Table des matières

PRÉDICTION DES EFFORTS DE DÉVELOPPEMENT ET DE TEST À PARTIR DES CAS D'UTILISATION : UNE ÉTUDE EXPLORATOIRE

Le cycle de développement d'un logiciel est composé de plusieurs étapes allant de la phase d'analyse jusqu'à la phase de test et de mise au point. Les dernières phases du cycle de développement, soit les phases de développement et de test, sont les plus coûteuses en termes d'effort, de temps et de ressources. Il devient donc important de pouvoir faire une prévision des efforts relatifs à ces étapes, idéalement le plus tôt possible dans le processus, pour que les ressources et activités du projet puissent être gérées et allouées de façon optimale. Ce projet vise à présenter et explorer une nouvelle méthode pour pouvoir effectuer la prévision de l'effort de développement et de test à partir des cas d'utilisation. La méthode utilise essentiellement cinq métriques qui peuvent être collectées dans les premières phases du cycle de développement et les compare avec des métriques réelles représentant l'effort de développement et de test de plusieurs projets. Pour évaluer la méthode, des modèles d'analyse de régression logistique univariée et multivariée sont produits. La démarche expérimentale est répliquée pour deux méthodes existantes dans la littérature, soit les méthodes UCP (Use Case Points) et OCP (Objective Class Points). Les résultats de ces analyses sont donnés et discutés à la fin du mémoire. Ces résultats démontrent le potentiel de la nouvelle méthode, basée sur les Métriques de Cas d'Utilisation (MCU), à effectuer des prévisions plus précises que les autres méthodes présentées.

USING USE CASES TO PREDICT DEVELOPMENT AND TESTING EFFORT: AN EXPLORATORY STUDY

The software development lifecycle is composed of many phases starting at the analysis phase to the testing and debugging phase. The last phases of the development lifecycle, the development and testing phases, are the most expensive in terms of cost, effort and resources. It therefore becomes important to be able to make predictions, as soon as possible, as to the efforts related to these phases so that resources and activities can be optimally managed and allocated. This project aims at presenting and exploring a new method for predicting development and testing efforts from use cases. The method uses five metrics that can be collected early in the development lifecycle and compares them to metrics representing the real development and testing efforts of several real projects. In order to evaluate this method, univariate and multivariate logistic regression analysis models are produced. The experimental approach is replicated using two well-known methods in the literature: the UCP (Use Case Points) and OCP (Objective Class Points) methods. Results are shown and discussed at the end of this thesis. Results demonstrate that the new method has the potential to give more accurate predictions than the existing methods presented.

Chapitre 1 - Introduction

Le cycle de développement d'un logiciel est un processus long et complexe, avec beaucoup d'étapes. Les différentes phases du cycle de développement ont toutes leur importance et leur coût associé. La phase d'analyse produit les documents de spécification du système. La phase de conception permet de construire et schématiser le logiciel par rapport aux exigences recueillies dans la phase d'analyse. La phase de développement est celle où les fonctionnalités du système sont codées. Finalement, une série de vérifications et validations sont effectuées durant la phase de test. La capacité de faire des prévisions sur les dernières phases (développement et test) est un facteur des plus importants dans la gestion d'un projet de développement [17, 33].

Les phases d'analyse et de développement sont généralement considérées comme des phases coûteuses en termes de temps et de ressources. Avec la création de logiciels de grande envergure, il devient d'autant plus important de pouvoir estimer cet effort le plus rapidement possible pour pouvoir planifier correctement les activités et l'attribution des ressources d'un projet [6, 15, 41]. Les mêmes objectifs sont d'autant plus valables pour les tests. Il s'agit, dans ce projet, de mettre l'accent sur la prédiction des efforts relatifs aux phases de développement et de test. L'effort de la phase de développement peut se mesurer en termes de taille du code source qui doit être écrit pour compléter les exigences décrites durant la phase de conception. De façon similaire, pour la phase des tests, on considère la taille des suites de tests automatisés permettant de tester la totalité des fonctionnalités codées [14, 15].

En principe, une estimation précise des efforts nécessaires pour compléter un projet permet de faciliter la prise de décision dans le but de produire un logiciel de la plus grande qualité possible. Elle permet aussi de suivre plus facilement l'avancement du projet, de faire des ajustements plus tôt lorsqu'un imprévu survient et de localiser les parties les plus critiques d'un système qui demanderont plus d'attention (en termes d'assurance qualité en particulier) et de ressources. Sous cet angle, plus on peut obtenir l'information tôt dans le cycle de développement, plus les avantages seront grands. Certaines parties du développement logiciel sont, cependant, complexes à estimer dans les premières phases d'un projet. L'effort de test est une de ces parties. Un bon nombre de métriques ont été suggérées pour mesurer la testabilité (sous différentes perspectives) d'un système [2, 9, 15, 18, 38, 39].

Le processus d'assurance qualité d'un logiciel est composé de plusieurs aspects. Le test logiciel est un des aspects les plus importants du processus. Il s'agit de développer une suite de tests qui permette de vérifier le fonctionnement correct des fonctionnalités du logiciel. Il n'est, cependant, pas tâche facile d'estimer précisément l'effort requis pour le développement de ces tests, car beaucoup de facteurs entrent en compte dans l'équation, incluant les techniques et outils utilisés, les caractéristiques du logiciel, les critères de test, ainsi que le facteur humain [25].

Plusieurs études [21, 25, 32-34] ont été publiées sur le sujet de l'estimation de l'effort de développement d'un logiciel. Ces études proposent un nombre de métriques et formules permettant d'obtenir, avec des degrés variables de précision, une idée du temps que prendra le processus complet du développement d'un logiciel. L'estimation de l'effort de test est, cependant, rarement traitée dans ces articles. Il s'agit d'un aspect généralement inclus dans l'estimation du temps (effort) de développement,

pensant que plus l'effort de développement d'un projet est élevé, plus son effort de test le sera.

Quoi qu'il soit effectivement possible d'obtenir un ordre de grandeur de l'effort de test de cette façon, les facteurs qui influencent l'effort de test ne sont pas nécessairement les mêmes que ceux qui influencent l'effort de développement. Il serait intéressant d'explorer la possibilité d'établir une méthode pour estimer précisément l'effort de test d'un logiciel, du moins classer par ordre de priorité les différentes parties d'un logiciel.

Les études qui traitent de cette possibilité le font généralement d'un point de vue de prédiction des tests à partir du code source. C'est-à-dire, une fois le logiciel développé, mais avant qu'il ne soit testé. On peut facilement estimer l'effort de test qui sera nécessaire pour assurer la qualité du produit. Quoi qu'intéressante, cette information arrive relativement tard dans le cycle de développement d'un projet, sans mentionner que certaines méthodologies préconisent le développement des tests en parallèle à la programmation du logiciel. Il serait donc intéressant de pouvoir obtenir l'information plus tôt dans le cycle de développement, idéalement suite à la phase d'analyse du projet.

Les estimations d'efforts, que ce soit de développement ou de test, utilisent pour la plus part un nombre de métriques extraites à partir de livrables du projet. On cherchera des livrables qui sont produits le plus tôt possible dans le cycle de développement. La phase d'analyse d'un projet produit des livrables qui correspondent aux besoins. L'information disponible sera donc sous forme de spécifications fonctionnelles, soit les cas d'utilisation et leurs diagrammes de séquence. Le

diagramme de classes contient aussi beaucoup d'informations qui sont utiles à l'estimation de l'effort, mais il s'agit d'un document produit durant la phase de conception, soit plus tard dans le cycle de développement. Il est, néanmoins, intéressant de considérer l'information qu'il contient pour faire des prévisions plus tardives.

L'objectif premier de ce projet est de démontrer la possibilité d'obtenir une estimation de l'effort de test à partir d'informations recueillies dans les premières phases du cycle de développement d'un logiciel. L'objectif secondaire est de comparer la précision de la méthode présentée aux méthodes existantes permettant de prédire l'effort de test et de développement.

La première étape dans cette démarche est de déterminer les documents à utiliser et les métriques à extraire. Parmi les documents produits lors de l'étape d'analyse d'un projet, le diagramme des cas d'utilisation, le diagramme de séquences et le diagramme de classes d'analyse ressortent comme ayant le plus d'information. Le diagramme de séquences étant une extension des cas d'utilisation (chaque diagramme représentant le déroulement fonctionnel d'un cas). Nous nous référerons à ces modèles comme étant les informations des « cas d'utilisation ».

L'information provenant des cas d'utilisation est variée [24, 27]. Les cas d'utilisation nous donnent les acteurs en jeu, les fonctionnalités requises, les transactions effectuées, les entités présentes, les différents scénarios, etc. Nous avons défini plusieurs métriques permettant de capturer l'information procurée dans les cas d'utilisations. Dans le but d'effectuer les analyses, il est nécessaire d'obtenir des études

de cas à partir desquelles peuvent être extraites ces métriques. Pour obtenir des analyses précises, un nombre d'études de cas nous permettant d'obtenir un échantillonnage de cas d'utilisation et de classes raisonnable est nécessaire. Les études de cas choisies varient en nature et domaines d'affaire. Le but était surtout d'obtenir un échantillonnage de données plus large. Dans le but d'extraire des métriques, on utilise ces études de cas qui contiennent des documents d'analyse et des tests unitaires.

Avec les études et les métriques en main, il est possible d'effectuer une série d'analyses statistiques pour démontrer qu'il existe un lien entre les métriques d'analyse et les métriques d'effort de test. À la fin de cette étape, on peut constater la validité de la première hypothèse, à savoir s'il est possible de prédire l'effort de test à partir des documents d'analyse.

La suite nécessite de faire une comparaison avec les méthodes existantes pour prédire l'effort de test et de développement. Pour ce faire, une méthode de chaque type a été choisie. Parmi les méthodes présentement acceptées dans la littérature pour la prédiction de la testabilité d'un logiciel, la méthode UCP (Use Case Points) [25] a été choisie parce qu'elle utilise les mêmes documents pour effectuer ses prédictions en s'y prenant différemment (offre donc une bonne base de comparaison). Plusieurs méthodes de prédiction de l'effort de développement ont été considérées et la méthode OCP (Objective Case Points) [26] est ressortie comme ayant un bon potentiel pour prédire l'effort de test avec peu d'altérations.

En extrayant les métriques des mêmes études de cas et en effectuant les mêmes analyses statistiques que pour la première partie, on obtient des résultats pour ces deux méthodes qui peuvent être comparés aux résultats obtenus dans la première partie. Avec ces informations, on peut arriver à une conclusion pour déterminer si la méthode présentée dans ce projet donne des résultats comparables aux méthodes existantes.

Finalement, on effectue l'expérience inverse. La méthode introduite dans ce mémoire est utilisée pour prédire l'effort de développement. Il s'agit également de la comparer aux deux autres méthodes. Avec cette expérimentation, on tente d'obtenir une confirmation de l'efficacité de chaque méthode pour les deux types de prévision.

Le reste du mémoire est organisé comme suit : Dans la prochaine section, on procède à une revue de l'état du domaine actuel en analysant la littérature publiée sur le sujet. La section suivante décrit les méthodes et études de cas qui sont utilisées dans le projet. On traite ensuite, la prédiction des efforts de test et des analyses effectuées pour comparer les différentes méthodes. La section suivante est similaire, hormis orientée vers la prédiction de la taille du code source. Chacune de ces deux dernières sections se termine par une analyse des résultats et une discussion sur les conclusions à retenir. Les sections finales comprennent une discussion de tous les résultats obtenus dans les deux parties précédentes et une conclusion quant à ce que l'on peut retenir des résultats obtenus.

Chapitre 2 - Revue de la littérature

La prévision d'effort est un sujet qui a été exploré sous beaucoup d'angles et par de nombreuses études. La majorité de la recherche sur le sujet est effectuée sur l'estimation de l'effort de développement (sous différentes perspectives). Parmi les études publiées, une grande partie fait usage des cas d'utilisations pour effectuer les prévisions d'effort de développement [21, 25, 32-35] et de test [4-6, 14, 15, 22, 35-37, 42].

Dans le cas de l'estimation de l'effort de développement, un bon nombre de méthodes [21, 25, 32-35] ont été mises au point pour effectuer les prévisions à partir des premières phases du processus de développement. La méthode des Functions Points (FP) [1] est une des premières méthodes à avoir été proposées pour faire l'évaluation de la complexité d'un logiciel. De cette complexité, on peut aussi prédire d'une certaine manière la taille du logiciel, et donc l'effort de développement requis. Certaines des étapes de la méthode des Functions Points sont cependant biaisées [17]. Elles sont dépendantes de décisions prises par la personne qui les applique. Par ailleurs, Les données sur lesquelles se base la méthode sont parfois aussi difficiles à obtenir; elles sont dépendantes de l'historique des projets de l'entreprise [17].

Les cas d'utilisations sont largement utilisés comme point de départ par plusieurs méthodes d'estimation. Karner [25] introduit une méthode inspirée des Function Points pour faire l'estimation de l'effort de développement en utilisant les cas d'utilisation. Mohagheghi et al. [30] effectuent des modifications sur la méthode UCP (Use Case Points) pour l'adapter au développement incrémental. Ochodek et al. [32]

analysent la même méthode pour en déterminer la précision et recherchent des simplifications possibles.

