FUNDP - Namur
Faculté d'Informatique
Rue Grandgagnage, 21
B-5000 Namur

Evaluation de l'impact d'un patron de conception sur la compréhension et la maintenance de programmes

-

Une expérimentation par un système d'eye-tracking

Sébastien Jeanmart

Mémoire présenté en vue de l'obtention
du grade de maître en informatique

Troisième Maîtrise
Année académique 2007-2008

Résumé

Le génie logiciel existe depuis environ une quarantaine d'années. La recherche empirique dans ce domaine a, quant à elle, considérablement mûri ces 10–20 dernières années [Perry 00]. Néanmoins, l'absence de consensus témoigne d'un manque de maturité du domaine. Plusieurs publications témoignent des débats ayant actuellement lieu. Ces derniers tentent de trouver un moyen de créer des études plus rigoureuses et de formuler des interprétations plus crédibles à partir des résultats obtenus.

Ce mémoire a pour objectif, dans un premier temps, de tenter de mieux cerner les techniques propres au génie logiciel dit empirique, et ce, en passant par la réalisation personnelle d'une expérience. L'objectif dans un second temps, est d'identifier les différents problèmes inhérents à la réalisation de telles expériences dans le domaine informatique.

Cette recherche expérimentale a permis d'obtenir certains résultats qui ont fait l'objet d'un article, soumis à la conférence «*ACM SIGSOFT 2008 / FSE 16*».

Mots clés : génie logiciel empirique, maintenance, design pattern, eye-tracking, compréhension de programmes, diagramme de classes UML

Abstract

Software engineering arose about forty years ago. Empirical research in this domain has considerably matured over the last 10–20 years [Perry 00]. Nevertheless, the current lack of consensus gives evidence of the lack of maturity in this field. Many workshops are currently working on finding new techniques to create more rigorous studies and to formulate more credible interpretations based on these studies.

This master thesis is aimed firstly at attempting to better understand the different strategies in empirical software engineering. This can be performed by conducting an experimentation. The second aim of this master thesis is to identify different issues inherent to the conduct of such experimentation in computer science field.

This empirical research leads to some results presented in a submitted paper to the «*ACM SIGSOFT 2008 / FSE 16*» symposium.

Keywords : empirical software engineering, program maintenance, design pattern, eye-tracking, program comprehension, UML class diagram

Ce mémoire est le fruit d'un stage de fin d'études réalisé à l'Université de Montréal (UdeM) au Canada par un étudiant de troisième maîtrise en informatique aux Facultés Universitaires Notre-Dame de la Paix (FUNDP) de Namur en Belgique.

Je tiens à remercier toutes les personnes sans qui ce mémoire n'aurait pu voir le jour.

En particulier, je remercie mon promoteur, le Professeur Naji Habra, de m'avoir trouvé ce stage dans des conditions un peu particulières. Je le remercie également pour ses conseils avisés lors de la rédaction de ce mémoire, ainsi que pour les nombreux débats intéressants que nous avons eu autour de la rédaction d'articles scientifiques.

Je tiens également à remercier mes maîtres de stage, les Professeurs Houari Sahraoui et Yann-Gaël Guéhéneuc, pour leur chaleureux accueil au sein des équipes GEODES et Ptidej. Je les remercie également pour leur soutien et leurs conseils durant le stage mais aussi après celui-ci.

Finalement, je souhaiterais remercier ceux sans qui cette formidable aventure n'aurait pu voir le jour, en particulier mes parents et ma grand-mère, mais aussi ma famille de manière générale.

Table des matières

Table des figures

Liste des tableaux

Introduction

Chaque projet informatique de grande taille suit un processus de développement qui, quelque soit les différences de découpe, d'agencement et de structuration, inclut invariablement différentes *activités* telles que la définition des exigences, la conception de l'architecture, l'implémentation, l'intégration, etc. Cependant, la manière dont les projets sont menés à bien et la manière dont les outils sont utilisés, varient fortement d'une entreprise à l'autre. Certaines entreprises suivront un processus plus rigide, tandis que d'autres s'en remettront plutôt aux décisions de managers, basées sur leur propre expérience professionnelle [Perry 00]. A cet effet, bon nombre de modèles de cycle de vie ont d'ailleurs été présentés, chacun répondant de manière plus adéquate à certains contextes dans lesquels s'inscrivent différents projets.

La situation actuelle indique un important écart entre la recherche et la pratique en génie logiciel. Nous ne connaissons actuellement pas les mécanismes fondamentaux qui définissent les coûts et les bénéfices des méthodes et des outils utilisés en GL[Perry 00]. Les études empiriques sont potentiellement une manière d'obtenir ces informations, permettant ainsi de prendre des décisions sur une base plus objective et plus rationnelle.

Le génie logiciel (GL) existe depuis environ une quarantaine d'années. La recherche empirique dans ce domaine a considérablement mûri ces 10–20 dernières années [Perry 00]. Ce type de recherche définit un processus, précisant rigoureusement les actions à réaliser à chaque étape de la recherche. Cependant, comme tout processus en génie logiciel, cette approche présente un certain nombre de forces mais aussi de faiblesses. Les études empiriques sont donc actuellement au coeur des débats afin de trouver un moyen de créer des études plus rigoureuses et de formuler des interprétations plus crédibles à partir de celles-ci.

Ce mémoire s'inscrit dans le contexte du génie logiciel, dit empirique. La **première partie** de ce mémoire consistait à réaliser une étude empirique afin d'assimiler les concepts liés à ce type de recherche en génie logiciel. Cette étude empirique avait également pour objectif de comprendre comment se déroulait précisément la conduite d'une «*expérience*». La **seconde partie** de ce mémoire était destinée à critiquer l'expérience réalisée et à lister les problèmes inhérents à la conduite d'un tel type d'étude. Ces critiques ont permis de rejoindre des constats formulés par plusieurs auteurs d'articles scientifiques en génie logiciel empirique (GLE). Cette expérience est aussi une opportunité d'apporter une vue concrète des problèmes qui peuvent se présenter dans la recherche empirique.

L'expérience menée par ce travail a cherché à étudier l'impact que pouvait avoir le patron de conception (*design pattern*) du «*Visiteur*» sur la compréhension et la maintenance de programmes orientés-objet (OO). Pour ce faire, un échantillon de programmes OO a été représenté à l'aide de diagrammes de classes UML, et un système d'*eye-tracking* a été utilisé afin de capturer l'attention des sujets sur les différentes parties de ces diagrammes.

Depuis leur parution dans le célèbre livre du «GoF» (*Gang-of-Four*), terme faisant référence aux quatre auteurs célèbres de cet ouvrage, les patrons de conception n'ont cessé de susciter un intérêt grandissant au sein du monde académique et professionnel. Des 23 patrons initialement présentés dans le GoF en 1994, certains auteurs n'en recensent pas moins d'une centaine actuellement [Beck 96]. Néanmoins, parmi les vertus qui sont généralement conférées à ces patrons, telles que par exemple l'amélioration de la *flexibilité* ou de la *réutilisabilité*, peu d'articles apportent des résultats empiriques pour valider ces propos. La pertinence des patrons de conception est, de ce fait, tout simplement remise en question. Si l'on veut contrôler les impacts que les patrons de conception engendrent et prouver leur utilité à la communauté, il est nécessaire d'apporter des preuves empiriques.

La technologie de l'*eye-tracking* permet, comme son nom l'indique, d'identifier les zones d'une image ou d'une interface par exemple, pour lesquelles une personne porte un intérêt. Un grand nombre de disciplines, telles que les sciences cognitives, l'interaction Homme-Machine, la recherche en marketing ou encore la recherche médicale, utilisent des techniques d'*eye-tracking*. Le domaine de la compréhension de programmes a permis notamment de comprendre la création de modèles mentaux, l'identification d'informations pertinentes, mais aussi de proposer de nouvelles techniques pour représenter l'information. La technologie de l'*eye-tracking* permet d'aller plus loin dans l'activité de compréhension étant donné qu'elle fournit de l'information représentant l'aperçu des zones attirant l'attention [Duchowski 03]. Ces données permettent, par la suite, d'inférer sur les processus cognitifs sous-jacents [Rayner 98]. Les récentes avancées techniques et l'accessibilité de cette technologie sont également des raisons qui ont poussé à y avoir recours dans cette expérience.

Après avoir réalisé cette expérience, plusieurs éléments témoignent de la difficulté qu'un tel type d'étude implique. Un certain nombre de problèmes vont dans le sens des discours actuels, apportant des preuves évidentes qu'un manque de maturité persiste au sein de génie logiciel empirique. Le manque d'études empiriques, de consensus et de guidelines dans les différents domaines abordés au cours de cette expérience ont donné lieu à certains problèmes de conception, d'interprétation, mais aussi de validation. Ces problèmes ne permettent pas d'assurer, à l'heure actuelle, que les résultats de l'expérience reflètent les traitements observés.

Afin de mieux comprendre la manière dont s'articule ce mémoire, un bref résumé des différents chapitres est mis à la disposition du lecteur :

- Le **Chapitre 1** présente ce qu'on entend par «*Génie Logiciel Empirique*» (GLE) et décrit son contexte d'apparition. Ce chapitre présente successivement (1) les différents types de stratégies empiriques actuelles, (2) une introduction à la théorie de la mesure et (3) dresse l'état actuel de la maturité du GLE.

- Le **Chapitre 2** se base sur le livre de C. Wohlin, [Wohlin 00], pour présenter, succinctement (Section 2.1) ou de manière détaillée, le processus d'une «*expérience*». Celui-ci correspond au processus suivi et mis en application dans ce mémoire.

- Le **Chapitre 3** présente successivement (1) la problématique générale et (2) particulière de l'expérience réalisée.

- Le **Chapitre 4** présente au lecteur les éléments techniques qui ont permis de construire une solution à la problématique particulière, exposée au Chapitre 3.

- Le **Chapitre 5** présente, succinctement (Section 5.1) ou de manière détaillée, l'expérience réalisée. Cette dernière met en application les différentes étapes du processus présenté au Chapitre 2. Cette expérience vise à répondre à la problématique particulière, exposée au Chapitre 3, à l'aide des éléments techniques introduits au Chapitre 4.

- Le **Chapitre 6** répertorie les différents problèmes majeurs rencontrés tout au long de l'application du processus d'une expérience.

- Le **Chapitre 7** dresse une liste des leçons apprises au cours de cette expérience. Ces constatations concernent aussi bien le processus d'une expérience que la technologie d'*eye-tracking* utilisée.

Le lecteur familier du génie logiciel empirique (GLE) pourra passer le Chapitre 1. Néanmoins, s'il désire découvrir ou se remémorer les problèmes actuels en GLE, la Section 1.5 présente l'état de la maturité de cette discipline.

Le lecteur familier de l'expérimentation, et en particulier du livre de C. Wohlin, [Wohlin 00], disposera d'un bref rappel des concepts et du processus d'une expérience au Chapitre 2, Section 2.1. La suite de ce chapitre peut lui être dispensée.

Le lecteur intéressé par la technologie d'*eye-tracking* pourra lire, dans un premier temps, l'état de l'art rapide au Chapitre 4, Section 4.3. Le lecteur pourra, dans un second temps, lire, via le Chapitre 5, Section 5.3.6, la présentation du système utilisé durant l'expérience. Afin de comprendre les problèmes que cette technologie a engendrés tout au long du processus de cette expérience, le lecteur est invité à parcourir le Chapitre 6. Finalement, le Chapitre 7, Section 7.1 présente les quelques leçons apprises au sujet de l'utilisation du système considéré, mais aussi de l'utilisation de la technologie d'*eye-tracking* en général.

Le lecteur intéressé par les patrons de conception (design patterns) est invité à parcourir l'état de l'art qui s'y rapporte, au Chapitre 3, Section 3.2.1. Le lecteur pourra ensuite lire plus en détails les résultats particuliers qui ressortent de cette expérience, via le Chapitre 5, Section 5.5.

De manière générale, le lecteur pressé, ou tout autre lecteur désirant se faire une idée assez rapide de ce mémoire, est invité à lire les quelques sections suivantes. Afin de bien comprendre les problèmes qui résident actuellement en GLE, le lecteur est invité à parcourir la Section 1.5 du Chapitre 1 qui présente la maturité de cette discipline. Le lecteur pourra ensuite se familiariser rapidement avec le vocabulaire et le processus d'une expérience via la Section 2.1 du Chapitre 2. La connaissance du processus permettra au lecteur de mieux comprendre la découpe opérée aux Chapitres 5 et 6. Il est ensuite conseillé au lecteur de parcourir la problématique particulière du mémoire au Chapitre 3, Section 3.3. Le Chapitre 4 peut être parcouru rapidement, voire ignoré si le lecteur n'attache pas d'importance particulière à la présentation des diagrammes de classes UML et à la technologie d'*eye-tracking*. Le Chapitre 5 présente l'expérience réalisée. Si le lecteur ne désire pas s'encombrer de détails, la Section 5.1 présente succinctement les informations élémentaires de chaque étape du processus de l'expérience réalisée. Il est ensuite conseillé au lecteur de lire le Chapitre 6 qui présente les problèmes que cette expérience a rencontrés. Ces problèmes permettront au lecteur de mieux comprendre les leçons apprises, présentées au Chapitre 7.

1 Génie Logiciel Empirique

Sommaire

Avant-propos

Conduire une expérience nécessite une certaine rigueur. Afin de mieux comprendre le cadre dans lequel s'inscrivent les différentes démarches présentées en seconde partie de ce mémoire, une introduction au cadre du «génie logiciel empirique» est offerte au lecteur à travers ce chapitre.

Celui-ci s'inspire en partie de l'ouvrage intitulé ***«Experimentation In Software Engineering : An Introduction»*** *de* ***Claes Wohlin et al.*** [Wohlin 00].

1.1 Introduction

L'introduction de la technologie de l'information dans la vie quotidienne a eu notamment pour conséquence que le logiciel fasse partie de plus en plus de produits. Il est possible de retrouver ces logiciels dans bon nombre de produits allant du célèbre grille-pain à la navette spatiale. Pour ce faire, il a fallu développer un grand nombre de logiciels.

Le développement de logiciels n'est pas une tâche facile, de plus, il s'agit d'un processus extrêmement créatif. L'évolution rapide du domaine a fait que la masse des projets informatiques a considérablement augmenté, résultant en grande partie en de multiples problèmes. L'absence de fonctionnalités nécessaires, les surcoûts, le dépassement des échéances et la faible qualité des logiciels en général font partie des problèmes généralement présentés dans l'histoire du développement de logiciels [Pressman 00]. Malgré ces constatations, le «*Standish Group International, Inc.*» publie près de 20 ans plus tard, [SGI 94], les mêmes causes d'échecs des projets informatiques.

C'est en 1968 que le terme «*génie logiciel*» vit le jour. Une volonté certaine de créer une discipline d'ingénieur, basée sur le développement de systèmes informatiques, se fit alors ressentir. Depuis lors, bon nombre d'organisations et de personnes ont tenté de définir formellement le concept de «génie logiciel». C'est notamment le cas de l'IEEE qui définit le génie logiciel comme «*une application d'une approche systématique, disciplinée et quantifiable aux étapes de développement, d'opération et de maintenance de logiciel*» [IEEE 90]. La théorie du génie logiciel est également présentée et discutée dans des livres tels que [Pressman 00, Pfleeger 01].

Les recherches en génie logiciel évaluent et comparent notamment les effets de l'utilisation de certaines technologies dans des environnements aux interactions complexes [Sjoberg 07]. Ces dernières font intervenir des personnes, des équipes, des projets, des organisations et différents types de tâches et de logiciels [Sjoberg 07].

Afin d'évaluer les technologies en génie logiciel, une approche empirique, incluant une collaboration avec le monde industriel, a débuté dans les années 70. Depuis lors, un intérêt grandissant est apparu au niveau de l'application de méthodes empiriques dans les recherches en génie logiciel [Perry 00].

Cependant, les recherches empiriques actuelles sont loin de la vision initialement prévue. A titre d'exemple, Zelkowitz *et al.* ont réalisé une étude dans le but d'examiner les conceptions expérimentales et les collectes de données en génie logiciel sur une période de 10 ans [Zelkowitz 98]. Les résultats montrent clairement que plus d'un tiers des articles scientifiques ne contient aucune validation expérimentale. Plusieurs articles discutent des problèmes actuels dans la recherche empirique et présentent quelques idées afin d'améliorer les études futures [Basili 96, Perry 00, Basili 99a, Sjoberg 07]. Ces débats seront présentés à travers la Section 1.5 de ce chapitre.

Le but de ce chapitre est de fournir un aperçu global du génie logiciel empirique. Pour ce faire, la Section 1.2 présente les rapports qui existent entre le génie logiciel et la science. La Section 1.3 décrit les différentes stratégies empiriques. La Section 1.4 fournit, quant à elle, les bases de la théorie de la mesure. Finalement, la Section 1.5 résume la situation actuelle du génie logiciel empirique.

1.2 Science et génie logiciel

1.2.1 Contexte du génie logiciel

Un modèle du processus de développement de logiciel a pour but de décrire les différentes étapes à réaliser ainsi que les activités qui composent ce développement. Parmi ces modèles, on retrouve le modèle en cascade, le développement incrémental, le développement évolutif ainsi que le modèle en spirale. Il est possible de retrouver un grand nombre de livres dans la littérature débattant de ces modèles [Pressman 00]. Le processus de conception de logiciel peut se représenter de manière très simplifiée tel que présenté en Figure 1.1.

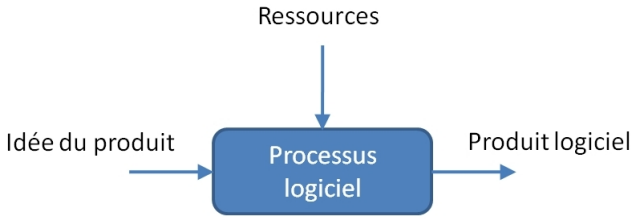


FIG. 1.1 – Illustration du processus logiciel

L'«*idée du produit*» ainsi que les «*ressources*» sont en réalité des personnes qui constituent les «*inputs*» de ce processus. Le «*produit logiciel*» est ensuite développé par des personnes à travers différentes étapes qui forment le processus de conception de logiciel.

La complexité du logiciel à développer peut résulter en un processus s'étendant sur une très grande période, impliquant la présence de beaucoup de personnes. Durant le développement, plusieurs documents sont délivrés avant de fournir le produit final. La complexité du processus de développement est telle qu'il est difficile de trouver une seule bonne manière de faire. Les entreprises essaient donc en permanence d'améliorer leur processus afin d'améliorer leurs produits, de diminuer leurs coûts, mais aussi d'avoir une meilleure maîtrise des processus. Tout comme pour le processus de développement, une approche systématique et disciplinée est requise pour l'amélioration de processus elle-même. Il est donc nécessaire de voir l'amélioration de processus comme un processus.

Le Quality Improvement Paradigm (QIP) [Basili 85] ainsi que le Plan/Do/Check/Act [Demming 86] sont des exemples de processus d'amélioration de développement de logiciels. Le QIP consiste en un ensemble d'étapes qui supporte une approche disciplinée et systématique dans le but d'améliorer un processus. Le second exemple, quant à lui, est un processus d'amélioration plus général. Cependant, il existe 2 activités communes aux processus d'amélioration :

- L'évaluation du processus de développement
- L'évaluation d'une proposition d'amélioration du processus de développement

Le premier type d'évaluation a pour but d'identifier les parties du processus qui permettent des améliorations. Bon nombre de modèles existent en matière d'évaluation de processus. Le plus connu est certainement le Capability Maturity Model Integration (CMMI) [Ahern 05]. Ces modèles d'évaluation permettent d'identifier les pratiques qui nécessitent des améliorations.

Le second type d'évaluation a pour but de déterminer comment améliorer réellement ces pratiques nécessitant des améliorations. L'objectif ici est de proposer des processus concrets d'amélioration sur base à la fois de l'évaluation du processus actuel et de la connaissance actuelle de l'état de l'art. Une fois ces propositions d'amélioration identifiées, il est nécessaire de déterminer celles qui seront introduites, et dans quel cadre. Cependant, il est très souvent impossible de déterminer quelles seront les conséquences de l'amélioration du processus. C'est pourquoi, il est indispensable d'évaluer ces propositions d'amélioration avant d'appliquer réellement des changements au sein du processus.

Le problème est que l'évaluation de propositions d'amélioration de processus ne peut se faire sans la présence d'une personne. Pour évaluer un produit, il est possible de concevoir un prototype de ce produit et d'obtenir le consentement des personnes afin de savoir si les démarches vont dans le bon sens. Dans le cas d'un processus, il est impossible de concevoir un prototype [Wohlin 00]. Le seul choix qui s'offre est de réaliser des simulations et de comparer ces processus. Il faut cependant bien garder à l'esprit que ces simulations ne sont basées que sur des modèles, modèles que nous avons nous-mêmes définis. S'il fallait évaluer réellement un processus ou une proposition d'amélioration de processus, cela devrait se dérouler sans conteste avec la présence de personnes, étant donné qu'un processus n'est juste qu'une description tant qu'il n'est pas utilisé par des personnes [Wohlin 00]. Les études empiriques apparaissent donc comme essentielles à l'évaluation de processus grâce à leur approche systématique, disciplinée et quantifiable.

Parallèlement à cette idée d'«amélioration de processus», plusieurs rapports [SGI 94, Basili 99a] et articles scientifiques [Perry 00, Basili 96] débattent des problèmes, ainsi que des causes de ces problèmes, inhérents au développement de logiciels.

Le rapport du PITAC (*President's Information Technology Advisory Committee*) met en exergue la *fragilité* des infrastructures logicielles. Cette notion se réfère à la *non-fiabilité*, *au manque de sécurité*, *aux écarts de performance*, *aux erreurs* et *à la difficulté de mettre à jour* des infrastructures logicielles [Basili 99a]. En d'autres termes, trop de logiciels présentent des «*surprises*» [Basili 99a]. L'objectif, à terme, est de développer des techniques afin d'augmenter le nombre de logiciels «sans surprise» et de comprendre plus précisément à quel moment les logiciels risquent de dépasser le seuil de «surprise».

De manière similaire, Perry *et al.* énoncent que le génie logiciel «*ne connait pas les mécanismes fondamentaux qui définissent les coûts et les bénéfices des méthodes et outils logiciels*» [Perry 00]. Sans cette information, il n'est pas possible d'évaluer correctement les différentes méthodes et/ou de s'axer sur des réductions de coûts [Perry 00].

Perry *et al.* présentent les études empiriques comme un moyen clé pour l'obtention d'informations nécessaires à la prise de décision. Basili *et al.* ont énoncé un certain nombre d'actions à entreprendre dans la recherche informatique afin d'augmenter le nombre de logiciels «sans surprise» [Basili 99a]. Parmi ces actions, on y retrouve le développement d'une science empirique propre au domaine informatique. Celle-ci devrait permettre d'analyser et d'apprendre comment certaines organisations ont réussi à développer des logiciels «sans surprise». Avec ces analyses, il devrait être possible de transférer le savoir afin d'augmenter le nombre de logiciels «sans surprise».

Une autre action suggérée par Basili *et al.* est l'amélioration de la compréhension des éléments de base de la discipline informatique. Ces derniers représentent les fondements de la construction de n'importe quel logiciel [Basili 99a]. L'objectif serait d'augmenter les liens entre la recherche théorique d'un coté, notamment avec les méthodes formelles, les algorithmes, les systèmes d'exploitation, les systèmes de gestion de bases de données, les langages de programmation, et les analyses empiriques d'un autre coté.

Basili *et al.* fournissent une constatation résumant la situation de la recherche en informatique [Basili 99a]. L'ensemble des modèles pour supporter le raisonnement au niveau du domaine informatique est insuffisant. Il existe actuellement un manque de compréhension des limitations de ces modèles, ainsi qu'un nombre insuffisant d'analyses et d'expérimentations. Pour ce faire, la discipline de l'informatique a besoin de créer de nouvelles méthodes de recherche afin de les faire mûrir et de les faire évoluer. Il est également important d'étudier et de classer les développements passés qui ont réussi et échoué en fonction des paramètres qui limitent le progrès [Basili 99a].

1.2.2 Science et génie logiciel

Le génie logiciel est un domaine multidisciplinaire. Il traite d'aspects techniques tels que les bases de données et les systèmes d'exploitation, mais aussi d'aspects liés aux langages tels que la syntaxe et la sémantique, ou encore d'aspects sociaux. Le développement de logiciels est, quant à lui, fortement basé sur les capacités humaines. Il s'agit d'un processus très créatif, basé sur l'ingéniosité des personnes, rendant, de par sa définition, la production en chaîne totalement impossible. Le génie logiciel n'a cependant jamais été vraiment considéré comme une science. Pourtant, considérer le génie logiciel comme une science permettrait d'ouvrir la voie à l'utilisation de méthodes dites scientifiques, offrant ainsi de nouvelles techniques de recherche.

Avant d'effectuer des recherches dites scientifiques en génie logiciel, il serait toutefois bon de déterminer concrètement quelles sont les méthodes qui sont offertes, ainsi que leurs limitations et leurs cadres d'application.

Quatre types de recherche en génie logiciel sont résumés dans [Glass 94, Zelkowitz 98]. Ils furent cependant présentés initialement par [Basili 93]. Ces méthodes de recherche sont les suivantes :

- **La méthode scientifique**. Le monde est observé et un modèle est établi sur base de ces observations. Ce modèle est ensuite validé. Exemple : un modèle de simulation.
- **La méthode ingénieuriale**. Les solutions actuelles sont étudiées et des changements sont proposés. Ceux-ci sont ensuite évalués.
- **La méthode empirique**. Un modèle est proposé et évalué à travers des études empiriques, comme par exemple les études de cas ou les expériences.
- **La méthode analytique**. Une théorie formelle est proposée et est ensuite comparée avec les observations empiriques.

La méthode ingénieuriale ainsi que la méthode empirique peuvent être perçues comme des variations de la méthode scientifique [Basili 93].

Les méthodes empiriques ont été traditionnellement utilisées dans les sciences sociales et la psychologie, où il est impossible de définir des règles, comme on en trouve en physique. Il est important de remarquer que le génie logiciel, tout comme les sciences sociales et la philosophie, est fortement en rapport avec le comportement humain. Le génie logiciel concerne un «*système*» (objet formel mathématique) mais *dans son environnement* (humain et organisationnel). Il n'est donc pas étonnant qu'aucune règle ou loi, au sens des sciences exactes, ne puisse être établie en génie logiciel, à moins de se focaliser uniquement sur les aspects techniques.

En principe, rien n'empêche l'utilisation de méthodes analytiques, scientifiques et ingé-

nieuriales en génie logiciel. Les méthodes de recherche ne sont d'ailleurs pas orthogonales, il est tout à fait concevable de combiner une étude empirique avec une méthodologie ingénieuriale [Wohlin 00].

La raison principale pour laquelle le génie logiciel devrait utiliser les expériences est qu'il est nécessaire de comprendre et d'identifier les relations entre différentes variables. Un nombre incalculable d'idées préconçues existent en informatique. Beaucoup d'entre elles semblent logiques aux yeux des praticiens. Cependant, est-ce que ces idées préconçues sont pour autant fondées ? C'est dans cette optique que s'inscrit le génie logiciel empirique. Une meilleure compréhension permettra de changer et d'améliorer la manière de travailler.

1.2.3 Contexte du génie logiciel empirique

Le génie logiciel s'occupe du développement, de la maintenance et de la gestion de la qualité des systèmes informatiques d'une manière prévisible et efficace au niveau des coûts. Les systèmes informatiques constituent les bases de la société moderne, et beaucoup d'entres eux représentent les créations les plus complexes réalisées jusqu'à présent [Sjoberg 07]. Les recherches en génie logiciel étudient les phénomènes du monde réel et concernent : (1) le développement de nouvelles technologies ou la modification de technologies existantes (des modèles de processus, des méthodes, des techniques, des outils ou des langages) afin de fournir un support aux activités du génie logiciel. Ces recherches servent également à (2) évaluer et comparer l'effet de l'utilisation d'une technologie dans un environnement aux interactions complexes entre individus, équipes, organisations, etc [Sjoberg 07]. Afin d'étudier les phénomènes du monde réel, les recherches en génie logiciel se doivent d'utiliser des méthodes empiriques [Sjoberg 07].

Une approche empirique pour évaluer les technologies en génie logiciel a débuté à large échelle dans les années 70 avec les travaux de Victor R. Basili et son groupe à l'Université de Maryland [Basili 02, Boehm 05]. Les travaux de Basili sont présentés à travers [Basili 86][Basili 93][Basili 96][Basili 99b][Basili 00] et une synthèse de ses recherches est disponible dans [Boehm 05]. Dans ses articles, Basili discute le développement d'un corps de connaissances via l'établissement d'un modèle itératif : la prédiction, les tests d'hypothèses, l'observation et les analyses.

La recherche empirique cherche à explorer, décrire, prédire et expliquer les phénomènes naturels, sociaux ou cognitifs en utilisant des preuves basées sur l'observation ou l'expérimentation [Sjoberg 07]. Son but est de tenter d'apprendre quelque chose d'intéressant en comparant la théorie à la réalité et d'éventuellement améliorer cette théorie [Perry 00]. Ce type de recherche est important en génie logiciel car les résultats de telles recherches permettent de caractériser les problèmes techniques et d'évaluer de nouvelles techniques dans un contexte pertinent [Brilliant 99]. L'obtention de preuves et leurs interprétations s'effectuent via des expérimentations, des études de cas, des sondages ou via l'examen de documents et artefacts [Perry 00, Sjoberg 07], voir Section 1.3.

La plupart des méthodes empiriques requièrent plusieurs étapes communes aux processus d'acquisition de connaissances [Sjoberg 07]. Le chercheur doit spécifier le sujet de la recherche, concevoir l'étude, collecter les données, analyser et interpréter celles-ci. Ce processus est fort semblable à l'approche développée par Basili. Perry *et al.* pointent l'importance des interprétations des données. C'est par ces interprétations qu'il sera possible de guider et de changer la théorie sous observation [Perry 00].

L'utilisation de méthodes empiriques au niveau de la recherche en génie logiciel fait partie d'une vision plus large. Celle-ci vise à relever certains challenges [Sjoberg 07] :

- Faire de la recherche plus basée sur l'utilisation de méthodes empiriques
- Augmenter la qualité, et notamment la pertinence des études utilisant ces méthodes
- Synthétiser plus et mieux les constatations empiriques
- Construire et tester plus de théories

Les moyens permettant d'atteindre ces challenges sont [Sjoberg 07] :

- L'augmentation des compétences sur la manière d'appliquer et de combiner les méthodes empiriques
- Le renforcement des liens entre le monde académique et le monde industriel
- Le développement d'agendas communs de recherche axés sur les méthodes empiriques
- L'accès à plus de ressources pour la recherche empirique

Depuis l'apparition des travaux de Victor R. Basili, de plus en plus de recherches en génie logiciel se sont focalisées sur l'application de méthodes empiriques. Cette focalisation sur le génie logiciel empirique se reflète par la présence de journaux, conférences et autres colloques tels que «The Journal of Empirical SE»(EMSE, depuis 1996), «IEEE International Symposium on Software Metrics»(METRICS, depuis 1993), «Empirical Assessment & Evaluation in SE»(EASE, depuis 1997) et «IEEE International Symposium on Empirical SE»(ISESE, depuis 2002). Parallèlement, d'autres grandes conférences ont prêté également attention aux évaluations empiriques. C'est le cas notamment de l'«International Conference on Software Engineering» (ISCE) et de «Foundations of Software Engineering» (FSE). D'autres organismes sont en train de reconnaître la valeur des études empiriques. C'est le cas notamment des programmes de la National Science Foundation (NSF) avec le programme «Experimental and Integrative Activities» [Perry 00].

Suite à cet intérêt de plus en plus grandissant dans le domaine, bon nombre de chercheurs et praticiens s'affairent de nos jours à améliorer l'usage et l'efficacité des méthodes empiriques [Perry 00]. Cependant, la vision voulue par les recherches empiriques en génie logiciel est loin d'être atteinte. Les recherches empiriques actuelles présentent encore des faiblesses pour lesquelles plusieurs voies d'amélioration ont été proposées [Sjoberg 07]. La maturité du génie logiciel empirique sera discutée plus largement à travers la Section 1.5.

1.3 Stratégies empiriques

1.3.1 Types de stratégies empiriques

Il existe 2 paradigmes de recherche qui ont chacun une approche différente des études empiriques : le paradigme **qualitatif** et le paradigme **quantitatif**.

La **recherche qualitative** se préoccupe d'étudier les objets dans leur cadre naturel. Un chercheur utilisant l'aspect qualitatif tente d'interpréter un phénomène sur base des explications que les gens lui apportent [Denzin 94]. La recherche qualitative commence par le fait d'accepter qu'il existe différentes interprétations possibles. L'objectif est de découvrir ce que les sujets remarquent dans une étude, ainsi que de comprendre leur vision du problème. On entend par sujet, toute personne prenant part à une expérience, dans le but d'évaluer un objet.

La **recherche quantitative** se préoccupe surtout de l'aspect quantitatif des relations ou la comparaison de 2 ou plusieurs groupes [Creswell 94]. L'objectif ici est d'identifier une relation de cause à effet. La recherche quantitative est souvent réalisée à travers la mise en place d'expériences contrôlées ou la collecte d'informations via des cas d'étude. Les recherches quantitatives sont appropriées lorsqu'on cherche à tester les effets d'une activité. De plus, les données ressortant de telles recherches permettent d'effectuer des comparaisons ainsi que des tests statistiques.

Il n'est cependant pas interdit d'effectuer à la fois une étude qualitative et quantitative sur un même sujet. Cependant, chacune d'entre elles sera dirigée sur un aspect différent de la question. A titre d'exemple, une étude quantitative peut être réalisée dans le but de calculer le nombre d'erreurs qu'une nouvelle méthode d'inspection permet de supprimer. Une étude qualitative sur le même sujet s'attellerait, quant à elle, à analyser quelles seraient les sources de variations entre les différents groupes d'inspection. Les stratégies quantitatives sont plus destinées à calculer les effets d'une action alors que les stratégies qualitatives sont plus destinées à comprendre, à trouver le «*pourquoi*» des résultats ressortant des études opérées.

Il est possible de caractériser les recherches empiriques avec un autre facteur orthogonal, en fonction des auteurs [Zelkowitz 98] ou [Sjoberg 07].

Pour Sjøberg *et al.*, ce facteur orthogonal est représenté par deux types de recherches : les **recherches primaires** et les **recherches secondaires**.

Les **recherches primaires** impliquent la collection et l'analyse de données originales, l'utilisation de méthodes telles que des expériences, des études de cas, des sondages ou encore des recherches en action (voir définitions en Section 1.3.2) [Sjoberg 07].

Les **recherches secondaires** utilisent les données publiées préalablement dans des études afin de synthétiser la recherche dans le domaine. Ce type de recherche permet d'identifier les questions et les zones cruciales qui n'ont pas été traitées correctement dans le passé. De plus, ce genre de recherche permet de tirer des conclusions plus générales en parcourant toutes les observations réalisées pour une même thème [Sjoberg 07].

Pour Zelkowitz *et al.*, ce facteur orthogonal est représenté par trois types de méthodes : les **méthodes d'observation**, les **méthodes historiques** et les **méthodes contrôlées**.

Les **méthodes d'observation** collectent les données tels que les projets le permettent. Ce type de méthodes a relativement peu de contrôle sur l'étude empirique.

Les **méthodes historiques** collectent les données à partir de projets qui ont déjà été conduits. Ces données sont ensuite analysées.

Les **méthodes contrôlées** fournissent plusieurs instances d'une observation afin d'assurer la validité des résultats.

1.3.2 Résumé des différentes stratégies

Cette section donne un aperçu rapide des stratégies présentes en génie logiciel empirique.

Recherche primaire

Selon le but de la recherche, qu'il s'agisse de l'évaluation de techniques, d'outils ou de méthodes, ainsi que les conditions d'investigation, 3 types de stratégies peuvent être développés [Robson 93] :

• **Le sondage**. Un sondage est généralement réalisé en rétrospective, lorsque, par exemple, un outil ou une technique a été utilisé pendant une longue période [Pfleeger 95]. Les techniques généralement utilisées pour collecter les informations, qualitatives ou quantitatives, sont les interviews ainsi que les questionnaires.

• **L'étude de cas**. Les études de cas sont utilisées pour l'étude de projets ou d'activités. Elles sont généralement destinées à suivre un attribut particulier ou à établir des relations entre différents attributs. Une étude de cas consiste en une étude observatrice alors qu'une expérience est une étude contrôlée [Zelkowitz 98].

• **L'expérience**. Les expériences sont généralement réalisées dans des laboratoires, ce qui permet d'avoir plus de contrôle. Le but ici est de manipuler une ou plusieurs variables et de contrôler toutes les autres. Les effets de ces manipulations sont ainsi mesurés, et permettront des analyses statistiques par la suite. Une expérience est un processus formel, rigoureux et contrôlé. Elle peut être réalisée dans une situation *hors ligne*, par exemple dans un laboratoire sous des conditions contrôlées, où les événements sont organisés de manière à simuler leur apparition dans le monde réel. Une expérience peut également se dérouler *en ligne*, ce qui signifie que l'étude s'exécute sous des conditions normales [Babbie 90].

Les expériences sont généralement utilisées pour les raisons suivantes :

- Confirmer une théorie, *i.e.* tester les théories existantes.
- Confirmer les conventions collectives, *i.e.* tester les conventions des gens.
- Explorer les relations, *i.e.* tester qu'une relation tient la route.
- Evaluer la précision de modèles.
- Valider des mesures, *i.e.* s'assurer qu'une mesure calcule réellement ce qu'elle est supposée calculer.

Ces différentes stratégies peuvent être classées selon leur type de recherche, qu'il soit qualitatif ou quantitatif, voire les deux. Le Tableau 1.1 présente cette classification. Les sondages,

tout comme les études de cas, peuvent être classés dans les 2 catégories selon leur conception, selon les différentes informations enregistrées et s'il est possible de leur appliquer des méthodes statistiques. Les expériences, quant à elles, sont purement quantitatives étant donné qu'elles se basent sur les mesures des différentes variables. Ces variables fournissent des données sur lesquelles différentes méthodes statistiques pourront être appliquées.

Stratégies	Qualitative / Quantitative
Sondage	Les deux
Cas d'étude	Les deux
Expérience	Quantitative

Tab. 1.1 – Approches qualitatives vs. approches quantitatives

Ces trois types d'études correspondent aux types les plus couramment discutés dans les articles et livres de génie logiciel empirique, tels que [Zannier 06, Sjoberg 07, Wohlin 00]. Zannier *et al.* pointent cependant du doigt, suite à une étude réalisée sur le contenu d'articles scientifiques, le manque de consensus à propos des définitions des différents types de stratégies empiriques [Zannier 06]. Ceux-ci dénoncent le manque de cohérence dans les types d'études présentés dans les conférences d'ICSE, malgré les taxonomies publiées, telles que [Zelkowitz 98], voir Section 1.5.4.

D'autres types d'études existent, tels que :

• **La recherche en action**. Ce type de recherche se focalise essentiellement sur le fait de combiner théorie et pratique [Avison 99, Greenwood 06]. Son objectif est de fournir une valeur pratique à l'organisation cliente tout en contribuant simultanément à acquérir de la nouvelle connaissance théorique [Sjoberg 07].

Recherche secondaire

• **La synthèse de recherche**. Egalement utilisée à la place de «*revue systématique*» ou de «*revue systématique de la littérature*» et vice versa, ce type de recherche permet de minimiser les chances de tirer des conclusions incorrectes [Sjoberg 07].

• **La revue systématique**. Ce type de recherche synthétise les recherches primaires via des méta-analyses qui utilisent des méthodes pour combiner les tailles des effets. Cependant, ce type de méta-analyses ne peut avoir lieu que si les recherches primaires sont suffisamment similaires. Or, il semblerait que les études primaires soient trop hétérogènes actuellement pour permettre une synthèse via des tests statistiques [Sjoberg 07].

Par soucis de concision, la taxonomie présentée par Zelkowitz et Wallace, [Zelkowitz 98], ne sera pas présentée. De plus, les types de stratégies fournis dans cette taxonomie sont relativement peu utilisés en comparaison de ceux présentés par Sjøberg *et al.* dans [Sjoberg 07]. De plus amples informations peuvent également être trouvées dans [Wohlin 00] ou [Zannier 06].

1.3.3 Comparaison des différentes stratégies empiriques

Le choix des stratégies de recherche est fortement limité par les pré requis d'une investigation. Une comparaison de ces stratégies peut être basée sur un certain nombre de facteurs :

• **Le contrôle de l'exécution.** Ce facteur décrit le niveau de contrôle qu'un expérimentateur a lorsqu'il réalise son étude. L'exemple donné dans [Wohlin 00] est celui d'une étude de cas dans une entreprise. Si cette dernière décide de cesser ses activités par faute de moyens, l'expérimentateur n'est plus capable de poursuivre son étude. L'opposé de cette situation est une *expérience* où l'expérimentateur a le contrôle total de l'exécution.

• **Le contrôle de la mesure.** Ce facteur représente le degré que détient l'expérimentateur de décider quelles informations doivent être collectées durant l'exécution de l'étude. L'exemple que présente C. Wohlin est celui de la collecte de données concernant la volatilité d'une exigence. Une étude de cas ou une expérience permettra d'inclure ce genre de mesures contrairement au sondage qui ne pourra uniquement recueillir que les opinions des gens à propos de la volatilité d'une exigence.

• Le troisième facteur, quant à lui, est fortement lié aux deux précédents. Il s'agit du **coût de l'investigation**. La stratégie qui présente le coût le moins élevé est celle du sondage, étant donné qu'elle ne nécessite que très peu de ressources. La différence entre une étude de cas et une expérience est que, dans une étude de cas, ce qui en ressort est quelque chose sous la forme d'un produit qui peut être vendu. Il s'agit d'une investigation «*en ligne*». Dans le cas d'une expérience, ce qui en ressort est plutôt sous la forme d'une expérience personnelle qui n'est pas profitable en tant que produit. Il s'agit dans ce cas d'une investigation «*hors ligne*».

• Le quatrième facteur à prendre en considération est la **facilité de réplication** de l'investigation. Le but de la réplication est de montrer que les résultats trouvés sont également applicables à une plus grande population. Une réplication sera définie comme une «*vraie*» réplication s'il est possible de répliquer aussi bien les résultats que la conception elle-même de l'investigation.

Facteur	Sondage	Etude de cas	Expérience
Contrôle de l'exécution	Non	Non	Oui
Contrôle des mesures	Non	Oui	Non
Coût de l'investigation	Faible	Moyen	Elevé
Facilité de réplication	Elevée	Faible	Elevée

Tab. 1.2 – Facteurs des stratégies de recherche

1.4 Mesure

La mesure (méthode) des logiciels est cruciale pour contrôler les projets, produits et processus comme Wohlin le mentionne [Wohlin 00]. DeMarco énonce par ailleurs que : «*On ne peut contrôler ce qu'on ne peut mesurer*» [DeMarco 82]. Cette mesure est la partie centrale de n'importe quelle discipline ingénieuriale [Habra 08] et notamment des études empiriques. Ces dernières évaluent les effets de certains «*inputs*» sur le ou les objets étudiés. Afin de contrôler l'étude d'une part, et de décrire ce qui cause les effets sur l'«*output*» d'autre part, il est important de mesurer aussi bien les «*inputs*» que l'«*output*». Sans mesure, aucun contrôle ne pourrait être établi et l'étude empirique ne pourrait être dirigée.

«*La mesure (méthode) est une correspondance entre le monde empirique et le monde formel et relationnel. Par conséquent, une mesure (résultat) est le nombre ou symbole assigné à une entité via cette correspondance, dans le but de caractériser un attribut.*» [Fenton 96] Le principe est d'étudier ces mesures et de faire des jugements sur celles-ci, au lieu de faire des jugements directement sur les entités réelles.

Le but de cette section est de fournir une introduction à la «*théorie de la mesure*». La Section 1.4.1 présente les concepts de base de cette théorie tandis que la Section 1.4.2 présente brièvement les enjeux de la description du monde empirique.

1.4.1 Concepts de base

Plusieurs alternatives ont été proposées dans le domaine informatique afin de mesurer les mêmes attributs. Ce problème est dû à l'absence de consensus sur un cadre de références définissant comment analyser et choisir les mesures appropriées. De la même manière, il n'existe pas à l'heure actuelle de consensus sur les caractéristiques des attributs à mesurer [Habra 08]. Récemment, Habra *et al.* ont proposé un cadre de références pour le cycle de vie des mesures de logiciels. Ce cadre de références contient également un glossaire présentant un vocabulaire unifié pour les concepts clés des méthodes de mesure [Habra 08]. Les définitions des concepts présentés à travers cette section sont issues à la fois de [Wohlin 00] et de [Habra 08]. Cependant, seul les concepts de base seront abordés. De plus amples informations sont fournies dans [Habra 08].

Entité

Le terme «*entité*» se réfère à n'importe quel objet distinguable du monde empirique pour lequel une mesure peut être appliquée [Habra 08]. Fenton distingue trois types d'entités : les **produits**, les **processus** et les **ressources** [Fenton 96].

- **Processus**. Le processus définit quelles sont les activités nécessaires à la production du logiciel.
- **Produit**. Le produit est l'artefact, le délivrable ou le document qui résulte d'un processus.
- **Ressources**. Les ressources sont les objets, tels que les personnes, le matériel, ou le logiciel, nécessaires à un processus.

Des exemples d'entités mesurables en informatique sont un morceau de code, une base de données, une tâche de programmation, un processus de maintenance, etc.

«*Une entité est caractérisée par un ensemble d'attributs, chacun correspondant à une propriété simple et observable*» [Habra 08].

Attribut

Le terme «*attribut*» se réfère à la propriété d'une entité qui peut être déterminée de manière quantitative, c'est-à-dire, pour laquelle une magnitude peut être assignée [Habra 08]. Dans le vocabulaire de la métrologie [ISO VIM 93], le terme «*quantité mesurable*» est employé, ainsi que «*quantité*». Plusieurs modèles d'attributs de qualité, tels que [ISO/IEC 01] [ISO/IEC TR 03a] [ISO/IEC TR 03b] [ISO/IEC TR 04], utilisent les termes «*caractéristique de qualité*» et «*sous-caractéristique*». Un certain nombre d'approches qualitatives, plus particulièrement les modèles d'informations cités précédemment, définissent plusieurs caractéristiques de qualité sur base d'autres sous-caractéristiques, et ce, de manière hiérarchique [Habra 08]. Les concepts d'«*attributs de base*» et d'«*attributs dérivés*» sont également mentionnés dans le vocabulaire présenté par [Habra 08, Wohlin 00].

Attribut de base et attribut dérivé

Un «*attribut de base*» est une simple propriété définie par une convention, sans aucune référence à d'autres attributs, et est éventuellement utilisé pour définir d'autres attributs [Habra 08]. Le terme «*caractéristique de base*» est également utilisé. Dans le vocabulaire de la métrologie [ISO VIM 93], le terme «*quantité de base*» sera plutôt utilisé.

Un *attribut dérivé* est une propriété dans un système d'attributs, définie comme une fonction d'attributs de base [Habra 08]. Le terme «*caractéristique dérivée*» est également utilisé. Dans le vocabulaire de la métrologie, le terme «*quantité dérivée*» est employé.

Attribut interne et attribut externe

Un «*attribut interne*» est un attribut qui peut être mesuré uniquement au niveau de l'entité en elle-même. Un «*attribut externe*» est un attribut qui ne peut uniquement être mesuré qu'en fonction des relations que l'entité possède avec son environnement [Habra 08, Wohlin 00]. Les modèles de qualité ISO 9126 distinguent ces deux types d'attributs de manière différente. Leurs définitions ne s'appliquent que pour des attributs de produits logiciels exécutables. Les définitions de [Habra 08] et [Wohlin 00] sont plus génériques et permettent d'adresser différents produits logiciels, incluant ceux non-exécutables comme les spécifications d'exigences fonctionnelles ou les produits de conception [Habra 08].

Mesure d'un attribut

«*La mesure (méthode) d'un attribut d'une entité est la caractérisation de cet attribut en termes de nombres ou symboles*» [Habra 08]. «*Une mesure est une correspondance entre un attribut d'une entité et une valeur de mesure, généralement une valeur numérique*» [Wohlin 00]. C'est pourquoi, une des caractéristiques essentielles de base de la mesure (méthode) est qu'elle doit préserver les observations empiriques d'un attribut [Fenton 94].

A coté de ces définitions générales, la littérature relative aux mesures fait généralement une distinction à propos de deux points de vue complémentaires, l'opérationnel et le théorique. Le point de vue opérationnel se focalise sur l'acte de «*mapping*» (comment faire correspondre), alors que le point de vue théorique se focalise sur les propriétés du «*mapping*» [Habra 08].

Il faut cependant remarquer que le mot «*mesure*» peut avoir plusieurs significations dans le langage courant. La mesure peut, par exemple, désigner le «*mapping*» entre des entités et des valeurs ou désigner les actions qui appliquent ce «*mapping*». Le terme «*mesure*» est également employé pour désigner le résultat de l'application de ce «*mapping*»

[Habra 08]. Il est important d'utiliser les termes adéquats afin d'éviter toute confusion. Ainsi les termes «*résultat de la mesure*», «*appliquer la mesure*» et «*mesurer*» seront préférés au terme «*mesure*», terme présentant une ambiguïté [Habra 08].

Valeur et résultat de la mesure

Le terme «*valeur*» se rapporte à la magnitude assignée à un attribut d'une entité représentée par un nombre et une référence [ISO VIM 04]. Une référence est généralement une unité mais il pourrait également s'agir d'une procédure de mesure jouant le rôle de référence [Habra 08]. Le «*résultat de la mesure*» d'un attribut d'une entité est la *valeur* assignée à cet attribut via un «*mapping*».

Type d'échelle

Une «*échelle*» est un ensemble structuré de valeurs associé à un attribut, utilisé pour comparer différentes entités selon cet attribut [Habra 08]. Ce type de structure peut posséder le concept d'*ordre* parmi les attributs mesurés.

Un «*type d'échelle*» correspond à une famille d'échelles pour lesquelles il est possible d'établir des correspondances en appliquant des transformations homomorphiques (*i.e.* transformer une échelle de longueur utilisant les centimètres en une échelle de longueur utilisant les pouces) [Habra 08].

Pour rappel, la mesure est perçue comme une correspondance entre le monde empirique et le monde numérique [Habra 08]. La description de ce monde numérique est relié au concept de *type d'échelle* présenté dans la littérature classique de la mesure [Fenton 96]. Il est important de remarquer que le type d'échelle définit le type d'opérations qu'il est possible d'effectuer. Un type d'échelle correspond à une catégorie de structures mathématiques [Habra 08]. Les *types d'échelle* généralement utilisés sont [Fenton 94, Fenton 96, Briand 96b] : *nominale*, *ordinale*, *d'intervalle*, *de rapport* et *absolue*.

- **Type d'échelle nominale.** Le type d'échelle nominale est le moins performant de tous les types d'échelle. Sa structure mathématique inclut un ensemble fini de valeurs sans aucune opération, étant donné qu'aucune magnitude n'est associée aux valeurs. Une correspondance est faite entre un attribut d'une entité et un nom ou symbole. Cela peut être vu comme une sorte de classification des entités en fonction de l'attribut en question. Aucun ordre n'est possible sur des éléments d'un tel ensemble. Les seules comparaisons possibles sont les opérateurs «=» et «≠» [Habra 08].
  La «*classification*» et le «*labeling*» sont des exemples d'échelles nominales.

- **Type d'échelle ordinale.** L'échelle ordinale est plus puissante que l'échelle nominale étant donné qu'elle range les entités selon un critère d'ordre. Ce critère d'ordre est la seule relation d'ordre de la structure mathématique associée [Habra 08]. A titre d'exemple, on trouve les critères d'ordre comme «*plus grand que*», «*meilleur que*». Les «*grades*», la «*complexité*» des logiciels sont des exemples d'échelles ordinales.

- **Type d'échelle d'intervalle.** L'échelle d'intervalle est plutôt utilisée lorsque deux mesures sont significatives, mais pas la valeur elle-même. La structure mathématique de ce type d'échelle utilise également un critère d'ordre comme le type d'échelle ordinale mais avec en plus une notion de «*distance relative*» entre deux entités, ce qui rend ce type d'échelle plus puissant que l'échelle ordinale. Par «*plus puissant*», il faut entendre que «*plus d'informations*», «*plus d'opérations*» et «*plus de propriétés*» sont

disponibles. Des exemples typiques de ce type d'échelle sont les mesures de température en degrés Celsius ou Fahrenheit. Il faut cependant remarquer que ce genre de mesures sont peu communes en génie logiciel.

- **Type d'échelle de rapport.** La structure mathématique du type d'échelle de rapport est identique à celle du type d'échelle d'intervalle mais inclut également une valeur zéro. La longueur, la température mesurée en Kelvin et la durée d'une phase de développement sont des exemples de type d'échelle de rapport.

- **Type d'échelle absolue.** Un type d'échelle absolue est comme un type d'échelle de rapport, excepté qu'une échelle absolue est une échelle unique. Elle ne permet aucune transformation vers une autre échelle structurellement équivalente.

Il faut également remarquer que ces échelles de mesure sont liées au type de recherche. Selon [Kachigan 86], l'utilisation d'échelles nominales et ordinales se fera dans des recherches de type qualitatif, alors que les échelles d'intervalle et de rapport seront plus utilisées dans des recherches de type quantitatif.

Mesures objectives et subjectives

La mesure (méthode) d'un attribut nécessite parfois de prendre en compte le point de vue selon lequel elle a été calculée. Il est possible de répartir ces mesures en deux catégories distinctes :

- **Mesure objective.** Une mesure objective est une mesure qui n'est le fruit d'aucun jugement. La seule dépendance que possède un tel type de mesure est celle qu'il détient avec l'objet mesuré. Une mesure objective peut être mesurée plusieurs fois (répétitive) et par plusieurs personnes (réplicable), la même valeur de mesure sera obtenue, en considérant l'erreur inhérente à la mesure. Les «*LOC*» (*Lines Of Code*) sont un exemple de mesure objective.

- **Mesure subjective.** Une mesure subjective dépend du mesureur. Celui qui mesure apporte un certain jugement à la valeur mesurée. Une valeur mesurée subjectivement plusieurs fois sur le même objet peut résulter en des valeurs différentes. Ce type de mesure est généralement un type d'échelle nominale ou ordinale. La «*compétence*» ou l'«*utilisabilité*» sont des exemples de mesures subjectives.

Mesures directes et indirectes

Le calcul des mesures d'un attribut ne peut parfois pas se faire de manière directe. Il est parfois nécessaire de dériver certaines mesures à partir d'autres mesures.

- **Mesure directe.** Une mesure directe d'un attribut est directement mesurable sans utiliser aucune mesure d'autres attributs. Les «*LOC*» (*Lines Of Codes*) et le nombre de défauts trouvés lors de tests sont des exemples de mesures directes.

- **Mesure indirecte.** Une mesure indirecte implique l'utilisation d'autres mesures d'attributs. Elle est calculée à partir de ces autres mesures. La densité de défauts (nombre de défauts / le nombre de lignes de code) ainsi que la productivité des programmeurs (le nombre de lignes de code / l'effort des programmeurs) représentent ce type de mesure.

1.4.2 Description du monde empirique

L'attribut à mesurer devrait être défini de façon claire et précise de telle sorte qu'il serait caractérisé de manière suffisamment claire et non-ambigüe. Cette caractérisation devrait être reliée à un *modèle d'attribut* [Habra 08].

Dans le domaine informatique, il n'est pas rare de rencontrer des approches de mesure où l'attribut et/ou l'entité à mesurer sont peu clairs, voire même pas du tout spécifiés. La définition d'un attribut délimite la description de cet attribut en spécifiant ce qu'il signifie et ce qu'il ne signifie pas. Dans le domaine informatique, le manque de consensus est courant à propos des définitions d'attributs, et, par conséquent, à propos des méthodes et procédures de mesure [Habra 08].

La mesure s'occupe également de trouver des consensus largement documentés à propos de ce qui doit être inclus et ce qui doit être exclus. Il ne s'agit pas uniquement d'une agrégation de connaissances de praticiens, mais aussi d'un accord général sur un ensemble de définitions et une échelle pour les mesurer [Habra 08].

En résumé, la description du monde empirique implique de : (1) déterminer l'entité à considérer, (2) déterminer l'attribut à mesurer, et (3) construire un modèle adéquat pour caractériser cet attribut. Il existe actuellement des modèles pour les entités et les attributs, ainsi que des consensus sur ces modèles. Le véritable problème est celui de la caractérisation d'un attribut, pour lequel une méthode de mesure doit être établie [Habra 08].

1.5 Maturité du génie logiciel empirique

L'utilisation de méthodes empiriques en génie logiciel ne cesse d'augmenter depuis les premiers travaux de Victor R. Basili dans le domaine [Sjoberg 07]. Ces 10–20 dernières années, la maturité du génie logiciel empirique a considérablement augmenté [Perry 00]. Cependant, plusieurs articles répertorient certains des problèmes les plus récurrents et les plus critiques actuellement, témoignant des progrès qui doivent encore être réalisés dans le domaine [Basili 96][Perry 00][Zannier 06][Sjoberg 07][Basili 99a].

L'objectif de cette section est de présenter les problèmes principaux qui apparaissent actuellement en génie logiciel empirique. Cette section s'articule de la manière suivante. La Section 1.5.1 débat de la **quantité** insuffisante d'études empiriques. La Section 1.5.2 présente les problèmes liés à la **qualité** des études empiriques et les futurs chemins à suivre afin d'améliorer cette qualité. La Section 1.5.3 énonce les problèmes concernant la **pertinence** des études actuellement réalisées. Finalement, la Section 1.5.4 présente rapidement plusieurs constatations qui ont été réalisées par différents auteurs d'articles scientifiques en génie logiciel empirique.

1.5.1 Nombre d'études empiriques

Le nombre d'études devant être réalisé en génie logiciel empirique est une question à laquelle il n'y a pas de réponse exacte [Sjoberg 07]. Cependant, plusieurs auteurs s'accordent à dire que le nombre d'études empiriques est insuffisant [Fenton 94, Tichy 95, Basili 96, Zelkowitz 98]. Certaines constatations peuvent être réalisées concernant la situation actuelle des études en génie logiciel.

A partir de 5453 articles scientifiques publiés dans 12 journaux et conférences scientifiques importants en génie logiciel entre 1993 et 2002, Sjøberg *et al.* ont identifié 113 expériences contrôlées, publiées dans 103 (1,9%) articles. Glass *et al.* et Zelkowitz *et al.* ont répertorié approximativement 3% d'études contrôlées [Glass 02][Zelkowitz 98].

Concernant les sondages, seule la revue de littérature de Glass *et al.* semble être pertinente selon Sjøberg *et al.*. Cette revue de littérature classe 1,6% de leurs 369 articles comme des sondages «descriptifs/exploratoires».

Les études de cas identifiées par Glass *et al.* représentent 2,2% (8 *sur* 369) des articles. Zelkowitz *et al.* ont trouvé 10,3% (58 *sur* 612)[Zelkowitz 98]. Des chiffres si élevés pour les études de cas peuvent s'expliquer par la difficulté de définir et d'identifier ce qu'est une étude de cas réellement.

Concernant les recherches secondaires, les revues de littérature et les méta-analyses, Glass *et al.* ont identifié 1,1% des articles étudiés et Zelkowitz *et al.* ont identifié 3% de leurs articles étudiés. Tichy *et al.* ont identifié, quant à eux, une proportion de 17% (15 *sur* 87 articles de l'IEEE TSE). Zelkowitz *et al.* classent deux tiers de ces publications comme étant des «évaluations expérimentales».

Les différences de proportions trouvées dans les différentes revues de littérature peuvent s'expliquer par le nombre de sources, les différences de définition, les années de publications etc. [Sjoberg 07]. Néanmoins, ces chiffres montrent qu'en moyenne, 20% des articles publiés sont des études empiriques. Ceci est loin d'être suffisant lorsqu'on considère le nombre total d'articles publiés sur la période considérée, pour les conférences et journaux considérés [Sjoberg 07]. Un pourcentage si faible ne permet pas d'atteindre un nombre suffisant d'articles de qualité traitant de sujets les plus importants dans le monde industriel [Sjoberg 07].

1.5.2 Qualité des études empiriques

Afin de pouvoir prendre des décisions basées sur des études empiriques, il est nécessaire de pouvoir faire confiance à ces études, c'est-à-dire, des études avec une grande validité. Les paragraphes suivants s'inspirent fortement de l'article de Sjøberg *et al.* [Sjoberg 07]

Le but des études empiriques n'est pas de maximiser leur qualité, mais de trouver un niveau de qualité approprié. Ce niveau de qualité peut être établi notamment sur base de l'importance de ne pas tirer des conclusions erronées, et sur le coût de l'amélioration de la qualité. Un mauvais niveau de qualité peut avoir de lourdes conséquences pour la communauté. Ces conséquences sont : (1) la diffusion de résultats incorrects, (2) la diminution de la crédibilité des études empiriques auprès des gens, (3) la difficulté d'introduire une culture de qualité parmi les personnes utilisant des méthodes empiriques [Sjoberg 07].

