Simulation à base d'agents de l'évolution d'un logiciel Open Source

Quentin BAILLEUL Romain PHILIPPON

3 février 2015

Table des matières

1 Introduction **2**

2 Simulation de base **3**
2.1 Les Modules 3
2.2 Les Prérequis 4
2.3 Les Développeurs 4

3 Simulation modifiée **5**
3.1 Module 5
3.2 Développeur 6

4 Comparaison des résultats **6**

5 Conclusion **6**

1 Introduction

L'informatique est une discipline qui évolue rapidement et il est nécessaire de comprendre les processus évolutifs qui prévalent dans de nouvelles formes de développement de logiciels notamment dans un environement Open source.

Beaucoup de travaux et d'analyses sont fournies à propos de la création et de l'évolution de logiciels propriétaires, à l'inverse de l'étude sur les logiciels Open Sources.

Ceci suggère que les théories existantes sur l'évolution des logiciels sont différentes pour les logiciels open source.
Dans cette optique les recherches de N Smith, A Capiluppi, et J Fernandez-Ramil tendent à améliorer voir redéfinir ces théories.

Leur article utilise les théories de l'évolution du logiciel afin de reproduire et d'expliquer les observations empiriques d'un ensemble de systèmes Open Source. Pour cela, l'article se repose sur une simulation multi-agents de développement logiciel dans un environement Open Source.

Le but de ce rapport sera de modifier la simulation en ajoutant des paramètres afin de rajouter du réalisme afin de comparer les résultats pour déterminer si les règles du comportement suivent la règle du Rasoir d'Ocam, c'est à dire qu'il suffit d'un ensemble de règles réduits au strict minimum pour reproduire l'évolution du développement d'un logiciel open-source. Ou au contraire, si les résultats mis en lumière dans ce rapport tendent à prouver l'inverse.

Dans un premier temps, nous présenterons le modèle de N. Smith, A. Capiluppi et J. Fernandez-Ramil, puis dans une seconde partie s'attardera sur la version modifiée et les nouveautés implémentées. Puis dans la troisième partie, nous comparerons les résultats obtenus avec les résultats du modèle de base. Puis nous terminerons par une conclusion.

2 Simulation de base

FIGURE 1 – Un exemple de développement de logiciel Open Source simulé. Les carrés sont des patches de code. Les cercles noirs sont des prérequis non satisfaits.

La simulation du projet de N Smith, A Capiluppi, et J Fernandez-Ramil est basée sur trois types d'agents :

2.1 Les Modules

Un module représente une fonctionnalité du logiciel. Chaque module comporte un score de complexité et un score de santé (c'est-à-dire la stabilité du code). Ces scores varient selon l'action effectuée par le développeur. Un module peut créer des agents *prérequis* [2.2] si ses voisins (voisinage de Moore de 1) ne sont pas des modules.

2.2 Les Prérequis

Les prérequis symbolisent une spécification logicielle qui attend qu'un module [2.1] soit crée par développeur [2.3].

2.3 Les Développeurs

Un développeur se définit par son score d'ennui. Les développeurs se promènent aléatoirement dans l'espace de simulation. À chaque déplacement un développeur peut effectuer une des actions suivantes :

1. Si il rencontre un prérequis [2.2] le développeur produit à l'emplacement de celui-ci un module [2.1]. Le prérequis meurt à l'issue de cette opération.
2. Si il se trouve sur un module et que la santé et la complexité ne dépasse pas un seuil donné, le développeur factorise le module. Cela induit une réduction de la complexité de ce dernier.
3. Si il se trouve sur un module et qu'il ne peut pas le factoriser, il va le développer. C'est-à-dire qu'il augmente la complexité et la santé si et seulement si la complexité de ce module ne dépasse pas un seuil donné
4. Si il ne peut pas réaliser une des actions présentées ci-dessus alors il incrémente son score d'ennui de 1

Dès que le développeur achève l'une de ces actions, il meurt si son score d'ennui est supérieur à un seuil. Sinon il se déplace en se dirigeant dans une direction aléatoire.

3 Simulation modifiée

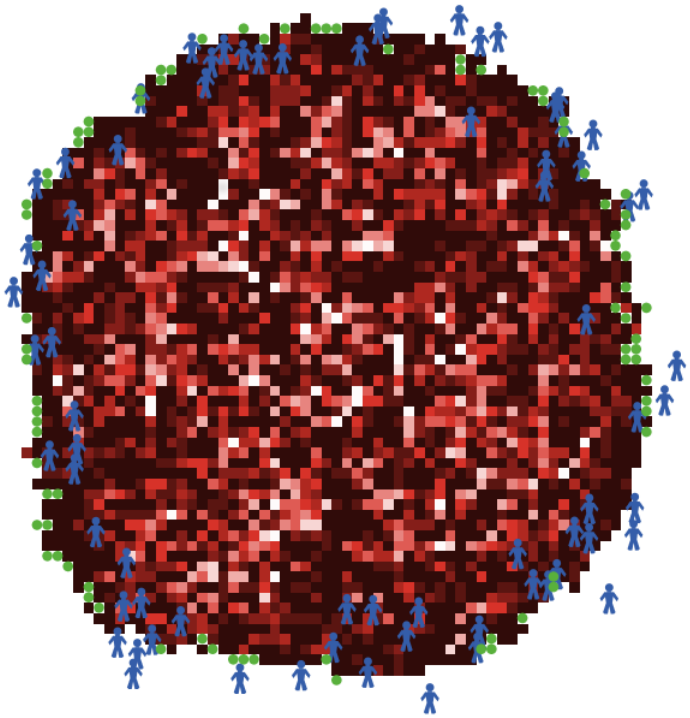

FIGURE 2 – Un exemple de développement de logiciel Open Source simulé via le modèle présenté dans ce papier.

Pour apporter du réalisme à la simulation, plusieurs paramètres ont été introduits :

3.1 Module

Un module est maintenant déterminé par un langage de programmation. Il ne peut pas être développé ou factorisé si par un développeur ne connait pas ce langage. En outres, le module a aussi une chance d'être déstabilisé par le développeur (sa santé baissera) proportionnellement à son expérience. Plus un développeur est expérimenté moins il aura de chance de déstabiliser le code. Et quand bien même un développeur expérimenté introduit du code néfaste il ne sera pas aussi instable qu'un code introduit par un développeur inexpérimenté.

3.2 Développeur

Un développeur est maintenant doté d'une expérience. Plus il développe ou factorise du code, plus son expérience augmente. L'expérience détermine la capacité plus ou moins importante à modifier le code (modification de la santé et/ou de la complexité du code). D'ailleurs, un développeur n'est plus en mesure de développer un module si celui-ci contient des *prérequis* dans son entourage. Le but est de modéliser les dépendances entre modules.

Le développeur est aussi doté d'une panoplie de langages connus qui lui permet de travailler sur les modules dont il connait le langage.

4 Comparaison des résultats

5 Conclusion