Zhou et al. [43] effectuent une analyse de plusieurs méthodes existantes et comparent leur efficacité sur une série d'études de cas. Les méthodes présentées varient en termes de biais et d'information requise pour fonctionner. Certaines font usage de documents produits plus tard dans le cycle de développement, tels que le diagramme de classes détaillé et les diagrammes de séquence (phase de conception). Une des méthodes abordées dans l'article de Zhou et al., la méthode Objective Class Points (OCP) [26], qui est utilisée comme comparatif dans notre projet.

Robiolo et Oscoro [33] explorent la possibilité de simplifier la méthode des Function Points pour tenter d'estimer la taille et l'effort de développement. Ils découvrent, entre autres, que les métriques « *nombre de transactions* » et « *nombre d'entités* » dans un diagramme de cas d'utilisations sont efficaces pour effectuer ce genre de prévision. La métrique « *nombre de transactions* » est réutilisée dans ce document dans deux des méthodes évaluées, dont celle proposée dans le cadre de cette recherche.

L'estimation de la testabilité des logiciels, quant à elle, a vu un intérêt grandissant dans les dernières années. Différentes méthodes de prédiction de l'effort de test ont été développées pour les systèmes orientés objet [4-6, 14, 18, 31, 35].

Bruntink et Van Deursen [14, 15] étudient des projets open-source en Java et explorent la possibilité d'une relation entre les métriques de conception et les classes de test

JUnit. Ils définissent la testabilité d'un logiciel par le *nombre de lignes de code* devant être écrites pour développer les tests unitaires et le *nombre de déclarations* de type « *assert* » dans ces tests. L'étude démontre une corrélation entre les métriques de conception et la testabilité telle que définie. La notion de testabilité utilisée par cette étude est réutilisée dans le présent projet selon le même concept.

Singh et al. [35] développent une méthode d'estimation de l'effort de test à partir des métriques orientées objet. Leur méthode utilise les réseaux de neurones et définit l'effort de test comme étant le *nombre de lignes de code ajoutées ou changées* durant le cycle de vie suite à une défaillance logicielle. La conclusion tirée est que la performance de leur modèle est largement dépendante des données utilisées. Singh et Saha [37] explorent ensuite l'estimation de la testabilité des packages du projet Eclipse. Un lien est établi entre les métriques extraites du code source et les métriques relatives aux suites de test.

Nageswaran [31] présente la possibilité d'utiliser la méthode UCP pour effectuer la prédiction de l'effort de test. L'article explore aussi d'autres méthodes d'estimation d'effort de test utilisées dans d'autres organismes et les compare avec la méthode proposée. Dans le présent projet, la méthode UCP est aussi utilisée comme comparatif aux autres méthodes.

Badri et al. [4] explorent la relation entre les métriques de manque de cohésion et la testabilité des systèmes orientés objet. Ils explorent aussi plus tard [5] la capacité des métriques de manque de cohésion à prédire la testabilité des classes en faisant usage des méthodes de régression logistique. Dans leur approche, comme dans [14, 15], la

testabilité est aussi mesurée par le nombre de lignes de code et le nombre de déclaration de type « assert ».

Badri et Toure [6] explorent plus en détail la capacité de prédire la testabilité des classes à partir des métriques orientées objet (OO). Les résultats de leur recherche démontrent, entre autres, que les modèles de régression multivariée peuvent être utilisés pour estimer avec précision l'effort de test unitaire des classes.

La majorité des travaux de recherche effectués dans le domaine abordent l'estimation de l'effort de test à partir du code source. Il s'agit d'un point relativement avancé dans la progression du cycle de développement et un peu tard pour effectuer des prévisions sur l'effort de test (l'étape suivante dans le cycle de développement). Certains auteurs ont commencé à évaluer la possibilité de prédire très tôt les efforts de développement et de test [4-6, 33, 34]. Dans l'optique que les prévisions sont plus efficaces quand elles sont obtenues plus tôt, il y a donc tout avantage à explorer la possibilité de prédire l'effort de test dans les premières phases du cycle de développement.

Plusieurs approches [2, 9, 18, 38, 39] ont été proposées pour résoudre le problème de prévision de l'effort de test. Ces approches utilisent les données recueillies dans la documentation UML produite durant l'étape d'analyse fonctionnelle du cycle de développement. Beaudry et al. [9] abordent l'amélioration de la testabilité des logiciels orientés objet. Ils mentionnent l'utilisation des modèles de conception (*design patterns*) pour réduire les faiblesses relatives à la testabilité d'un système. Leur approche tente de détecter les problèmes du système à ce niveau à partir des diagrammes de classes UML.

Cependant, peu d'approches font usage des cas d'utilisation pour faire l'estimation de l'effort de test. Un nombre d'études les ont cependant utilisés pour estimer l'effort de développement [2, 18, 38, 39]. Yi et al. [38] présentent une approche pour l'estimation de la taille des suites de test basée sur les points de vérification des cas d'utilisation. Une relation linéaire est découverte entre les points de vérification et le nombre de cas de test. Zhu Xiaochun et al. [39] présentent une étude empirique sur l'estimation rapide de l'effort d'exécution des tests basée sur la prévision du nombre de cas de test et de leur complexité.

Almeida et al. [2] proposent une méthode pour estimer l'effort de test basée sur l'information obtenue des cas d'utilisation, soit les acteurs et le nombre de scénarios. Ils tiennent aussi compte de plusieurs facteurs techniques et environnementaux pour les tests. La méthode effectue une séparation des scénarios de cas d'utilisation standards des scénarios non-standards (ex : les cas d'erreurs), partant du principe qu'ils ont des influences différentes sur l'effort de test.

Chaudhary et Yadav [18] proposent une approche pour l'estimation de la taille des suites de test et de l'effort de développement de ces derniers en utilisant une approche appelée « test case points » qui fonctionne en effectuant une analyse des facteurs clés influençant la complexité de l'exécution des tests.

Chapitre 3 - Cadre global de l'étude

Dans cette section, nous présentons les objectifs et éléments de base du projet. Nous traitons d'abord des différentes métriques utilisées durant les différentes analyses, ainsi que des méthodes existantes qui seront utilisées à des fins comparatives. Les études de cas utilisées et la méthodologie de collecte des métriques sont ensuite présentées, suivies par les sections portant sur les analyses de prédiction.

3.1 Métriques des suites de test

Dans le développement d'un logiciel, un test peut être définit de plusieurs façons différentes : manuel ou automatique. Pour le test manuel, l'utilisateur pilote effectuera une série d'opérations pour en vérifier le résultat. Quant au test automatisé, la tâche sera effectuée par un ordinateur qui disposera d'une batterie de tests à exécuter. Les tests peuvent être de type unitaire ou d'intégration. Certains types de tests se prêtent mieux aux tests manuels (tests d'intégration), alors que d'autres peuvent être semi-automatisés à l'aide d'outils, tels que JUnit[1] (tests unitaires).

Pour effectuer la prédiction des efforts de test, nous avons considéré une catégorie de tests qui représente, à notre point de vue, une bonne partie de l'effort global nécessaire pour tester un système. Il s'agit des tests unitaires automatisés. Les tests unitaires automatisés ont un concept de couverture de code, représentant le pourcentage du code testé (du moins couvert) par la suite de tests. On considère que l'effort nécessaire pour développer une suite de tests unitaires couvrant une grande

[1] http://junit.org/

majorité du code d'un logiciel est un bon indicateur de l'effort global de test. Pour mesurer cet effort, nous utilisons deux métriques [14, 15].

Le *nombre de lignes de code de test* représente la taille totale de la suite de tests. Il s'agit de la façon la plus simple de mesurer l'effort dépensé par un développeur pour tester les classes à l'aide de tests unitaires semi-automatisés. Le total des lignes n'inclut que les classes et méthodes utilisées dans les tests d'un cas d'utilisation donné pour tous ses scénarios. La métrique est également facile à extraire : il suffit d'identifier les méthodes de test permettant de tester le fonctionnement du cas d'utilisation comptabilisé et de noter le total des lignes de code de ses méthodes. On ajoute aussi les lignes de code de la classe (par exemple, ses attributs et constructeurs) lorsque ces dernières sont nécessaires à l'exécution des méthodes identifiées.

La deuxième métrique est un peu plus complexe. Un test unitaire est généralement composé de trois parties : l'initialisation, l'exécution et la vérification (« Arrange, Act, Assert »). La partie vérification sert à s'assurer que les données sont dans un état voulu (attendu) après avoir exécuté les opérations nécessaires. Pour pouvoir faire cette vérification, les librairies de tests fournissent des opérations d'assertion. Dans le cas des études de cas considérées dans notre projet, la librairie JUnit est utilisée pour écrire les tests unitaires. Les opérations d'assertion de JUnit sont simplement nommées « Assert. » Le nombre d'assertions permet d'avoir de l'information quant à la complexité d'un test unitaire (le nombre de résultats à valider).

Les métriques de test sont relativement faciles à extraire à partir d'une étude de cas. Le seul préalable, dans l'optique de ce projet, est que l'étude doit posséder une suite de tests couvrant la grande majorité (nous avons fixé 90% comme étant une couverture

raisonnable) du code d'intérêt (i.e. le code développé pour répondre aux cas d'utilisations étudiés). Une fois cette exigence remplie, il faut associer chaque test à un cas d'utilisation. Cette partie est assez systématique pour ne pas avoir de biais par rapport à la collecte de données; un test est associé à un cas s'il teste le bon fonctionnement d'une fonctionnalité associée au cas. Pour les besoins de ce projet, un test peut être associé à plus d'un cas d'utilisation. Comme chaque cas d'utilisation est ici considéré comme une entité indépendante, cela ne cause pas de problème.

Avec les tests associés aux cas d'utilisation, on peut extraire le nombre de lignes de code de test et le nombre d'assertions, et les classer par cas d'utilisation. Avec cette information, il sera possible d'effectuer des analyses statistiques pour comparer les métriques de cas d'utilisation relatives aux différentes méthodes explorées avec les métriques de test.

3.2 Métriques de cas d'utilisation (MCU)

Un cas d'utilisation se définit comme étant une série de transactions effectuées entre un acteur et le système, ayant une collection de scénarios possibles résultant en un succès ou en un échec du cas [24, 27]. Le modèle de cas d'utilisation est un outil important pour définir la portée d'un système et lister les fonctionnalités que ce dernier doit implémenter pour satisfaire aux exigences du client. Le schéma typique d'un cas d'utilisation est représenté par un acteur envoyant un message au système, qui délègue ce message à des sous-systèmes jusqu'à complétion de la tâche, suite à quoi un message de retour est renvoyé à l'acteur. Certains cas peuvent nécessiter

plusieurs opérations de la part de l'acteur, alors que d'autres vont offrir plusieurs messages de retour possibles. Un cas peut aussi avoir une série de scénarios, représentant chacun un chemin du début du cas jusqu'à sa complétion. Dans certains cas, un seul scénario se termine en succès alors que tous les autres donnent des erreurs (cas exceptionnels).

Dans le but de prédire l'effort de test, nous devons extraire des métriques des cas d'utilisation. Voici la liste des métriques qui ont été considérées :

Nombre de classes impliquées (NCI) : Le nombre de classes (ou entités du côté système) impliquées dans la réalisation du cas d'utilisation (participant au cas). Cette information est disponible dans le diagramme de séquences d'un cas d'utilisation. À noter que seules les classes testables unitairement ont été considérées. Les classes ne contenant pas de logique de métier, comme les classes gérant l'interface graphique d'un logiciel, ne sont pas considérées « testables » pour les besoins de ce projet.

Nombre d'opérations externes (NOE) : Cette métrique représente le nombre d'opérations disponibles à l'interface d'un cas d'utilisation. La complétion du cas nécessitera généralement l'appel de toutes les opérations externes de ce dernier. Pour faire un parallèle avec la phase de conception, les opérations externes pourraient être regroupées dans une classe Controlleur (ou Façade).

Nombre de méthodes impliquées (NMI) : Le nombre de message différents utilisés dans le diagramme de séquences. Un message représente l'appel d'une méthode dans la phase de conception. Les messages permettent de comprendre comment les objets

interagissent entre eux. Cette métrique inclut la totalité des méthodes appelées dans la réalisation du cas, incluant tous ses scénarios.

Nombre de scénarios (NS) : Un scénario est définit comme un chemin valide pour compléter un cas d'utilisation. Certains cas d'utilisations n'ont qu'un seul scénario, alors que d'autres vont en avoir une multitude. Dépendant du cas, un scénario peut résulter en une erreur retournée à l'acteur du scénario.

Nombre de transactions (NT) : Cette métrique compte le nombre de transactions présentes dans un cas d'utilisation. Une transaction est définie comme étant un échange de messages entre l'acteur et le système, ou dans certains cas, entre deux sous-systèmes. En réalité, deux métriques ont été utilisées pour mesurer le nombre de transactions, l'une incluant toutes les transactions du système, y compris les transactions internes, et une autre n'incluant que les transactions où l'acteur participe (nommée Nombre de transactions de haut-niveau (NTHN)).

L'ensemble des métriques a évolué avec le projet, avec certaines métriques retirées et d'autres rajoutées après certaines étapes. La métrique NMI, constatée comme étant peu influente dans les analyses statistiques (corrélation faible), a été retirée du modèle finalement. Les métriques NT et NTHN ont été considérées plus tard dans le projet, avec l'inclusion de la méthode UCP, qui fait aussi usage de ces métriques [31].

