HAL
open science

Utilisation des logiciels libres comme outil de compréhension de programmes

Karine Mordal, Ivaylo Ganchev

To cite this version:

Karine Mordal, Ivaylo Ganchev. Utilisation des logiciels libres comme outil de compréhension de programmes. Sciences et technologies de l'information et de la communication (STIC) en milieu éducatif, 2013, Clermont-Ferrand, France. edutice-00875624

HAL Id: edutice-00875624

https://edutice.hal.science/edutice-00875624v1

Submitted on 22 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Utilisation des logiciels libres comme outil de compréhension de programmes

Karine Mordal, Ivaylo Ganchev
kmordal@gmail.com, ivaylo.ganchev@univ-paris8.fr

UFR MITSIC, Université Paris 8

Résumé. Nous présentons un cours, donné depuis deux ans à l'université Paris 8 dans le cadre de la licence informatique. Dans ce cours nous avons choisi une approche pratique reposant sur les logiciels libres et leur modèle de développement pour enseigner aux étudiants de troisième année des techniques permettant de comprendre et de modifier du code existant. Nous expliquons en quoi cette approche diffère d'un enseignement conventionnel : il permet de travailler de manière concrète sur du code opérationnel, de respecter des règles et des processus métiers et d'intégrer une communauté active de développeurs. Nous montrons que cette démarche permet aux étudiants d'acquérir une pratique et une autonomie qui facilitent leur intégration dans une équipe de développement.

Mots-clés: Logiciels Libres, informatique, compréhension de programmes

Introduction

La maintenance informatique prend une place très importante dans les métiers du développement de logiciels (Swebok, 2012). Celle-ci s'est accrue en même temps que la complexification des logiciels et le besoin d'en assurer la pérennité dans le temps. Il n'est donc pas suffisant pour un développeur de maîtriser stricto sensu un langage de programmation et des principes algorithmiques, il doit également maîtriser les techniques de compréhension de code pour travailler à la maintenance de logiciels[1] (Glass, 2001 ; Grubb & Takang, 2003 ; Lehman, 1996). De plus nous observons une évolution des pratiques de développement d'applications qui oblige à posséder un savoir-faire reposant sur des processus métier spécifiques, des règles, normes et méthodes de programmation ainsi que des outils de développement spécifiques, de plus en plus complexes (IEEE, standard board, 1998). Pour ces raisons, nous avons mis en place à l'université Paris 8 dans le cadre de la licence informatique un cours intitulé « Compréhension de programmes » au niveau de la dernière année d'étude. Ce cours aborde les techniques de compréhension de code par un apprentissage concret en confrontant les étudiants à des projets complets leur permettant d'acquérir une expérience en matière de maintenance de code source. Pour ce faire nous avons choisi d'utiliser les logiciels libres du fait de la mise à disposition de leur code source et de la possibilité de modifier ce code. De plus, ceci nous permet de travailler avec des logiciels de taille conséquente, bien documentés et qui suivent des processus de développement définis (Hess & Paulson, 2012 ; Pedroni et al., 2007).

Après avoir expliqué les visées pédagogiques et notre choix des logiciels libres, nous détaillons le déroulement du cours et nous présentons les résultats obtenus depuis deux ans. Nous expliquons ensuite en quoi ce cours répond à nos attentes mais aussi quels sont les écueils à éviter pour que ce type de cours puisse fonctionner correctement.

Le cours de compréhension de programmes

Pour enseigner la compréhension de programmes il faut prendre en compte la manière dont sont développés les logiciels aujourd'hui. En effet, ils reposent sur des méthodes de développement

[1] Nous reprenons la définition des standards IEEE qui désignent par ce terme toute activité exercée sur un logiciel après sa livraison, faite dans le but de corriger des bogues, améliorer les performances ou certaines parties du système (y compris la documentation, la localisation, l'internationalisation et les procédures propres au logiciel), l'adapter à des modification logicielles ou matérielles de l'environnement (IEEE, standard board, 1998).

spécifiques, des normes et des règles de programmation à respecter et font également appel à des outils de développement qu'il faut maîtriser. La recherche d'informations est également une tâche importante dans la maintenance de code. En effet, pour comprendre l'organisation d'une application et les techniques utilisées il faut savoir lire la documentation technique mais également communiquer avec l'équipe de développement : formuler une question, utiliser tout document susceptible d'être utile. Il est donc indispensable de savoir comment utiliser les informations mises à disposition et de posséder les outils pédagogiques qui permettent de s'approprier ces informations. Nous avons donc choisi d'orienter le cours de compréhension de programmes pour permettre aux étudiants d'appréhender la maintenance de code dans sa globalité : enseigner comment toutes ces techniques abordées séparément pouvaient s'articuler autour d'une même application pour préparer les étudiants à aborder le monde industriel du développement.

Pour y parvenir il nous fallait utiliser en support de cours des applications développées selon tous ces principes, utilisant ces différents outils et reposant sur l'ensemble de ce savoir-faire : des logiciels généralement de taille conséquente, répartis dans plusieurs répertoires et utilisant souvent différents langages de programmation et différentes techniques. Nous nous sommes donc tournés vers des applications existant déjà sous licence libre. Les expériences de Lipsa et al. (2011), Hislop et al. (2009) et Ellis et al. (2011) confirment que le fait de travailler sur des logiciels grandeur nature prépare mieux les étudiants à l'environnement professionnel. Le choix de travailler avec des logiciels en situation réelle a également été conforté par les propos de Dionisio et al. (2007), Pedroni et al. (2007) et Hess et Paulson (2010) qui partagent le fait que travailler sur des logiciels réels, utilisés en situation de production motive les étudiants beaucoup plus que les logiciels expressément créés pour les fins pédagogiques d'un cours.

