patrons de conception. Les auteurs ont dans le cadre de leur étude utilisé la mesure anti-patrons de testabilité, définies par Baudry et al. dans [4, 21], pour évaluer la testabilité des patrons de conception. À travers une expérience sur plusieurs versions de l'outil de refactoring JRefactory, Izurieta et al. ont montré que les grime buildup entrainaient la croissance du nombre de requis de test en augmentant le nombre d'anti-patrons de testabilité. Les auteurs ont donc suggéré au vu de cet impact négatif sur la testabilité, la suppression des grime buildup le plutôt possible.
Neelam et al. [31] ont proposé une approche afin de réduire les efforts de test des patrons de conception dans les systèmes. Ils ont pour cela défini pour chaque patron de conception un ensemble de "templates" de cas de test ("pattern test case templates ou PTCTs").

Ces templates codifient une structure de cas de test réutilisable conçue pour identifier les défauts associés aux applications d'un patron de conception donné dans tous les systèmes l'implémentant correctement. Ces templates ont été définies sur la base de de la formalisation des patrons de conception proposée par les auteurs dans [22, 32]. Cette approche est basée sur l'utilisation de contrats et de sous contrats appelées contrat de patrons. Les PTCTs doivent être adaptés en vue de leur utilisation dans un système donné. Cette approche a été illustrée sur l'exemple d'un patron de conception Observer.

Discussion

L'étude d'Izurieta et al. ont étudié l'impact de l'évolution des patrons de conception dans le système sur la testabilité de celui-ci. Bien que cette étude ne soit pas généralisable (utilisation d'un seul système), elle fournit des résultats intéressants qui appellent à être attentif à l'évolution des patrons de conception dans le système pour éviter de décroître la testabilité de tout le système. Notre étude se differencie de celle-ci car elle s'interresse à l'impact de l'utilisation des patrons de conception sur la testabilité des conceptions.
L'approche de Neelam et al. propose un parallèle de réutilisation des patrons de conception dans les conceptions dans les tests. Cette approche des cas de test génériques qui peuvent être ensuite adaptés suivant le système peut contribuer à réduire les coûts de tests et augmenter la fiabilité du système. Malheureusement, cette approche n'est pas supportée par des expérimentations concrètes montrant son applicabilité et son efficacité. De plus elle est basée sur une formalisation précise qui est une contrainte dans son utilisation. Elle reste néanmoins une bonne base pour explorer cette piste qui est l'objet des travaux futurs de notre projet de recherche.

Chapitre 3

Méthodologie

Dans le but d'atteindre notre objectif de recherche (cf. section 1.3) et de répondre aux différentes questions de recherche (cf section 1.6), nous suivrons la méthodologie décrite dans ce chapitre. Cette méthodologie décrite question par question, donne une vue générale du processus que nous mettrons en œuvre pour répondre à chacune de nos questions de recherche. Les diagrammes des figures 3.1 et 3.2 décrivent le processus de résolution de chaque question et les liens entre les différents processus. Ils sont expliqués dans les sous-sections qui suivent.

3.1 L'étude de la testabilité des patrons de conception

3.1.1 RQ1 : Quel est le niveau de testabilité des patrons de conception ?

Étape 1 : Identification de la méthode d'évaluation de la testabilité
Pour résoudre la première question de notre projet de recherche, nous commencerons par étudier les méthodes d'évaluation de la testabilité des systèmes orientés objets. Cette étude nous permettra d'identifier la méthode qui sera la plus appropriée à l'évaluation de la testabilité des patrons de conception. Nous nous intéresserons aux méthodes d'évaluation au niveau conceptuel. Il existe dans la littérature plusieurs approches [5, 21, 33] dans ce domaine utilisant différentes techniques et mettant en évidence différents facteurs influençant la testabilité (par exemple, les dépendances entre composants, la complexité). Des études préliminaires et expérimentales sur les performances de ces méthodes à évaluer et à prédire la testabilité d'une conception nous aideront dans notre choix. La méthode retenue devra notamment permettre de collecter les informations nécessaires à