3.3 Métriques de classes

Le diagramme de classes est une autre façon de découper les spécifications d'un système pour le voir sous un ensemble de classes. Le diagramme fournit l'information sur les classes contenues dans le système, leurs attributs et opérations ainsi que les liens qu'elles ont entre elles. L'information contenue dans le diagramme peut être utilisée pour effectuer une estimation de l'effort des étapes suivantes dans le cycle de développement. Le diagramme est parfois conçu en plusieurs itérations, chacune ajoutant de l'information jusqu'à l'obtention du diagramme de classes détaillé. Le diagramme de classes est typiquement produit durant la phase de conception, et non à l'étape d'analyse. Il reste tout de même intéressant de considérer les métriques que l'on peut extraire à partir de ce modèle. Voici la liste des métriques considérées :

Nombre d'attributs (NA) : Le nombre d'attributs contenus dans la classe. La portée de l'attribut n'a pas d'incidence sur cette métrique. Les accesseurs et mutateurs d'un attribut comptent comme faisant partie de l'attribut lui-même (i.e. un attribut, un accesseur et un mutateur comptent au total pour un seul attribut).

Nombre de méthodes publiques (NMP) : Le nombre de méthodes publiques, accessibles par des objets externes à la classe. Les méthodes publiques sont généralement dans les premiers détails inclus dans le diagramme de classes (premières itérations). Il est donc intéressant de voir si cette métrique peut être utilisée à des fins de prévision.

Nombre de méthodes (NM) : Le nombre de méthodes est la somme des méthodes publiques et des méthodes non-publiques (non accessibles de l'extérieur de la classe).

Le nombre de méthodes non-publiques est aussi une métrique qui a été considérée mais n'a pas été retenue dans les premières phases du projet. Le nombre de méthodes au total devient, cependant, disponible plus tard dans les itérations du diagramme de classes, rendant cette métrique moins intéressante à utiliser.

Couplage efférent (CE) : Le couplage représente les liens entre les classes. Un couplage efférent est une référence qu'une classe a vers une autre classe (e.g., si la classe A fait référence à la classe B, la classe A à un couplage efférent). Cette métrique, disponible dans les premières itérations du diagramme de classes, permet d'évaluer les dépendances entre les différentes composantes du système.

Niveau d'héritage (NH) : Le niveau d'héritage représente le nombre de classes « Ancêtre » dans la hiérarchie d'une classe. Cette métrique sert à mesurer la complexité de la hiérarchie de la classe. Comme le couplage, elle est généralement disponible tôt dans les premières itérations du diagramme de classes.

3.4 Méthodes existantes

Dans le but d'évaluer la précision et la pertinence de la méthode de prévision proposée dans ce projet, il nous faut la comparer aux méthodes déjà existantes et reconnues dans le domaine. Des recherches ont été effectuées pour trouver des méthodes avec une prémisse et un but similaire, c'est à dire tentant de prédire l'effort de test à partir des documents d'analyse. La méthode UCP [25] est ressortie comme la utilisée dans ce contexte. De plus, cette méthode s'applique également sur les métriques de cas d'utilisation, la rendant encore plus intéressante comme point de comparaison.

Pour l'autre méthode, une autre approche a été considérée. En 2012, Zhou et al. [43] publient un article qui effectue une comparaison de plusieurs méthodes dont le but est la prédiction de l'effort de développement à partir de métriques d'analyse et de conception. Les méthodes analysées sont les métriques de classes (semblables à celles présentée plus haut), Predictive Object Points (POP) [29], Object-Oriented Project Size (OOPS) [12], Fast&&Serious Class Points (FSCP) [16], Objective Class Points (OCP) [26] et Object-Oriented Function Points (OOFP) [3]. Nous avons examiné si nous pouvions faire usage d'une de ces méthodes pour la comparer à l'approche que nous préconisons.

Plusieurs des méthodes présentées utilisent des métriques disponibles seulement durant la phase de conception, et d'autres sont très subjectives (OOPS entre autres). Les méthodes OCP et OOFP ont été retenues parce qu'elles différaient grandement de la méthode utilisée dans ce projet et qu'elles n'avaient pas les problèmes décrits plus haut. Seule OCP est présentée dans ce projet.

3.4.1 La méthode UCP (Use Case Points)

UCP a été présentée la première fois par Karner en 1993 [25]. Il s'agit d'une méthode simple et rapide d'utilisation pour effectuer l'estimation de l'effort de développement et permet aussi d'effectuer l'estimation de l'effort de test [31] et l'effort de maintenance [19]. La méthode a été utilisée est utilisée dans un bon nombre de projets industriels pour prédire l'effort de développement [10, 11, 17, 19, 30, 32]. Elle permet

d'effectuer l'estimation de l'effort en personne/heures à partir des documents de cas d'utilisation.

La méthode UCP tente de calculer la métrique AUCP (Adjusted UCP). C'est à partir de cette métrique qu'il devient possible de calculer l'effort en personne/heures. Pour les besoins de ce projet, nous allons nous concentrer principalement sur la métrique AUCP, car c'est elle qui sera la base de comparaison avec les autres méthodes. Le calcul d'AUCP tient compte de trois facteurs, soit la complexité des acteurs, la complexité des cas d'utilisation et plusieurs facteurs techniques et environnementaux du projet. Les deux métriques de complexité sont assez explicites et concrètes dans leur calcul, alors que les facteurs sont plutôt subjectifs [17, 32]. Une explication de chaque métrique est donnée dans ce qui suit :

Complexité des acteurs : Cette métrique est déterminée à partir de la nature de l'acteur du cas d'utilisation. Il existe trois possibilités : l'acteur peut être un humain utilisant le système à l'aide d'une interface graphique, il peut être un programme externe appelant le système à l'aide de lignes de commandes ou d'appels de bibliothèque ou l'acteur peut être le système lui-même, comme dans le cas d'une action automatisée par le système (e.g. calcul d'un relevé mensuel). Chaque type d'acteur est associé à une valeur selon le Tableau 1.

Tableau 1 - Complexité des acteurs selon leur type

Type d'acteur	Valeur
Humain	Complexe (3)
Programme externe	Moyen (2)
Système	Simple (1)

Il est à noter que comme plusieurs des études de cas de ce projet sont des bibliothèques de fonctions, la nature des acteurs de chaque cas n'est parfois pas très claire. Par exemple, dans l'étude Joda-Time, nous avons un cas « Afficher une date avec une mise en forme ». L'acteur de ce cas pourrait aussi bien être un programme externe qu'un humain utilisant une interface permettant d'entrer un format manuel dans un champ. Dans ce genre de cas, la valeur la plus élevée est retenue (dans le cas présenté, on garde 3, la valeur associée à un acteur humain).

Complexité du cas : La complexité du cas d'utilisation est la métrique la plus variable et la plus importante du calcul. On attribue une valeur basée sur le nombre de transactions (voir métriques de cas d'utilisation) du cas d'utilisation. Le Tableau 2 liste les valeurs possibles.

Tableau 2 - Complexité d'un cas d'utilisation par rapport au nombre de transactions

Nombre de transactions	Valeur
Moins de 4	Simple (5)
Entre 4 et 7	Moyen (10)
Plus de 7	Complexe (15)

Facteurs environnementaux : Les facteurs environnementaux sont liés à l'environnement de développement du projet. Il s'agit de facteurs comme l'expérience des programmeurs avec le produit, la présence d'un senior dans l'équipe, etc. Comme cette information n'est pas disponible pour les études de cas de ce projet, ces facteurs seront simplement ignorés pour les besoins de l'étude. Par soucis de complétion, le Tableau 3 donne la liste des facteurs environnementaux et leur poids.

Tableau 3 - Facteurs environnementaux

Facteur	Poids
Familiarité de l'équipe avec le processus de développement utilisé	1,5
Expérience de l'équipe avec l'application	0,5
Expérience de l'équipe en développement orienté-objet	1,0
Compétence de l'analyste principal	0,5
Motivation de l'équipe	1,0
Stabilité des spécifications	2,0
Membres de l'équipe à temps partiel	-1,0
Langage de programmation compliqué	-1,0

Facteurs techniques : Les facteurs techniques sont basés sur les attentes et la nature du projet. Comme pour les facteurs environnementaux, une liste de facteurs est attribuée au projet. Le Tableau 4 donne la liste des facteurs techniques.

Tableau 4 - Facteurs techniques

Facteur	Poids
Système distribué	2,0
Temps de réponse / objectifs de performance	1,0
Efficacité utilisateur	1,0
Processus internes complexes	1,0
Réutilisabilité	1,0
Facile d'installation	0,5
Facile d'utilisation	0,5
Portable vers d'autres systèmes	2,0
Nécessite maintenance	1,0
Processus concurrents / parallèles	1,0
Sécurité	1,0
Bibliothèque de fonctions	1,0
Nécessite formation utilisateur	1,0

Une fois les trois métriques obtenues, il devient possible de calculer AUCP à partir de la formule suivante :

AUCP=(ActeurUCP + CasUCP) * (0.065 * (0.01 * (SommeFacteursTechniques))

3.4.2 La méthode OCP (Objective Case Points)

La méthode OCP est une tentative de retirer les facteurs subjectifs dans les méthodes utilisant les concepts de Class Points [43]. L'idée est de calculer une métrique à partir des informations disponibles dans les premières itérations du diagramme de classes. Cette méthode a été développée dans le but de prédire l'effort de développement, mais nous l'utilisons ici pour voir son efficacité à prédire l'effort de test et comme comparatif avec les deux autres méthodes retenues pour notre étude.

Cette méthode est basée sur le diagramme de classes, contrairement à la méthode UCP qui est basée sur le modèle des cas d'utilisation. La méthode OCP fait usage du nombre de classes, du nombre de méthodes, du nombre d'attributs ainsi que du couplage, ce dernier divisé en catégories (associations, dépendances, généralisations, etc.). La métrique OCP est calculée selon la formule suivante :

OCP = NbClasses + NbCouplage (toutes catégories) + NbMéthodes + NbAttributs.

Comme les autres méthodes présentées utilisent les cas d'utilisation comme base et que les données recueillies des études de cas sont groupées par cas, nous ne pouvons utiliser la méthode OCP telle quelle pour faire une comparaison. C'est pourquoi nous allons créer une multitude de diagramme de classes, soit un par cas d'utilisation. La méthode est simple : on prend chaque classe et méthode impliquées dans un cas d'utilisation et on monte le diagramme de classes résultant (sous-ensemble du diagramme de classes complet), ignorant le reste des classes et méthodes. Cette méthode a le désavantage de parfois causer une duplication des données dans plusieurs cas d'utilisation, mais devrait tout de même nous donner des résultats utilisables à des fins de comparaison.

3.4.3 Études de cas

Pour notre étude, nous avons utilisé six études de cas différentes. Les études de cas sélectionnées sont hétéroclites, appartenant à plusieurs domaines d'affaires différents. Le critère de sélection initial était que les études de cas contiennent les documents d'analyse utilisés pour concevoir le système et que le système contienne une suite de

tests unitaires permettant de le tester avec une couverture d'au moins 90%. Les études de cas devaient aussi contenir un nombre de cas d'utilisation intéressant. Nous visions au un nombre total de cas d'utilisation nous permettant d'effectuer des analyses statistiques significatives, soit au moins 40.

Après recherche, il s'est avéré en effet difficile de trouver des études de cas d'envergure et ayant à la fois les documents d'analyse et les tests unitaires. Le critère d'analyse a donc été éliminé et nous avons procédé plus en nous basons sur le critère relatif à la disponibilité des tests unitaires seulement. Les documents d'analyse ont été produits par rétro ingénierie du code source des études de cas. Pour certaines études, la couverture du code visée (90%) des tests unitaires a été atteinte en codant les tests manquants.

Voici une présentation sommaire des études de cas utilisées :

ATM[2] : Un simulateur de guichet automatique. Le logiciel implémente les opérations de base fournies par un guichet, soit le retrait, le dépôt, le transfert et l'état de compte. L'étude de cas a été adaptée et ses tests unitaires complétés pour répondre aux besoins de notre étude.

NextGen : Il s'agit d'une extension d'une application développée dans le livre d'introduction à l'analyse et au développement orientée-objet et au processus unifié par C. Larman [27]. Des améliorations et des extensions ont été faites au code pour les

[2] http://www.math-cs.gordon.edu/courses/cs211/ATMExample/

besoins de l'étude, tels que l'ajout de fonctionnalités pour la gestion des comptes recevables, des fournisseurs et des employés.

Commons Exec[3] : Une bibliothèque de fonctions Java offrant une gamme de fonctionnalités pour faciliter l'exécution de processus sous une multitude de systèmes d'exploitation différents. La librairie permet de faire l'observation et d'interagir avec les processus une fois en exécution.

Commons Email[4] : Une autre bibliothèque Java utilisée pour envoyer des courriels. La bibliothèque étend les fonctionnalités de l'API Java avec la même utilité et facilite son utilisation.

Commons IO[5] : Aussi une bibliothèque Java, qui facilite et étend les opérations d'entrée-sortie telles que la lecture et l'écriture de fichiers, la gestion de répertoires, la gestion des noms et autres. La librairie est massive et comporte un grand nombre de cas d'utilisation. Seul un sous-ensemble des cas d'utilisation est utilisé, soit les cas reliés à la lecture, à l'écriture et à la comparaison de fichiers. Par ailleurs, la bibliothèque ne respectant pas tout à fait les standards de la programmation orientée-objet, n'a pas été utilisée dans toutes les analyses (seules celles où ce facteur a peu d'incidence).

