Améliorer la qualité de son package R avec l'intégration continue

Géraud Dugé de Bernonville

Valtech_

1, rue Dalayrac 31000 Toulouse

geraud.dugedebernonville@gmail.com

Mots clefs : Méthodes agiles, qualité, intégration continue

Lorsque l'on publie un package R, il est important de fournir une application qui réponde à un certain niveau de qualité. L'intégration continue est une démarche du développement logiciel permettant de garantir la qualité du code source d'une application, et ce à chaque modification du code source[1]. Elle repose sur l'exécution de tests automatisés, développés en même temps que le code source du logiciel. Cette démarche repose sur plusieurs piliers:

- un gestionnaire de source commun
- une construction automatisée de l'application
- l'existence de tests automatiques au sein du code source
- l'exécution du processus à intervalles réguliers

Issue des méthodes agiles (notamment l'eXtreme Programming[2] et le Test Driven Development[3]), l'intégration continue permet ainsi:

- de valider le bon fonctionnement du logiciel
- de limiter les non-régressions en cas d'évolution du logiciel
- de garantir la reproductibilité de la construction du logiciel

Cette démarche est soutenue par de nombreux outils propriétaires et open-source. Parmi les plus utilisés[4], nous pouvons citer Jenkins, CircleCI, Bamboo ou TravisCI. Chaque outil fonctionne sur le même principe : à savoir l'exécution régulière de tests automatisés sur la base du code source de l'application. Ensuite, des rapports sont produits à chaque exécution afin de recenser le statut des tests et éventuellement des indicateurs sur la qualité du code. En cas d'erreur, une notification peut ainsi être émise aux contributeurs du projet afin d'être réactifs et de pouvoir régler les problèmes le plus tôt possible.

L'objectif de ce *Lightning Talk* est de montrer comment mettre en oeuvre une chaîne d'intégration continue afin d'améliorer la qualité de votre package R. Pour cela nous utiliserons le package *testthat* pour l'implémentation des tests unitaires et l'outil d'intégration continue TravisCI. Ces deux outils sont pris à titre d'exemple mais la démarche présentée reste applicable avec d'autres outils.

Références

[1] Fowler, M., & Foemmel, M. (2006). Continuous integration. Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf, 122.

[2] Beck, K. (1999). Embracing change with extreme programming. Computer, 32(10), 70-77.

[3] Williams, L., Maximilien, E. M., & Vouk, M. (2003, November). Test-driven development as a defect-reduction practice. In Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on (pp. 34-45). IEEE.

[4] Comparison of continuous integration software. (n.d.). In Wikipedia. Retrieved April 15, 2016, from https://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software