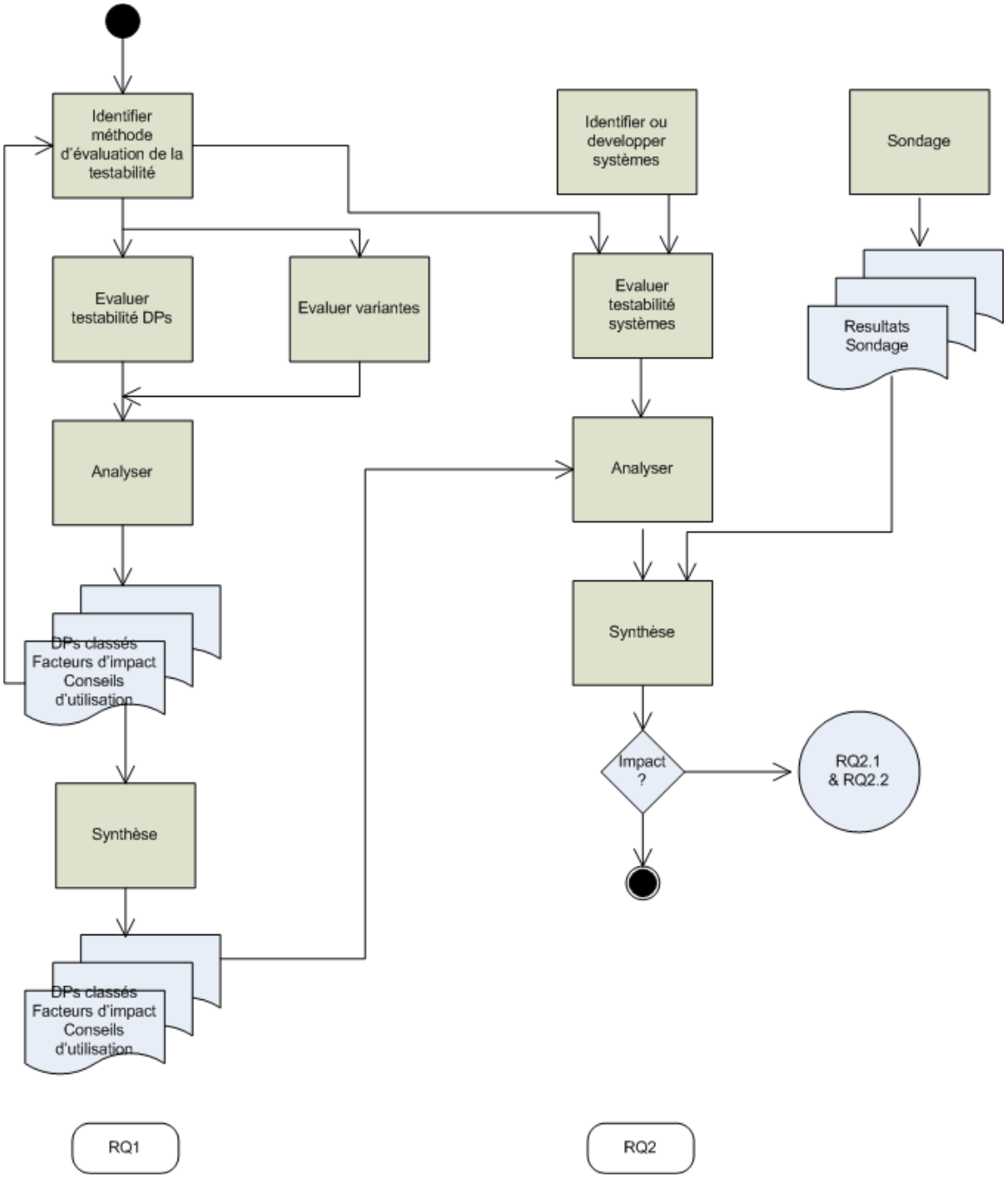

FIGURE 3.1 – Méthodologie de résolution des questions RQ1 et RQ2

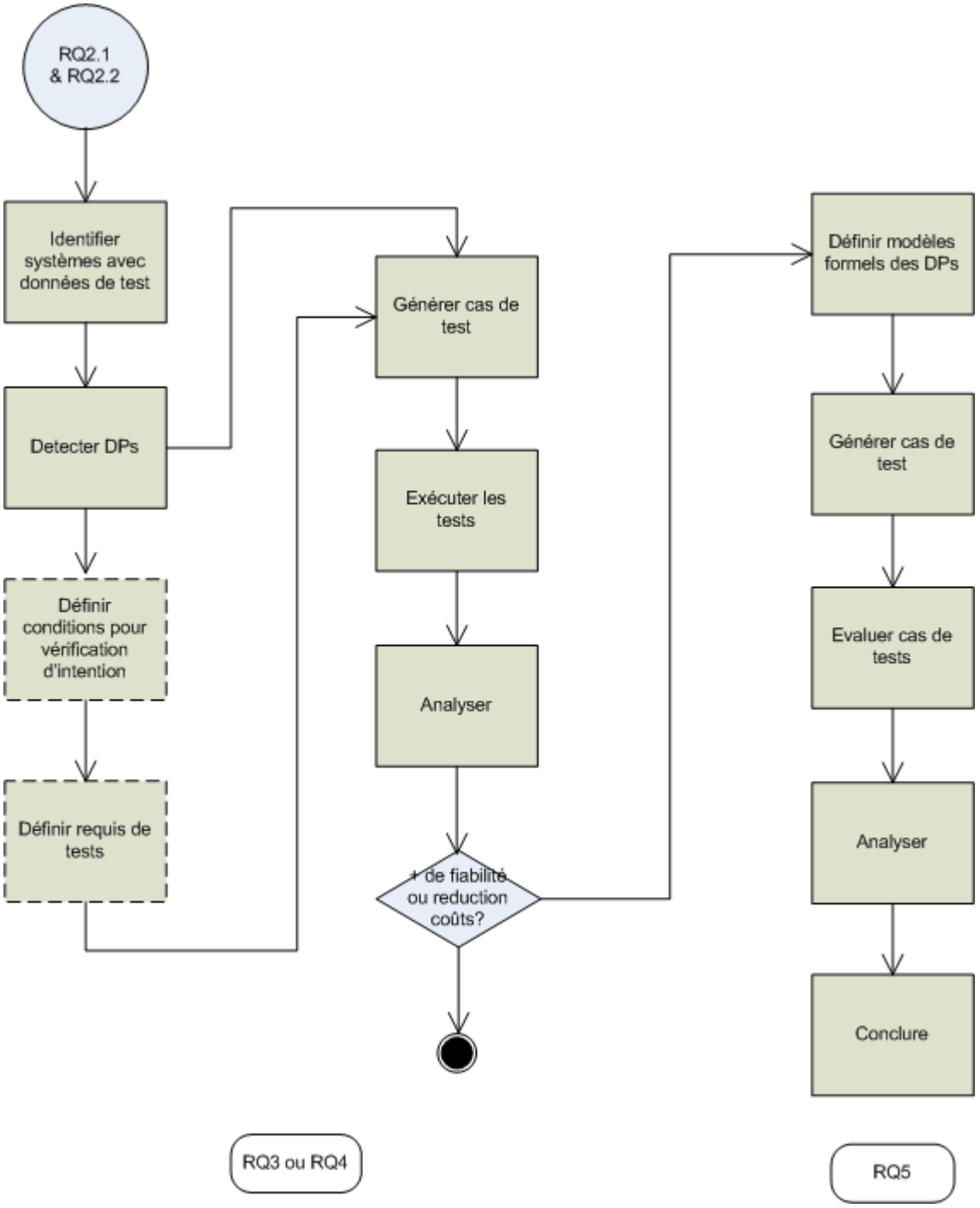

FIGURE 3.2 – Méthodologie de résolution des questions RQ3, RQ4 et RQ5

travers des diagrammes UML ou dérivés.

Étape 2 : Évaluation de la testabilité des patrons de conception
À l'aide de la méthode identifiée à l'étape 1, nous évaluerons la testabilité de chaque patron de conception défini dans [11]. Nous pourrons ensuite classer les patrons de conception de chaque catégorie selon leur degré de testabilité. L'analyse des résultats de cette évaluation permettra en outre de déterminer les facteurs qui impactent la testabilité des patrons de conception et de proposer des recommandations pour l'améliorer au besoin.
Étape 3 : Évaluation de la testabilité des variantes de patrons de conception
Une autre étape en parallèle de l'étape 2, sera d'évalué la testabilité des variantes des patrons de conception. On pourra ainsi mettre en évidence et expliquer les différences entre les différentes variantes. Un autre classement pourra être ainsi élaboré : classer les différentes variantes selon le degré de testabilité.
Étape 4 : Synthèse
La dernière étape dans la résolution de cette question permettra de faire la synthèse des différents résultats et améliorer si possible les recommandations qui seront proposées à l'étape 2. Le processus décrit ci-dessus pourra être réitéré avec d'autres méthodes d'évaluation afin de confirmer les résultats. Cela conduira à considérer d'autres facteurs et à avoir des conclusions plus généralisables.
La mise en évidence des facteurs qui influencent la testabilité des patrons de conception pourront être utiles dans l'analyse de l'impact des patrons de conception sur la testabilité du système.

3.1.2 RQ2 : Les patrons de conception ont-ils un impact sur la testabilité du système ?

Étape 1 : Identification des systèmes pour l'expérience