[3] https://commons.apache.org/proper/commons-exec/
[4] https://commons.apache.org/proper/commons-email/
[5] https://commons.apache.org/proper/commons-io/

Joda Time[6] : La dernière étude de cas est une autre bibliothèque Java, remplaçant la gestion des dates de l'API traditionnel. Joda Time permet la manipulation de dates sous plusieurs calendriers et offre beaucoup plus de fonctionnalités que l'API de base. Encore une fois, vu le grand nombre de cas d'utilisation de cette bibliothèque, un sous-ensemble a été retenu pour éviter que la majorité des cas ne viennent de la même étude. Ce sous-ensemble est tiré du tutoriel d'utilisation de la librairie. Cette étude a été rajoutée tard dans le projet et n'est pas utilisée dans les premières analyses. Elle figure parmi les études de cas utilisées dans l'article de Zhou et al. [43].

Les cas d'utilisation de chacune de ces études de cas sont combinés pour pouvoir obtenir un nombre suffisant de données. Au total, 52 cas d'utilisation ont été assemblés. Les analyses sont effectuées sur un sous-ensemble de ces cas (généralement autour de 40) pour répondre aux besoins spécifiques d'une analyse. L'évolution du projet a fait que certaines études de cas ont dû être retirées et d'autres ajoutées dans certains cas (expliqué plus loin). Certaines analyses sont effectuées sur plus d'un sous-ensemble pour pouvoir effectuer des comparaisons exactes. Le Tableau 5 liste les statistiques descriptives des métriques recueilles à partir des études de cas.

[6] http://www.joda.org/joda-time/

Tableau 5 -Statistiques descriptives

Variables	Observations	Min	Max	Moyenne	Écart-type
NC	53	1	15	3,755	3,285
NOE	53	1	36	2,679	5,050
NMI	53	1	60	6,113	8,797
NS	53	1	32	3,491	4,398
NTHN	53	1	41	6,962	6,670
NT	53	1	41	4,698	5,712
UCP	53	4,380	15,810	9,890	3,899
OCP	38	2	134	20,289	21,918
NLC	46	4	399	59,174	66,125
TLC	41	14	1556	201,561	342,752
TASSERT	41	1	284	21,878	49,253

Chapitre 4 - Prédiction des efforts de test

Les premières analyses de ce projet portent sur l'effort de test. Les méthodes d'analyse statistique sont décrites dans cette section, avec la justification de leur utilisation. Les résultats des analyses et les interprétations correspondantes sont présentés à la fin.

4.1 Étude empirique

4.1.1 Analyses de corrélations

Pour pouvoir repérer s'il existe un lien entre les différentes méthodes de prévision et les métriques de test, une analyse de corrélation est effectuée. Pour chacune des méthodes, les hypothèses sont les suivantes :

- H0 : Il n'existe pas de corrélation significative entre la métrique de cas d'utilisation et la métrique de test.
- H1 : Il existe une corrélation significative entre la métrique de cas d'utilisation et la métrique de test.

Pour cette étude, deux méthodes d'analyse de corrélations sont utilisées, la technique de Pearson et de Spearman. La technique de Pearson est utilisée en statistique pour mesurer la relation entre deux variables linéaires distribuées de façon normale. La technique de Spearman, quant à elle, est un test non-paramétrique utilisé pour mesurer le degré d'association entre deux variables. Contrairement à Pearson, Spearman se base sur les rangs et ne tient pas compte de la distribution des variables.

Pour les deux analyses, le seuil d'importance alpha utilisé est le seuil standard 0.05. L'outil utilisé pour faire ces analyses est XLSTAT [40].

Pour chacune des métriques de chaque méthode (soit chacune des métriques de cas d'utilisation (MCU) et les métriques UCP et OCP), nous avons associé des métriques de test (nombre de lignes de code et nombre d'assertions). La première itération a été faite pour les méthodes MCU et UCP, utilisant les études de cas ATM, NextGen, CommonsExec, CommonsEmail et CommonsIO. Les tableaux 6 et 7 contiennent les résultats de la première itération.

Tableau 6 - Corrélation Pearson de MCU et UCP avec les métriques de test

Métriques	NC	NEO	NIM	NS	NT	NTHN	UCP	UCP HN	TLC	TASSERT
NC	**1**	0,267	**0,892**	0,223	**0,836**	**0,363**	**0,504**	**0,376**	**0,844**	**0,366**
NOE	0,267	**1**	**0,573**	-0,086	**0,519**	**0,736**	**0,440**	**0,523**	**0,557**	0,168
NMI	**0,892**	**0,573**	**1**	0,240	**0,920**	**0,621**	**0,644**	**0,574**	**0,958**	**0,409**
NS	0,223	-0,086	0,240	**1**	**0,556**	**0,609**	**0,738**	**0,711**	**0,310**	**0,328**
NT	**0,836**	**0,519**	**0,920**	**0,556**	**1**	**0,788**	**0,828**	**0,750**	**0,914**	**0,454**
NTHN	**0,363**	**0,736**	**0,621**	**0,609**	**0,788**	**1**	**0,854**	**0,904**	**0,658**	**0,353**
UCP	**0,504**	**0,440**	**0,644**	**0,738**	**0,828**	**0,854**	**1**	**0,894**	**0,665**	**0,427**
UCP HN	**0,376**	**0,523**	**0,574**	**0,711**	**0,750**	**0,904**	**0,894**	**1**	**0,649**	**0,452**
TLC	**0,844**	**0,557**	**0,958**	**0,310**	**0,914**	**0,658**	**0,665**	**0,649**	**1**	**0,560**
TASSERT	**0,366**	0,168	**0,409**	**0,328**	**0,454**	**0,353**	**0,427**	**0,452**	**0,560**	**1**

Tableau 7 - Corrélation Spearman de MCU et UCP avec les métriques de test

Métriques	NC	NEO	NIM	NS	NT	NTHN	UCP	UCP HN	TLC	TASSERT
NC	**1**	**0,541**	**0,639**	0,012	**0,458**	0,249	0,302	0,171	**0,329**	0,234
NOE	**0,541**	**1**	**0,535**	-0,184	**0,404**	**0,436**	0,257	0,271	**0,334**	0,269
NMI	**0,639**	**0,535**	**1**	0,224	**0,675**	**0,528**	**0,593**	**0,480**	**0,652**	**0,556**
NS	0,012	-0,184	0,224	**1**	**0,709**	**0,727**	**0,792**	**0,758**	**0,486**	**0,706**
NT	**0,458**	**0,404**	**0,675**	**0,709**	**1**	**0,919**	**0,936**	**0,835**	**0,764**	**0,859**
NTHN	0,249	**0,436**	**0,528**	**0,727**	**0,919**	**1**	**0,903**	**0,901**	**0,719**	**0,858**
UCP	0,302	0,257	**0,593**	**0,792**	**0,936**	**0,903**	**1**	**0,926**	**0,775**	**0,888**
UCP HN	0,171	0,271	0	0,758	0,835	0,901	0,926	**1**	**0,706**	**0,844**
TLC	**0,329**	**0,334**	**0,652**	**0,486**	**0,764**	**0,719**	**0,775**	**0,706**	**1**	**0,872**
TASSERT	0,234	0,269	**0,556**	**0,706**	**0,859**	**0,858**	**0,888**	**0,844**	**0,872**	**1**

Les valeurs en gras dans les deux dernières colonnes sont celles qui ont une p-value (probabilité qu'elle soit non-significative) inférieure à alpha (0.05 dans notre cas). Ces valeurs sont celles qui ont une corrélation significative.

Sur les mêmes projets, une itération est réalisée pour les métriques de classe. Les tableaux 8 et 9 montrent les résultats.

Tableau 8 - Corrélation Pearson des métriques de classe avec les métriques de test

Métriques	C	NH	NA	NMP	NM	TLC	TASSERT
C	**1**	-0,236	0,172	-0,090	-0,025	**0,380**	**0,390**
NH	-0,236	**1**	-0,226	-0,156	-0,142	0,052	-0,111
NA	0,172	-0,226	**1**	**0,535**	**0,544**	**0,467**	**0,378**
NMP	-0,090	-0,156	**0,535**	**1**	**0,941**	**0,418**	**0,408**
NM	-0,025	-0,142	**0,544**	**0,941**	0	**0,456**	**0,419**
TLC	**0,380**	0,052	**0,467**	**0,418**	**0,456**	**1**	**0,891**
TASSERT	**0,390**	-0,111	**0,378**	**0,408**	**0,419**	**0,891**	**1**

Tableau 9 - Corrélation Spearman des métriques de classe avec les métriques de test

Métriques	C	NH	NA	NMP	NM	TLC	TASSERT
C	**1**	-0,201	0,102	-0,040	0,030	-0,065	-0,052
NH	-0,201	**1**	-0,232	-0,275	-0,297	-0,145	-0,278
NA	0,102	-0,232	**1**	**0,622**	**0,606**	0,290	0,210
NMP	-0,040	-0,275	**0,622**	**1**	**0,965**	**0,404**	**0,498**
NM	0,030	-0,297	**0,606**	**0,965**	0,030	**0,466**	**0,543**
TLC	-0,065	-0,145	0,290	**0,404**	-0,065	**1**	**0,803**
TASSERT	-0,052	-0,278	0,210	**0,498**	-0,052	**0,803**	**1**

La dernière itération est celle effectuée entre MCU et OCP. L'étude CommonsIO est remplacée par Joda-Time en raison d'OCP qui fonctionne sur des projets orientés objet. CommonsIO utilise une architecture proche de la programmation procédurale, ce qui biaise un peu les résultats de la méthode OCP qui est basée sur une architecture objet. Les tableaux 10 et 11 contiennent les résultats.

Tableau 10 - Corrélation Pearson de MCU et OCP avec les métriques de test

Métri-ques	NC	NOE	NMI	NS	NT	NTHN	OCP	TLC	TASSERT
NC	**1**	**0,495**	**0,726**	**0,640**	**0,722**	**0,580**	**0,816**	0,130	**0,613**
NOE	**0,495**	**1**	**0,926**	**0,867**	**0,860**	**0,971**	**0,822**	**0,746**	**0,918**
NMI	**0,726**	**0,926**	**1**	**0,943**	**0,949**	**0,966**	**0,957**	**0,599**	**0,941**
NS	**0,640**	**0,867**	**0,943**	**1**	**0,917**	**0,961**	**0,900**	**0,501**	**0,919**
NT	**0,722**	**0,860**	**0,949**	**0,917**	**1**	**0,916**	**0,934**	**0,591**	**0,859**
NTHN	**0,580**	**0,971**	**0,966**	**0,961**	**0,916**	**1**	**0,890**	**0,658**	**0,950**
OCP	**0,816**	**0,822**	**0,957**	**0,900**	**0,934**	**0,890**	**1**	**0,528**	**0,876**
TLC	0,130	**0,746**	**0,599**	**0,501**	**0,591**	**0,658**	**0,528**	**1**	**0,692**
TASSERT	**0,613**	**0,918**	**0,941**	**0,919**	**0,859**	**0,950**	**0,876**	**0,692**	**1**

Tableau 11 - Corrélation Spearman de MCU et OCP avec les métriques de test

Métriques	NC	NOE	NMI	NS	NT	NTHN	OCP	TLC	TASSERT
NC	**1**	0,285	**0,718**	**0,438**	**0,632**	**0,365**	**0,779**	0,032	**0,395**
NOE	0,285	**1**	**0,668**	0,167	**0,617**	**0,712**	**0,451**	**0,340**	**0,368**
NMI	**0,718**	**0,668**	**1**	**0,548**	**0,896**	**0,772**	**0,904**	**0,458**	**0,696**
NS	**0,438**	0,167	**0,548**	**1**	**0,749**	**0,713**	**0,609**	**0,340**	**0,540**
NT	**0,632**	**0,617**	**0,896**	**0,749**	**1**	**0,891**	**0,826**	**0,466**	**0,626**
NTHN	**0,365**	**0,712**	**0,772**	**0,713**	**0,891**	**1**	**0,676**	**0,564**	**0,666**
OCP	**0,779**	**0,451**	**0,904**	**0,609**	**0,826**	**0,676**	**1**	**0,382**	**0,653**
TLC	0,032	**0,340**	**0,458**	**0,340**	**0,466**	**0,564**	**0,382**	**1**	**0,723**
TASSERT	**0,395**	**0,368**	**0,696**	**0,540**	**0,626**	**0,666**	**0,653**	**0,723**	**1**

La première observation que l'on peut faire sur ces résultats, c'est que presque toutes les métriques de cas d'utilisation sont significativement (et parfois fortement) corrélées aux métriques de test. Parmi les métriques non-corrélées, on trouve le nombre d'opérations externes (NOE) avec le nombre d'assertions (TASSERT) dans la première itération (Tableaux 6 et 7), mais la métrique est corrélée dans 10 et 11. Le nombre de classes (NC) n'est pas corrélé avec le nombre de lignes de code de test (TLC) dans 10 et 11, mais l'est dans 6 et 7. Finalement, dans 7, NC n'est pas corrélés avec TASSERT.

Toutes les autres métriques, soit le nombre de méthodes impliquées (NMI), le nombre de scénarios (NS) et le nombre de transactions, autant détaillées (NT) que de haut niveau (NTHN), ainsi que UCP et OCP sont toutes significativement (et parfois fortement) corrélées avec les deux métriques de test dans toutes les analyses. On remarque aussi que toutes les métriques ont des valeurs de corrélation positives. Cela signifie que lorsque la valeur de la métrique de cas d'utilisation augmente, les valeurs des métriques de test augmentent aussi, rendant ces résultats plausibles car plus un cas d'utilisation est complexe (les métriques relatives à ses attributs sont élevées), plus l'effort de test de ce cas sera élevé.