Le niveau de qualité de ces études reste cependant difficile à déterminer. Ce niveau de qualité est un concept relativement subjectif. Différents chercheurs peuvent interpréter différemment le niveau de qualité d'une même étude. De plus, de jeunes chercheurs peuvent éprouver une certaine difficulté à atteindre un haut niveau de qualité, ce qui pourrait les décourager dans l'élaboration d'études scientifiques suite au nombre d'articles refusés. Il est peut-être nécessaire de tolérer certains articles de faible qualité afin d'éduquer les jeunes chercheurs [Sjoberg 07].

Un autre point important lié à la qualité est la «*validité de la construction*» (voir Section 2.3.7). D'un coté, il est important de mesurer quelque chose afin de le comprendre. D'un autre coté, il est important de comprendre ce *quelque chose* afin de le mesurer [Sjoberg 07]. Plusieurs études donnent l'impression de mesurer un phénomène qui est mal compris, ce qui entraîne une validité de la construction assez faible.

La portée d'une expérience doit également être prise en compte afin de tirer des conclusions adéquates. Il n'est pas rare de trouver des expériences utilisant des étudiants comme sujets. Cependant, ces expériences ne précisent pas le type de professionnels qu'elles visent. Cela peut mener à certains problèmes dont notamment la publication de résultats conflictuels avec d'autres articles [Sjoberg 07].

Afin d'éviter ces problèmes, il serait plus préférable de **répliquer** les études que d'autres ont réalisées. Basili énonce également qu'une même expérience réalisée deux fois peut mener à des résultats différents et que, par conséquent, la réplication d'études empiriques est très importante [Basili 86]. Trop d'études sont menées de manière isolée et ne sont pas répliquées, par le même expérimentateur ou par d'autres [Basili 00].

1.5.3 Pertinence des études empiriques

Le terme «*pertinence*» peut s'interpréter de différentes manières. Pour Benbasat *et al.*, la pertinence se rapporte à deux concepts : le *thème de l'étude* et les *implications des résultats* [Benbasat 99]. Höfer et Tichy ont établi que le spectre des sujets étudiés empiriquement jusqu'à présent en informatique était assez faible et que certains sujets importants n'avaient été le fruit d'aucune étude [Höfer 07]. Sjøberg *et al.* fournissent un aperçu des différents sujets étudiés lors d'expériences en génie logiciel [Sjoberg 05]. Généralement, au plus les conditions d'une expérience sont similaires aux conditions de projets réels, au plus l'expérience est considérée comme *pertinente* [Sjoberg 07].

Fenton *et al.* énoncent que la recherche empirique devrait impliquer des projets réels avec des sujets réels, le tout avec une rigueur suffisante pour assurer que les résultats obtenus reflètent au mieux le concept sous étude [Fenton 94].

La méthode classique pour identifier des relations de cause à effet est de réaliser des expériences [Wohlin 00, Sjoberg 07]. La question est de savoir si les résultats obtenus à partir de ces expériences sont suffisamment pertinents pour le secteur industriel du logiciel. Cela peut s'analyser selon quatre classes d'archétype : les «*acteurs*», la «*technologie*», l'«*activité*» et le «*système informatique*» [Sjoberg 07].

La plupart des expériences en génie logiciel utilisent des individus ou des équipes comme **acteurs**. Cependant, 90% des sujets prenant part aux expériences sont des étudiants [Sjoberg 05]. Or, certaines expériences ont démontré que certains aspects en génie logiciel ne se manifestaient uniquement qu'avec un nombre élevé de sujets [Arisholm 04, Arisholm 07]. L'utilisation d'un faible nombre d'étudiants et la pertinence de leurs résultats vis à vis du monde industriel peuvent ainsi être remis en question [Sjoberg 07]. Tichy *et al.* proposent d'utiliser les étudiants afin de tester le design et les hypothèses de l'expérience ou de les utiliser uniquement dans des buts éducatifs [Tichy 00]. L'important, c'est que la compétence et le niveau d'expertise des sujets, relatifs à la technologie étudiée soient mentionnés explicitement. Ceci permettra d'indiquer la population à laquelle les résultats s'adressent [Sjoberg 07].

La pertinence de la **technologie** utilisée est reliée au problème du thème de l'expérience. Cependant, il est relativement peu fait mention d'impacts de la technologie sur les résultats d'expériences. Toutefois, les configurations des locaux utilisés lors d'expériences sont souvent artificielles et les outils utilisés ne correspondent pas à ceux utilisés par des professionnels. Cela peut constituer des menaces à la validité des résultats [Wohlin 00, Sjoberg 07].

Les **activités** sont constituées de tâches. Celles-ci peuvent prendre des mois ou plusieurs heures dans le monde industriel. Cependant, les tâches typiques lors d'expériences en génie logiciel sont bien plus petites que les véritables tâches du monde industriel. Seulement la moitié des articles analysés dans [Sjoberg 05] ont conçu des tâches et utilisé des systèmes qui n'étaient pas représentatifs des tâches et des systèmes du monde industriel. Par «*représentatif*», Sjøberg *et al.* entendent *taille/durée*, *complexité*, *domaine d'application* des logiciels. De plus, ces tâches expérimentales ont des limites de temps. Un sondage a montré que le temps médian des expériences était d'une heure, ce qui est insuffisant par rapport aux tâches de professionnels.

La plupart des **logiciels** utilisés lors d'expériences sont soit construits en fonction des objectifs de l'expérience, soit des projets d'étudiants [Sjoberg 07]. Une étude a rapporté que seulement 14% des expériences utilisaient des logiciels commerciaux [Sjoberg 05], ce qui est nettement insuffisant. Plusieurs articles prétendent qu'utiliser des systèmes non commerciaux ne constituent pas une menace à la validité à partir du moment où l'expérience s'est déroulée dans un contexte industriel [Sjoberg 07].

La communauté semble être d'accord pour affirmer que la plupart des expériences ne reflètent pas les situations industrielles. Cependant, l'objectif est de définir ce qu'on entend pas une «*situation industrielle*». Vu la diversité des technologies, des tâches et des systèmes, le choix d'un élément représentatif parmi ces catégories sera assez complexe. Sjøberg *et al.* proposent de définir des taxonomies et d'échantillonner ensuite la population en fonction des catégories de ces taxonomies [Sjoberg 07].

1.5.4 Autres constatations sur les études empiriques

La première constatation importante provient de Perry *et al.* qui dénoncent le manque d'hypothèses dans les études empiriques et invitent à tirer des conclusions plus crédibles sur base de ces études [Perry 00]. Ces études ne définissent aucune question et servent des finalités mal définies [Perry 00]. Une conséquence de cette absence d'hypothèses est le manque de conclusions ou la présence de conclusions fausses [Perry 00]. L'important avec une étude empirique est de pouvoir tirer des conclusions afin de pouvoir relier des observations avec la théorie et la pratique [Perry 00]. L'important n'est pas d'avoir une étude parfaitement décrite, mais plutôt d'avoir une étude et des conclusions tirées de manière crédible sur base de cette étude [Perry 00]. Pour cela, il est important de comprendre et expliquer les limites de l'étude. De la même manière, il est important de connaître les conclusions qu'il est possible de tirer avec l'étude empirique réalisée [Perry 00].

Dans sa revue de littérature, Zannier *et al.* font deux constatations concernant la définition du type d'étude utilisé [Zannier 06]. La première constatation concerne le manque de définitions de la part des auteurs concernant le type d'étude qu'ils ont réalisé. Au moins la moitié des études (18 *sur* 44) ne contiennent aucune explication du type d'étude réalisé. La seconde constatation concerne le nombre d'articles scientifiques avec une dénomination erronée du type d'étude. Ce manque de consensus au niveau des dénominations indiquent que la communauté du génie logiciel empirique ne s'entend actuellement pas sur les définitions des différents types d'étude [Zannier 06]. Ce manque de cohérence au niveau des définitions apparait malgré la publication de plusieurs taxonomies telles que [Zelkowitz 98].

1.6 Résumé

Le génie logiciel (GL) traditionnel a montré ses limites avec la présence de problèmes récurrents lors du développement de logiciels.

La présence d'études empiriques dans ce domaine de recherche est apparue comme un moyen clé pour obtenir l'information nécessaire à la prise de décision. Victor R. Basili lance, dans les années 1970, les premières recherches empiriques à large échelle en GL. Depuis lors, de plus en plus de recherches en GL se sont focalisées sur l'application de méthodes empiriques. Parmi les méthodes les plus couramment utilisées, on retrouve les sondages, les études de cas et les expériences. Ces types d'études peuvent être qualitatifs et/ou quantitatifs. Afin de quantifier, ces études se empiriques se basent la théorie de la mesure.

Cependant, la vision voulue par les recherches empiriques en GL est loin d'être atteinte. La maturité actuelle de ce type de recherche indique que certaines faiblesses doivent être traitées. Parmi les problèmes les plus importants, la communauté a identifié les points suivants : la *qualité*, la *quantité*, mais aussi la *pertinence* des études empiriques. Il est également fréquent de se heurter à un manque de consensus dans les diverses disciplines abordées lors d'une étude empirique.

2 Processus d'une expérience

Sommaire

Avant-propos

Conduire une expérience nécessite une certaine rigueur. Il est par conséquent important d'avoir recours à un processus. Ce chapitre vise à introduire, dans un premier temps, les concepts de base ainsi que le processus d'une expérience. Il vise, dans un second temps, à détailler les différentes étapes qui composent le processus d'une expérience.

Ce chapitre s'inspire fortement de l'ouvrage intitulé ***«Experimentation In Software Engineering : An Introduction»*** *de* ***Claes Wohlin et al.*** [Wohlin 00].

2.1 Introduction

La conduite d'une expérience requiert une certaine rigueur. Celle-ci est nécessaire durant les étapes de *préparation* et d'*exécution* de l'expérience, ainsi que durant l'étape d'*analyse des données*. Le principal avantage d'une expérience est qu'elle procure un certain contrôle sur les sujets, sur les objets d'étude ainsi que sur les matériaux. Ce contrôle permet de tirer par la suite des conclusions plus générales. Cette stratégie empirique offre également, comme avantage, la possibilité de réaliser des tests statistiques sur base d'hypothèses formulées préalablement. Il est important d'avoir recours à un processus afin d'exploiter au mieux ces avantages. Les principes de base derrière une expérience sont illustrés en Figure 2.1.

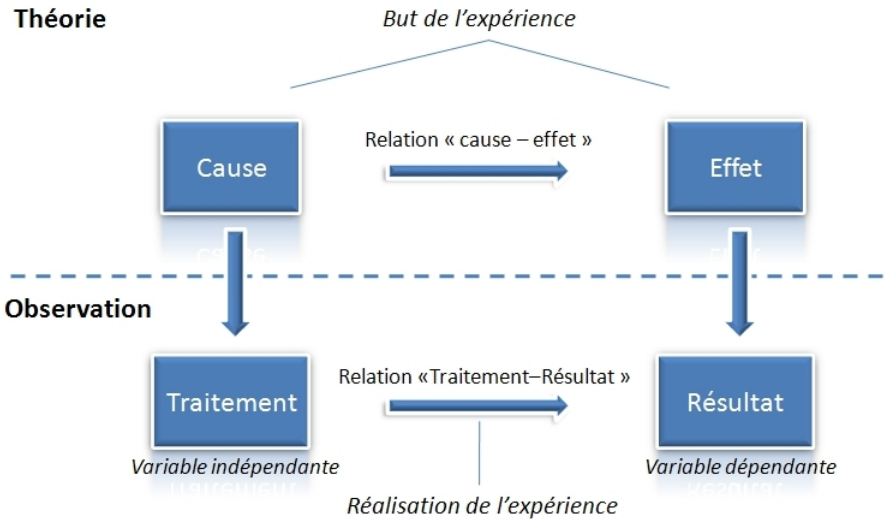


FIG. 2.1 – Principes d'une expérience (adapté de [Trochim 99])

Le point de départ est l'idée qu'un expérimentateur possède à propos de l'existence d'une *relation «cause–effet»*. Pour cela, soit il existe une théorie, soit une hypothèse doit être formulée. Cette hypothèse signifie que l'expérimentateur a une idée qu'il peut énoncer formellement à travers une hypothèse.

Afin d'évaluer la théorie ou son hypothèse, l'expérimentateur conçoit sa propre expérience. Dans cette expérience, l'expérimentateur a le contrôle d'un certain nombre de «*traitements*». Une fois l'expérience réalisée, il est possible d'observer le «*résultat*» et ainsi de tester la *relation «traitement–résultat»*. Si l'expérience a été conçue de manière rigoureuse, il sera possible de tirer un certain nombre de conclusions concernant la *relation «cause–effet»* pour laquelle une ou des hypothèses ont été formulées.

2.1.1 Variables, traitements, objets et sujets

Cette section introduit quelques définitions afin d'acquérir un vocabulaire propre aux expériences. L'un des objectifs d'une expérience est d'analyser l'impact des variables d'entrée sur la ou les variables de sortie. Dans une expérience, il existe deux types de variables : les **variables indépendantes**, et les **variables dépendantes** (voir Figure 2.2).

Les **variables dépendantes** correspondent aux variables sur lesquelles les effets de changements au niveau des variables indépendantes sont étudiés. Il n'y a généralement qu'une seule variable dépendante dans une expérience. Toutes les variables contrôlées et manipulées sont appelées **variables indépendantes**. En d'autres termes, les variables indépendantes sont des variables qui *influencent*. Et les variables dépendantes sont des variables *influencées*.

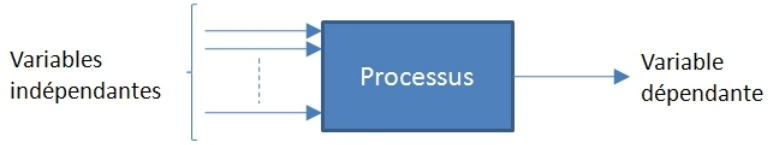


FIG. 2.2 – Illustration des variables indépendantes et dépendantes

Parmi les variables indépendantes, il est possible de définir deux sous-catégories de variables durant la conduite d'une expérience. La première sous-catégorie regroupe les variables indépendantes pour lesquelles les effets de changement sont étudiés. Ces variables s'appellent des «*facteurs*». La seconde sous-catégorie regroupe l'ensemble des variables indépendantes restantes qui sont contrôlées à un niveau fixe. Cette répartition des variables indépendantes est nécessaire pour identifier/déterminer les variables qui influencent. Un «*traitement*» est une valeur particulière d'un facteur.

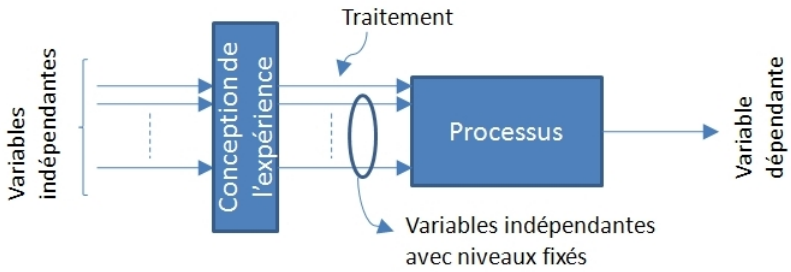


FIG. 2.3 – Illustration d'une expérience

Les traitements sont appliqués sur des combinaisons d'objets et de sujets. Les personnes qui réalisent le traitement sont appelées «*sujets*». Les sujets tout comme les objets peuvent être caractérisés par des variables indépendantes dans l'expérience. Les programmes sont un exemple d'«*objet*» et les étudiants un exemple de «*sujet*».

Une expérience est composée d'un ensemble de «*tests*», chacun composé d'un traitement, de sujets et d'objets. Ces tests doivent être distingués des tests statistiques présentés dans la Section 2.5.3. Il faut garder à l'esprit que plus le nombre de tests sera élevé, plus l'erreur sera grande. Or cette erreur permet à l'expérimentateur de déterminer la confiance qu'il doit porter dans les résultats de son expérience.

A titre d'exemple, un test peut être présenté de la sorte : un étudiant X (*sujet*) utilise une nouvelle technique de représentation graphique (*traitement*) pour concevoir l'architecture d'un programme (*objet*).

2.1.2 Processus

Un processus fournit un certain nombre d'étapes permettant de supporter une activité, et ce, en énonçant ce qu'il y a à faire et comment le faire. La réalisation d'une expérience nécessite également plusieurs étapes devant être réalisées dans un certain ordre. Un processus est, par conséquent, requis pour la conduite d'une expérience.

Le processus présenté à travers ce chapitre s'applique aux expériences mais les étapes de base peuvent être appliquées dans n'importe quel type d'étude empirique. La principale différence réside dans le travail à réaliser au sein d'une activité, comme par exemple, la conception d'un sondage, d'une expérience, ou d'une étude de cas. Ces conceptions sont différentes les unes par rapport aux autres mais l'étape de «*conception*» reste cependant nécessaire dans le processus.

La toute première étape avant de réaliser une expérience est de se demander si cette expérience permettra de répondre à la question que l'on veut étudier. Cette étape n'est pas facile, surtout en génie logiciel, vu que le côté empirique est très peu utilisé [Tichy 95, Zelkowitz 98]. Un certain nombre de raisons pour lesquelles les informaticiens devraient expérimenter plus a été fourni à travers la Section 1.2.1, voir également [Tichy 98].

Le processus d'une expérience peut être divisé en cinq étapes principales. Il s'agit premièrement de la **définition** de l'expérience en termes de problèmes, deuxièmement de la **planification** dans laquelle la conception de l'expérience est réalisée. L'**exécution** de l'expérience est la troisième étape. C'est à travers cette étape que les données sont collectées et qu'elles serviront de base à la quatrième étape : l'**analyse et l'interprétation** de ces données. La dernière étape, appelée **présentation et paquetage**, consiste à présenter les résultats des analyses. Cette dernière étape a également pour objectif d'archiver les conclusions et autres leçons apprises tout au long de l'expérience.

Ce processus est illustré à travers la Figure 2.4. La fin de la section est consacrée à l'explication succincte de chacune des étapes du processus. Les Sections 2.2–2.5 expliquent plus en détails chacune de ces étapes.

Le processus d'une expérience n'est pas un véritable modèle en cascade. Il n'est pas nécessaire de terminer une étape avant d'en commencer une autre. Le processus présenté à travers la Figure 2.4 indique uniquement la manière initiale de réaliser une expérience. Ce processus se veut itératif et il n'est pas interdit de revenir en arrière afin de raffiner une étape précédente avant de continuer l'expérience. Une exception doit cependant être retenue. Une fois que l'*exécution* de l'expérience a commencé, il est impossible de revenir aux phases de *définition* et de *planification*. La raison est qu'une expérience ne peut être réalisée qu'une seule fois pour chaque sujet sous peine de biaiser les résultats. Il est important de savoir que le lancement de la phase d'exécution a pour conséquence d'influencer tous les sujets qui participent à cette expérience. Faire passer deux fois la même expérience à un sujet ne donnera pas les mêmes résultats, et ce, à cause dudit «*phénomène d'apprentissage*».

Définition. Dans cette première étape, la ou les hypothèses sont clairement établies. L'important est de formuler le problème que l'on veut résoudre, peu importe si les hypothèses ne sont pas formulées de manière formelle. Un certain cadre de références a été établi afin de fournir cette définition [Briand 96a, Lott 96]. Les différents éléments sont les suivants :

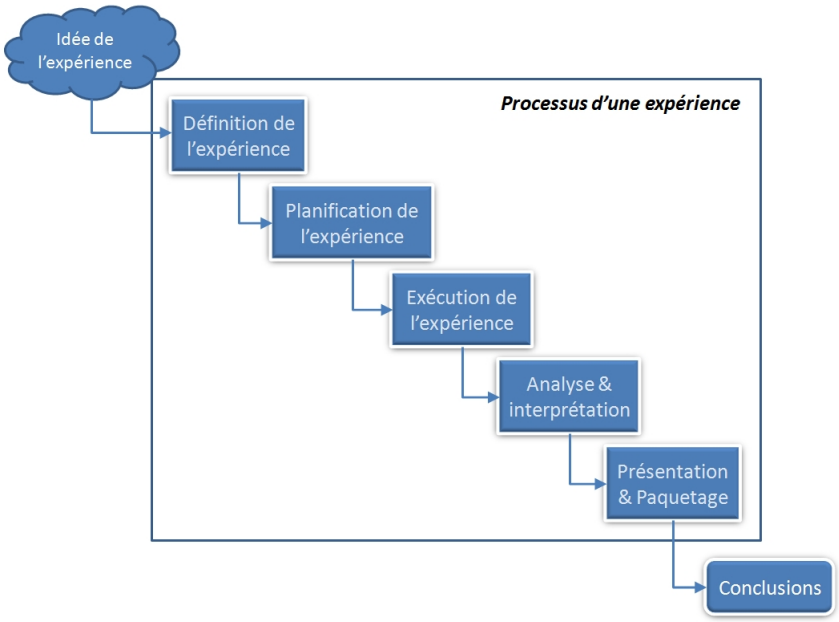


FIG. 2.4 – Processus d'une expérience

- Objet d'étude (qu'est ce qui est étudié ?)
- But (quelle est l'intention ?)
- Focalisation au niveau de la qualité (quel est l'effet étudié ?)
- Perspective (de quel point de vue ?)
- Contexte (où est réalisée l'expérience ?)

Planification. Cette seconde étape est importante pour l'*exécution* de l'expérience. Un certain nombre d'activités rigoureuses sont réalisées à travers cette étape.

L'explication détaillée du contexte doit d'abord être donnée. Il est important de connaître les personnes ainsi que l'environnement dans lequel se déroule l'expérience, qu'il s'agisse d'un environnement académique ou industriel. Les hypothèses sont ensuite définies formellement en passant par une ou des hypothèses nulles et respectivement, une ou des hypothèses alternatives (voir Sections 2.3.2 et 2.5.3).

L'identification des variables indépendantes (*inputs*) et dépendantes (*output*) doit ensuite être réalisée. Il est également important d'identifier quelles seront les valeurs que ces variables prendront. Cette étape sert également à déterminer l'échelle de mesure qui permettra de connaître les méthodes d'analyses statistiques qui pourront être utilisées. Les sujets de l'expérience sont également identifiés.

Une fois toutes ces tâches réalisées, le déroulement de l'expérience est conçu. Cette conception passe par un certain nombre de questions telles que l'utilisation du facteur aléatoire, l'utilisation d'équipements ainsi que la production de documents et logiciels nécessaires au déroulement de l'expérience.

La dernière partie de cette étape consiste en la réflexion sur la validité de l'expérience, et plus particulièrement sur les menaces à la validité. Il existe quatre classes principales de

validité : interne, externe, de construction et de conclusion. La **validité interne** concerne la pertinence de l'environnement utilisé ainsi que la fiabilité des résultats. La **validité externe** concerne les conditions de généralisation des découvertes. Est-ce que les relations trouvées dans le contexte de l'expérience sont applicables à n'importe quel contexte ? La **validité de la construction** évalue si le traitement reflète bien la cause et si la sortie représente bien l'effet, voir Figure 2.1. La **validité de la conclusion** concerne la relation entre le traitement et la sortie de l'expérience. Le but de ces tâches est d'assurer que l'expérience fournisse des résultats pertinents.

Exécution. Cette étape est divisée en trois étapes :

- Préparation
- Exécution
- Validation des données

La **préparation** consiste à préparer les sujets tout comme le matériel. Les sujets doivent être informés des intentions de l'expérience et leur consentement doit être obtenu. De plus, ces derniers doivent être impliqués. L'**exécution** de l'expérience ne pose généralement pas de problème. Il faut cependant s'assurer que le déroulement suive bien la conception initiale. Finalement, il faut s'assurer, via la phase de **validation des données**, que les données collectées soient correctes et qu'elles reflètent bien l'expérience.

Analyses et interprétations. Les données récoltées lors de l'étape précédente servent d'«*inputs*» à cette activité.

La première étape consiste à se faire une première idée des résultats en utilisant la statistique descriptive. Cela permet de visualiser les données, de les comprendre, mais aussi de les interpréter de manière non formelle. L'étape suivante a pour objectif de déterminer si l'ensemble des données est pertinent ou non. Cela nécessite parfois la suppression de données ou la réduction du nombre de variables. Une fois cette étape réalisée, les données peuvent être utilisées dans des tests d'hypothèses en fonction du type d'échelle utilisé.

Un point important de cette activité est qu'elle forme la base de la prise de décision et de la conclusion. C'est à travers cette étape qu'il est décidé si la ou les hypothèses sont retenues ou rejetées.

Présentation et paquetage. Cette dernière étape présente et fournit les documents expliquant les résultats obtenus à travers l'étude empirique. Un papier pour une publication est un des objectifs de cette étape.

L'important dans cette activité est de s'assurer que les leçons apprises seront prises en compte dans le futur. De plus, cette documentation doit faciliter la réplication de l'expérience étant donné qu'elle ne fournit jamais une réponse finale catégorique. C'est pourquoi cette documentation doit être compréhensible et détaillée.

2.2 Définition

La définition d'une expérience permet de déterminer les fondements de celle-ci. Si ces fondements ne sont pas correctement établis, une révision s'impose. Dans le pire des cas, l'expérience peut ne pas être réalisée pour étudier ce qui était visé. Le but de cette phase est de définir les objectifs de l'expérience selon un cadre d'évaluation particulier. Il faut toujours garder à l'esprit que l'important est de s'assurer que l'intention d'une expérience doit être satisfaite par l'expérience en elle-même.

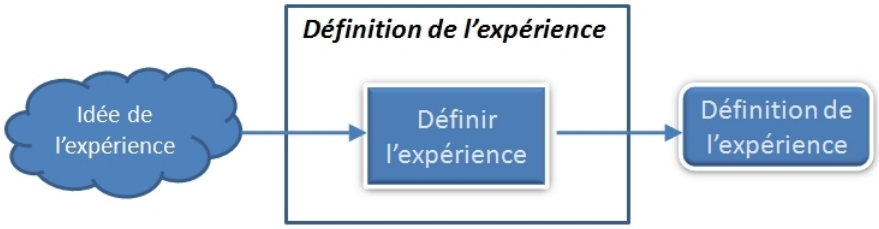


FIG. 2.5 – Phase de définition

2.2.1 Définir l'expérience

La définition des buts de l'expérience via un canevas assure le fait que les aspects importants de l'expérience seront bien définis avant les phases de *planification* et d'*exécution*. L'utilisation de ce canevas permet de définir correctement les fondements de l'expérience. Ce canevas se présente sous la forme suivante :

> Analyse du ou des <*Objet(s) d'étude*>
> dans le but de <*Objectif*>
> selon la <*Focalisation sur la qualité*>
> sous l'angle de <*Perspective*>
> dans le contexte du <*Contexte*>

Objet d'étude. L'objet d'étude est l'entité qui est étudiée durant l'expérience. Il délimite l'étendue de l'expérience. Ces objets peuvent être des produits, des ressources, des processus, des métriques, des modèles ou des théories.
But. Le but définit l'intention de l'expérience. Il peut s'agir d'évaluer l'impact de certains facteurs ou de comparer deux techniques.
Focalisation sur la qualité. Il s'agit des effets analysés lors de l'expérience. Ces effets sont par exemple l'efficacité, le coût, la fiabilité, la compréhensibilité, la maintenabilité, etc.
Perspective. La perspective définit le point de vue sous lequel les résultats de l'expérience sont analysés. Le point de vue du développeur, du chef de projet, du client sont des exemples de perspective.
Contexte. Le contexte définit l'environnement dans lequel l'expérience se déroule. Ce contexte définit quels sont les personnes (sujets) ainsi que les artefacts (objets) utilisés dans l'expérience. Les sujets peuvent être caractérisés par leur expérience, la taille de l'équipe, la charge de travail, etc. Les objets, quant à eux, peuvent être caractérisés par la taille, la complexité, le domaine d'application, la priorité, etc.

2.3 Planification

La phase de *planification* succède à la phase de *définition*. Alors que cette dernière s'attelait à définir *pourquoi* l'expérience était réalisée, la phase de planification prépare, quant à elle, *comment* l'expérience va être réalisée.

Cette phase peut être divisée en six étapes. La définition de l'expérience détaillée dans la section précédente représente l'entrée de cette phase. La **sélection du contexte** définit l'environnement dans lequel se déroulera l'expérience, et ce, sur base de la définition de l'expérience. Viennent ensuite **la formulation des hypothèses** ainsi que la **sélection des variables** indépendantes et dépendantes. La **sélection des sujets** se fait ensuite, suivie du choix de la **conception de l'expérience** basée sur les hypothèses et les variables sélectionnées. L'activité d'**instrumentation** réfléchit sur l'implémentation pratique de l'expérience. Finalement, l'**évaluation de la validité** vise à vérifier la validité de l'expérience. Ce processus de planification se veut itératif jusqu'à l'obtention d'une conception complète de l'expérience. Un aperçu du processus de *planification* est fourni à travers la Figure 2.6.

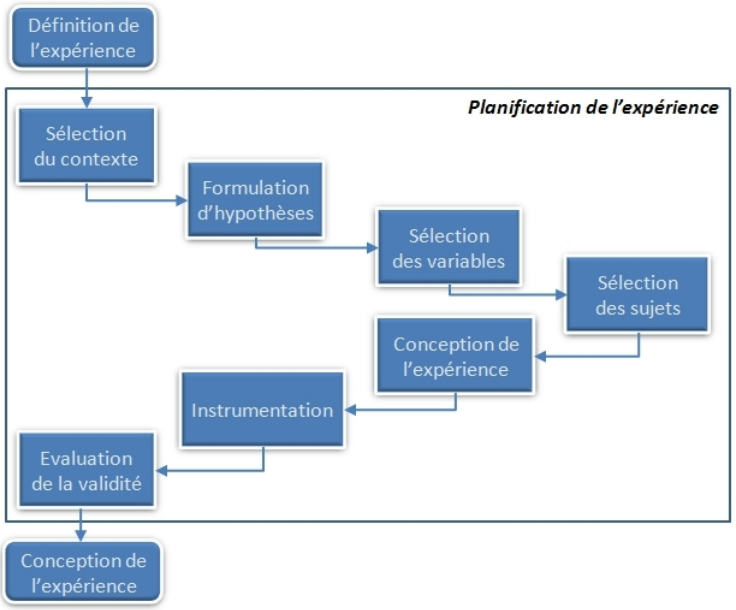


Fig. 2.6 – Phase de planification

2.3.1 Sélection du contexte

Afin d'obtenir les résultats les plus généraux possibles, une expérience devrait être réalisée dans un grand nombre de projets de taille réelle, impliquant des professionnels. Cependant les expériences comportent certains risques. Une solution est de réaliser l'expérience «*hors ligne*», c'est-à-dire utiliser les ressources, l'environnement, les professionnels mais ne pas utiliser les résultats au niveau professionnel. Cette solution est fort coûteuse mais permet de réduire les risques. Une autre alternative est de réaliser les expériences avec des étudiants. Ces expériences sont moins coûteuses, plus faciles à contrôler mais sont plus dirigées vers un contexte particulier. De plus, elles ne travaillent pas sur des problèmes réels mais plutôt sur des problèmes de petite taille suite à des contraintes de temps et de coûts.

Tous ces éléments doivent être pris en compte afin de déterminer si l'étude se veut valide pour un contexte spécifique ou valide pour le domaine du génie logiciel en général.

Le contexte d'une expérience peut ainsi être caractérisé selon les critères suivants [Wohlin 00] :

- Hors ligne vs. en ligne
- Etudiant vs. professionnel
- Problème de petite taille vs. problème réel
- Spécifique vs. général

La situation habituelle est une expérience réalisée dans le but de comparer un ancien élément à un nouveau.

2.3.2 Formulation d'hypothèses

La base des analyses statistiques lors d'expériences est le test d'hypothèse. La première étape consiste à formuler rigoureusement une hypothèse. Il faut ensuite collecter des données durant l'expérience afin de rejeter, si possible, la ou les hypothèses formulées en amont. Si la ou les hypothèses peuvent être rejetées, il est possible de tirer certaines conclusions basées sur les résultats des tests d'hypothèses en considérant certains risques.

Lors de la phase de formulation, au moins deux hypothèses doivent être formulées :

- **Hypothèse nulle, H_0 :** Cette hypothèse énonce qu'il n'y a pas de tendance ou de relation entre des éléments. La seule raison pour laquelle on observe des variations est le fruit du hasard. C'est cette hypothèse que l'expérimentateur tente de rejeter.

- **Hypothèse alternative, H_a, H_1 :** Il s'agit de l'hypothèse qui correspond au rejet de l'hypothèse nulle. Généralement, l'hypothèse alternative correspond à la négation de l'hypothèse nulle.

L'utilisation des tests d'hypothèses implique également la présence de risques. Ces risques sont présentés dans [Wohlin 00] sous les noms de «***erreur de type I***» et «***erreur de type II***». D'autres termes sont utilisés comme dans [Noirhomme 04] qui les présente sous les noms d'«***erreur de 1re espèce***» et d'«***erreur de 2e espèce***».

- **Erreur de 1re espèce** : Ce type d'erreur apparait lorsque l'hypothèse nulle est rejetée alors qu'elle est vraie. Dit autrement, il s'agit de la probabilité de rejeter H_0 alors que H_0 est vraie, ce qui peut s'exprimer sous la forme :
$P(erreur\ de\ 1^{re}\ espèce) = P(rejeter\ H_0 \mid H_0\ vraie)$.

- **Erreur de 2e espèce** : Ce type d'erreur apparait lorsque l'hypothèse nulle n'est pas rejetée alors qu'elle est fausse. Dit autrement, il s'agit de la probabilité de ne pas rejeter H_0 alors que H_0 est fausse, ce qui peut s'écrire sous la forme :
$P(erreur\ de\ 2^{e}\ espèce) = P(ne\ pas\ rejeter\ H_0 \mid H_0\ fausse)$.

2.3.3 Sélection des variables

Après avoir formulé les hypothèses, l'expérimentateur doit déterminer quelles seront les variables dépendantes et indépendantes.

Choix des variables indépendantes

Les variables indépendantes correspondent aux variables que l'expérimentateur peut contrôler lors de l'expérience. Ces variables peuvent éventuellement avoir un effet sur les variables dépendantes. Les réflexions concernant les choix de ces variables sont importantes pour au moins deux raisons. La première est que les variables résultant de ces choix permettront, suites aux analyses statistiques, de formuler, ou non, certaines conclusions. La seconde raison est que le choix de ces variables nécessite une connaissance du domaine. Lors du choix de ces variables, le type d'échelle de mesure est défini ainsi que le niveau auquel les tests seront réalisés.

Choix des variables dépendantes

Les variables indépendantes sont les variables influencées par certaines variables dépendantes. Généralement, il n'y a qu'une seule variable dépendante qui devrait être dérivée des hypothèses. Une variable dépendante n'est généralement pas mesurable directement. Elle nécessite une mesure indirecte qui doit être validée afin de ne pas affecter les résultats. Une fois le choix des variables dépendantes fait, il est possible de raffiner la ou les hypothèses. Cependant le choix de l'échelle et l'intervalle des variables doit déjà être déterminé.

2.3.4 Sélection des sujets

La sélection des sujets est importante lors de la conduite d'une expérience [Robson 93]. L'important ici est de trouver un échantillon suffisamment représentatif de la population que l'on désire analyser, afin de pouvoir généraliser les résultats à cette population. Le terme également utilisé pour la sélection des sujets porte le nom d'«*échantillonnage*» d'une population.

Les échantillons peuvent être de deux natures différentes : *probabiliste* ou *non-probabiliste*. Dans le premier cas, la probabilité de sélection de chaque sujet est connue, alors que dans le second cas, elle est inconnue. A titre d'exemple, voici quelques techniques probabilistes généralement utilisées :

- **Echantillonage simplement aléatoire**. Les sujets sont sélectionnés parmi une liste d'une population de manière aléatoire.
- **Echantillonage systématique**. Le premier sujet est sélectionné aléatoirement au sein d'une liste d'une population. Toutes les n^{e} personnes sont ensuite sélectionnées de la liste.
- **Echantillonage aléatoire en strates**. La population est répartie en un certain nombre de groupes ou strates avec une distribution connue entre les groupes. Un échantillonnage aléatoire est ensuite appliqué au sein des strates.

En ce qui concerne les techniques non-probabilistes, il existe :

- **Echantillonage approprié**. La personne la plus proche et la plus appropriée est choisie comme sujet.

La taille de l'échantillon est un paramètre à prendre en compte lors de cette phase de sélection. Plus un échantillon sera grand, plus il sera possible de généraliser les résultats, et moins il y aura d'erreurs lors de ces généralisations.

Il existe certains principes de base lors du choix de l'échantillon :

- Lors d'une forte variabilité au sein de la population, un large échantillon doit être choisi.
- La taille de l'échantillon est également influencée par les analyses qui seront opérées dessus. Il faut donc réfléchir, lors de l'étape de *conception*, aux analyses qui seront utilisées.

2.3.5 Conception de l'expérience

Afin de tirer le meilleur parti d'une expérience, il est important de planifier et de concevoir celle-ci consciencieusement. Il est généralement fait usage de méthodes d'analyses statistiques sur les données collectées afin de les interpréter dans un premier temps. Dans un second temps, ces méthodes d'analyses permettront de tirer un certain nombre de conclusions. Ces méthodes d'analyses dépendent de la conception de l'expérience et des échelles de mesures utilisées, voir Section 1.4.1. C'est pourquoi les phases de *conception* et d'*analyse–interprétation* sont fortement corrélées.

Choix de la conception de l'expérience

Une expérience est composée d'un ensemble de tests. La conception de cette expérience décrit *comment* ses tests sont organisés et réalisés.

Comme mentionné précédemment, la conception d'une expérience est fortement liée aux analyses statistiques. Le choix de cette conception peut affecter ces analyses et réciproquement. Avant de concevoir une expérience, les hypothèses doivent être observées afin de trouver les analyses statistiques qu'il est possible de réaliser pour rejeter la ou les hypothèses nulles. Cette conception se base également sur les échelles de mesures, les objets et les sujets qui seront utilisés. Le nombre de traitements est également défini lors de la conception afin de s'assurer que les effets des traitements seront visibles. Si la conception d'une expérience est bien faite, elle en assurera la facilité de réplication.

Principes généraux de conception

Les principes généraux de conception sont la **randomisation**, le **blocage**, l'**équilibrage**. La plupart du temps, les conceptions d'expériences combinent plusieurs de ces principes.

- **Randomisation**. Il s'agit d'un des principes les plus importants de conception. La randomisation est généralement appliquée sur les sujets, les objets et sur l'ordre dans lequel les tests sont réalisés. La raison pour laquelle la randomisation est fortement utilisée lors d'expériences est que les méthodes statistiques utilisées la requièrent pour analyser les données. Elle permet également de lisser l'effet d'un facteur.
- **Blocage**. Certains facteurs ont parfois un effet pour lequel l'expérimentateur ne porte pas d'intérêt. Si cet effet est connu et contrôlable, il est possible d'utiliser cette technique du blocage. Cette dernière consiste à éliminer systématiquement un effet non désiré dans une étude. Elle permet aussi d'augmenter la précision d'une expérience.
- **Equilibrage**. Le but ici est d'assigner des traitements à un même nombre de sujets. L'équilibrage permet de faciliter et de renforcer les analyses statistiques des données.

Types standards de conception

Le but de cette section est de présenter au lecteur les différents types de conception, et plus particulièrement la technique utilisée dans l'expérience présentée au Chapitre 5.

Les différents types de conception d'une expérience sont :

- **Un facteur avec deux traitements**.
- **Un facteur avec plus de deux traitements**.
- **Deux facteurs avec deux traitements**.
- **Plus de deux facteurs avec deux traitements**.

Lors de l'expérience réalisée au sein du D.I.R.O., il a été fait usage de deux techniques, chacune dans un but particulier.

Les paragraphes suivants fournissent au lecteur la base nécessaire à la compréhension des explications fournies dans la Section 5.3.5 concernant la conception de l'expérience réalisée. Pour chaque type de conception, une explication et une conception particulière sont définies.

Un facteur avec deux traitements. Cette catégorie compare deux traitements, l'un par rapport à l'autre. En général, c'est la moyenne des variables dépendantes qui est comparée pour chaque traitement.

Voici l'une des conceptions possibles pour cette catégorie :

- **Conception complètement aléatoire** : Cette technique de conception assez simple est faite pour comparer deux moyennes. Les mêmes objets sont utilisés pour les deux traitements à comparer. L'attribution des traitements se fait de manière aléatoire pour chaque sujet. Celui-ci n'utilise qu'un seul traitement sur un seul objet, voir Tableau 2.1. Dans le cas où le nombre de sujets est le même pour chaque traitement, la conception est dite équilibrée.

Sujets	Traitement 1	Traitement 2
1	X	
2		X
3		X
4	X	
5		X
6	X	

TAB. 2.1 – Exemple de conception complètement aléatoire (deux traitements)

Exemple d'hypothèses : $H_0 : \mu_1 = \mu_2$
$H_1 : \mu_1 \neq \mu_2,\ \mu_1 < \mu_2\ ou\ \mu_1 > \mu_2$
Exemple d'analyses : t-test, Mann-Whitney

Un facteur avec plus de deux traitements. Tout comme les expériences avec seulement deux traitements, cette technique compare les traitements entre eux. La comparaison est également réalisée sur les moyennes.

Voici l'une des conceptions possibles pour cette catégorie :

- **Conception complètement aléatoire** : De la même manière que pour la technique précédente, cette conception utilise un seul objet pour tous les traitements, et les sujets sont assignés à un traitement de manière aléatoire.

Sujets	Traitement 1	Traitement 2	Traitement 3
1		X	
2			X
3	X		
4	X		
5		X	
6			X

TAB. 2.2 – Exemple de conception complètement aléatoire (plus de deux traitements)

Exemple d'hypothèses : $H_0 : \mu_1 = \mu_2 = \mu_3 = \ldots = \mu_n$
$H_1 : \mu_i \neq \mu_j$ *pour au moins une pair* (i, j)
Exemple d'analyses : ANOVA (ANalysis Of VAriance), Kruskal-Wallis

2.3.6 Instrumentation

Il existe trois types d'instruments lors d'une expérience : les **objets**, les **guidelines** et les **instruments de mesure**. Ces instruments sont choisis lors de la phase de planification.

- **Les objets**. Ils peuvent correspondre à des spécifications ou à de la documentation de code. L'important est de choisir les objets appropriés aux informations à collecter lors de l'expérience. Si le but est, par exemple, d'identifier les défauts de conception dans un document, il sera préférable de travailler sur un document pour lequel les défauts seront préalablement identifiés et bien connus.

- **Les guidelines**. Ils aident les participants durant l'expérience. Il s'agit par exemple de descriptions de processus ou de «*check-lists*». Si le sujet doit utiliser plusieurs méthodes, des guidelines pour chacune des méthodes doivent lui être fournies.

- **Les instruments de mesure**. Les mesures lors d'une expérience sont réalisées via des collectes d'informations. Généralement, dans des expériences orientées sur des personnes, les données sont collectées via des formulaires ou des questionnaires. Cette étape de planification sert précisément à préparer ces formulaires et questions dans un premier temps, et à les faire valider, dans un second, par des personnes ayant le même bagage et les mêmes compétences que ceux des sujets.

Le but général de l'instrumentation est de fournir des moyens pour réaliser une expérience et pour la diriger, sans en affecter son contrôle. Il faut cependant retenir une chose. Les résultats d'une expérience doivent être indépendants de l'instrumentation utilisée pour l'expérience. Si cela n'est pas le cas, les résultats sont invalides. Ce concept de validité au niveau des résultats est présenté à travers la Section 2.3.7 suivante.

2.3.7 Evaluation de la validité

Le but ici est de constater à quel point les résultats sont valides. On parle également de validation *adéquate*. Celle-ci concerne la validité des résultats pour une population d'intérêt. Premièrement, les résultats doivent être valides pour la population à partir de laquelle l'échantillon a été prélevé. Deuxièmement, pour affirmer que les résultats ont une validité adéquate, les résultats doivent pouvoir être valides pour une plus large population que celle à partir de laquelle l'échantillon a été prélevé, cela, dans un but de généralisation uniquement.

Les résultats d'une expérience ne doivent pas toujours être généralisables à toute population. Une expérience peut, par exemple, être réalisée pour une entreprise. Dans ce cas, les résultats ne doivent être valides que pour la population de l'entreprise en question.

Il existe différentes classifications des types de menaces à la validité dans une expérience. Campbell et Stanley ont défini deux types de menaces : la validité *interne* et la validité *externe* [Campbell 63]. Ces deux types ont été étendus par Cook et Campbell dans [Cook 79]. Ces menaces portent sur la validité : *interne*, *externe*, de *conclusion* et de *construction*. Ces menaces se basent sur les principes de base d'une expérience, présentés en Figure 2.7.

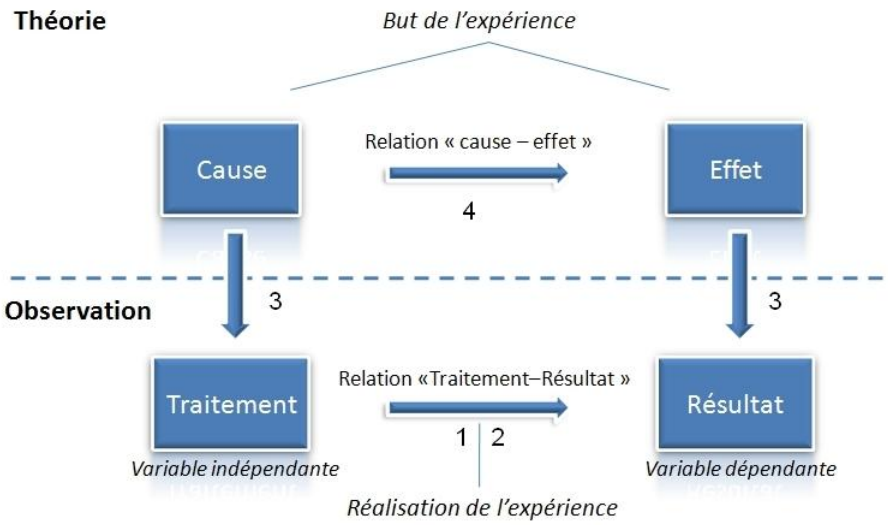


FIG. 2.7 – Principes d'une expérience et de sa validité (adapté de [Trochim 99])

Pour rappel, il existe une *zone théorique* et une *zone d'observation*. Le but est de tirer des conclusions concernant la théorie exprimée en termes d'hypothèses, via des observations. A chaque transition, représentée par une flèche dans la Figure 2.7, un certain type de menace est identifiable. Ces menaces sont explicitées ci-après.

1 **Validité de la conclusion**. Cette validité concerne la relation entre le traitement et le résultat. Il s'agit ici de s'assurer qu'il existe une relation statistique avec un certain degré de signification.
2 **Validité interne**. Le but ici est de vérifier, lors de la présence d'une relation entre le traitement et le résultat, si cette relation est une relation de cause à effet. Il s'agit donc d'analyser si cette relation n'est pas le fruit d'un facteur pour lequel l'expérimentateur n'a pas le contrôle. Autrement dit, «est-ce que le traitement cause le résultat ?».

3 **Validité de la construction**. Cette validité traite de la relation entre la théorie et l'observation. Le but, après s'être assuré que la relation entre la cause et l'effet soit bien une relation causale, est de vérifier deux choses : premièrement que le traitement reflète le concept de «*cause*», et deuxièmement que le résultat reflète le concept d'«*effet*».
4 **Validité externe**. La validité externe concerne la généralisation des résultats. Le but ici est de se demander, lors de la présence d'une relation causale entre le concept de «*cause*» et d'«*effet*», si les résultats peuvent être généralisés à d'autres spectres en dehors de l'étude réalisée.

Les menaces à la validité de conclusion concernent les problèmes qui peuvent affecter l'établissement de conclusions correctes au niveau des relations entre le traitement et le résultat d'une expérience. Ces problèmes sont, par exemple, le choix de tests statistiques, le choix de la taille de l'échantillon, etc.

Les menaces à la validité interne concernent les problèmes qui pourraient indiquer qu'il existe une relation causale alors qu'il n'en existe pas. Les facteurs qui peuvent affecter la validité interne concernent la manière dont les sujets ont été sélectionnés, répartis entre les différents groupes, si un événement est survenu durant l'expérience, etc. Tous ces facteurs peuvent affecter le comportement du sujet.

Les menaces à la validité de construction concernent l'étendue pour laquelle l'expérience reflète le concept étudié. Par exemple, le nombre d'années de pratique est une bonne mesure pour calculer l'expérience d'une personne dans un langage de programmation. Dans ce cas, l'expérience aura une bonne validité de construction.

Les menaces à la validité externe concernent la possibilité de généraliser les résultats en dehors du cadre de l'expérience. Cette validité dépend fortement de la conception de l'expérience, notamment du choix des objets et des sujets utilisés. Les trois risques principaux sont : l'utilisation des participants inappropriés, la réalisation de l'expérience dans un mauvais environnement et la réalisation de l'expérience avec un laps de temps qui affecte les résultats.

Différentes menaces pour chacune des validités sont présentées à travers le livre C. Wohlin [Wohlin 00]. De plus amples informations sont données également au travers des ouvrages de Campbell et de Cook, voir [Cook 79] et [Campbell 63].

2.4 Exécution

Une fois la *définition* et la *planification* de l'expérience réalisées, il convient d'*exécuter* cette expérience. Le but ici est de collecter les données pour les analyser par la suite. Cette étape est le moment où l'expérimentateur rencontre les sujets. C'est l'occasion, par exemple, pour l'expérimentateur d'expliquer aux sujets le but de l'expérience ainsi que son déroulement. Les expériences en génie logiciel sont souvent liées à l'utilisation de sujets humains. C'est pourquoi la Section 2.4.1 se focalise en partie sur *comment* motiver les sujets à participer aux expériences.

Il faut être très prudent lors de conduites d'expériences. Une expérience parfaitement conçue ainsi que des données analysées avec des méthodes appropriées peuvent cependant mener à des résultats invalides si les sujets ne sont pas appliqués. L'application et la motivation sont des facteurs à prendre en compte lors de l'exécution de l'expérience.

Cette phase d'*exécution* consiste en la succession de trois étapes : une étape de **préparation** des sujets et des matériaux nécessaires à l'expérience, une étape d'**exécution** où les sujets réalisent certaines tâches à partir desquelles les données sont collectées, et une étape de **validation des données** où les données collectées sont validées. Un aperçu de ces trois étapes est fourni à travers la Figure 2.8.

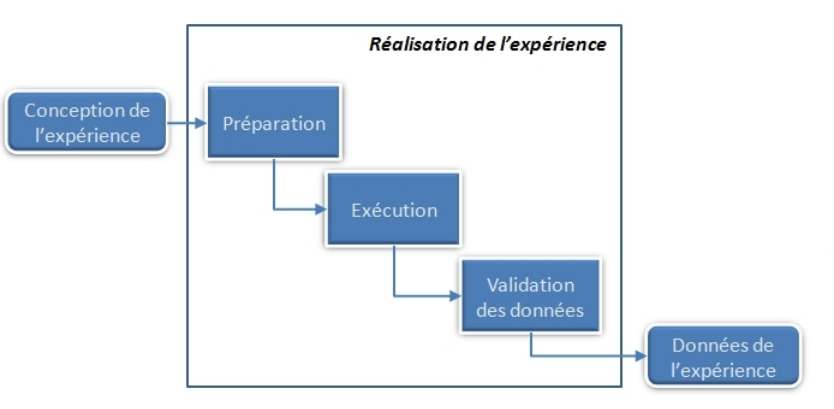


FIG. 2.8 – Phase d'exécution

2.4.1 Préparation

Avant de réaliser l'expérience concrètement, quelques activités préliminaires doivent avoir lieu. La première activité est la sélection et l'information des sujets. La seconde est la préparation des matériaux nécessaires à l'exécution de l'expérience. Il faut garder à l'esprit qu'au plus les réalisations des préparations seront meilleures, au plus il sera facile d'exécuter l'expérience.

Engager les participants

L'important dans une expérience est de trouver des participants motivés pour réaliser cette expérience. Il faut cependant faire attention à ne pas prendre n'importe quel type de sujets même s'il démontre une certaine motivation. Une expérience est généralement réalisée dans un domaine particulier. Si une expérience traite de diagrammes de classes, de

code orienté objet, il ne sera pas pertinent de recruter des programmeurs de code C. Il est important de trouver un échantillon représentatif de la population mise sous observation, sans quoi cela sera considéré comme une menace à la validité externe (voir Section 2.3.7).

Quelques aspects à prendre en considération lors du recrutement de sujets sont présentés ci-dessous :

- **Obtenir le consentement**. Les participants doivent être d'accord avec les objectifs de l'expérience. Ils doivent tout connaître de cette expérience sans quoi, des tâches pourraient affecter leur manière de répondre et par conséquent les résultats. Si les participants sont affectés par l'expérience, cela dépréciera la validité de l'expérience. Les sujets doivent connaître les intentions de l'expérience ainsi que l'utilisation qui sera faite des données.

- **Résultats sensibles**. Si les résultats sont sensibles aux yeux des participants, l'expérimentateur se doit d'assurer la confidentialité des données récoltées. La sensibilité des données est difficile à évaluer mais ces dernières sont généralement jugées comme sensibles à partir du moment où les participants attachent de l'importance et que ces données représentent quelque chose pour eux en dehors du cadre de l'expérience. Un exemple de donnée sensible est la productivité d'un programmeur.

- **Avantages**. Une manière d'attirer des sujets à participer à une expérience est de leur offrir certains avantages. Il faut cependant faire attention à ne pas tomber dans l'excès, sans quoi le participant pourrait n'être attiré que par la récompense et non plus par l'expérience en elle-même. Le but ici n'est pas de motiver fortement les sujets à participer à l'expérience.

- **Tromperie**. Tromper un sujet doit être évité à tout prix. S'il existe des méthodes alternatives, elles doivent être préférées aux tromperies. Si tromper le sujet est inévitable, il faut alors s'assurer que cela ne se fasse pas sur des données sensibles.

Problèmes d'instrumentation

Avant de réaliser l'expérience, tout le matériel doit être prêt à l'utilisation. Ce matériel peut très bien être des appareils de mesure, des questionnaires, des guidelines, etc. C'est la phase de conception qui définit le matériel à utiliser lors de l'expérience.

Une des premières questions à se poser, est de savoir si les données sont anonymes ou non. Si l'expérimentateur ne doit pas faire d'autres expériences avec le sujet et que les données personnelles ne l'intéressent pas, les données devraient alors être plutôt anonymes.

En général, il est plus approprié de faire une préparation personnalisée de l'instrumentation pour chaque sujet. Beaucoup d'expériences utilisent le côté aléatoire, ce qui a pour conséquence qu'une même configuration n'est utilisée que très peu de fois.

Si l'expérience est une interview, un ensemble de questions doit être préparé. Il peut être approprié, dans certains cas, de créer des questions propres à chaque participant.

2.4.2 Exécution

Une expérience peut être réalisée de plusieurs manières. Certaines expériences peuvent se réaliser en une fois alors que d'autres peuvent s'étendre sur plusieurs années. Dans le premier cas, l'expérimentateur dispose de tous les sujets en même temps, ce qui lui facilite la tâche. Il est possible de répondre aux questions immédiatement et les données sont toutes collectées en même temps. Dans le second cas, il se peut que l'expérimentateur ne participe pas à toutes les expériences. Certaines d'entre elles pourraient avoir lieu en même temps par exemple.

Collecte des données

Il existe différentes manières de collecter les données : soit manuellement par les participants remplissant des formulaires, soit manuellement via des outils, ou encore automatiquement par des outils. La première technique est la plus répandue et la dernière la moins repondue.

Faire remplir des formulaires par les participants n'oblige pas l'expérimentateur à prendre part à la collecte des données. Par contre, cette technique ne permet pas à l'expérimentateur d'identifier certaines incertitudes ou incohérences dans les formulaires. Les interviews, quant à elles, permettent de mieux communiquer avec les participants lors de la collecte des données. Cela prend cependant plus de temps et d'effort pour l'expérimentateur.

Environnement expérimental

L'important lors de la conduite d'une expérience est de réaliser celle-ci dans un environnement très semblable à celui habituel. Etant donné qu'une expérience vise à observer certains effets, si l'environnement habituel n'est pas reproduit ou conservé, ces effets seront perdus. Certaines exceptions peuvent cependant être tolérées selon les cas.

2.4.3 Validation des données

Une fois les données collectées, l'expérimentateur doit déterminer si celles-ci peuvent être conservées ou rejetées. Cela dépend premièrement si les participants ont compris les questions et le type de réponses qu'ils doivent donner. Cela dépend ensuite de l'implication et du sérieux du participant lors de ses réponses.

Cette étape est le meilleur moment pour vérifier si la réalisation de l'expérience permet de répondre aux objectifs fixés en amont.

2.5 Analyse et interprétation

Après avoir collecté les données, des conclusions doivent être tirées. Des interprétations quantitatives permettent de tirer ces conclusions de manière valide, et ce, en suivant trois étapes telles qu'elles sont représentées dans la Figure 2.9.

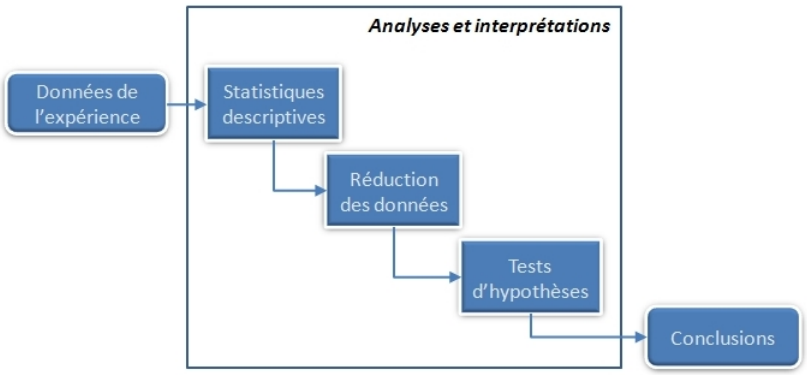


FIG. 2.9 – Trois étapes de l'interprétation quantitative des données

La première étape est l'utilisation des **statistiques descriptives** qui permettent de visualiser la tendance centrale, la dispersion, etc. La seconde étape est la **réduction de l'ensemble des données** dans le but d'obtenir un ensemble de données valides. Finalement, la troisième étape consiste à analyser les données via des **tests d'hypothèses**.

2.5.1 Statistique descriptive

La *statistique descriptive* permet de travailler et de présenter un ensemble de données. Elle peut être utilisée pour représenter graphiquement certains aspects intéressants des données. Le but de la statistique descriptive est d'obtenir une idée de *comment* sont distribuées les données. Il est intéressant, avant de tester les hypothèses, d'avoir recours à cette statistique, afin de se faire une idée sur la nature des données et d'identifier les points anormaux.

Le but de cette section n'est pas de présenter toute la théorie derrière la statistique descriptive, ni de définir toutes les mesures existantes dans le domaine. Cette section vise à donner au lecteur un certain vocabulaire concernant la statistique descriptive. Quelques définitions seront énoncées et une technique de visualisation particulière sera détaillée afin d'offrir au lecteur le bagage nécessaire à la compréhension de la Section 5.5. Plus d'informations sur les mesures en statistique descriptive sont fournies dans [Wohlin 00, Noirhomme 04].

Rappels théoriques

- **La moyenne**. Avec n données brutes $x_1, \ldots, x_n$, la **moyenne** arithmétique, notée $\bar{x}$ sera calculée comme :

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

  La moyenne est applicable lors d'échelles d'intervalles et de rapport, voir Section 1.4.1. Par exemple, la moyenne de l'ensemble $(1, 1, 2, 4)$ sera $\bar{x} = 2.0$.

- **La médiane**. Il s'agit de la valeur qui partage un ensemble de données en deux groupes égaux. En d'autres termes, le nombre de données supérieures à la médiane est le même que le nombre de données inférieures à la médiane. La première étape pour calculer la médiane consiste à trier les données par ordre croissant (ou décroissant) et de prendre ensuite la valeur centrale. Si n est impair, il n'y pas de problème pour trouver la médiane. Si, par contre, n est pair, la médiane sera calculée sur base de la moyenne arithmétique des deux valeurs centrales. La médiane est applicable lors d'échelles ordinales, d'intervalles et de rapport.
  A titre d'exemple, la médiane de l'ensemble $(1, 1, 2, 4)$ suivant sera $\tilde{x} = 1.5$.