Afin d'évaluer si les patrons de conception ont un impact sur la testabilité des systèmes qui les contiennent, il nous faudra identifier des paires de systèmes, l'un contenant des motifs de conception et l'autre non. On pourrait pour cela utiliser des versions d'un même système dans lequel des refactorings (donc pas de modification de fonctionnalités) auraient supprimé ou rajouté des motifs de conception. À défaut de paires identiques, nous pourrons également utiliser des paires de systèmes ou sous systèmes de tailles et de fonctionnalités similaires, l'un contenant des motifs de conception et l'autre n'en contenant pas. Pour identifier ces systèmes, nous nous aiderons de la littérature, des bases de logiciels et également d'historiques de logiciels. Une autre option serait de développer nous-mêmes ces paires de systèmes. Afin de pouvoir étudier l'impact de la présence d'un

seul patron et celui de la composition de plusieurs patrons de conception, il nous faudrait deux groupes de systèmes : le premier groupe contiendra des systèmes (ou sous systèmes) avec un seul motif de conception et le second groupe des compositions de plusieurs motifs de conception. Nous aurons également besoin dans cette phase d'outils de détection de patrons afin de détecter les patrons de conception des différents systèmes. Nous pourrions, par exemple, utiliser l'outil proposé par Guéhéneuc et Antoniol [18], DEMIMA. Notre objectif étant d'évaluer la testabilité au niveau de la conception, il faudrait au besoin reconstruire les modèles conceptuels des systèmes identifiés à l'aide d'outils de retroconception. En effet, généralement, les modèles conceptuels des systèmes sont soit inexistants soit obsolètes. Il existe de nombreux outils de retroconception selon les langages de programmation, les environnements ou encore les modèles qu'ils fournissent. Par exemple, les outils de la suite Ptidej [1] permet la retroconception de programmes écrits en Java. JUPE [2] est quant à lui un plugin integré à Eclipse qui permet de reconstruire les diagrammes de classes des programmes écrits en Java [3]. On peut également citer BoUML [4] pour les programmes écrits en Java, C++, PHP et ArgoUML [5] pour des programmes écrits en Java. Les paires de systèmes identifiées à cette étape serviront de base pour la seconde étape de ce processus.