On remarque aussi que les valeurs de corrélations sont irrégulières entre elles, signifiant que les métriques n'ont pas toutes le même impact sur l'effort de test. Il est possible que les métriques mesurent différentes dimensions de l'effort de test. La validation de cet aspect est, cependant, en dehors de la portée de ce projet. Cependant, comme les métriques NT et NTHN mesurent exactement la même chose, seulement avec des méthodes de collecte différentes, il serait naturel de ne garder que la plus précise des deux métriques. Dans toutes les analyses de corrélation, NTHN est ressortie comme étant plus fortement corrélée que NT. Nous gardons donc NTHN pour la suite du projet et éliminons NT.

Finalement, la dernière observation que nous pouvons faire est que les métriques de classes sont largement non-corrélées. Seules les métriques ayant très au nombre de méthodes (soit NM et NMP) ont une corrélation significative sur l'effort de test (Pearson détecte le couplage et le nombre d'attributs, mais leur valeur de corrélation démontre une influence très réduite). Vu les faibles résultats de cette méthode et le fait qu'elle demande des métriques obtenues plus tard dans les phases du cycle de développement, elle ne sera plus considérée pour le reste du projet.

Pour toutes les autres méthodes, soit MCU, UCP et OCP, il est donc plausible de rejeter H0. Les analyses de corrélation nous permettent d'identifier qu'il existe une relation entre les métriques de cas d'utilisation et les métriques de test. Cependant, il n'est pas réellement possible de comparer l'efficacité des différentes méthodes avec juste cette information. Ceci nécessite une analyse plus poussée.

4.1.2 Analyses de régression logistique

La régression logistique est une méthode utilisée en modélisation statistique pour effectuer une analyse de prédictibilité sur un groupe de variables indépendantes vers une variable dépendante binaire. Dans le cas de ce projet, les variables indépendantes sont les métriques des différentes méthodes, soit MCU, UCP et OCP. La variable dépendante est extrapolée à partir des métriques de test. La technique a été utilisée dans un bon nombre d'études empiriques en génie logiciel [13, 23, 28, 36, 41]. Une régression logistique peut s'effectuer de façon univariée, lorsqu'une seule variable indépendante tente de faire la prédiction de la variable dépendante, ou multivariée, dans le cas de plusieurs variables indépendantes. La régression univariée est utilisée pour les méthodes UCP et OCP (ces méthode sont basées sur une seule métrique) alors que la multivariée est utilisée pour MCU (la méthode est basée sur plusieurs métriques).

Le résultat de l'analyse est un nombre entre 0 et 1 représentant la probabilité de prédire la valeur de la variable dépendante (binaire) avec les informations des variables indépendantes. Un modèle de régression logistique multivariée utilise la formule suivante :

$$P(X_1, \dots X_n) = \frac{e^{(a+\sum_{i=1}^{i=n} b_i X_i)}}{1 + e^{(a+\sum_{i=1}^{i=n} b_i X_i)}}$$

où l'ensemble X_i représente les variables indépendantes et l'ensemble b_i contient les estimations des coefficients de régression, ou leur contribution approximative, correspondant aux variables indépendantes. Plus la valeur d'un coefficient est élevée

plus la variable offre une contribution importante au modèle. P représente la probabilité d'obtenir un cas d'utilisation qui nécessitera un effort de développement (test) élevé. La régression univariée est un cas particulier de cette formule où les ensembles X et b ne contiennent qu'une seule valeur chacun (n= 1).

Plusieurs variables sont collectées durant l'analyse de régression logistique. La p-value joue le même rôle que dans l'analyse de corrélation en représentant la probabilité d'obtenir un coefficient différent de zéro par chance. Comme dans la corrélation, on utilise une valeur alpha de 0.05 pour déterminer la valeur maximale qu'une p-value d'un coefficient peut avoir avant de devenir non-significative. La variable R^2 (Nagelkerke) représente la proportion de la variance de la variable dépendante expliquée par le modèle. En d'autres termes, il s'agit du degré d'efficacité des variables indépendantes à prédire les variables dépendantes.

La performance du modèle obtenu par la régression logistique peut être mesurée par l'analyse ROC (Receiver Operating Charactistic) [20]. L'analyse ROC nous donne une courbe représentant la capacité du modèle à prédire la variable dépendante. Pour évaluer la performance des différentes méthodes présentées, nous utilisons en particulier la variable AUC (Area Under the Curve). La variable représente une proportion des valeurs de la variable dépendante ayant été prédites correctement par le modèle. Une valeur AUC près de 0.5 représente un modèle aléatoire, alors qu'une valeur supérieure à 0.7 représente un bon modèle.

Le choix de la variable dépendante pour l'analyse de régression n'est pas nécessairement évident, considérant que nous avons 2 métriques de test et que ni

l'une ni l'autre ne sont des variables binaires. Pour transformer les métriques de test en variables binaires, nous avons utilisé une transformation simple pour classifier un cas d'utilisation en cas *Complexe* ou *Simple*. Trois perspectives sont explorées : complexité du nombre de lignes de code, complexité du nombre d'assertions et complexité totale.

Les complexités du nombre de lignes de code et du nombre d'assertions fonctionnent de la même façon. Nous avons pris la moyenne de chaque métrique et classifié un cas d'utilisation comme étant complexe si la valeur de la variable analysée était supérieure à cette moyenne. Donc, si un cas d'utilisation a 100 lignes de code de test et que la moyenne est de 80, ce cas d'utilisation est classifié comme étant complexe sous la perspective des lignes de code. Même principe pour le nombre d'assertions.

La troisième perspective est abordée en combinant les deux premières : si un cas est classifié comme étant complexe au niveau des lignes de code et des assertions, on le classifie comme étant complexe, sinon il est considéré simple. Nous obtenons de cette façon trois variables binaires utilisables pour notre analyse de régression logistique. Le tableau 12 représente la distribution des cas d'utilisations selon les différentes métriques.

Tableau 12 - Distribution de la complexité des cas d'utilisation

Métrique	Modalité	Cas d'utilisations	%
TLC	1	9	20,93%
	0	33	79,07%
TASSERT	1	9	20,93%
	0	33	79,07%
Complexité totale	1	8	18,6%
	0	34	81,4%

Avec cette analyse, on tente d'abord de valider trois hypothèses, soit une par méthode (MCU, UCP et OCP). Pour chaque méthode, l'hypothèse est la suivante :

Un cas d'utilisation avec des valeurs de métriques élevées a une plus grande probabilité de mener à un effort de test plus élevé (métriques de test élevées) qu'un cas d'utilisation avec des valeurs de métriques faibles.

Les hypothèses sont :

- H0 : Un cas d'utilisation avec des métriques élevées n'a pas une plus grande probabilité de mener à un effort de test plus élevé qu'un cas d'utilisation avec des métriques basses.
- H1 : Un cas d'utilisation avec des métriques élevées signifie une plus grande probabilité de mener à un effort de test plus élevé qu'un cas d'utilisation avec des métriques basses.

Comme pour les analyses de corrélation, les analyses de régression logistique sont séparées en plusieurs itérations, dû à l'évolution du projet à travers le temps et aux préalables des méthodes UCP et OCP. La première itération est effectuée entre MCU et UCP pour les études ATM, NextGen, CommonsExec, CommonsEmail et CommonsIO. La

régression univariée est utilisée pour la métrique UCP et pour chacune des métriques incluses dans MCU. Le tableau 13 contient les résultats des analyses. Par soucis d'espace et pour éviter les répétitions, seules les résultats d'analyse concernant le nombre de lignes de code de test seront affichés. Les résultats des autres analyses sont très semblables et suivent tous une même tendance. Une discussion concernant ces résultats sera abordée à la fin de cette section.

Tableau 13 - Résultats des analyses de régression logistique univariée entre MCU et UCP

	NC	NOE	NMI	NS	NTHN	UCP
R^2	18,5%	47,9%	53,7%	7,7%	55,1%	53,8%
AUC	0,625	0,816	0,847	0,736	0,927	0,899
p-value	0,022	< 0,0001	< 0,0001	0,148	< 0,0001	< 0,0001

On remarque qu'hormis le nombre de classes (NC) et le nombre de scénarios (NS), toutes les métriques ont une aire sous la courbe (AUC) et une p-value significative, montrant ainsi, que chacune d'entre elles permet une estimation adéquate de la métrique binaire TLC. Les résultats de la méthode UCP sont très intéressants. On peut constater que la métrique NTHN, lorsque prise seule, a les meilleurs résultats autant au niveau de la proportion expliquée (R^2), qu'au niveau de la fiabilité (AUC). Les valeurs ne sont cependant pas tant différentes d'UCP et il serait difficile de déterminer laquelle des deux est la plus efficace.

Comme la méthode MCU est conçue pour utiliser une multitude de métriques, différentes analyses de régression logistique multivariées ont aussi été effectuées pour tous les ensembles de métriques. Le tableau 14 contient les résultats les plus intéressants de ces analyses :

Tableau 14 - Résultats intéressants d'analyses multivariées pour MCU

	NS-NTHN	NC-NTHN	NOE-NS-NTHN	Toutes MCU
R^2	66,4%	56,8%	73,4%	76,8%
AUC	0,934	0,941	0,955	0,955
p-value	< 0,0001	< 0,0001	< 0,0001	< 0,0001

Comme on peut le constater, les résultats de ce tableau sont très révélateurs : les métriques de la méthode MCU, lorsque combinées, offrent un excellent modèle prédictif pour la métrique binaire TLC. Toutes les combinaisons présentées dans le tableau 14 sont supérieures aux modèles univariés, incluant UCP. Les résultats de la métrique MCU, soit la combinaison de toutes les sous-métriques de la méthode, dépassent largement les valeurs obtenues par UCP (76,8% et 0.955 contre 53,8% et 0.899).

Une deuxième itération d'analyses de régression logistique est effectuée. On remplace le projet CommonsIO par Joda-Time et on utilise les méthodes MCU et OCP. Le tableau 15 contient les résultats des régressions univariées :

Tableau 15 - Résultats des régressions logistiques multivariées entre MCU et OCP

	NC	NOE	NMI	NS	NTHN	OCP
R^2	0,3%	42,8%	28%	12,7%	42,5%	21%
AUC	0,488	0,783	0,763	0,587	0,817	0,733
p-value	0,788	< 0,0001	0,006	0,072	< 0,0001	0,019

Cette fois-ci, les résultats sont notamment moins élevés que dans l'itération précédente. Les tendances restent cependant les mêmes : les métriques de MCU avec NC et NS ayant des résultats moins intéressants et NTHN restant au-dessus, suivi de NOE. La métrique OCP démontre un potentiel intéressant qui est presque identique à NTHN dans les résultats de comparaison entre MCU et UCP. Mais ce résultat est inférieur à celui obtenu au niveau d'UCP.

Le tableau 16 contient les résultats des régressions multivariées effectuées pour les mêmes combinaisons de métriques présentées dans la première itération :

Tableau 16 - Résultats intéressants des régressions logistiques multivariées pour MCU

	NS-NTHN	NC-NTHN	NOE-NS-NTHN	Toutes MCU
R^2	48,1%	52,5%	44,8%	81%
AUC	0,842	0,875	0,808	0,979
p-value	0,001	< 0,0001	0,002	< 0,0001

Encore une fois, les résultats des analyses multivariées sont significativement plus élevés que ceux des univariées. Il est intéressant de noter que ces résultats semblent plutôt constants, lorsqu'on les compare à ceux de l'itération précédente. MCU, particulièrement, offre toujours une proportion expliquée et une précision très supérieures à toutes les autres métriques.

D'autres analyses, moins détaillées, ont aussi été effectuées pour les deux autres métriques binaires présentées, soit la complexité du nombre d'assertions et la complexité totale. Les résultats ne sont pas exactement les mêmes, mais suivent tous

la même tendance : les résultats de TASSERT sont toujours légèrement plus bas que ceux de TLC et ceux de TTOTAL sont plus bas que ceux de TASSERT. Refaire les analyses pour ces métriques de test nous amènerait très probablement vers les mêmes conclusions.

Somme toutes, on peut constater que la majorité des métriques répondent aux critères nécessaires pour être considérées comme offrant des résultats significatifs pour la prédiction des métriques de test. Il est donc possible de rejeter l'hypothèse nulle pour les trois méthodes présentées.

En observant en détail les résultats, on peut aussi constater une différence entre l'efficacité des métriques. En effet, la métrique MCU NC ne semble pas avoir une grande influence sur le modèle de régression. La métrique NS a un résultat moindre aussi, mais semble avoir une influence intéressante lorsque combinée avec d'autres métriques, notamment NTHN. Les résultats des métriques MCU combinées sont grandement supérieurs à ceux des autres, incluant les métriques OCP et UCP. Il serait donc plausible d'affirmer qu'une méthode basée sur les métriques présentées par MCU serait plus efficace pour prédire l'effort de test que les méthodes existantes OCP et UCP.

Chapitre 5 - Prédiction des efforts de développement

La deuxième série d'analyses effectuées porte sur la prédiction des efforts de développement. L'effort du développement a été abordé sous la perspective de la taille du code source. D'abord, une courte explication sur la différence des métriques d'effort utilisées entre les deux sections sera abordée. Par la suite, similaire à la première section d'analyses, les différentes analyses et hypothèses sont présentées, suivies immédiatement des résultats et des interprétations.

