Caractérisation des tests logiciels

Études empiriques à l'aide d'algorithmes d'apprentissage artificiel

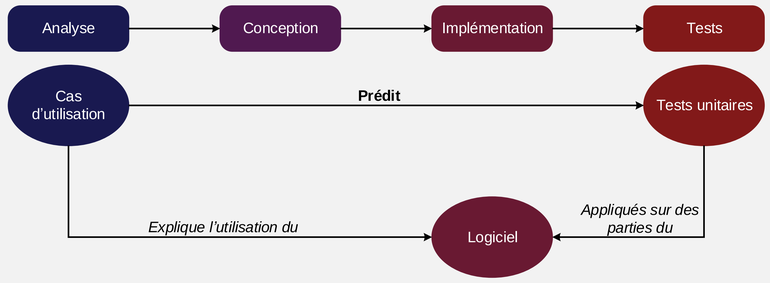
Alexandre Ouellet, Mourad Badri
Laboratoire de recherche en génie logiciel
Université du Québec à Trois-Rivières | Département de Mathématiques et d'Informatique

Introduction

La phase de tests occupe une place importante au sein du processus de développement du logiciel. Cette phase peut nécessiter plus de 40% du budget alloué au développement. C'est pourquoi qu'il est fréquent que les développeurs priorisent une partie des tests et que toutes les classes ne soient pas testées lors du développement. À l'heure actuelle, plusieurs méthodes de priorisation des tests reposent sur le jugement des concepteurs. Ces méthodes demandent aux concepteurs d'assigner diverses côtes aux classes et ils se servent de ces côtes pour choisir les tests à prioriser. Notre objectif est de produire un modèle qui quantifie « objectivement » les classes pour choisir sur lesquelles orienter l'effort de test.

Des objectifs secondaires de cette recherche sont de trouver la méthode la plus simple possible et celle offrant des résultats le plus tôt possible dans le processus du développement. Pour cela, les métriques mesurées dans les expérimentations qui suivent proviennent des cas d'utilisations, l'un des premiers documents écrits.

Figure 1 : Étapes et artefacts du développement logiciel



Les travaux présentés visent à comparer l'efficacité de deux méthodes de prédictions basées sur des métriques des cas d'utilisation. La méthode UCP [1] qui caractérise les cas d'utilisation à travers divers facteurs auxquels sont associées des pondérations. Bien que connue dans le domaine, le calcul de cette métrique est assez technique et fait intervenir le jugement et l'expertise, qui sont assez subjectifs, de la personne le réalisant. L'autre méthode proposée est d'utiliser une combinaison de métrique des cas d'utilisation plus simple à calculer et ne contenant pas de facteurs subjectifs[2].

Algorithmes d'apprentissage

Des études statistiques ont été préalablement menées sur les données. Les résultats portent à croire que l'usage d'une combinaison de métriques des cas d'utilisation permet d'obtenir de meilleures prédictions, tout en étant plus simple. Par ailleurs, la nature des données les rend appropriées aux algorithmes d'apprentissage artificiel. Ces algorithmes, selon nos hypothèses, seraient plus à même de décrire la relation entre les cas d'utilisation et les tests unitaires.

Plusieurs algorithmes ont été testés sur les données à l'aide du logiciel *Tanagra*. Les résultats présentés ici sont ceux de l'algorithme avec lequel les meilleurs résultats ont été obtenus, l'algorithme k-NN (k nearest neighbours). Pour un certain exemple à classifier, cet algorithme lui assigne la classe prédominante dans son voisinage (5 plus près voisins). La distance des voisins est mesurée à l'aide d'une métrique particulière, l'*Heterogeneous Eucledian-Overlap Metric* (HEOM).

Équation 1 : Distance HEOM pour des vecteurs de $\mathbb{R}^m$

$$HEOM(x,y) = \sqrt{\sum_{\alpha=1}^{m} \left(\frac{|x-y|}{range_\alpha}\right)^2}$$

Taux d'erreur

La mesure du taux d'erreur d'une méthode d'apprentissage artificiel indique la précision de la méthode. Il existe diverses mesures de l'erreur qui apporte des précisions particulières sur le comportement de l'algorithme par rapport aux données.

Les trois taux d'erreur évalués dans les expérimentations sont :

- **Resubstitution** : taux d'erreur en appliquant la méthode sur les données ayant servies à l'apprentissage.
- **Bootstrap** : les exemples servant à l'apprentissage sont sélectionnés grâce à un tirage avec remise. Les exemples qui demeurent non sélectionnés sont ceux sur lesquels l'erreur est mesurée.
- **Validation croisée** : Chaque segment des données (généralement 10%) est exclu à tour de rôle de l'ensemble d'apprentissage. L'apprentissage est effectué sur les données non exclues et l'erreur est évaluée sur les données qui ont été exclues. Le taux d'erreur est le cumul des taux mesurés sur les données exclues.

Expérimentations

Méthodologie

Les expérimentations portent sur cinq projets Java *open source*. Deux métriques ont été testées pour la prédiction, soient la métrique TLOC qui mesure le nombre de lignes de code présentes dans un test unitaire *JUnit*. L'autre métrique, TASSERT, compte le nombre d'assertions présentes dans un test unitaire *JUnit*. Une assertion est une opération qui vérifie si une donnée prend ou non une valeur parmi celles attendues. L'algorithme k-NN fait une prédiction d'appartenance à une classe (variable discrète) et non une variable continue, qui est le format dans lequel TLOC et TASSERT sont exprimés. Les données ont dû être regroupées par classes. Le regroupement s'est fait en deux classes, l'une groupant les données inférieures à la moyenne, l'autre les supérieures.