Étape 2 : Évaluation de la testabilité de chaque système

Les méthodes d'évaluation identifiées dans la résolution de la question RQ1, nous servirons à évaluer la testabilité de chacun des systèmes identifiés. Nous pourrons ainsi comparer pour un même système, sa testabilité dans sa variante avec motifs de conception à celle de sa variante sans motifs de conception. Nous pourrions ainsi déterminer si la présence des motifs de conception influence la testabilité des systèmes et dans quel sens.

Étape 3 : Sondage

Toujours dans le but de répondre à cette question, un sondage sera lancé auprès des ingénieurs en test afin de recueillir leurs avis sur la testabilité des systèmes contenant des motifs de conception. Ils nous diront à travers ce sondage qui pourrait se faire via un questionnaire, s'ils pensent que la présence des motifs de conception influence la testabilité des systèmes. Si oui, ils pourraient classer les patrons de conception selon cet impact, identifiant ainsi le pire des patrons de conception en termes d'impact sur la testabilité et à l'opposé le meilleur. Ce classement servira à comparer les avis des ingénieurs aux résultats obtenus à l'étape 2. Les ingénieurs pourraient également nous révéler si cet impact pourrait résulter d'autres facteurs comme le rôle auquel participe le motif de conception (important, moyen, périphérique), le langage de programmation

1. http ://www.ptidej.net/research/
2. http ://jupe.binaervarianz.de/
3.
4. http ://bouml.free.fr/
5. http ://argouml.tigris.org/

ou la méthode et le critère de couverture utilisés.

Étape 4 : Analyse et synthèse

Si cette étude nous révèle un impact des patrons de conception sur la testabilité des systèmes, nous devrions alors déterminer les facteurs de cet impact afin de proposer des recommandations ou techniques pour le réduire au besoin mais aussi suggérer des conseils quant à l'utilisation des patrons de conception. Il serait également utile de refaire le processus avec d'autres systèmes et d'autres groupes d'ingénieurs afin de confirmer et généraliser nos résultats. La révélation de cet impact nous conduira également à investiguer l'exploitation des patrons de conception dans les tests, objet du second volet de notre projet de recherche. Nous décrivons dans la section qui suit les étapes qui pourraient nous mener à répondre aux questions de ce volet. Si par contre les résultats ne révèlent pas d'impact, nous recommencerons l'étude afin de confirmer ces résultats.

3.2 Génération de cas de tests pour les patrons de conception

3.2.1 RQ3 : Le test de l'intention des patrons de conception peut-il améliorer la qualité du système ?

Étape 1 : Identification des systèmes pour l'expérience

La première étape pour répondre à cette question consistera à identifier des systèmes contenant des motifs de conception et idéalement dont les données de tests sont disponibles à l'instar de JHotDraw et JUnit. Nous pourrions être amenés à développer les tests si nécessaire. Les systèmes identifiés dans le processus de résolution de la question RQ2 pourront à nouveau être utilisés. Les outils de détection des patrons de conception nous permettront une fois de plus de détecter les motifs de conception de ces systèmes.

Étape 2 : Test de l'intention des patrons de conception

Si les tests effectués sur les systèmes identifiés à l'étape 1 contiennent des tests explicites de l'intention des patrons de conception, nous analyserons les résultats de ces tests afin d'évaluer leur apport dans la détection de fautes. Sinon pour chaque système, nous définirons des listes de conditions que doit satisfaire l'intention des patrons de conception correspondants aux motifs de conception qu'ils contiennent. Ces conditions seront ensuite traduites en requis de test. Nous générerons enfin des cas de test à la base de ces requis. L'exécution de ces cas de test sur le système pourrait permettre de déceler d'éventuelles fautes. Nous pourrons aussi injecter des fautes qui violent ces requis et voir si ces cas de test permettent de les détecter. L'injection de fautes pourra également permettre d'évaluer les cas de test du système (sans test d'intention) par rapport à la détection de ce genre de fautes.

Étape 3 : Analyse des résultats
La dernière étape dans la résolution de cette question consistera à analyser les résultats des expérimentations de l'étape précédente. Nous évaluerons l'apport des listes de conditions et des tests correspondants dans la préservation de l'intention des patrons de conception. La généralisation de ces résultats dépendra du nombre, du type et de la taille des systèmes étudiés. Nous pourrons donc être amenés à réitérer ce processus afin de pouvoir généraliser nos conclusions.

3.2.2 RQ4 : Tester le patron de conception comme étant un sous système peut-il réduire le coût des tests (réduire le nombre de cas de tests) ?