5.1 Métrique d'effort de développement

La prédiction de l'effort de développement est un sujet d'intérêt dans la littérature de génie logiciel. Beaucoup d'études [21, 25, 32-34] l'abordent et plusieurs méthodes ont été développées pour résoudre le problème. Même les méthodes OCP et UCP, présentées plus haut dans le contexte de la prédiction de l'effort de test, ont leur origine dans l'estimation de l'effort de développement. En constatant que ces méthodes peuvent aussi être utilisées pour prédire l'effort de test, il serait intéressant de voir si MCU, développée à l'origine pour prédire l'effort de test, peut aussi être utilisée pour prédire l'effort de développement avec une précision satisfaisante. Ceci rendrait la méthode encore plus intéressante dans la pratique.

Il est possible de tracer des liens entre l'effort de développement et de test. Les métriques de test analysées étaient le nombre de lignes de code de la suite de test et le nombre d'assertions. Il est plus ou moins possible de faire la transition du nombre

d'assertions vers l'effort de développement, mais le nombre de lignes de code est un concept existant dans les deux mondes.

Pour pouvoir faire le parallèle avec les résultats et analyses de la partie précédente, il faut définir comment la métrique du nombre de lignes de code source est utilisée dans notre projet. Pour chaque cas d'utilisation extrait des études de cas dans la partie précédente, on considère les classes (logicielles) et les méthodes développées pour répondre au cas d'utilisation. Le nombre de lignes de code source de ces méthodes et classes est comptabilisé et associé au cas d'utilisation. Seules les lignes ayant trait au cas d'utilisations ont été considérées (aucune incidence au niveau des méthodes, mais certains attributs des classes peuvent être ignorés s'ils ne sont pas utilisés dans les méthodes d'un cas).

5.2 Étude empirique

5.2.1 Analyses de corrélations

Pour pouvoir repérer les liens entre les métriques d'analyse et le code source, nous utilisons, comme dans le chapitre précédent, une analyse de corrélation pour tester les hypothèses suivantes :

- H0 : Il n'existe pas de corrélation significative entre la métrique de cas d'utilisation et le nombre de lignes de code source.

- H1 : Il existe une corrélation significative entre la métrique de cas d'utilisation et le nombre de lignes de code source.

Une fois de plus, les méthodes Pearson et Spearman sont utilisées avec un seuil significatif alpha = 0.05. Les analyses sont faites en deux itérations. La première entre MCU et UCP utilisant les études ATM, NextGen, CommonsExec, CommonsEmail et CommonsIO. Les tableaux 17 et 18 contiennent les résultats de cette itération. La métrique NLC représente le nombre de lignes de code source.

Tableau 17 - Corrélation Pearson entre MCU et UCP avec les métriques de code source

Métriques	NIC	NEO	NIM	NS	NT	NTHN	UCP	UCP HN	NLC
NC	**1**	**0,517**	**0,748**	**0,537**	**0,697**	**0,527**	**0,510**	0,254	**0,655**
NOE	**0,517**	**1**	**0,926**	**0,816**	**0,851**	**0,944**	**0,323**	**0,453**	**0,764**
NMI	**0,748**	**0,926**	**1**	**0,860**	**0,932**	**0,913**	**0,455**	**0,440**	**0,856**
NS	**0,537**	**0,816**	**0,860**	**1**	**0,893**	**0,930**	**0,455**	**0,549**	**0,807**
NT	**0,697**	**0,851**	**0,932**	**0,893**	**1**	**0,935**	**0,682**	**0,648**	**0,869**
NTHN	**0,527**	**0,944**	**0,913**	**0,930**	**0,935**	**1**	**0,512**	**0,648**	**0,833**
UCP	**0,510**	**0,323**	**0,455**	**0,455**	**0,682**	**0,512**	**1**	**0,820**	**0,588**
UCP HN	0,254	**0,453**	**0,440**	**0,549**	**0,648**	**0,648**	**0,820**	**1**	**0,580**
NLC	**0,655**	**0,764**	**0,856**	**0,807**	**0,869**	**0,833**	**0,588**	**0,580**	**1**

Tableau 18 - Corrélation Spearman entre MCU et UCP avec les métriques de code source

Métriques	NIC	NEO	NIM	NS	NT	NTHN	UCP	UCP HN	NLC
NC	**1**	**0,485**	**0,832**	0,149	**0,581**	0,216	**0,519**	0,232	**0,581**
NOE	**0,485**	**1**	**0,749**	0,087	**0,608**	**0,564**	**0,470**	**0,416**	**0,522**
NMI	**0,832**	**0,749**	**1**	0,190	**0,775**	**0,484**	**0,712**	**0,477**	**0,744**
NS	0,149	0,087	0,190	**1**	**0,658**	**0,764**	**0,684**	**0,726**	**0,613**
NT	**0,581**	**0,608**	**0,775**	**0,658**	**1**	**0,829**	**0,925**	**0,752**	**0,852**
NTHN	0,216	**0,564**	**0,484**	**0,764**	**0,829**	**1**	**0,816**	**0,903**	**0,730**
UCP	**0,519**	**0,470**	**0,712**	**0,684**	**0,925**	**0,816**	**1**	**0,864**	**0,852**
UCP HN	0,232	**0,416**	**0,477**	**0,726**	**0,752**	**0,903**	**0,864**	**1**	**0,736**
NLC	**0,581**	**0,522**	**0,744**	**0,613**	**0,852**	**0,730**	**0,852**	**0,736**	**1**

La deuxième itération est effectuée entre les méthodes MCU et OCP en excluant l'étude CommonsIO et en ajoutant Joda-Time, ayant une meilleure architecture objet, pour tenir compte du fait que la méthode OCP est construite pour fonctionner sur des projets orientés-objet. Les résultats de cette analyse sont donnés dans les tableaux 19 et 20.

Tableau 19 - Corrélation Pearson entre MCU et OCP avec les métriques de code source

Métriques	NIC	NEO	NIM	NS	NT	NTHN	OCP	NLC
NC	**1**	**0,495**	**0,726**	**0,640**	**0,722**	**0,580**	**0,816**	**0,668**
NOE	**0,495**	**1**	**0,926**	**0,867**	**0,860**	**0,971**	**0,822**	**0,767**
NMI	**0,726**	**0,926**	**1**	**0,943**	**0,949**	**0,966**	**0,957**	**0,871**
NS	**0,640**	**0,867**	**0,943**	**1**	**0,917**	**0,961**	**0,900**	**0,851**
NT	**0,722**	**0,860**	**0,949**	**0,917**	**1**	**0,916**	**0,934**	**0,879**
NTHN	**0,580**	**0,971**	**0,966**	**0,961**	**0,916**	**1**	**0,89**	**0,840**

							0	
OCP	**0,816**	**0,822**	**0,957**	**0,900**	**0,934**	**0,890**	**1**	**0,933**
NLC	**0,668**	**0,767**	**0,871**	**0,851**	**0,879**	**0,840**	**0,93 3**	**1**

Tableau 20 - Corrélation Spearman entre MCU et OCP avec les métriques de code source

Métriques	NIC	NEO	NIM	NS	NT	NTHN	OCP	NLC
NC	**1**	0,285	**0,718**	**0,438**	**0,632**	**0,365**	**0,779**	**0,653**
NOE	0,285	**1**	**0,668**	0,167	**0,617**	**0,712**	**0,451**	**0,479**
NMI	**0,718**	**0,668**	**1**	**0,548**	**0,896**	**0,772**	**0,904**	**0,882**
NS	**0,438**	0,167	**0,548**	**1**	**0,749**	**0,713**	**0,609**	**0,741**
NT	**0,632**	**0,617**	**0,896**	**0,749**	**1**	**0,891**	**0,826**	**0,873**
NTHN	**0,365**	**0,712**	**0,772**	**0,713**	**0,891**	**1**	**0,676**	**0,786**
OCP	**0,779**	**0,451**	**0,904**	**0,609**	**0,826**	**0,676**	**1**	**0,916**
NLC	**0,653**	**0,479**	**0,882**	**0,741**	**0,873**	**0,786**	**0,916**	**1**

Aucune itération n'est faite cette fois-ci pour les métriques de classe. Les résultats peu concluants de la première partie ne nous poussent pas à approfondir cette analyse dans ce projet.

En observant les résultats des corrélations, on remarque rapidement que toutes les métriques d'analyse, sans exceptions, sont fortement corrélées avec la métrique de code source. Pour UCP et OCP, il n'y a rien de surprenant, compte tenu du fait que ces méthodes ont été créées pour prédire l'effort de développement. Mais il est intéressant de constater que même MCU offre également de bons résultats pour l'estimation de la taille du code source.

Encore une fois, toutes les métriques ont des valeurs de corrélation positives, ce qui signifie que le lien entre les métriques d'analyse et la taille du code est un lien positif;

plus une métrique d'analyse est élevée, plus la métrique de code source le sera aussi. Les valeurs de corrélation étant irrégulières, on peut aussi conclure que les métriques n'ont pas toutes le même impact sur la taille du code source. Dans la dernière section, nous avions éliminé la métrique NT en constatant que NTHN, son équivalent plus haut niveau, donnait une meilleure corrélation. Cette fois-ci, il s'agit de l'inverse. Il serait logique de spéculer sur le fait que le nombre de transactions de bas niveau, étant plus près du nombre de méthodes visibles dans un diagramme de séquence de l'étape de conception, offre une meilleure visibilité sur la taille du code que le simple nombre de transactions systèmes. Cependant, et pour rester le plus loin possible des métriques de conception, nous gardons encore une fois NTHN comme métrique représentative du nombre de transactions pour les analyses suivantes.

Suite à ces observations, il est possible de rejeter l'hypothèse H0 pour toutes les métriques d'analyse, incluant MCU, UCP et OCP. Nous pouvons affirmer qu'il existe une relation entre les métriques de cas d'utilisation et la métrique de code source. Nous procédons par la suite aux régressions logistiques pour pousser plus loin l'analyse de cette relation.

5.2.2 Analyses de régression logistique

Comme dans la première partie, les variables indépendantes des régressions logistiques seront les métriques de MCU, UCP et OCP. La variable dépendante est dérivée du nombre de lignes de code source selon le même principe que celui utilisé dans la première partie de notre étude. Le tableau 21 contient la distribution de la complexité des cas d'utilisation en termes d'effort de développement.

Tableau 21 - Distribution de la complexité des cas d'utilisation (code source)

Métrique	Modalité	Cas d'utilisations	%
NLC	1	20	43,48%
	0	26	56,52%

Nous utilisons la méthode univariée pour les différentes métriques de MCU, et pour UCP et OCP, et la méthode multivariée sur les mêmes séries de combinaison des métriques MCU utilisées dans la première partie. Les hypothèses sont aussi les mêmes, soit :

- H0 : Un cas d'utilisation avec des valeurs de métriques élevées n'a pas une plus grande probabilité de mener à un effort de test plus élevé qu'un cas d'utilisation avec des valeurs de métriques faibles.
- H1 : Un cas d'utilisation avec des valeurs de métriques élevées a une plus grande probabilité de mener à un effort de test plus élevé qu'un cas d'utilisation avec des valeurs de métriques faibles.

Plusieurs itérations sont aussi utilisées pour cette analyse. La première itération, entre MCU et UCP, est faite pour les études ATM, NextGen, CommonsExec, CommonsEmail et CommonsIO. Les résultats de cette itération sont disponibles dans le tableau 22.

Tableau 22 - Résultats des analyses de régression logistique univariée entre MCU et UCP

	NC	NOE	NMI	NS	NTHN	UCP
R^2	46,4%	8,2%	47,5%	38%	38,5%	37,1%
AUC	0,821	0,733	0,838	0,813	0,835	0,837
p-value	< 0,0001	0,088	< 0,0001	< 0,0001	< 0,0001	< 0,0001

On remarque encore une fois, que presque toutes les métriques ont une p-value significative (< 0.05). L'exception cette fois-ci est le nombre d'opérations externes (NOE). Au niveau de la proportion expliquée (R^2), les résultats sont un peu moins élevés qu'ils ne l'étaient pour l'estimation de l'effort de test. Les métriques NMI et NC ressortent comme étant les plus hautes, probablement dû au fait que ces métriques mesurent des éléments qui se traduisent presque directement au niveau du code source (le nombre de classes et le nombre de méthodes). Étonnamment, la métrique UCP n'affiche pas une proportion particulièrement élevée. Les résultats démontrent tout de même que la métrique permet une estimation adéquate du nombre de lignes de code source, mais elle semble moins efficace que pour la prédiction de l'effort de test.

Le tableau 23 contient les résultats des analyses faites sur des combinaisons des métriques de la méthode MCU.

Tableau 23 - Résultats intéressants d'analyses multivariées pour MCU

	NS-NTHN	NC-NTHN	NOE-NS-NTHN	Toutes MCU
R^2	44,4%	69,2%	45,8%	80,5%
AUC	0,862	0,931	0,867	0,965
p-value	< 0,0001	< 0,0001	< 0,0001	< 0,0001

Encore une fois, les résultats sont légèrement moins élevés que pour l'effort de test pour la plupart des métriques, hormis MCU (l'ensemble des métriques). Dans ce dernier cas, le résultat est supérieur au résultat obtenu dans la première partie pour la

même itération, mais la différence n'est pas très grande. Il serait raisonnable de considérer que, si la tendance se maintient pour la deuxième itération, MCU donnera un résultat semblable pour l'estimation de l'effort de test et l'effort de développement. D'après les résultats de cette itération, il semble clair que la méthode MCU a pu prédire avec plus de précision le nombre de lignes de code que la méthode UCP.

La deuxième itération s'effectue, comme pour la partie précédente de notre étude, entre MCU et OCP en remplaçant l'étude CommonsIO par Joda-Time. Le tableau 24 contient les résultats de cette itération.