Nous avons identifié quatre manières principales à contribuer à un programme à code ouvert : correction de bug, développement de nouvelle fonctionnalité, traduction du logiciel, de sa documentation, écriture de pages de documentation (Marmorstein, 2011 ; Patterson, 2006). Il existe bien-sûr des approches plus originales d'un point de vue pédagogique comme le portage d'une application existante sous licence libre (German, 2005), l'usage de méta-données à des fins pédagogiques (Squire et al., 2009), la découverte d'un bogue, le développement from scratch d'une application en utilisant les méthodes de travail propres au monde du libre (Basdarakas et al., 2006). Dans le cadre de notre cours il nous a semblé opportun de limiter cette liste au deux premiers éléments : le développement d'une fonctionnalité et la correction de bogue.

Le choix des logiciels libres

Le fait d'utiliser les logiciels libres comme support de cours est motivé par une première raison d'importance : le libre accès au code source de ces logiciels. Les logiciels libres étant distribués sous des licences garantissant l'accès et la possibilité de modifier le code source selon des règles définies[2], nous avons établi que ces logiciels serviraient de support d'enseignement. De plus, la palette de choix proposés en matière de logiciels libres nous donne accès à une banque de données quasi-inépuisable pour notre cours et d'une très grande diversité que ce soit dans les langages ou les techniques utilisés[3].

La seconde motivation dans le choix des logiciels libres repose sur la qualité du code développé. Selon une étude récente de Coverity (Coverty scan report, 2012) les logiciels libres sont en moyenne de meilleure qualité que les standards reconnus par l'industrie. D'autre part, ils suivent des processus de validation et de déploiement à respecter pour que les modifications effectuées puissent être intégrées dans une version ultérieure du logiciel, ce qui correspond également à un des objectifs du cours. De plus, comme le soulignent Hess et Paulson (2010), les outils de gestion de projet, les environnements de développement et les méthodes de travail du monde du libre sont les raisons pour lesquels une telle expérience peut être considérée comme un vrai atout dans la recherche d'emploi. Le fait de travailler et échanger (le plus souvent en anglais) dans des communautés internationales avec

---

2 La licence GPL (General Public Licence — la licence qui est historiquement à l'origine du mouvement des logiciels libres) permet de disposer du code source d'une application et de le modifier sous respect de continuer à distribuer le code sous la même licence (Free Software Foundation, 2007).

3 le site Framasoft http://www.framasoft.net/rubrique2.html recense plus de 1600 logiciels libres classés dans 14 rubriques.

des développeurs confirmés ajoute à cet aspect professionnalisant, ce qui correspond à une autre motivation dans notre choix.

Un autre objectif du cours consiste à permettre aux étudiants d'acquérir des méthodes de recherche d'information et de communication avec une équipe de développement. La communauté du libre nous est là encore apparue idéale pour notre cours. En effet, ces communautés ont mis en place des processus de communication à travers les wiki, les FAQ, les forums, IRC, d'une grande qualité et souvent cités en exemple. De plus, la communauté est réactive et accessible. Nous pouvions donc mettre en relation les étudiants avec la communauté en situation réelle. D'ailleurs, comme témoigne Marmorstein (2011) un des aspects importants dans l'usage des logiciels libres dans le cadre d'un enseignement informatique repose sur les apprentissages en matière de communication, d'argumentation, de formulation grâce aux interactions avec la communauté.

Enfin, Goldweber (Goldweber et al., 2011) déclare que pour motiver les étudiants à intégrer la filière informatique il faut les sensibiliser aux effets de l'informatique sur la société et des aspects éthiques de certains choix technologiques. C'est précisément ce que font les enseignants à l'origine du projet HFOSS (projets libres humanitaires) qui essaient de sensibiliser les étudiants de différentes origines à l'informatique à travers les visées humanitaires de leur projet (Hellis et al., 2007 ; Hislop, 2009).

Le déroulement du cours

Dans cette partie nous détaillons comment se déroule le cours de compréhension de programmes selon un schéma de cours de type travaux pratiques plutôt que selon un schéma de type cours magistral.

Nous avons, préalablement à la mise en place du cours, pris contact avec différents membres de la communauté du libre pour leur demander s'ils étaient intéressés par notre projet. Notre but était de trouver des interlocuteurs qui faciliteraient la prise de contact et les échanges tout en fournissant des conseils techniques précis et détaillés. Nous avons recensé des projets structurés selon nos attentes. En effet, il existe différentes approches quant au choix des projets : soit une liste préétablie (Ellis et al., 2011), soit le libre choix laissé aux étudiants (German, 2005 ; Marmorstein, 2011), voire une combinaison des deux (Pedroni et al., 2007). L'expérience de la première année nous a conforté dans l'établissement d'une liste prédéfinie (comme discuté dans la section Résultats).

