Université du Québec
École de technologie supérieure

RAPPORT TECHNIQUE

PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

DANS LE CADRE DU COURS LOG792 - PROJET FIN D'ÉTUDE

Jonathan Ducharme

DUCJ02108309

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur superviseur

Alain April

MONTRÉAL, 13 AVRIL 2011
HIVERS 2011

Table des matières

Sommaire 4
Avant-propos et remerciements 5
Glossaire 6
Introduction 7
L'importance du code de qualité 7
Code Complete 8
Problématique 9
Analyse des besoins 9
Analyse actuelle 10
Revue de la documentation 11
Structure du projet 12
Risque 12
Plan cadre .... **Error! Bookmark not defined.**
Laboratoire 12
Énoncer 12
Machine Virtuelle 13
Examen final 13
Solution 14
Outils 14
Intégration Continue 14
Gestion de dépendance 15
Système de contrôle de version 15
Outil d'analyse statique de code 15
Gestion de tâche 16
Concept de programmation 17
Pseudo code et commentaire 17
Variable 17
es instructions 18
Gestion d'erreur et journalisation 18
Les tests 18
Discussion 20

Amélioration potentiel 20
Conclusion 21
Référence 22
Annexes 23
Information de la VM 23

Sommaire

Dans le présent document, il sera question des étapes suivies pour arriver à l'élaboration d'un tout nouveau cours de génie logiciel et des technologies de l'information. Il sera question des concepts pressentis pour ce cours, des laboratoires, des outils vus dans le cours. Vous trouverez l'analyse de l'auteur qui a mené à l'idée de créer un tel cours.

Ce projet ne se veut en rien final, il y aura donc une partie sur l'amélioration possible, ou tout simplement des sujets qui pourraient être vue par le cours plus celui-ci sera mature.

Il est cependant à noter que le présent document sera biaisé plus vers les technologies entourant le monde Java puisque l'auteur, Jonathan Ducharme, est un développeur Java depuis 5 ans et il en a fait son domaine d'expertise. Les concepts se veulent cependant généraux et le but est qu'il soit applicable dans tous les langages de programmation moderne.

Avant-propos et remerciements

Je tiens d'abord à remercier M. Alain April, mon superviseur de projet, qui m'a donné beaucoup de liberté pour ce projet. Réaliser un corpus de cours n'est pas une tâche facile et il a été présent quand j'ai eu besoin d'aider.

Je tiens à remercier Sébastien Martin, Manuel Darveau, Martin Morisette, Jean-Sébastien Bettez et François Lamarre, tous des amis et collègues de travail chez 8D pour m'avoir aidé, sans le savoir, à réaliser ce projet. Leur support et leur expérience m'ont aidé à devenir une meilleure personne, et un meilleur développeur depuis les huit derniers mois.

Je ne pourrais passer sous silence la contribution à ma vie personnelle et professionnelle du club S.O.N.I.A. qui m'a apporté beaucoup pendant mes études à l'ÉTS.

Finalement, je tiens à remercier Gennifer Greiss, ma compagne dans la vie, la personne qui m'endure dans les bons, comme moins bons moments, et ce, depuis maintenant 4 ans.

Glossaire

ETS	École de technologie supérieure
C++	**Langage de programmation orienté objet**
C#	**Langage de programmation orienté objet**
Java	**Langage de programmation orienté objet maintenu par Sun Microsystems (Oracle)**
SVN	**Outils de gestion de version pour le code source**
TI	**Technologies de l'information – Programme offert à l'ÉTS**

Introduction

Le corpus de cours en génie logiciel et en technologie de l'information de l'ÉTS est diversifié. On y trouve des cours sur l'architecture, la conception, la maintenance, les requis et plusieurs d'autre. Cependant, une des lacunes de ce corpus, et ce, selon l'auteur de ces lignes, est qu'il manque un cours sur la construction logiciel. Ce cours aiderait les étudiants à comprendre plusieurs principes propres à la programmation.

On retrouve plusieurs références dans le domaine pour la construction logiciel. Un qui se démarque par sa réputation est Code Complete. Ce livre a servi d'inspiration pour ce projet, et il est la référence obligatoire du cours.

L'importance du code de qualité

Écrire du code de qualité à plusieurs bénéfices. Si on laisse de côté l'impression de professionnalisme, qui est un bonus pour le développeur ayant écrit le code, il y a des économies de coût qui est difficilement quantifiable rattaché à cet effet.

Une de ces économies se fait au niveau de la maintenance. Peu importe la qualité du code, la qualité du design, des requis, ou l'alignement des planètes, il y aura toujours de la maintenance à faire dans le code. Or, du code bien écrit, clair, bien documenté et ayant la bonne portée est beaucoup plus simple à maintenir. Le développeur qui travaillera dans ce code aura beaucoup moins de difficulté à comprendre le code, et faire les rectifications qui se doivent, que si le code aurait été de piètre qualité.

Les bogues sont partie intégrante du travail du développeur. Il doit en régler constamment, quand il en règle, d'autre sont peut-être intégré, et ce que le code soit d'une grande qualité ou non. Cependant, plus le code est fait, plus la chance d'avoir des problèmes est mince, et quand il y en a, les régler est plus facile pour les mêmes raisons que la maintenance est plus simple.

Pour une entreprise, ces économies peuvent représenter beaucoup d'argent. Encore une fois il est dur de quantifier le montant des économies. Par contre, voir du beau code pour des développeurs est souvent signe d'un meilleur logiciel, et c'est beaucoup plus plaisant de travailler dans un bel environnement.