La résolution de cette question est semblable à la précédente sauf qu'ici, nous nous intéresserons à la structure des patrons de conception. Les processus de résolution de ces deux questions peuvent être mis en œuvre en parallèle. L'ensemble des systèmes ayant servi aux expérimentations dans le processus précédent pourra être utilisé dans cette phase.
Étape 1 : Test des motifs de conception et test du système réécrit
L'idée dans cette phase est de tester séparément les motifs de conception et ensuite de tester le système entier en considérant ces motifs de conception comme des entités élémentaires du système. Nous générerons et exécuterons donc des cas de tests pour les motifs de conception à l'aide de la même méthode et suivant le critère de test utilisés pour les tests d'intégration du système afin de ne pas introduire de biais. Ensuite, des cas de test du système avec les motifs de conception comme entités élémentaires seront également générés et exécutés avec pour objectif d'atteindre le même taux de couverture que les tests originels.
Étape 2 : Analyse des résultats de tests
Cette étape permettra d'évaluer la pertinence de l'approche proposée grâce aux résultats de l'étape précédente. Si le nombre de cas de test utilisés dans nos tests est inférieur au nombre de cas de test utilisés pour tester le système et disponibles grâce aux données de test, nous pourrons alors conclure à la pertinence de l'approche proposée dans la recherche de l'efficacité des tests et de la réduction des coûts de test. D'autres aspects tels que la facilité de générer les cas de test, d'écrire l'oracle ou encore d'exécuter les cas de test seront également pris en compte dans cette évaluation. L'itération de ce processus sera nécessaire afin de généraliser ce résultat. Si par contre, la réduction du nombre de cas de tests n'est pas effective avec l'approche proposée, nous explorerons d'autres approches de tests afin de déterminer celles qui pourraient être appropriées à la réécriture du système que nous proposons.

3.2.3 RQ5 : Les cas de tests pour les patrons de conception peuvent-ils être assez génériques pour être utilisés dans plusieurs systèmes ?

Cette phase de notre travail de recherche est surtout une perspective de nos travaux futurs. Elle est conditionnée par les résultats des deux précédentes questions. En effet, si le test de l'intention ou de la structure des motifs de conception apporte un gain dans l'efficacité des tests ou une réduction des coûts de test, il serait alors utile de disposer si possible de cas de test génériques pour faciliter de tels tests. Nous nous inspirerons alors des travaux présentés dans [31] pour investiguer cette piste.

Chapitre 4

Projet en cours

Dans le cadre de notre travail de recherche, nous avons initié une étude expérimentale qui vise à identifier les tendances d'évolution des microarchitectures en particulier celles décrites par les patrons de conception et les anti-patrons dans les logiciels. Cette investigation nous permettra de répondre entre autre à des questions telles que : comment et quand est introduite une microarchitecture, comment et quand disparait-t-elle ? Se transforme-t-elle vers une autre microarchitecture entre ces deux moments ? Quelle est son évolution dans le temps ? Le but de cette analyse est de dégager les tendances d'évolution de ces microarchitectures dans les systèmes et partant de là leur impact sur l'évolution des systèmes. Cette analyse servira également à déterminer leur impact sur les attributs de qualité des systèmes notamment la testabilité qui constitue l'un des volets de notre projet de recherche. En outre, nous pourrons mettre en évidence les causes de l'introduction ou de la suppression des microarchitectures et aussi les facteurs qui influencent leur évolution selon telle ou telle tendance. Les résultats de la présente étude pourront servir ainsi de base pour prédire l'évolution des microarchitectures d'un système et prendre ainsi les mesures adéquates. Elle pourra également servir de base pour élaborer des mécanismes de prévention ou de correction de leur impact.

4.1 Données de l'expérience

Nous avons, pour cette expérience, collecté les codes sources d'Eclipse JDT à intervalle de deux semaines de janvier 2002 à décembre 2009. Nous disposons donc d'un total de 192 snapshots pour effectuer notre analyse.
Eclipse JDT est un plugin d'Eclipse qui implémente l'environnement de développement intégré pour Java de la plateforme d'Eclipse. Il permet ainsi le développement, la compilation et l'exécution des projets Java dans Eclipse. Eclipse a été développé à la fois par

une compagnie industrielle et une communauté du logiciel libre. C'est donc un logiciel qui incorpore les pratiques industrielles. Il est largement adopté tant dans l'industrie que dans le domaine de la recherche. C'est également l'un des logiciels utilises dans l'étude [34] sur laquelle est basée la présente étude et qui est décrite dans la section suivante.
Ce logiciel se prête bien à notre étude en raison du fait que plusieurs travaux ont mis en évidence le fait qu'il contenait une grande variété et un grand nombre de microarchitectures [17, 27**?**]. Aussi sa longue durée de vie et la disponibilité de son code source fait de lui un candidat idéal pour ce genre d'étude.
Les outils de détection des microarchitectures de la suite ptidej nous aideront à détecter dans les différents snapshots les patrons de conception et les anti-patrons à analyser.

4.2 Travail précurseur

Notre étude s'inspire du travail présenté dans [34] et se veut une extension de celui-ci. Vaucher et al. [34] ont, à travers différentes versions des systèmes Xerces et Eclipse, étudié l'évolution de l'anti-patron "God class". Leur but était d'identifier comment cette classe était introduite dans le système, comment elle y évoluait et comment elle en était supprimée afin de proposer des mécanismes de prévention et de correction. La God class, comme le mentionnent les auteurs, se caractérise par un grand nombre de méthodes non reliées et est généralement perçue comme le résultat d'une mauvaise conception qu'il faut supprimer. Elle est souvent créée accidentellement au cours de l'évolution du logiciel par ajouts successifs de méthodes mais peut être dans certaines circonstances, un choix de conception éclairé et réfléchi.
Cette étude a permis aux auteurs d'identifier 8 tendances d'évolution de la God class (cf Figure 4.1). Ces tendances ont été identifiées en utilisant la technique de clustering basée sur le Dynamic Time-Warping sur les God classes identifiées dans les différentes versions d'Eclipse JDT et de Xerces.