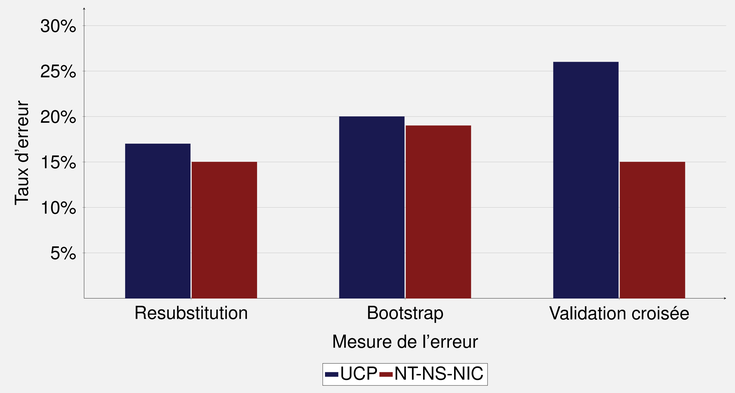
Pour les métriques servant à la prédiction, les trois étudiées sont :

- **NT** : (*Number of transactions*) Nombre d'interactions dans un cas d'utilisation entre un acteur et le système.
- **NS** : (*Number fo scenarios*) Nombre de scénarios distincts possibles dans un cas d'utilisation.
- **NIC** : (*Number of involved classes*) Nombre de classes conceptuelles impliquées dans la réalisation du cas.

Pour le nombre d'assertions, la combinaison de seulement deux métriques, NT et NIC, permet d'obtenir de meilleurs résultats.

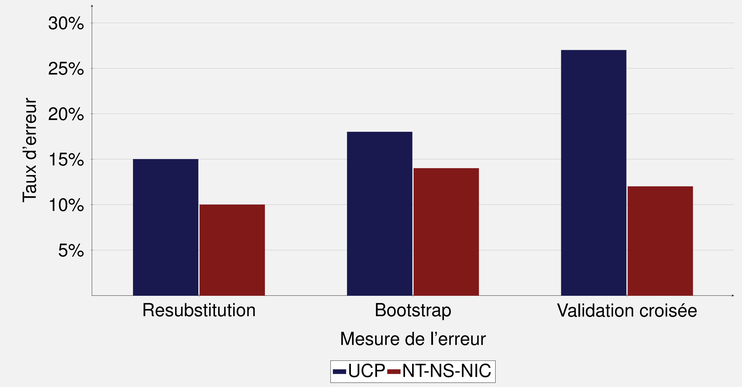
Résultats

Graphique 1 : Prédcition du nombre de lignes de code des tests



Expérimentations (*suite*)

Graphique 2 : Prédcition du nombre d'assertions dans les classes de tests



Analyse

Sur les données étudiées, la combinaison des métriques des cas d'utilisation offre de meilleurs résultats que l'utilisation de la méthode UCP seule. On peut également remarquer que le nombre d'assertions tend à être prédit avec plus d'exactitude que le nombre de lignes de code des tests. L'une des causes possibles est que le nombre de lignes dans un code varie beaucoup selon le programmeur qui l'a écrit, donc cette métrique possède un biais dû à sa nature.

Aussi, on remarque le biais présent dans l'erreur par rebustitution avec la méthode UCP, mais la combinaison de métriques des cas d'utilisation semble moins soumise à ce biais, car les méthodes moins biaisées (bootstrap, validation croisée) gardent un taux d'erreur du même ordre.

Conclusion et travaux futurs

En conclusion, la combinaison de métriques introduites NT-NS-NIC permet, grâce aux algorithmes d'apprentissage artificiel d'effectuer de meilleures prédictions sur les caractéristiques des tests unitaires que la méthode UCP. De plus, les métriques NT, NS et NIC sont plus simples à mesurer et plus objectives que la méthode UCP.

Toutefois, comme pour toutes les méthodes basées sur les cas d'utilisation, il faut tenir compte de leur subjectivité pour nuancer ces résultats. En effet, deux analystes pourraient développer des cas d'utilisation différents, au niveau des métriques, pour une même tâche. Aussi, le modèle développé est valide uniquement sur les données ayant servi à l'apprentissage. Un modèle construit à l'aide d'un plus grand nombre d'exemples est nécessaire pour qu'il puisse être réutilisé sur d'autres projets.

Finalement, la classification des métriques des tests est aussi un aspect du développement qui reste à travailler. D'abord, dans une optique de réutilisation sur d'autres projets, car chaque projet possède des caractéristiques singulières et la classification doit en tenir compte. Puis, pour que le modèle développé soit compréhensible et facilement utilisable par un analyste humain, afin qu'il puisse s'y référer pour le guider dans la conception, la planification et la conduite des tests.

Références

[1] Karner Gustave, *Resource Estimation for Objectory Projects*, 1993.
[2] Badri Mourad, Badri Linda, Flageol William, *Symplifying Test Suites Size Prediction Using Use Case Metrics : An Empirical Comparaison*, 2013.
[3] Roongruangsuwan Siripong, Daengdej Jirapun, *Test Case Prioritization Techniques*, 2010.