- **Le quantile**. Le quantile est une généralisation du concept de médiane. Il est aussi appelé α-quantile où α prend ses valeurs dans l'intervalle $]0, 1[$. Certains quantiles ont des noms particuliers [Noirhomme 04] :

  $\alpha = \frac{1}{2}$ la médiane
  $\alpha = \frac{k}{4}$ le k^{e} quartile
  $\alpha = \frac{k}{10}$ le k^{e} décile
  $\alpha = \frac{k}{100}$ le k^{e} centile

  A titre d'exemple, le 45^{e} centile séparera l'ensemble des valeurs en deux groupes : le premier comprendra les premiers 45% des valeurs, et le second comprendra les 55% des valeurs restantes.

  Le *quartile*, quant à lui, est calculé comme 4-quantile [Wikipedia 08b, Wohlin 00].

  - Le 1^{er} quartile divise les 25% inférieurs de l'ensemble des données.
  - Le 2^{e} quartile est la *médiane*.
  - Le 3^{e} quartile divise les 75% inférieurs de l'ensemble des données.

  Ces notions sont importantes pour la compréhension de la section suivante ainsi que pour les analyses de données présentées dans le Chapitre 5, Section 5.5.

Visualisation graphique

La technique de visualisation graphique présentée ci-dessous se nomme la **boîte à moustaches** ou encore le **diagramme en boîte**. Cette technique permet de visualiser facilement la dispersion d'un échantillon.

Chaque boîte est composée de manière à faire ressortir les différents centiles graphiquement. Comme présenté sur la Figure 2.10, la boîte est construite en partant du premier quartile jusqu'au troisième quartile. Cette boîte est coupée par la médiane. Pour construire les deux moustaches (extrémités en haut et en bas), des segments sont tracés, soit jusqu'aux valeurs extrêmes, soit jusqu'aux 1^{er} et 9^{e} décile, soit jusqu'aux 5^{e} et 95^{e} centiles [Wikipedia 08a]. Ces deux segments portent les noms de «*queue inférieure*» et «*queue supérieure*». Les valeurs en dehors de ces queues sont appelées «*valeurs extrêmes*» ou «*valeurs solitaires*» (*outliers*).

Les valeurs dépassant le premier et le troisième quartile sont considérées comme «*outliers*». Ces «*outliers*» sont représentées par des cercles ou des étoiles, voir Figure 2.10. Les étoiles représentent les valeurs extrêmes.

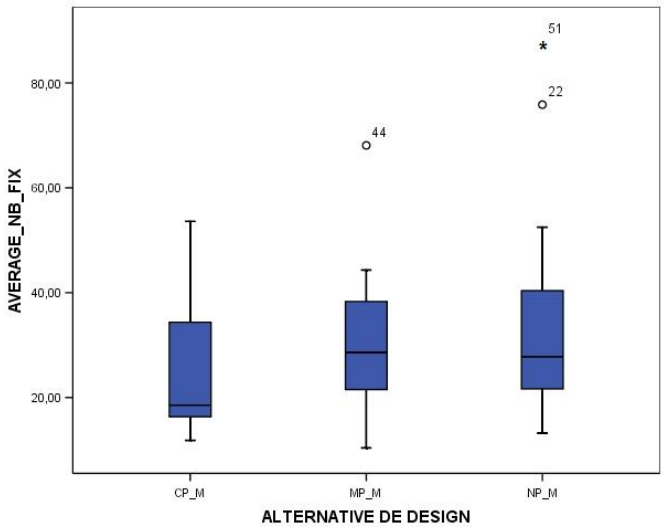


FIG. 2.10 – Boîte à moustaches

D'autres techniques de visualisation existent mais ne sont pas présentées dans cet ouvrage par soucis de concision. Parmi ces techniques, il existe le *diagramme de dispersion* (scatter plot), l'*histogramme* ou encore le *camembert*, voir détails dans [Wohlin 00].

2.5.2 Réduction de l'ensemble des données

L'important avec les données, lorsque des méthodes statistiques sont appliquées dessus, est de vérifier si ces données représentent ce que l'expérimentateur croie qu'elles représentent. Etant donné que les résultats dépendent fortement de la qualité des données en entrée, si ces données ne sont pas pertinentes, les résultats seront par conséquent erronés.

Le but de cette activité est, dans un premier temps, d'identifier les valeurs solitaires. S'il existe de telles valeurs, l'expérimentateur doit se demander ce qu'il doit en faire. Une bonne technique pour identifier ces valeurs solitaires est de tracer un graphique, soit un diagramme de dispersion, soit une boîte à moustaches. Les valeurs solitaires apparaîtront comme des points écartés du nuage de points ou de la boîte à moustache, voir Figure 2.10. Pour déterminer les valeurs solitaires, il existe un certain nombre de méthodes statistiques. Une technique, par exemple, consiste à comparer une valeur solitaire potentielle avec la moyenne de toutes les valeurs. Le but de ces méthodes est de déterminer si les valeurs, considérées comme potentiellement solitaires à première vue, proviennent d'une distribution normale ou non.

Une fois ces valeurs solitaires identifiées, l'expérimentateur doit déterminer ce qu'il compte en faire. Si une valeur solitaire provient d'un événement rare qui ne se reproduira plus jamais, elle peut être éliminée. C'est par exemple le cas lorsqu'un élément a mal compris la question ou la tâche. Si, par contre, la valeur solitaire provient d'une événement qui risque de se reproduire, cette valeur ne doit pas être retirée des données. Un exemple de ce type de valeur est la faible qualité d'une fonctionnalité réalisée par une équipe inexpérimentée. Ce genre de cas n'est pas le premier et le dernier à se produire. Les données doivent donc être conservées pour l'information importante qu'elles procurent.

2.5.3 Tests d'hypothèses

Concept de base

Le but des tests d'hypothèses est de déterminer s'il est possible de rejeter un certaine hypothèse nulle, H_0, basée sur un échantillon ayant une certaine distribution statistique. A partir d'un échantillon, une hypothèse nulle est formulée, décrivant les propriétés de la distribution de cette échantillon. L'expérimentateur cherche à rejeter ces propriétés avec une certaine marge de signification. Le cas habituel est de fixer un paramètre duquel dépend la distribution. En formulant H_0, l'expérimentateur détermine la distribution et assigne une valeur au paramètre qui sera testé.

Concrètement, pour tester H_0, un test t est défini et une région critique R_C est donnée. Cette région critique est une partie de la région dans laquelle t varie. Le test de signification peut être formulé de la manière suivante :

- Si $t \in R_C$, rejeter H_0
- Si $t \notin R_C$, ne pas rejeter H_0

L'hypothèse nulle devrait être formulée négativement vu que le but du test est de rejeter cette hypothèse. Dans le cas où l'hypothèse nulle n'est pas rejetée, il n'est pas possible de conclure quoi que ce soit concernant le résultat. A l'inverse, si l'hypothèse nulle est rejetée, il est possible d'énoncer l'hypothèse comme *fausse* avec un certain degré de signification (α). Avec ce genre de tests, il est possible de calculer la plus petite valeur significative (appelée aussi *p-value*), de différentes manières, ce qui permet de rejeter l'hypothèse nulle.

Pour rappel, voici trois valeurs importantes lors de réalisations de tests d'hypothèses :

- $\alpha = P(\text{erreur de } 1^{\text{re}}\text{espèce}) = P(\text{rejeter } H_0 | H_0 \text{ est vraie})$
- $\beta = P(\text{erreur de } 2^{\text{e}}\text{espèce}) = P(\text{ne pas rejeter } H_0 | H_0 \text{ est fausse})$
- Puissance $= 1 - \beta = P(\text{rejeter } H_0 | H_0 \text{ est fausse})$

Exemples de tests

Il existe un certain nombre de tests applicables selon les circonstances. Parmi ces tests, il existe : le **t-test**, le test de **Mann-Whitney**, le **F-test**, le **Paired t-test**, le test de **Wilcoxon**, le **Sign test**, le test **ANOVA** (ANalysis Of VAriance), le test de **Kruskal-Wallis**, le test de **Chi-2** et le **test binomial**.

L'expérience présentée au Chapitre 5 a fait usage, lors des analyses de données, de **t-tests** ainsi que de tests d'**ANOVA**.

t-test. Le t-test est utilisé dans le but de comparer deux échantillons indépendants. Il est utilisé lorsqu'il y a présence d'un facteur à deux niveaux.

ANOVA. L'analyse de la variance (**AN**alysis **O**f **VA**riance) est une méthode pour étudier les différences de moyenne entre des populations. Autrement dit, est-ce que les différences de moyennes entre plusieurs populations sont significatives ? Cette méthode doit son nom au fait qu'elle utilise des mesures de la variance afin de calculer le caractère significatif des différences de moyennes mesurées sur les populations.

3 Problématiques

Sommaire

Avant-propos

«Une étude empirique est tout simplement un test qui compare ce qu'on croie avec ce qu'on observe» [Perry 00].

Le but de ce chapitre est de présenter la problématique générale ainsi que la problématique particulière ayant conduit à la réalisation de l'expérience, présentée au Chapitre 5.

3.1 Introduction

La conduite d'une expérience sert généralement des objectifs plus larges que ceux de l'expérience en elle-même. Essayer de vouloir répondre à de larges problèmes en une seule expérience ne fait pas partie des bonnes pratiques [Perry 00]. Au lieu de ne réaliser qu'une seule expérience, il est généralement conseillé d'en réaliser plusieurs, chacune focalisée sur un problème particulier [Perry 00]. Cette manière de procéder permet de renforcer la *validité externe* de l'expérience (voir définition au Chapitre 2, Section 2.3.7).

L'objectif de ce chapitre est de préciser, dans un premier temps, la problématique générale dans laquelle s'inscrit la problématique particulière de l'expérience réalisée. Cette problématique générale sera détaillée dans la Section 3.2. L'objectif, dans un second temps, est de définir la problématique particulière de l'expérience. Cette problématique particulière sera détaillée à travers la Section 3.3.

3.2 Problématique générale

La problématique générale dans laquelle s'inscrit la problématique de l'expérience peut se définir comme :

« ***l'étude de la pertinence des patrons de conception*** »

Plusieurs études ont été réalisées en rapport avec cette problématique générale. Ces études peuvent être caractérisées selon trois aspects :

- Le nombre de patrons de conception étudiés
- Les qualités étudiées des patrons de conception
- Les contextes d'utilisation des patrons de conception étudiés

Ces études ont permis de constituer un corps de connaissances autour des patrons de conception. La Section 3.2.1 suivante présente ce que la communauté entend par «*patron de conception*». La Section 3.2.2 présente les qualités étudiées sur les patrons de conception et présente également les difficultés d'obtenir une théorie formelle, liant les patrons de conception avec les concepts de qualité logicielle. Les Sections 3.2.3 et 3.2.4 présentent respectivement les avantages et les inconvénients des patrons de conception, répertoriés par des experts dans le domaine. La Section 3.2.5 présente, quant à elle, les différents contextes d'utilisation des patrons de conception, répertoriés dans plusieurs articles. Finalement, la Section 3.2.6 énonce quelques constatations sur les patrons de conception, formulées par les recherches dans le domaine.

Remarque Le terme «*patron de conception*» utilisé à travers les différents chapitres de cet ouvrage est la traduction française du terme anglais «*design pattern*».

3.2.1 Patron de conception

Définition

La définition du terme «*patron de conception*» est la première pierre d'achoppement entre les experts du domaine. Néanmoins, les différentes définitions convergent pour la plupart vers un concept commun, nuancé subtilement par les experts.

Certains définissent le terme «*patron de conception*» comme [Prechelt 01, Prechelt 02] :

> « *Une solution éprouvée à un problème récurrent de conception de logiciel dans le but de rendre cette solution réutilisable.* »

D'autres définissent le terme «*patron de conception*» comme [Gamma 94, Briand 06] :

> « *Une bonne pratique documentée pour un problème récurrent donné, appliquée à une variété d'environnements et de situations.* »

Wendorff, dans [Wendorff 01], nuance cependant les termes de Gamma *et al.* [Gamma 94] et de Briand *et al.* [Briand 06] :

> « *Un patron de conception n'est pas nécessairement une bonne pratique, il s'agit plutôt d'une approche éprouvée d'un praticien.* »

Wendorff énonce qu'un patron de conception est plus une option de conception parmi d'autres disponibles lors du développement d'un programme. Le choix d'utiliser un patron de conception doit dépendre des objectifs de qualité requis dans le programme [Wendorff 01]. Indépendamment de ces définitions, un patron de conception peut être caractérisé par quatre éléments [Gamma 94] :

- Le «**nom**» du patron de conception qui exprime le concept
- Le «**problème**» décrivant les classes de situations auxquelles le concept s'applique
- La «**solution**» qui fournit une description abstraite d'un système générique d'éléments et leurs relations afin de solutionner le problème
- Les «**conséquences**» qui résultent de l'application de la solution, c'est-à-dire, les compromis, les coûts, les bénéfices, etc.

La taille d'un «*patron de conception*» doit être distinguée de celle d'un «*patron architectural*». Ce dernier exprime un schéma d'organisation structurelle fondamentale de tout un système. Alors qu'un patron de conception est un arrangement pour raffiner des sous-systèmes [Pfleeger 01].

Il existe actuellement plus d'une centaine de patrons de conception. Les principaux d'entres-eux sont tirés du célèbre livre du «*GoF*» (*Gang-of-Four*[1]). 23 patrons de conception y sont présentés. Depuis la parution de ce livre, de plus en plus de personnes ont développé leurs propres patrons de conception. Néanmoins, on distingue 3 grandes familles de patrons de conception parmi les 23 présentés [Gamma 94]. Il s'agit de :

- **Patrons créationnels**. Ils définissent comment faire l'instanciation et la configuration des classes et des objets.
- **Patrons structurels**. Ils définissent comment organiser les classes d'un programme dans une structure plus large (séparant l'interface de l'implémentation).
- **Patrons comportementaux**. Ils définissent comment organiser les objets pour que ceux-ci collaborent (distribution des responsabilités) et expliquent le fonctionnement des algorithmes impliqués.

[1] Fait référence aux quatre auteurs célèbres de cet ouvrage, à savoir : Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

3.2.2 Qualités

Les patrons de conception sont généralement utilisés pour les qualités qu'ils apportent ou qu'ils sont censés apporter à la conception du logiciel. Cependant, aucun des livres spécialisés dans le domaine, tels que [Gamma 94] ou [Buschmann 96], ne mentionne explicitement un modèle de qualité.

Parmi les qualités fréquemment mentionnées, on retrouve : la «*flexibilité*», la «*réutilisabilité*», la «*compréhensibilité*», la «*maintenabilité*» ainsi que l'«*extensibilité*» [Noda 01, Wendorff 01, Aversano 07]. Il n'est cependant pas rare qu'un attribut de qualité entre en conflit avec un autre attribut. L'utilisation d'un patron de conception peut, par exemple, augmenter la flexibilité et résulter en même temps en une complexité plus accrue du code [Wendorff 01]. L'utilité des patrons de conception dépend des besoins futurs du système à développer. Wendorff explique que le fait d'arriver à prédire les qualités futures requises par un système informatique, et ce, de manière compréhensible et objective, serait une avancée majeure dans le génie logiciel [Wendorff 01].

En 2001, Wendorff énonçait que la décision d'utiliser ou non un patron de conception relevait, d'une certaine manière, d'un jugement subjectif pour au moins deux raisons [Wendorff 01]. La première raison était que les moyens de mesurer les effets que possédaient les patrons de conception sur la qualité des logiciels étaient très limités [Pfleeger 97]. La seconde raison était qu'il n'existait pas de théorie formelle qui liait les patrons de conception avec les concepts de qualité logicielle. Selon Wendorff, il n'était donc pas possible d'obtenir un modèle précis expliquant *comment* fonctionnaient les patrons de conceptions, *quels* étaient leurs facteurs de succès et *comment* ces facteurs interagissaient, etc. [Prechelt 01].

En 2007, Hsueh *et al.* publient une étude réalisée dans le but de répondre indirectement aux deux problèmes identifiés par Wendorff [Hsueh 08]. Le but de cette étude est double :

- Fournir une approche évaluative afin d'aider les développeurs de patrons de conception à vérifier si ceux-ci sont bien conçus
- Fournir une méthode quantitative permettant de mesurer l'efficacité de l'amélioration de la qualité, fournie par un patron de conception

Les résultats obtenus par le deuxième objectif de cette étude permettront de déterminer si les patrons de conception utilisés permettent d'atteindre les exigences fonctionnelles et de qualité d'un logiciel [Hsueh 08].

3.2.3 Avantages

Les patrons de conceptions (PCs) possèdent un certain nombre d'avantages qui justifient leur utilisation. Plusieurs articles énoncent certains de ces avantages, sans toutefois apporter systématiquement des arguments empiriques en faveur de ces affirmations. C'est le cas notamment de [Aversano 07, Cline 96, Wendorff 01, Prechelt 01, Prechelt 02].

Parmi ces avantages, on retrouve les points suivants :

- Les PCs augmentent la réutilisabilité [Aversano 07]
- Les PCs améliorent la compréhension et la maintenance de systèmes existants [Gamma 94]
- Les PCs permettent de découpler une requête des opérations spécifiques [Aversano 07]
- Les PCs permettent de rendre un système indépendant de plate-formes hardware [Aversano 07]

- Les PCs permettent d'éviter de modifier les implémentations [Aversano 07]
- Les PCs permettent d'améliorer la productivité du programmeur [Prechelt 02]
- Les PCs permettent d'améliorer la qualité du logiciel [Prechelt 02]
- Les PCs facilitent l'intégration d'un nouveau programmeur dans un équipe car les connaissances des autres membres sont définies explicitement à travers les PCs [Cline 96]
- Les PCs permettent de gérer facilement les futurs changements d'un logiciel [Cline 96]
- Les PCs fournissent un vocabulaire standard entre les architectes [Cline 96] [Prechelt 01]
- Les PCs améliorent la flexibilité de la conception du logiciel [Wendorff 01] [Prechelt 01]

Ce dernier avantage permet de fournir une liste de points qui doivent être analysés lors de la revue d'un design [Cline 96]. Cet avantage permet également d'améliorer la communication entre les architectes et les responsables de la maintenance du logiciel [Prechelt 01].

3.2.4 Inconvénients

Les patrons de conception présentent également plusieurs inconvénients qu'il est important de connaître avant de les intégrer dans la conception d'un logiciel. Parmi ces inconvénients, les suivants peuvent être présentés [Wendorff 01, Cline 96, Prechelt 01] :

- Les patrons de conception apportent une complexité supplémentaire qui rend la compréhension du logiciel plus difficile [Gamma 94, Wendorff 01]
- Les patrons de conception augmentent le nombre d'artefacts (classes, fichiers, associations, etc.) [Wendorff 01]
- L'utilisation anticipée de patrons de conception peut mener à des dégradations de design. La présence de ces patrons de conception peut être oubliée et maintenue de manière inadéquate [Wendorff 01]

Il est important de faire remarquer que peu d'articles témoignent des problèmes liés aux patrons de conception.

3.2.5 Contextes d'utilisation

Les études impliquant les patrons de conception sont nombreuses. Elles peuvent être répertoriées en deux catégories distinctes :

- Les études remettant en cause les patrons de conception
- Les études se basant sur les patrons de conception

Ce qui distingue la deuxième catégorie de la première, c'est le domaine qui voit ses connaissances accrues. La première catégorie répertorie des études qui augmentent uniquement les connaissances sur les patrons de conception. La deuxième catégorie, quant à elle, répertorie des études qui augmentent les connaissances dans des domaines connexes aux patrons de patrons de conception. De plus, la seconde catégorie ne remet pas en cause l'utilité des patrons de conception.

A titre d'exemple, la première catégorie compte des études telles que :

- L'analyse des patrons de conception dans des projets commerciaux [Wendorff 01]
- La comparaison des patrons de conception avec des solutions de conception plus simples [Prechelt 01]
- L'identification des avantages et des inconvénients des patrons de conception dans le monde industriel [Cline 96]
- L'évaluation de l'utilité de la documentation des patrons de conception lors de la maintenance de programmes [Prechelt 02]
- L'analyse de l'évolution des patrons de conception dans un programme [Aversano 07]

La seconde catégorie, quant à elle, compte des études telles que :

- L'élaboration d'un outil guidant l'insertion de patrons de conception sur base de modèles UML [Briand 06]
- L'élaboration de techniques de visualisation des patrons de conception afin d'amélioration la compréhension des programmes [Trese 07]

3.2.6 Recommandations d'experts

Prechelt *et al.* relèvent le fait que le nombre de publications témoignant de l'efficacité des patrons de conception est très faible [Prechelt 02]. Plusieurs rapports d'études de cas sur les effets positifs des patrons de conception peuvent être trouvés dans [Beck 96] ou [Gamma 94].

La littérature actuelle concernant l'efficacité des patrons de conception sur la maintenance est assez pauvre. Il manque actuellement de recherches empiriques afin de prouver l'efficacité des patrons de conception sur la maintenance [Prechelt 02].

L'approche quantitative fournie par Hsueh *et al.*, dans [Hsueh 08], et l'étude de cas présentée ne suffisent pas à prouver l'efficacité des patrons de conception. Des expériences contrôlées devraient être réalisées afin de trouver des résultats pertinents de manière empirique.

Suite à une expérience réalisée dans le but de comparer des patrons de conception à des solutions plus simples, Prechelt *et al.* fournissent plusieurs suggestions [Prechelt 01] :

- Il est généralement, mais pas toujours, utile d'utiliser un patron de conception s'il y a des alternatives plus simples
- Le sens commun doit être utilisé afin de trouver les situations exceptionnelles pour lesquelles une solution plus simple devrait être préférée, même si le patron de conception peut être appliqué facilement
- En cas de doute (entre un patron de conception et une solution plus simple), l'utilisation d'un patron de conception plutôt qu'une solution plus simple est une bonne approche par défaut
- La compréhension accrue d'un patron de conception aide souvent lors de la maintenance de programmes, même si ces programmes ne sont pas complexes ni de grande taille.

Cette dernière constatation de Prechelt rejoint en partie l'idée défendue par Cline [Cline 96]. Celui-ci énonce que le tout premier pré requis d'un architecte orienté-objet est

de connaître comment appliquer les patrons de conception [Cline 96]. Les activités tournant autour des patrons de conception doivent être réalisées par des personnes qui désirent travailler avec des patrons de conception [Cline 96].

3.3 Problématique de l'expérience

Tel qu'il a été énoncé dans l'introduction, la conduite de l'expérience présentée au Chapitre 5 tend à répondre à une problématique plus large que celle abordée. La Section 3.2 a d'ailleurs permis de présenter plus amplement cette problématique générale. Sur base de cette dernière, il est à présent possible de présenter la problématique particulière de cette expérience. Celle-ci s'est attelée à étudier :

- **Un** patron de conception
- **Deux** qualités de ce patron de conception
- **Deux** contextes d'utilisation

Les Sections 3.3.1–3.3.3 décrivent successivement le patron de conception, les qualités et les contextes d'utilisation étudiés.

3.3.1 Patron de conception étudié

La patron de conception considéré dans cette expérience est le patron du «*Visiteur*» tiré du terme anglais «*Visitor pattern*». Ce patron de conception fait partie des 23 patrons de conception présentés dans le célèbre livre du «*GoF*» (*Gang-of-Four*) [Gamma 94].

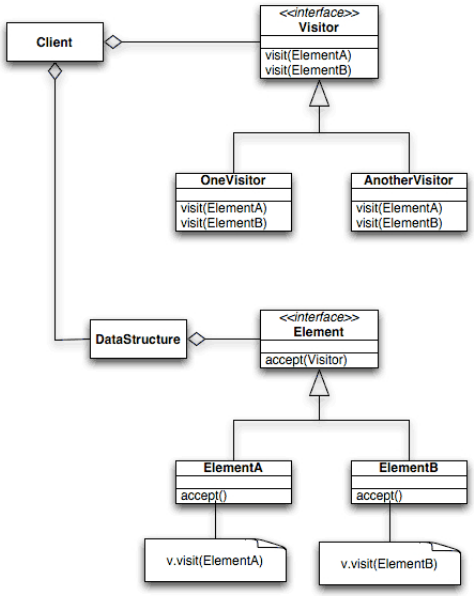


FIG. 3.1 – Patron de conception «*Visiteur*»

Le patron du «*Visiteur*», voir Figure 3.1 a pour but de séparer un algorithme d'une structure d'objets [Gamma 94]. Cette séparation facilite l'ajout de nouvelles opérations, fonctionnalités à la structure d'objets existante sans modifier cette structure.

L'exemple généralement présenté est celui d'un arbre de syntaxe abstraite et l'ajout d'une fonctionnalité de «*pretty printing*». La structure d'objets étant relativement conséquente, l'ajout d'une méthode d'affichage propre à chaque noeud est fastidieux. Une solution à ce problème est l'utilisation du patron du «*Visiteur*». Sur base de la Figure 3.1, l'ajout de la fonctionnalité «*Pretty Printing*» se ferait via l'introduction d'une nouvelle classe, sous type de la classe «*Visitor*». Cette nouvelle classe devra ainsi définir une méthode particulière appelée «*visit()*» pour chaque noeud visé de la structure d'objets. Cette méthode doit posséder comme argument le type du noeud visé.

Pour que ce patron fonctionne correctement, les noeuds de la structure d'objets doivent cependant répondre à une condition. Chaque classe de la structure d'objets doit implémenter une méthode appelée «*accept()*» avec comme paramètre un «*Visiteur*», soit la superclasse «*Visitor*» pour plus de généralité, soit un *Visiteur particulier*, sous-classe de «*Visitor*». Le corps de la méthode «*accept()*» devra appeler la méthode «*visit()*» du *Visiteur* passé en argument. Cette méthode «*visit*» possèdera comme paramètre le type courant de l'objet, soit «*this*».

Ce patron de conception a été choisi car il figure parmi les patrons les plus utilisés et étudiés [Prechelt 01, Aversano 07, Vokac 04]. De plus, il est l'exemple typique du patron de conception pour lequel plusieurs alternatives de conception existent.

3.3.2 Qualités étudiées

La Section 3.2.2 a expliqué les problèmes qui liaient les patrons de conception à la qualité logicielle. Afin d'éviter toute ambiguïté, et par soucis de conformité aux standards actuels, deux qualités tirées de la norme ISO 9126 ont été étudiées [ISO/IEC 01]. Il s'agit des qualités suivantes :

- L'analysibilité (*analysability*)
- La modifiabilité (*changeability*)

Ces deux qualités se présentent toutes deux comme des sous-caractéristiques de la «*Maintenabilité*».

Afin de mieux comprendre le modèle de qualité ISO 9126 définissant les «*qualités internes*» et les «*qualités externes*», la Figure 3.2 est mise à disposition du lecteur. Ce modèle de qualité se compose de six «*caractéristiques*» (la fonctionnalité, la fiabilité, l'utilisabilité, l'efficacité, la maintenabilité, la portabilité), chacune divisée en «*sous-caractéristiques*». Ces dernières peuvent être mesurées par des «*mesures internes*» ou des «*mesures externes*».

La **qualité interne** est l'ensemble des caractéristiques d'un produit logiciel du point de vue interne. La qualité interne est mesurée et évaluée à l'aide d'exigences de qualité interne. Ces dernières sont utilisées afin de spécifier les propriétés des produits intermédiaires [ISO/IEC 01].

La **qualite externe** est l'ensemble des caractéristiques d'un produit logiciel du point de vue externe. La qualité externe est mesurée et évaluée à l'aide de mesures externes lorsque le logiciel est exécuté, c'est-à-dire lors des phases de tests dans un environnement simulé [ISO/IEC 01].

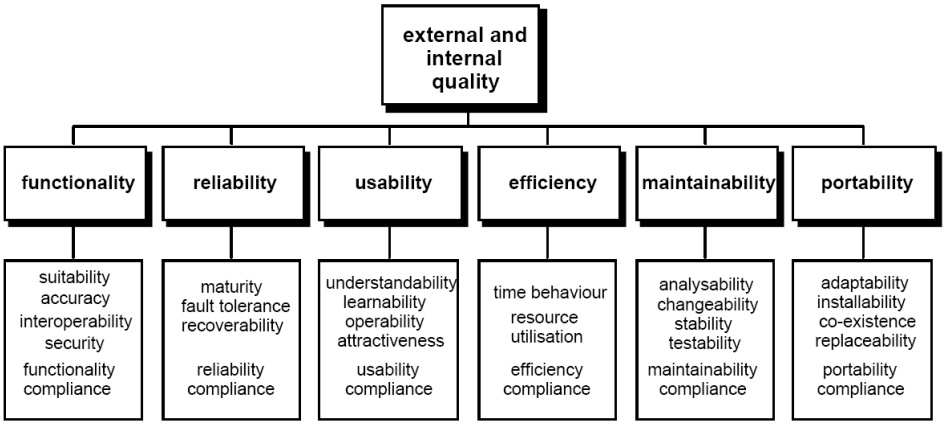


FIG. 3.2 – Modèle de qualité des qualités internes et externes

Concernant le concept de «*maintenabilité*» et les qualités étudiées sur le patron de conception «*Visiteur*», l'ISO 9126 fournit les définitions suivantes [ISO/IEC 01] :

- **La maintenabilité**. La capacité du produit logiciel d'être modifié. Ces modifications peuvent inclure des corrections, des améliorations ou l'adaptation du logiciel à son environnement, aux exigences et aux spécifications fonctionnelles.
- **L'analysibilité**. La capacité du produit logiciel d'être analysé afin de trouver des déficiences ou des causes d'échec dans ce logiciel, ou afin d'identifier les parties à modifier.
- **La modifiabilité**. La capacité du produit logiciel de permettre d'implémenter une modification spécifiée. Cette modification inclut le codage, la conception (*designing*) et le changement de la documentation.

Ces deux dernières qualités doivent être mises en regard avec les deux contextes d'utilisation étudiés au cours de l'expérience présentée au Chapitre 5.

3.3.3 Contextes d'utilisation

Deux contextes d'utilisation des patrons de conception ont été sélectionnés et étudiés durant cette expérience. Il s'agit de :

- La compréhension de programmes orientés-objet.
- La maintenance de programmes orientés-objet.

Tel qu'énoncé précédemment, les deux qualités étudiées doivent être mises en relation avec ces deux contextes d'utilisation des patrons de conception. Ainsi, l'expérience s'est attelée, d'une part à étudier l'*analysibilité* dans un contexte de *compréhension* de programmes. D'autre part, l'expérience s'est attelée à étudier la *modifiabilité* dans le contexte de la *maintenance* de programmes.

La question de savoir si la *maintenabilité* contient ou non de la *compréhension* a fait l'objet de réflexions. Il est certain que pour maintenir un programme, une personne doit préalablement s'approprier le fonctionnement de celui-ci. Néanmoins, des tâches bien distinctes ont été demandées aux sujets afin d'étudier adéquatement chacun des deux contextes.

4 Eléments de contexte d'une solution

Sommaire

Avant-propos

Différents éléments techniques peuvent prétendre répondre à un problème posé. Chacun de ces éléments influence cependant la manière de résoudre ce problème.

Ce chapitre présente les choix techniques qui ont été retenus afin de répondre à la problématique spécifique, présentée au cours du chapitre précédent.

4.1 Introduction

Face à la problématique spécifique exposée dans le Chapitre 3, Section 3.3, différents éléments techniques peuvent être retenus afin de constituer un contexte de solution. Il convient cependant d'analyser la pertinence de chacun de ces éléments techniques face à la problématique posée.

Les Sections 4.2 et 4.3 suivantes présentent les éléments techniques retenus. La Section 4.2 rappelle brièvement le langage de modélisation UML et le type de diagrammes utilisé lors de l'expérience. La Section 4.3 présente, quant à elle, la technologie d'*eye-tracking* et les raisons pour lesquelles cette technologie a été choisie. Chacune de ces deux sections sera ponctuée par les motivations qui ont conduit à retenir ces éléments techniques.

Alors que les *diagrammes de classes UML* permettent de représenter la structure statique des systèmes[1] abordés, la technologie d'*eye-tracking* permet de capturer les processus cognitifs des personnes [Rayner 98]. Ces deux technologies constituent le contexte de la solution, à savoir l'expérience réalisée, présentée au Chapitre 5.

4.2 Unified Modeling Language (UML)

UML est un ensemble de notations standardisées par l'OMG [Heymans 05, OMG 08]. Ces notations standardisées sont destinées à la modélisation de systèmes d'informations.

UML définit 13 types de diagrammes divisés en trois catégories : une catégorie permettant de représenter la **structure**, une autre le **comportement**, et la dernière les **interactions** [OMG 08].

Dans cette expérience, il a été fait usage de *diagrammes de classes* permettant de représenter la *structure* d'un système. Ce type de diagramme permet plus particulièrement de «*représenter les propriétés statiques des éléments du domaine d'application ou du système à construire, logiciels ou non logiciels*» [Heymans 05].

Suite à la popularité des diagrammes de classes, les différents concepts utilisés dans ces diagrammes (Classe, Association, Attributs, Méthodes, etc.) ne seront pas présentés à travers ce chapitre. Le lecteur peut cependant se référer, soit au cours d'AMSI[2] [Heymans 05], soit à la spécification du langage [OMG 08].

Les diagrammes de classes ont principalement été choisis car ils se sont imposés de facto ces dernières années comme un standard en matière de modélisation pour les systèmes orientés-objets [Guéhéneuc 06, Eichelberger 03, Eiglsperger 03]. Ces diagrammes sont souvent utilisés dans le monde de l'informatique pour communiquer, modéliser.

L'idée générale derrière l'utilisation massive de ce type de diagrammes est qu'ils fournissent des constructions appropriées à la modélisation et à la communication entre praticiens. En utilisant ce type de diagrammes, largement utilisés par la communauté, la validité de l'expérience présentée à travers le Chapitre 5 se voit renforcée.

[1] Voir Chapitre 5, Section 5.3.5 – **Systèmes**
[2] **A**nalyse et **M**odélisation des **S**ystèmes d'**I**nformation

4.3 Système d'eye-tracking

4.3.1 Objectifs

La discipline d'«*eye-tracking*», ou «*suivi du regard*» en français, existe depuis plus de 30 ans [Rayner 98]. Durant cette période, beaucoup d'études utilisant les mouvements des yeux ont analysés les processus cognitifs [Rayner 98]. Ces études se répartissent en trois ères de recherche sur les mouvements des yeux.

La première ère a traité le rôle des mouvements des yeux durant la lecture. Durant cette ère, plusieurs faits basiques ont été découverts à propos des mouvements de yeux. Parmi ces constatations, on retrouve la *suppression saccadique* (le fait qu'aucune information ne soit perçue durant les mouvements des yeux), ainsi que la *latence d'une saccade* (le temps que prend l'initiation d'un mouvement de l'oeil) [Rayner 98].

La deuxième ère désirait avoir une approche plus appliquée. Cependant, peu de recherches ont été menées avec les mouvements des yeux dans le but d'inférer sur les processus cognitifs [Rayner 98]. Les constatations de l'époque concluaient que presque tout ce qui aurait pu être appris sur les mouvements de yeux avait déjà été découvert [Rayner 98].

La troisième ère de recherche, quant à elle, commença au milieu des années 70 et a été marquée par les améliorations au niveau des systèmes d'enregistrement des mouvements des yeux. Ces systèmes ont permis d'enregistrer plus facilement les mouvements des yeux, et ce, de manière plus précise [Rayner 98]. Le plus important dans cette troisième ère est que les avancées technologiques dans le domaine ont permis d'interfacer des ordinateurs avec des systèmes d'*eye-tracking*. Cet interfaçage a rendu possible la collecte et l'analyse de grandes quantités de données [Rayner 98].

4.3.2 Types de systèmes

Les mouvements des yeux peuvent être capturés de différentes manières suivant la technologie utilisée par le système d'*eye-tracking*.

Les systèmes d'*eye-tracking* actuellement présents utilisent : (a) des *électrodes de surface* (qui sont relativement bonnes pour mesurer la latence d'une saccade mais mauvaises pour mesurer les coordonnées d'une saccade), (b) les réflexions de rayons infrarouges sur la cornée, (c) le contrôle de la pupille basée sur une caméra, (d) le suivi par rayons infrarouges des images Purkinje, et (e) la recherche basée sur des anneaux attachés à la surface des yeux tels que les verres de contact [Rayner 98].

Malgré la présence de nombreuses discussions concernant la manière de mesurer ou d'évaluer les mouvements des yeux, aucun standard de mesure n'a été adopté [Rayner 98].

4.3.3 Informations enregistrées

A partir du mouvement des yeux, il est possible d'extraire deux types d'information : les «*fixations*» et les «*saccades*».

Les «*fixations*» représentent la stabilisation des yeux sur un zone d'intérêt pendant un certain temps. Les «*saccades*» représentent les mouvements des yeux d'une zone à l'autre, c'est-à-dire, entre deux fixations [Rayner 98, Yusuf 07, Guéhéneuc 06].

Sur base de ces informations de base fournies par les systèmes d'*eye-tracking*, plusieurs articles définissent de nouvelles variables en fonction des buts poursuivis.

[Bednarik 06]	«*temps de fixation*» et «*nombre de fixations*»
Objectif :	▶ Révéler la stratégie utilisée lors de la lecture de code.
[Yusuf 07]	«*nombre de fixations*» et «*nombre moyen de fixations*»
Objectif :	▶ Identifier les caractéristiques les plus efficaces des diagrammes de classes. Par «*caractéristique*», Yusuf *et al.* entend le layout, la couleur et l'utilisation de stéréotypes au sein de diagrammes de classes UML.
[Pan 07]	«*nombre de fixations*» et «*nombre moyen de fixations*»
Objectif :	▶ Caractériser le comportement de sujets face aux résultats que produit le moteur de recherche Google.

TAB. 4.1 – Exemple de variables liées à l'*eye-tracking* et leur interprétation

Les valeurs obtenues à partir de ces variables sont, pour la plupart, liées à un *concept*. Ce dernier permet d'interpréter les variations de ces valeurs. Le concept généralement associé aux variables est l'«*effort*». Celui-ci peut, selon le contexte de l'expérience, servir à expliquer des concepts de plus haut niveau, tels que l'«*efficacité d'un layout*» ou la «*complexité des tâches*», etc. Les trois auteurs cités ci-dessus, établissent tous le même rapport entre le nombre total de fixations et le concept d'effort : «*un nombre plus élevé de fixations indique qu'un effort plus important a dû être réalisé*».

L'établissement du rapport entre les variables et le concept choisis nécessite une réflexion approfondie. En effet, c'est ce rapport entre les variables et le concept qui servira de base aux interprétations des données. A leur tour ces interprétations permettront de valider ou d'invalider une théorie. Il est important de constater ici le poids de la relation qui sera établie entre les variables et le concept. Dans le cadre de cette expérience, de nombreuses discussions ont eu lieu sur cette relation. Ces problèmes sont exposés à la Section 6.5.2.

4.3.4 Avantages

L'utilisation d'une technologie d'*eye-tracking* possède un certain nombre d'avantages. Ces derniers ont motivé le choix d'avoir recours à un tel type de système afin de traiter au mieux la problématique définie dans le chapitre précédent.

La compréhension de programmes a fait, et fait encore à l'heure actuelle, l'objet de bon nombre d'études. Ces études se sont attelées notamment à comprendre la création de modèles mentaux [Soloway 86], l'identification d'informations pertinentes [Murphy 05] mais aussi à proposer de nouvelles techniques permettant de représenter l'information [Seemann 97].

L'*eye-tracking* permet, de par sa nature, d'aller plus loin dans l'activité de compréhension. Etant donné que les systèmes d'*eye-tracking* sont basés sur la physiologie de la capacité visuelle de l'être humain [Duchowski 03], les données qui ressortent de tels systèmes

fournissent un aperçu des zones attirant l'attention, rendant ainsi possible les inférences sur les processus cognitifs sous-jacents [Rayner 98].

La combinaison des systèmes représentés via des diagrammes de classes avec l'utilisation d'un système d'*eye-tracking* permet de répondre à la problématique posée au chapitre précédent. Ce type de système permet d'étudier les processus cognitifs des sujets, fournissant implicitement la pertinence des patrons de conception via l'effort effectué.

D'autres articles vantent les avantages de la technologie d'*eye-tracking*. Cutrell *et al.* expliquent que la technologie d'*eye-tracking* permet de fournir bien plus d'observations détaillées moment par moment sur la manière dont les utilisateurs interagissent avec les informations [Cutrell 07].

Rayner ainsi que Yusuf *et al.* expliquent tous deux que la technologie d'*eye-tracking* devient de plus en plus populaire suite aux nombreuses avancées technologiques dans le domaine [Rayner 98, Yusuf 07]. Les systèmes disponibles actuellement offrent une haute qualité, une extrême précision et une certaine convivialité. De plus, ces systèmes sont relativement faciles à utiliser et permettent de capturer les mouvements des yeux des sujets de manière non-intrusive [Yusuf 07]. Cet aspect de «*précision*» assure une partie de la validité interne de l'expérience (voir Chapitre 5, Section 5.6.1).

Un autre avantage des systèmes d'*eye-tracking* est qu'il ne requiert pas d'entraînement particulier pour les participants à l'expérience [Bednarik 06].

4.3.5 Inconvénients

L'utilisation de systèmes d'*eye-tracking* possède également des inconvénients.

Actuellement, sous réserve, seul l'article de Johansen *et al.* présente certaines critiques des systèmes d'*eye-tracking* [Johansen 06]. Celui-ci énonce que l'analyse des données prend énormément de temps, même en utilisant les logiciels fournis avec les systèmes d'*eye-tracking*. De plus, les systèmes d'*eye-tracking* peuvent constituer une menace à la validité en ralentissant, par exemple, le système de réponse ou en obligeant les utilisateurs à re-calibrer l'appareil entre chaque tâche [Johansen 06].

Finalement, l'inconvénient majeur des systèmes d'*eye-tracking* est le manque de définitions et d'évaluations de mesures appropriées [Goldberg 99]. Goldberg *et al.* ont présentés toute une série de mesures basées sur la localisation des yeux et sur les chemins d'analyse. Ces mesures ont également été testées par les mêmes auteurs afin d'évaluer la validité de celles-ci lors de l'évaluation de la qualité d'une interface [Goldberg 99].

Les problèmes énoncés par Johansen *et al.* semblent être pertinents étant donné qu'ils se sont présentés lors de l'exécution de l'expérience présentée au Chapitre 5. Ces problèmes seront détaillés plus amplement à travers les Sections 5.6.1 et 6.4.3.

5 Expérience réalisée

Sommaire

Avant-propos

Définir sa propre expérience n'est pas une sinécure. C'est pourquoi ce chapitre se consacre exclusivement à l'explication détaillée des différentes étapes composant cette expérience.

Un résumé des diverses tâches réalisées est mis à disposition du lecteur pour chacune des étapes du processus, et ce, à travers la section ***Introduction****. Une section est ensuite consacrée entièrement à chaque étape de l'expérience afin d'expliquer plus en détails le travail réalisé.*

5.1 Introduction

Ce chapitre présente la mise en application des quatre étapes du processus d'une expérience (voir Chapitre 2) dans le but d'étudier les points présentés au Chapitre 3. L'articulation de ce chapitre se présente de la manière suivante. La Section 5.1.1 décrit sommairement chacune des étapes de l'expérience réalisée. Les motivations des choix n'y sont cependant pas présentées afin d'éviter toute répétition par rapport aux Sections 5.2–5.5. Chacune des étapes fait également l'objet d'une section à part entière afin de présenter le travail réalisé de manière plus détaillée d'une part, et de présenter les motivations des choix opérés au niveau de la conception d'autre part.

5.1.1 Définition

Conformément au canevas (voir Section 2.2) généralement utilisé, la définition de cette expérience se présente de la manière suivante :

> Analyse de <***Trois logiciels open-source (JHotDraw, JRefactory, PADL)***>
> dans le but de <***Déterminer si les patrons de conceptions sont utiles lors de la compréhension et la maintenance de logiciels***>
> selon l' <***Effort réalisé***>
> sous l'angle du <***Développeur, expert en maintenance***>
> dans le contexte des <***Laboratoires du D.I.R.O, avec la présence d'étudiants jouant le rôle de sujets***>

5.1.2 Planification

Formulation des hypothèses

Le but de cette expérience est d'évaluer quatre hypothèses nulles, deux pour la compréhension (C) et deux pour la maintenance (M) :

- HC_{0_1} : Un diagramme de classes avec le patron de conception «*Visiteur*» n'aide pas les sujets lors de la compréhension de programmes en comparaison avec un diagramme de classes sans le patron de conception «*Visiteur*».

- HC_{0_2} : Un diagramme de classes utilisant la représentation canonique du patron de conception «*Visiteur*» n'aide pas les sujets lors de la compréhension de programmes par rapport à un diagramme de classes identique, utilisant un layout différent.

- HM_{0_1} : Un diagramme de classes avec le patron de conception «*Visiteur*» n'aide pas les sujets lors de la maintenance de programmes en comparaison avec un diagramme de classes sans le patron de conception «*Visiteur*».

- HM_{0_2} : Un diagramme de classes utilisant la représentation canonique du patron de conception «*Visiteur*» n'aide pas les sujets lors de la maintenance de programmes par rapport à un diagramme de classes identique, utilisant un layout différent.

Si ces hypothèses nulles sont rejetées, les hypothèses alternatives suivantes seront vérifiées (si la validité de l'expérience le permet) :

- HC_{a_1} : Un diagramme de classes avec le patron de conception «*Visiteur*» aide les sujets lors de la compréhension de programmes.
- HC_{a_2} : Un diagramme de classes utilisant la représentation canonique du patron de conception «*Visiteur*» aide les sujets lors de la compréhension de programmes.
- HM_{a_1} : Un diagramme de classes avec le patron de conception «*Visiteur*» aide les sujets lors de la maintenance de programmes.
- HM_{a_2} : Un diagramme de classes utilisant la représentation canonique du patron de conception «*Visiteur*» aide les sujets lors de la maintenance de programmes.

Sélection des variables indépendantes

Suite aux hypothèses précédentes, les variables indépendantes suivantes ont été identifiées :

- **Alternative de Design** : CP, MP, et NP sont les valeurs que peut prendre cette variable suite aux trois versions réalisées pour chaque système. Chaque version d'un système possède la même sémantique que les autres. Un diagramme ne contenait pas de patron de conception : NP pour «*No Pattern*». Un diagramme utilisait le patron de conception sous sa représentation canonique : CP pour «*Canonical Pattern*». Le dernier diagramme utilisait le patron de conception sous une représentation modifiée : MP pour «*Modified Pattern*».

- **Tâche** : Deux tâches ont été comparées, la tâche de compréhension (C) de programmes et celle de maintenance (M) de programmes.

Ces variables ci-dessus représentent les deux variables principales de l'expérience réalisée. Cependant, deux autres variables pouvant mitiger les résultats ont également été identifiées :

- **Connaissance UML** : La connaissance des sujets en UML. Cette connaissance était évaluée via un questionnaire à la fin de chaque expérience.

- **Connaissance DP** : La connaissance des sujets en **Design Patterns**. Cette connaissance était évaluée de la même manière que celle en UML.

Sélection des variables dépendantes

La variable dépendante a été choisie en fonction les informations primitives fournies par le système d'*eye-tracking* utilisé. Cette variable a également été choisie sur base des hypothèses formulées et du choix des variables indépendantes. Il s'agit de :

- **Normalized Rate of Relevant Fixations** (NRRF) : L'objectif est d'identifier le rapport qui existe entre le nombre de fixations réalisées dans les classes pertinentes et le nombre total de fixations réalisées par sujet pour chaque question. La pertinence d'une classe est déterminée à l'avance. Pour être pertinente, une classe doit contenir l'information ou une partie de l'information permettant de répondre à la question posée (nom de classe, attribut, méthode). Etant donné que les alternatives de design (CP, MP, et NP) n'ont pas toutes le même nombre de classes, une normalisation (Normalized) du nombre total de fixations pertinentes et non pertinentes a été réalisée respectivement, sur base du nombre de classes pertinentes et non pertinentes présentes dans le diagramme. La variable NRRF est calculée de la manière suivante :

$$\frac{\frac{\sum_{c\in\{Classes\ Pertinentes\}} F(c)}{\#\{Classes\ Pertinentes\}}}{\frac{\sum_{c\in\{Classes\ Pertinentes\}} F(c)}{\#\{Classes\ Pertinentes\}} + \frac{\sum_{c\in\{Classes\ Non\ Pertinentes\}} G(c)}{\#\{Classes\ Non\ Pertinentes\}}}$$

où $F(c)$ et $G(c)$ sont des fonctions F et G qui renvoient respectivement, le nombre de fixations pertinentes et le nombre de fixations non pertinentes d'une classe c, présentes dans un diagramme.

Cette variable a été utilisée car elle permet de représenter l'«*effort*» d'un sujet. L'interprétation du concept d'*effort* considérée dans cette expérience est la suivante : «*Au plus le taux de fixations pertinentes est élevé, au moins le sujet fait d'effort*».

Sélection des sujets

Cette expérience a été réalisée avec 25 sujets dont un non considéré : 7 étudiants post-gradués[1] et 17 étudiants gradués[2] au D.I.R.O. à l'Université de Montréal. Les 24 étudiants ont été répartis dans 3 groupes équilibrés, suite à leur demande volontaire de participation.

Conception de l'expérience

Cette phase permet de construire l'expérience ainsi que tous les matériaux nécessaires lors de l'exécution de cette expérience. De nombreuses réflexions ont également eu lieu concernant les choix de conception et les biais que ces choix pouvaient introduire.

Ainsi, les activités suivantes ont été réalisées durant cette phase de conception :

- **Définir les principes de conception à utiliser** : randomisation et équilibrage.
- **Identifier le type de conception à utiliser** : un facteur avec deux et plus de deux traitements, conception complètement aléatoire.
- **Concevoir les systèmes/diagrammes** : rétro-ingénierie partielle de trois systèmes open-source afin d'obtenir pour chacun d'entre eux trois versions de diagrammes des classes UML (No-Pattern, Classical-Pattern, Modified-Pattern), équivalentes au niveau pragmatique.
- **Concevoir les questions** : une question de compréhension et de maintenance pour chacun des trois systèmes, soit six questions au total.
- **Concevoir des matériaux** : *check-lists* afin de vérifier l'exactitude des réponses des sujets lors des expériences, affiche de recrutement, site web pour enregistrer la participation des clients et planifier les heures de rendez-vous.

La validité de l'expérience fait normalement partie de la conception de l'expérience. Cependant, elle sera détaillée ultérieurement, voir Section 5.6, afin de permettre au lecteur de développer sa critique suite à la présentation des étapes d'*exécution* et des *analyses de données*.

[1] Etudiants possédant au moins un diplôme de maîtrise en sciences informatiques
[2] Etudiants possédant au moins un diplôme de bachelier en sciences informatiques

5.1.3 Exécution

L'exécution des expériences a débuté avec l'étape de recrutement des sujets. Pour ce faire, des affiches ont été posées dans le département du D.I.R.O. et un site web a été mis à la disposition des personnes intéressées par l'expérience.

Après s'être inscrits, les sujets passaient l'expérience pendant environ une heure. Cette heure incluait la présentation d'un didacticiel (15–20 minutes), la présentation du système d'*eye-tracking* (5 minutes), la collecte des données (20–30 minutes), et un questionnaire d'évaluation des niveaux de connaissances notamment en UML et en patrons de conception (5 minutes).

Le didacticiel servait à rappeler les différentes constructions (classe, attribut, opération, type d'associations) des diagrammes de classes UML. Il servait également à simuler l'expérience afin de présenter le type de questions auxquels les sujets allaient devoir répondre. La présentation du système d'*eye-tracking* permettait d'expliquer le fonctionnement du système et de limiter le stress, l'inquiétude des sujets vis-à-vis d'un tel système. Durant la collecte des données, les sujets devaient répondre à six questions, soit deux questions pour chacun des trois systèmes. Une fois cette collecte de données terminée, le sujet était invité à répondre à un questionnaire dans lequel il lui était notamment demandé d'évaluer ses propres connaissances en UML et en patrons de conception. Ce questionnaire permettait de collecter quelques informations supplémentaires, potentiellement utiles lors des futures analyses de données.

5.1.4 Analyses et interprétations

Des *t-tests* de *Student* ont été réalisés pour les tâches de compréhension (Tableau 5.1) et de maintenance (Tableau 5.2).

	CP_C	MP_C	NP_C	p-value (HC_{01})	p-value (HC_{02})
NRRF (%)	75,77	77,95	81,49	0.221	0.663

Tab. 5.1 – Effet du «*Visiteur*» sur la Compréhension

Les *p-values* ressortant des *t-tests* pour la compréhension ne sont pas significatives pour les hypothèses HC_{01} et HC_{02}, voir Tableau 5.1. Il est impossible de conclure quoi que ce soit.

	CP_M	MP_M	NP_M	p-value (HM_{01})	p-value (HM_{02})
NRRF (%)	71,97	73,46	80,18	0.027	0.725

Tab. 5.2 – Effet du «*Visiteur*» sur la Maintenance

En revanche, pour la maintenance, la *p-value* reprise pour l'hypothèse HM_{01} indique que le *t-test* est significatif en faveur de l'alternative de design NP_C, *i.e.* le taux de fixations pertinentes est plus élevé. Un diagramme sans «*Visiteur*» semble être plus adapté pour effectuer des tâches de maintenance.

5.2 Définition

L'idée ayant conduit à la réalisation d'une expérience reste souvent quelque chose d'abstrait, de non formel. Définir l'expérience de manière plus formelle va permettre de déterminer les fondements de celle-ci [Wohlin 00]. A la fin de cette étape, l'expérimentateur doit avoir formulé la définition de l'expérience et doit posséder des éléments de réponse sur la manière concrète de réaliser l'expérience. Pour cela, il est généralement fait usage d'un canevas qui assurera la bonne définition des différents aspects importants avant les phases de planification et d'exécution [Wohlin 00].

L'expérience réalisée peut se définir de la manière suivante :

> Analyse du ou des <*Objet(s) d'étude*>
> dans le but de <*Objectif*>
> selon la <*Focalisation sur la qualité*>
> sous l'angle de <*Perspective*>
> dans le contexte du <*Contexte*>

Objets d'étude : Trois systèmes open-source. Il s'agit de JHotDraw, JRefactory, et de PADL. Une description de ces différents systèmes est donnée en Page 79.

Objectif : Déterminer si les patrons de conception sont utiles lors de la compréhension et de la maintenance de logiciels orientés-objet.

Focalisation sur la qualité : L'«*effort*» réalisé. Cet effort peut se calculer de plusieurs façons. Ce concept fera l'objet d'une discussion dans la Section 5.3.3.

Perspective : Le développeur ou l'expert en maintenance.

Contexte : Département d'Informatique et de Recherche Opérationnelle (D.I.R.O.), Université de Montréal, Montréal, Canada. Le groupe de sujets était composé exclusivement d'étudiants.

Remarque :

L'expérience présentée à travers ce chapitre représente l'intégralité d'un travail réalisé dans le cadre d'un stage. Etant donné que la définition de l'expérience s'apparente aux objectifs du stage, aucun débat n'a eu lieu sur cette définition. Cette étape n'a présenté aucun problème notable, et les choix opérés appartiennent plus aux maîtres de ce stage.

5.3 Planification

Alors que la phase de *définition* présentait *pourquoi* l'expérience était réalisée, la phase de *planification* s'attelle à définir *comment* l'expérience va être réalisée [Wohlin 00].

Cette section présente les différents choix qui ont été opérés ainsi que les motivations de ces choix. Il y sera présenté successivement la *sélection du contexte*, la *formulation des hypothèses*, la *sélection des variables* ainsi que *des sujets* et la *conception de l'expérience* à proprement parler.

5.3.1 Sélection du contexte

Le but ultime d'une expérience serait d'obtenir des résultats généraux suite à la réalisation d'un grand nombre de projets de taille réelle, impliquant des professionnels [Wohlin 00]. Cependant, il est très difficile de réaliser de telles expériences suite aux coûts qu'elles engendrent. Afin de réduire ces coûts, l'expérimentateur doit généralement restreindre la portée de son expérience.

L'expérience réalisée au D.I.R.O. est également passée par une phase de sélection du contexte. Cependant la plupart des choix des critères étaient implicites : cette expérience était l'objet d'un stage prédéfini. Ces choix seront présentés et débattus ci-après.

Pour caractériser la sélection du contexte d'une expérience, Wohlin définit quatre critères (voir Section 2.3.1) [Wohlin 00]. Dans le cas de cette expérience, la sélection du contexte peut se caractériser, en fonction des quatre critères fournis par Wohlin, de la manière suivante :

- HORS LIGNE vs. en ligne
- ETUDIANT vs. professionnel
- Problème de petite taille vs. PROBLÈME RÉEL
- SPÉCIFIQUE vs. général

Les paragraphes suivants décrivent plus précisément ce que représentent les valeurs de ces quatre critères appliqués à l'expérience réalisée.

HORS LIGNE

Définition : Une expérience se déroulant de manière «*en ligne*» signifie qu'elle se déroule dans le monde réel, sous des conditions habituelles. A l'opposé, une expérience dite «*hors ligne*» signifie qu'elle se déroule dans un laboratoire, par exemple, dans lequel les événements sont agencés de manière à simuler le monde réel [Wohlin 00].

L'aspect «*en ligne*» ne peut être appliqué à toute expérience [Wohlin 00]. Cette expérience en est typiquement un exemple. Qui dit «*en ligne*», dit utilisation des différentes ressources lors de tâches quotidiennes. Il ne s'agit pas de simuler une situation réelle. Une expérience «*en ligne*» s'effectue durant une situation réelle quotidienne. Par conséquent, le concept d'expérience «*en ligne*» est incompatible avec le matériel choisi lors de cette expérience. En effet, l'utilisation du système d'*eye tracking* choisi ne fait pas partie du quotidien de professionnels. Etant donné son poids mais aussi sa taille, ce type d'appareil ne permet pas à une personne de travailler continuellement de manière efficace. L'utilisation de ce matériel

ne va pas sans introduire certains biais. De tels systèmes sont généralement utilisés à des fins de recherche scientifique et n'ont pas leur place au sein d'entreprises.

Un autre facteur ayant conduit au choix d'une expérience «*hors ligne*» est le cadre dans lequel s'inscrit cette expérience. Cette dernière s'inscrit dans le cadre d'un stage et ne laisse pas place à beaucoup de libertés, notamment au niveau du temps. Ceci limite les marges de manoeuvre. En effet, réaliser une expérience «*en ligne*» nécessite la mise en place de celle-ci au sein d'une activité. A titre d'exemple, cette activité peut correspondre à un examen ou à un travail de groupe si l'expérience se déroule avec des étudiants. Cette activité s'apparente plus à une tâche au sein d'une entreprise si l'expérience se déroule avec des professionnels.

Dans un cas comme dans l'autre, l'expérimentateur doit négocier préalablement avec les responsables, professeurs ou patrons d'entreprise, la mise en place de cette expérience. Excepté le fait que les négociations avec les responsables prennent un certain temps, les activités connexes à l'expérience requièrent également beaucoup de temps. Si l'expérience présentée à travers ce chapitre avait dû être réalisée «*en ligne*», il aurait fallu (1) s'approprier le domaine, (2) concevoir l'expérience tout en négociant avec les responsables, (3) installer l'expérience au sein d'une activité, (4) faire passer les expériences et finalement (5) analyser les résultats.

Toutes ces activités auraient pris trop de temps par rapport aux trois mois et demi qui étaient impartis. Une expérience «*en ligne*» requiert plus de temps par rapport à une expérience «*hors ligne*». Ce temps supplémentaire s'explique également par la réflexion qui doit être opérée sur l'intégration d'une telle expérience au sein d'une activité et l'introduction des éventuels biais qu'elle pourrait engendrer.

Ces deux facteurs sont principalement les raisons pour lesquelles cette expérience a été réalisée de manière «*hors ligne*». Il faut cependant noter que ces deux facteurs ont plus imposé l'utilisation du mode «*hors ligne*» que donné l'opportunité de choisir son propre mode.

Etudiant

Définition : Ce critère définit le type de sujets utilisés durant l'expérience.

Le choix de la nature des participants est une tâche assez délicate. La question des participants est au coeur de bon nombre de débats concernant la pertinence d'étudiants lors d'expériences, et les menaces à la validité que ceux-ci peuvent engendrer. Etant donné que les expériences se veulent aussi solides que possible, scientifiquement parlant, elles recherchent également les sujets les plus représentatifs de la population étudiée. L'idée à première vue, et partagée par beaucoup de gens [Sjoberg 07], est d'utiliser des professionnels, de véritables praticiens lors d'expériences afin d'obtenir des résultats les plus pertinents possibles.