Nous distinguons cinq phases[4] qui s'enchainent à différents rythmes selon les groupes :

1. la préparation qui consiste à répartir les étudiants et les projets. Les étudiants forment des groupes de quatre personnes qui doivent être homogènes. Nous privilégions l'entente entre les différents membres du groupe pour favoriser le travail et les échanges. Ils choisissent ensuite un projet et un travail à faire parmi une liste préalablement établie, en fonction de leur compétences techniques (les langages utilisés dans ces projets, les systèmes d'exploitation) et de leur connaissance dans le domaine du projet (son, web, image, jeux, etc). Ces choix sont validés par l'enseignant lors d'une discussion au cours de laquelle celui-ci s'assure que tous ont compris les enjeux et les buts ;
2. la mise en place des outils de développement et de l'environnement de travail. Le groupe doit envoyer un premier mail au tuteur pour prendre contact et lui demander les premiers conseils visant à installer de manière adéquate les outils de développement. Les groupes ont des outils différents à installer selon le logiciel et la plate-forme utilisée. Cette étape est validée par la compilation du code de l'application ;
3. le travail à effectuer. Pour y parvenir, les étudiants doivent mettre en oeuvre tous les moyens à leur disposition : les contacts avec le tuteur, les forums et autres sources d'information sur internet, les conseils des enseignants, les efforts et les échanges au sein du groupe. Chaque semaine le cours a lieu dans une salle machine et chaque groupe doit faire un point sur l'avancé de leurs travaux.

Les groupes sont encadrés de manière totalement personnelle : l'enseignant débat avec eux sur leurs difficultés et leur approche pour les résoudre. Il donne des conseils personnalisés et leur enseigne au cas par cas les méthodes de compréhension et d'analyse de code. Le premier travail consiste d'abord à comprendre l'architecture générale du projet pour ensuite se focaliser sur la partie

[4] Les phases que nous avons définies se rapprochent à celles définies par Ellis et al. dans (Ellis et al., 2011).

concernée par le travail à proprement parler . Chaque semaine le groupe doit également fournir un rapport écrit de ses avancés sur un wiki mis à leur disposition ;

4. la présentation des résultats. Pour cela, chaque groupe fournit un rapport détaillé de présentation de ses travaux et des méthodes employées pour parvenir à effectuer la tâche demandée . Le rapport doit mettre en avant à la fois le travail du groupe mais aussi les stratégies qu'il a mis en place pour y parvenir, que ce soit la communication avec la communauté autour du projet , les méthodes de fonctionnement au sein du groupe, etc. De plus, chaque groupe présente ensuite son travail dans un exposé oral d'une quinzaine de minutes devant l'ensemble de la classe . Ces exposés ouvrent également à débat, certains étudiants posant des questions sur les méthodes utilisées par d'autres ou bien encore sur la manière dont la communication et la recherche d'informations s'est déroulée

5. la validation des acquis . Il a tout de suite été établi que la note ne pouvait pas dépendre totalement de la réussite de la tâche à effectuer , entre autre parce que cette réussite ne dépend pas uniquement des étudiants eux -mêmes (comme nous en discutons dans la section Résultats). Nous avons donc mis en place le rapport et la présentation de fin de semestre pour matérialiser l'aboutissement du cours. Cependant, la validation ne repose pas uniquement sur ces documents qui permettent de se voir attribuer une note calculée sur l'ensemble des travaux du semestre . Pour établir un tel contrôle continu , avec des groupes qui travaillent sur des projets personnalisés et parfois totalement différents les uns des autres , nous avons établi la liste des tâches qu'il fallait prendre en compte, afin de constituer la note finale des étudiants.

Résultats

Les tableaux 1, à 4 résument les principaux résultats après d eux années d'enseignement du cours . Le tableau 1 détaille la présence des étudiants : sur les 40 présents au début du cours en 2011 seul deux ont abandonnés tandis qu'en 2012 tous les étudiants inscrits sont allés au bout du cours. De fait, les groupes formés au début ont tous validé le cours avec un mémoire et une présentation orale malgré les difficultés éventuelles rencontrées : les groupes n'ayant pas pu aboutir leur projet ont tout de même eu matière à présenter leurs travaux. Rappelons ici que les étudiants de licence peuvent choisir entre plusieurs options. Pour ce faire ils s'inscrivent souvent dans plusieurs cours et ne gardent que les cours qui correspondent à leurs attentes. De fait, un tel niveau de présence en fin de cours est exceptionnel.

Table 1. Présence dans le cours

	2011		2012	
	Début du cours	**Fin du cours**	**Début du cours**	**Fin du cours**
Etudiants présents	40	38	22	22
Groupes présents	11	11	8	8
Groupes notés	X	11	X	8
Groupes au dessus de la moyenne	X	9	X	8

Le tableau 3 répartit les étudiants en fonction de ce qu'ils ont exprimé dans un questionnaire remis à la fin du semestre . Nous leur avons demandé de comparer leur ressenti dans le cours par rapport à un cours classique . Les résultas, exprimés en pourcen tage, fusionnent les deux années . En grande majorité les étudiants sont restés motivés jusqu'à la fin du semestre et ont trouvé le cours plus motivant et plus encourageant qu'un cours classique. En revanche, le résultat concernant la frustration est beaucoup moins tranché . Nous interprétons ce résultat en le mettant en corrélation avec les résultats obtenus sur les projets . En effet, les groupes ayant menés à bien leur tâche et ayant eu une bonne communication avec leur tuteur ont été satisfaits et se sont déclarés enthousiastes jusqu'à la fin du semestre. Les groupes s'étant retrouvés confrontés à une difficulté technique ou de compréhension

de code et qui ne sont pas parvenus à trouver de solution rapidement , voire pas du tout, ont été les plus frustrés, voire même parfois totalement découragés.