Tableau 24 - Résultats des analyses de régression logistique univariée entre MCU et OCP

	NC	NOE	NMI	NS	NTHN	OCP
R^2	66,3%	9,4%	69,5%	50,6%	38,3%	90,2%
AUC	0,937	0,756	0,946	0,906	0,864	0,989
p-value	< 0,0001	0,096	< 0,0001	< 0,0001	< 0,0001	< 0,0001

Contrairement à la première partie, où les valeurs des résultats de la seconde itération étaient inférieures à la première, cette fois-ci la seconde itération nous donne des valeurs supérieures, mais la tendance se maintient pour chacune des métriques de MCU. NOE est toujours presque non-significative, avec une proportion expliquée très basse, une aire sous la courbe (AUC) plus faible et une p-value trop élevé. La performance d'OCP, par contre, est très intéressante. Il s'agit de la plus élevée jusqu'à maintenant, surpassant toutes les métriques individuelles de MCU. La proportion expliquée est aussi inégalée dans les autres analyses effectuées jusqu'à maintenant.

Pour pouvoir finaliser la comparaison, le tableau 25 contient les résultats des analyses multivariées effectuées sur les combinaisons de MCU.

Tableau 25 - Résultats intéressants d'analyses multivariées pour MCU

	NS-NTHN	NC-NTHN	NOE-NS-NTHN	Toutes MCU
R^2	54,9%	81%	54,4%	88,3%
AUC	0,892	0,966	0,886	0,98
p-value	< 0,0001	< 0,0001	< 0,0001	< 0,0001

La tendance reste semblable à l'itération précédente, avec NC-NTHN offrant de bons résultats et MCU offrant des résultats très intéressants. Les résultats de MCU sont légèrement inférieurs à OCP, mais la différence est minime et il est difficile de juger laquelle des deux méthodes offre la meilleure estimation. On peut tout de même retenir de cette itération qu'OCP, même si elle a été proposée pour prédire l'effort de développement, obtient d'excellents résultats et que MCU permet aussi d'obtenir des résultats intéressants.

Il est intéressant d'observer que les métriques de MCU ayant obtenu des résultats élevés lorsqu'observées individuellement ne sont pas les mêmes entre la première partie et la deuxième. Dans la première partie, la métrique NC donnait des résultats médiocres et NTHN donnait les meilleures résultats, alors que dans cette dernière partie, NTHN n'a pas autant performé alors que NC a donné de bons résultats. Ceci nous permet de déterminer que, même si la méthode MCU est efficace pour prédire l'effort de développement et l'effort de test, parmi les métriques de MCU, certaines sont plus efficaces pour prédire l'un ou l'autre.

D'après les analyses de régression logistique, nous pouvons constater que la méthode MCU est aussi efficace pour prédire l'effort de test et l'effort de développement que la méthode OCP. Quant à UCP, elle semble plus efficace pour l'estimation de l'effort de test, mais a tout de même obtenu des résultats adéquats pour l'effort de développement. On peut rejeter l'hypothèse nulle.

Chapitre 6 - Discussion et interprétation des résultats

6.1 Sommaire des résultats

L'objectif initial de ce projet était de développer une méthode (du moins les bases de ce que pourrait être à l'avenir une méthode) qui utiliserait des métriques disponibles très tôt dans le cycle de développement pour prédire l'effort de test. Nous avons effectué une série d'analyses statistiques pour démontrer l'efficacité de cette méthode et l'avons comparée aux méthodes existantes. En plus des analyses sur la prédiction de l'effort de test, des analyses pour la prédiction de l'effort de développement (sous la perspective de la taille du code source) ont aussi été effectuées.

Nous avons émis plusieurs hypothèses quant à l'utilisation des métriques de cas d'utilisation pour prédire l'effort de test et de développement. Les mêmes hypothèses ont été utilisées pour les métriques des méthodes MCU, les métriques de classe, UCP et OCP. Pour UCP et OCP, il était prévisible que les hypothèses soient validées (confirmées), mais le processus a été fait dans le but de pouvoir comparer les méthodes entre elles. Comme prévu, les méthodes UCP et OCP ont été efficaces dans la prévision de l'effort de test ainsi que l'effort de développement.

Les métriques de classe ont donné de faibles résultats dans les analyses de corrélation. De plus, la méthode était basée sur des métriques disponibles (relativement) tard dans le processus de développement, surtout comparée à MCU. Vu la piètre performance de cette méthode, il a été décidé de ne pas continuer avec celle-ci pour les analyses subséquentes.

La méthode MCU, quant à elle, s'est montrée aussi efficace et plus dans certains cas que les méthodes existantes. Presque toutes les métriques individuelles de la méthode ont eu une corrélation significative avec les métriques de test et de développement. Les seules exceptions ont été observées au niveau de la métrique de test TASSERT. Ceci dit, les analyses de régression logistique effectuées sur TASSERT démontrent des résultats similaires à ceux du nombre de lignes de code de test, mais avec de moindres performances (les valeurs sont généralement inférieures).

Les analyses de régression logistiques de MCU donnent des résultats intéressants même pour les métriques individuelles, mais c'est lorsqu'on les combine que la méthode MCU est plus efficace. La métrique combinée MCU dépasse ou minimalement égalise les méthodes existantes dans toutes les analyses de régression logistique effectuées. Il est donc clair que la méthode a le potentiel d'effectuer une excellente prévision, autant au niveau de l'effort de test qu'au niveau de l'effort de développement.

Suite aux résultats des analyses de régression logistique univariée, on a pu constater une différence dans la proportion expliquée et l'aire sous la courbe (AUC) des différentes métriques de MCU entre la partie sur l'effort de test et l'effort de développement. Ceci indique que différentes métriques sont responsables de la précision des estimations dans les deux cas. Ce point est intéressant, car il confirme qu'une méthode peut avoir des résultats différents dépendamment si on l'utilise pour estimer l'effort de développement ou de test. La méthode MCU offre un ensemble de métriques permettant, lorsqu'elles sont combinées, de faire l'estimation des deux cas,

ce qui lui donne un caractère plus intéressant que les autres méthodes auxquelles elle a été comparée.

Il pourrait être intéressant de voir si les métriques de MCU ont des proportions expliquées qui s'entrecoupent largement. Il est possible qu'une ou plusieurs des métriques soit en majeure partie expliquée par une autre et, par conséquent, superflue dans le calcul. Pour pouvoir le découvrir, il serait possible d'effectuer une analyse par composantes principales sur l'ensemble des métriques recueillies dans ce projet. Cela permettrait de raffiner la méthode MCU en enlevant la nécessité de récolter des métriques inutiles (due à la duplication éventuellement de l'information).

6.2 Obstacles à la validité

Pour pouvoir tirer des conclusions plus larges sur le potentiel d'utilisation des métriques de la méthode MCU, les analyses faites dans ce projet devront être reproduites sur plusieurs autres études de cas. Un nombre de facteurs entrent en compte lorsqu'on considère la validité des résultats du projet.

Les cas d'utilisation récoltés pour mener les analyses statistiques sont au nombre d'une cinquantaine. Quoi que ce nombre ait une valeur statistique significative pour les analyses effectuées, il n'est pas possible d'affirmer que les résultats obtenus dans ce projet peuvent être généralisées. De plus, les cas d'utilisations ont été extraits de cinq études de cas différentes, signifiant qu'ils ont des différences dans leur complexité et leur nature. L'effet de différence est diminué par le fait que la rétro-ingénierie des cas

d'utilisation a été majoritairement effectuée par une même personne (l'auteur), nous permettant d'éliminer les différences de style dans l'écriture des documents d'analyse. En particulier, le nombre de transactions est une métrique affectée par le style d'écriture des diagrammes de séquence, car il s'agit de compter le nombre d'événements inclus dans chaque diagramme. De même, la rédaction des tests unitaires manquants dans certaines études a été faite par la même personne.

Un autre facteur à considérer dans la comparaison entre les différentes méthodes présentées est le fait que la méthode UCP n'a pas pu être utilisée à plein escient sur les études de cas. Comme mentionné dans la section réservée à cette méthode, UCP requiert le calcul des facteurs techniques et environnementaux d'un projet. Les facteurs techniques ont pu être déterminés par rétro-ingénierie, mais il est impossible, sans consulter l'équipe originale responsable du développement des études, de déterminer les facteurs environnementaux. Ces derniers représentent la composition de l'équipe, la familiarité de cette dernière avec le domaine traité par le logiciel et d'autres facteurs ayant trait à l'environnement de travail de l'équipe. Les résultats obtenus par la méthode UCP pourraient donc être supérieurs si cette information avait été disponible. Ils pourraient aussi être inférieurs, car ces facteurs sont souvent jugés subjectifs et peuvent affecter la qualité de la précision dans la prédiction, observation soulignée par plusieurs études. Les résultats de ce projet sont tout de même intéressants, car ils démontrent que la méthode MCU fonctionne mieux que la méthode UCP dans un contexte où les facteurs environnementaux ne sont pas disponibles

Les résultats et découvertes auxquels nous sommes arrivés dans ce projet devraient être vu comme exploratoires et indicatifs plutôt que conclusifs. Nous avons pu

déterminer que l'utilisation des métriques de cas d'utilisation offre une alternative simple, basée sur des métriques simples et objectives, pouvant être utilisée rapidement dans le cycle de développement, pour prédire l'effort de test et de développement d'un logiciel.

Chapitre 7 - Conclusions

7.1 Résumé

Dans ce projet, nous avons exploré la possibilité et le besoin d'effectuer la prédiction de l'effort de développement et de test d'un logiciel le plus tôt possible dans son cycle de développement. Plusieurs méthodes proposées dans la littérature ont été présentées pour effectuer cette prédiction. De plus, deux nouvelles méthodes, une basée sur les métriques de cas d'utilisation (MCU) et l'autre sur les métriques de diagramme de classes, ont été proposées et évaluées. Les métriques de ces méthodes sont simples et faciles à obtenir, et sont mathématiquement valides comparées aux constructions de certaines méthodes (telle que UCP). Elles sont aussi disponibles tôt dans le cycle de développement, surtout dans le cas de la première méthode. L'objectif de ce projet était de faire l'analyse et la comparaison de ces méthodes avec d'autres méthodes existantes, notamment les méthodes Use Case Points (UCP) et Objective Class Points (OCP). La méthode UCP est aussi basée sur les cas d'utilisation, alors que la méthode OCP est basée sur les métriques du diagramme de classes.

L'analyse et la comparaison ont été effectuées en utilisant des études de cas venant de six projets Java différents. L'idée était de comparer les quatre méthodes entre elles pour déterminer les méthodes les plus efficaces dans différentes situations pour la prévision de l'effort de développement et de test. L'emphase avait initialement été mise sur l'effort de test, puis sur l'effort de développement par la suite pour compléter la recherche. En premier lieu, nous avons analysé les relations entre les différentes métriques issues des méthodes présentées et les métriques d'effort de développement et de test (telles quelles ont été définies dans ce document). Ensuite, nous avons

effectué différentes analyses de régression logistique (univariée et multivariée) pour évaluer de manière poussée l'effet que chaque métrique d'analyse et de conception pouvait avoir sur les métriques d'effort de développement et de test.

L'analyse des relations entre les métriques des méthodes de prévision et les métriques d'effort a été faite en utilisant deux analyses de corrélation statistique; les techniques de Pearson et Spearman. Pour la majorité des métriques, autant pour la méthode proposée que pour les méthodes existantes, il y avait une corrélation significative entre la métrique d'analyse et de conception et la métrique d'effort. La seule exception fut les métriques de classe, qui, quoi que corrélées en partie, n'offraient pas de résultats comparables aux autres méthodes, malgré le fait que ses métriques avaient été collectées durant la phase de conception, soit plus tard dans le cycle de développement. Cette découverte a mené à l'abandon de cette méthode pour le reste du projet. Pour les autres méthodes, les résultats obtenus suggèrent qu'il est possible d'effectuer la prévision d'effort à partir des métriques d'analyse et de conception.

Plusieurs modèles de régression logistique ont été utilisés pour mener les analyses suivantes. Une régression logistique univariée a été utilisée pour chaque métrique d'analyse et conception (soit les métriques indépendantes de MCU et les métriques uniques d'UCP et OCP). Les résultats de cette étape ont démontré plusieurs éléments intéressants. Pour la prévision de l'effort de test, la métrique *Nombre de Transactions* (NT) a donné des résultats très significatifs autant au niveau de la proportion expliquée que de la probabilité à prédire correctement l'effort de test. Pour l'effort de développement, c'est le *Nombre de Classes* (NC) qui a permis d'obtenir les meilleurs résultats parmi les métriques de la méthode MCU. Les méthodes UCP et OCP ont performé de façon significative pour les deux types de prévisions, malgré le fait que la

méthode UCP ait obtenu des résultats moins favorables pour la prévision de l'effort de développement.

Finalement, quelques modèles de régression logistique multivariés ont été testés avec des combinaisons des métriques MCU. Les résultats démontrent que, lorsqu'on combine toutes ses métriques, elle devient au moins aussi précise, sinon plus, que les modèles univariés obtenus par les méthodes UCP et OCP, autant pour la prévision de l'effort de développement que l'effort de test. Parmi les combinaisons étudiées, nous avons pu constater que les meilleurs résultats étaient obtenus par des combinaisons différentes entre les prévisions d'effort de développement et de test. Quoi que la combinaison NC et NT obtient d'excellents résultats pour les deux cas. Dans le cas de l'effort de test, la combinaison du *Nombre de Scénarios* (NS) et NT offre des résultats comparables. On constate donc que différentes métriques sont utiles soit pour la prévision de l'effort de développement ou de l'effort de test. Il semble aussi clair que les métriques NC et NT sont les plus propices à être utilisées pour la prévision d'effort de toute sorte. Les métriques NC et NT sont simples à obtenir à partir des documents de cas d'utilisation.