Cependant, une étude a montré que 90% des sujets prenant part aux expériences sont en réalité des étudiants [Sjoberg 05]. L'important est que la compétence et le niveau d'expertise des sujets, relatifs à la technologie étudiée soient mentionnés explicitement. Ceci permettra d'indiquer la population à laquelle les résultats s'adressent [Sjoberg 07]. Jakob Nielsen va même jusqu'à dire qu'un échantillon de cinq personnes serait suffisant pour assurer la pertinence des données [Nielsen 00].

Il est important de garder à l'esprit que la réalisation de cette expérience-ci s'est effectuée

dans des laboratoires universitaires. L'utilisation de professionnels s'en trouve d'autant plus difficile étant donné qu'il faut faire venir ces professionnels à l'université. C'est la raison principale pour laquelle cette expérience a fait appel à des étudiants plutôt que des professionnels. Les étudiants sont une ressource bien moins coûteuse et bien plus abondante dans un milieu universitaire. De plus, le temps «*hors expérience*» s'en trouve très fortement réduit étant donné que les étudiants travaillent à proximité. Ce facteur «*perte de temps*» est un élément à prendre en considération lors du recrutement de sujets. Moins les sujets devront passer du temps hors expérience, au plus il sera facile de les inciter à passer l'expérience.

Un autre facteur lié à la localisation du matériel, est qu'il est plus difficile de faire passer les expériences en masse avec des professionnels étant donné que ces personnes ont un emploi du temps relativement chargé. De la même manière, faire passer des expériences requiert des négociations qui auraient dû être réalisées antérieurement au début du stage.

L'implantation du matériel au sein d'un milieu universitaire a fait pencher le choix vers l'utilisation d'*étudiants* pour les deux raisons citées précédemment, à savoir la **forte présence** de sujets potentiels et la **distance** qui séparait les professionnels de l'université. L'absence de négociations préalables au stage avec des professionnels peut également avoir conduit implicitement au choix des étudiants comme sujets.

Problème Réel

Définition : Ce critère concerne la nature du problème traité. Les *problèmes réels* traitent de véritables problèmes du génie logiciel ou du monde industriel contrairement aux *problèmes de petite taille*.

La question de la taille du problème est assez délicate dans le cas de cette expérience. Certains éléments de l'expérience tendent à démontrer qu'un réel problème a été traité. D'autres éléments montrent clairement les limites de l'expérience, la caractérisant plutôt comme une étude sur un problème de petite taille.

Le problème traité dans cette expérience reste cependant un problème de taille réelle. En effet, évaluer l'impact d'un patron de conception au niveau de la compréhension et de la maintenance est un sujet pertinent aussi bien dans le monde académique que professionnel. Les patrons de conceptions sont présents dans relativement beaucoup de programmes orientés-objets [Wendorff 01]. De même, la compréhension et la maintenance sont des activités essentielles et récurrentes dans le cycle de vie des logiciels. Même si cette expérience ne s'est focalisée uniquement que sur le patron de conception «*Visiteur*», les objectifs restent cependant suffisamment généraux pour s'adresser à un problème réel. De plus, le patron du «*Visiteur*» fait partie des patrons les plus couramment utilisés, aussi bien dans le monde industriel qu'académique [Wendorff 01, Prechelt 01, Vokac 04].

A l'opposé, cette expérience ne s'est focalisée que sur des systèmes de petite taille. Seule une partie de ces systèmes a été rétro-construite, ce qui ne permet pas de dire que l'expérience a entièrement traité un problème réel. De plus, les systèmes utilisés sont généralement issus du monde académique et témoignent de systèmes aux choix architecturaux étudiés. Ces systèmes contrastent avec les différents systèmes présents dans le monde professionnel dans lesquels de mauvais choix architecturaux ainsi que des dégradations de conception peuvent être identifiés.

Tous ces éléments font que cette expérience se trouve à la limite entre un problème *réel* et un problème *de petite taille*.

Spécifique

Définition : Ce critère caractérise le degré de validité d'une expérience. Soit l'expérience est valide pour un contexte *spécifique*, soit elle est valide pour le domaine du génie logiciel en *général*.

Conformément à la problématique spécifiée à travers le Chapitre 3, cette expérience s'inscrit dans une problématique plus large visant à étudier la pertinence des patrons de conception dans les programmes orientés-objet. Cette expérience traite par conséquent d'un problème spécifique dans lequel un patron de conception (*Visiteur*) a été étudié dans un contexte de compréhension et de maintenance de programmes.

Les motivations des choix opérés à propos de la problématique de l'expérience ont été présentés précédemment à travers le Chapitre 3.

5.3.2 Formulation des hypothèses

La formulation des hypothèses est une étape clé dans la construction d'une expérience. Cette formulation va définir plus précisément les grands axes de cette expérience. C'est à partir de ce moment-là que les objectifs deviennent clairs, vus de l'extérieur.

Le but ici est de préciser les hypothèses nulles, c'est-à-dire les affirmations qui sont testées lors de l'expérience. Si ces hypothèses nulles sont rejetées, en d'autres termes si l'expérience prouve que ces hypothèses ne sont pas valides, les hypothèses alternatives sont choisies. Généralement les hypothèses alternatives sont la négation des hypothèses nulles.

Dans cette expérience, quatre hypothèses nulles ont été formulées. De manière complémentaire, quatre hypothèses alternatives ont également été formulées. Suite aux objectifs globaux fixés lors de la phase de *définition*, les quatre hypothèses formulées ont été divisées en deux groupes. Deux hypothèses concernent la *compréhension* (HC) et deux hypothèses concernent la *maintenance* (HM).

Pour rappel, les hypothèses nulles se présentent de la manière suivante :

- HC_{0_1} : Un diagramme de classes avec le patron de conception «*Visiteur*» n'aide pas les sujets lors de la compréhension de programmes en comparaison avec un diagramme de classes sans le patron de conception «*Visiteur*».
- HC_{0_2} : Un diagramme de classes utilisant la représentation canonique du patron de conception «*Visiteur*» n'aide pas les sujets lors de la compréhension de programmes par rapport à un diagramme de classes utilisant le patron de conception «*Visiteur*» avec un layout différent.
- HM_{0_1} : Un diagramme de classes avec le patron de conception «*Visiteur*» n'aide pas les sujets lors de la maintenance de programmes en comparaison avec un diagramme de classes sans le patron de conception «*Visiteur*».
- HM_{0_2} : Un diagramme de classes utilisant la représentation canonique du patron de conception «*Visiteur*» n'aide pas les sujets lors de la maintenance de programmes par rapport à un diagramme de classes utilisant le patron de conception «*Visiteur*» avec un layout différent.

Si ces hypothèses nulles sont rejetées, les hypothèses alternatives suivantes seront vérifiées (si la validité de l'expérience le permet) :

- HC_{a_1} : Un diagramme de classes avec le patron de conception «*Visiteur*» aide les sujets lors de la compréhension de programmes.
- HC_{a_2} : Un diagramme de classes utilisant la représentation canonique du patron de conception «*Visiteur*» aide les sujets lors de la compréhension de programmes.
- HM_{a_1} : Un diagramme de classes avec le patron de conception «*Visiteur*» aide les sujets lors de la maintenance de programmes.
- HM_{a_2} : Un diagramme de classes utilisant la représentation canonique du patron de conception «*Visiteur*» aide les sujets lors de la maintenance de programmes.

Comme il a été énoncé en début de section, la formulation des hypothèses énonce plus clairement les axes d'une expérience. Dans le cadre de cette expérience, les objectifs fixés ont donné lieu à ces quatre hypothèses nulles. Il était nécessaire de formuler deux hypothèses pour chaque type de TÂCHE (compréhension et maintenance) afin de pouvoir réaliser des analyses statistiques différentes. De la même manière, deux hypothèses concernant le patron de conception «*Visiteur*» ont été formulées pour chaque type de «TÂCHE» afin d'isoler les effets. Cette notion de TÂCHE constitue une des *variables indépendantes* et sera présentée en Section 5.3.3 ci-dessous.

5.3.3 Sélection des variables

La sélection des variables et leur type d'échelle vont déterminer le type d'analyses qu'il sera possible d'effectuer. Ces analyses seront effectuées sur les données collectées lors de l'expérience. C'est pourquoi, il est important de choisir adéquatement des variables afin de pouvoir réaliser des analyses solides et riches en interprétations. De mauvaises variables mettront la validité de l'expérience à mal et ne pourront être considérées comme pertinentes pour réfuter les hypothèses formulées [Wohlin 00].

Pour rappel, il existe deux types de variables : les variables **dépendantes** et les variables **indépendantes**. Les variables indépendantes correspondent aux variables qui peuvent avoir de l'influence sur les variables dépendantes. Elles correspondent aussi aux variables que l'expérimentateur peut contrôler. Les variables dépendantes, quant à elles, sont les variables influencées.

Variables indépendantes

A partir des hypothèses précédemment formulées, les variables indépendantes suivantes ont été identifiées :

- **ALTERNATIVE DE DESIGN** : CP, MP, et NP sont les valeurs attribuées à cette variable, suite aux trois versions qui avaient été réalisées pour chacun des trois systèmes. Chaque version d'un système possédait la même sémantique que les autres. Un diagramme ne contenait pas de patrons de conception : NP pour «*No Pattern*». Un diagramme utilisait le patron de conception sous sa représentation canonique : CP pour «*Canonical Pattern*». Le dernier diagramme utilisait le patron de conception sous une représentation modifiée : MP pour «*Modified Pattern*».

- **TÂCHE** : Deux tâches ont été comparées, la tâche de compréhension de programmes et celle de maintenance de programmes : C ou M.

Ces variables ci-dessus représentent les deux variables principales de l'expérience réalisée. Cependant, deux autres variables pouvant mitiger les résultats ont été identifiées :

- **Connaissance UML** : La connaissance des sujets en UML. Ce niveau de connaissance était évalué via un questionnaire à la fin de chaque expérience. Les valeurs étaient prises dans un ensemble $\{1, 2\}$ où 2 signifiait que le sujet avait un très bon niveau de connaissance en UML et 1 signifiait qu'il connaissait UML.
- **Connaissance DP** : La connaissance des sujets en **Design Patterns**. Cette connaissance était évaluée de la même manière que la connaissance en UML.

Sur base des hypothèses nulles formulées, les deux variables indépendantes «Alternative de Design» et «Tâche» ont été choisies. Les valeurs que prennent ces deux variables modélisent chacune une partie des hypothèses formulées.

De façon intuitive, deux variables supplémentaires ont été choisies afin d'étudier certains impacts éventuels. La Connaissance UML et Connaissance DP sont des variables dont les valeurs ont été tirées du questionnaire rempli par chaque participant à la fin de l'expérience, voir Section 5.4.2 et Annexes I. Le but ici était d'évaluer s'il existait réellement une différence de résultats en fonction du niveau de connaissance. Les résultats sur ces variables seront présentés à travers la Section 5.5.3.

Variables dépendantes

La variable dépendante a été choisie selon les informations primitives fournies par le système d'*eye-tracking* utilisé. Cette variable a également été choisie sur base des hypothèses formulées et du choix des variables indépendantes. Néanmoins, la sélection actuelle des variables dépendantes est le fruit de nombreuses discussions. Ces dernières sont présentées au lecteur afin de lui montrer les arguments qui ont conduit à rejeter certaines variables, et ceux qui ont conduit à conserver la variable actuelle.

Démarche A :

Cette démarche, non retenue au final, consistait à caractériser l'*effort* en fonction du «*temps global*» de chaque tâche. Cette variable n'a pas été retenue car elle semblait être un indicateur trop grossier. La manifestation de certains facteurs explicatifs pouvait être neutralisée. Deux exemples illustrent les craintes à propos de cette variable :

a) Pour un temps global égal, deux personnes peuvent passer des temps différents dans des classes différentes. Une personne peut passer tout son temps dans les classes pertinentes tandis que l'autre peut passer tout son temps dans les classes non pertinentes. Cependant, ces deux personnes ont le même temps global. C'est pourquoi, les concepts de «*temps pertinent*» et de «*temps non pertinent*» ont été introduits.

b) Pour un temps global égal, deux personnes peuvent avoir réalisé plus ou moins d'effort dans les classes pertinentes ou les classes en général. C'est pourquoi le concept de «*fixation*» doit être relié d'une certaine manière aux variables dépendantes.

Démarche B :

Suite au caractère trop grossier de la démarche précédente, une nouvelle variable a été considérée afin d'interpréter plus finement les résultats obtenus lors des expériences :

- **Normalized Rate of Relevant Fixations** (NRRF) : L'objectif est d'identifier le rapport qui existe entre le nombre de fixations réalisées dans les classes pertinentes et le nombre total de fixations réalisées par sujet pour chaque question. La pertinence d'une classe est déterminée à l'avance. Pour être pertinente, une classe doit contenir l'information ou une partie de l'information permettant de répondre à la question posée (nom de classe, attribut, méthode). Etant donné que les alternatives de design (CP, MP, et NP) n'ont pas toutes le même nombre de classes, une normalisation (Normalized) du nombre total de fixations pertinentes et non pertinentes a été réalisée sur base respectivement, du nombre de classes pertinentes et non pertinentes présentes dans le diagramme. La variable NRRF est calculée de la manière suivante :

$$\frac{\frac{\sum_{c\in\{Classes\ Pertinentes\}} F(c)}{\#\{Classes\ Pertinentes\}}}{\frac{\sum_{c\in\{Classes\ Pertinentes\}} F(c)}{\#\{Classes\ Pertinentes\}} + \frac{\sum_{c\in\{Classes\ Non\ Pertinentes\}} G(c)}{\#\{Classes\ Non\ Pertinentes\}}}$$

  où $F(c)$ et $G(c)$ sont des fonctions F et G qui renvoient respectivement, le nombre de fixations pertinentes et le nombre de fixations non pertinentes d'une classe c, présente dans un diagramme. Cette variable permet de représenter l'«*effort*». Un NRRF élevé correspond à un sujet réalisant peu d'effort et un NRRF faible correspond à un sujet réalisant beaucoup d'effort.

Autres démarches :

Malheureusement, il n'est pas possible de garantir que la variable retenue permette d'exprimer au mieux ce qui est étudié. Suite à manque de connaissance dans le domaine de la compréhension de programmes, nous n'arrivons pas à critiquer objectivement et, par conséquent, à trouver une variable permettant de refléter l'effort fourni par les sujets.

Cependant, malgré ce manque de connaissance, plusieurs variables, intuitives ou plus réfléchies sur base d'articles lus, ont été proposées. Elles n'ont toutefois pas été retenues. Voici succinctement les variables et les réflexions opérées sur celles-ci :

- **Nb Relevant Fixations** : Ensemble des fixations réalisées dans les classes pertinentes. L'interprétation généralement adoptée est qu'un *petit nombre de fixations signifie peu d'effort.*
  Problème(s) : Qu'en est-il si deux sujets avec un même nombre de fixations pertinentes effectuent un nombre de fixations total différent ? Qu'en est-il si deux sujets effectuent un même nombre de fixations pertinentes et un même nombre total de fixations, mais avec un nombre de classes différent ?
- **Average Nb Relevant Fixations** : Nb Relevant Fixations divisé par le nombre de classes pertinentes.
  Problème(s) : Un diagramme avec 10 classes est 10 fois meilleur qu'un diagramme avec 100 classes. Sur quoi se baser pour prouver une telle affirmation ? Il n'y a aucun argument fondé pour avancer cela.
- **Average Duration Relevant Fixations** : Représente le temps consacré à visualiser les classes pertinentes, divisé par le nombre de classes pertinentes.
  Problème(s) : Malheureusement, il n'est pas possible de savoir si peu de temps pertinent doit être considéré comme positif et beaucoup de temps comme négatif, ou l'inverse. De plus, suite à un test opéré, il s'est avéré que le temps et le nombre de fixations étaient corrélés.

- **Rate Relevant Fixations** : Nb Rel. Fixations / Nb Total Fixations
  Problème(s) *:* Cette mesure est assez similaire à celle retenue actuellement. Il y a cependant deux différences. La première se trouve au dénominateur. Au lieu de ne considérer que les fixations réalisées dans les classes, cette variable considère également les fixations réalisées dans le blanc du schéma. Ces fixations ont été supprimées dans la variable considérée actuellement car elles ne présentent pas d'information pertinente. Elles expriment éventuellement un moment de réflexion au niveau du sujet, ce qui n'est pas pertinent dans le cas de cette expérience. La deuxième différence réside dans le fait que les fixations au numérateur et au dénominateur ne sont pas *normalisées*. Cela peut poser problème lors de l'interprétation de résultats où le nombre de classes varie assez fortement.

La sélection des variables dépendantes a relativement posé problème car les choix initiaux ont été influencés par certains articles scientifiques. Ces derniers s'accordent sur la relation entre l'*eye-tracking* et leurs variables afin de caractériser la *complexité* ou l'*effort* réalisé [Bednarik 06, Yusuf 07, Pan 07]. L'interprétation retenue par ces auteurs à propos de la notion d'«*effort*» est la suivante : «*Moins il y a de fixation et au plus le temps est petit, au moins le sujet fait d'effort*».

Yusuf *et al.* explique, dans [Yusuf 07], la raison pour laquelle il ne définit aucune de ses variables sur base du temps de lecture. Yusuf explique que chaque lecteur a une vitesse de lecture différente et qu'il n'est par conséquent pas pertinent de se baser sur le temps d'une expérience pour calculer l'effort. Dans le cas de cette expérience, deux éléments permettent de baser les analyses sur la notion de temps. Le première élément est la *normalisation* des données (Normalized) qui permet de lisser un éventuel effet auprès d'un sujet. Le deuxième élément est l'utilisation de groupes équilibrés (voir Page 77) permettant de lisser un ou plusieurs effets au sein d'un groupe [Wohlin 00].

Malgré cette apparition de consensus, les variables présentées par ces auteurs n'ont pas été retenues dans cette expérience car elles ne possédaient pas de réponses à certaines de nos interrogations.

Remarque :

La variable retenue et toutes celles non retenues ont un point commun. Ces variables sont basées sur le concept de «*fixations*». Ces dernières permettent de matérialiser à première vue l'effort opéré dans un raisonnement. Les «*scanpaths*» et «*saccades*» permettent plus de comprendre le raisonnement des sujets.

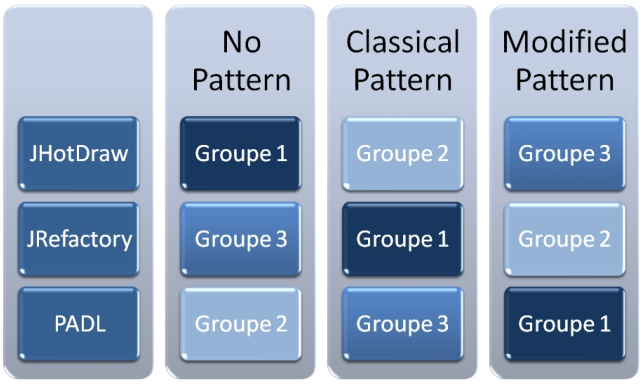
5.3.4 Sélection des sujets

L'échantillonnage reste une activité importante lors de la conduite d'expériences [Robson 93]. Etant donné que le type de sujets a été défini préalablement lors de la phase de *définition* de l'expérience, il ne reste plus qu'à trouver un échantillon suffisamment représentatif de la population analysée.

Cette expérience a été réalisée avec 25 sujets dont 24 retenus : 7 étudiants post-gradués[3] et 17(18) étudiants gradués[4] au D.I.R.O. à l'Université de Montréal. Les 24 étudiants ont

[3] Etudiants possédant au moins un diplôme de maîtrise en sciences informatiques
[4] Etudiants possédant au moins un diplôme de bachelier en sciences informatiques

été répartis dans 3 groupes équilibrés. La répartition est présentée à travers le Tableau 5.3. Un des 25 sujets a réalisé l'expérience mais, suite à certains problèmes techniques, les données n'ont pu être conservées pour les analyses statistiques, voir explications des problèmes rencontrés au Chapitre 6, Section 6.4.3.



TAB. 5.3 – Répartition des questions au sein des groupes

L'échantillonnage réalisé dans cette expérience suit une technique d'échantillonnage non probabiliste nommée «*échantillonage approprié*», voir Section 2.3.4. Etant donné que le nombre de sujets était relativement faible, toute personne satisfaisant aux pré requis a été acceptée comme sujet. Suite à cette faible population, l'utilisation de techniques probabilistes lors la constitution de l'échantillon aurait réduit encore plus le nombre de sujets (voir Section 2.3.4). Cependant, suivant J. Nielsen, calculer des mesures sur 20 personnes durant des expériences offre un intervalle de confiance relativement fort [Nielsen 06].

Une précision doit cependant être ajoutée. Les personnes acceptées en tant que sujets ont toutes suivi un cursus orienté vers l'informatique et disposent toutes des bases en programmation orientés-objet nécessaires à la réalisation des tâches abordées.

5.3.5 Conception de l'expérience

Les analyses de données et la conception de l'expérience sont fortement corrélées étant donné que les analyses se basent sur les données collectées. Or ces données sont collectées en fonction de la conception de l'expérience. Cette étape va décrire *comment* les tests de l'expérience vont être organisés et réalisés. Les objets, sujets et échelles de mesures seront également pris en compte pour définir le déroulement de l'expérience.

Principes de conception utilisés

Les principes retenus dans cette expérience sont les suivants :

- **Randomisation**. Cette technique a été utilisée à deux reprises.
  La première a été réalisée au niveau de l'ordre dans lequel les tests s'enchaînaient. Etant donné que les sujets devaient travailler aussi bien sur des systèmes différents (JHOTDRAW, JREFACTORY, PADL)[5] mais également sur des tâches différentes (C,M), la randomisation a été utilisée sur ces deux aspects.

[5] voir Section 5.3.5 – **Systèmes**, Page 79

L'ordre de passage des sujets a également fait usage d'une *semi randomisation*. Les sujets ont été répartis dans trois groupes distincts. Pour chaque groupe, une configuration de l'expérience a été définie, voir Section 5.3.5 – **Types de conception utilisés**. En fonction de l'heure et du jour que le sujet choisissait, toutes les personnes ayant choisi une date de passage ultérieure dans l'année s'en trouvaient décalées dans les groupes. L'objectif était d'avoir un nombre de sujets multiple de trois. Cela pouvait faciliter la répartition des sujets au sein des groupes. Avec un nombre multiple de trois, les groupes restaient équilibrés (voir point suivant). Pour rappel, la randomisation permet notamment de lisser l'effet d'un facteur [Wohlin 00].

- **Equilibrage**. Tel qu'il vient d'être présenté, les sujets ont été répartis au sein de trois groupes de manière équilibrée. Le but derrière cette action est de faciliter et de renforcer les analyses statistiques des données [Wohlin 00]. De plus, avec trois versions de diagrammes pour chacun des trois systèmes, soit neuf diagrammes, il était plus facile de composer trois groupes, chacun ayant sa propre «*configuration*». Cette dernière permettait l'utilisation d'une seule version de chaque système lors d'une expérience, voir Tableau 5.3 et Section **Types de conception utilisés** suivante. Il est important de rappeler que le sujet ne peut être au courant de ce qui est analysé précisément. Ainsi, si le sujet avait dû travailler sur deux versions d'un même système, il aurait pu identifier les différences entre ces deux versions et, dès lors, biaiser ses réponses. Cette notion de «*connaissance de l'objectif*» est un élément important qu'il faut prendre en compte lors de la réalisation d'une expérience. Elle sera débattue à travers la Section 5.6.2.

Types de conception utilisés

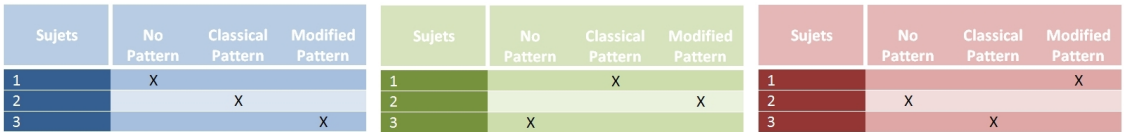
La conception de cette expérience est assez particulière. Le type de conception utilisé pourrait être assimilé à celui d'un type hybride. Certains éléments concernant l'enchaînement des tâches ont déjà été introduits précédemment. Cependant, cette section définit plus précisément la *répartition* des tâches au sein des groupes. Une section ultérieure précisera, quant à elle, l'*enchaînement* de ces tâches durant les expériences (voir Page 81).

Plusieurs catégories de conception ont été présentées à travers la Section 2.3.5. La catégorie la plus représentée dans la conception de cette expérience est celle utilisant **un facteur avec plus de deux traitements**. Pour chacune de ces catégories, il existe plusieurs types de conceptions, c'est-à-dire, plusieurs manières de les «*implémenter*». Le type de conception le plus utilisé dans cette expérience s'apparente à **la conception complètement aléatoire** (voir Page 37).

Il faut remarquer que ce type de conception détermine le fonctionnement d'une expérience pour **un** seul objet. Or, dans cette expérience, les sujets devaient travailler sur **trois** objets. La Figure 2.2 présentée en Page 37 peut ainsi être construite trois fois afin que chaque image représente la conception de l'expérience pour un objet, un *système*[6] dans notre cas, voir Figure 5.1.

L'objectif, ensuite, est de superposer ces images afin d'obtenir les différents traitements que chaque sujet devra réaliser lors de l'expérience. L'agencement de ces différents traitements pour chaque sujet doit cependant vérifier deux contraintes :

[6] voir Section 5.3.5 – **Systèmes**, Page 79



(a) JHotDraw (b) JRefactory (c) PADL

Fig. 5.1 – Conception complètement aléatoire de chaque système

- *Chaque sujet doit avoir un traitement sur chaque objet*
- *Aucun sujet ne peut avoir le même traitement sur deux objets différents*

En vérifiant ces deux contraintes, la superposition des conceptions de chaque objet assure le fait que *chaque sujet travaillera sur chaque système avec une version à chaque fois différente des autres*. Cette disposition particulière a été dénommée «*configuration*» lors de cette expérience. Elle correspond à une nuance de bleu dans le Tableau 5.3. Avec ce concept de «*configuration*», il est à présent plus aisé de comprendre pourquoi les groupes étaient au nombre de trois. Il était plus facile de créer trois configurations attribuées chacune à un groupe, tel que le montre le Tableau 5.3.

Systèmes

Les noms des différents systèmes utilisés lors de cette expérience ont souvent été abordés, mais n'ont jamais été présentés concrètement. Voici une explication de ces trois systèmes :

- **JHotDraw** : JHotDraw est un cadriciel («*framework*») pour réaliser des dessins techniques et structurés. Ce cadriciel fournit un support pour la création de formes géométriques et de formes personnalisées par l'utilisateur. Il permet d'éditer ces formes, et d'y ajouter des contraintes comportementales [JHotDraw 07].

- **JRefactory** : JRefactory est un outil de restructuration de code («*refactoring*») pour les programmes Java [JRefactory 04].

- **PADL** : PADL est un méta-modèle pour décrire des programmes orientés-objets, c'est une bibliothèque qui offre des classes et des méthodes permettant de décrire un programme, comme le vocabulaire et la grammaire française permettent de faire des phrases. Ce méta-modèle est similaire au méta-modèle UML.

Ces trois systèmes ont été choisis pour deux raisons :

- La première raison est que ces trois systèmes sont tous des projets open-source issus du monde académique. Il est donc plus aisé d'en acquérir le code source. Le but de l'expérience n'était pas de travailler à proprement parler sur le code source, mais bien d'utiliser ce dernier à des fins de rétro-ingénierie. En effet, une rétro-ingénierie partielle de chaque système a été réalisée afin de reconstituer un diagramme de classes. Cette rétro-ingénierie était effectuée sur un sous-ensemble du système. Ce sous-ensemble était déterminé par la présence du patron de conception «*Visiteur*».

- La deuxième raison est que ces trois systèmes contenaient tous le patron «*Visiteur*».

JHotDraw	
Compréhension	**Maintenance**
Est-il possible de modifier la couleur d'une ligne de connexion ? Pourquoi ?	Où faut-il ajouter une méthode permettant de dupliquer deux figures liées entre elles, et quelles méthodes existantes appeler ?

Tab. 5.4 – Exemples de questions

Etant donné que le but de l'expérience était d'analyser l'impact du «*Visiteur*» sur la compréhension et la maintenance de programmes, ces trois projets constituaient de très bons supports, de par leurs caractéristiques, sur lesquels il était possible de réaliser les expériences.

Diagrammes

Tel qu'il a été expliqué précédemment, une rétro-ingénierie partielle a été effectuée sur les différents systèmes. Cette rétro-ingénierie n'a cependant pas été automatisée. Le but était de fournir un diagramme lisible sur un écran de taille normale, soit 17". Le nombre de classes n'était pas prédéfini mais a été guidé par le bon sens. Il est bon de rappeler que l'objectif est de simuler une activité de compréhension et de maintenance en répliquant une situation réelle. Le nombre de classes rétro-construites pour chaque système n'était par conséquent ni trop faible ni trop élevé.

Pour rappel, trois versions équivalentes au niveau *pragmatique* ont été construites pour chaque système. Suite à un manque de connaissance dans le domaine et à certains critiques, il n'est pas possible d'affirmer que les diagrammes étaient *sémantiquement* équivalents. Le problème réside dans le comportement du code généré sur une machine à partir des diagrammes de classes UML. L'équivalence pragmatique a été considérée afin d'exprimer le fait que les sujets réalisaient les mêmes tâches, *i.e.* ils répondaient aux mêmes exigences, avec ou sans patron de conception. La version «*rétro-construite*» peut être assimilée à la version «*classical pattern*» (CP). Sur base de cette version, il était possible de dériver les deux autres versions. La version «*modified pattern*» (MP) était construite suite à un déplacement des différentes classes au sein du diagramme. A nouveau, aucune modification systématique n'a été réalisée. Le but dans cette expérience était simplement de comparer la version «*canonique*» des patrons de conception face à une version «*non familière*». La version «*no pattern*» était, quant à elle, construite en réinjectant les méthodes des «*Visiteur*» dans les classes appelant ces «*Visiteur*». Les différentes versions des trois systèmes sont disponibles en Annexes F.

Questions

Les *questions* présentées dans cette section sont liées au concept de «Tâche». Il existe deux types de questions : les questions de **compréhension** et de **maintenance**. Etant donné que le but était de comparer les différences entre les réponses des trois versions, une seule question de compréhension et une seule question de maintenance ont été conçues pour chaque système. Trois questions de compréhension et trois questions de maintenance ont par conséquent été réalisées. Ces dernières sont disponibles en Annexes H. Le Tableau 5.4 présente un exemple pour chaque type de question posée lors de l'expérience.

Les différentes questions sont le fruit de réflexions de plusieurs personnes. La difficulté de la conception des questions réside dans le fait que ces dernières ne devaient être ni trop simples, ni trop compliquées. De plus, chaque question devait posséder plus ou moins la même durée de traitement. Ces questions ont donc été mûrement réfléchies afin de trouver des tâches plus ou moins identiques entre chaque système.

Enchaînement des tâches

Jusqu'à présent, seules les tâches que devaient réaliser les sujets ont été présentées. La phase de conception sert également à définir l'*enchaînement* de ces tâches à travers l'exécution des différentes expériences. Tel qu'il a été énoncé en début de Section 5.3.5, la conception d'une expérience doit opérer certains choix qui seront décisifs pour l'utilisation de méthodes statistiques lors de la phase d'*analyse des données collectées*.

Cette conception doit également prendre en compte un autre facteur lors de ces choix. Il s'agit de la «*validité*». Ce concept n'a pas encore été abordé mais représente un élément crucial dans l'établissement d'une expérience. Cette validité fera l'objet d'une section à part entière, voir Section 5.6.

La Section 5.3.5 – **Principes de conception utilisés** – a présenté les applications du concept de *randomisation*, et notamment celles sur les systèmes et les tâches. Le lien peut à présent être fait entre cette randomisation et le concept de «*validité*». Etant donné que les sujets devaient travailler sur trois systèmes différents avec, pour chacun d'entre eux, une tâche de compréhension et de maintenance, la randomisation a été utilisée à des fins de lissage d'effets. En effet, certains biais pourraient apparaître suite à l'enchaînement des tâches et des systèmes lors d'une expérience. La randomisation permet précisément de limiter les effets de ces biais. Cette question de biais sera également débattue en Section 5.6.

Concrètement, chaque sujet suivait une «*configuration*» (voir **Types de conception utilisés**, Page 78), soit trois systèmes avec deux tâches pour chacun d'eux. L'expérience comportait donc six tâches à répartir de manière aléatoire. Cependant, afin de ne pas biaiser le sujet, les deux tâches propres à un même système devaient se succéder obligatoirement. Ainsi, le sujet ne devait pas travailler sur la compréhension de JHotDraw puis enchaîner sur la maintenance de PADL pour revenir à la maintenance de JHotDraw, par exemple. Cette dernière configuration aurait obligé le sujet à se replonger systématiquement dans le contexte. Cela aurait introduit un effort supplémentaire non désiré et, par conséquent, aurait introduit un biais.

Il reste cependant une dernière précision à fournir avant de présenter l'enchaînement réel des tâches lors de cette expérience. Etant donné que les sujets ne disposaient ni du temps, ni des ressources nécessaires pour s'approprier le but et le fonctionnement de chaque système, une spécification leur a été fournie. Cette dernière était mise à leur disposition juste avant qu'ils réalisent les deux tâches assignées sur le système. Les informations fournies dans la spécification suffisaient à répondre aux questions, comme en ont témoigné les réponses mais aussi les sujets. Les spécifications fournies aux sujets sont reprises en Annexes E.

Tous les détails concernant les tâches ayant été fournis, l'enchaînement de ces dernières peut être illustré. La Figure 5.2 présente cet enchaînement mais nécessite quelques explications. Cette figure est divisée en deux grandes parties :

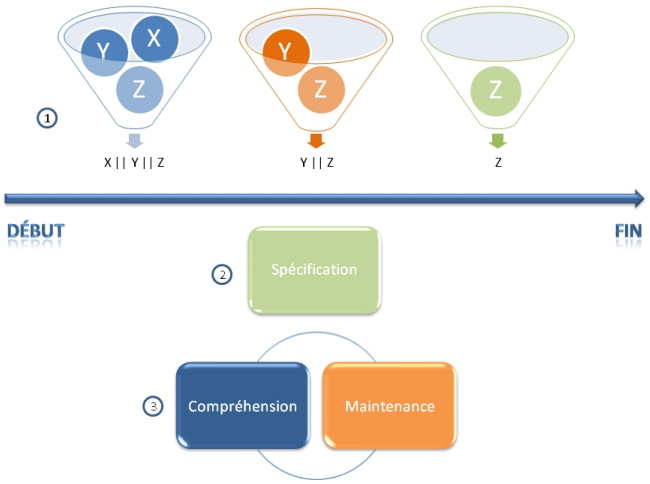


FIG. 5.2 – Enchaînement des tâches

- **Les systèmes** : JHOTDRAW, JREFACTORY, PADL.
- **Les activités** sur ces systèmes : la lecture de la **spécification** et les **deux tâches**.

Ces deux parties sont divisées par l'*axe du temps* de l'expérience. Le point 1 représente le fait qu'un système soit choisi aléatoirement parmi les trois systèmes utilisés. Une fois que ce système est déterminé, la spécification le concernant est mise à disposition du sujet (point 2). Lorsque le sujet a terminé avec la spécification, une tâche de compréhension ou de maintenance (point 3) est choisie de manière aléatoire. Une fois cette tâche accomplie, le sujet travaille sur l'autre tâche. Lorsque ce cycle est terminé, un nouveau système est choisi parmi les deux systèmes restants, et le même enchaînement d'activités est réalisé. Cette enchaînement se déroule de manière similaire pour le troisième système.

5.3.6 Instrumentation

Plusieurs systèmes d'*eye-tracking* existent sur le marché. Le système utilisé dans cette expérience est dénommé EyeLink II de la compagnie SR Research[7]. EyeLink II possède la plus haute résolution (le bruit étant limité à $< 0.01°$) et le taux de données le plus rapide (500 échantillons par seconde) de tous les systèmes d'*eye-tracking* dit «*head mounted*» (en comparaison avec «*head supported*»).

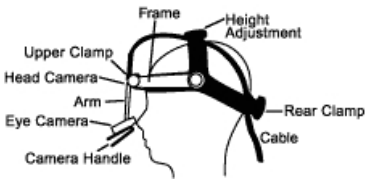


FIG. 5.3 – Casque du système d'*eye-tracking* «EyeLink II»

[7]http ://www.eyelinkinfo.com/

Un «*eye-tracker*» est composé de deux ordinateurs et d'un casque. Le casque, représenté en Figure 5.3, inclut deux caméras et un émetteur infrarouge. Les caméras utilisent les rayons infrarouges qui sont reflétés sur la pupille du sujet afin d'enregistrer les mouvements des yeux. Quatre capteurs sont placés sur l'écran du sujet. Ces capteurs travaillent avec l'émetteur infrarouge et calculent la position du casque en fonction de l'écran. Conjointement avec les rayons reflétés sur la pupille du sujet, ces quatre capteurs permettent de calculer précisément la position du regard des sujets à l'écran.

L'ordinateur du sujet et de l'expérimentateur sont reliés via une connexion de type «*Ethernet*», comme le présente la Figure 5.4. Leur communication est essentiellement basée sur le principe d'«Action/Evénement» → «Réaction». Lorsqu'une action est émise par l'ordinateur du sujet, l'ordinateur de l'expérimentateur réagit et reprend le contrôle. Typiquement, l'ordinateur du sujet envoie la position du regard du sujet à l'ordinateur de l'expérimentateur, qui enregistre à la volée ces données sur le disque. Une fois que l'expérience est terminée, l'ordinateur de l'expérimentateur renvoie tout le fichier de sortie à l'ordinateur du sujet afin de faciliter son accès et les analyses.

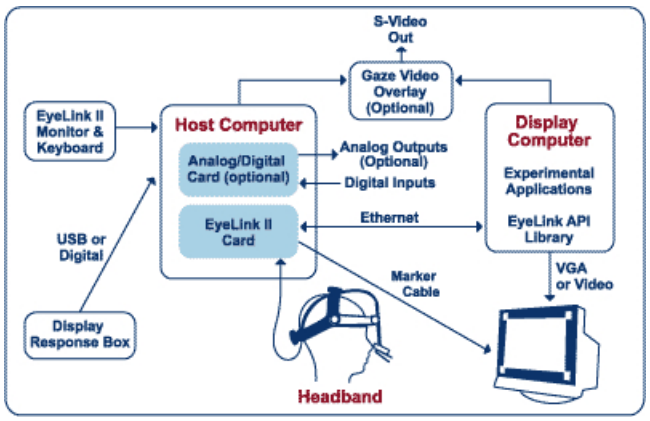


FIG. 5.4 – Architecture du système d'*eye-tracking* «EyeLink II»

Pour rappel, les données collectées par le système d'*eye-tracking* sont de deux types : les *fixations* et les *saccades*. Les **fixations** représentent les stabilisations des yeux durant un regard, alors que les **saccades** correspondent aux mouvements d'une fixation à une autre. Le nombre de fixations et de saccades peuvent cependant varier d'une dizaine à plusieurs centaines en fonction de la taille et de la complexité des diagrammes de classes utilisés. Ces deux informations de base fournissent les données pour les analyses. Comme dans les travaux précédents, par exemple [Guéhéneuc 06, Yusuf 07], les fixations ont été utilisées comme la manifestation concrète d'une focalisation de l'attention du sujet dans une zone de l'écran. C'est pourquoi, le nombre de fixations a été utilisé afin de représenter l'attention que livre un sujet sur un diagramme.

5.3.7 Validité de l'expérience

La validité de cette expérience sera présentée en Section 5.6, après avoir détaillé les analyses de données. Le lecteur pourra critiquer l'expérience et comparer cette critique par la suite avec les menaces à la validité.

5.4 Exécution

5.4.1 Préparation

La préparation d'une expérience regroupe les différentes activités préliminaires à l'exécution de cette expérience. Ces deux activités préliminaires sont le recrutement de sujets et la préparation du matériel.

Recrutement des sujets

Le recrutement des sujets s'est effectué en deux temps. Le premier temps avait pour objectif d'informer les personnes de la présence d'une expérience. Différentes affiches ont ainsi été placées au sein de D.I.R.O afin de toucher le plus de gens possible. Cette affiche est disponible en Annexe C. De même, les étudiants de MM. Sahraoui et Guéhéneuc ont été mis au courant par ces derniers de la présence d'une expérience. De nombreux e-mails ont également circulé sur les différents alias.

Dans un second temps, un site web a été mis à disposition des personnes afin qu'elles puissent s'inscrire à l'expérience. Ce site web est une adaptation du travail original de M^{elle} Alicia Heraz, étudiante-doctorante à l'Université de Montréal. Il a permis de fournir l'horaire des expériences, des informations complémentaires, ainsi qu'une vidéo expliquant le déroulement des expériences. Des images du site web sont disponibles en Annexe D. Le site web servait essentiellement à enregistrer les sujets dans une base de données et à faciliter la réservation des plages horaires. De plus, via le lien envoyé dans les mails de recrutement, les personnes pouvaient choisir à leur aise la plage horaire qui leur convenait le mieux, et ce, de n'importe quel endroit et à n'importe quel moment de la journée.

Préparation du matériel

Le matériel nécessaire à l'expérience a été préparé avant l'arrivée de chaque sujet. Le système d'*eye-tracking* était testé. Les différents documents liés à la configuration à laquelle était assigné le sujet étaient rassemblés afin d'éviter toute erreur. De même, tout facteur connu pouvant perturber le fonctionnement de l'expérience a été traité (débranchement du téléphone, affiche sur la porte, aucune personne dans le local, etc). Tous ces traitements ont permis d'assurer aux sujets un environnement à l'identique, fortement limité par les biais externes.

5.4.2 Exécution des expériences

Tel que l'a présenté la section précédente, les expériences se sont déroulées dans une pièce calme, à l'abri de perturbations. L'exécution de l'expérience se compose de quatre étapes (voir Figure 5.5). Pour chaque sujet, l'expérience prenait environ une heure, incluant un didacticiel (15–20 minutes), la présentation du système d'*eye-tracking* (5 minutes), la collecte des données (20–30 minutes), et un questionnaire (5 minutes).

Le didacticiel rappelait aux sujets les différentes constructions utilisées dans les diagrammes de classes UML. Le but ensuite était de simuler une expérience en taille réduite afin de familiariser les sujets aux questions qu'ils allaient être amenés à répondre. Cela permettait également de leur expliquer quel type de réponse était attendu. Le *feedback* des sujets d'une part, et les observations réalisées, d'autres part, ont montré que ce didacticiel était très utile pour préparer les sujets. Cela permettait également de diminuer leur stress.

Après ce didacticiel, quelques informations techniques étaient données aux sujets à propos du système d'*eye-tracking*. De même, l'explication du déroulement de l'expérience était détaillé afin de détendre le sujet.

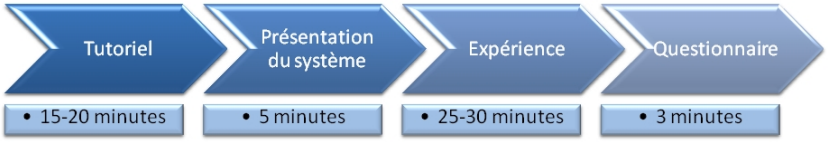


FIG. 5.5 – Processus de l'exécution de l'expérience

Avant de réaliser les tâches de compréhension et de maintenance sur chacun des systèmes, le système d'*eye-tracking* devait être calibré. La calibration se fait sur base du casque. Elle exige que les sujets fixent neuf points à l'écran de manière consécutive. Une fois que cette calibration était réalisée, la collecte des données pouvait commencer.

Cette collecte des données s'effectuait sur les trois questions de compréhension et de maintenance. Les différents systèmes, types de diagrammes, configuration, questions et l'enchaînement de ces questions ont déjà été présentés précédemment. Ces concepts ne seront pas expliqués à nouveau dans cette section. Pour plus d'informations, veuillez vous référer à la Section 5.3.5.

Finalement, un questionnaire était mis à la disposition des sujets afin d'évaluer leur connaissance en UML et en patrons de conception. Ce questionnaire était fourni après l'expérience afin de ne pas divulguer le but de celle-ci, ce qui aurait introduit un biais au niveau du raisonnement des sujets.

Cependant, certaines personnes (extérieures à l'expérience) ont fait part de leurs craintes quant au biais de l'introduction d'un tel questionnaire en fin d'expérience. Ils craignaient que l'évaluation des sujets soient influencées par les réponses qu'ils auraient pu donner durant l'expérience. Ainsi, selon eux, un sujet qui aurait bien répondu, aurait forcément prétendu qu'il avait un bon niveau de connaissance. De manière analogue, une personne qui aurait mal répondu, aurait prétendu avoir un mauvais niveau de connaissance. Ce problème avait également été étudié. Avant de répondre au questionnaire, chaque sujet recevait des instructions. L'une d'entre elles concernait *l'évaluation personnelle de ses connaissances avant de participer à l'expérience.*

5.4.3 Matériel technique

La section précédente a détaillé l'enchaînement des étapes de chaque expérience. Cette section explique, quant à elle, le côté plus technique en présentant les matériaux utilisés, autres que le système d'*eye-tracking*.

Les sujets étaient assis durant les expériences sur une chaise ordinaire de bureau. Il leur a été demandé d'effectuer le moins de mouvements possible. Les sujets utilisaient un écran 17"CRT et un clavier ordinaire. Afin de faciliter le déroulement de l'expérience et pour éviter toute erreur de manipulation, toutes les touches du clavier étaient désactivées, à l'exception de la touche «*Echap*». Cette dernière permettait aux sujets de contrôler l'expérience. La gestion de la souris était également désactivée. Chaque question était affichée

à l'écran, au bas de chaque diagramme. Une fois que les sujets avaient réalisé la tâche et donné leur réponse, ils appuyaient sur la touche «*Echap*». Cela leur affichait une nouvelle question/diagramme. Un *game pad* a été utilisé afin d'enregistrer l'exactitude de la réponse du sujet. Cependant, évaluer l'exactitude d'une réponse relève d'une certaine subjectivité. C'est pourquoi deux expérimentateurs étaient présents pour évaluer les réponses, dans le but d'augmenter l'objectivité. Chaque expérimentateur disposait d'une *check-list* pour chaque question des trois diagrammes de chaque système. Les éléments de ces *check-lists* correspondaient aux classes/méthodes/attributs nécessaires pour répondre à la question posée. Si toutes les cases étaient cochées, la réponse était «*correcte*». Si la moitié des cases étaient cochées, la réponse était «*partiellement correcte*», sinon elle était «*fausse*». Les deux évaluations étaient utilisées afin d'enregistrer une seule valeur pour chaque réponse de chaque sujet. En cas de conflit entre les deux expérimentateurs, une discussion avait lieu afin de décider de la valeur à enregistrer.

5.4.4 Validation des données

La motivation et l'implication des sujets sont essentielles pour valider les données, tout comme l'environnement dans lequel se sont déroulées les expériences.

La section précédente a présenté les conditions dans lesquelles les sujets ont dû réaliser les différentes tâches. Ces conditions peuvent sembler particulières mais n'ont cependant pas affecté la manière de répondre. Cette affirmation peut être énoncée au vu des réponses. En effet, celles-ci semblaient relativement homogènes au sein des groupes, ce qui laisse à penser que le seul facteur ayant pu influencer les résultats était la présence du casque du système d'*eye-tracking*.

De même, la section précédente s'est attelée à expliquer comment les réponses des sujets étaient évaluées. Ces techniques d'évaluation ont permis notamment de conclure à la fin des expériences qu'aucune personne n'avait fait preuve d'un manque de motivation et d'implication.

Remarque Suite à la présence d'un problème technique, les données d'un sujet ont du être rejetées. Ce problème est discuté plus longuement dans la Section 5.6.1.

5.5 Analyses et interprétations

Cette section présente les résultats des tests d'hypothèses qui ont été réalisés sur les données collectées. Le test t de Student a été réalisé sur la variable normale «*NRRF*», voir Section 5.3.3. Des analyses ont également été réalisées sur deux facteurs qui auraient pu influencer les résultats, à savoir les variables indépendantes «Connaissance UML» et «Connaissance DP».

Avant de détailler les différentes analyses, le Tableau 5.5 présente quelques chiffres résultant de la collecte des données.

Données Collectées			
Sujets (#)	24	Temps moy. par question (C)(*min*)	2,71
Données (*Mb*)	135	Temps moy. pert. par question (C)(*min*)	0,73
Vidéos (#)	144	Nb Fix. moy. par question (C)(#)	447
Temps Total Expériences (*heure*)	25	Nb Fix. pert. moy. par question (C)(#)	162
Temps Total *eye-tracking* (*min*)	451	Temps moy. par question (M)(*min*)	3,54
Temps Pert. *eye-tracking* (*min*)	115	Temps moy. pert. par question (M)(*min*)	0,87
Total Fixations (#)	72 416	Nb Fix. moy. par question (M)(#)	558
Fixations Pertinentes (#)	25 319	Nb Fix. pert. moy. par question (M)(#)	190

Tab. 5.5 – Données collectées

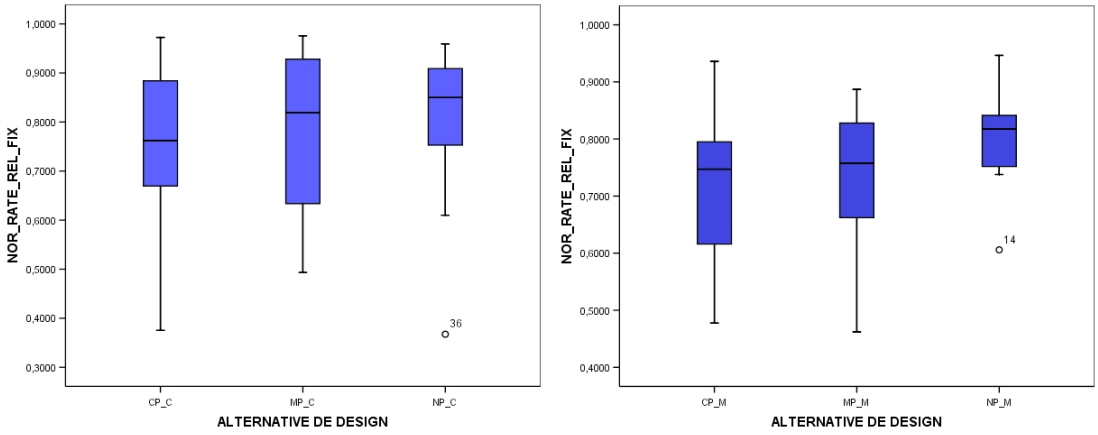
5.5.1 Analyses des données sur les tâches de Compréhension

Le Tableau 5.6 résume l'effet de l'utilisation du patron de conception «*Visiteur*» sur la compréhension des diagrammes de classes UML. La variable NRRF est représentée pour les trois alternatives de design CP_C, MP_C, et NP_C. Les colonnes nommées «*p-value*» présentent le coté significatif des tests statistiques sur les différences entre CP_C et NP_C (HC_{0_1}), et MP_C et NP_C (HC_{0_2}). Pour rappel, un NRRF élevé indique un effort faible et un NRRF faible indique un effort important.

	CP_C	MP_C	NP_C	p-value (HC_{01})	p-value (HC_{02})
NRRF (%)	75,77	77,95	81,49	0.221	0.663

Tab. 5.6 – Effet du «*Visiteur*» sur la Compréhension

Le taux de fixations pertinentes varie légèrement entre CP_C et MP_C (76% *et* 78%). Par contre, le pourcentage pour NP_C indique un taux de fixations plus important (81%) par rapport à CP_C et MP_C. Les distributions présentées en Figure 5.6(a) montrent que les médianes pour NRRF varient autour de 75% et 85% pour toutes les tâches. Cependant,



(a) Compréhension (b) Maintenance

FIG. 5.6 – Distribution des données pour la Compréhension et la Maintenance

il existe une large variance pour NRRF entre les sujets travaillant sur MP_C, *i.e.* boîte à moustache large, par rapport aux sujets travaillant sur CP_C et NP_C. Il faut également constater la variance assez forte pour CP_C dont le premier quartile regroupe des taux compris entre approximativement 38% et 67%. Cette variance pourrait indiquer que le niveau de connaissance en UML et/ou en patrons de conception auprès des sujets peut jouer un rôle important lorsque le «*Visiteur*» est présent dans les diagrammes.

En conclusion, le fait que les tests ne soient pas significatifs au Tableau 5.6 indique que la présence du «*Visiteur*», tout comme son layout, n'ont pas d'impact significatif sur la compréhension des diagrammes de classe UML.

5.5.2 Analyses des données sur les tâches de Maintenance

Les diagrammes sans «*Visiteur*» semblent demander moins d'effort. Les différences de moyenne entre les tâches {CP_M, MP_M}, et NP_M sont plus importantes : environ $7-8\%$ en plus de taux de fixations pertinentes pour les diagrammes sans le patron du «*Visiteur*», voir Tableau 5.7.

	CP_M	MP_M	NP_M	p-value (HM_{01})	p-value (HM_{02})
NRRF (%)	71,97	73,46	80,18	0.027	0.725

TAB. 5.7 – Effet du «*Visiteur*» sur la Maintenance

La Figure 5.6(b) montre que les sujets travaillant sur les diagrammes sans le patron du «*Visiteur*» présentent des taux supérieurs par rapport aux groupes ayant travaillé sur les conceptions CP_M et MP_M. De plus, ces sujets travaillant sur NP_M ont une performance uniforme, *i.e.* boîtes à moustaches aplaties, par rapport à celles des deux autres groupes.

En ce qui concerne les hypothèses étudiées, il n'est pas possible de conclure qu'un diagramme sans patron du «*Visiteur*» a un impact significatif sur les efforts de maintenance, par rapport à un diagramme incluant le «*Visiteur*» avec son layout classique (voir p-value proche de 0.027 pour NRRF). En effet, cette valeur va dans le sens de l'hypothèse HM_{01} formulée. S'il est possible de réfuter une hypothèse en trouvant une valeur significative, il n'est cependant pas possible de trouver une telle valeur qui permette de valider une hypothèse. En d'autres termes, il est toujours possible d'affirmer l'absence d'une relation mais il est impossible d'en affirmer la présence.

Néanmoins, cet impact n'est pas significatif lorsque le patron du «*Visiteur*» est présenté avec le layout modifié (voir p-values 0.725). A nouveau, il est possible de remarquer qu'avec une valeur médiane presque identique pour CP_M et MP_M, la variance de ces groupes est assez forte, *i.e.* boîtes à moustaches larges. Les niveau de connaissance en UML et en patrons de conception peuvent éventuellement expliquer à nouveau cette variance.

5.5.3 Impact des facteurs seconds

Les résultats présentés en Sections 5.5.1 et 5.5.2 montrent qu'il n'existe pas d'impact du «*Visiteur*» sur la maintenance lorsqu'il est affiché sous un layout classique. Dans le cas de la compréhension, aucun impact significatif ne peut être établi. Suite aux variances observées pour NRRF pour les tâches de compréhension de maintenance, présentées respectivement en Sections 5.5.1 et 5.5.2, il a été décidé d'étudier si les niveaux de connaissance en UML et en patrons de conception impactaient les résultats obtenus. Etant donné que tous les sujets s'y connaissaient en UML et en patrons de conception, deux niveaux ont été considérés pour chaque facteur : 1 pour «*bon*» et 2 pour «*très bon*».

La Figure 5.7(a) présente un impact intéressant sur la connaissance en UML au niveau de la compréhension. En effet, sur les diagrammes sans le «*Visiteur*» (NP_C), les sujets ayant une très bonne connaissance en UML (niveau 2) fournissent des résultats meilleurs que ceux avec une bonne connaissance (niveau 1). Il est également possible de remarquer que les sujets avec une bonne connaissance en UML réalisent des taux de fixations pertinentes relativement équivalents pour les trois alternatives de design. Néanmoins, le taux de fixations pertinentes reste plus élevé pour le diagramme sans patron de conception (NP_C).

Pour la maintenance, voir Figure 5.7(b), il est intéressant de voir que les sujets ayant une très bonne connaissance en UML (niveau 2) réalisent des taux bien plus élevés que les sujets ayant une bonne connaissance (niveau 1). Cela pourrait expliquer que les personnes ayant une connaissance accrue d'UML arriveraient plus facilement à modifier les éléments qui le nécessitent. Cependant, il est possible de remarquer une forte différence entre les alternatives CP_C et NP_C au niveau des taux pour les sujets ayant une bonne connaissance (niveau 1), *i.e.* respectivement $< 67,50\%$ et 80%.

Ces résultats ne peuvent être confirmés par un test *2-way* d'ANOVA. La Table 5.8(a) donne les valeurs significatives des tests pour l'impact de la connaissance d'UML et l'impact combiné (Alternative de design × Connaissance UML) pour les tâches de compréhension et de maintenance. Toutes les *p-values* sont plus grandes que 0.05, ce qui indique qu'il n'y a pas de différence significative dans le comportement des sujets en fonction des différents niveaux de connaissance en UML. Néanmoins, la *p-value* obtenue pour la maintenance (0.061) présente un résultat prometteur qui mériterait d'être approfondi en répliquant l'expérience.

(a) ANOVA Connaissance UML

	NRRF (Comp.)	NRRF (Maint.)
Connaiss. UML	0.494	0.061
Combiné	0.471	0.267

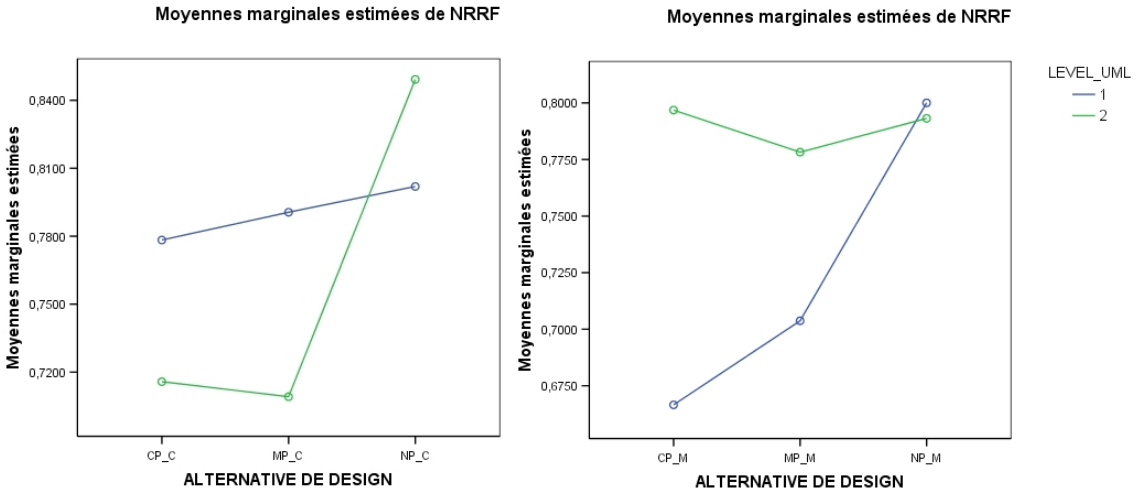
(b) ANOVA Connaissance DP

	NRRF (Comp.)	NRRF (Maint.)
Connaiss. DP	0.692	0.245
Combiné	0.217	0.546

Tab. 5.8 – Impact de «Connaissance UML» et de «Connaissance DP»

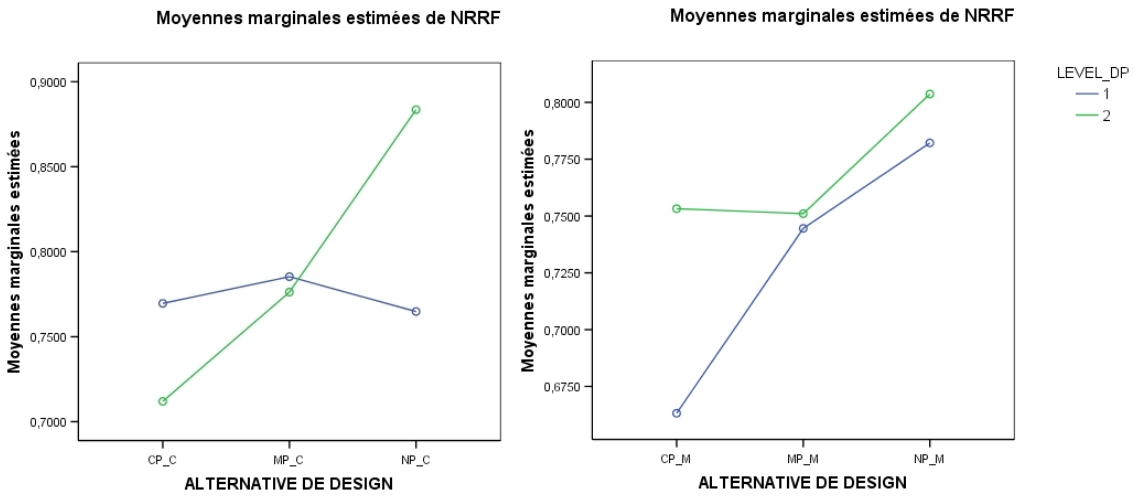
Les mêmes analyses ont été réalisées pour l'impact du niveau de connaissance en patrons de conception. Une importante différence a été observée au niveau du comportement sur NP_C entre les sujets ayant un très bon niveau de connaissance (niveau 2) et les sujets ayant un bon niveau de connaissance (niveau 1). Un écart tout aussi important peut être constaté pour l'alternative de design CP_M. Les Figures 5.8(a) et 5.8(b) présentent ces différences d'environ 10% pour le taux de fixations pertinentes. Il est possible également d'observer pour les tâches de maintenance, voir Figure 5.8(b), qu'il existe une différence entre les groupes de sujets (niveau 1 et 2) sur toutes les alternatives de design. Cette différence est une indication que le niveau de connaissance en patrons de conception a un impact sur les tâches de maintenance.