Table 2. Etat des travaux demandés en fin de cours

	2011	2012
nombre total de projets	11	8
Projets aboutis	5	6
Projets plus avancés que prévus	0	
Projets moins avancés que prévus	6	9
Projets ayant aboutis à autre chose que prévu	2	1
Projets en difficulté	2	1
Rédaction hebdomadaire	10	8
Présence hebdomadaire	11	8

Le tableau 2 résume l'état des projets en fin de semestre par rapport aux objectifs fixés . En 2011, plus de la moitié des projets ne sont pas parvenus au bout de la tâche fixée au départ. Deux projets ont aboutis à autre chose que ce qui avait été fixé : les travaux étaient trop conséquents pour parvenir à finir le projet et les groupes se sont orientés sur un travail de recherche et de veille technologique qui a conduit à la rédaction d'un dossier de proposition. Deux projets se sont vus en grande difficulté : les étudiants ne sont pas parvenus à réaliser un travail satisfaisant pour le cours. En 2012, la grande majorité des projets a abouti. Les travaux menés ont même conduit deux groupes à aller au delà de ce qui était prévu au départ. Seul un groupe a eu une réelle difficulté due en grande partie au fait que les technologies utilisées pour le logiciel étudié n'étaient pas maîtrisées par le groupe. Cet écart entre les deux années traduit la différence fondamentale dans le choix des projets entre la première et la seconde année : en 2011 les projets et les tâches à effectuer ont été en grande partie choisis par les étudiants tandis que la seconde année ils ont choisi dans une liste définie au préalable

Table 3. Ressenti des étudiants en comparaison à un cours classique

	Etudiants
Plus motivant	77,00%
Moins motivant	9,00%
Plus intéressant	85,00%
Moins intéressant	5,00%
Plus encourageant	68,00%
Moins encourageant	12,00%
Plus frustrant	35,00%
Moins frustrant	43,00%

Table 4. Répartition des projets en fonction des difficultés rencontrées

	2011	2012
Communication tuteur	3	3
Mauvais choix de projet	5	1
Difficulté technique de la tâche	3	2
Trop grand volume horaire	4	1
Manque de travail de groupe	1	

Le tableau 4 répartit les projets en fonction des difficultés rencontrées. Certaines tâches se sont avérées impossibles à réaliser soit parce qu'elles nécessitaient plus de travail que prévu (dépassement du volume horaire du cours) soit parce qu'elles s'avéraient hors de portée techniquement des étudiants (nécessité d'une plus grande connaissance technique). Nous remarquons qu'il n'y a qu'un seul groupe qui n'est pas allé au bout du projet par manque de travail. L'amélioration des chiffres sur les deux années confirme l'importance critique du choix des projets. La communication avec les tuteurs contrairement à ce que les chiffres peuvent laisser apparaitre s'est améliorée d'une année sur l'autre : en 2011 il s'agissait de trois tuteurs différents tandis qu'en 2012 il s'agissait du même tuteur ayant à sa charge trois groupes.

En plus de ces résultats, nous donnons une adresse qui recense la totalité des projets sur les deux années --- http://www.bocal.cs.univ-paris8.fr/dokuwiki/doku.php?id=cours:comprehension --- et nous détaillons ici quelques exemples significatifs de projets, que ce soient par leur déroulement exemplaire ou parce qu'ils représentent les écueils à éviter :

- Certains projets ont été une véritable réussite, tel que le projet 18[5]. Les étudiants ont découvert concrètement les processus de développement depuis l'installation des outils nécessaires jusqu'à la soumission de corrections et les phases de tests. Ils ont bénéficié d'une aide personnalisée avec un tuteur très réactif.
- Le projet 21 a également été une réussite. Le tuteur organisait des réunions très régulières avec le groupe qu'il a rencontré physiquement à maintes reprises, ce qui a renforcé d'autant la communication.
- Les projets 13 et 19 n'ont pas abouti du fait du manque de maturité du cours dans le choix des projets. En effet, le code du logiciel étant de grande qualité, associé à un tuteur motivé, nous avons pensé que celui-ci pourrait être prometteur pour le cours. L'échec est venu de part le choix des travaux assignés aux groupes qui nécessitaient des compétences techniques supérieures au niveau des étudiants. Dès que nous nous sommes aperçus que ces groupes ne pourraient rendre un travail satisfaisant nous les avons orienté sur la rédaction d'un rapport détaillé de leur recherche de des pistes explorées. Ceci a permis aux étudiants de pérenniser leur travaux mais ils ont tout de même été déçus de ne pouvoir réaliser les tâches qui leur avaient été assignées au départ.
- Les projets 17 et 11 ont été une erreur. Le code pas assez mûr ne cessait de changer plus rapidement que les étudiants n'avaient le temps d'intégrer les modifications précédentes.
- Le projet 28 a été une erreur de stratégie de notre part. Les étudiants n'avaient pas les compétences techniques requises et n'ont pas pu dépasser le stade de la compilation du code source. Même s'ils ont appris beaucoup en découvrant des outils comme maven ou eclipse ils ont tout de même été très déçus de ne pouvoir se consacrer à leur tâche.
- Sur le projet 4 le groupe n'est pas parvenu à implémenter totalement la nouvelle fonctionnalité dont ils avaient la charge mais ils ont découvert un bogue non répertorié par la communauté lors de l'installation du logiciel. Les étudiants, dans un premier temps, n'ont retenu que le négatif et il a fallu leur montrer que la découverte d'un autre bogue peut être considérée comme un réel travail.

[5] Nous avons remplacé les noms des projets par les numéros des groupes pour ne pas risquer d'assimiler les résultats obtenus avec le logiciel lui-même.