Les auteurs ont également mis en évidence les facteurs qui favorisaient certaines de ces tendances. Ces résultats constituent ainsi une base pour élaborer des mécanismes de prévention et de correction en vue d'éviter l'introduction accidentelle des God classes mais aussi au besoin les supprimer ou réduire leurs effets. Bien que cette étude soit soumise à des limitations du point de vue de la généralisation, elle constitue un bon point de départ pour de futures études plus généralisables et aussi l'analyse d'autres microarchitectures. En effet, les auteurs ont utilisé 22 versions d'Eclipse JDT pour leur étude et nous envisageons pour la notre, analyser 192 snapshots. Cette étude a été effectuée uniquement sur l'anti-patron God class. Nous envisageons la reprendre sur un plus grand nombre de microarchitectures. Le sous ensemble de microarchitectures à étudier sera identifié par une étude préliminaire.

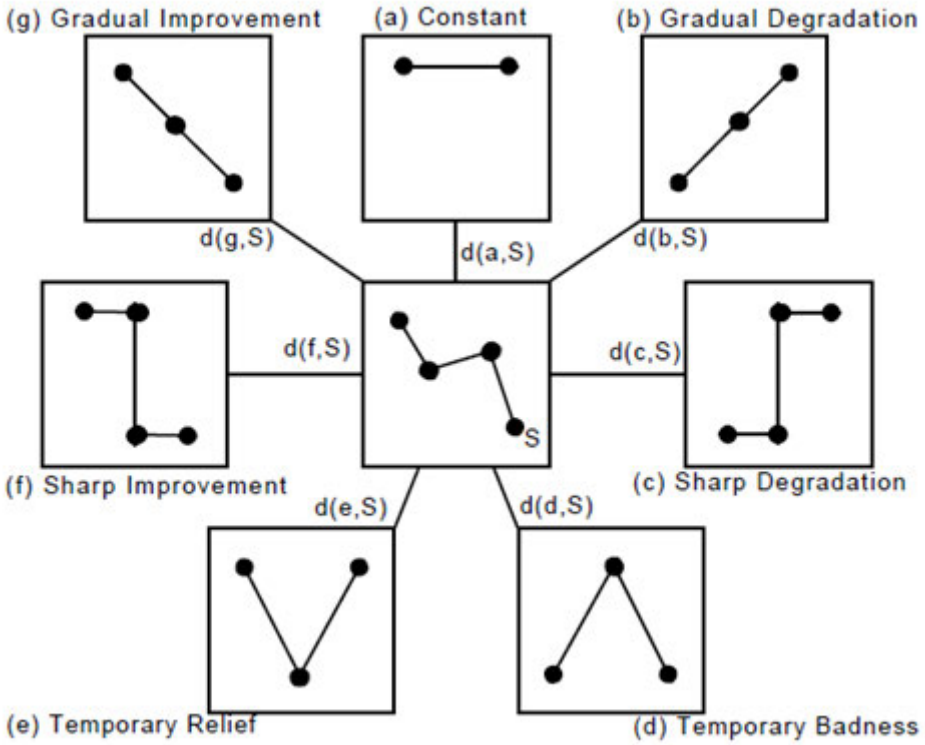

FIGURE 4.1 – Tendances d'évolution des God Classes [34]

Les tendances identifiées par Vaucher et al. seront-elles-confirmées par notre étude ? Si tel est le cas, nous analyserons d'autres logiciels afin de généraliser ces résultats. L'analyse de ces résultats nous permettra de faire des recommandations pour corriger ou prévenir une évolution négative d'une microarchitecture. Ces recommandations pourront être par la suite supportées par des outils de refactoring. Si par contre, nos résultats ne confirment pas ceux Vaucher et al., d'autres analyses nous permettrons d'expliquer les différences et de mener d'autres expériences afin de confirmer et généraliser nos résultats.

4.3 L'étude de l'impact des patrons de conception et des anti-patrons sur la testabilité

La base de codes sources utilisée dans le cadre de cette étude servira également dans le cadre de l'étude de l'impact des microarchitectures sur la testabilité du système. En effet, cette étude nous permettra d'identifier les snapshots dans lesquels des microarchitectures sont introduites ou supprimées. Nous pourrons ainsi évaluer la testabilité des snapshots précédents ceux-ci afin de voir l'impact de l'introduction ou de la suppression d'une microarchitecture sur le système et aussi analyser les facteurs conduisant à un tel impact.

Il faut cependant que cette introduction ou cette suppression ne rajoute ou n'enlève pas de fonctionnalité ; typiquement une introduction ou une suppression de refactoring.

Chapitre 5

Planning prévisionnel et conclusion

5.1 Planning prévisionnel

Le tableau ?? décrit le planning prévisionnel de notre projet de recherche et à titre indicatif, quelques conférences auxquelles nous envisageons présenter les résultats de nos travaux. Nous espérons pouvoir également publier nos travaux dans des journaux tels que TSE, TOSEM.