Dans le cas de la connaissance en patrons de conception, les tests *2-way* d'ANOVA donnent des *p-values* plus grandes que 0.05, comme le montre la Table 5.8(b). Il n'y a, par conséquent, aucune différence significative au niveau de la connaissance en patrons de conception entre les groupes.



(a) Compréhension (b) Maintenance

FIG. 5.7 – Impact de «CONNAISSANCE UML» sur la Compréhension et la Maintenance



(a) Compréhension (b) Maintenance

FIG. 5.8 – Impact de «CONNAISSANCE DP» sur la Compréhension et la Maintenance

5.6 Validité de l'expérience

Certaines menaces à la validité peuvent apparaître pour les résultats présentés dans la section précédente. Ces menaces sont liées à l'utilisation de sujets humains lors de l'expérience [Wohlin 00]. Les autres menaces sont dues à l'utilisation d'un système d'*eye-tracking*.

5.6.1 Validité interne

Trois menaces à la validité interne ont été identifiées : la **maturation**, l'**instrumentation** et la **diffusion des traitements**.

Maturation : Il s'agit de l'effet qui amène les sujets à réagir différemment au fur et à mesure que le temps passe dans une expérience. Par exemple, un sujet peut se fatiguer ou acquérir des compétences durant l'expérience.

Instrumentation : Il s'agit de l'effet que peut entraîner l'utilisation de certains artefacts durant l'exécution de l'expérience comme, par exemple, un formulaire ou les documents qui doivent être étudiés.

Diffusion des traitements : Cet effet apparait lorsque les sujets d'un groupe apprennent les traitements étudiés lors de l'expérience via les sujets qui ont déjà travaillé sur ces traitements.

• Dans le cas de la maturation, l'«*effet d'apprentissage*» a été traité en présentant les tâches (les questions à répondre) et les systèmes de manière aléatoire, voir Section 5.3.5. Cela évitait de favoriser une tâche par rapport à une autre en fonction de son ordre d'apparition. Cette utilisation du facteur aléatoire permettait également d'empêcher l'effet de fatigue. Cela aurait pu être le cas, par exemple, pour une tâche qui surviendrait toujours en dernière lieu. Cette tâche aurait été désavantagée suite au «*facteur de fatigue*». Ce dernier a également été réduit en limitant les efforts des sujets réalisant toutes les tâches (20 à 30 minutes pour la partie *eye-tracking*).

• La menace de l'instrumentation est liée à l'utilisation de l'équipement d'*eye-tracking*. Les sujets devaient porter un casque relativement lourd avec des caméras infrarouges. De plus, ces sujets devaient limiter leurs mouvements de la tête afin d'éviter toute décalibration. Afin de circonvenir cette menace, chaque film résultant de chaque expérience a été analysé. Ces films retracent le parcours des mouvements des yeux de chaque sujet pour chaque tâche. Seulement un seul cas de décalibration a été détecté et les données correspondantes ont été éliminées de l'échantillon (voir explications «*nuage de points*» ci-dessous). Cependant, les sujets devraient travailler pendant 6 × (4–5) minutes.

La concentration des personnes impliquait souvent de petits mouvements de la tête. Etant donné que l'équipement d'*eye-tracking* est extrêmement précis, ces petits mouvements de la tête ont créé des translations de coordonnées. Cela rendit les données temporairement erronées. Ces mouvements de la tête sont inévitables. Les gens se focalisent généralement sur un problème à la fois : soit sur les mouvements de leur tête, soit sur les tâches qui leur sont assignées. Dans le premier cas, les sujets ne sont pas concentrés suffisamment, ce qui biaise les réponses. Dans le second cas, des translations de coordonnées apparaissent suite aux mouvements de la tête. A nouveau, étant donné que le but était de réaliser une expérience rigoureuse, les sujets ont été autorisés à bouger leur tête. Cela permettait d'enregistrer des données cohérentes.

Le système d'*eye-tracking* utilisé faisait usage de petites caméras infrarouges et de la réflexion des pupilles des sujets. Ce posa certains problèmes d'enregistrement. Dans le cas de l'effet «*nuage de points*», aucune zone n'était focalisée plus qu'une autre. Des fixations et saccades étaient également trouvées dans des zones où il n'y avait ni classe, ni relation.

La différence entre les translations de coordonnées et l'effet «*nuage de points*» est que le premier peut être facilement réparé. Cela nécessite du travail supplémentaire mais ne représente pas une tâche complexe. De simples corrections devaient être appliquées aux données afin de créer les translations de coordonnées appropriées. Cela signifiait que les translations de coordonnées ne coûtaient rien en termes de sujets alors que l'effet «*nuage de points*» impliquait le rejet des données du sujet. Heureusement, les données d'un seul sujet ont été rejetées suite à ce problème. Les translations de coordonnées étaient plus fréquentes mais ont été réparées facilement.

• Finalement, afin d'éviter que les sujets ne connaissent les tâches à réaliser à l'avance, chaque sujet a été prié de ne parler en aucun cas de l'expérience aux autres sujets avant la fin de toutes les expériences. Ce critère a été respecté suite à la motivation, l'implication et le professionnalisme observés auprès des sujets.

5.6.2 Validité de construction

Durant cette étude, quatre menaces à la validité de la construction ont été identifiées : les biais liés à la **mono-exécution** et la **mono-méthode**, la **déduction d'hypothèses**, et l'**appréhension**.

Mono-exécution : L'utilisation d'une seule variable indépendante, d'un seul sujet ou traitement peut rendre l'expérience non suffisamment représentative.

Mono-méthode : L'utilisation d'un seul type de mesure peut impliquer un risque si cette mesure présente un biais.

Déduction d'hypothèse : Les sujets peuvent vouloir deviner les hypothèses de l'expérience. Si c'est le cas, ils seront tentés d'adapter leur comportement en fonction de cette déduction, ce qui altèrera les résultats.

Appréhension : Certains personnes ont peur d'être évaluées. Les gens tentent de paraître meilleurs lorsqu'ils passent des expériences.

• Le but de cette expérience est d'étudier l'impact des patrons de conception sur la compréhension et la maintenance. En ce qui concerne les objets, il a été fait usage de diagrammes rétro-construits de trois programmes ayant des domaines d'applications différents. Afin de traiter le biais de la mono-méthode, une seule variable dépendante a été considérée : NRRF. Tel qu'expliqué précédemment, il n'est pas possible, à l'heure actuelle, d'assurer que la variable considérée permette de représenter ce qui est étudié. De plus le temps et le nombre de fixations sont deux indicateurs corrélés. Le temps n'a pu être considéré comme deuxième variable.

• Les sujets n'ont pas été informés du but de l'étude à l'avance afin d'éviter qu'ils devinent les hypothèses. Ils n'étaient pas au courant de la présence des patrons de conceptions dans les diagrammes. Il leur a été simplement expliqué dans le didacticiel qu'ils allaient devoir

réaliser plusieurs tâches sur différents diagrammes provenant de plusieurs programmes. Aucun des sujets ne devait travailler sur plus d'une version de chaque diagramme (voir Section 5.3.5, Tableau 5.3).

• Différentes actions ont été réalisées afin d'éviter le facteur d'appréhension. Premièrement, le système d'*eye-tracking* était expliqué à chaque sujet. Celui-ci était assuré de l'absence de risques liés à l'utilisation des caméras infrarouges sur leurs yeux. Deuxièmement, un numéro aléatoire a été assigné à chaque sujet afin de garantir l'anonymat des données. Cela évitait par conséquent de faire le lien entre la qualité d'une réponse et l'identité d'un sujet. Finalement, aucun temps de réponse maximum n'a été prédéfini pour chaque question. Cela permettait d'éviter toute pression au niveau du sujet, et évitait l'introduction de biais.

5.6.3 Validité externe

Pour cette famille de menaces, les interactions de la **sélection** et de la **configuration** avec les traitements ont été considérées.

Interaction de la sélection avec le traitement : Ce biais correspond à la sélection d'une population non représentative de celle sur laquelle on désire généraliser les résultats.

Interaction de la configuration avec le traitement : Ce biais correspond au fait de ne pas avoir une configuration et des matériaux adaptés pour étudier ce qui est désiré. C'est par exemple le cas lorsqu'un appareil vétuste est utilisé.

• Le problème de savoir si des sujets-étudiants sont suffisamment représentatifs des professionnels en informatique a déjà été discuté dans plusieurs études [Sjoberg 07, Nielsen 00]. Dans le cadre de cette expérience, seuls des étudiants gradués et post gradués formés à UML ont été utilisés comme sujets. La majorité de ces étudiants disposaient d'une connaissance en modélisation UML et en patrons de conception relativement semblable aux professionnels du domaine.

• Pour les interactions avec la configuration, la taille et la complexité des diagrammes utilisés ont également été étudiés. Comme expliqué précédemment, les diagrammes ont été rétro-construits sur trois programmes open-source couramment utilisés. Les diagrammes présentés comptaient environ une vingtaine de classes. Il semblerait que ce nombre de classes soit habituel lors de tâches de compréhension et de maintenance. Il n'est cependant pas possible de répondre à la question cherchant à savoir si la présence de patrons de conception dans des diagrammes plus larges et plus complexes a un impact plus ou moins important. Des réplications spécifiques dans un environnement industriel sont requises pour tirer de telles conclusions.

6 Problèmes rencontrés

Sommaire

Avant-propos

En lisant la théorie, l'application d'un processus semble à première vue facile à mettre en oeuvre. Cependant, dans la pratique, il en est parfois tout autrement.

Ce chapitre présente les multiples problèmes survenus tout au long des différentes étapes du processus suivi lors de cette expérience. Pour chacun de ces problèmes, une discussion ainsi que la ou les solutions apportées y sont présentées.

6.1 Introduction

L'expérience, telle que présentée dans le chapitre précédent, semble s'être déroulée sans contretemps. Dans la réalité, il en a été tout autrement. La mise en oeuvre d'une expérience reste une tâche difficile. Elle exige de la rigueur mais aussi des compromis. Une expérience peut également mener à des situations inattendues qu'il faudra gérer le cas échéant.

Ce chapitre présente les différents problèmes et situations auxquels il a fallu faire face tout au long de l'expérience réalisée. Les solutions apportées à ces problèmes sont également discutées. Afin de faciliter la lecture, les problèmes sont présentés selon la même découpe que celle du processus suivi.

6.2 Définition

La phase de *définition* de cette expérience est un peu l'exception qui confirme la règle. Cette phase n'a pas posé de problèmes notables. La raison en est double :

- Cette expérience est la suite de travaux préliminaires réalisés par M. Guénéheuc et des étudiants du D.I.R.O. Ces travaux avaient pour objectif d'orienter l'axe de recherche de cette expérience.
- Cette expérience a fait l'objet d'un stage dont le sujet avait été défini antérieurement.

Ces deux raisons ont permis la définition préalable de l'expérience. C'est pourquoi, cette phase n'a pas posé de problèmes notables.

6.3 Planification

6.3.1 Problème 1 : La conception des tâches

Explication du problème :

La conception des tâches est une activité très délicate. L'enjeu derrière celle-ci est la validité de l'expérience. Il convient de définir des tâches qui correspondent au mieux aux activités du monde professionnel. En agissant de la sorte, il sera possible d'attribuer un certain poids à l'expérience et de pouvoir ainsi généraliser ses résultats, voir Section 2.3.7.

Le problème majeur qui s'est posé dans cette étape est l'identification de tâches pertinentes. Par «*tâche pertinente*», il faut entendre «*tâche de complexité raisonnable, tout en étant représentative des activités professionnelles visées*». Le problème consistait à trouver ce bon niveau de complexité.

L'autre problème lié à la conception des tâches est la difficulté de se mettre à la place du futur participant de l'expérience. Travailler sur les différents matériaux d'une expérience influence la connaissance de l'expérimentateur. Cette influence rend la prise de recul difficile pour ce-dernier, ce qui l'affecte dans l'évaluation de la complexité des tâches qu'il conçoit.

Solution du problème :

Plusieurs tâches ont été conçues dans un premier temps. Les avis de tierces personnes, en l'occurrence des professeurs du D.I.R.O. jouant le rôle d'experts, ont ensuite été utilisés afin de déterminer la tâche la plus pertinente. La conception de chaque tâche de compréhension et de maintenance est le résultat d'un consensus obtenu à la suite de réunions.

L'utilisation d'avis de tierces personnes est toujours bénéfique, et ce, pour diverses raisons. Dans le cas de cette expérience, les professeurs sollicités ont tous une certaine expérience dans la formulation de questions d'examens. Cette expérience dans le domaine a ainsi pu être utilisée afin de guider la conception des tâches. De plus, ces professeurs ne possédaient qu'une vision très partielle des trois systèmes sur lesquels les tâches étaient conçues. Ils ont ainsi constitué un public objectif pour évaluer la complexité de ces tâches.

6.3.2 Problème 2 : La prévention de l'introduction de biais

Explication du problème :

Les choix opérés lors de la phase de *planification* peuvent mener à l'introduction de certains biais lors de l'exécution de l'expérience. Il convient d'étudier les différentes alternatives de conception qui peuvent exister afin de limiter l'introduction de ces biais.

Dans le cas de cette expérience, différentes activités, introduisant potentiellement des biais, ont fait l'objet de réflexions. Deux réflexions ont particulièrement posé problème. Elles ont concerné le concept de «*récompense*» liée à la participation à l'expérience, et la «*manière*» de fournir un didacticiel aux sujets.

D'autres réflexions ont été réalisées afin d'éviter l'introduction de biais mais n'ont cependant pas conduit à des problèmes. Ces réflexions ont par ailleurs déjà été présentées dans la section concernant la validité de l'expérience, voir Section 5.6.

Solution de la récompense :

Les récompenses sont des moyens qui servent un objectif plus large : la «*motivation*». Celle-ci est nécessaire lors de la phase de recrutement des sujets, voir Section 5.4.1. Cependant, la motivation des sujets doit être gérée en gardant à l'esprit la validité de l'expérience. Offrir des crédits pour des cours ou offrir de l'argent peut modifier le comportement des sujets. Ces derniers pourraient venir uniquement dans le but de recevoir cette récompense et ne s'impliqueraient aucunement dans l'expérience. Ce genre de récompense a donc un impact négatif sur la validité de l'expérience. A l'inverse, ne rien offrir pourrait produire un effet tout aussi négatif sur cette validité. En effet, les sujets pourraient ne pas faire d'effort pour effectuer correctement les tâches, vu que rien ne les inciterait à le faire.

C'est pourquoi, dans le cadre de cette expérience, il a été jugé bon de n'offrir aucune récompense «*systématique*» à chaque participant. Le concept retenu, suite aux conversations effectuées, a été celui d'une «*loterie*». L'inscription de chaque participant à l'expérience assurait la participation à cette loterie. Chaque participant était averti que deux personnes allaient être tirées au sort à la fin des expériences. Les cadeaux étaient cependant gardés secrets.

Cette solution semblait être un juste milieu afin d'impliquer les participants d'une manière raisonnable. Ainsi, les participants ne venaient pas uniquement pour la récompense et ne venaient également pas parce qu'ils s'y sentaient obligés. Etant donné que les participants potentiels de l'expérience étaient relativement peu nombreux, ces derniers devaient constituer un échantillon suffisamment représentatif. Il était donc important de trouver des personnes naturellement motivées pour lesquelles cette motivation allait être récompensée.

Solution du didacticiel :

La présence d'un *didacticiel* dans cette expérience est à mettre en rapport avec le niveau de connaissance des sujets concernant UML et les patrons de conception. Certaines expériences se réalisent avec des groupes répartis selon le niveau d'expertise des personnes dans un domaine [Sim 06]. Dans le cas de cette expérience, le nombre de sujets potentiels ne permettait pas de former des groupes selon ce critère. C'est pourquoi, un didacticiel sur les patrons de conception a été présenté aux étudiants lors d'un cours quelconque de leur cursus, et ce, à leur insu. Un didacticiel sur les constructions de base des diagrammes de classes UML a également été donné à chaque sujet, mais de manière individuelle, voir **Exécution des expériences** Section 5.4.2.

La présentation inopinée d'un didacticiel sur les patrons de conception lors d'un cours est également le résultat de débats. Rassembler tous les sujets dans un local afin de parler des patrons de conception aurait biaisé les sujets. En effet, en parlant des patrons de conceptions lors de ce didacticiel, les sujets auraient immédiatement fait le rapprochement entre leur présence et le but de l'expérience. La crainte était que les sujets ne cherchent absolument la présence d'un patron de conception le jour de l'expérience. Cela aurait, par conséquent, biaisé le raisonnement, ce qui aurait conduit à des résultats non pertinents. En fournissant ce didacticiel dans un cours de génie logiciel, les sujets ne pouvaient se douter du lien entre ce didacticiel et leur participation à l'expérience.

En ce qui concerne le didacticiel sur les constructions des diagrammes de classes UML, le choix a été tout autre. Etant donné que la plupart des sujets connaissaient les diagrammes de classes, le but de ce didacticiel était de fournir un bref rappel. A l'inverse, le didacticiel sur les patrons de conception présentait, pour la plupart des participants, de nouveaux concepts. Les informations fournies lors de celui-ci étaient donc trop conséquentes que pour être assimilées durant les 10 minutes précédant l'expérience. Le rappel des constructions UML, à l'inverse, ne fournissait aucune nouvelle information pour le sujet et pouvait, par conséquent, figurer juste avant l'expérience à proprement parler.

Remarque Le seul pré requis de cette expérience était de connaitre les diagrammes de classes UML. C'est la raison pour laquelle le didacticiel se faisait en début d'expérience et ne comportait qu'un simple rappel des constructions (classes, opérations, attributs, associations)

6.3.3 Problème 3 : La conception en fonction de la technique

Explication du problème :

L'utilisation d'un système d'*eye-tracking* limite les informations qui peuvent être enregistrées lors d'une expérience. De plus, chaque système d'*eye-tracking* a ses propres fonctionnalités. Suivant les systèmes, il est possible d'enregistrer certaines données et d'effectuer certaines actions particulières. Le système d'*eye-tracking* utilisé lors de cette expérience possédait certaines limites qui ont influencé la conception de l'expérience. A titre d'exemple, voici quelques problèmes rencontrés :

- Utilisation obligatoire des diagrammes de classes sous format d'images
- Utilisation de la souris non prise en charge par le système d'*eye-tracking*

Solution du problème :

L'absence de certaines fonctionnalités au sein du système d'*eye-tracking* considéré a conduit à la recherche de solutions alternatives de conception. La solution retenue, suite à l'absence de gestion de la souris, est l'utilisation de la *voix*. La souris aurait permis au participant de désigner les classes UML répondant à la question. La voix a permis de pallier ce problème. Cependant, la substitution de l'utilisation de la souris par l'utilisation de la voix fait partie d'un problème plus important, présenté dans la Section 6.3.4. Le problème de l'affichage des diagrammes sous format d'image n'a cependant pas été solutionné. Il a été jugé que les tâches requises sur les diagrammes des sous-systèmes ne requéraient pas la notion d'«*interaction*» avec ces sous-systèmes.

Les tâches imposées aux sujets ne requéraient pas l'utilisation d'applications externes (browser, outils de bureautique, etc.), ni d'informations supplémentaires (fichiers stockés sur disque, contenu web, etc.). C'est pourquoi le principe d'*affichage d'images* à l'écran a été conservé dans cette expérience. Des techniques d'interaction existent néanmoins au niveau du système d'*eye-tracking* utilisé. Elles sont cependant très lourdes et gourmandes en traitement. En effet, l'utilisation d'un système d'*eye-tracking* ne fournit que les coordonnées des zones que l'oeil regarde. Il ne fournit aucune information sur le contenu réel auquel les coordonnées s'appliquent. Or, introduire de l'interaction, c'est-à-dire, permettre l'utilisation de la souris et toutes les fonctionnalités associées, nécessite un travail supplémentaire de *mapping* entre un instant dans l'expérience, une image et une coordonnée. Ce mapping entre ces trois informations est extrêmement coûteux en temps et ne pouvait être permis dans le cadre d'une telle expérience.

6.3.4 Problème 4 : Les contraintes liées à l'éthique

Explication du problème :

L'utilisation de matériel scientifique intrusif tel qu'un système d'*eye-tracking* sur des sujets nécessite l'approbation de certaines autorités. Le *Comité d'Ethique* de l'Université de Montréal a donné son accord concernant l'utilisation d'un matériel d'*eye-tracking*. Cet accord a été octroyé sous certaines conditions. La principale condition est la conservation du caractère anonyme des expériences. Cela impliquait qu'aucune information permettant de reconnaître un sujet ne pouvait être conservée après l'expérience[1]. Ces informations ne concernent pas seulement la fiche signalétique d'un sujet. Elles concernent également toute photo ou enregistrement de la voix qui aurait pu s'effectuer durant cette expérience.

L'approbation du *Comité d'Ethique* ayant été donnée, l'expérience devait respecter les conditions de cette approbation. Cela posa certains problèmes au niveau de la conception de l'expérience. Etant donné que les réponses des sujets devaient être enregistrées quelque part, l'interdiction d'enregistrer la voix modifia la conception de l'expérience. Une réflexion intensive a été réalisée conjointement avec MM. Sahraoui et Guéhéneuc afin de trouver un moyen optimal d'enregistrer les réponses sans les biaiser.

Solution du problème :

Différentes solutions pouvaient prétendre répondre au problème. Cependant, leur utilisation aurait introduit un certain biais. A titre d'exemple, les solutions suivantes avaient

[1] Par «*expérience*», il faut entendre ici le «*processus de l'expérience*», voir Section 2.1.2, qui se termine par la phase d'*Analyses et interprétations* ou *Présentation et paquetage*.

été proposées mais n'ont pas été retenues suite aux biais qu'elles introduisaient :

- Le participant répond sur un nouvel écran en tapant sa solution au clavier
- Le participant répond en écrivant sa solution sur une feuille de papier

La première solution ne pouvait être retenue étant donné qu'elle pouvait biaiser les solutions des participants. En effet, en faisant disparaître le diagramme, les participants ne pouvaient compter que sur leur mémoire à court terme pour répondre à la question. Deux participants, possédant tous les deux la bonne réponse au départ, auraient ainsi pu répondre différemment. Cette solution ne pouvait être retenue.

La seconde solution n'était techniquement pas réalisable vu qu'elle aurait obligé le sujet à bouger son corps pour écrire sur la feuille de papier. De tels mouvements du corps, et plus particulièrement de la tête, auraient causé des problèmes au niveau des calculs des coordonnées (décalibration), nécessitant de recalibrer le casque après chaque question.

Suite aux différentes réflexions réalisées, une autre solution a été choisie. Il s'agit de la *réponse orale*. Malheureusement, cette réponse orale ne pouvait être enregistrée directement, étant donné la clause que l'expérience devait respecter. C'est pour cela qu'un «*game pad*» et une «*check-list*» ont été utilisés afin de conserver une trace des réponses des différents sujets. La Section 5.4.3 fournit plus de précisions sur l'utilisation de ce «*game pad*» et de cette «*check-list*». L'Annexe G présente les *check-lists* qui ont été utilisées lors de l'expérience.

Le choix de cette conception est motivé par trois facteurs :

- L'obligation de respecter l'approbation du *Comité d'Ethique*
- La volonté d'éviter l'introduction de biais (validité de l'expérience accrue)
- La limitation due à la technique (voir aussi **Problème 3 : La conception en fonction de la technique** en Page 98)

Ainsi, la présence du contrat auquel devait se conformer l'expérience a conduit, d'une manière plus ou moins directe, à modifier la conception de cette expérience.

Remarque Il est important d'observer à quel point l'interdiction de concevoir une expérience d'une certaine manière peut, suite à la combinaison de plusieurs facteurs, mener à certains problèmes de conception bien plus importants.

6.4 Exécution

6.4.1 Problème 1 : Le recrutement des sujets

Explication du problème :

Recruter des sujets pour une expérience est une étape aussi importante que les autres lors de la phase d'*opération*. Elle reste cependant une activité difficile à mettre en oeuvre, tel qu'en témoigne cette expérience. En effet, malgré l'utilisation d'affiches dans le D.I.R.O. et la mise à disposition d'un site web, voir Section 5.4.1, le recrutement ne s'est pas déroulé aussi facilement que prévu. Dans le cas de cette expérience, le problème ne résidait pas dans les moyens utilisés pour recruter, mais bien dans la motivation des sujets potentiels.

Solution du problème :

Il n'existe pas beaucoup de manières d'impliquer et de recruter des gens dans une expérience. Il faut toujours garder à l'esprit que l'introduction de biais nuit à la validité de l'expérience. Dans ce cas-ci, la solution adoptée a été de motiver les gens. Lorsqu'une personne ne s'était pas encore inscrite à l'expérience, elle était conviée à trouver une date qui lui conviendrait au mieux parmi les choix proposés sur le site de recrutement.

La carte de la sympathie a semblé être une bonne solution afin d'impliquer le sujet dans l'expérience, sans lui mettre pour autant une certaine pression sur le dos. Chaque sujet a été invité à s'inscrire sur le site suite à une visite de courtoisie. L'e-mail a également été utilisé afin de toucher un public plus large. Il reste cependant moins rentable en termes d'inscriptions que la visite personnelle auprès du futur sujet.

6.4.2 Problème 2 : La prévention des biais potentiels

Explication du problème :

Alors que le **Problème 2** de la phase de *planification* s'attelait à décrire les biais que la conception de l'expérience pouvait introduire, cette section présente les biais pour lesquels il a fallu porter une attention particulière durant la phase d'*exécution*. Ces biais peuvent se présenter sous diverses formes. Leur principale cause d'apparition durant les différentes expériences est la *difficulté de réplication à l'identique* de chaque expérience. Ce problème est en effet relativement difficile à traiter étant donné que chaque sujet se comporte différemment durant l'exécution de l'expérience. Ce comportement influence le déroulement de celle-ci. Il convient d'être vigilant afin de limiter ces influences.

Solution du problème :

Les solutions étudiées, proposées et retenues sont les suivantes :

- Rester attentif et garder le contrôle de l'expérience (temps, déroulement)
- Rester impartial durant la réalisation des tâches («silence radio»)
- Fournir un environnement identique à chaque sujet

Toutes ces actions ont permis aux différentes exécutions de l'expérience de se dérouler presque à l'identique.

Contrôler le temps et le déroulement de l'expérience est une manière d'assurer que chaque sujet ne prendra ni trop de temps, ni trop peu de temps pour effectuer les différentes tâches qui lui sont assignées. Etant donné que l'expérience était planifiée par tranches d'une heure, chaque sujet devait réaliser dans cette heure les trois parties de l'expérience, voir Section 5.4.2. Contrôler le temps et le déroulement permettait ainsi d'éviter tout débordement.

La question de l'impartialité intervient essentiellement durant l'exécution des tâches avec le système d'*eye-tracking*. Chaque participant savait avant de commencer l'expérience, qu'il n'y aurait de réponses pour aucune de leurs questions. Cela évitait ainsi d'influencer le participant dans son raisonnement et ses réponses.

La question de l'environnement est essentielle. Il était important d'assurer qu'aucun facteur externe ne vienne perturber le déroulement de l'expérience. C'est pourquoi, aucune personne n'était autorisée à rentrer dans le laboratoire durant l'expérience. De même, tout téléphone et autres appareils étaient débranchés afin de ne pas distraire le sujet durant son travail. Etant donné que les analyses se sont basées sur le temps de fixation et le nombre de fixations, l'introduction de ce genre de biais aurait mis à mal la validité des données, et pas conséquent, la validité de l'expérience.

6.4.3 Problème 3 : La gestion des problèmes techniques

Explication du problème :

L'utilisation de matériel peut également constituer une source de problèmes lors du déroulement de l'expérience. Le système d'*eye-tracking* utilisé a posé deux problèmes particuliers, un lié aux yeux et un lié aux mouvements des sujets. Ces problèmes ont déjà été introduits dans la Section 5.6.1.

Le premier problème est appelé «*effet nuage de points*». Il est impossible de déterminer, à la fin de l'expérience, quelles ont été les zones qui ont été observées le plus. Des fixations et saccades peuvent être observées dans des régions où il n'y a ni classe, ni association.

Le deuxième problème est une conséquence de la précision de l'appareil. Etant donné que celui-ci est très précis, un simple mouvement de la tête produit une translation (offset) de toutes les coordonnées enregistrées.

Solution du problème :

Il n'existe pas de solutions pour le premier problème. La seule finalité possible est le rejet des données du sujet. Ce genre de problème est évidemment extrêmement coûteux étant donné qu'il est impossible de faire quoi que ce soit sur les données. De plus, ce problème impose de trouver un nouveau sujet afin de conserver l'équilibre au sein des groupes.

Le second problème peut, quant à lui, être traité de manière plus ou moins automatique. Etant donné que ce problème se rattache plus à la phase d'analyses des données, il sera plus détaillé à travers le **Problème 1 : Le traitement des données brutes**, voir Section 6.5.1.

La suppression des données, pour le premier problème, était nécessaire pour assurer la validité des données. Des données résultant d'un problème technique ne sont pas pertinentes pour effectuer des analyses dessus. Cela pourrait biaiser les résultats des analyses et, in fine, les conclusions qui en seraient tirées.

6.5 Analyses et interprétations

6.5.1 Problème 1 : Le traitement des données brutes

Explication du problème :

Après avoir réalisé toutes les expériences sur les sujets, uniquement des fichiers bruts apparaissent. Ces fichiers ne sont pas directement exploitables pour des analyses de données.

Il convient de réaliser certains traitements sur ces données afin de calculer des mesures. Cependant, avant d'effectuer ces traitements, il faut vérifier en amont si les données sont valides. Le logiciel TAUPE [Guéhéneuc 06] permet de représenter graphiquement les fixations et les saccades effectuées sur un diagramme lors d'une expérience. L'utilisation de ce logiciel a permis d'identifier la présence de translations au niveau des coordonnées des fixations et des saccades. L'architecture et le fonctionnement global de TAUPE sont présentés dans l'Annexe B.

Solution du problème :

Suite à l'identification visuelle des différentes translations de coordonnées via le logiciel TAUPE, il a été décidé d'observer les différentes vidéos des expériences. Cette visualisation des vidéos est relativement coûteuse en temps. En effet, pas moins de 144 vidéos de 3 à 7 minutes ont dû être parcourues afin de déterminer sur papier les translations des coordonnées et à quel moment elles devaient être effectuées.

Une fois ces translations identifiées, TAUPE a dû être adapté afin d'intégrer ces translations. Ce logiciel n'a pas servi uniquement à représenter graphiquement les données. Il a également servi à calculer les différentes mesures après avoir agrégé tous les fichiers bruts des expériences. L'intégration des translations se justifie par la précision et la pertinence que les calculs des mesures requièrent.

Etant donné que les résultats bruts ne pouvaient être traités immédiatement, et que les corrections des coordonnées ne constituaient qu'un faible coût de traitement, il a été décidé de parcourir toutes les vidéos. Cela a permis de rendre les données plus représentatives de la réalité. Cependant, cela a introduit une menace à la validité interne de l'expérience. Le travail a donc été effectué avec une certaine rigueur afin de limiter cette menace.

6.5.2 Problème 2 : Le choix de l'angle d'analyse

Explication du problème :

Suite aux différentes données et mesures résultant des premiers traitements, il convient de choisir quels indicateurs de mesures vont être traités lors des analyses et interprétations. Ce choix guidera les analyses qui seront utilisées, mais aussi les interprétations qui pourront être réalisées. Dans cette expérience, le choix s'est posé à deux reprises. Le premier problème concernait la notion de «*normalisation*» des données. Le second problème concernait la définition du concept d'«*effort*».

La question de la «*normalisation*» est apparue lors des analyses sur le NOMBRE DE FIXATIONS TOTAL et ses dérivés. Etant donné que les différentes versions des diagrammes de chaque système comportaient un nombre de classes différent, la pertinence des analyses sur des données non normalisées est apparue.

Concernant la notion d'«*effort*», certains ont pensé que celui-ci se calculait sur base du NOMBRE TOTAL DE FIXATIONS PERTINENTES, d'autres pensaient qu'il se calculait plutôt via le rapport présenté à la Section 5.3.3 – **Variables dépendantes**, Page 74. Dans un cas comme dans l'autre, les interprétations diffèrent et conduisent à des résultats opposés. Considérer une définition unique d'un concept permettra de renforcer l'avis de certains travaux préalablement réalisés. Cela contredira également d'autres avis. Il convient d'être

très prudent lorsque de tels choix sont réalisés.

La solution choisie actuellement présente l'«*effort*» de la manière suivante : «*au plus un sujet réalise un taux de fixations pertinentes élevé, au moins il fait d'effort*». Si cet effort est faible, cela signifie que le diagramme est plus optimal, voir Section 5.3.3 – **Variables dépendantes**, Page 74.

Tel que dit précédemment, la variable actuelle ne garantit pas de refléter les traitements observés. Ceci est du au manque de consensus dans le domaine de l'*eye-tracking*. Cette variable est cependant basée sur une variable tirée d'un article scientifique [Goldberg 99].

6.6 Divers

Explication du problème :

L'utilisation de matériel informatique implique aussi la présence de certains problèmes. Ainsi, plusieurs problèmes techniques (droits d'administrateur, configuration de logiciels, erreurs de compilation particulières, ordinateur en panne, CVS indisponible, etc) ont été rencontrés. Ces problèmes ralentissent le bon déroulement des activités, voire même empêchent toute activité.

Solution du problème :

Face à ces problèmes, différentes attitudes ont été adoptées. Les avis d'autres personnes ont été sollicités afin de répondre rapidement aux problèmes techniques (erreurs de compilations, configuration de logiciels, etc). Cependant, pour les problèmes de matériels purs, l'attitude adoptée a été celle de la patience. Ces problèmes techniques sont à prendre en compte lors des différentes phases afin de limiter l'improductivité qu'elles entrainent.

6.7 Résumé

Ce chapitre a permis de montrer les nombreux problèmes que la conduite de cette expérience a rencontrés.

Malgré de nombreuses discussions réalisées avec des experts dans les domaines étudiés, l'apparition de biais n'a pu être évitée. L'introduction de biais est apparue aux différentes étapes de cette expérience. C'est notamment le cas des problèmes liés à l'utilisation du système d'*eye-tracking* considéré. Alors que la conception de l'expérience a dû prendre en compte les contraintes imposées par ce type de système, certaines données ont toutefois dû être supprimées, suite à un problème technique.

Ce chapitre a montré également que la conception en fonction de la technique et en fonction des contraintes externes, dans ce cas liées à l'éthique, peuvent affecter la validité de l'expérience.

7 Leçons apprises

Sommaire

Avant-propos

Réaliser une expérience permet à l'expérimentateur de tirer certaines conclusions. Ces dernières ne concernent pas uniquement les données récoltées durant les expériences. Il est également possible de tirer des conclusions sur les moyens techniques utilisés durant l'expérience ainsi que sur le processus en tant que tel.

*Ce chapitre présente les diverses leçons apprises au niveau de l'utilisation d'un système d'*eye-tracking *et au niveau de la conduite d'une expérience.*

7.1 Leçons concernant le système d'eye-tracking

Cette section présente les différentes constations réalisées au niveau du système d'*eye-tracking* utilisé mais également sur cette technologie en général.

- Le concept d'*eye-tracking* est un excellent moyen pour identifier les éléments d'une interface graphique qui retiennent le plus l'attention des personnes. Il est intéressant d'observer à quel point les résultats peuvent être surprenants et souvent révélateurs. Une première observation rapide et grossière permet généralement d'identifier sans ambiguïté les zones ayant attiré le plus l'attention d'une personne.
- Ce concept est également un moyen intéressant pour étudier les processus cognitifs des personnes. Grâce aux données collectées, il est possible de reconstituer le raisonnement d'une personne. Ces raisonnements peuvent ensuite être analysés afin de guider et d'améliorer les recherches de l'expérimentateur.

L'utilisation d'un système d'*eye-tracking* peut s'avérer très utile selon les buts poursuivis. Cependant, après utilisation du matériel présenté dans ce mémoire, certains éléments peuvent remettre en question la présence d'un tel matériel au sein d'une expérience. Certains points peuvent entrer en conflit avec la rigueur que requiert une expérience. Voici quelques points négatifs du système d'*eye-tracking* utilisé. Ces points ont mis à mal le déroulement et la rigueur nécessaire durant les expériences.

- L'utilisation prolongée du système cause une gêne au niveau des yeux. Plusieurs sujets ont signalé une légère irritation de leurs yeux à la fin des expériences. Aucun élément n'a pu déterminer la cause exacte de cette irritation. La **première hypothèse** logique qui pourrait expliquer cette irritation est la présence de rayons infra-rouge. La **deuxième hypothèse** permettant d'expliquer cette gêne est tout simplement le résultat d'une concentration accrue, opérée au niveau des yeux. Finalement, la **troisième hypothèse** est peut-être un effet psychosomatique.
- L'utilisation prolongée du casque entraine des raideurs dans le cou ou la nuque. Ce problème est une conséquence directe des contraintes imposées par le système utilisé. En adoptant une position particulière et en étant contraint de la conserver durant l'expérience, les participants ont été victimes de cette immobilité.
- Les sujets ont une tendance naturelle à bouger la tête. Ils effectuent tous un geste particulier lorsqu'ils se concentrent (remuer la jambe, se toucher le visage, bouger les mains, etc.). Ce point mérite d'être mentionné avec autant d'importance que les autres, à partir du moment où tout mouvement avec ce type de système doit être considéré comme le problème à traiter en premier lieu.
- Le temps d'une expérience ne doit pas excéder 20 minutes, sans quoi, les sujets se fatiguent et n'agissent plus de manière naturelle. Ce temps semble très court et est d'ailleurs peu représentatif des tâches que les professionnels réalisent habituellement. Néanmoins, si la simulation d'une tâche peut ne pas dépasser 20 minutes et conserver pour autant toute sa pertinence, elle doit être réalisée de la sorte.

Tous ces points témoignent des limites du système d'*eye-tracking* utilisé. La présence d'un casque restreint le nombre de tâches, leur complexité, leur durée. Ces restrictions affectent la validité de l'expérience. Il convient plutôt d'utiliser un autre appareillage appelé «*head supported system*», si des expériences doivent être réalisées sur des tâches fastidieuses et de longue durée. Cet appareillage est une autre variante que le type de système utilisé lors de cette expérience, à savoir un «*head mounted system*».

Une dernière constatation concernant l'utilisation de systèmes d'*eye-tracking* munis d'une caméra : ce type de système peut se heurter à certaines clauses d'anonymat telles que présentées en Section 6.3.4. Ce genre de problème peut paraître anodin mais peut en réalité faire perdre un temps considérable. La recherche d'une solution alternative peut faire courir le risque d'obtenir une conception de l'expérience moins solide vis à vis des critiques. Au pire des cas, l'expérimentateur aura perdu son temps et sera obligé d'abandonner toutes les informations biaisées qu'il aura récoltées.

7.2 Leçons concernant l'expérience

Cette section liste les différentes leçons apprises au niveau du processus d'une expérience.

- Le temps de conception d'une expérience est relativement plus élevé par rapport à celui planifié. Différents problèmes techniques, réflexions, situations inattendues sont inévitables et retardent les différentes étapes du processus, notamment le début de la phase d'exécution.
- Il est très difficile de se mettre à la place des sujets lors de la conception des tâches. La complexité des questions et le temps de réponse doivent être ainsi évalués par une, voire des tierces personnes.
- Réaliser un ou des tests pilotes est un excellent moyen de se faire une idée du fonctionnement global de l'expérience. Sur base de ces tests pilotes, il est ainsi possible de conserver la conception de l'expérience ou de modifier certains éléments de cette conception.
- Les sujets sont très «*coûteux*». Ils sont très difficiles à trouver et à motiver. Il faut cependant prêter attention à l'introduction de biais suite aux différentes techniques de motivation utilisées.
- Des situations inattendues apparaissent relativement fréquemment (coup de téléphone, personne s'introduisant dans le local, sujet fatigué, etc). Il convient d'être prêt à faire face à toute situation afin d'assurer le bon suivi de l'expérience.
- Il faut également être très prudent dans la *manipulation* de données. L'utilisation d'appareils peut nécessiter certaines corrections au niveau des données qui doivent être réalisées avec beaucoup de vigilance.
- Les analyses des données doivent également être considérées avec beaucoup de prudence. L'angle sous lequel les analyses sont effectuées détermine les résultats, et par conséquent, les conclusions qui seront tirées de l'expérience (voir Section 6.5.2).
- L'introduction de biais est cependant inévitable. Il faut tenter de les réduire au maximum afin d'obtenir une expérience la plus valide possible.
- Si l'expérience est bien conçue, elle facilite sa réplication soit avec d'autres sujets, soit avec des objets similaires à observer.
- L'utilisation d'un certain matériel peut limiter l'expérience au niveau du temps et des tâches à réaliser. Cela doit être pris en compte lors de la phase de conception.
- Être clair et transparent améliore la confiance des gens lors de l'expérience, ce qui limite le biais de la conduite d'une expérience en tant que telle.
- Après avoir interprété les données récoltées, il est possible que l'on s'aperçoive que les variables dépendantes considérées ne permettent pas d'exprimer le ou les facteurs étudiés. Dans ce cas, si les données le permettent, il est possible de modifier ces variables dépendantes et d'interpréter à nouveau les données récoltées, sur base de ces nouvelles variables. Néanmoins, il est important de ne pas tomber dans le coté pervers de cette pratique et d'en arriver à ce qu'on appelle communément le «*fishing*».

Cette pratique ne doit pas avoir sa place dans une démarche scientifique. Elle consiste à changer sans cesse la définition d'une variable dans le but d'obtenir des résultats satisfaisant aux yeux de l'expérimentateur.

De manière plus globale,

- Il ne faut pas se laisser dominer par la «*technique*». Pour cela il est nécessaire de prendre le temps de se renseigner sur les fonctionnalités des appareils utilisés. Ces derniers joueront en effet un rôle clé dans l'expérience. Il convient d'utiliser des appareils appropriés aux tâches qui veulent être étudiées.
- Il faut prendre le temps de bien réfléchir sur l'introduction de biais. Ce point a été largement discuté au travers des différents chapitres afin d'exprimer l'importance de ce concept.

Conclusions et travaux futurs

Conclusions

Ce mémoire avait pour objectif (1) d'appréhender le contexte de la recherche empirique en génie logiciel (GL), et (2) d'identifier les problèmes qui peuvent survenir dans un tel type de recherche.

Afin de se familiariser rapidement avec la recherche empirique en GL, une expérience a été menée au sein du D.I.R.O de l'Université de Montréal. Cette expérience avait pour objectif d'évaluer l'impact du patron de conception (design pattern) «*Visiteur*» sur la compréhension et la maintenance de logiciels orientés-objet (OO). Des diagrammes de classes UML et un système d'*eye-tracking* ont été utilisés afin de pouvoir traiter le problème posé.

Le Chapitre 1 a présenté la maturité actuelle de la recherche empirique en GL. Sur base de l'expérience réalisée, un certain nombre de problèmes généraux peuvent être identifiés. Ces problèmes tendent à confirmer en grande partie les éléments négatifs que les experts du domaine ont identifiés et qu'ils désirent améliorer en créant des études empiriques plus rigoureuses.

Le premier grand problème concerne la conduite d'une expérience du début jusqu'à sa fin. Conduire une expérience implique des objectifs clairs, des moyens adéquats (matériaux, sujets, contexte, etc.) et une connaissance accrue du domaine de recherche. Or cette expérience a montré à plusieurs reprises que des changements, notamment au niveau de la définition des objectifs et des variables dépendantes, avaient été opérés. Ces problèmes impactent négativement sur la rigueur et, in fine, sur la validité de l'expérience.

Le besoin de changement de la définition de l'expérience témoigne d'un manque de clarté et de rigueur à différents niveaux : au niveau des objectifs poursuivis, des mesures utilisées, de la définition des attributs de qualité étudiés, des processus cognitifs intervenant. Ce besoin changement témoigne également que l'interprétation peut varier fortement d'une personne à l'autre. Utiliser de bonnes pratiques pour la définition de l'étude, conserver un vocabulaire commun et reconnu par la communauté, notamment en ce qui concerne les concepts manipulés, les attributs étudiés et leurs mesures, devraient permettre de présenter, le plus clairement possible, ce que la recherche vise à étudier, mais également ce qu'elle ne vise pas à étudier.

La modification soutenue des variables dépendantes liées à l'*eye-tracking* a fourni plusieurs preuves évidentes que : (1) le choix des variables n'est pas anodin, (2) un manque de consensus persiste dans le domaine de l'*eye-tracking*. De plus, aucun élément ne permet d'assurer que les variables issues de l'*eye-tracking* peuvent être utilisées comme des indicateurs pour évaluer la qualité.

Finalement, il semblerait que les vertus conférées aux patrons de conception ne se basent sur aucun modèle rigoureux ou autre consensus. Une uniformisation de la perception et de la mesure des effets positifs et négatifs de ces patrons de conception permettrait de guider les recherches afin de tester ces propos. De plus, cette uniformisation permettrait d'ouvrir la voie qui lierait plus rigoureusement les patrons de conception à la qualité.

L'absence de rapports bien établis entre les patrons de conceptions et la qualité, tout comme le rapport entre les indicateurs de la qualité et l'*eye-tracking*, sont des exemples concrets des débats qui doivent encore être menés avant de pouvoir assurer à grande échelle la conduite de recherches empiriques solides.

Les résultats obtenus lors des analyses des différentes variables dépendantes apportent une preuve tangible de ce problème. Ces résultats sont pour le moins perturbants. En interprétant les données avec une variable dépendante particulière, il est possible de valider une des hypothèses nulles formulées. En interprétant ces mêmes données avec une autre variable dépendante, il est possible de réfuter cette même hypothèse nulle. Il semblerait que l'interprétation de la variable choisie ait des conséquences bien plus lourdes qu'il n'y parait. C'est pourquoi des consensus doivent être trouvés afin d'éviter de se retrouver avec plusieurs expériences se contredisant les unes les autres et pour lesquels aucun argument solide ne permettrait d'en favoriser une plus que les autres.

Tant que les domaines étudiés ne pourront être liés d'une manière ou d'une autre à la théorie de la mesure, il sera difficile de conduire des expériences rigoureuses sur base desquelles des interprétations valides pourront être exposées.

En résumé, la plupart des problèmes identifiés dans ce mémoire résultent en partie d'une certaine inexpérience mais surtout d'un manque de maturité certain qui sévit dans les différents domaines abordés. Afin d'éviter de tomber dans une situation semblable à celle présentée dans ce mémoire, il est conseillé au futur expérimentateur d'établir assez rapidement un état de l'art des différents domaines abordés au cours de son étude empirique. Cet état de l'art permettra d'identifier les éléments qui mériteront une attention toute particulière. En agissant de la sorte, il sera possible de renforcer autant que possible la validité de l'étude empirique mise en oeuvre.

Travaux futurs

La réalisation de l'expérience suscite plusieurs questions. Certains points méritent d'être approfondis, d'autres méritent d'être explorés. Finalement, certains travaux moins fastidieux méritent également une attention particulière.

Suite aux résultats prometteurs, la première tâche à réaliser serait peut-être de conduire à nouveau cette expérience avec de nouveaux sujets, tout en étant plus vigilant face aux remarques formulées. Répliquer une expérience est un excellent moyen de s'assurer que les résultats obtenus ne proviennent pas d'une situation fortuite.

Bien évidemment, cette expérience suscite instinctivement des questions sur les variantes de la conception présentée. Cette expérience a montré que ses résultats ne pouvaient être généralisés suffisamment. Etudier d'autres patrons de conception est, sans conteste, un des

travaux qui devra être réalisé dans le futur. Celui-ci permettrait de synthétiser les diverses observations et d'identifier des propriétés communes ou particulières à certaines situations et à certains patrons de conception.

La question de la taille des diagrammes mérite également d'être étudiée. Il est difficile de trouver des articles traitant des patrons de conception et de l'aspect «*taille des diagrammes*». Or, les expériences généralement réalisées se basent sur une taille de diagramme relativement petite par rapport aux diagrammes utilisés dans le monde industriel. Nous ne savons pas comment les patrons de conception se comportent avec des diagrammes de plus grande taille.

Beaucoup d'études examinent l'impact des patrons de conception sur divers facteurs. Cependant, ces études se déroulent pour la plupart avec la présence d'un seul patron de conception par diagramme. Une future voie de recherche pourrait s'atteler à étudier la combinaison de patrons de conception au sein d'un diagramme et les problèmes que cette combinaison implique. D'autres qualités peuvent également être étudiées conformément au modèle ISO 9126 suivi dans cette expérience.

Concernant le matériel d'*eye-tracking*, au moins deux voies de recherche peuvent être suivies. Cette expérience a montré ses limites au niveau de l'interaction que les sujets pouvaient avoir avec les diagrammes. La première voie serait d'utiliser la partie dynamique du système d'*eye-tracking* considéré. La seconde voie pourrait être abordée suite aux problèmes techniques rencontrés et aux limites que le système impose. Un autre appareillage (caméra, *head supported system*, etc.) pourrait être utilisé afin de réaliser des tâches plus longues, sur des interfaces dynamiques. Ceci augmenterait d'avantage la validité de l'expérience.

Il est important également, suite aux critiques reçues, de renforcer la validité de l'expérience. Cela implique d'appuyer le côté «*systématique*» de celle-ci et d'éviter tout facteur permettant de formuler une critique. Ainsi, les diagrammes étudiés devraient contenir le même nombre de classes. Si des *layouts* doivent être étudiés, il est conseillé d'appliquer plutôt des *layouts* reconnus via des outils de transformation systématique.

Finalement, cette expérience a montré à plusieurs reprises les problèmes apparus suite au manque de connaissance dans le monde de la compréhension de programmes. Plusieurs futures études peuvent être menées, soit au niveau du code, soit au niveau du *design* afin de comprendre comment les sujets raisonnent en présence des patrons de conception.

Il peut être intéressant de trouver également des motifs au niveau des chemins (*scanpaths*) que les sujets effectuent lorsqu'ils analysent un objet à l'écran. Des motifs peuvent être obtenus notamment grâce aux techniques utilisées dans les graphes. Cette analyse par les graphes peut s'obtenir en implémentant des algorithmes dans l'outil Taupe, logiciel introduit au cours de ce mémoire, voir Annexe B. Les différents mesures présentées par Goldberg *et al.* semblent également être pertinentes pour des recherches futures. Ces mesures méritent également d'être implémentées dans Taupe afin d'automatiser la production de résultats basés sur des mesures reconnues par la communauté au niveau de l'*eye-tracking*.

Bibliographie

[Ahern 05] Dennis M. Ahern, Jim Armstrong, Aaron Clouse, Jack R. Ferguson, Will Hayes & Kenneth E. Nidiffer. CMMI SCAMPI Distilled Appraisals for Process Improvement. Addison Wesley Professional, 2005. 7

[Arisholm 04] Erik Arisholm & Dag I. K. Sjoberg. *Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software*. IEEE Transactions on Software Engineering, vol. 30, no. 8, pages 521–534, 2004. 23

[Arisholm 07] Erik Arisholm, Hans Gallis, Tore Dyba & Dag I.K. Sjoberg. *Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise*. IEEE Transactions on Software Engineering, vol. 33, no. 2, pages 65–86, 2007. 23

[Aversano 07] L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso & M. Penta. *An Empirical Study on the Evolution of Design Patterns*. In 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on The Foundations of Software Engineering, pages 385–394, 2007. 50, 51, 52, 54

[Avison 99] David E. Avison, Francis Lau, Michael D. Myers & Peter Axel Nielsen. *Action Research*. Commun. ACM, vol. 42, no. 1, pages 94–97, 1999. 14

[Babbie 90] E. Babbie. Survey Research Methods. Wadsworth, 1990. ISBN 0-524-12672-3. 13

[Basili 85] Victor R. Basili. *Quantitative Evaluation of Software Engineering Methodology*. In Proceedings of the First Pan Pacific Computer Conference, volume 1, pages 379–398, Melbourne, Australia, 1985. 7

[Basili 86] Victor R. Basili, R. W. Selby & D. H. Hutchens. *Experimentation in Software Engineering*. IEEE Transactions on Software Engineering, vol. 12(7), pages 733–743, 1986. 10, 22

[Basili 93] Victor R. Basili. The Experimental Paradigm in Software Engineering, Experimental Software Engineering Issues : Critical Assessment and Future Directives. Springer-Verlag, 1993. 9, 10

[Basili 96] Victor R. Basili. *The Role of Experimentation in Software Engineering : Past, Current, and Future*. In Proceedings of the 18th International Conference on Software Engineering (ICSE'96), pages 442–449. IEEE Computer Society, 1996. 6, 8, 10, 21

[Basili 99a] Victor R. Basili, Laszlo Belady, Barry Boehm, Frederick Brooks, James Browne, Richard DeMillo, Stuart I. Feldman, Cordell Green, Butler Lampson, Duncan Lawrie, Nancy Leveson, Nancy Lynch, Mark Weiser & Jeannette Wing. *NSF Workshop on a Software Research Program*

for the 21st Century. SIGSOFT Software Engineering Notes, vol. 24, no. 3, pages 37–44, 1999. 6, 8, 9, 21

[Basili 99b] Victor R. Basili, Forrest Shull & Filippo Lanubile. *Building Knowledge through Families of Experiments*. IEEE Transactions on Software Engineering, vol. 25, no. 4, pages 456–473, 1999. 10

[Basili 00] Victor R. Basili, Forrest Shull & Filippo Lanubile. *Using Experiments to Build a Body of Knowledge*. In Proceedings of the Third International Andrei Ershov Memorial Conference on Perspectives of System Informatics (PSI '99), pages 265–282, London, UK, 2000. Springer-Verlag. 10, 22

[Basili 02] Victor R. Basili, Frank E. McGarry, Rose Pajerski & Marvin V. Zelkowitz. *Lessons learned from 25 years of process improvement : the rise and fall of the NASA software engineering laboratory*. In Proceedings of the 24th International Conference on Software Engineering (ICSE '02), pages 69–79, New York, NY, USA, 2002. ACM. 10

[Beck 96] Kent Beck, Ron Crocker, Gerard Meszaros, John Vlissides, James O. Coplien, Lutz Dominick & Frances Paulisch. *Industrial Experience with Design Patterns*. In Proceedings of the 18th International Conference on Software Engineering (ICSE '96), pages 103–114, Washington, DC, USA, 1996. IEEE Computer Society. 2, 52

[Bednarik 06] Roman Bednarik & Markku Tukiainen. *An eye-tracking methodology for characterizing program comprehension processes*. In ETRA '06 : Proceedings of the 2006 symposium on Eye tracking research & applications, pages 125–132, New York, NY, USA, 2006. ACM. 60, 61, 76, 141

[Benbasat 99] Izak Benbasat & Robert W. Zmud. *Empirical research in information systems : the practice of relevance*. MIS Quaterly, vol. 23, no. 1, pages 3–16, 1999. 22

[Boehm 05] B. Boehm, H. D. Rombach & M. V. Zelkowitz. Foundations of Empirical Software Engineering Experiments : The Legacy of Victor R. Basili. Springer-Verlag, 2005. 10

[Briand 96a] Lionel C. Briand, C. M. Differding & H. D. Rombach. *Practical Guidlines for Measurement-Based Process Improvement*. Software Process Improvement and Practice, vol. 2(4), pages 253–280, 1996. 28

[Briand 96b] Lionel C. Briand, K. El Emam & S. Morasca. *On the Application of Measurement Theory in Software Engineering*. Journal of Empirical Software Engineering, vol. 1(1), pages 61–88, 1996. 18

[Briand 06] Lionel C. Briand, Yvan Labiche & Alexandre Sauve. *Guiding the Application of Design Patterns Based on UML Models*. In Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM '06), pages 234–243, Washington, DC, USA, 2006. IEEE Computer Society. 49, 52

[Brilliant 99] Susan S. Brilliant & John C. Knight. *Empirical research in software engineering : a workshop*. SIGSOFT Software Engineering Notes, vol. 24, no. 3, pages 44–52, 1999. 10

[Buschmann 96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad & M. Stal. Pattern-Oriented Software Architecture–A System of Patterns. John Wiley & Sons, 1996. 50

[Campbell 63] D. T. Campbell & J. C. Stanley. *Experimental and Quasi-Experimental Design for Research.* 1963. 38, 39

[Cline 96] Marshall P. Cline. *The Pros and Cons of Adopting and Applying Design Datterns in the Real World.* Communications of the ACM, vol. 39, no. 10, pages 47–49, 1996. 50, 51, 52, 53

[Cook 79] T. D. Cook & D. T. Campbell. *Quasi-Experimentation – Design and Analysis Issues for Field Settings.* 1979. 38, 39

[Creswell 94] J. W. Creswell. Research Design, Qualitative and Quantitative Approaches. Sage Publications, 1994. 12

[Cutrell 07] Edward Cutrell & Zhiwei Guan. *What are you looking for ? : an eye-tracking study of information usage in web search.* In CHI '07 : Proceedings of the SIGCHI conference on Human factors in computing systems, pages 407–416, New York, NY, USA, 2007. ACM. 61

[DeMarco 82] T. DeMarco. Controlling Software Projects. Yourdon Press, New York, USA, 1982. 16

[Demming 86] E. Demming. Out of the Crisis. MIT Press, MIT Center for Advanced Engineering Study, 1986. 7

[Denzin 94] N. K. Denzin & Y. S. Lincoln. Handbook of Qualitative Research. Sage Publications, London, UK, 1994. 12

[Duchowski 03] Andrew T. Duchowski. Eye Tracking Methodology : Theory and Practice. Springer-Verlag, 2003. 2, 60

[Eichelberger 03] Holger Eichelberger. *Nice class diagrams admit good design ?* In SoftVis '03 : Proceedings of the 2003 ACM symposium on Software visualization, pages 159–ff, New York, NY, USA, 2003. ACM. 58

[Eiglsperger 03] Markus Eiglsperger, Michael Kaufmann & Martin Siebenhaller. *A topology-shape-metrics approach for the automatic layout of UML class diagrams.* In SoftVis '03 : Proceedings of the 2003 ACM symposium on Software visualization, pages 189–ff, New York, NY, USA, 2003. ACM. 58

[Fenton 94] N. Fenton, S. L. Pfleeger & R. Glass. *Science and Substance : A Challenge to Software Engineers.* IEEE Software, pages 86–95, Juillet 1994. 17, 18, 21, 23

[Fenton 96] N. Fenton & S. L. Pfleeger. Software Metrics : A Rigorous & Practical Approach. International Thomson Computer Press, 2^{e} edition, 1996. 16, 18

[Gamma 94] E. Gamma, R. Helm, R. Johnson & J. Vlissides. Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley, 1er edition, 1994. 49, 50, 51, 52, 53

[Glass 94] R. Glass. *The Software Research Crisis.* IEEE Software, pages 42–47, Novembre 1994. 9

[Glass 02] R. L. Glass, I. Vessey & V. Ramesh. *Research in Software Engineering : An Analysis of the Literature.* Journal of Information and Software Technology, vol. 44(8), pages 491–506, 2002. 21

[Goldberg 99] Joseph H. Goldberg & Xerxes P. Kotval. *Computer interface evaluation using eye movements : methods and constructs.* Journal of Industrial Ergonomics, vol. 24, pages 631–645, 1999. 61, 104, 141

[Greenwood 06] D. J. Greenwood & M. Levin. Introduction to action research : Social research for social change. Thousand Oaks, California, 2e edition, 2006. 14

[Guéhéneuc 06] Yann-Gaël Guéhéneuc. *TAUPE : Towards Understanding Program Comprehension.* In H. Erdogmus & E. Stroulia, editeurs, Proceedings of the 16th IBM Centers for Advanced Studies Conference, pages 1–13. ACM Press, Octobre 2006. xiii, 58, 59, 83, 103, 134, 140