- Pour certains groupes qui se sont vus devenir contributeur du projet sur lequel ils ont travaillé, cela a agit comme un moteur leur permettant de reconsidérer leur capacité et ont pris confiance dans leur possibilité. Certains étudiants, suite au projet ont même décroché un stage de fin de licence.
- Les projets 10 et 11, reposant sur la même application , ont donné lieu à une collaboration entre les groupes, ce qui a renforcé leur motivation et permis d'augmenter la qualité du travail fourni.

A chaque fin du semestre nous avons remis un questionnaire aux étudiants (http://goo.gl/A70k7L) dont nous présentons dans la figure 1 quelques résultats significatifs. Ce questionnaire a été élaboré à partir de nos interrogations concernant le cours. Nous voulions pouvoir obtenir une mesure la plus objective possible des ressentis des étudiants et de ce qu'ils ont eu le sentiment d'avoir appris durant le cours. Nous avons également cherché à distinguer les acquis ou les inconvénients provenant du fait de travailler sur un logiciel déjà fonctionnel par rapport au fait de travailler dans la communauté du logiciel libre. Nous avons choisi un ensemble de questions à réponse fixe puis un ensemble de questions à réponse libre réparties en cinq catégories :

- Logiciel fonctionnel. Il s'agissait de déterminer les apprentissages fait en travaillant sur un logiciel déjà existant.
- Logiciel libre. Il s'agissait de déterminer les apports dus au type de logiciel utilisé (les logiciels libres et leur communauté)
- Capacité demandées. Il s'agissait de déterminer les compétences qui ont semblées indispensables en prérequis aux étudiants.
- Motivation. Il s'agissait de déterminer la différence entre ce cours et un cours classique.
- Pérennité du travail. Il s'agissait de réponses libres pour les étudiants portant sur leur ressenti.

Nous leur avons demandé de donner les réponses les plus franches possibles . Les étudiants ont été incités à répondre tout en leur précisant que les réponses n 'influenceraient en aucun cas la note obtenue. L'impression générale ressortant de ces réponses est la satisfaction qu'ils ont eu de participer à ce cours. L'expérience de travailler sur de vrais logiciels a été pour eux une expérience enrichissante qui leur a permis d'apprendre à devenir autonomes et à prendre confiance en eux

Les graphes de la figure 1 présentent les résultats des deux années confondues, soit une centaine d'étudiants. Les trois premiers graphiques de la figure 1 décrivent les résultats obtenus en matière d'apprentissage : dans l'ensemble, travailler sur des logiciels de grande taille a permis aux étudiants de découvrir l'organisation en module du code source et leur a é galement permis de parfaire leur apprentissage en matière de normes de programmation . Ils déclarent également avoir fait des progrès en matière de compréhension de code . Ces résultats s'expliquent en particulier par le fait que les étudiants en troisième année de licence possèdent déjà un bon niveau de programmation mais manquent d'expérience quant au développement de logiciel de grande taille. Aussi ont-ils pu parfaire leur connaissance dans ce domaine. Il en est de même pour les normes de programmation : jusqu'à présent ils n 'avaient eu pour la plupart que des exemples théoriques . Ce cours leur a permis d'appréhender concrètement ces normes. Ces résultats tendent ainsi à prouver que le cours a atteint ses objectifs en matière de compréhension de code.

Les trois autres graphes de la figure 1 décrivent les ressentis en matière de communication avec la communauté du libre . Les étudiants ont majoritairement fait des progrès en matière de recherche d'informations et de communication avec la communauté. Nous nous sommes aperçu que les étudiants avaient bien sûr des notions de ce qu'est un logiciel libre mais ils n'avaient pour la plupart pas encore eu l'opportunité de côtoyer la communauté open source ni même parfois d'appréhender en détail les notions de licence GPL . Les mettre sur de tels projets leur a permis de mieux aborder la communauté et ces résultats tendent à prouver que notre second objectif(à savoir la communication) a là encore été atteint pour la plupart des étudiants