	A10	H11	E11	A11	H12	E12	A12	H13	E13	Publications
Projet en cours	x	x								WCRE
RQ1	x	x								ICSM
RQ2		x	x							ICST
RQ3				x	x					ICSE
RQ4					x	x				ISSTA
Redaction							x	x		
Soutenance									x	

Table 5.1 – Planning prévisionnel du déroulement du projet de recherche

Légende : A : Automne, H : Hiver, E : Été

Projet en cours : Identification et analyse des tendances d'évolution des microarchitectures

RQ1 : Quel est le niveau de testabilité des patrons de conception ?

RQ2 : Les patrons de conception ont-ils un impact sur la testabilité du système ?

RQ3 : Le test de l'intention des patrons de conception peut-il améliorer la fiabilité du système ?

RQ4 : Tester le patron de conception comme étant un sous système peut-il réduire le coût des tests ?

5.2 Conclusion

Les logiciels sont aujourd'hui indispensables dans presque tous les domaines de notre société. Leur fiabilité est donc un réel enjeu. Les activités de test, bien que très complexes et coûteuses, demeurent le moyen le plus répandu pour assurer la fiabilité des systèmes et augmenter notre confiance en eux. Le présent projet de recherche vise à contribuer à réduire les coûts de test et augmenter la fiabilité des systèmes en exploitant les microarchitectures, en particulier celles décrites par les patrons de conception et les anti-patrons. À cette fin, nous avons proposé une méthodologie et un planning prévisionnel qui devront nous aider à atteindre cet objectif et à produire des résultats qui nous l'espérons seront d'un apport significatif dans la recherche de la réduction des coûts des tests et par voie de conséquence des coûts de développement logiciel.

Bibliographie

[1] Mary Jean Harrold. Testing : a roadmap. In *ICSE '00 : Proceedings of the Conference on The Future of Software Engineering*, pages 61–72, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. doi : http://doi.acm.org/10.1145/336512.336532.

[2] Benoit Baudry, Yves Le Traon, Gerson Sunyé, and Jean-Marc Jézéquel. Measuring and improving design patterns testability. In *METRICS '03 : Proceedings of the 9th International Symposium on Software Metrics*, page 50, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1987-3.

[3] Robert V. Binder. Design for testability in object-oriented systems. *Commun. ACM*, 37(9) :87–101, 1994. ISSN 0001-0782. doi : http://doi.acm.org/10.1145/182987.184077.

[4] Benoit Baudry and Yves Le Traon. Measuring design testability of a uml class diagram. *Inf. Softw. Technol.*, 47(13) :859–879, 2005. ISSN 0950-5849. doi : http://dx.doi.org/10.1016/j.infsof.2005.01.006.

[5] Samar Mouchawrab, Lionel C. Briand, and Yvan Labiche. A measurement framework for object-oriented software testability. *Inf. Softw. Technol.*, 47(15) :979–997, 2005. ISSN 0950-5849. doi : http://dx.doi.org/10.1016/j.infsof.2005.09.003.

[6] Richard Bache and Monika Mullerburg. Measures of testability as a basis for quality assurance. *Softw. Eng. J.*, 5(2) :86–92, 1990. ISSN 0268-6961.

[7] Magiel Bruntink and Arie van Deursen. An empirical study into class testability. *J. Syst. Softw.*, 79(9) :1219–1232, 2006. ISSN 0164-1212. doi : http://dx.doi.org/10.1016/j.jss.2006.02.036.

[8] Jeffery Payne, Roger T. Alex, and Charles D. Hutchinson. Design-for-testability for object-oriented software, 1997.

[9] Bret Pettichord and Bret Pettichord. Design for testability. In *In Pacific Northwest Software Quality Conference*, 2002.

[10] Stefan Jungmayr. Testability measurement and software dependencies, 2002.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[12] Bill Venners. How to use design patterns A conversation with Erich Gamma, part I, year = 2005, publisher=http ://www.artima.com/lejava/articles/gammadp.htm.

[13] James M. Bieman, Greg Straw, Huxia Wang, P. Willard Munger, and Roger T. Alexander. Design patterns and change proneness : An examination of five evolving systems. In *METRICS '03 : Proceedings of the 9th International Symposium on Software Metrics*, page 40, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1987-3.

[14] Lerina Aversano, Gerardo Canfora, Luigi Cerulo, Concettina Del Grosso, and Massimiliano Di Penta. An empirical study on the evolution of design patterns. In *ESEC-FSE '07 : Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 385–394, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-811-4. doi : http://doi.acm.org/10.1145/1287624.1287680.

[15] William J. Brown, Raphael C. Malveau, Hays W. McCormick III, and Thomas J. Mowbray. *AntiPatterns : Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, Inc., March 1998.

[16] Wei Li and Raed Shatnawi. An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. *J. Syst. Softw.*, 80 (7) :1120–1128, 2007. ISSN 0164-1212. doi : http://dx.doi.org/10.1016/j.jss.2006.10.018.