[Habra 08] Naji Habra, Alain Abran, Miguel Lopez & Asma Sellami. *A framework for the design and verification of software measurement methodes.* The Journal of Systems and Software (ElSEVIER), vol. 81, pages 633–648, 2008. 16, 17, 18, 20

[Heymans 05] Patrick Heymans. *Analyse et Modélisation des Systèmes d'Informations.* Syllabus d'A.M.S.I., 2005. 58

[Höfer 07] A. Höfer & W. F. Tichy. Status of Empirical Research in Software Engineering. In : Basili et al. (eds), Experimental Software Engineering Issues : Assessment and Future Directions. Springer-Verlag, 2007. 22

[Hsueh 08] Nien-Lin Hsueh, Peng-Hua Chu & William Chu. *A quantitative approach for evaluating the quality of design patterns.* The Journal of Systems and Software (ElSEVIER), vol. 81, pages 633–648, 2008. 50, 52

[IEEE 90] IEEE. *"IEEE Standard Glossary of Software Engineering Terminology", IEEE Std 610.12-1990*, 1990. 6

[ISO VIM 93] ISO VIM. *International Vocabulary of Basic and General Terms in Metrology.* Rapport technique, International Organization for Standardization – ISO, 1993. 17

[ISO VIM 04] ISO VIM. *International Vocabulary of Basic and General Terms in Metrology (Draft).* Rapport technique, International Organization for Standardization – ISO, 2004. 18

[ISO/IEC TR 03a] ISO/IEC TR. *Software Eengineering – Product Quality – Part 2 : External Metrics.* Rapport technique, International Organization for Standardization – ISO, 2003. 17

[ISO/IEC TR 03b] ISO/IEC TR. *Software Eengineering – Product Quality – Part 3 : Internal Metrics.* Rapport technique, International Organization for Standardization – ISO, 2003. 17

[ISO/IEC TR 04] ISO/IEC TR. *Software Eengineering – Product Quality – Part 4 : Quality in Use Metrics.* Rapport technique, International Organization for Standardization – ISO, 2004. 17

[ISO/IEC 01] ISO/IEC. *Software Eengineering – Product Quality – Part 1 : Quality Model.* Rapport technique, International Organization for Standardization – ISO, 2001. 17, 54, 55

[JHotDraw 07] JHotDraw. *JHotDraw Start Page.* http://www.jhotdraw.org/, 2007. 79

[Johansen 06] Sune Alstrup Johansen & John Paulin Hansen. *Do We Need Eye Trackers to Tell Where People Look ?* In CHI '06 : CHI '06 extended abstracts on Human factors in computing systems, pages 923–928, New York, NY, USA, 2006. ACM. 61, 141

[JRefactory 04] JRefactory. *JRefactory Start Page*. http://jrefactory.sourceforge.net/, 2004. 79

[Kachigan 86] S. K. Kachigan. Statistical Analysis : An Interdisciplinary Introduction to Univariate & Multivariate Methods. Radius Press, New York, 1986. 19

[Lott 96] C. M. Lott & H. D. Rombach. *Repeatable Software Engineering Experiments for Comparing Defect-Detection Techniques*. Journal of Systems and Software, vol. 1(3), pages 241–277, 1996. 28

[Murphy 05] Gail C. Murphy, Mik Kersten, Martin P. Robillard & Davor Čubraniś. *The emergent structure of development tasks*. In Andrew P. Black, editeur, Proceedings of the 19th European Conference on Object-Oriented Programming, pages 33–48. Springer-Verlag, Juillet 2005. 60

[Nielsen 00] Jakob Nielsen. *Why You Only Need to Test With 5 Users*. http://www.useit.com/alertbox/20000319.html, 2000. 70, 94

[Nielsen 06] Jakob Nielsen. *Quantitative Studies : How Many Users to Test ?* http://www.useit.com/alertbox/quantitative_testing.html, 2006. 77

[Noda 01] Natsuko Noda & Tomoji Kishi. *Design Pattern Concerns for Software Evolution*. In Proceedings of the 4th International Workshop on Principles of Software Evolution (IWPSE '01), pages 158–161, New York, NY, USA, 2001. ACM. 50

[Noirhomme 04] Monique Noirhomme. *Probabilités et Statistiques*. Syllabus de probabilités et statistiques, 2004. 33, 43, 44

[OMG 08] OMG. *Unified Modeling Language Specification – Version 2.0*. http://www.uml.org/technology/documents/formal/uml.htm, 2008. 58

[Pan 07] Bing Pan, Helene Hembrooke, Thorsten Joachims, Lori Lorigo, Geri Gay & Laura Granka. *In Google We Trust : Users' Decisions on Rank, Position, and Relevance*. Journal of Computed-Mediated Communication, vol. 12, pages 801–823, 2007. 60, 76

[Perry 00] Dewayne E. Perry, Adam A. Porter & Lawrence G. Votta. *Empirical Studies of Software Engineering : A Roadmap*. In Proceedings of the Conference on The Future of Software Engineering, pages 345–355, 2000. iii, 1, 6, 8, 10, 11, 21, 24, 47, 48

[Pfleeger 95] S. Pfleeger. *Experimental Design and Analysis in Software Engineering Part 1-5*. ACM Sigsoft, Software Engineering Notes, pages Vol. 19,No. 4,pages 16–20 ; Vol. 20,No. 1,pages 22–26 ; Vol. 20,No. 2,pages 14–16 ; Vol. 20,No. 3,pages 13–15 and Vol. 20,No. 4,pages 14–17, 1994-1995. 13

[Pfleeger 97] Shari Lawrence Pfleeger, Ross Jeffery, Bill Curtis & Barbara Kitchenham. *Status Report on Software Measurement*. IEEE Software, vol. 14, no. 2, pages 33–43, 1997. 50

[Pfleeger 01] Shari Lawrence Pfleeger. Software Engineering–Theory and Practice. Prentice Hal, 2e edition, 2001. 6, 49

[Prechelt 01] Lutz Prechelt, B. Unger, W. F. Tichy, P. Brössler & L. G. Votta. *A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions*. IEEE Transactions on Software Engineering, vol. 27, no. 12, pages 1134–1144, 2001. 49, 50, 51, 52, 54, 71

[Prechelt 02] Lutz Prechelt, Barbara Unger-Lamprecht, Michael Philippsen & Walter F. Tichy. *Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance.* IEEE Transactions on Software Engineering, vol. 28, no. 6, pages 595–606, 2002. 49, 50, 51, 52

[Pressman 00] Roger S. Pressman. Software Engineering–A Practitioner's Approach. Mc Graw Hill, 5^{e} edition, 2000. 6, 7

[Rayner 98] Keith Rayner. *Eye Movements in Reading and Information Processing : 20 Years of Research.* Psychological Bulletin 124, pages 372–422, 1998. 2, 58, 59, 61

[Robson 93] C. Robson. Real World Reseach : A Resource for Social Scientists and Practitioners-Researchers. Blackwell, 1993. 13, 34, 76

[Seemann 97] Jochen Seemann. *Extending the Sugiyama algorithm for drawing UML class diagrams : Towards automatic layout of object-oriented software diagrams.* In Giuseppe Di Battista, editeur, Proceedings of the 5th International symposium on Graph Drawing, pages 415–424. Springer-Verlag, Septembre 1997. 60

[SGI 94] SGI. *The CHAOS Report.* Rapport technique, Standish Group International, 1994. 6, 8

[Sim 06] Susan Elliott Sim, Sukanya Ratanotayanon, Oluwatosin Aiyelokun & Erin Morris. *An Initial Study to Develop an Empirical Test for Software Engineering Expertise.* Rapport technique UCI-ISR-06-6, Institute for Software Research Technical, 2006. 98

[Sjoberg 05] Dag I. K. Sjoberg, Jo E. Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, Nils-Kristian Liborg & Anette C. Rekdal. *A Survey of Controlled Experiments in Software Engineering.* IEEE Transactions on Software Engineering, vol. 31, no. 9, pages 733–753, 2005. 22, 23, 70

[Sjoberg 07] Dag I. K. Sjoberg, Tore Dybå & Magne Jørgensen. *The Future of Empirical Methods in Software Engineering Research.* In International Conference on Software Engineering – Future of Software Engineering (FOSE'07), pages 358–378. IEEE Computer Society, 2007. 6, 10, 11, 12, 14, 21, 22, 23, 70, 94

[Soloway 86] Elliot Soloway. *Learning to program = Learning to construct mechanisms and explanations.* Communications of the ACM, vol. 29(9), pages 850–858, Septembre 1986. 60

[Tichy 95] W. F. Tichy, P. Lukowicz, L. Prechelt & E. A. Heinz. *Experimental Evaluation in Computer Science : A Quantitative Study ?* Journal of Systems and Software, vol. 28(1), pages 9–18, 1995. 21, 28

[Tichy 98] W. F. Tichy. *Should Computer Scientists Experiment More ?* IEEE Computer, vol. 31(5), pages 32–39, 1998. 28

[Tichy 00] Walter F. Tichy. *Hints for Reviewing Empirical Work in Software Engineering.* Empirical Software Engineering, vol. 5, no. 4, pages 309–312, 2000. 23

[Trese 07] Tim Trese & Scott Tilley. *Documenting Software Systems with Views V : Towards Visual Documentation of Design Patterns as an Aid to*

Program Understanding. In Proceedings of the 25th annual ACM International Conference on Design of Communication (SIGDOC '07), pages 103–112, New York, NY, USA, 2007. ACM. 52

[Trochim 99] W. Trochim. The Research Methods Knowledge Base. Cornell Custom Publishing, Cornell University, Ithaca, New York, 2ième edition, 1999. xi, 26, 38

[Vokac 04] Marek Vokac, Walter Tichy, Dag I. K. Sjøberg, Erik Arisholm & Magne Aldrin. *A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns—A Replication in a Real Programming Environment*. Empirical Software Engineering, vol. 9, no. 3, pages 149–195, 2004. 54, 71

[Wendorff 01] P. Wendorff. *Assessment of Design Patterns during Software Reengineering : Lessons Learned from a Large Commercial Project*. In Proceedings of 5th Conference on Software Maintenance and Reengineering, pages 77–84. IEEE Computer Society Press, Mai 2001. 49, 50, 51, 52, 71

[Wikipedia 08a] Wikipedia. *Boites à moustaches*. http://fr.wikipedia.org/wiki/Bo%C3%AEte_%C3%A0_moustaches, 2008. 44

[Wikipedia 08b] Wikipedia. *Statistique descriptive*. http://fr.wikipedia.org/wiki/Statistique_descriptive#Quartiles, 2008. 44

[Wohlin 00] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell & Anders Wesslén. Experimentation In Software Engineering : An Introduction. Kluwer Academic Publishers, 2000. 2, 3, 5, 8, 10, 14, 15, 16, 17, 23, 25, 33, 39, 43, 44, 45, 68, 69, 73, 76, 78, 92

[Yusuf 07] Shehnaaz Yusuf, Huzefa Kagdi & Jonathan I. Maletic. *Assessing the Comprehension of UML Class Diagrams via Eye Tracking*. In Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC '07), pages 113–122, Washington, DC, USA, 2007. IEEE Computer Society. 59, 60, 61, 76, 83

[Zannier 06] Carmen Zannier, Grigori Melnik & Frank Maurer. *On the Success of Empirical Studies in the International Conference on Software Engineering*. In Proceedings of the 28th International Conference on Software Engineering (ICSE'06), pages 341–350. ACM Press, 2006. 14, 21, 24

[Zelkowitz 98] M. V. Zelkowitz & D. R. Wallace. *Experimental Models for Validating Technology*. IEEE Computer, vol. 31(5), pages 23–31, 1998. 6, 9, 12, 13, 14, 21, 24, 28

A Article soumis à la conférence «*ACM SIGSOFT 2008/FSE 16*»

A Study of the Impact of Design Patterns on Program Comprehension and Maintenance Activities

Sébastien Jeanmart
CS Department
Facultés Universitaires Notre-Dame de la Paix
Namur, Belgium
sjeanmar@student.fundp.ac.be

Yann-Gaël Guéhéneuc
Ptidej Team – GEODES – DIRO
Université de Montréal
Montréal, QC, Canada
guehene@iro.umontreal.ca

Houari Sahraoui
Verso Team – GEODES – DIRO
Université de Montréal
Montréal, QC, Canada
sahraouh@iro.umontreal.ca

Naji Habra
CS Department
Facultés Universitaires Notre-Dame de la Paix
Namur, Belgium
nha@info.fundp.ac.be

ABSTRACT

In the software engineering literature, many works claim that the use of design patterns improves the quality of programs, in particular, that it improves their comprehensibility and maintainability. Yet, little work attempted to study the impact of design patterns on the developers' activities of program comprehension and maintenance. We design and perform an experiment to collect data on the impact of design patterns on program comprehension and maintenance when developers use class diagrams. Specifically, we use eye-trackers to register saccades and fixations, the latter representing the focus of the developers' attention. Collected data show that design patterns play a role in maintenance tasks: class diagrams without design patterns require more attention from developers.

1. INTRODUCTION

Since the early work on design pattern [2, 11], the usefulness of design patterns for program development activities has been highlighted by many researchers and practitioners.

On the one hand, intuition suggests that design patterns are useful during program development activities. Much work is based on the premises that design patterns improve the understandability and the maintainability of programs, for example [1, 22]. Design patterns are the focus of many works studying their relevance, visualisation, identification, and so on, for example [1, 9, 13, 15, 16, 17, 25], with the hypothesis that their use improves quality. In education, design pattern are taught in software engineering and programming courses, see for example the conference series by ACM SIGCSE, and presented as best practices leading to their use in day-to-day development activities.

ACM SIGSOFT 2008 / FSE 16 Altanta, Georgia USA

On the other hand, some studies suggested that design patterns may impact negatively some aspects of quality because of the complexity that they introduce in the design and implementation of programs and because of the assumption that they are understood by developers. For example, Wendorff [28] assessed the impact of design patterns in some industrial projects and showed that their use impacted negatively comprehensibility. More recently, Bieman *et al.* [5] showed that classes playing certain roles in some design patterns are more change prone that others classes.

The goal of our work is to determine whether design patterns are useful during program comprehension and maintenance activities. In this paper, we propose:

1. To compare developers' performance in the presence and absence of a design pattern when performing comprehension and maintenance activities.

2. To compare developers' performance when a design pattern is shown as described in [11] and with a different layout when performing comprehension and maintenance activities.

We design experiments to collect data to compare the developers' performances when performing comprehension and maintenance tasks using different yet semantically-equivalent UML class diagrams. Informally, we define and measure performance as the amount of attention and time that developers must spend to perform given tasks: the less attention and the less time, the greater the performance. We choose UML class diagrams because UML is a *de-facto* standard and the main constructs of class diagrams are well understood by our subjects. The class diagrams in our experiments either include classes playing roles in a design pattern or not and either show the design pattern following its *canonical* representation (*i.e.*, as described in [11]) or not.

In this paper, we only use the Visitor design pattern, shown in Figure 1, because this pattern is widely used and is a good example of a pattern for which several design alternatives exist. Moreover, using more than one pattern would have made it difficult to isolate their respective impact on the subjects' performances. However, we will not be able generalise our results to any design pattern.

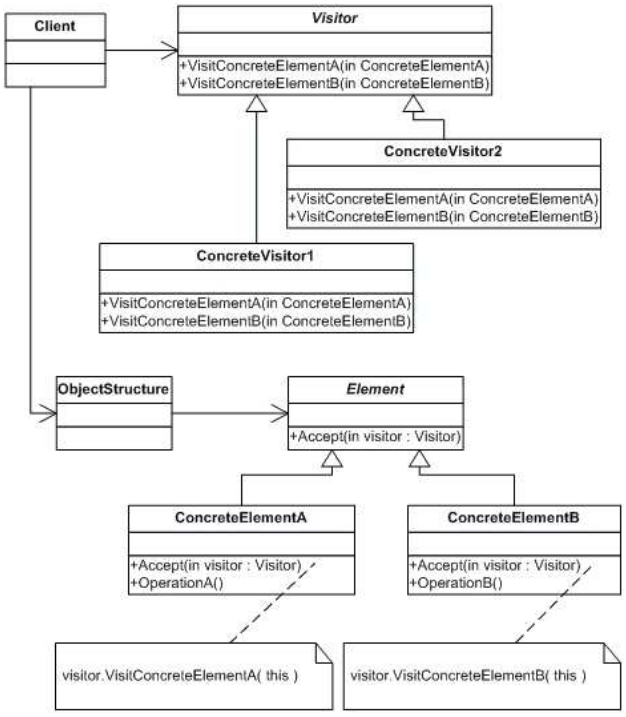


Figure 1: Visitor pattern

We measure developers' performance using data collected with eye-tracking systems, to decide whether a class diagram improves their performance with respect to the others. We collect data for three programs and 24 developers. We report that no significant difference exits between class diagrams with/without/with a modified Visitor for comprehension tasks. However, we show that for maintenance tasks, developers perform significantly better on diagrams where the Visitor is represented as shown in [11]. Additionally, we report that the levels of knowledge of UML and of design patterns do not modify significantly our findings.

The remainder of this paper is organised as follows. In the following Section 2, we present related work. Then, we define the experimental design in Section 3. We present in Section 4 the running of our experiments, their technical settings, and subject management. We analyse the collected data and present results in Section 5. We assess the validity of our experiments in Section 6 and conclude in Section 7.

2. RELATED WORK

This paper relates to two fields of study: program comprehension and maintenance and eye-tracking studies.

2.1 Program Comprehension and Maintenance

Program comprehension and maintenance are the subjects of much work in the software engineering community. This paper is specifically related to code and diagram reading, diagram layouts and visualisation, and mental models.

Code and diagram reading are important activities in software engineering because most of the information about a program is conveyed by text and diagrams. Spinellis [24] presents and discusses in his book techniques to read code more efficiently, *i.e.*, in less time, while maximising the gathered information. Recently, Uwano *et al.* studied the code reading habits of some developers and identified typical "patterns", such as the scan patterns that distinguishes "efficient" code readers from less efficient ones.

All developers use diagrams (or sketches thereof) to convey information to their peers or to better understand a program. Purchase performed several studies to provide (1) a better understanding on the use of diagrams by developers (such as entity–relationship diagrams [18]), (2) criteria to assess the aesthetics of diagrams (in particular node–link diagrams [27]), and (3) set of user preferences for graph layouts (specifically in the context of the UML notations [8]). Other authors, for example Seeman [21], proposed visualisation technique for UML-like class diagrams.

Finally, several authors proposed and evaluated models of program comprehension. One of the first model, proposed by Brooks [7], described program comprehension as a process of building a sequence of knowledge domains, bridging the gap between problem domain and program execution. Soloway [23] suggested that developers uses plans [20] when comprehending a program. Another model by von Mayrhauser [26] described the program comprehension process as a combination of top-down and bottom-up comprehension processes, working on a common knowledge base. This integrated model accounts for the dynamics of forming and of abstracting a mental representation of a program.

Previous work provides a sound basis on which to build experiments to study program comprehension. We follow Purchase's work on the importance of layouts and get inspiration from Uwano and others on the use of eye-trackers.

2.2 Eye Tracking Studies

Eye tracking is a means to analyse the activity of comprehension. The fundamental design of eye-tracking systems is based on the physiology of human visual capability and cognitive processes [10]. Therefore, the data collected using eye-trackers provides insight on the focus of attention of subjects, making it possible to infer underlying cognitive processes [19].

Eye-trackers have been used for many years, in particular in psychology, to understand the mental processes of subjects (animal and human alike). There is a large body of work on using eye-tracking and a dedicated conference (Eye Tracking Research & Applications Symposium). For example, Nevalainen [14] studied the short-term effects of textual and graphical visualisation on perception.

Several studies of program comprehension using eye tracker have appeared recently, possibly due to the more easy access to eye-trackers. For example, Bednarik *et al.* [4] proposed an approach how to investigate trends in repeated-measures sparse-data of few cases captured by an eye-tracker. Using this approach, they characterised the development of comprehension strategies during program visualisation. The same authors also studies program debugging [3].

Independently to Bednarik *et al.*'s study, one of the authors also conducted an experimentation to understand concretely how software engineers acquire and use information from UML-like class diagrams, using eye trackers [12]. He concluded on the importance of the classes and interfaces and reported, surprisingly, on the little use of binary class relationships by developers. A similar study has been carried out by Yusuf *et al.* [30] to identify the relative importance

of specific characteristics of UML-like class diagrams during program comprehension, such as layout, color, and stereotypes. They also studied how developers navigate in class diagrams. They concluded that layouts with additional information were the most effective and that the use of stereotypes plays an important role during comprehension.

We follow previous work in this study of the impact of layout (*i.e.*, with/without/with modified design pattern) and semantic data (*i.e.*, pattern) on program comprehension.

3. DESIGN

The design of our experiments is directed by our use of eye-trackers to collect relevant data to assess the developers' performance. We first present briefly our eye-tracking systems and then detail our hypotheses, variables, objects, and subjects.

3.1 Eye-trackers

Many eye-tracking systems exist on the market. We use the EyeLink II system from SR Research[1]. EyeLink II has the highest resolution (noise-limited at $< 0.01^{\circ}$) and fastest data rate (500 samples per second) of any head mounted eye tracker.

An eye-tracker is composed of two computers and a headband. The headband, shown in Figure 2, include two cameras and an infra-red emitter. The cameras use infra-red rays that are reflected on the subject's cornea to register eye-movements. Four sensors are placed on the subject's screen. These sensors work with the the infra-red emitter and allow computing the position of the head-band with respect to the screen. Along with the head-band cameras, these four sensors allow the system to compute precisely the position of the subject's gaze on the screen.

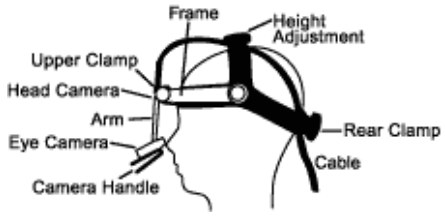


Figure 2: Headband of an Eye Tracker

The subject's and experimenter's computers are linked via an Ethernet connection, as shown in Figure 3. Their communication is based on the principle of "Action/Event" → "Reaction". When an event is emitted by the subject's computer, the experimenter's computer reacts and takes the control back. Typically, the subject's computer sends the position of the subject's gaze to the experimenter's computer, which in turn record this data on the disk, on the fly. Once the experimentation is finished, the experimenter's computer sends back the whole data file to subject's computer for easy access and analysis.

The data collected from the eye-tracking systems are of two types: raw positions and parsed positions. Parsed positions are expressed in terms of fixations and saccades based on physiological thresholds. Fixations are stabilisations of

[1]http://www.eyelinkinfo.com/

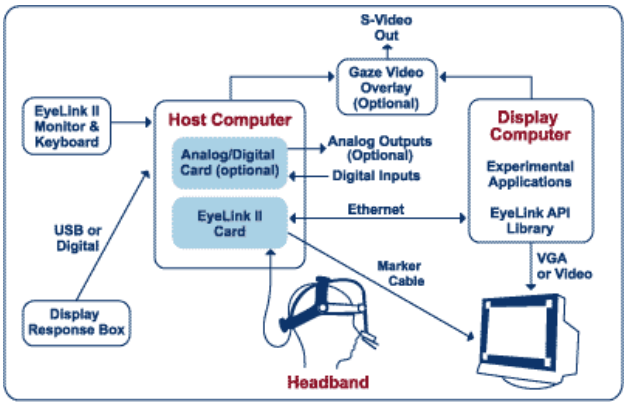


Figure 3: Eye-tracker Installation

the eye during a gaze while saccades are movements from one fixation to another. The numbers of fixations and saccades can vary from a dozen to several hundreds depending on the size and complexity of the class diagram.

These two pieces of data provide data for our analyses. As in previous work, for example [12, 30], we use fixations as the concrete manifestation of a subject focus of attention in an area of the screen. Therefore, we use the number of fixations and their duration as indicator of the attention that a subject spends on a diagram.

3.2 Hypotheses

We want to assess the following four null hypotheses, two for comprehension (HC) and two for maintenance (HM):

- HC_{0_1}: A class diagram with a design pattern does not help subjects during program comprehension when compared to a class diagram without design pattern.
- HC_{0_2}: A class diagram using the canonical representation of a design pattern does not help subjects during program comprehension when compared to a class diagram using another layout.
- HM_{0_1} A class diagram with a design pattern does not help subjects during program maintenance when compared to a class diagram without the design pattern.
- HM_{0_2} A class diagram using the canonical representation of a design pattern does not help subjects during program maintenance when compared to a class diagram using another layout.

If the previous null hypotheses are rejected, we could assume (with respect to the threats to the validity in Section 6) that the following alternative hypotheses are verified:

- HC_{a_1}: A class diagram with a design pattern helps subjects during program comprehension.
- HC_{a_2}: A class diagram using the canonical representation of a design pattern helps subjects during program comprehension.
- HM_{a_1} A class diagram with a design pattern helps subjects during program maintenance.

- HM_{a_2} A class diagram using the canonical representation of a design pattern helps subjects during program maintenance.

3.3 Independent Variables

From the previous hypotheses, we identify the following independent variables for our experiments:

- Design Alternative: CP, MP, and NP are the possible values for this variable because three class diagrams are built. Each class diagram conveys the same semantics. One diagram does not use the design pattern: NP for *no pattern*. One diagram uses the design pattern shown with its canonical representation: CP for *canonical pattern*. The last diagram uses the design pattern with a modified layout: MP for *modified pattern*.

- Tasks: Two different tasks are compared, one program comprehension task and one maintenance tasks: C or M.

The CP class diagram shows the design of a program using the classical layout of the pattern as shown in [11], for an example see Figure 4. The MP diagram displays the design and pattern with a modified layout, as illustrated in Figure 6. The NP diagram displays a design without any design pattern, as exemplified in Figure 5. No systematic layout modification has been performed. The aim here is to analyse how subjects react to familiar design patterns layout compared to design patterns with a non familiar layout.

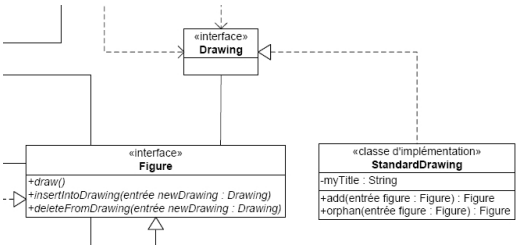


Figure 5: JHotDraw – No Pattern Design (NP)

In the rest of this paper, the different experiments performed by the subjects are named as:
<Design_alternative>_<Task>.

We retain two mitigating variables to put into perspective and better understand the results of our experiments:

- UML Knowledge: The subjects' knowledge of UML. The level is established using a questionnaire. An example of question is given at the end of Section 4.1. Values are taken from $[1, 2]$ where 2 means that a subject has a very good knowledge of UML class diagrams and 1 that the subject knows UML.

- DP Knowledge: The subjects' knowledge of design patterns. This knowledge is also established using a questionnaire following the same method as for the knowledge of UML.

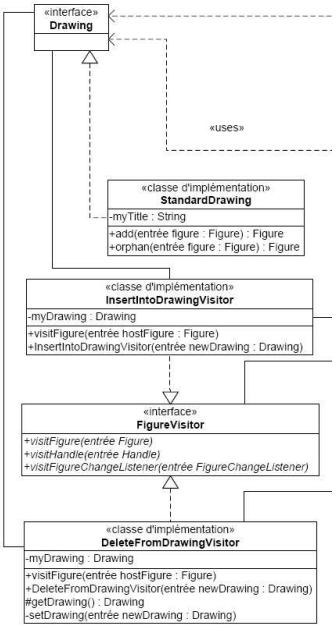


Figure 6: JHotDraw – Modified Design (MP)

3.4 Dependent Variables

The dependent variables are chosen based on the capabilities of the eye-trackers and according to our hypotheses and independent variables. We establish for each diagram (NP, CP, and MP) a list of *area of interest* (AOI). An area of interest is a *relevant* class in our diagrams that should be the focus of the subjects' attention to perform their comprehension or maintenance tasks (C or M).

- Average Number of Relevant Fixations (ANRF): We collect the number of relevant fixations performed by each subject for each question (*i.e.*, sum of all fixations performed in all relevant classes). Design alternatives (CP, MP, and NP) do not all have the same number of classes and therefore we normalise the number of relevant fixations by the numbers of relevant classes.

- Average Duration of Relevant Fixations (ADRF): This average is based on the ANRF and is computed as follows:

$$\frac{\sum_{i=1}^{\text{Number of Relevant Fixations}} (ET(F_i) - ST(F_i))}{\#\{relevant\ classes\}}$$

where $ST(F_i)$ and $ET(F_i)$ represent respectively start time and end time for a fixation F_i.

We choose these two dependent variables because they can be used to assess the developers' performance: when performing tasks on diagrams, the average number of fixations and their average duration correspond to the developers' efforts: less fixations and a shorter time means less efforts.

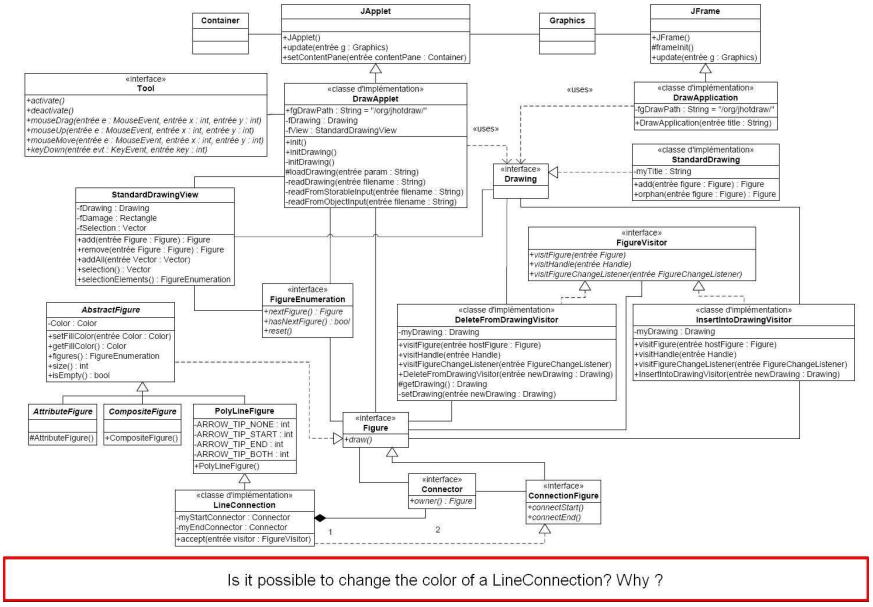


Figure 4: JHotDraw – Classical Design (CP)

3.5 Objects

We choose three open-source programs for our experiment: JHotDraw, JRefactory and PADL. JHotDraw is a framework to implement technical and structured drawings. This framework provides support for the creation of geometric and user defined shapes. JRefactory is code refactoring tool for Java programs. PADL is a meta-model for describing object-oriented programs, it is similar to the UML meta-model.

These programs all use the Visitor design pattern. Since full documentation was not available, partial reverse engineering has been done on each of these three programs to represent their design using UML class diagrams. For each of these three programs, three semantically-equivalent versions of their structure have been designed, see Design Alternative in Section 3.3.

3.6 Subjects

The experimentation was performed with 24 subjects: 7 post-graduate and 17 graduate students at the Department of Informatics and Operations Research at University of Montreal. All students have been involved during their courses in designing software architectures and in using UML class diagrams. These 24 subjects are placed into 3 balanced groups. The balancing simplifies and strengthen our statistical analysis of the collected data [29].

	CP	MP	NP
JHotDraw	G_1	G_2	G_3
JRefactory	G_3	G_1	G_2
PADL	G_2	G_3	G_1

Table 1: Questions distribution

3.7 Questions

It has been shown [4] that appropriate questions can trigger the subjects' mental processes when performing their tasks. Therefore, we ask specific questions to the subjects with each design alternatives. Table 2 shows examples of questions. Half the questions focus on the comprehension of the diagram displayed on the screen, the other half on a maintenance tasks. Questions were designed in such a way that the class diagrams are enough to answer the questions. Typically, questions involved two or three classes and do not take more than five minutes to answer for comprehension tasks, and more than seven minutes for maintenance tasks.

Comprehension
Is it possible to change the color of a LineConnection?
Explain how to instantiate a "Method" and list all involved classes.
Maintenance
We want to be able to realise a method refactoring. We would like to add two new functionalities: "addMethod", "moveMethod". Explain what you would change in the current diagram.
Which classe(s) and method(s) would you add to create a new type of relationship called "commentedRelationship".

Table 2: Kind of questions

4. EXPERIMENTS

We now present the running of our experiments, their technical settings, and subject management.

4.1 Running of the Experiments

The first step of the experiment is registration. We use a Web site to allow subject to register. The Web site provides information on the experiments: a description and a video showing the eye-tracking system.

The experiments are conducted in a quiet room without any disturbances. For each subject, an experiment takes about one hour, including a tutorial (15–20 minutes), data collection (20–30 minutes), and a questionnaire (5 minutes).

The tutorial reminds subjects of the different constructions typically found in UML class diagrams. We then simulate a real but smaller experiment to familiarise the subjects with the questions they are to answer. The feedback from subjects after the experiment and our observations show that this tutorial is helpful to prepare the subjects and to alleviate anxiety. At the end of the tutorial, we give a short technical explanation about the eye-tracking systems used to collect data.

Before performing comprehension and maintenance tasks, the eye-tracking system is calibrated. Calibration requires the subjects to fix one by one nine points on the screen. After calibration, data collection begins. During data collection, each subject perform six tasks by answering as many questions: three comprehension questions and three maintenance questions. The questions are randomised to prevent any learning bias while ensuring that each question is answered for each diagram by an equal number of subjects. Therefore, some subjects performs comprehension tasks with the *classical version* (CP) of JHotDraw while others do so with the *no-pattern* (NP) version of the same program. Configuration for each group is shown in Table 1. Before showing the diagram of a program, a text is displayed that resumes the purpose and main characteristics of the program. This text provides enough information to perform the tasks as confirmed by the subjects. No time limits are set but subjects are requested to answer as soon as they think that they know the answer.

Finally, we provide each subject with a short questionnaire to evaluate their knowledge of UML and of the design patterns. We provide this questionnaire after the experiment such as not to reveal the purpose of the experiment.

4.2 Technical Settings

Subject are seated during experiments in an ordinary office chair and ask to avoid any movement if possible. They face a 17"CRT screen and use an ordinary keyboard. To facilitate the conduct of the experiment and to avoid errors, all keys of the keyboard are disabled except for the escape key that allows subjects to control the experiment. Mouse handling is also disabled.

Each question is displayed on the screen at the bottom of a diagram. Once subjects have performed a task and given an answer to the question, they press the "ESC" key to go to the next diagram/question. We use a game pad to record whether a subject has correctly answered a question. However, assessing the correctness of an answer is subjective. Therefore, two experimenters assess the answers to increase objectivity. Each experimenter has a check-list for each question of each diagram for each program. Items of the check-lists are classes, methods, and attributes required to answer the questions. If all check-boxes are filled, the subject's answer is correct. If more than half the check-boxes are filled, the subject's answer is partially correct, else it is incorrect. We use the two assessments to record in a file a single evaluation for each answer of each subject. In case of conflict between the two experimenters, discussions was used to decide of the answer was correct or not.

5. ANALYSIS AND RESULTS

Table 3 summarises some figures on the collected data. We now discuss the results of testing our hypotheses on the collected data using the Student t-test, the ANRF and ADRF variables being normal. We also explore two factors that could mitigate these results, UML Knowledge and DP Knowledge.

Collected Data	
Subjects (#)	24
Data (Mb)	135
Videos (#)	144
Total Time Experiments ($hour$)	25
Total Time Eye-Tracking (min)	451
Time Relevant Eye-Tracking(min)	115
Total Fixations (#)	72.416
Relevant Fixations	25.319
Avg Time per question (C)(min)	2,71
Avg Time Rel. per question (C)(min)	0,73
Avg Fix. per question (C)(#)	447
Avg Rel. Fix. per question (C)(#)	162
Avg Time per question (M)(min)	3,54
Avg Time Rel. perquestion (M)(min)	0,87
Avg Fix. per question (M)(#)	558
Avg Rel. Fix. per question (M)(#)	190

Table 3: Collected data

5.1 Analysis of Comprehension Data

Table 4 summarises the effect of using the design pattern Visitor on the comprehension of UML class diagrams. The ANRF and ADRF are given respectively for the three design alternatives CP_C, MP_C, and NP_C. The p-value columns report on the statistical significance of the differences between respectively CP_C and NP_C (HC_{0_1}), and MP_C and NP_C (HC_{0_2}).

	CP_C	MP_C	NP_C	p-value (HC_{0_1})	p-value (HC_{0_2})
ANRF (#)	44.56	48.193	44.58	0.998	0.691
ADRF (ms)	12,354	12,878	11,992	0.887	0.69

Table 4: Effect of Visitor on comprehension

We observe almost the same number of fixations for CP_C and NP_C (44.5), and less than four fixations more for MP_C (48.2). The distributions presented in Figure 7 (left) show that the medians for ANRF are around 40 for all tasks. However, there is a large variance for ADRF between subjects working on CP_C, *i.e.*, wide box plot, compared to subjects

working on MP_C and NP_C. This variance could indicate that the level of knowledge in UML and–or design patterns of the subjects may play an important role when the Visitor is present in the diagrams. The same trend is found for the fixation duration ADRF, see Figure 7 (right).

In conclusion, the significance tests presented in Table 4 indicate that the presence of Visitor as well as its layout do not have a significant impact on the comprehension of UML class diagrams.

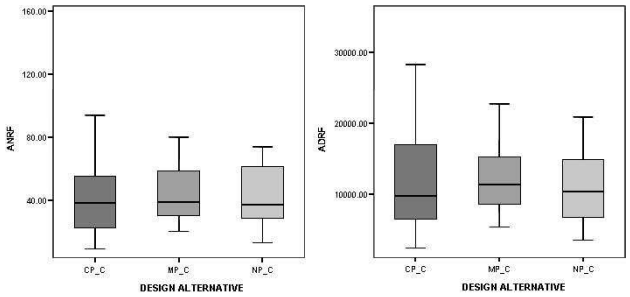


Figure 7: Comprehension data distribution

5.2 Analysis of Maintenance Data

Diagrams without Visitor are effort-consuming. The mean differences between tasks CP_M, MP_M, and NP_M are more important: roughly 20 to 33 more fixations and 5,000 to 7,000 *ms* more in fixation time for diagrams without the Visitor design pattern, see Table 5.

	CP_M	MP_M	NP_M	p-value (HM_{0_1})	p-value (HM_{0_2})
ANRF (#)	41.27	54.7	74.6	0.027	0.195
ADRF (*ms*)	11,543	14,003	19,040	0.026	0.126

Table 5: Effect of *Visitor* on maintenance

Figure 8 shows that subjects working on diagrams with the pattern Visitor and the classical layout exhibit clearly lower values for ANRF and ADRF. Moreover, they have an uniform performance, *i.e.*, flat box plots, compared to those of the two other groups.

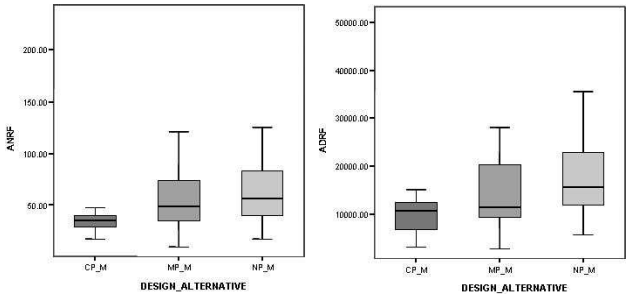


Figure 8: Maintenance Data Distribution

For the studied hypotheses, we can conclude that the design pattern Visitor, when present in diagrams with the classical layout, has a significant impact on the maintenance effort (p-values around 0.026 for both ANRF and ADRF). However, its impact is not significant when it is presented with the modified layout (p-values 0.195 and 0.126). We conjecture that when the layout is modified, the Visitor design pattern is difficult to recognise in the diagram, and then his effect is attenuated.

5.3 Impact of Secondary Factors

The results presented in Sections 5.1 and 5.2 show that there is an impact of Visitor on maintenance when it is displayed using its classical layout. In the case of comprehension, no significant impact is reported. Considering the variances between ANRF and ADRF for the program comprehension task reported in Section 5.1, we decided to investigate if the levels of knowledge in UML and design patterns mitigate the obtained results. As all our subjects were knowledgeable about UML and design patterns, we decided to consider two levels for each factor: 1 for good and 2 for very good.

Figure 9 (top) shows an interesting impact of UML knowledge on comprehension. Indeed, on diagrams with Visitor and the classical layout (CP_C), subjects that have very good knowledge in UML (level 2) perform better than those having good knowledge (level 1). The mean differences between the groups are more than 20 for the number of fixations and 6,000 ms for the fixation time. These differences are less important when the layout is modified (MP_C) or in the absence of the design pattern (NP_C). For the maintenance, both categories of subjects are consistent with the results obtained in Section 5.2 as can be seen in Figure 9 (bottom).

This finding cannot be confirmed by a 2-way ANOVA test. Table 6 gives the test significance values for the impact of UML knowledge and the combined impact (Design Alternative × UML Knowledge) for comprehension and maintenance tasks. All p-values are by greater than 0.05, which indicates that there is no significant difference in behavior between subjects with different levels of UML knowledge.

	ANRF (Comp.)	ADRF (Comp.)	ANRF (Maint.)	ADRF (Maint.)
UML Know.	0.374	0.437	0.676	0.456
Combined	0.413	0.271	0.933	0.739

Table 6: Impact of UML knowledge

The same analysis was conducted for the impact of the level of knowledge in design patterns. We notice an important difference in behavior on CP_C between groups with respectively a very good knowledge (level 2) and a good knowledge (level 1). Figure 10 (top) shows differences of roughly 25 of fixations and 6000 *ms* in time. We observe also that for maintenance tasks, in Figure 10 (bottom), there is a difference between groups of subjects (levels 1 and 2) on all the design alternatives. This difference is an indication that the level of knowledge of design patterns has an impact on maintenance tasks.

In the case of design pattern knowledge, 2-way ANOVA tests give p-values greater than 0.05, as shown in Table 7. However, p-values of 0.07 and 0.08 indicate that our observation on the impact on maintenance would need to be explored in a replication study.

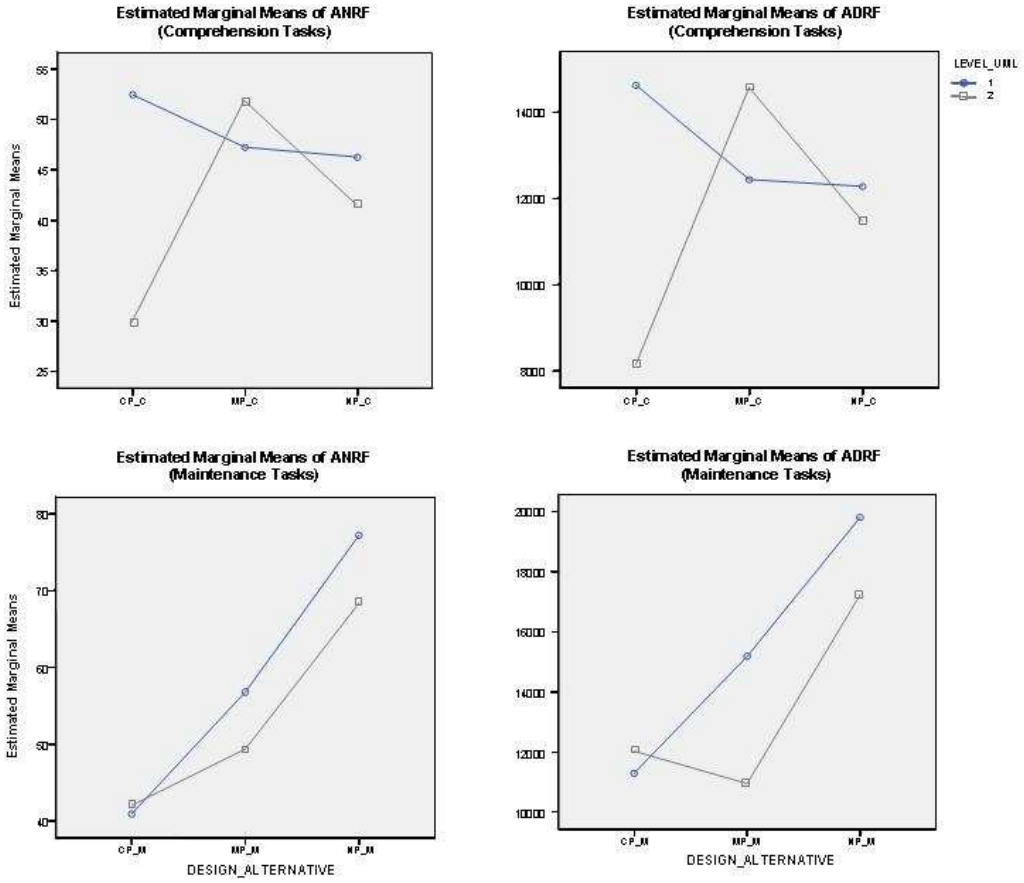


Figure 9: Combined impact of UML knowledge and presence of Visitor

	ANRF (Comp.)	ADRF (Comp.)	ANRF (Maint.)	ADRF (Maint.)
DP Know.	0.335	0.297	0.072	0.080
Combined	0.422	0.392	0.946	0.946

Table 7: Impact of design pattern knowledge

6. THREATS TO VALIDITY

Some threats to the validity of the reported results [29] are related to the use of human subjects in the study. Others are due to the use of the eye-tracking system.

6.1 Internal Validity

We identified three possible threats to internal validity: **maturation**, **instrumentation** and **diffusion of the treatments**.

In the case of maturation, we addressed the learning effect by presenting the tasks (questions to answer) in random and different orders to subjects. This avoids favoring a particular task because it is always given at the end. The random orders allows also to prevent the fatigue effect for which, tasks always given at the end can be disadvantaged. We reduced the fatigue effect also by limiting the subject effort performing all the tasks (20 to 30 minutes).

The instrumentation threat is related to the use of the eye-tracking equipment. Subjects had to wear a fairly heavy headband with the infrared camera. They had to minimise their head movement to avoid decalibration. To circumvent this threat, we analysed eye-movement movies recorded during the experiment of each subject. We only detected one decalibration case and the data of the concerned subject was eliminated from the sample. On the other hand, we have subject performing $6 \times (4,5) minutes$. Concentration made people often move slightly their head. As equipment is highly accurate, slight head movements have created coordinates offset and so temporarily erroneous data. Head movements are unavoidable. People tend to focus on only one concern at time: either on their head movements or on their assigned tasks. In first case, people will not be concentrated enough and answer will be biased and in the second case, coordinates offset will be performed due to head movements. Again, as we were interested in performing a rigorous experiment, we let subjects move their head. This allowed us to record relevant answers.

Our eye-tracker uses small infra-red cameras and human's corneal reflection that can lead to recording problems: in the case of "point cloud effect", no area are focused much more than the others. Fixations and saccades are also found in areas where there are neither classes nor relationships.

Difference between coordinates offset and point cloud effect is that first case can be fixed easily. This needs some extra work but does not represent a complex task. Simple

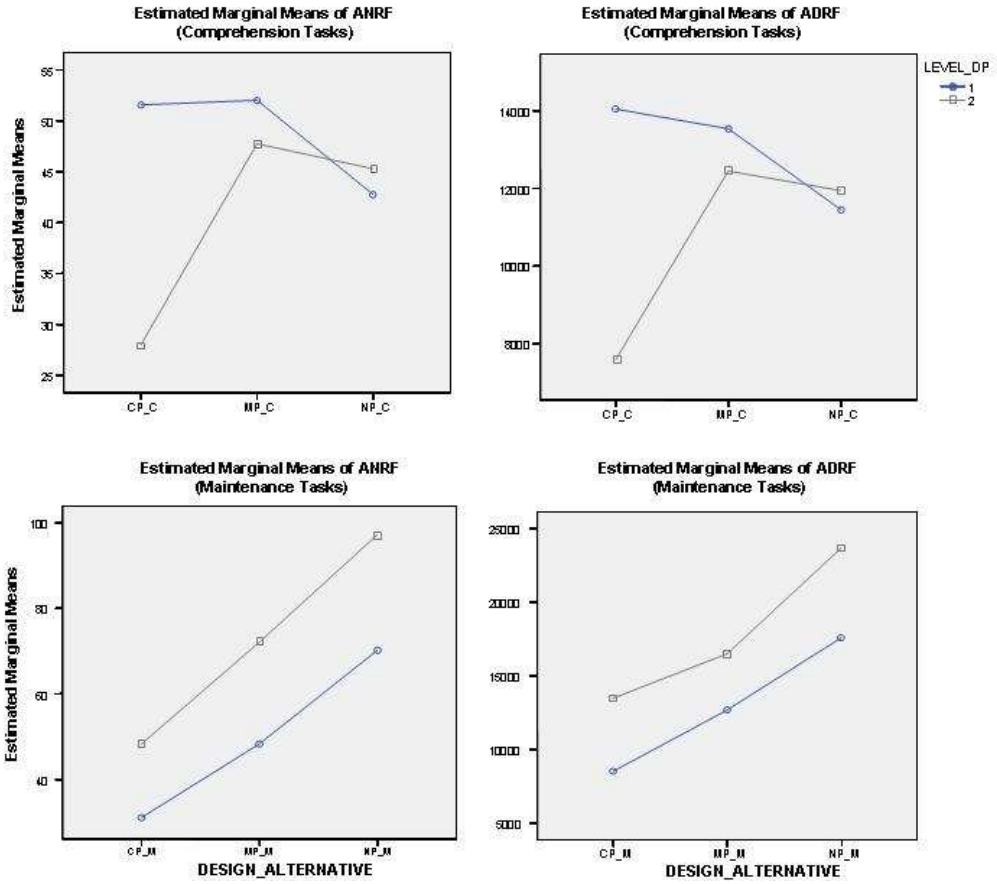


Figure 10: Combined impact of design pattern knowledge and presence of Visitor

drift correction has to be applied to data to create coordinates translation. In the second case, nothing can be done. This means that coordinates offset do not cost anything in terms of subjects while point cloud effect means subject rejection. Hopefully, only one subject has been rejected due to this problem. Coordinates offsets were much more frequent and this was fixed easily.

Finally, to prevent subjects from learning the treatments before hand, we gave instructions to each subject to not talk about the experiment before a fixed date corresponding to the end of all the experiments with subjects. We are confident that the instructions were followed by the subjects considering their perceived motivation.

6.2 Construct Validity

During the study, we specifically addressed four threats to construct validity : **mono-operation** and **mono-method** biases, **hypothesis guessing**, and **apprehension**.

Although the goal of the study is to study the impact of design patterns on comprehension and maintenance. Regarding the objects, we used the diagrams extracted from three programs with different application domains. To address the mono-method bias, we decided to use two dependent variables, *i.e.*, ANRF and ADRF and both had to be significative to reject the null hypotheses.

We did not inform the subject about the goal of the study before hand to anticipate the hypothesis guessing. They were not aware about presence of the patterns in the diagrams. We just explain to them in the tutorial that they have to perform tasks on different diagrams coming from different programs. None from the subjects had more than one version of the same diagram.

Different actions were performed to prevent the apprehension threat. First, the eye-tracking system was explained to the subjects and they were reassured about the absence of risks related to infrared camera on their eyes. Second, the subject were assigned identifiers to assure that no link will be made between the quality of their answers and their identity. Finally, we avoided to set a predefined time to perform the tasks. The subject were asked to perform the task diligently but without a time pressure.

6.3 External Validity

For this family of threats, we considered the interactions of **selection** and **setting** with the treatments.

The issue of whether student subjects are representative of software professionals has been discussed in several studies (see [6] for example). In our case, we used graduate and postgraduate student that were trained on UML and the majority of them has comparable knowledge on software modeling with UML and design patterns to most software professionals.

For the setting interaction, we considered the size and the complexity of the used diagrams. We reverse-engineered them from open-source programs that are extensively used. The diagrams presented to the subjects contains roughly 20 classes which is usual for comprehension and maintenance tasks. We cannot answer the question if the presence of design patterns in larger and more complex diagrams will have a higher or a lower impact. Specific replications with industry setting is need to draw such a conclusion.

6.4 Conclusion Validity

Threats to conclusion validity relates to random **irrelevancies in setting** and **heterogeneity of subjects**.

To prevent the first threat, we used a quiet laboratory with no disturbing factors. We performed the experiment several times with some subjects (not included in the study sample) to detect any non-considered factor that may influence the results. The use of the keyboard with the shift key, the presence of a second evaluator, and the subject aloud answers are examples of decisions that emerged from this calibration period.

Regarding the choice of subjects, our sample is heterogeneous enough in terms of level of knowledge to reflect the targeted population. However, to avoid that the results may be related mainly to subject knowledge, we used this later as a mitigating factor (see Section 5) and verified that its impact is less important than the presence of patterns.

7. CONCLUSIONS AND FUTURE WORKS

The goal of our work was to determine whether design patterns are useful during program comprehension and maintenance activities. We proposed:

1. To compare developers' performance in the presence and absence of a design pattern when performing comprehension and maintenance activities.
2. To compare developers' performance when a design pattern is shown as described in [11] and with a different layout when performing comprehension and maintenance activities.

We designed and performed experiments to collect data to compare the developers' performances when performing comprehension and maintenance tasks using different yet semantically-equivalent UML class diagrams that include the Visitor design pattern.

The analyses of the collected data shows that the presence of the Visitor design pattern do not improve the subjects' performance for comprehension tasks. However, cognitive factors related to the subjets' familiarity with patterns and UML (in general) have observable influences on comprehension tasks.

The analyses showed that the use of the Visitor design pattern shown with its *canonical* representation improves the subjects' performance for maintenance tasks. This improvement is dependent on the layout used to represent the patterns graphically and, therefore, probably on syntactical issues (*i.e.*, classes names, and so on).

In conclusion, the largely-admitted intuition that design patterns help developers during development activities seemed confirmed for the Visitor design pattern only for maintenance tasks. Therefore, other experiments are required to confirm our findings and conclude for other design patterns.

In future work, we will investigate design patterns as *communication* artifacts to study a larger set of variables, *i.e.*, variables related to the shape and syntax of graphic objects and to the knowledge of subjects. Other future work includes:

- Replication studies to confirm our observations and to address the different threats to the validity.
- Replication studies to confirm our observations using other variables than ADFR and ANFR.
- Replication studies to confirm our observations using design patterns.

8. REFERENCES

[1] L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso, and M. Penta. An Empirical Study on the Evolution of Design Patterns. In *6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on The Foundations of Software Engineering*, pages 385–394, 2007.

[2] K. Beck and W. Cunningham. Using pattern languages for object-oriented programs. In N. Kerth, editor, *Proceedings of the OOPSLA workshop on the Specification and Design for Object-Oriented Programming*, pages 9–16. ACM Press, September 1987.

[3] R. Bednarik and M. Tukiainen. Visual attention tracking during program debugging. In *The Third Nordic Conference on Human-Computer Interaction*, pages 331–334, 2004.

[4] R. Bednarik and M. Tukiainen. An eye-tracking methodology for characterizing program comprehension processes. In *2006 symposium on Eye tracking research & applications*, pages 125–132, 2006.

[5] J. Bieman, G. Straw, H. Wang, P. W. Munger, and R. T. Alexander. Design patterns and change proneness: An examination of five evolving systems. In M. Berry and W. Harrison, editors, *Proceedings of the* 9^{th} *international Software Metrics Symposium*, pages 40–49. IEEE Computer Society Press, September 2003.

[6] L. C. Briand, Y. Labiche, M. D. Penta, and H. Yan-Bondoc. An experimental investigation of formality in UML-based development. *Transaction on Software Engineering*, 31(10):833– –849, October 2005.

[7] R. Brooks. Using a behavioral theory of program comprehension in software engineering. In M. V. Wilkes, L. Belady, Y. H. Su, H. Hayman, and P. Enslow, editors, *Proceedings of the* 3^{rd} *International Conference on Software Engineering*, pages 196–201. IEEE Computer Society Press, May 1978.

[8] H. C.Purchase, J.-A. Allder, and D. Carrington. Graph layout aesthetics in UML diagrams: User preferences. *journal of Graph Algorithms and Applications*, 6(3):255–279, June 2002.

[9] J. Dong and Y. Zhao. Experiments on Design Pattern Discovery. In *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07)*, 2007.

[10] A. T. Duchowski. *Eye Tracking Methodology: Theory and Practice*. Springer-Verlag, 2003.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1^{st} edition, 1994.

[12] Y.-G. Guéhéneuc. TAUPE: Towards understanding program comprehension. In H. Erdogmus and E. Stroulia, editors, *Proceedings of the* 16^{th} *IBM Centers for Advanced Studies Conference*, pages 1–13. ACM Press, October 2006.

[13] J. K. H. Mak, C. S. T. Choy, and D. P. K. Lun. Precise Modeling of Design Patterns in UML. In *Proceedings of the 26th International Conference on Software Engineering*, pages 252–261, 2004.

[14] S. Nevalainen and J. Sajaniemi. Short-term effects of graphical versus textual visualization of variables on program perception. In *17th Annual Psychology of Programming Interest Group Workshop*, pages 77–91, 2005.

[15] T. H. Ng, S. C. Cheung, W. K. Chan, and Y. T. Yu. Work experience versus refactoring to design patterns: a controlled experiment. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'06/FSE-14)*, pages 12–22, 2006.

[16] L. Prechelt, B. Unger, M. Philippsen, and W. F. Tichy. Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. In *IEEE Transactions on Software Engineering*, volume 28(6), pages 595–606, June 2002.

[17] L. Prechelt, B. Unger, W. F. Tichy, P. Brössler, and L. G. Votta. A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions. In *IEEE Transactions on Software Engineering*, volume 27(12), pages 1134–1144, December 2001.

[18] H. C. Purchase, R. Welland, M. McGill, and L. Colpoys. Comprehension of diagram syntax: An empirical study of entity relationship notations. *International Journal of Human-Computer Studies*, 61(2):187–203, August 2004.

[19] K. Rayner. Eye Movements in Reading and Information Processing: 20 Years of Research. *Psychological Bulletin 124*, pages 372–422, 1998.

[20] C. Rich and R. C. Waters. *The Programmer's Apprentice*. ACM Press Frontier Series and Addison-Wesley, 1^{st} edition, January 1990.

[21] J. Seemann. Extending the Sugiyama algorithm for drawing UML class diagrams: Towards automatic layout of object-oriented software diagrams. In G. D. Battista, editor, *Proceedings of the* 5^{th} *international symposium on Graph Drawing*, pages 415–424. Springer-Verlag, September 1997.

[22] A. Shalloway and J. R. Trott. *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Addison-Wesley, 2000.

[23] E. Soloway, J. Pinto, S. Letovsky, D. Littman, and R. Lampert. Designing documentation to compensate for delocalized plans. *Communication of the ACM*, 31(11):1259–1267, November 1988.

[24] D. Spinellis. *Code Reading: The Open Source Perspective*. Addison Wesley, 1^{st} edition, May 2003.

[25] T. Trese and S. Tilley. Documenting software systems with views V: towards visual documentation of design patterns as an aid to program understanding. In *Proceedings of the 25th annual ACM International Conference on Design of Communication (SIGDOC'07)*, pages 103–112, 2007.

[26] A. von Mayrhauser. Program comprehension during software maintenance and evolution. *IEEE Computer*, 28(8):44–55, August 1995.

[27] C. Ware, H. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, June 2002.

[28] P. Wendorff. Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. In P. Sousa and J. Ebert, editors, *Proceedings of* 5^{th} *Conference on Software Maintenance and Reengineering*, pages 77–84. IEEE Computer Society Press, March 2001.

[29] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 1^{st} edition, December 1999.

[30] S. Yusuf, H. Kagdi, and J. I. Maletic. Assessing the comprehension of UML diagrams via eye tracking. In E. Stroulia and P. Tonella, editors, *Proceedings of the* 15^{th} *IEEE International Conference on Program Comprehension*, pages 145–154. IEEE Computer Society Press, June 2007.

B Taupe

B.1 Introduction

Cette annexe décrit le logiciel qui a été développé partiellement pour l'expérience présentée au Chapitre 5. Suite aux modifications apportées, ce logiciel a permis d'extraire un certain nombre d'informations pertinentes, utiles aux analyses de données.

Le coeur du logiciel TAUPE provient de travaux initiaux, réalisés par Yann-Gaël Guéhéneuc [Guéhéneuc 06]. TAUPE est un acronyme pour «Thoroughly Analysing the Understanding of Programs through Eyesight». Ce nom est en réalité un jeu de mot effectué par rapport aux rendus graphiques que la version initiale était capable de générer.

TAUPE peut être caractérisé comme un outil de visualisation de données. Ce logiciel a été développé initialement pour permettre de traiter et d'afficher les données brutes produites par un système particulier d'*eye-tracking*. Ce programme est développé entièrement en Java et utilise une API fournie par la compagnie du système d'*eye-tracking* utilisé[1].