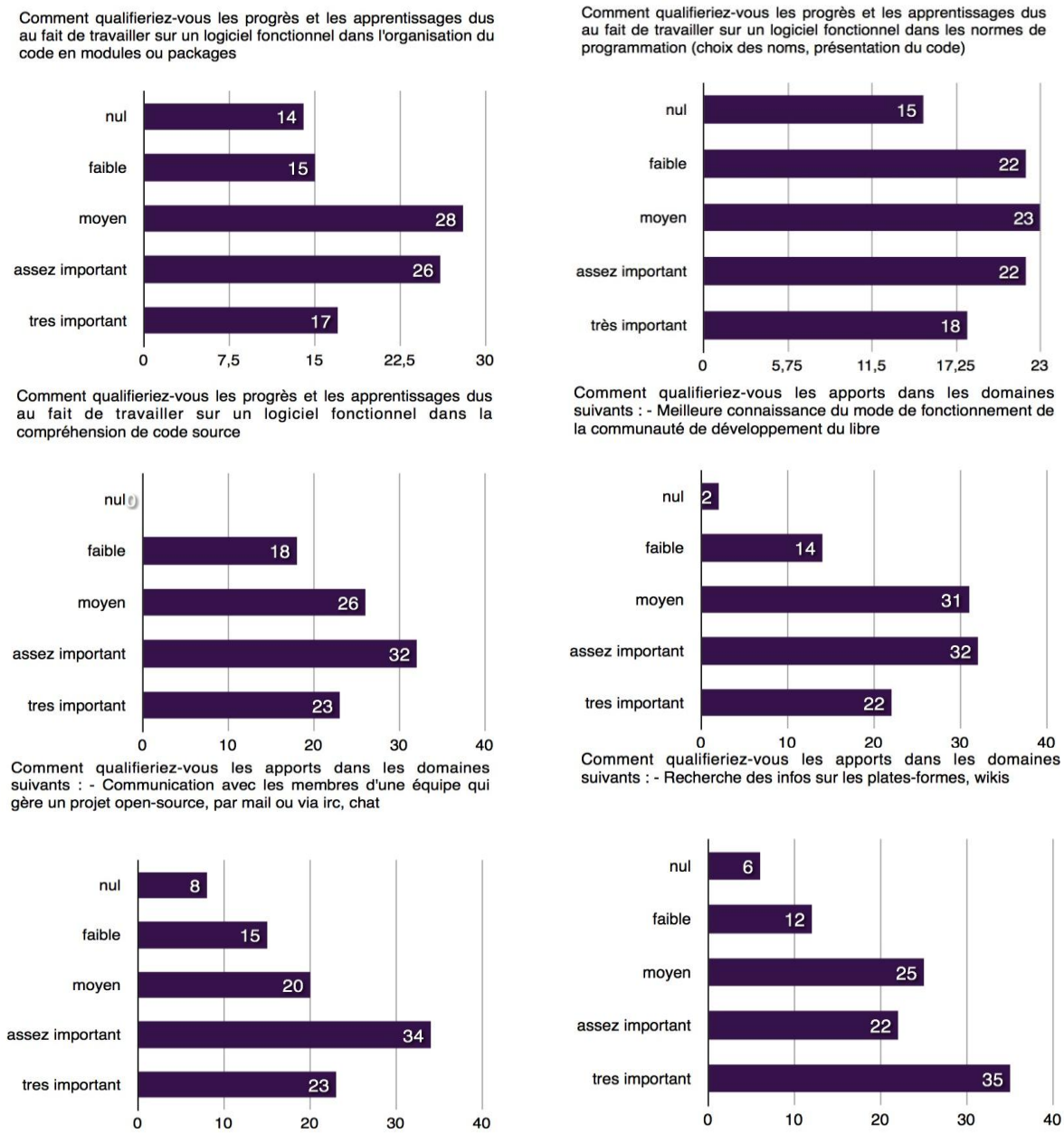

Figure 1. Principaux résultats du questionnaire remis aux étudiants en fin de semestre

Discussion

La réussite du cours de compréhension de programmes tel qu'il est proposé repose en grande partie sur le choix des logiciels utilisés. Ce n'est plus un cours magistral qui permet de définir précisément les enseignements qui seront dispensés mais un cours pratique dont l'enseignement repose sur les projets et les difficultés rencontrées par les étudiants durant le semestre ainsi que sur l'investissement des tuteurs. Les projets trop compliqués et les tuteurs qui ne sont pas assez réactifs découragent les étudiants et il est ensuite très difficile de faire avancer le groupe correctement.

Chaque séance nécessite un investissement au cas par cas, totalement personnalisé. Il faut apprendre aux étudiants des techniques de compréhension mais sous un angle passant totalement par la pratique. Il appartient ensuite à l'enseignant de monter aux différents groupes en quoi le travail effectué est reproductible dans d'autres situations, tout comme l'enseignant doit également faire acquérir aux étudiants une plus grande autonomie et de leur montrer qu'ils possèdent d'ores et déjà les

compétences pour travailler sur des logiciels en production . Il s'agit de faire prendre conscience aux étudiants de leur propre compétence . Finalement, le principal apprentissage du cours ne réside par uniquement dans les méthodes de compréhension mais dans la manière d'utiliser des outils , d'interagir avec une équipe déjà en place pour obtenir des informations et de l'aide.

D'un point de vue enseignant , ce cours a réservé des surprises . En effet, il a permis de mettre en avant la motivation réelle des étudiants à travailler sur des projets grandeur nature. Ce cours permet également à des étudiants de niveau moyen mais motivés de se dépasser et de se prouver qu'ils ont de réelles capacités s'ils sont mis en situation concrète de travail. Tous les étudiants au début du semestre considéraient les tâches impossibles à réaliser et ceux qui ont eu la possibilité de se trouver sur un bon projet ont acquis, au delà des techniques de compréhension de code , une confiance et une autonomie beaucoup plus grande. D'un autre côté, il a fallu expliquer aux étudiants que le travail ne se mesure pas en nombre de lignes de code écrites , tout comme il a fallu les sensibiliser sur le fait qu'une contribution ne consiste pas uniquement en l'ajout de lignes de code mais peut également prendre la forme d'un nouveau bogue découvert ou encore d'un travail de recherche et de veille technologique qui permettra ensuite d'implémenter une nouvelle fonctionnalité.

L'enseignant doit assurer deux fonctions : celle d'ingénieur capable de trouver les solutions aux problèmes que les étudiants ne savent résoudre seuls et celle de pédagogue qui explique comment parvenir à la solution, comment chercher, comment mettre en pratique les connaissances acquises. En effet, il faut d'une part un enseignant qui discute avec les étudiants des méthodes de travail , qui enseigne au cas par cas des méthodes de résolution , de compréhension, qui aide à la rédaction des documents et d'autre part un ingénieur capable d'aider les groupes à résoudre les difficultés techniques telles que les problèmes d'installation des outils ou la compilation du projet , par exemple. Si l'une ou l'autre de ces fonctions ne sont pas exercées correctement le cours souffre d'un déséquilibre qui entraîne vite le découragement des étudiants. Si une seule personne peut endosser les deux rôles, nous avons choisi de faire ce cours avec deux personnes identifiées par les étudiants dans leur rôle respectif de manière à permettre aux étudiants de distinguer les apprentissages des difficultés techniques.