Les résultats obtenus démontrent que les métriques de cas d'utilisation, particulièrement NC et NT, sont de bonnes mesures pour effectuer la prévision de l'effort de développement et de test, du moins dans un environnement où la documentation des cas d'utilisation est disponible et complète à un niveau adéquat (où l'on peut obtenir ces métriques). Les analyses ont démontré que la méthode MCU offre des prévisions efficaces, rivalisant et dépassant, dans plusieurs cas, les méthodes UCP et OCP existantes. La limitation principale de ces trouvailles est la quantité relativement petite d'études de cas utilisées pour effectuer les analyses.

Les approches proposées diffèrent des méthodes UCP et OCP de plusieurs façons. UCP nécessite des données techniques et environnementales pour pouvoir effectuer ses prévisions. La récolte de ces données nécessite un certain niveau de jugement de la part de la personne les récoltant, pouvant introduire un biais dans les résultats de la méthode. Ce défaut n'est pas présent dans la méthode MCU. La méthode OCP, quant à elle, utilise certaines métriques en commun avec MCU, mais obtient d'autres métriques à partir des documents de la phase de conception (couplage, héritage, nombre d'attributs), ce qui est relativement tard dans le processus de développement. La méthode MCU fait usage du *Nombre de Méthodes* (NM), une métrique généralement disponible dans la phase de conception, mais comme cette métrique n'a pas démontré d'efficacité ou d'importance primordiale dans les modèles de régression, il pourrait potentiellement être intéressant de la retirer de MCU. De cette façon, MCU garderait l'avantage d'être utilisable à partir uniquement des métriques d'analyse.

7.2 Lectures connexes

Dans le courant de cette recherche et la rédaction de ce projet, plusieurs articles ont été publiés sur les sujets traités dans ce document. Ces articles explorent de nouvelles facettes et ajoutent de l'information intéressante aux analyses et aux résultats obtenus dans ce projet.

Dans [7], Badri et al. explorent la possibilité d'effectuer l'estimation de la taille des suites de test à partir des métriques de cas d'utilisation. Les métriques présentées sont essentiellement les mêmes que celles présentées dans ce projet, en notant l'absence

de NT, car cette dernière n'avait pas encore été ajoutée à l'époque de la publication de cet article. L'article effectue des analyses similaires sur MCU et les métriques de test, en allant un peu plus loin dans la quantité de détail inclus dans les analyses.

Ensuite, dans [8], Badri et al. poursuivent dans le même chemin en poussant plus loin les analyses de MCU avec les métriques de test. Entre autres, une analyse par composantes principales est effectuée sur les métriques de la méthode MCU. Cette analyse démontre qu'il existe effectivement une redondance d'information dans les métriques de la méthode. Parmi les quatre métriques (NT toujours manquante à ce moment) de la méthode, trois composantes ont été trouvées expliquant 99% de l'information. La même analyse a été faite sur les métriques de test (au nombre de trois à ce moment, rajoutant le *Nombre de Tests* (NBTEST) au groupe) et la première composante trouvée expliquait 88% des données. Les métriques MCU retenues dans les différentes combinaisons explorées (analyse de régression multivariée) dans le mémoire figurent parmi les métriques qui se complètent.

Finalement, trois autres articles (un premier soumis à un journal spécialisé (IJSEKE) – en cours d'évaluation, des corrections mineures ont été demandées et effectuées; un deuxième soumis à une conférence internationale en génie logiciel (SEDE 2015), qui vient d'être accepté, et le troisième en cours de finalisation) ont été montés, poursuivant, affinant et complétant les analyses. Cette fois-ci, les articles portent sur la comparaison entre la méthode MCU et la méthode UCP, de façon plus poussée et effectuant plus d'analyses (e.g., des analyses de régression linéaire ont été rajoutées, utilisation d'algorithmes d'apprentissage automatique, etc.) sur les données. Suite aux résultats de l'analyse par composantes principales et au fait que NM est une métrique de conception, cette dernière a été retirée de la méthode.

7.3 Recommandations

Les résultats de ce projet démontrent clairement une relation entre les métriques de la méthode proposée basée sur les cas d'utilisation et les métriques de développement et de test. Les modèles de régression logistique utilisés nous permettent aussi de voir que la méthode MCU possède le potentiel de rivaliser ou même de surpasser les méthodes existantes UCP et OCP pour la prévision de l'effort de développement et de test.

Les analyses effectuées dans ce projet devraient cependant être répliquées sur une multitude de systèmes orientés-objet pour pouvoir en retirer des conclusions plus générales. Il est nécessaire d'effectuer plus de recherche sur la précision de la prévision en utilisant la méthode proposée dans des projets de nature de toute sorte. Les résultats démontrent le potentiel d'une méthode pouvant être utilisée dans les premières étapes du cycle de développement pour effectuer les prévisions sur l'effort à déployer lors des dernières étapes.

Dans des travaux futurs, il serait intéressant de comparer la méthode MCU à d'autres méthodes existantes, utiliser d'autres outils pour manipuler les métriques de cas d'utilisation pour étendre la méthode MCU (e.g. des méthodes d'apprentissage automatique) et, comme mentionné plus haut, répliquer l'étude sur un nombre de projets orientés objet pour pouvoir effectuer une généralisation des résultats.

Références

[1] Albrecht, A.: « Measuring application development productivity. » IBM Application Development Symposium, 1979, pp.83-92.

[2] Almeida, É.R.C., Abreu, B.T., Moraes, R.: « An Alternative Approach to Test Effort Estimation Based on Use Cases. » Proceedings of the International Conference on Software Testing, Verification and Validation. IEEE Computer Society, (2009).

[3] Antoniol, G., Calzolari, F., Cristoforetti, L., Fiutem, R., Caldiera, G., « Adapting function points to object-oriented information systems », Proceedings of the 10th International Conference on Advanced Information Systems Engineering, 1998, pp. 59–76.

[4] Badri, L., Badri, M., Toure, F.: « Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems. » Kim, T.-H., Kim, H.-K., Khan, M.K., et al. (eds.) ASEA 2010. CCIS, vol. 117, pp. 78–92. Springer, Heidelberg, (2010).

[5] Badri, L., Badri, M., Toure, F.: « An empirical analysis of lack of cohesion metrics for predicting testability of classes. » International Journal of Software Engineering and Its Applications, 5(2), (2011).

[6] Badri, M., Toure, F.: « Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes. » Journal of Software Engineering and Applications, 5(7), (July 2012).

[7] Badri, M., Badri, L., Flageol, W.: « On the Relationship between Use Cases and Test Suites Size: An Exploratory Study. » ACM SIGSOFT Software Engineering Notes, 38(4), (Juillet 2013).

[8] Badri, M., Badri, L., Flageol, W.: « Predicting the size of test suites from use cases: An empirical exploration. » H. Yenigün, C. Yilmaz, and A. Ulrich (Eds.): ICTSS 2013, LNCS 8254, pp. 114–132, (Novembre 2013).

[9] Baudry, B., Le Traon, B., Sunyé, G.: « Testability analysis of a UML class diagram. » Proceedings of the 9th International Software Metrics Symposium. IEEE CS, (2003).

[10] Bou Nassif, A., Capretz, L.F., Ho, D.: « Software effort estimation in the early stages of the software life cycle using a cascade correlation neural network model », 13th ACIS International Conference on Software Engineering, Artificial intelligence, Networking and Parallel/distributed Computing, IEEE, (2012).

[11] Bou Nassif, A., Capretz, L.F., Ho, D.: « Calibrating Use Case Points », ICSE Companion'14, May 31-june 7, Hyderabad, India - Copyright ACM 978-1-4503-2768—8/68-8/14/05, (2014).

[12] Bradine, D., « Oops, there it is: object-oriented project size estimation », Enterprise Systems Journal (2000).

[13] Briand, L.C., Wust, J., Daly, J., Porter, V.: « Exploring the Relationship between Design Measures and Software Quality in Object-Oriented Systems. » Journal of Systems and Software, 51(3), 245–273, (2000).

[14] Bruntink, M., Van Deursen, A., « Predicting class testability using object-oriented metrics », in Proceedings of the 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM '04), pp. 136–145, Septembre 2004.

[15] Bruntink, M., Van Deursen, A., « An empirical study into class testability », Journal of Systems and Software, vol. 79, no. 9, pp. 1219–1232, 2006.

[16] Carbone, M., Santucci, G., « Fast&&Serious: a UML based metric for effort estimation, in: Proceedings of the 6th International ECOOP Workshop on Quantitative Approaches in Object-oriented Software Engineering », 2002, pp. 35–44.

[17] Carroll, E.R. : « Estimating Software Based on Use case Points », OOPSLA'05, San Diego, California, USA, Octobre 16-20 (2005).

[18] Chaudhary, P., Yadav, C.S.: « An Approach for Calculating the Effort Needed on testing Projects. » International Journal of Advanced Research in Computer Engineering & Technology, 1(1), (March 2012).

[19] Diev, S. : « Software estimation in the maintenance context », ACM SIGSOFT Software Engineering Notes, 31(2), Mars 2006.

[20] El Emam, K.: A Methodology for Validating Software Product Metrics. National Research Council of Canada NRC/ERB 1076, (2000).

[21] Fan, W., Xiaohu, Y., Xiaochun, Z., Lu, C.: « Extended Use Case Points Method for Software Cost Estimation. » International Conference on Computational Intelligence and Software Engineering, (2009).

[22] Gupta, V., Aggarwal, K.K., Singh, Y.: « A Fuzzy Approach for Integrated Measure of Object-Oriented Software Testability. » Journal of Computer Science, 1(2), (2005).

[23] Gyimothy, T., Ferenc, R., Siket, I.: Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. IEEE TSE 3(10), 897–910, (2005).

[24] Jacobson, I., Christerson, M., Jonson, P., Overgaard, G. : « Object-Oriented Software Engineering : A Use Case Driven Approach », Addison-Wesley, (1993).

[25] Karner, G.: « Resource Estimation for Objectory Projects », Objective systems, (1993).

[26] S. Kim, W. Lively, D. Simmons, « An effort estimation by UML points in the early stage of software development, » Proceedings of the 2006 International Conference on Software Engineering Research & Practice, 2006, pp. 415–421.

[27] Larman, C., « Applying UML and Design Patterns, An introduction to object-oriented analysis and design and the unified process », Prentice Hall, 2004.

[28] Marcus, D.P., Ferenc, R.: Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems. IEEE Transactions on Software Engineering, 34(2), 287–300, (2008).

[29] A.F. Minkiewicz, « Measuring object oriented software with predictive object points », PRICE Systems (1997).

[30] Mohagheghi, P., Anda, B., Conradi, R.: « Effort Estimation of Use Cases for Incremental Large-Scale Software Development. » Proceedings of the International Conference on Software Engineering, ICSE'05, May 15-21, 2005, St. Louis Missouri, USA, (2005).

[31] Nagheshwaran, S.: Test Effort Estimation Using Use Case Points. In: Quality Week 2001, San Francisco, California, USA (2001).

[32] Ochodek, M., Nawrocki, J., Kwarciak, K.: « Simplifying effort estimation based on Use Case Points. » Information and Software Technology, 53, 200–213, (2011).

[33] Robiolo, G., Orosco, R.: « Employing use cases to early estimate effort with simpler metrics. » Innovations in Systems and Software Engineering, 4, (2008).

[34] Robiolo, G., Badano, C., Orosco, R.: « Transactions and Paths: two use case based metrics which improve the early effort estimation. » Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, (2009).

[35] Singh, Y., Kaur, A., Malhota, R.: « Predicting testability effort using artificial neural network. » Proceedings of the World Congress on Engineering and Computer Science, CA, USA, (2008).

[36] Singh, Y., Kaur, A., Malhotra, R.: « Empirical validation of object-oriented metrics for predicting fault proneness models. » Software Quality Journal 18(1), 3–35, (2009).

[37] Singh, Y., Saha, A. « Predicting testability of eclipse: a case study. » Journal of Software Engineering, 4(2), (2010).

[38] Yi, Q., Bo, Z., Xiaochum, Z., « Early Estimate the Size of Test Suites from Use Cases. » Proceedings of the 15th Asia-Pacific Software Engineering Conference, IEEE CS, (2008).

[39] Xiaochun, Z., Bo, Z., Fan, W., Chen Lu, Q.Y.: Estimate Test Execution Effort at an Early Stage: An Empirical Study. In: International Conference on Cyber World. IEEE CS, (2008).

[40] XLSTAT, http://www.xlstat.com.

[41] Zhou, Y., Leung, H.: « Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. » IEEE Transactions on Software Engineering, 32(10), 771–789, (2006).

[42] Zhou, Y., Leung, H., Song, Q., Zhao, J., Lu, H., Chen, L. and Xu, B. (2012). An in-depth investigation into the relationships between structural metrics and unit testability in OOS. Information Sciences, 55(12) Science China, (2012).

[43] Zhou, Y., Yang, Y., Xu, B., Leung, H., Zhou, X., « Source code size estimation approaches for object-oriented systems from UML class diagrams: A comparative study, » Information and Software Technologie, vol. 56, 2014.