Les fonctionnalités de la version de TAUPE présentée dans [Guéhéneuc 06] ne permettaient cependant pas de traiter l'ensemble des données produites lors d'une série d'expériences. Certaines modifications ont du être apportées afin de pouvoir gérer notamment l'affichage de résultats agrégés. Les fonctionnalités initiales ainsi que les fonctionnalités développées durant le stage sont détaillées dans le Section B.2 suivante.

B.2 Fonctionnement global

Le fonctionnement global de TAUPE est de générer à partir de fichiers bruts, issus d'un système d'eye-tracking, des informations intéressantes aux yeux de l'utilisateur du logiciel. La génération de ces informations intéressantes peut se matérialiser de deux manières. La **première manière** consiste à fournir un *aperçu graphique* des fixations et saccades d'un ou plusieurs utilisateurs pour un diagramme donné. La **deuxième manière** consiste à produire un *fichier de sortie* regroupant plusieurs valeurs de mesure calculées sur base d'informations de base (fixations, saccades, temps, etc.). Ces informations sont reprises dans un ou plusieurs fichiers de sortie du système d'eye-tracking utilisé. Suite aux modifications apportées, le fonctionnement global actuel de TAUPE peut être résumé à travers la Figure B.1 suivante.

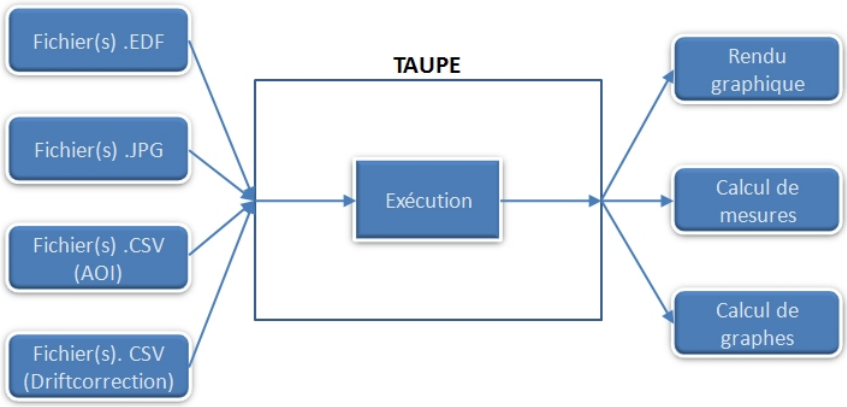


FIG. B.1 – Fonctionnement global de TAUPE

[1] http://www.eyelinkinfo.com

Inputs

Taupe requiert quatre types de fichiers en entrée afin de produire l'output désiré. Ces types de fichiers sont les suivants :

- Le ou les fichiers .EDF correspondant aux fichiers bruts résultant des expériences, voir description plus détaillée en Section B.4.1. Chacun de ses fichiers représente les données collectées durant l'expérience d'un seul sujet. Si l'output désiré doit s'effectuer sur plusieurs sujets, il convient de fournir en input les fichiers de chacun de ces sujets.

- Le ou les fichiers .JPG[2] correspondant aux images/diagrammes affichés durant les expériences. Si plusieurs «*trials*» (cfr. Section B.4.1) ont été réalisées durant une expérience, il convient de fournir chacune des images affichée durant chaque «*trial*».

- Le ou les fichiers .CSV[3] comprenant les coordonnées des zones d'intérêt (AOI, voir explication à la Section B.4.2) de chaque image/diagramme. En fonction du nombre d'images affichées durant une expérience, il convient de fournir un nombre identique de fichier .CSV reprenant plusieurs informations sur les zones d'intérêt de chacune de ces images. Plus d'informations sur le contenu de ce type de fichiers .CSV sont fournies à travers la Section B.4.3.

- Le ou les fichiers .CSV[4] comprenant les translations de coordonnées à réaliser pour chaque fichier .EDF. En fonction du nombre de «*trials*» réalisées durant une expérience, il est possible de définir les translations de coordonnées à réaliser afin de corriger les erreurs de coordonnées enregistrées. Ces erreurs apparaissent suite aux mouvements de la tête d'un sujet. Plus d'informations sur le contenu de ce type de fichiers .CSV sont fournies à travers la Section B.4.4.

Outputs

La version actuelle de Taupe permet, sur base des modifications effectuées, d'aller plus loin dans les analyses de données. L'ancienne version de ce logiciel ne permettait, à l'époque, que de générer graphiquement le rendu des résultats d'une seule personne pour une seule question. Après apports personnels, Taupe permet désormais de :

- Produire un fichier .CSV regroupant les différentes valeurs de mesures calculées pour chaque fichier .EDF, fichier .EDF représentant les données enregistrées durant l'expérience de chaque sujet. Ce fichier .CSV permet d'être intégrer dans des tableurs de type «*Excel*» ou dans des logiciels d'analyses de données tels que «*SPSS*». Ce dernier type de logiciel permet d'effectuer, entre autres, des tests statistiques afin d'évaluer la pertinence des données (cfr. Section 5.5).

- Afficher les fixations et les saccades d'une question de manière agrégée (ou non) sur base des résultats de plusieurs (ou d'une seule) expérience(s). Cette fonctionnalité permet à l'utilisateur d'observer la tendance générale (s'il y a utilisation de plusieurs fichiers) ou de la tendance particulière des zones fixées durant une ou plusieurs expérience. Ce type de fonctionnalité ne permet pas d'effectuer de conclusions statistiques

[2] Les noms des fichiers .JPG doivent être identiques aux noms des images utilisées lors des expériences
[3] Les noms des fichiers .CSV doivent être identiques aux noms des images auxquelles ils se rapportent
[4] Les noms des fichiers .CSV doivent être identiques aux noms des fichiers .EDF auxquels ils se rapportent

mais représente une étape préliminaire à la production du fichier .CSV présenté au point précédent.

- Produire des documents .DOT (fichier *dotty*) décrivant des graphes matérialisant les connexions entre les différentes zones d'intérêt. Chaque document .DOT décrit le graphe de chaque «*trial*». Un document .DOT représente un graphe comprenant un ou plusieurs noeuds ainsi qu'un ou plusieurs arcs. Les zones d'intérêts définies préalablement (AOI, cfr. Section B.4.2) correspondent aux «*noeuds*» tandis que les «*arcs*» matérialisent les interconnexions entre les zones d'intérêt. A chacun de ces arcs, un poids nul de départ a été attribué. Pour chaque passage d'une fixation d'une zone d'intérêt à une autre, le poids de l'arc reliant les deux noeuds matérialisant ces deux zones d'intérêt a été augmenté de 1. Le but ultime de cette démarche est de pouvoir identifier, sur base des tendances générales résultant des comportements des sujets, les facteurs qui facilitent la compréhension et la maintenance de programmes. Cependant, par manque de temps et d'outils performants, cette démarche n'a pu être poussée jusqu'au bout, seuls les calculs des graphes peuvent être obtenus via l'exécution du programme.

REMARQUE Il faut cependant préciser qu'un seul output ne peut être produit par exécution du programme.

B.3 Architecture

TAUPE est un petit logiciel comportant quatre packages dans lesquels se répartissent 22 classes. La structure des packages est représentée par la Figure B.2.

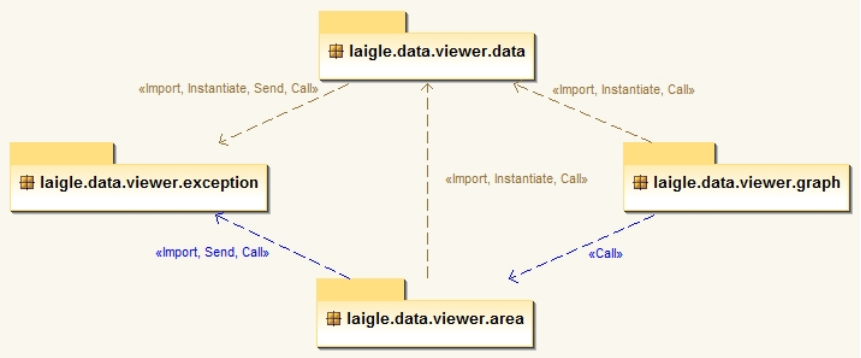


FIG. B.2 – Structuration des packages de TAUPE

Ces différents packages sont décrits aux travers des Sections B.3.1–B.3.4.

B.3.1 Package laigle.data.viewer.area

Le package **laigle.data.viewer.area** contient les classes principales gérant les différentes fonctionnalités du système. Ce package est représenté par la Figure B.3.

La classe «**ExperimentPreProcessor**» a pour objectif de charger en mémoire tous les fichiers .EDF afin de les traiter ensuite. Ce traitement consiste à créer les objets appropriés, cfr. classes présentes dans le package «laigle.data.viewer.data» à la section suivante.

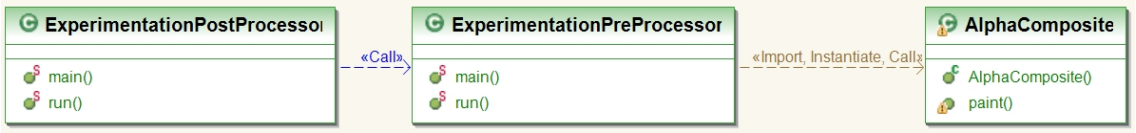


FIG. B.3 – Classes du package «**laigle.data.viewer.area**»

Ces objets peuvent ensuite être traités par les classes «**ExperimentPostProcessor**» ou «**AlphaComposite**».

La classe «**ExperimentPostProcessor**» permet, sur base des données résultant des fichiers .EDF préalablement chargés en mémoire, de produire un fichier .CSV en sortie. Ce fichier .CSV reprend les différentes valeurs des mesures calculées en fonction des données en mémoire

La classe «**AlphaComposite**» permet, également sur base des données chargées préalablement en mémoire, de visualiser graphiquement les fixations et saccades réalisées par une ou plusieurs personnes (en fonction du nombre de fichiers .EDF fournis) pour une question donnée. Si plusieurs fichiers .EDF sont fournis en entrée, le visualisation des données d'un diagramme représente l'agrégation de toutes les fixations et saccades réalisées par tous les sujets ayant travaillé sur ce diagramme.

B.3.2 Package laigle.data.viewer.data

Le package **laigle.data.viewer.data** contient les classes relatives aux structures de données du programme.

B.3.3 Package laigle.data.viewer.exception

Le package **laigle.data.viewer.exception** contient les exceptions relatives à certaines classes de données du package **laigle.data.viewer.data** ainsi qu'à la classe «**Edge**», présente dans le package **laigle.data.viewer.graph** et représentant le concept de «*noeud*» dans le graphe construit par le logiciel. Ce package est représenté par la Figure B.5.

B.3.4 Package laigle.data.viewer.graph

Le package **laigle.data.viewer.graph** contient les classes relatives à la création d'un graphe sur base des classes de données présentes dans le package **laigle.data.viewer.data**.

La classe «**Vertex**» matérialise le concept de «*noeud*». Un noeud correspond à une AOI (cfr. Section B.4.2). Pour plus de réutilisabilité, un «*Vertex*» est composé d'un *java Object*, représentant le concept jouant le rôle de noeud. Dans ce cas-ci, un «*Vertex*» correspond a une AOI. Pour chaque «*Vertex*», deux «Edge» sont associés : un «*Edge*» incident et un «*Edge*» sortant.

La classe «**Edge**» représente le concept de «*noeud*». Un «*Edge*» est composé d'un «*Vertex*» source et d'un «*Vertex*» destination. A chaque «*Edge*» est également associé un poids positif ou nul. Pour chaque passage d'une fixation se trouvant dans le «*Vertex*» source vers une fixation se trouvant dans le «*Vertex*» destination, le poids de l'arc concerné est augmenté de 1.

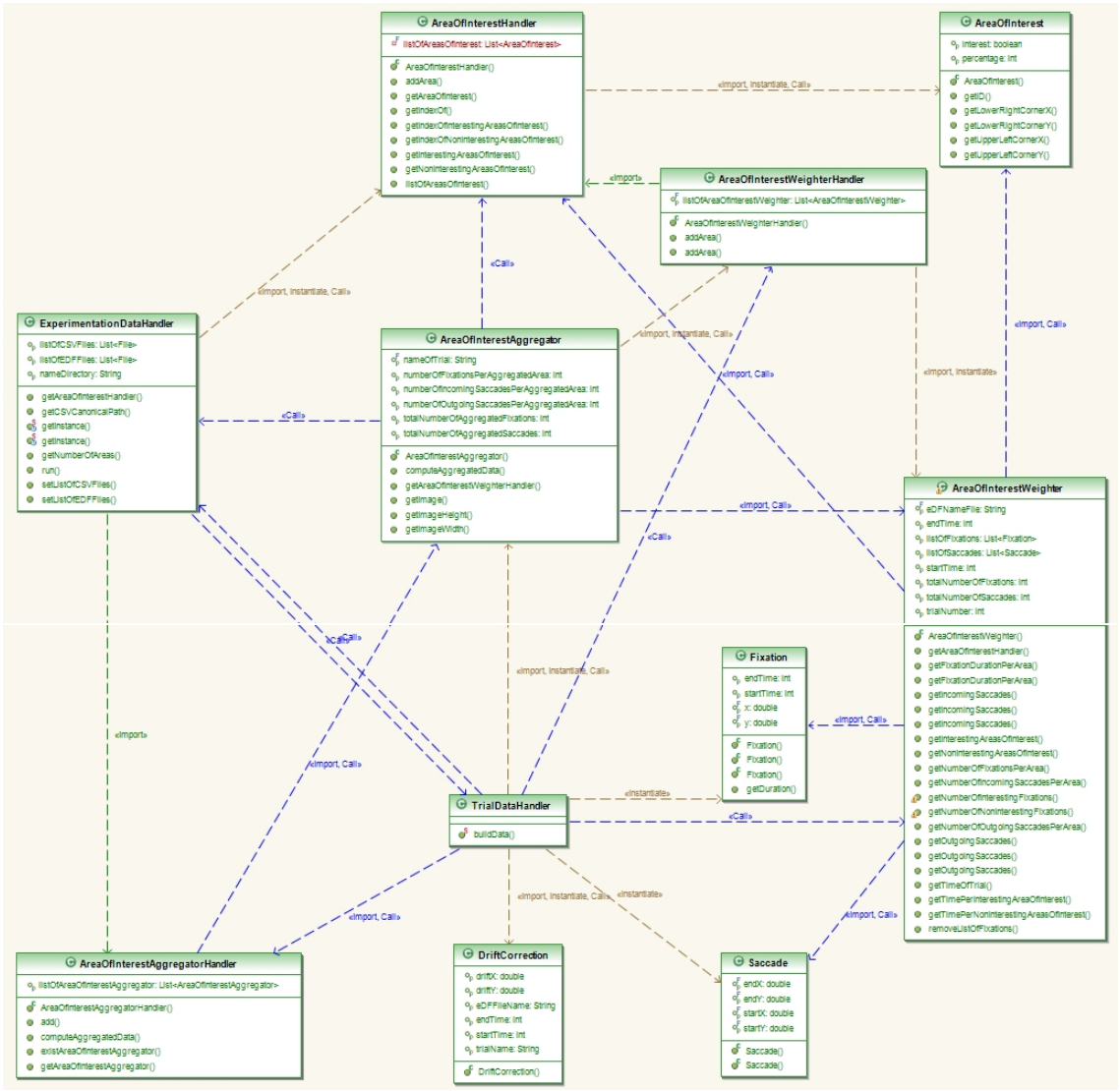


FIG. B.4 – Classes du package «**laigle.data.viewer.data**»

La classe «**GraphProcessor**» permet, sur base des informations enregistrées en mémoire (cfr Classe «*ExperimentPreProcessor*» en Page 136), de constituer, dans un premier temps, le graphe correspondant. Dans le cas de cette expérience, cette classe permet de construire un graphe correspondant à un diagramme UML, pour un ou plusieurs fichiers .EDF. Dans un second temps, cette méthode extrait, sur base du graphe en mémoire, les informations pertinentes afin de les enregistrer dans un fichier .DOT, conformément à la syntaxe de *Dotty*. Ce fichier peut ensuite être lu par le programme associé à *Dotty*, générant un rendu graphique de ce graphe.

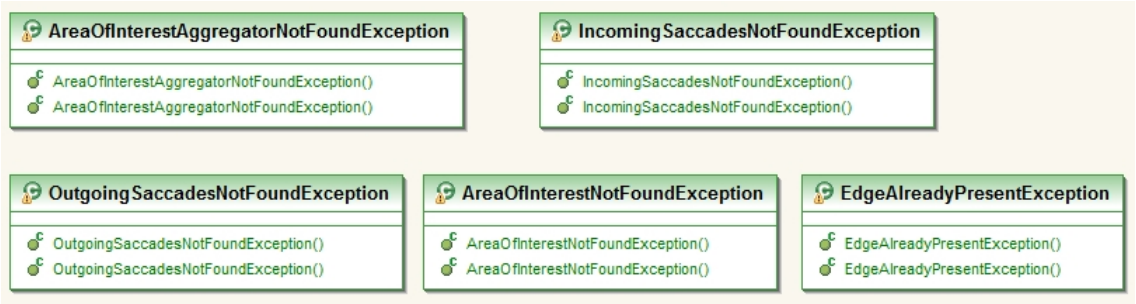


FIG. B.5 – Classes du package «**laigle.data.viewer.exception**»

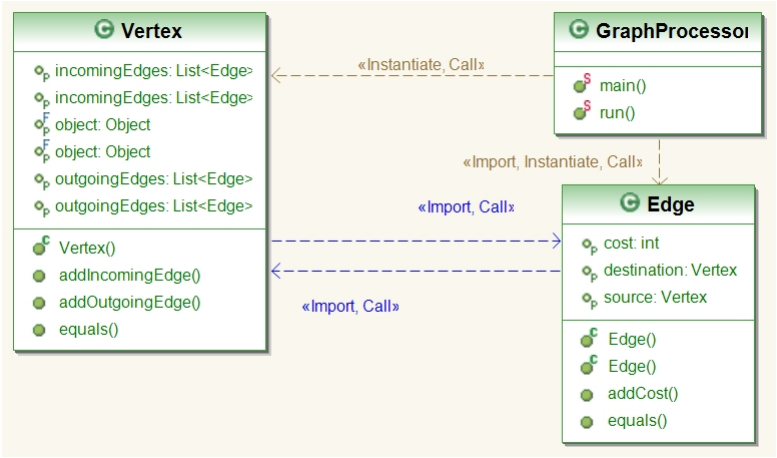


FIG. B.6 – Classes du package «**laigle.data.viewer.graph**»

B.4 TAUPE et Eye-Tracking

B.4.1 Fichiers EDF

Le système d'*eye-tracking* considéré dans cette expérience, et présenté à travers la Section 5.3.6, possède son propre type de fichier de sortie. Ces fichiers possèdent une extension de type .EDF. Le contenu de ces fichiers ne peut être lisible directement. Il est nécessaire de convertir ces fichiers EDF en fichiers utilisant des caractères ASCII. Ces fichiers se présentent tel qu'il est présenté au Tableau B.1.

A partir de ce fichier ASCII, il est possible de retracer le déroulement de l'expérience du sujet auquel le fichier se rattache.

Une expérience, aussi bien au niveau conceptuel que telle qu'elle est représentée dans les fichier ASCII, se compose d'un ou plusieurs «*trial*». Un «*trial*» correspond au concept de «*tâche*» abordé dans ce mémoire. A chaque «*trial*» est associé un temps de début et de fin, représentés en millisecondes. Une image est également associée à chaque «*trial*». Cette image possède un nom. Tel que le Tableau B.1 l'a présenté, un «*trial*» possède une suite ordonnée de fixations et de saccades, en fonction du moment du moment où elles ont été effectuées. Sur base de ces trois informations (temps, fixation et saccade), il est possible de créer des mesures telles que présentées au Chapitre 5, Section 5.3.3.

Header	
Informations générales, incluant le nom du diagramme de classes	MSG 82177 TRIALID PIX1 Normal MSG 82178 !V TRIAL_VAR_DATA Normal MSG 82210 !V IMGLOAD FILL images/UMLDiagram1.jpg
Calibration	MSG 208698 !CAL Calibration points : MSG 208698 !CAL 176.2, 74.5 -808, 3467 ... MSG 230854 !CAL VALIDATION HV9 R RIGHT GOOD ERROR 0.55 avg. 0.97 max OFFSET 0.45 deg. -13.2,5.2 pix.
Correction de position	MSG 253038 DRIFTCORRECT R RIGHT at 640,512 OFFSET 0.83 deg. -20.9,14.5 pix.
Informations sur les données enregistrées	MSG 253043 !MODE RECORD P 500 2 1 START 253044 RIGHT SAMPLES EVENTS PRESCALER 1 VPRESCALER 1 PUPIL AREA EVENTS GAZE RIGHT RATE 500.00 TRACKING P FILTER 2 SAMPLES GAZE RIGHT RATE 500.00 TRACKING P FILTER 2
Body	
Positions des yeux	253048 641.7 512.0 499.0 . ...
Début et fin d'une fixation avec des données sur la position des yeux	SFIX R 253052 ... 253310 643.7 508.0 496.0 . ... EFIX R 253052 253310 260 641.5 509.1 504
Début et fin d'une saccade avec des données sur la position des yeux	SSACC R 253312 ... 253328 691.9 435.4 522.0 . ... ESACC R 253312 253346 36 644.0 505.0 706.3 394.3 4.43 260
Début et fin d'une fixation et début d'une saccade avec des données sur la position des yeux	SFIX R 253348 ... EFIX R 253348 253610 264 714.8 404.2 452 SSACC R 253612 ...
End	
Fin de l'enregistrement	END 267755 SAMPLES EVENTS RES 32.69 29.12

TAB. B.1 – Détails de données collectées (adapté de [Guéhéneuc 06])

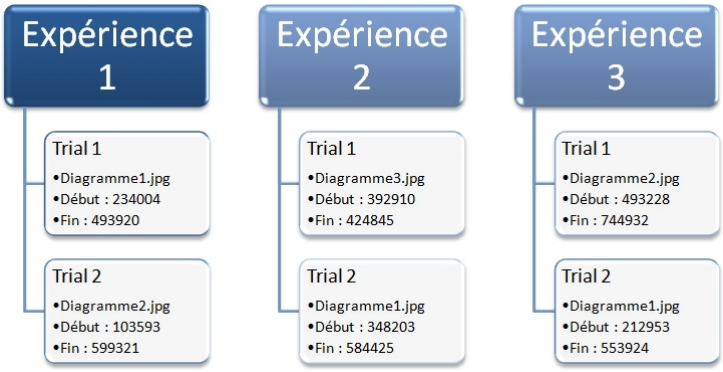


FIG. B.7 – Figure : Modélisation d'une expérience selon Eyelink II

La Figure B.7 modélise graphiquement une expérience telle que ce système d'*eye-tracking* le considère. Avec cette représentation graphique d'une expérience, il est à présent plus facile de comprendre comme TAUPE agrège les données de plusieurs sujets/expériences. Pour un nom d'image donné, TAUPE recherche, parmi un ensemble de fichier EDF, les fichiers qui contiennent un «*trial*» dont l'image utilisée correspond à celle recherchée. Pour chaque «*trial*» vérifiant cette contrainte, TAUPE extrait les fixations et saccades réalisées par le sujet correspond. Conformément aux précisions données sur les fonctionnalités de TAUPE en Section B.2, l'utilisateur pourra décider de, soit visualiser ces données, soit les enregistrer dans un fichier de sortie.

B.4.2 Zone d'intérêt

AOI est l'acronyme de «*Area of Interest*». Tel que son nom l'indique, cette «*zone d'intérêt*» représente une partie d'un diagramme, ou d'une image, pour laquelle une personne accorde de l'importance. Ce concept est souvent utilisé, notamment dans le domaine de la compréhension de programmes. Il permet de désigner les zones importantes/pertinentes d'une interface dans lesquelles le sujet devrait trouver l'information qu'il désire obtenir [Goldberg 99, Johansen 06, Bednarik 06].

Dans le cas de l'expérience présentée au Chapitre 5, le concept d'«*AOI*» a également été utilisé afin de déterminer les classes qui détenaient l'information nécessaire pour répondre à une question posée. Ces questions sont présentées en Annexe H et les informations pertinentes sont présentées dans les *check-lists* en Annexe G.

Une zone d'intérêt, pertinente ou non, correspond à un 4-uple (X_1, Y_1, X_2, Y_2) où (X_1, Y_1) représentent les coordonnées du coin supérieur gauche et (X_2, Y_2) représentent les coordonnées du coin inférieur droit de la zone d'intérêt. Etant donné que toutes les fixations et saccades définies par le système d'*eye-tracking* possèdent des coordonnées en (X, Y), une méthode a pu être conçue afin de déterminer à quelle zone d'intérêt ces fixations et saccades appartenaient.

B.4.3 Fichiers CSV pour les AOI

Etant donné que les AOI ne sont pas les mêmes d'un diagramme à l'autre, une technique a du être trouvée afin de charger automatiquement dans TAUPE les coordonnées des AOI pour chacun des diagrammes. La solution retenue est un fichier CSV (Comma-Separated-

Value). Cette extension de fichier permet d'être traitée dans n'importe quel tableur et ne requiert aucune librairie particulière. La syntaxe considérée dans TAUPE pour la définition des AOI d'une image/d'un diagramme est la suivante :

$$<\text{Nom de la zone d'intérêt}>,< X_1 >,< Y_1 >,< X_2 >,< Y_2 >,<\text{Y|N}>;$$

Le fichier CSV contient autant de lignes qu'il n'y a de zones d'intérêt. La définition de chaque AOI doit se terminer par une « ;» et un retour à la ligne. Le nom du fichier doit être le même que le nom du diagramme pour lequel les AOI sont définies. Par exemple, si l'image s'appelle «Bateau.jpg», le fichier CSV correspondant devra s'intituler «Bateau.csv». La dernière valeurs de chaque ligne permet de déterminer si la zone d'intérêt est pertinente ou non (**Y**es|**N**o).

B.4.4 Fichiers CSV pour les corrections de coordonnées

Conformément aux problèmes rencontrés avec le système d'*eye-tracking* lors de la collecte des données, une solution a été trouvée pour le problème le plus mineur. Cette solution consiste à définir dans un fichier les corrections/translations de coordonnées à effectuer à partir d'un temps t pour toutes les fixations et saccades apparues avec ce temps t. La syntaxe de ce fichier est la suivante :

<Nom image>,<Temps début>,<Temps fin>,<Translation en X>,<Translation en Y> ;

Différentes corrections peuvent être effectuées pour un même «*trial*». Dans ce cas, il suffit d'inscrire une nouvelle ligne avec un nom de «*trial*» identique et des temps et des coordonnées différents. A nouveau, chaque ligne devra se terminer par un « ;» et un retour à la ligne.

Remarque :

Si plusieurs translations doivent être effectuées sur un même «*trial*», elles doivent être inscrites successivement dans le fichier par ordre croissant de temps de départ. Pour rappel, les temps sont exprimés en millisecondes.

Le nom des fichiers CSV de corrections de coordonnées doit être le même que le fichier EDF auquel les corrections s'appliquent.

C Affiche de recrutement

Eye-Tracking Experimentation

- du 05/11 au 23/11 -

Envie de participer à une expérience originale ? Vous êtes le ou la bienvenu(e) !

Le but de cette expérience est d'évaluer l'impact de différents choix d'architecture de bas niveau (micro-architecture) au niveau de la compréhension, et de la maintenance sur des diagrammes de classes UML sémantiquement équivalents au moyen d'un système d'*Eye-Tracking*.

Comment faire pour participer à l'expérience ?

- **1ère étape : L'inscription sur le site**

  url : http://www-etud.iro.umontreal.ca/~jeanmars/

  Choisissez un jour et une heure sur le site web, de plus amples informations sur l'expérience apparaîtront ensuite.

- **2ème étape : L'expérience**

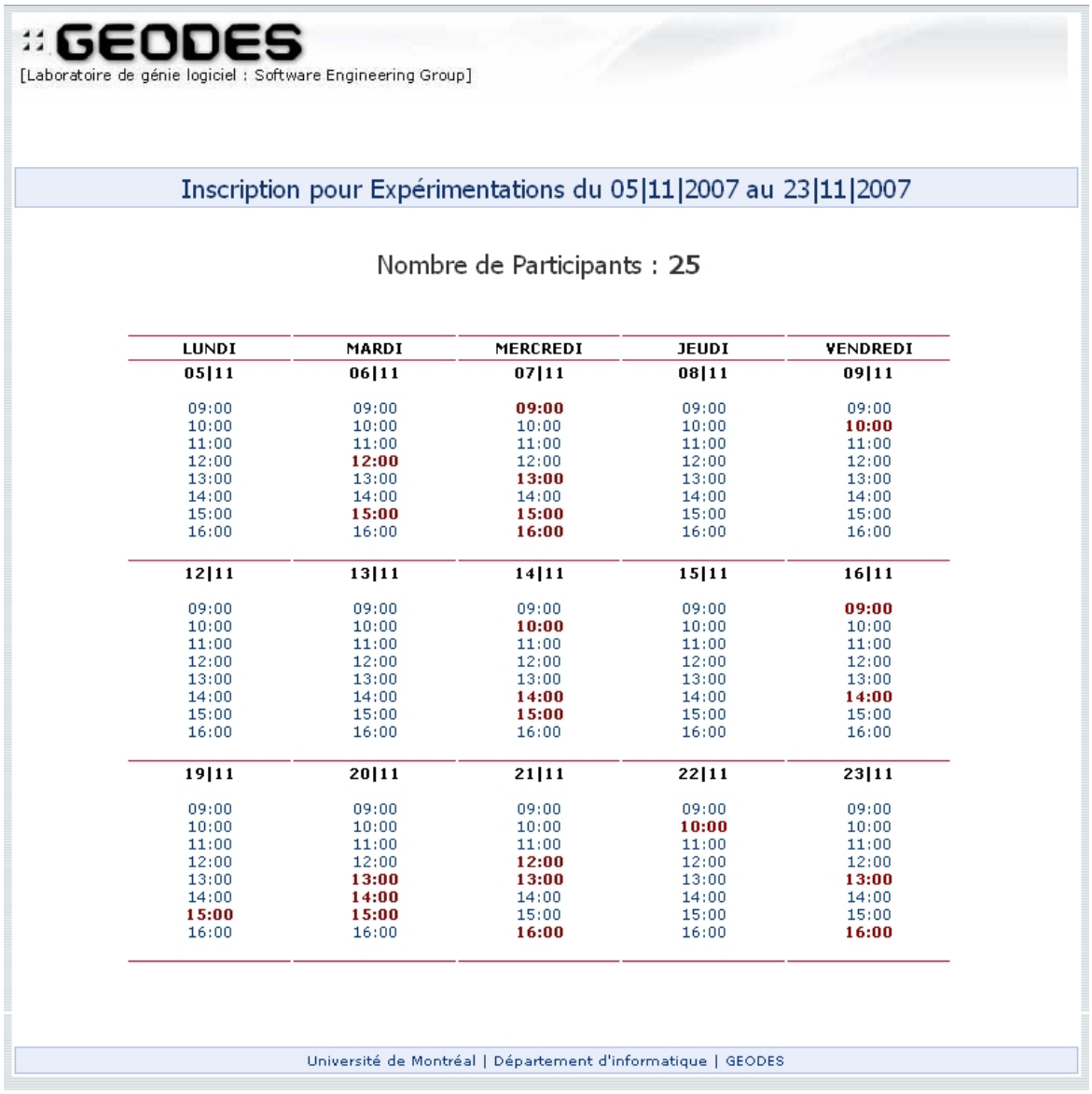
  Où : au local 2352, Pavillon André-Aisenstadt, 2920, Chemin de la Tour.
  Quand : à l'heure que vous aurez choisi, pas besoin de venir avant
  Durée approximative de l'expérience : 1 heure (au maximum) dont 5-10 minutes de présentation du système et de l'expérience, et 2 minutes pour répondre à un questionnaire.

ATTENTION !
Pré-requis pour l'expérience : connaissance au niveau des diagrammes de classes UML

Université de Montréal

Fig. C.1 – Affiche de recrutement

D Site web de recrutement



GEODES

[Laboratoire de génie logiciel : Software Engineering Group]

Inscription pour Expérimentations du 05|11|2007 au 23|11|2007

Nombre de Participants : **25**

LUNDI	MARDI	MERCREDI	JEUDI	VENDREDI
05|11	**06|11**	**07|11**	**08|11**	**09|11**
09:00	09:00	**09:00**	09:00	09:00
10:00	10:00	10:00	10:00	**10:00**
11:00	11:00	11:00	11:00	11:00
12:00	**12:00**	12:00	12:00	12:00
13:00	13:00	**13:00**	13:00	13:00
14:00	14:00	14:00	14:00	14:00
15:00	**15:00**	**15:00**	15:00	15:00
16:00	16:00	**16:00**	16:00	16:00
12|11	**13|11**	**14|11**	**15|11**	**16|11**
09:00	09:00	09:00	09:00	**09:00**
10:00	10:00	**10:00**	10:00	10:00
11:00	11:00	11:00	11:00	11:00
12:00	12:00	12:00	12:00	12:00
13:00	13:00	13:00	13:00	13:00
14:00	14:00	**14:00**	14:00	**14:00**
15:00	15:00	**15:00**	15:00	15:00
16:00	16:00	16:00	16:00	16:00
19|11	**20|11**	**21|11**	**22|11**	**23|11**
09:00	09:00	09:00	09:00	09:00
10:00	10:00	10:00	**10:00**	10:00
11:00	11:00	11:00	11:00	11:00
12:00	12:00	**12:00**	12:00	12:00
13:00	**13:00**	**13:00**	13:00	**13:00**
14:00	**14:00**	14:00	14:00	14:00
15:00	**15:00**	15:00	15:00	15:00
16:00	16:00	**16:00**	16:00	**16:00**

Université de Montréal | Département d'informatique | GEODES

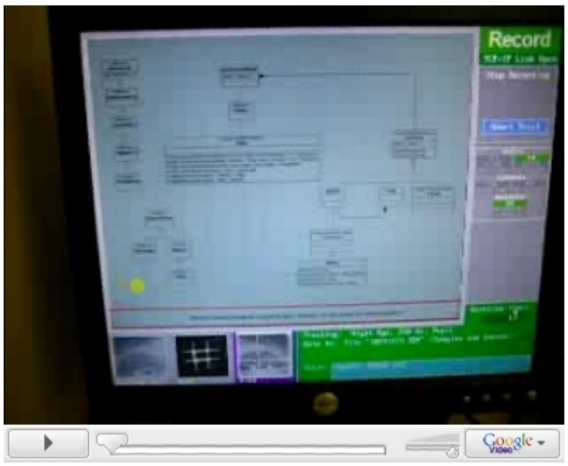
FIG. D.1 – Site web de recrutement – Page d'accueil

GEODES
[Laboratoire de génie logiciel : Software Engineering Group]

Inscription pour Expérimentations du 05|11|2007 au 23|11|2007

Description de l'Expérimentation

L'expérimentation se déroulera lors d'une séance d'une heure (au maximum). Nous vous rappellerons dans un premier temps le but de l'expérimentation et en quoi votre participation nous sera utile (5 minutes). Ensuite, nous vous présenterons le matériel que nous utiliserons pour cette expérimentation. Cela vous permettra d'une part de vous familiariser avec le système,et d'autre part, de vous présenter ce qu'il faut et ce qu'il ne faut pas faire avec le matériel (5 minutes). Nous vous présenterons finalement le genre de diagrammes auxquels vous serez confrontés, ainsi que le genre de réponses aux questions que nous attendons de votre part.



Google fait son caprice ? Cliquez ici pour voir la vidéo.

Veillez à être à l'heure, s'il vous plait. L'expérimentation se déroulera ainsi dans les meilleures conditions. Nous vous attendrons au bureau 2352 du pavillon André Aisenstadt. Si vous vous trouvez dans un état de stress ou de fatigue intense, vous pouvez toujours me contacter afin de trouver une éventuelle autre date. Veillez également à être sobre, sans quoi l'expérimentation devra être annulée. Si vous désirez plus d'informations, vous pouvez toujours m'envoyer un mail à l'adresse suivante: jeanmars[AT]iro.umontreal.ca. Merci de votre collaboration.

Date et Horaire Choisis

Date de l'Expérimentation: **2007.11.06 09:00:00**.

< Modifier Date et Horaire

Inscription *

Nom Email

s'inscrire Effacer

(*) Nous nous engageons à protéger votre vie privée et à ne divulguer ni commercialiser vos données personnelles à des tiers

Université de Montréal | Département d'informatique | GEODES

FIG. D.2 – Site web de recrutement – Informations complémentaires

E Spécifications utilisées durant l'expérience

JHotDraw

JHotDraw est un cadriciel (« framework ») pour réaliser des dessins techniques et structurés. Ce cadriciel fournit un support pour la création de formes géométriques et de formes personnalisées par l'utilisateur. Il permet d'éditer ces formes, et d'y ajouter des contraintes comportementales.

Les applications JHotDraw sont définies par la classe DrawingApplication.
Cette classe est responsable de la création de l'interface usager et de l'accès aux éléments principaux de l'application JHotDraw.

Dans JHotDraw, toutes les applications sont composées d'un "Drawing", d'une "Drawing view" associée au « Drawing », d'un ensemble de formes et d'au moins un outil (Tool).

Les formes qui composent un dessin sont représentées par le concept de "Figure".

Rappel : soyez le plus détaillé possible (classes/méthodes/relations/héritages/...) lorsque vous donnerez votre réponse.

FIG. E.1 – JHotDraw – Spécification

JRefactory

JRefactory est un outil de restructuration de code (« refactoring ») pour les programmes Java.

Une restructuration de code est définie comme "tout changement dans le code d'un programme qui permet d'améliorer sa lisibilité ou qui permet de changer sa structure sans en changer ses résultats". Une restructuration ne corrige cependant pas les bogues, et n'ajoute pas de nouvelles fonctionnalités. Elle améliore uniquement la compréhension du code, change sa structure interne, et retire le code mort. Il est possible d'appliquer des restructurations de champs, de méthodes, de classes.

La partie du système qui va être présentée permet d'effectuer un refactoring de champs/attributs de classe. Il est possible de "remonter" les champs/attributs dans une classe parente.
Il est également possible de renommer les champs/attributs d'une classe.

La méthode "run" de la classe "Refactoring" est appelée pour appliquer la restructuration du code désirée.

Rappel : soyez le plus détaillé possible (classes/méthodes/relations/héritages/...) lorsque vous donnerez votre réponse.

FIG. E.2 – JRefactory – Spécification

PADL

PADL est un méta-modèle pour décrire des programmes orientés-objets, c'est une bibliothèque qui offre des classes et des méthodes permettant de décrire un programme, comme le vocabulaire et la grammaire française permettent de faire des phrases. Ce méta-modèle est similaire au méta-modèle UML.

Pour représenter des programmes, PADL offre des classes, des interfaces pour représenter les concepts de « classes », d'« interfaces », de « méthodes », de « champs » et de « relations » entre interfaces/classes. Ces constituants forment le vocabulaire.

PADL offre aussi un ensemble de méthodes pour instancier et connecter entre elles les représentations des constituants d'un programme. Ces méthodes forment la grammaire.

Rappel : soyez le plus détaillé possible (classes/méthodes/relations/héritages/...) lorsque vous donnerez votre réponse.