Discussion à propos de l'usage des logiciels libres

Depuis deux ans que ce cours existe, nous nous sommes confortés dans le choix des logiciels libres comme support de cours . En effet, travailler sur du code de taille conséquente , dans des conditions quasiment réelles de travail, en suivant des normes et des règles et en s'intégrant dans une équipe de développement, tous ces points renforcent le poids de l'enseignement de la maintenance de code.

En revanche, nous émettons tout de même une réserve d'importance : tous les logiciels ne se prêtent pas à l'exercice. En effet, le choix du projet et du tuteur associé est fondamental . Les projets font donc l'objet d'une sélection rigoureuse préalablement au cours. Il s'agit de trouver des logiciels qui soient correctement écrits, qui suivent des normes et des règles de programmation et de soumission de code précises , qui ont une communauté de développeurs active et parmi cette communauté, une personne en particulier qui voudra bien endosser le rôle de tuteur . Les projets doivent être dynamiques mais pas immatures, ils doivent être encore en évolution mais avoir atteint un stade de maturité suffisant pour posséder des versions stables et bien écrites.Finalement un des socles du cours repose sur le partenariat fort à instaurer entre l'enseignant et la communauté du libre pour que le cours puisse avoir lieu dans de bonnes conditions.

De plus , les étudiants du cours n'ont pas tous les mêmes compétences techniques , ne serait-ce qu'en terme de langage de programmation qu'ils maîtrisent . Il faut donc impérativement adapter le choix du projet aux compétences du groupe. D'où la nécessité de proposer des projets variés en terme de langage, d'environnement, de difficulté technique, ce que nous permettent les logiciels libres . Il faut ensuite veiller à l'adéquation projet/étudiant.

Les compétences techniques annexes sont un autre élément indispensable à prendre en compte. En effet, si pour comprendre et développer/modifier le logiciel il est nécessaire de posséder des compétences propres au domaine ciblé, il peut s'avérer trop difficile de se mettre à niveau sur un aussi court laps de temps (6 mois pour le cours). Il faut donc là encore communiquer préalablement avec l'équipe de développement pour évaluer les prérequis nécessaires à la compréhension du logiciel et de son code.

Le rôle du tuteur.

Celui-ci doit être réactif, répondre à toutes les demandes qu'il reçoit et adapter ses réponses pour aider les étudiants à progresser : préciser lorsqu'il suffit au groupe de faire une simple recherche sur le forum ou répondre directement lorsque la question est plus technique. Il s'agit de faire avancer le groupe plus facilement sans pour autant répondre à des questions intempestives. Le tuteur est la personne qui connait le mieux l'application et il est donc celui qui répond en priorité aux questions techniques ou qui oriente vers le développeur compétent en la matière. Le fait que le groupe ait un tuteur permet également d'avoir une sorte de caution morale sur les forums et obtenir des réponses et de l'aide plus facilement.

Le tuteur a également un rôle déterminant dans le choix du travail à effectuer. Il guide le groupe sur une partie du code à analyser ou sur un objectif à atteindre ou encore sur les moyens d'y parvenir.

Lorsque le tuteur n'est pas assez présent ou que la communication entre le groupe et le tuteur est mauvaise, le groupe se prive d'une ressource essentielle pour l'aboutissement du projet et bien souvent ne peut plus aller au bout de celui-ci. Il est donc fondamental que le tuteur s'investisse. D'un autre côté, le tuteur doit rester à sa place et ne pas systématiquement résoudre les difficultés rencontrées si le groupe n'a pas fait l'effort de chercher et de rédiger correctement son problème. C'est ici le rôle de l'enseignant d'être le garant de la bonne communication entre les deux partenaires, en veillant à la disponibilité du tuteur et en veillant à la manière dont le groupe communique avec lui.

Conclusion et perspectives

Le cours de Compréhension de programmes est un cours pratique qui enseigne aux étudiants à modifier du code source sur une application existante dans le but d'un apprentissage des techniques de maintenance de code. Il repose sur les logiciels libres et sur un partenariat avec le monde de la communauté du libre. Fort de cette expérience de deux ans, nous préparons la troisième année en renforçant ce partenariat.

Bien que tous les projets n'aient pas conduits à la même satisfaction, il faut tout de même noter que les étudiants sont restés motivés durant tout le semestre et n'ont pas abandonné. Ils ont dans l'ensemble, malgré les échecs, trouvé l'expérience intéressante. En revanche, ils ne peuvent pas forcément quantifier ou décrire les enseignements que le cours leur a apportés. Ils ressentent ce cours comme une expérience enrichissante mais pas comme un cours traditionnel dans lequel ils ont appris de nouveaux concepts.

Si certains projets ont été un échec du fait de l'aspect expérimental du cours, nous nous efforçons de corriger les erreurs en ajustant le choix des applications pour qu'elles répondent aux besoins du cours. Nous ajustons également notre pédagogie pour prendre en considération les particularités d'un tel enseignement. Tout comme il est indispensable que le tuteur qui travaille avec nous sur le cours soit réactif mais trouve aussi une satisfaction dans un tel projet.