[17] Foutse Khomh, Massimiliano Di Penta, and Yann-Gael Gueheneuc. An exploratory study of the impact of code smells on software change-proneness. *Reverse Engineering, Working Conference on*, 0 :75–84, 2009. ISSN 1095-1350. doi : http://doi.ieeecomputersociety.org/10.1109/WCRE.2009.28.

[18] Yann-Gael Gueheneuc and Giuliano Antoniol. Demima : A multilayered approach for design pattern identification. *IEEE Transactions on Software Engineering*, 34 : 667–684, 2008. ISSN 0098-5589. doi : http://doi.ieeecomputersociety.org/10.1109/TSE.2008.48.

[19] Danny B. Lange and Yuichi Nakamura. Interactive visualization of design patterns can help in framework understanding. *SIGPLAN Not.*, 30(10) :342–357, 1995. ISSN 0362-1340. doi : http://doi.acm.org/10.1145/217839.217874.

[20] John D. McGregor and Satyaprasad Srinivas. A measure of testing effort. In *COOTS'96 : Proceedings of the 2nd conference on USENIX Conference on Object-Oriented Technologies (COOTS)*, pages 10–10, Berkeley, CA, USA, 1996. USENIX Association.

[21] Benoit Baudry, Yves Le Traon, and Gerson Sunyé. Testability analysis of a uml class diagram. In *In Proceedings of the Ninth International Software Metrics Symposium (METRICS03*, pages 54–66. IEEE Computer Society, 2002.

[22] Neelam Soundarajan and Jason O. Hallstrom. Responsibilities and rewards : Specifying design patterns. In *ICSE '04 : Proceedings of the 26th International Conference on Software Engineering*, pages 666–675, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2163-0.

[23] Holger Kampffmeyer. *The Design Pattern Intent Ontology- Finding the Pattern you need.* VDM Verlag, Saarbrücken, Germany, Germany, 2007. ISBN 3836411849, 9783836411844.

[24] Marek Vokac. Defect frequency and design patterns : An empirical study of industrial code. *IEEE Trans. Softw. Eng.*, 30(12) :904–917, 2004. ISSN 0098-5589. doi : http://dx.doi.org/10.1109/TSE.2004.99.

[25] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T. Halkidis. Design pattern detection using similarity scoring. *IEEE Trans. Softw. Eng.*, 32(11) :896–909, 2006. ISSN 0098-5589. doi : http://dx.doi.org/10.1109/TSE.2006.112.

[26] Naouel Moha, Yann-Gael Gueheneuc, Laurence Duchien, and Anne-Francoise Le Meur. Decor : A method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering*, 36 :20–36, 2010. ISSN 0098-5589. doi : http://doi.ieeecomputersociety.org/10.1109/TSE.2009.50.

[27] Massimiliano Di Penta, Luigi Cerulo, Yann gaël Guéhéneuc, and Giuliano Antoniol. An empirical study of the relationships between design pattern roles and class change proneness, 2008.

[28] Foutse Khomh and Yann-Gael Gueheneuc. An empirical study of design patterns and software quality. pages pages 1–19, 2008. doi : http://doi.ieeecomputersociety.org/10.1109/ICSM.2009.5306327.

[29] Clemente Izurieta and James M. Bieman. Testing consequences of grime buildup in object oriented design patterns. In *ICST '08 : Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*, pages 171–179,

Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3127-4. doi : http://dx.doi.org/10.1109/ICST.2008.27.

[30] Clemente Izurieta and James M. Bieman. How software designs decay : A pilot study of pattern evolution. In *ESEM '07 : Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, pages 449–451, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2886-4. doi : http://dx.doi.org/10.1109/ESEM.2007.58.

[31] Neelam Soundarajan, Jason Hallstrom, Guoqiang Shu, and Adem Delibas. Patterns : from system design to software testing. *Innovations in Systems and Software Engineering*, 4 :71–85, 2008. ISSN 1614-5046. URL http://dx.doi.org/10.1007/s11334-007-0042-z. 10.1007/s11334-007-0042-z.

[32] Benjamin Tyler, Jason O. Hallstrom, and Neelam Soundarajan. A comparative study of monitoring tools for pattern-centric behavior. In *SEW '06 : Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*, pages 37–46, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2624-1. doi : http://dx.doi.org/10.1109/SEW.2006.5.

[33] S. Jungmayr. Identifying test-critical dependencies. *Software Maintenance, IEEE International Conference on*, 0 :0404, 2002. doi : http://doi.ieeecomputersociety.org/10.1109/ICSM.2002.1167797.

[34] Stephane Vaucher, Foutse Khomh, Naouel Moha, and Yann-Gael Gueheneuc. Tracking design smells : Lessons from a study of god classes. *Reverse Engineering, Working Conference on*, 0 :145–154, 2009. ISSN 1095-1350. doi : http://doi.ieeecomputersociety.org/10.1109/WCRE.2009.23.

L'École Polytechnique se spécialise dans la formation d'ingénieurs et la recherche en ingénierie depuis 1873

École Polytechnique de Montréal

École affiliée à l'Université de Montréal

Campus de l'Université de Montréal
C.P. 6079, succ. Centre-ville
Montréal (Québec)
Canada H3C 3A7

www.polymtl.ca

ÉCOLE
POLYTECHNIQUE
MONTRÉAL