Fig. E.3 – PADL – Spécification

F Diagrammes utilisés durant l'expérience

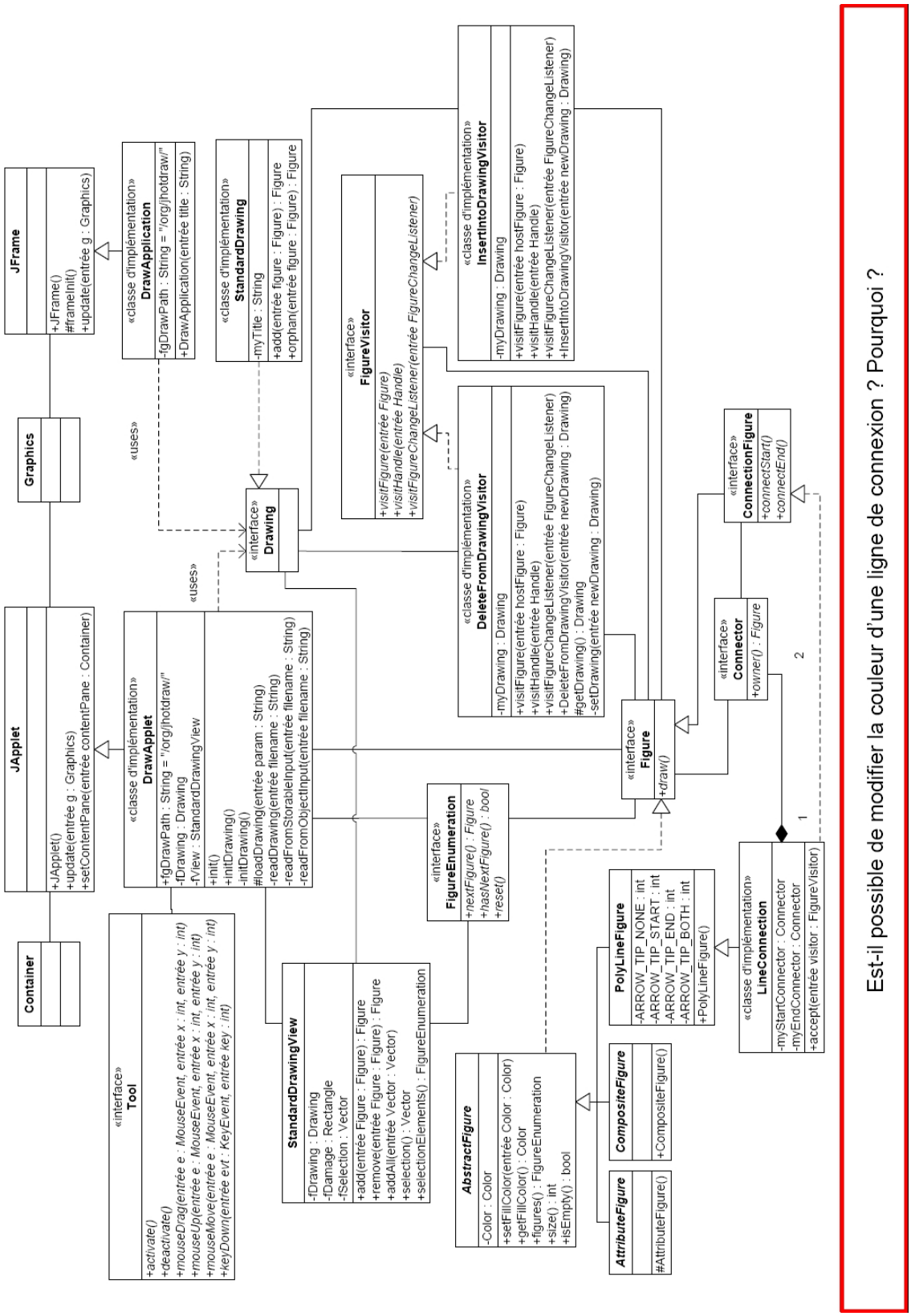


FIG. F.1 – JHotDraw – Classical Pattern – Compréhension

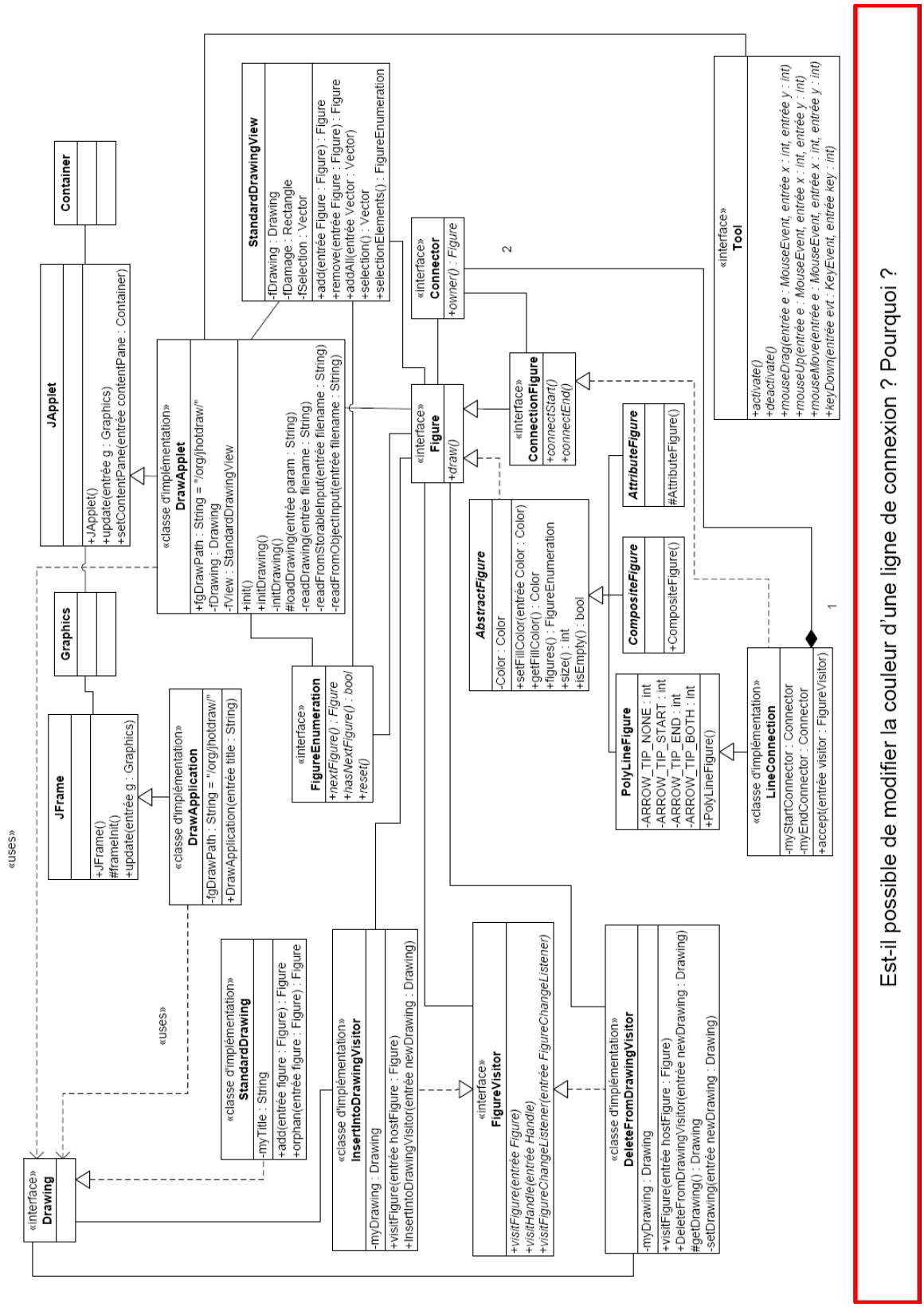


FIG. F.2 – JHotDraw – Modified Pattern – Compréhension

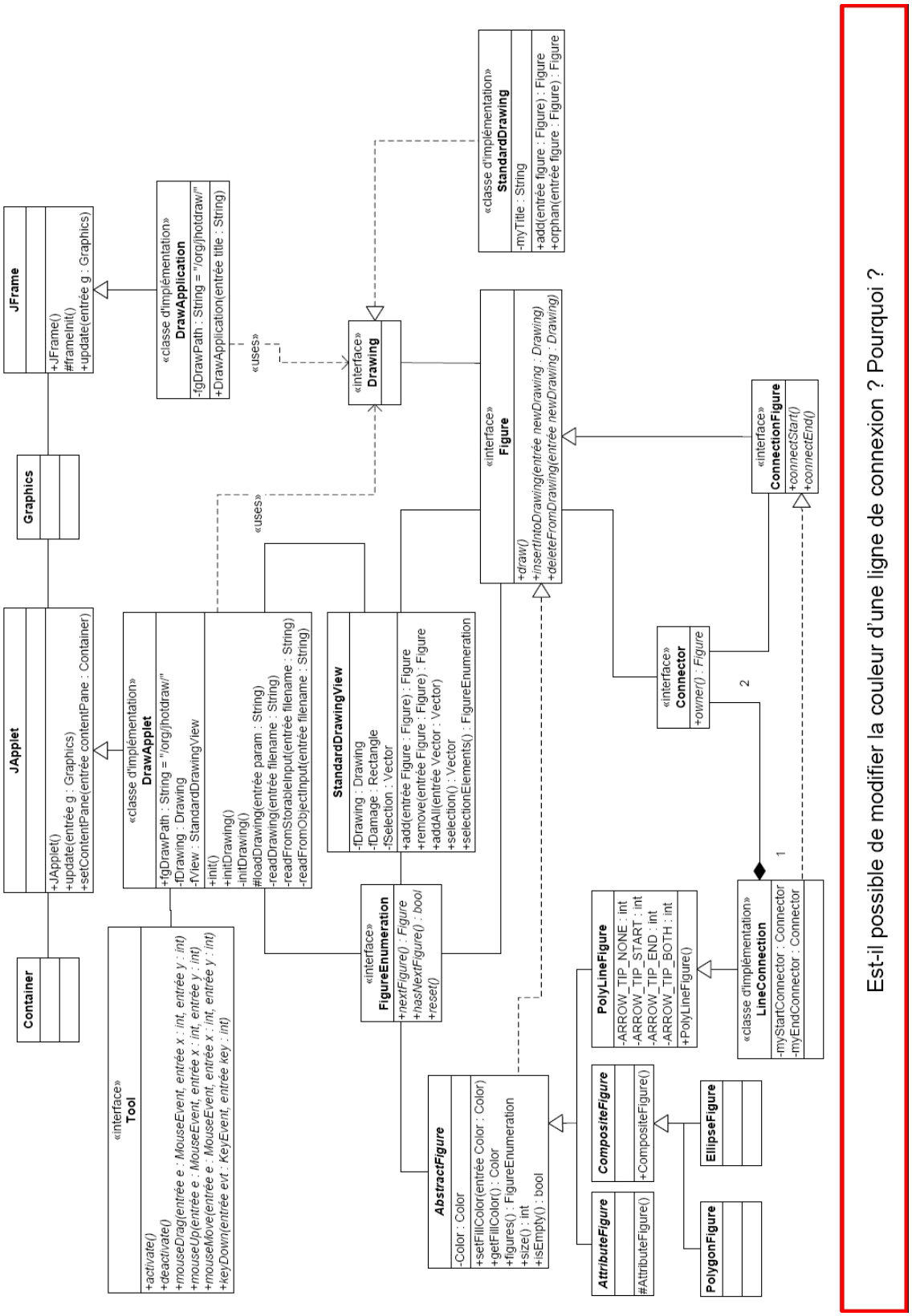


FIG. F.3 – JHotDraw – No Pattern – Compréhension

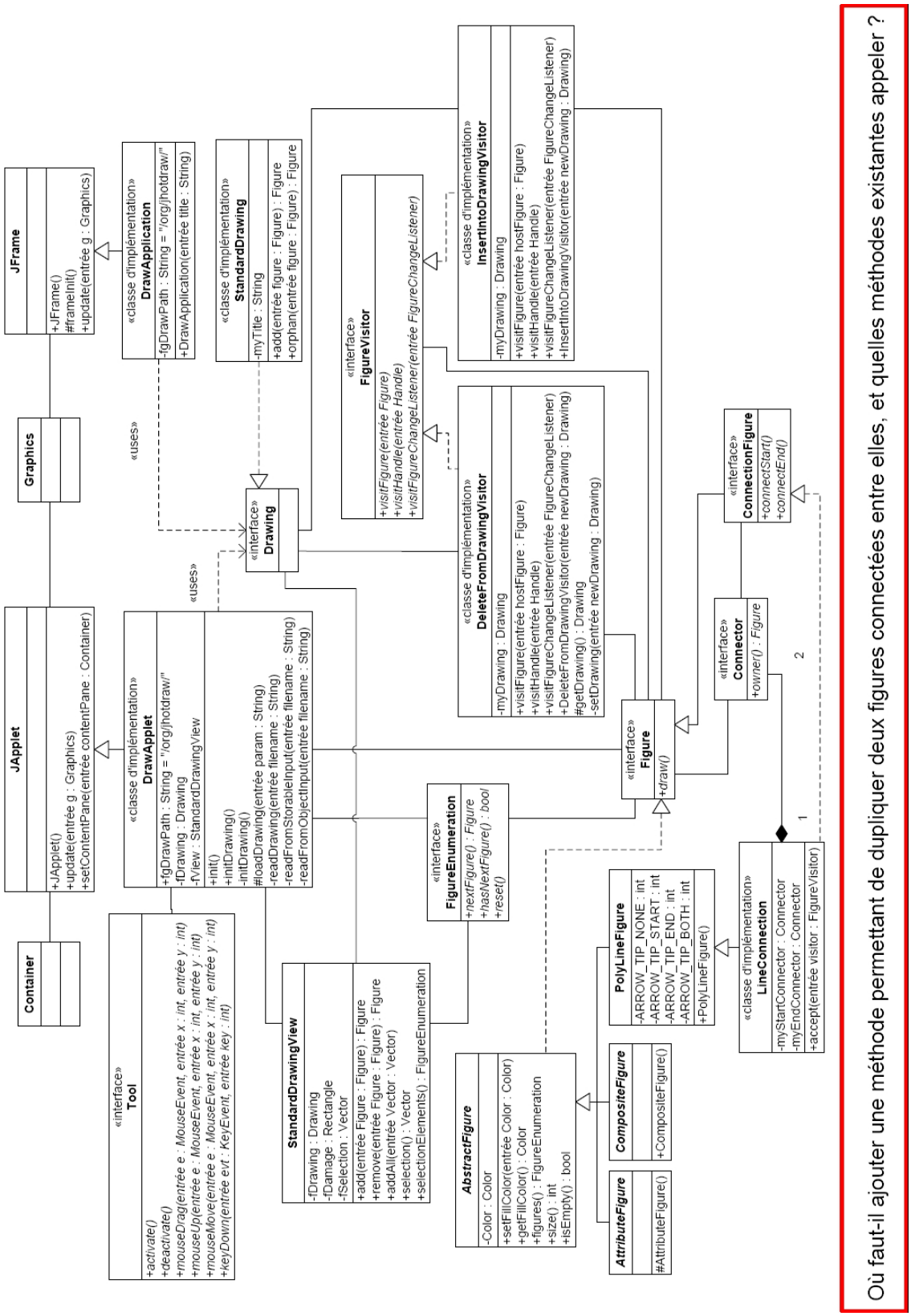


FIG. F.4 – JHotDraw – Classical Pattern – Maintenance

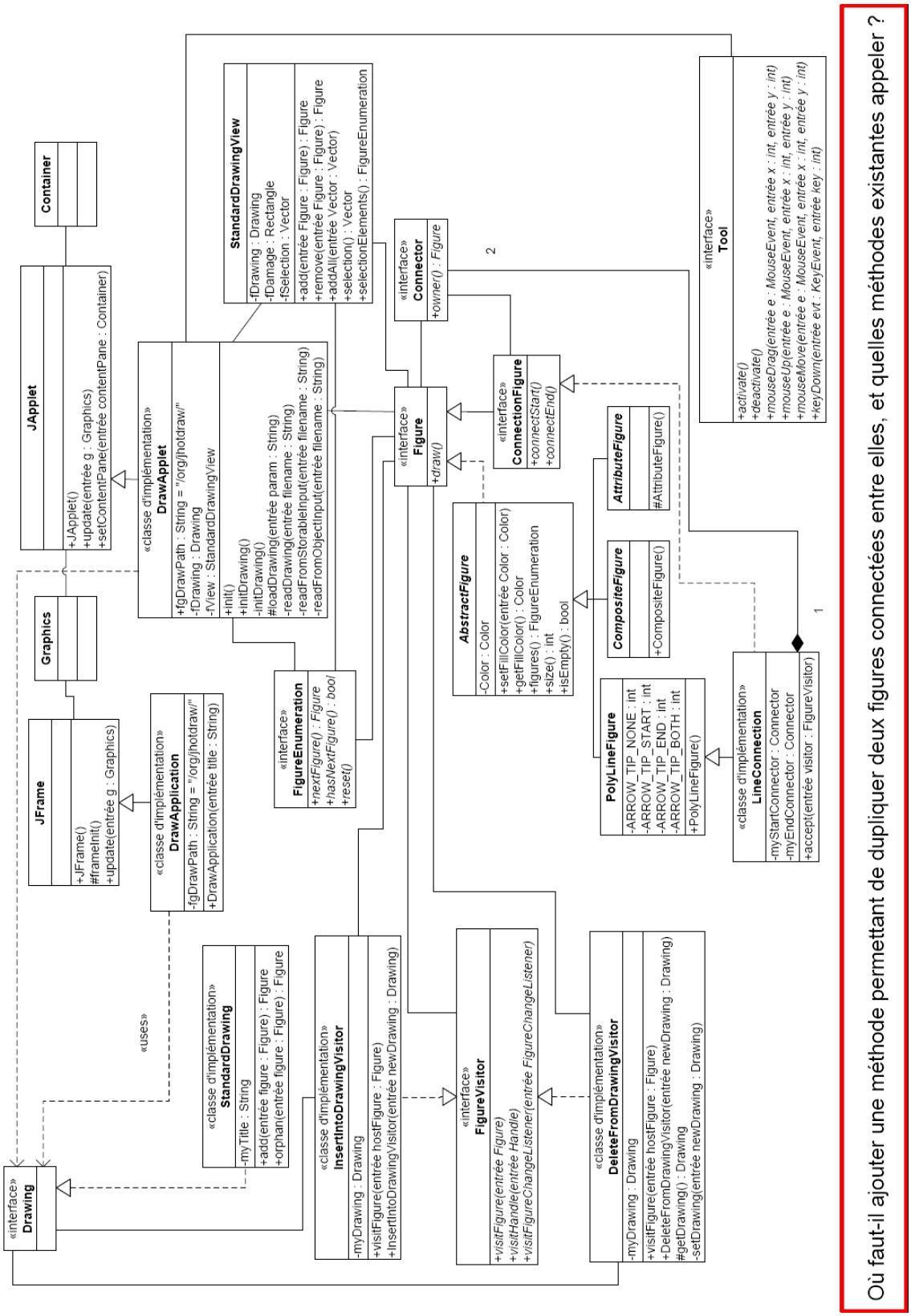


FIG. F.5 – JHotDraw – Modified Pattern – Maintenance

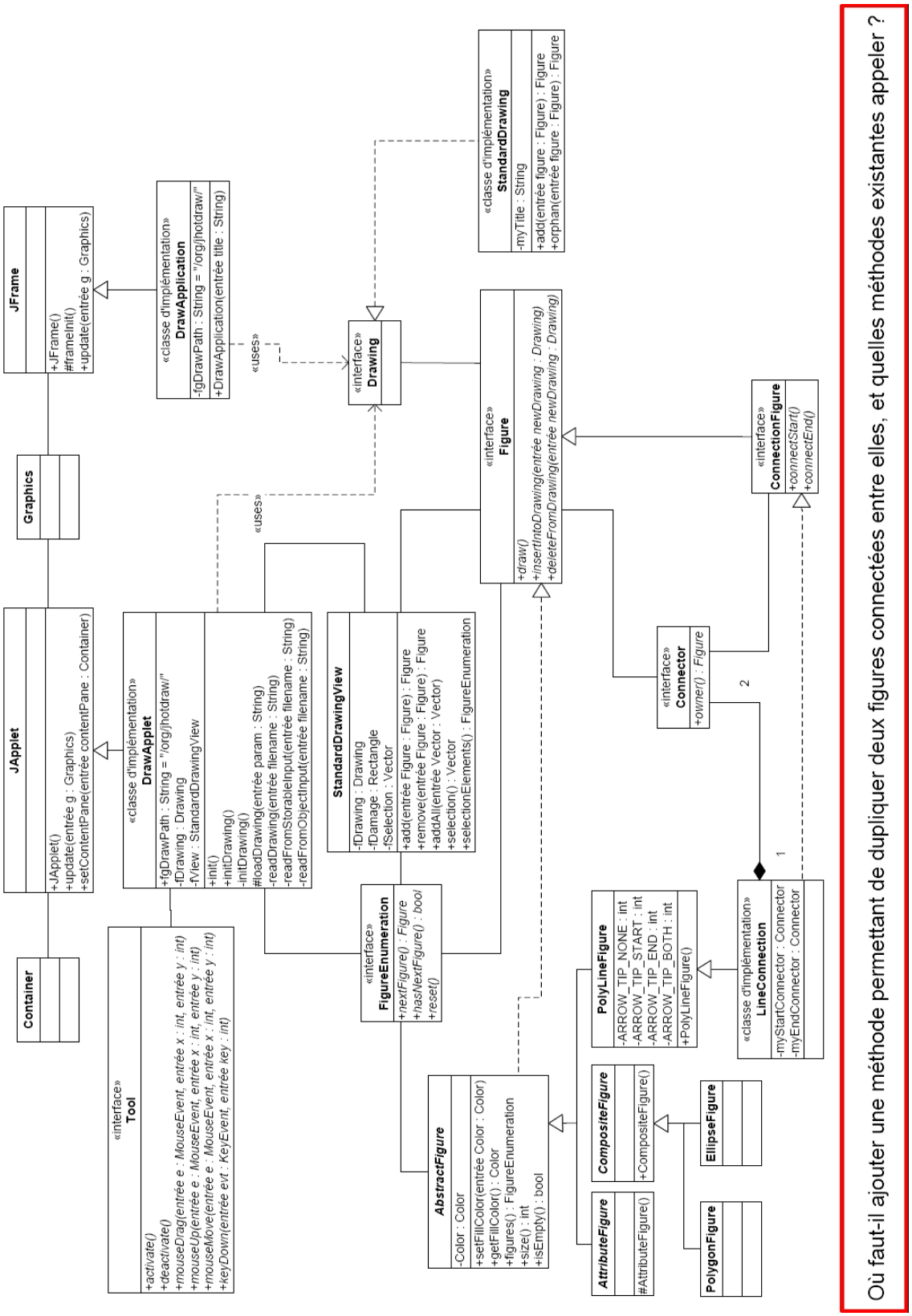


FIG. F.6 – JHotDraw – No Pattern – Maintenance

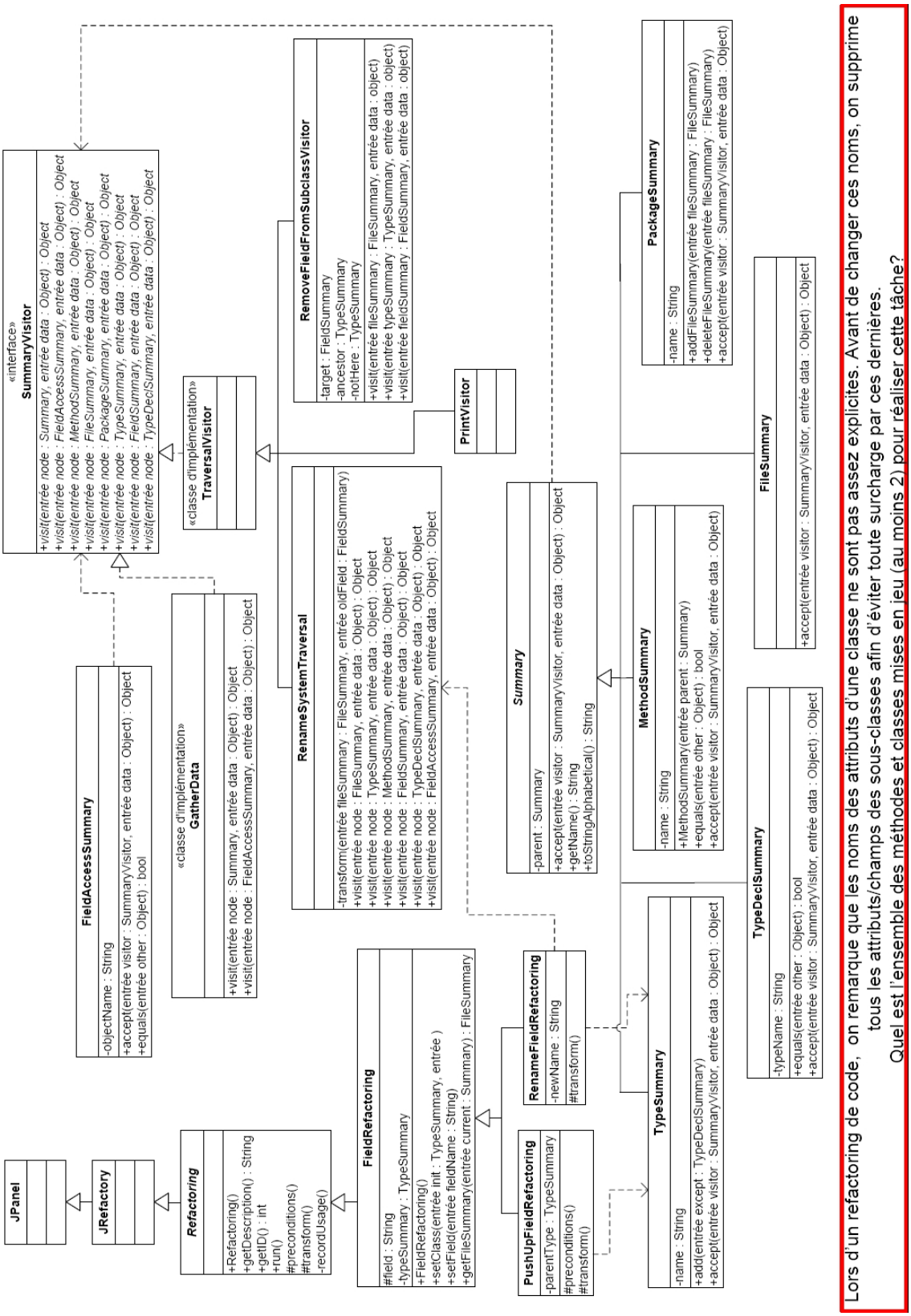


FIG. F.7 – JRefactory – Classical Pattern – Compréhension

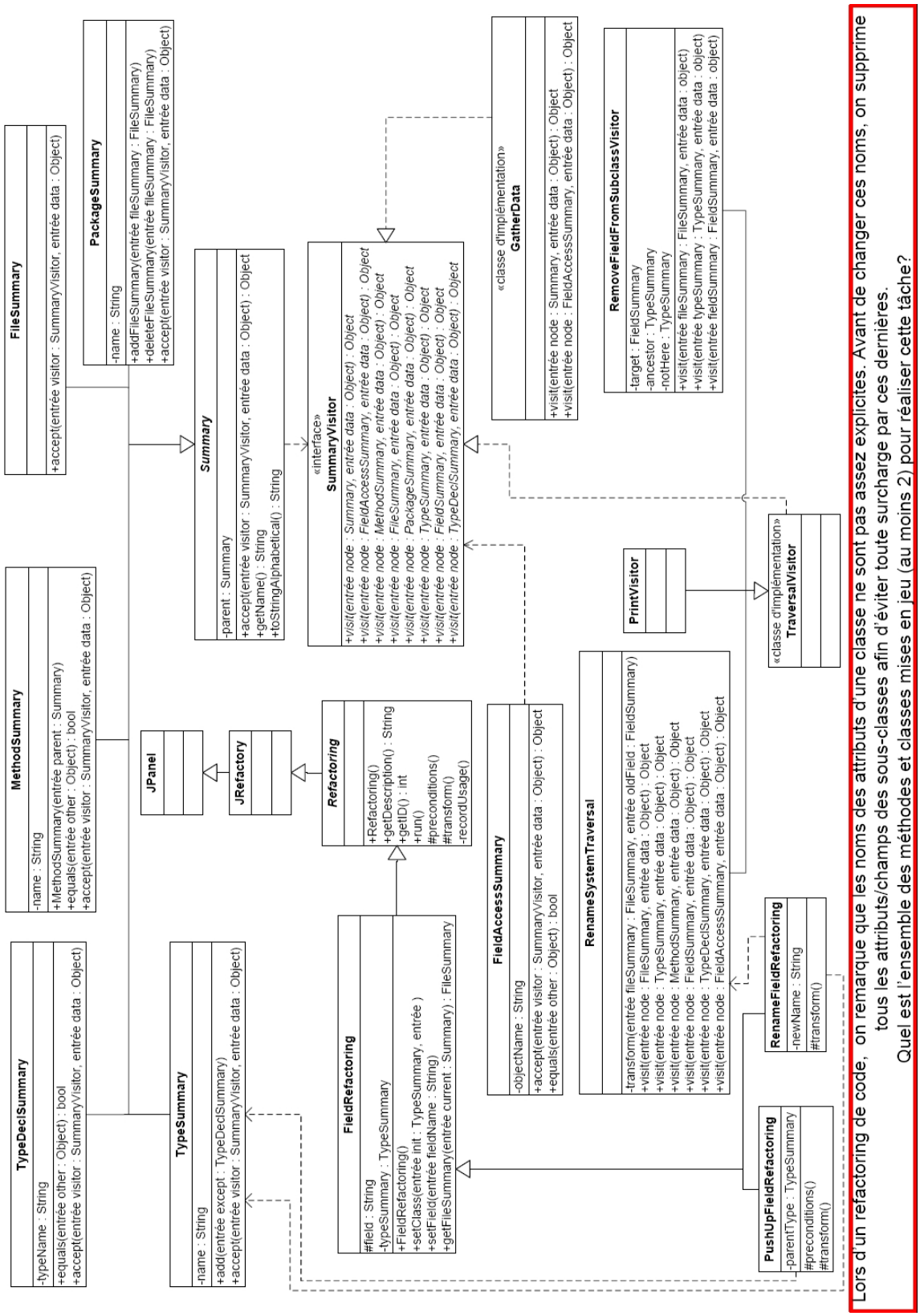


FIG. F.8 – JRefactory – Modified Pattern – Compréhension

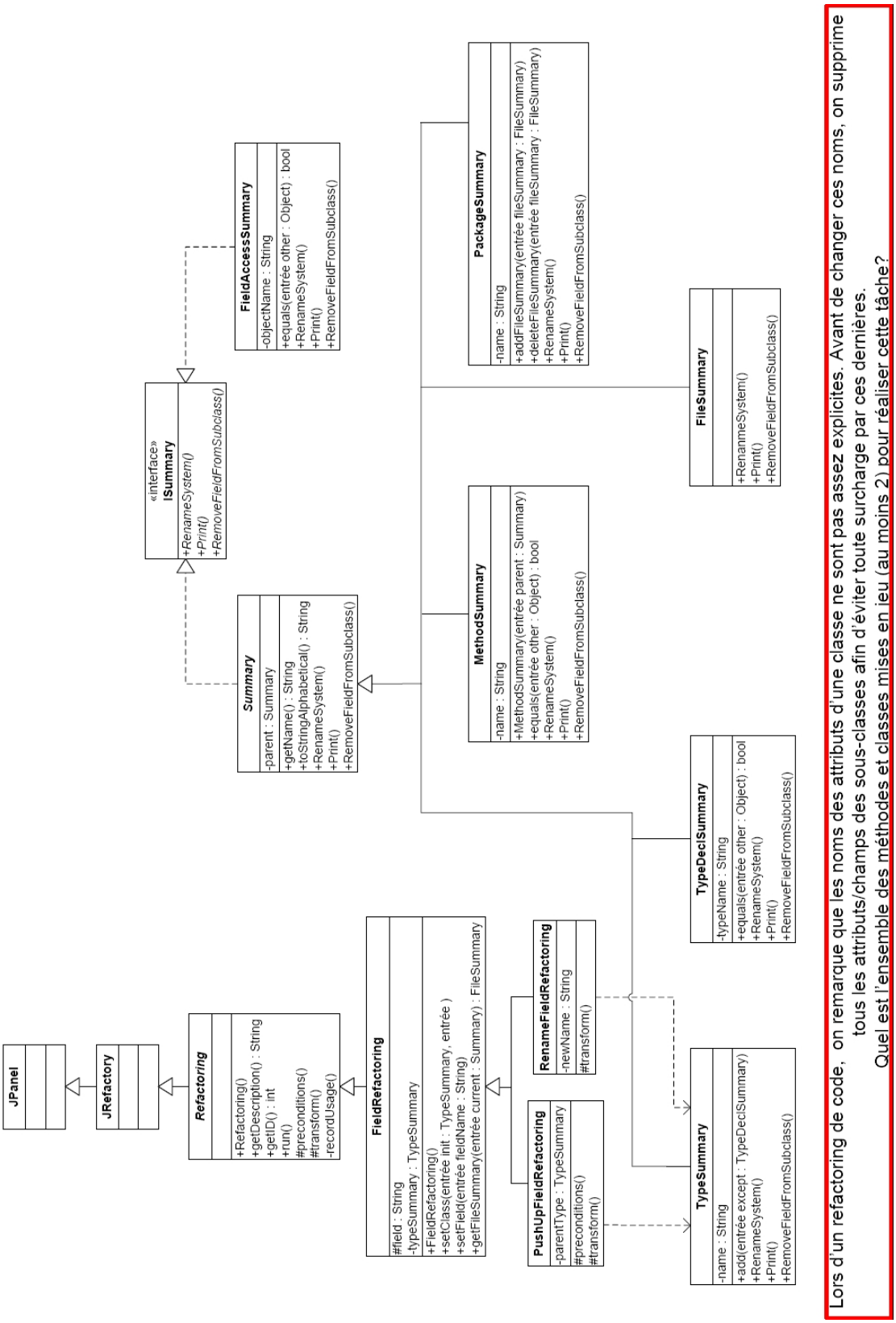


FIG. F.9 – JRefactory – No Pattern – Compréhension

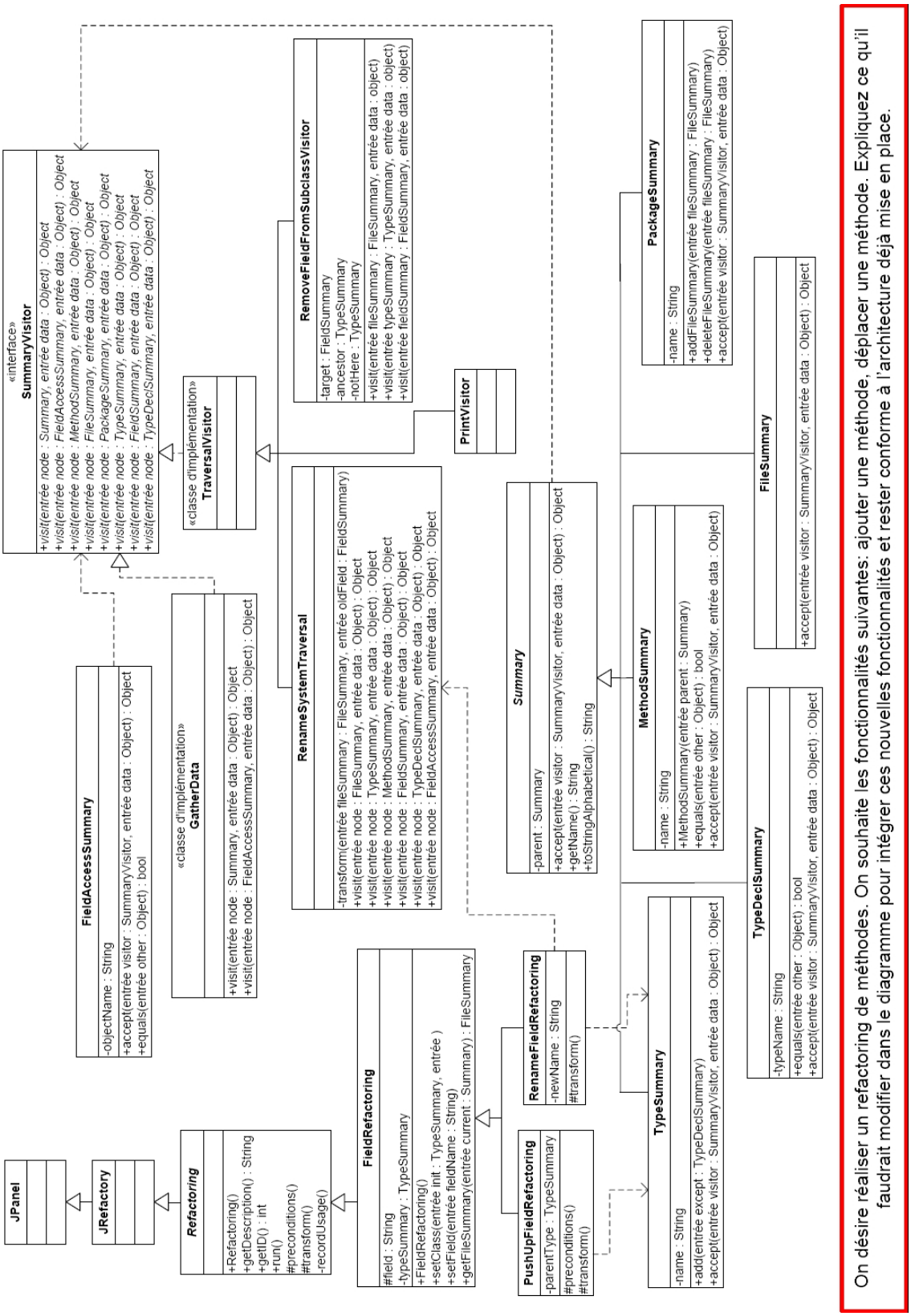


FIG. F.10 – JRefactory – Classical Pattern – Maintenance

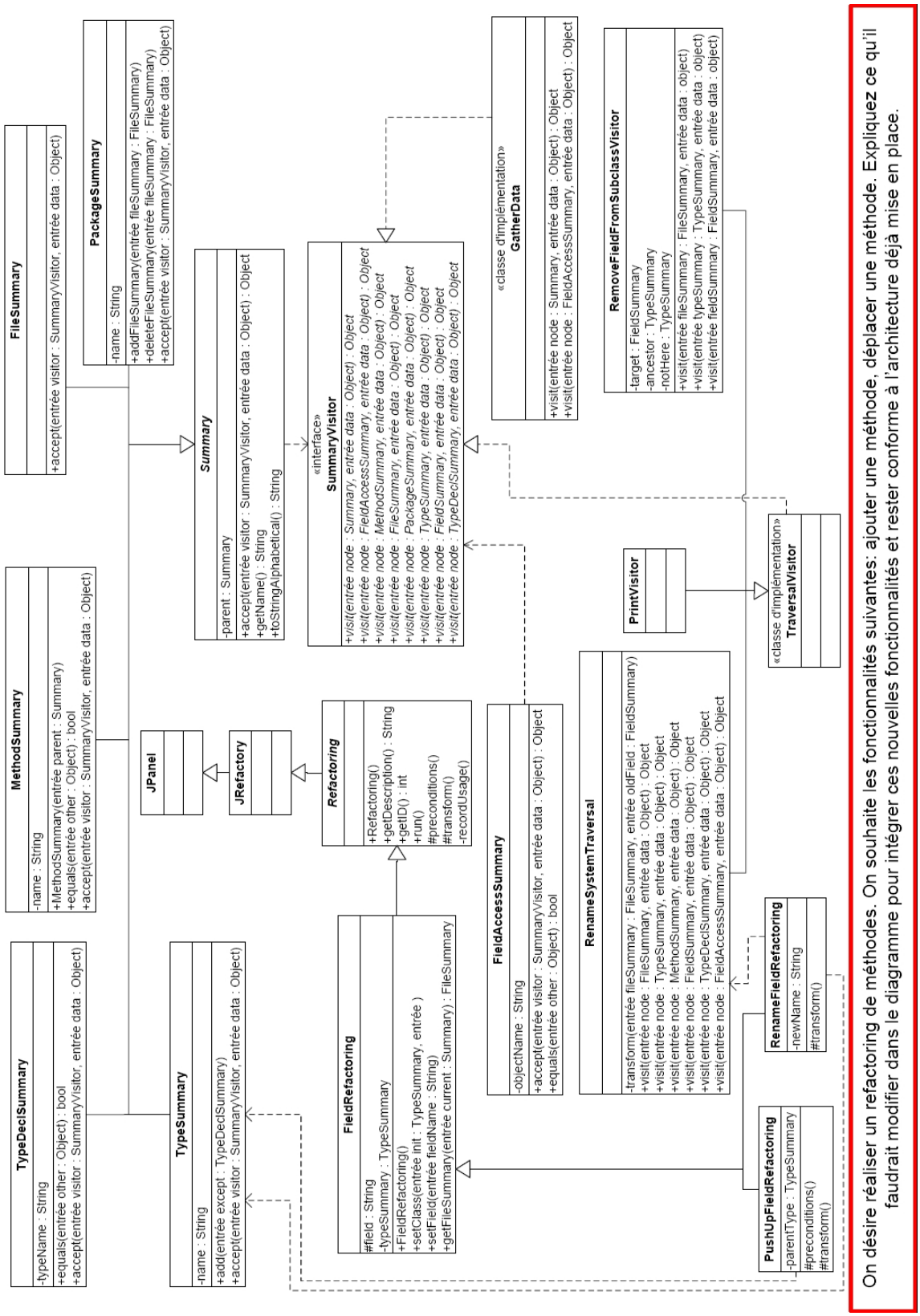


FIG. F.11 – JRefactory – Modified Pattern – Maintenance

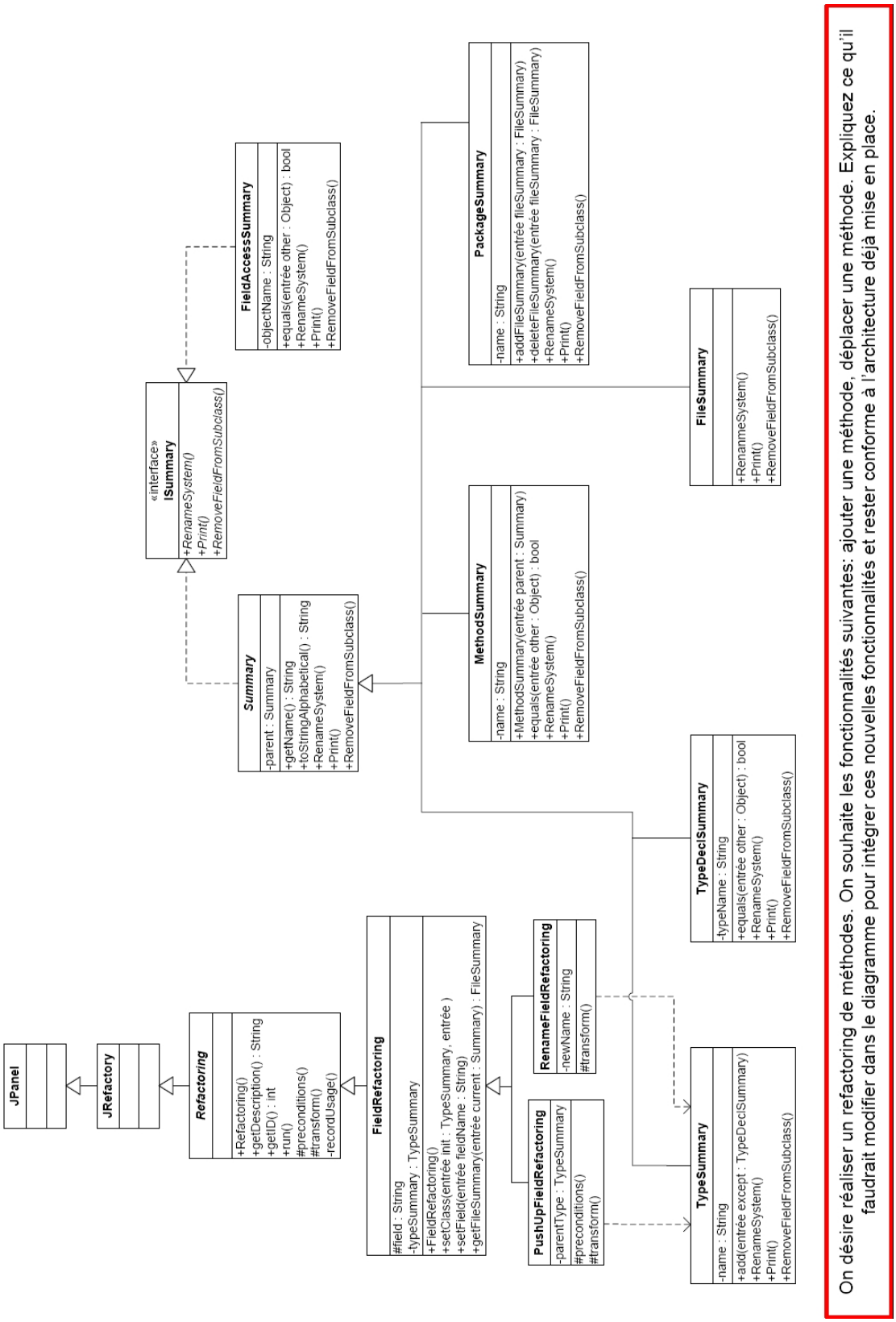


FIG. F.12 – JRefactory – No Pattern – Maintenance

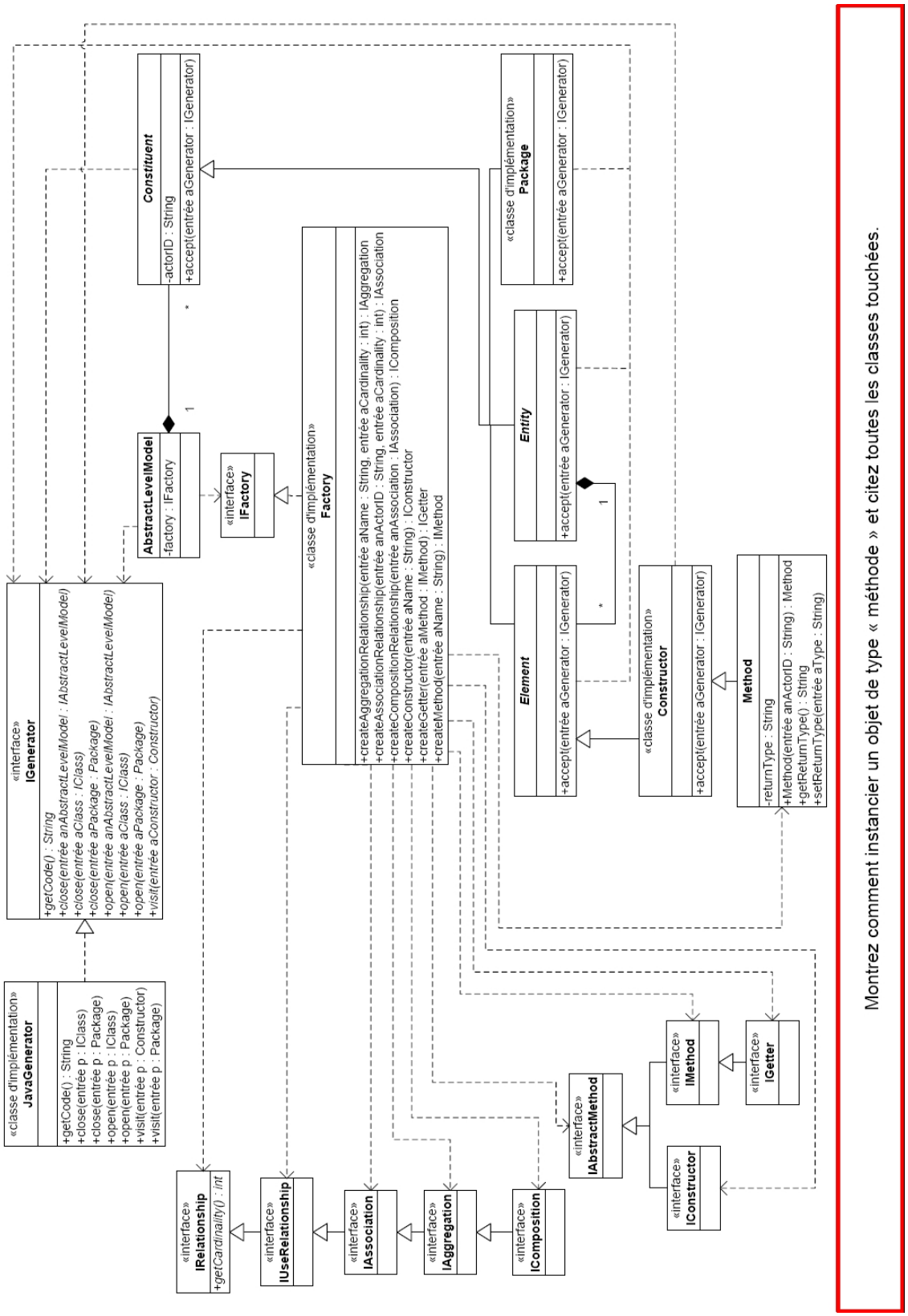


FIG. F.13 – PADL – Classical Pattern – Compréhension

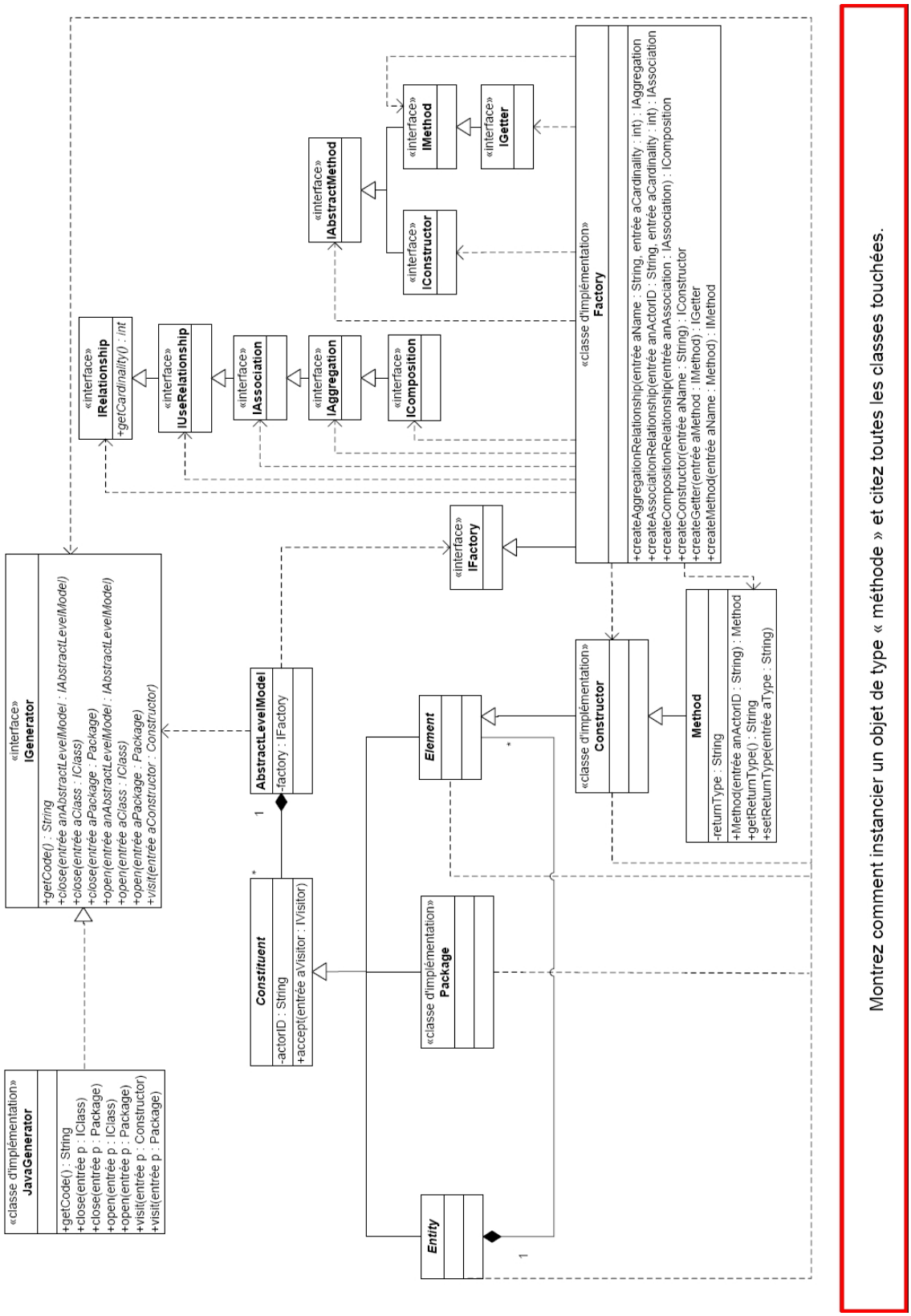


FIG. F.14 – PADL – Modified Pattern – Compréhension

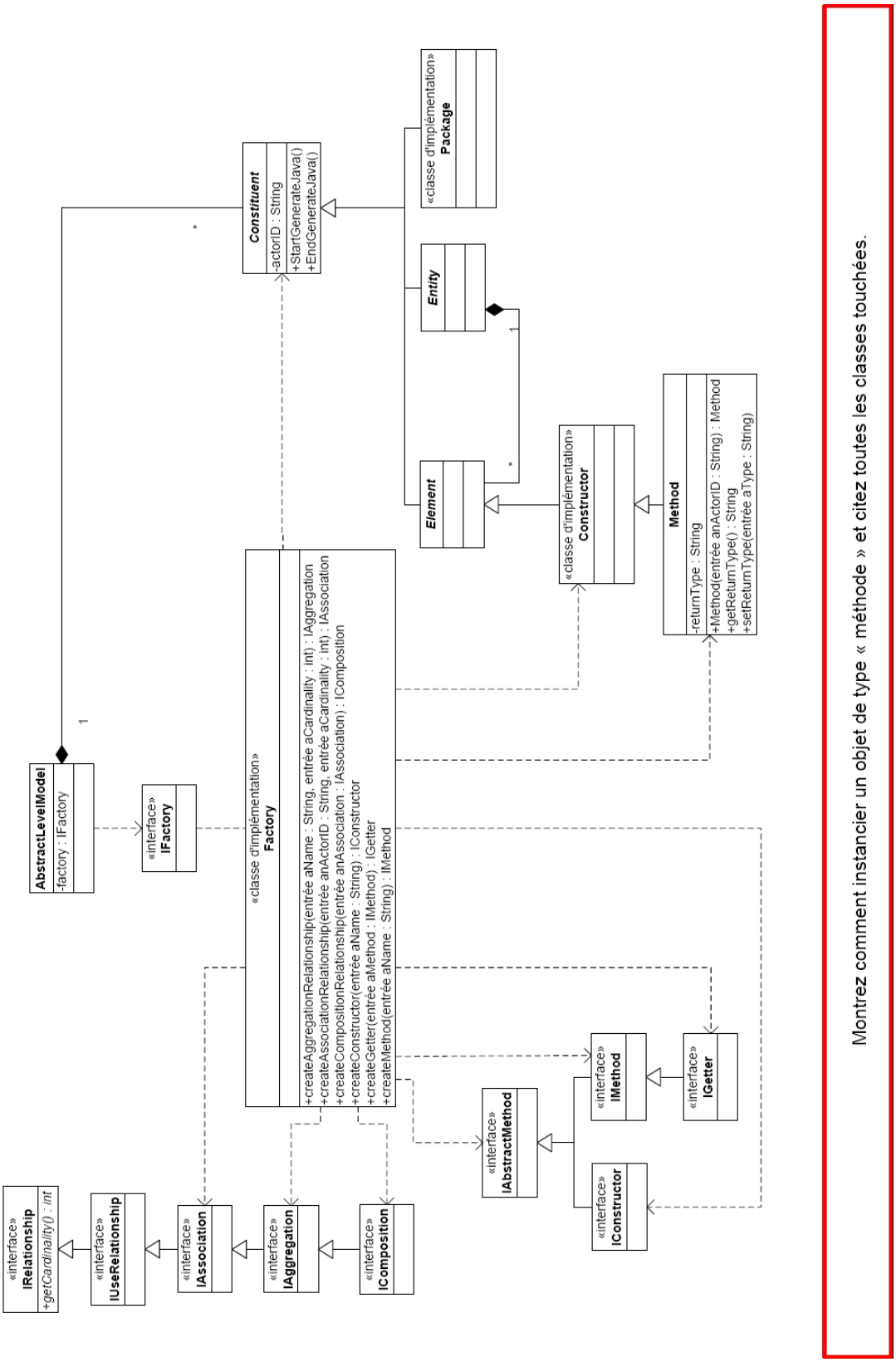


Fig. F.15 – PADL – No Pattern – Compréhension

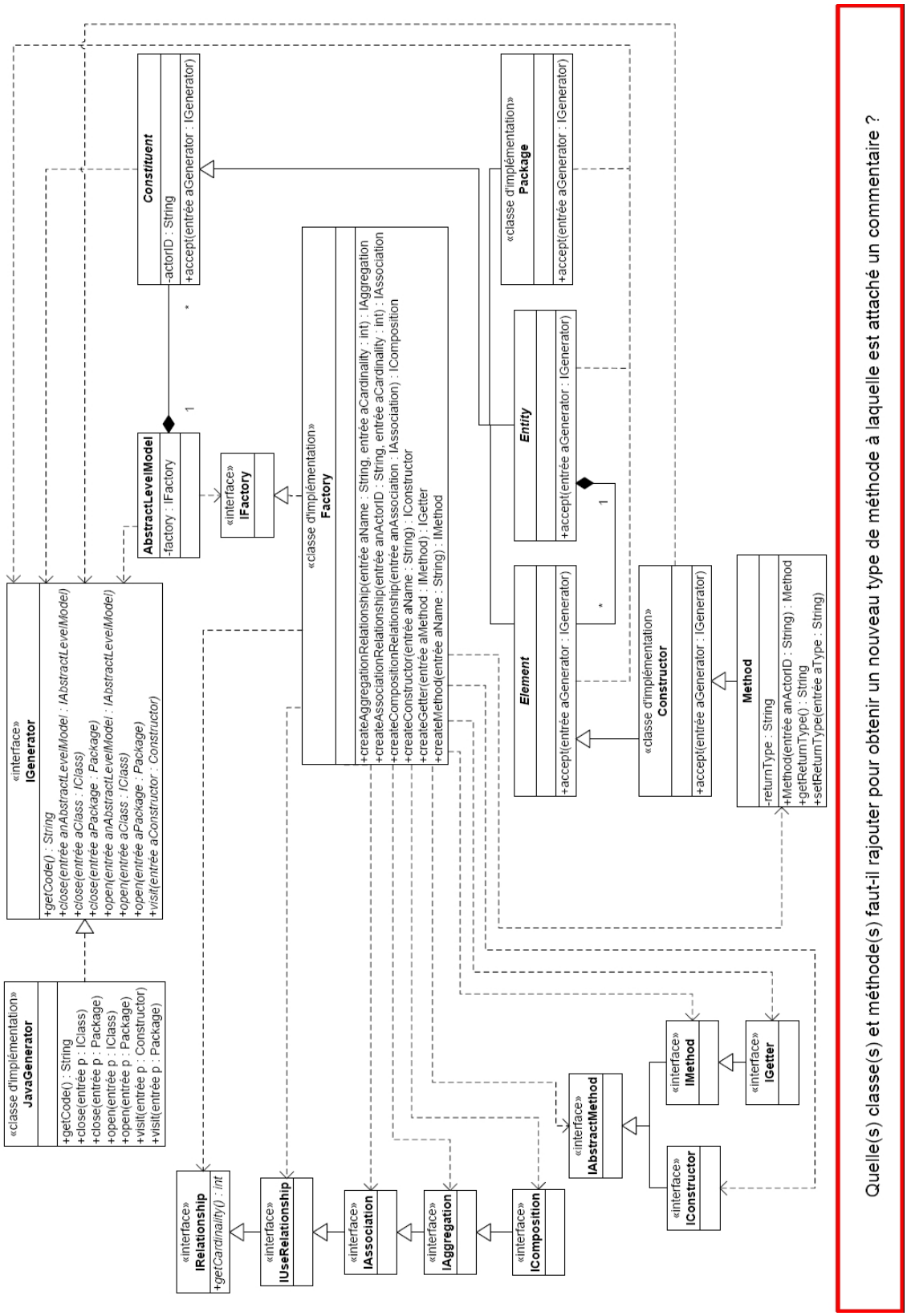


FIG. F.16 – PADL – Classical Pattern – Maintenance

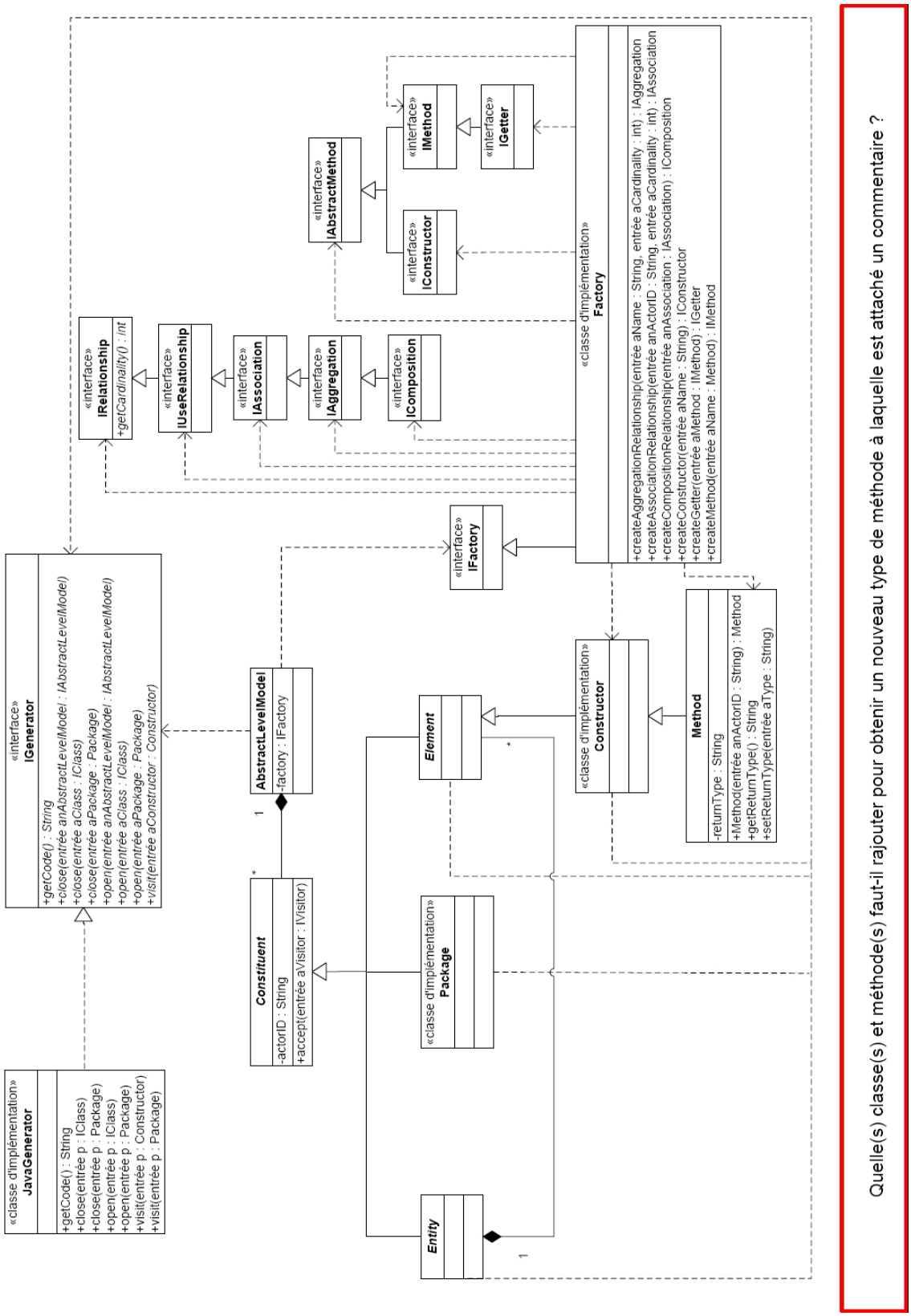


FIG. F.17 – PADL – Modified Pattern – Maintenance

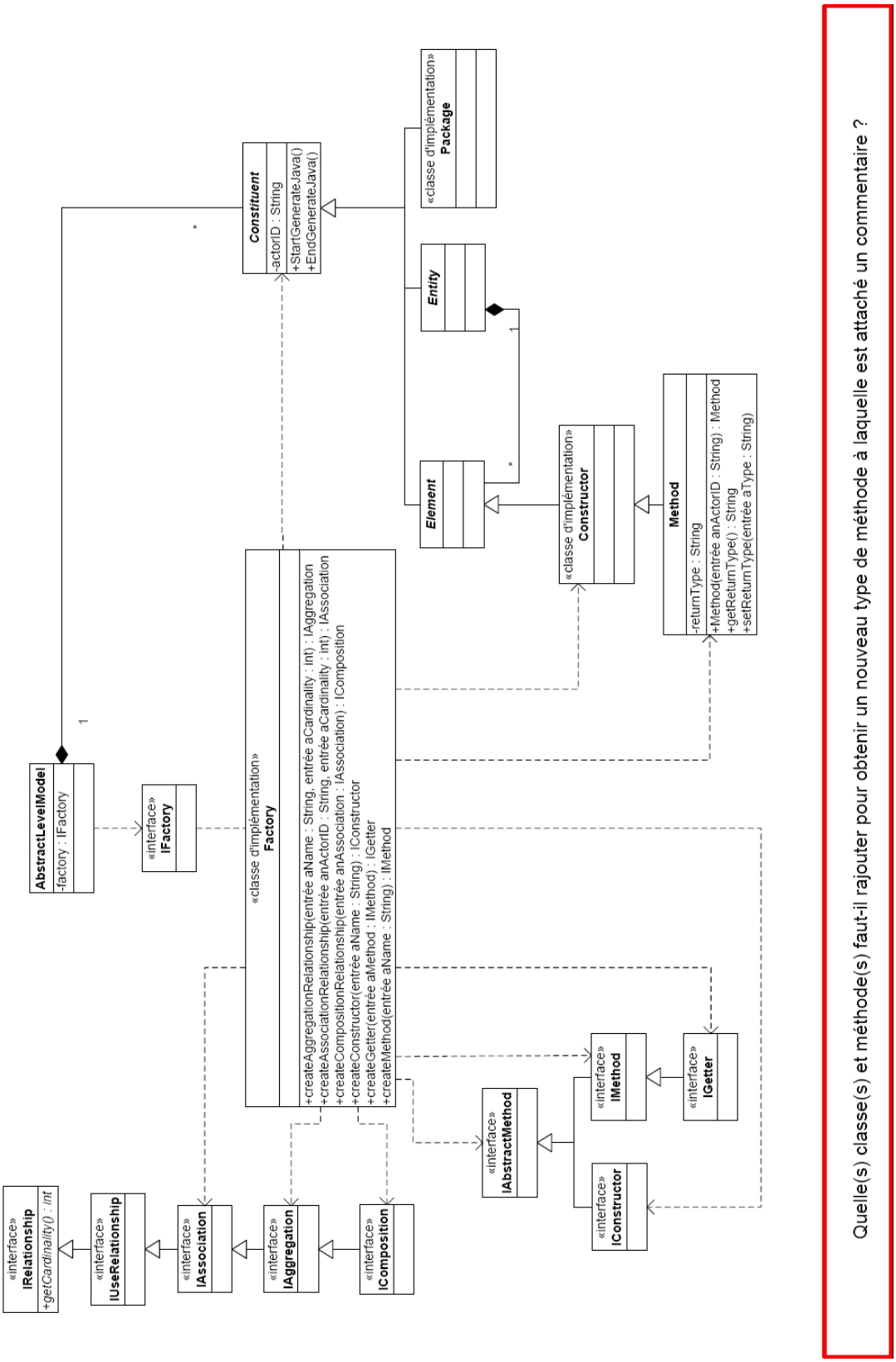


FIG. F.18 – PADL – No Pattern – Maintenance

G Check lists

JhotDraw (no pattern)			
Comprehension		**Maintenance**	
Est-il possible de modifier la couleur d'une ligne de connexion? Pourquoi?		***Où faut-il ajouter une méthode permettant de dupliquer deux figures liées entre elles, et quelles méthodes existantes appeler?***	
1. oui		1. Une nouvelle méthode dans LineConnection du style « duplicate»	
2. Classes : «lineConnection», «AbstractFigure»		2. Appeler la classe "Connector"	
3. Color/setFillColor dans AbstractFigure		3. Appeler la méthode «InsertIntoDrawing» dans « Figure »	

JRefactory (modified)			
Comprehension		**Maintenance**	
Lors d'un refactoring de code, on remarque que les noms d'attributs d'une classe ne sont pas assez explicites. Avant de changer ces noms, on supprime tous les attributs/champs des sous-classes afin d'éviter toute surcharge par ces dernières. Quel est l'ensemble des méthodes et des classes mises en jeu (au moins 2) pour réaliser cette tâche ?		***On désire réaliser un refactoring de méthodes. On souhaite les fonctionnalités suivantes : ajouter une méthode, déplacer une méthode. Expliquez ce qu'il faudrait modifier dans le diagramme pour intégrer ces nouvelles fonctionnalités et rester conforme à l'architecture déjà mise en place.***	
1. Classe : «RemoveFieldFromSubclass»		1. Classes : 2 nouvelles classes ou 1 classe parente et 2 sous classes	
2. Classe : «RenameFieldRefactoring»		2. Lien : nouvelles classes vers «MethodSummary»	
3. Méthodes : «transform», «visit»			

PADL (classical)			
Comprehension		**Maintenance**	
Montrez comment instancier un objet de type « méthode » et citez toutes les classes touchées.		***Quelle(s) classe(s) et méthode(s) faut-il rajouter pour obtenir un nouveau type de méthode à laquelle est attachée une notation ?***	
1. Classe : «IMethod»		1. New Classe : sous-classe de "Method"	
2. Classe : «Method»		2. New Interface : sous-interface de "IMethod"	
3. Méthode : «createMethod(string) : IMethod» dans "Factory"		3. New Attribut : String Commentaire	
		4. New Méthode : "createMethod(String,String) : NewMethod" dans "Factory"	
		5. New Méthodes : set/get(commentaire)	

FIG. G.1 – Checklist - Configuration 1

JhotDraw (classical)			
Comprehension		**Maintenance**	
Est-il possible de modifier la couleur d'une ligne de connexion? Pourquoi?		***Où faut-il ajouter une méthode permettant de dupliquer deux figures liées entre elles, et quelles méthodes existantes appeler?***	
1. oui		1. Un visiteur, sous classe de « FigureVisitor »	
2. Classes : «lineConnection», «AbstractFigure»		2. Une méthode "visitLineConnection" dans le nouveau visiteur	
3. Color/setFillColor dans AbstractFigure		3. Appeler la classe « Connector »	
		4. Appeler la méthode « InsertIntoDrawingVisitor » ou la classe « InsertIntoDrawing »	

JRefactory (no pattern)			
Comprehension		**Maintenance**	
Lors d'un refactoring de code, on remarque que les noms d'attributs d'une classe ne sont pas assez explicites. Avant de changer ces noms, on supprime tous les attributs/champs des sous-classes afin d'éviter toute surcharge par ces dernières. Quel est l'ensemble des méthodes et des classes mises en jeu (au moins 2) pour réaliser cette tâche ?		***On désire réaliser un refactoring de méthodes. On souhaite les fonctionnalités suivantes : ajouter une méthode, déplacer une méthode. Expliquez ce qu'il faudrait modifier dans le diagramme pour intégrer ces nouvelles fonctionnalités et rester conforme à l'architecture déjà mise en place.***	
1. Classe : «RenameFieldRefactoring», «TypeSummary»		1. Classes : 2 nouvelles classes ou 1 classe parente et 2 sous classes	
2. Méthode : « removeFieldFromSubClass »		2. Lien : nouvelles classes vers «MethodSummary»	

PADL (modified)			
Comprehension		**Maintenance**	
Montrez comment instancier un objet de type « méthode » et citez toutes les classes touchées.		***Quelle(s) classe(s) et méthode(s) faut-il rajouter pour obtenir un nouveau type de méthode à laquelle est attachée une notation ?***	
1. Classe : «IMethod»		1. New Classe : sous-classe de "Method"	
2. Classe : «Method»		2. Une sous-interface de "IMethod"	
3. Méthode : «createMethod(string) : IMethod» dans "Factory"		3. New Attribut : String Commentaire	
		4. New Méthode : "createMethod(String,String) : NewMethod" dans "Factory"	
		5. New Méthodes : set/get(commentaire)	

FIG. G.2 – Checklist - Configuration 2

JhotDraw (modified)			
Comprehension		**Maintenance**	
Est-il possible de modifier la couleur d'une ligne de connexion? Pourquoi?		***Où faut-il ajouter une méthode permettant de dupliquer deux figures liées entre elles, et quelles méthodes existantes appeler?***	
1. oui		1. Un visiteur, sous classe de « FigureVisitor »	
2. Classes : «lineConnection», «AbstractFigure»		2. Une méthode "visitLineConnection" dans le nouveau visiteur	
3. Color/setFillColor dans AbstractFigure		3. Appeler la classe « Connector »	
		4. Appeler la méthode « InsertIntoDrawingVisitor » ou la classe « InsertIntoDrawing »	

JRefactory (classical)			
Comprehension		**Maintenance**	
Lors d'un refactoring de code, on remarque que les noms d'attributs d'une classe ne sont pas assez explicites. Avant de changer ces noms, on supprime tous les attributs/champs des sous-classes afin d'éviter toute surcharge par ces dernières. Quel est l'ensemble des méthodes et des classes mises en jeu (au moins 2) pour réaliser cette tâche ?		***On désire réaliser un refactoring de méthodes. On souhaite les fonctionnalités suivantes : ajouter une méthode, déplacer une méthode. Expliquez ce qu'il faudrait modifier dans le diagramme pour intégrer ces nouvelles fonctionnalités et rester conforme à l'architecture déjà mise en place.***	
1. Classe : «RemoveFieldFromSubclass»		1. Classes : 2 nouvelles classes ou 1 classe parente et 2 sous classes	
2. Classe : «RenameFieldRefactoring»		2. Lien : nouvelles classes vers «MethodSummary»	
3. Méthodes : «transform», «visit»			

PADL (no pattern)			
Comprehension		**Maintenance**	
Montrez comment instancier un objet de type « méthode » et citez toutes les classes touchées.		***Quelle(s) classe(s) et méthode(s) faut-il rajouter pour obtenir un nouveau type de méthode à laquelle est attachée une notation ?***	
1. Classe : «IMethod»		1. New Classe : sous-classe de "Method"	
2. Classe : «Method»		2. Une sous-interface de "IMethod"	
3. Méthode : «createMethod(string) : IMethod» dans "Factory"		3. New Attribut : String Commentaire	
		4. New Méthode : "createMethod(String,String) : NewMethod" dans "Factory"	
		5. New Méthodes : set/get(commentaire)	

FIG. G.3 – Checklist - Configuration 3

H Tâches de compréhension et de maintenance

JHotDraw	
Compréhension	
1.	Est-il possible de modifier la couleur d'une ligne de connexion? Pourquoi?
Modification	
1.	Où faut-il ajouter une méthode permettant de dupliquer deux figures liées entre elles, et quelles méthodes existantes appeler?

JRefactory	
Compréhension	
1.	Lors d'un refactoring de code, on remarque que les noms d'attributs d'une classe ne sont pas assez explicites. Avant de changer ces noms, on supprime tous les attributs/champs des sous-classes afin d'éviter toute surcharge par ces dernières. Quel est l'ensemble des méthodes et des classes mises en jeu (au moins 2) pour réaliser cette tâche (=suppression des attributs/champs)?
Modification	
1.	On désire réaliser un refactoring de méthodes. On souhaite les fonctionnalités suivantes : ajouter une méthode, déplacer une méthode. Expliquez ce qu'il faudrait modifier dans le diagramme pour intégrer ces nouvelles fonctionnalités et rester conforme à l'architecture déjà mise en place.

PADL	
Compréhension	
1.	Montrez comment instancier un objet de type « méthode » et citez toutes les classes touchées.
Modification	
1.	Quelle(s) classe(s) et méthode(s) faut-il rajouter pour obtenir un nouveau type de relation à laquelle est attachée une notation ?

FIG. H.1 – Listes des tâches demandées

I Questionnaire de l'expérience

Questionnaire Eye-Tracking

Ce questionnaire a pour but de recueillir quelques informations concernant les sujets, notamment à propos de leur connaissance sur les constructions des diagrammes UML, sur les systèmes abordés dans cette expérience (JHotDraw, JRefactory, PADL) ainsi que sur leur niveau de connaissance des patrons de conception.

Toutes les informations que vous entrerez seront totalement anonymes, tout comme les informations recueillies lors de l'expérience.

Sujet N°: ☐

UML

1. Comment jugez-vous votre connaissance des constructions des diagrammes de classes UML ?

☐ Bonne (je comprends tout) ☐ Moyenne (j'hésite sur quelques constructions) ☐ Mauvaise (je ne connais rien ou pas grand chose)

2. Avez-vous eu du mal à comprendre les diagrammes qui vous ont été montrés ?

☐ Non (j'ai tout compris) ☐ Moyennement (je n'ai pas tout compris parfaitement) ☐ Oui (j'ai vraiment compris très peu de chose)

Patrons de conception

1. Comment jugez-vous votre connaissance des patrons de conception ?

☐ Bonne (j'en connais un gd nombre) ☐ Moyenne (je connais le concept et 2-3 exemples) ☐ Mauvaise (je connais à peine le concept)

2. Votre connaissance vous a-t-elle servi pour répondre aux questions ?

☐ Oui (cela m'a aidé) ☐ Un peu ☐ Pas du tout

Systèmes

1. Quel est votre niveau de connaissance de « JHotDaw » ?

☐ Bon ☐ Moyen ☐ Mauvais

2. Est-ce que votre connaissance vous a servi à mieux répondre aux questions ?

☐ Oui ☐ Un peu ☐ Non, pas du tout

3. Quel est votre niveau de connaissance de « JRefactory » ?

☐ Bon ☐ Moyen ☐ Mauvais

4. Est-ce que votre connaissance vous a servi à mieux répondre aux questions ?

☐ Oui ☐ Un peu ☐ Non, pas du tout

5. Quel est votre niveau de connaissance de PADL ?

☐ Bon ☐ Moyen ☐ Mauvais

6. Est-ce que votre connaissance vous a servi à mieux répondre aux questions ?

☐ Oui ☐ Un peu ☐ Non, pas du tout

Niveau d'étude

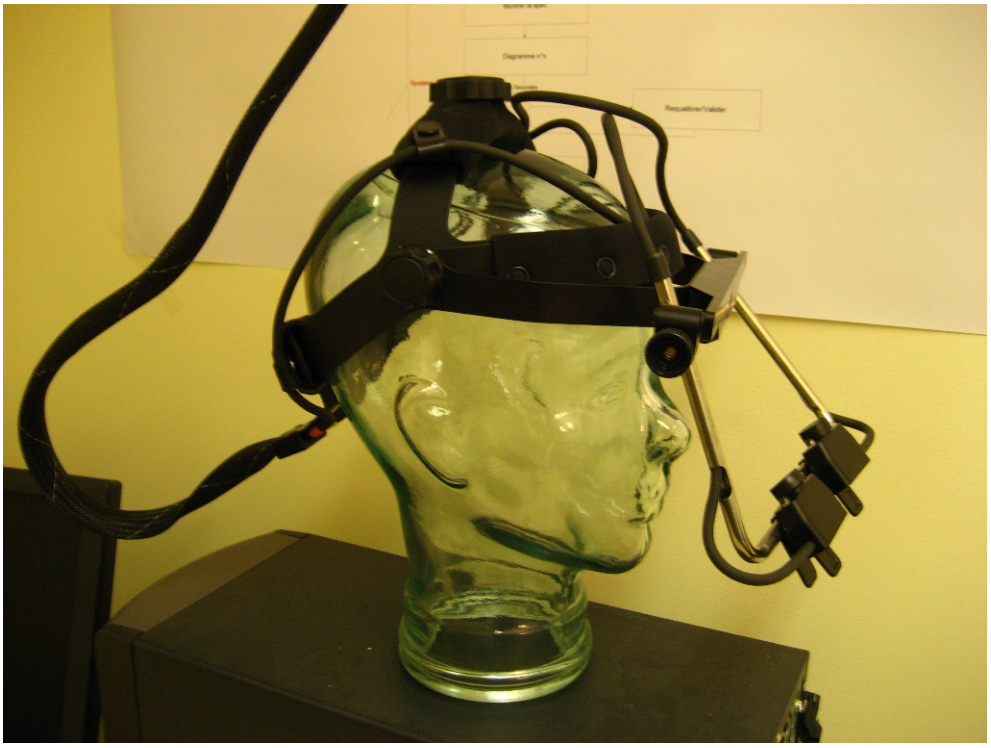
1. Quel est votre niveau d'étude ?

☐ Baccalauréat ☐ Maitrise ☐ Doctorat

FIG. I.1 – Questionnaire

J Images du système d'*eye-tracking*

(a) Ordinateur de l'expérimentateur et casque de l'«*eye-tracker*»



(b) Casque de l'«*eye-tracker*»

FIG. J.1 – Système d'*eye-tracking* utilisé (1)

(a) Système d'*eye-tracking* Eyelink II

(b) Ecran de calibration de la pupille

FIG. J.2 – Système d'*eye-tracking* utilisé (2)