Afin de diminuer le nombre de projet en échec, nous allons abandonner le partenariat avec certains projets tandis que nous allons en renforcer d'autres. Nous affinons chaque année notre choix de projets et nous nous dirigeons vers une définition assez précise des projets adéquats pour un tel cours. Nous pouvons dors et déjà déterminer certaines caractéristiques : la communauté de développeurs est primordiale et doit être active, le code doit être mature (pas de changements de taille dans l'architecture du code), le travail à effectuer doit être clair et concis – les étudiants doivent savoir précisément ce qu'on attend d'eux –, les sujets ne doivent pas être trop complexes – il vaut mieux un petit projet finalisé, qu'un projet trop prétentieux qui sera abandonné avant la fin. –, enfin il est souhaitable que les technologies utilisées pour le projet soient maîtrisées par l'étudiant. Cependant certaines paramètres sont difficiles à connaître par avance, notamment celles liées au tuteur (sa réactivité, sa capacité d'encadrer les étudiants).

Les apports du cours ont dépassé nos espérances. Nous ne sommes plus dans l'apprentissage de techniques de compréhension mais dans l'immersion en environnement proche du professionnel. De même, le fait que certains étudiants aient obtenus des stages nous orientent dans la mise en place d'un partenariat plus accentué de ce côté : mettre en place des projets qui se poursuivent lors de stage ou bien encore d'une année sur l'autre, sensibiliser au rôle social des logiciels libres, aux licences, aux notions de propriété intellectuelle.

Etant donné le retour positif de la part des étudiants, et nos observations concluantes, nous comptons reconduire ce cours pour la troisième année. Etant donné que ce cours reste en élaboration, nous apporterons certaines modifications comme celles mentionnées ci-dessus (choix de certains projets, parler de certains sujets annexes telles que les licences). Un des points importants du cours consiste à la tenue d'un journal par les étudiants dans le but de retracer l'évolution de leur travaux. Actuellement ce journal est tenu sur une plate-forme privée d'enseignement à distance. Nous souhaitons transformer ce journal en blog ou wiki public et inciter ainsi les étudiants à élever le niveau de langage utilisé et à professionnaliser leurs écrits et dans le même temps, augmenter la visibilité de notre expérience pour inciter d'autres cursus à suivre cet exemple.

Bibliographie

Basdarakas, P.; Chalkias, A.; Deligiannis, P.; Tsakiri, P. & Tsantalis, N. (2006). Lessons learned from an open-source university program. *ICCOMP'06 Proceedings of the 10th WSEAS international conference on Computers*, (p. 284–287).

Coverity scan report, (2012), http://softwareintegrity.coverity.com/register-for-the-coverity-2012-scan-report.html.

Dionisio, J. D. N., Dickson, C. L., August, S. E., Dorin, P. M., & Toal, R. (2007). An open source software culture in the undergraduate computer science curriculum. *ACM SIGCSE Bulletin*, 39(2), 70-74.

Ellis, H. J., Hislop, G. W., Chua, M., & Dziallas, S. (2011). How to involve students in FOSS projects. In *Frontiers in Education Conference* (*FIE*), 2011 (p. T1H-1-T1H-6). IEEE.

Ellis, H. J., Morelli, R. A., de Lanerolle, T. R., Damon, J., & Raye, J. (2007). Can humanitarian open-source software development draw new students to CS?. *ACM SIGCSE Bulletin*, 39(1), 551-555.

Free Software Foundation (2007). Gnu general public licence, version 3. Obtenu à : https://gnu.org/licenses/gpl.html

German, D. (2005). Experiences teaching a graduate course in open source soft- ware engineering. *Proceedings of the First International Conference on Open Source Systems*, (p.326–328).

Glass, R. (2001). Frequently forgotten fundamental facts about software engi-neering. *IEEE Software*, 18(3), (p.112–111).

Goldweber, M., Davoli, R., Little, J. C., Riedesel, C., Walker, H., Cross, G., & Von Konsky, B. R. (2011). Enhancing the social issues components in our computing curriculum: computing for the social good. *ACM Inroads*, *2*(1), 64–82.

Grubb, P. A., & Takang, A. A. (2003). *Software maintenance: concepts and practice*. World Scientific.

Hess, R., & Paulson, P. (2010, March). Linux kernel projects for an undergraduate operating systems course. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 485-489). ACM.

Hislop, G. W., Ellis, H. J. C., & Morelli, R. A. (2009). Evaluating student experiences in developing software for humanity. In *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education* (p. 263–267). New York, NY, USA: ACM.

IEEE. (2012). SWEBOK Guide V3 – Alpha Version. Chapter 5 - SOFTWARE MAINTENANCE.

IEEE-SA Standards Board. (1998) IEEE standard for software maintenance,.

Lehman, M. M. (1996). Laws of software evolution revisited. In *Software process technology* (pp. 108-124). Springer Berlin Heidelberg.

Lipşa, D. R., & Laramee, R. S. (2011). Open Source Software in Computer Science and IT Higher Education: A Case Study. *International Journal of Advanced Computer Science and Applications (IJACSA)* Editorial. 2(1) (p. 10–17)

Marmorstein, R. (2011, June). Open source contribution as an effective software engineering class project. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 268-272). ACM.

Patterson, D. A. (2006). Computer science education in the 21 st century. *Communications of the ACM*, 49(3), 27-30.

Pedroni, M., Bay, T., Oriol, M., & Pedroni, A. (2007). Open source projects in programming courses. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education* (p. 454–458). New York, NY, USA: ACM.

Squire, M., & Duvall, S. (2009). Using FLOSS Project Metadata in the Undergraduate Classroom. In *Open Source Ecosystems: Diverse Communities Interacting* (pp. 330-339). Springer Berlin Heidelberg.