Code Complete

Le choix de Code Complete pour réaliser ce cours a été plutôt simplet. Premièrement, le livre à une très bonne réputation dans le milieu. De plus, Code Complete ne se concentre pas sur une technologie en particulier. Les exemples de code ne sont pas seulement faits dans un langage particulier. Steve McConnell se concentre donc plus sur des concepts qui sont portables dans tous les langages de programmation plutôt que des spécificités d'un langage.

Un autre avantage du livre est qu'il couvre plusieurs concepts intéressants. Il parle quelque peu de conception, mais orienté au niveau de la construction logiciel et d'outils pour aider à produire du code de plus grande qualité. Évidemment, le sujet principal du livre est l'écriture du code, et c'est extrêmement bien couvert. L'auteur parle des variables, des classes, méthode, des instructions, de toutes les structures de contrôles (if, while, for, case…). Il donne généralement des exemples de code à éviter et d'autres exemples de ce qu'il est mieux de faire.

De plus, à la fin de chaque sujet, l'auteur offre des listes de contrôle pour aider les développeurs à s'assurer que le travail produit est de bonne qualité.

Problématique

Dans le cadre du BAC en génie logiciel de l'École de technologie supérieure, l'étudiant apprend à devenir un bon ingénieur. Il apprend, entre autres, à penser, faire des requis logiciel, du design, de l'architecture et des normes qui pourront potentiellement l'aider dans sa vie professionnelle.

Par contre, un des aspects un peu plus négliger du BAC est comment créer, du code, l'essence même d'un logiciel, de hautes qualités. On assume que l'étudiant sait déjà comment écrire du code, ce qui n'est pas mal, mais ce qui est parfois erroné, surtout depuis l'arrivée du programme de cursus qui accepte maintenant les gens qui n'ont pas un DEC technique.

Le BAC ne lui donne pas un survol complet des outils qui peuvent être utilisé pour écrire ce code de meilleure qualité. Des outils d'intégration continue ou d'agrégation de données statistiques de tests. Des outils qui vont faciliter le déploiement ou la construction du logiciel.

Analyse des besoins

Quoi que le département offre beaucoup de cours de qualité, quelques cours sur des technologies, ou des concepts, plus récente pourrait être offert. Un de ces concepts, qui est extrêmement important c'est d'écrire du code de haute qualité. Ce n'est pas un nouveau concept, mais la façon de faire est plus moderne et s'est améliorée avec le temps.

Le département devrait se munir d'un cours qui offrirait aux étudiants beaucoup de temps en laboratoire pour qu'ils puissent s'habituer à travailler avec des outils qui aideraient l'amélioration du code. De plus, écrire du code de qualité requiert beaucoup de pratique, il y a quelque concept théorique à comprendre, mais c'est un cours pratique ou les étudiants doivent produire une importante quantité de code pour bien comprendre et assimiler les concepts.

Les étudiants auraient donc besoin d'écrire beaucoup plus de code, dans un cadre dirigé et dans lequel ce code serait évalué. Ils devraient apprendre à utiliser une méthodologie plus moderne et répandue en entreprise, tel le scrum, ainsi que d'apprendre à utiliser des technologies récentes pour aider à l'amélioration continue de la qualité du code.

Analyse actuelle

Depuis les quatre dernières années, le programme de génie logiciel et des TI a évolué. Un nouveau cours, LOG240 sur la maintenance et le test du logiciel ont fait son apparition. C'est aussi le cours qui se rapproche le plus des concepts qui veulent être enseignés dans le cours décrit par ce projet. Le cours LOG240 couvre surtout les outils vus dans le cours.

Les autres cours tiennent en général pour acquis que les étudiants sachent écrire du code et ils ne sont donc pas vraiment évalués sur cet aspect, mais bien sur les principes que le cours doit enseigner. Par exemple, les cours de LOG120/210 enseignent les concepts de conception de logiciel et les laboratoires sont orientés vers ces concepts. LOG350 est orienté vers l'interface utilisateur, LOG430 vers l'architecture, LOG660 vers les bases de données.

Tous ces cours requièrent d'écrire du code. Par contre, peu de points sont donnés pour cet aspect précis. L'implémentation de la solution n'est pas aussi importante que le concept derrière et c'est tout à fait normal. Ce ne sont pas des cours qui sont orientés vers l'écriture du code, mais des cours qui sont orientés vers d'autres aspects du génie logiciel.

Pour ce qui est de la méthodologie scrum, c'est un des éléments qui pourrait être montré dans le cours de gestion de projet LOG515. Cependant, ce cours ne semble pas tout à fait à jour et ne se concentre pas ou peut, sur des méthodologies modernes de développement. On ne parle à peu près pas d'agilité, de concept comme le scrum ou l'extrême programming qui sont des méthodologies axées sur le développement logiciel.

Revue de la documentation

La documentation utilisée pour ce projet se résume en Code Complete. Le cours est entièrement basé sur ce livre. Il est d'ailleurs le livre obligatoire pour les étudiants. Code Complete couvre tous les sujets, et même plus, que les étudiants auront la chance de voir dans ce cours. C'est une des bibles du développeur logiciel.

Structure du projet

Risque

Les plus grands risques sont l'inexpérience de l'auteur pour un tel projet. Créer un cours demande un minimum de connaissance pédagogique, et elles ne sont pas maîtrisées.

Plan-cadre

Le plan-cadre a été réalisé avec tous les concepts clés que les étudiants devront apprendre et maîtriser pour être en mesure d'avoir la note de passage du cours. Le plan-cadre servira à l'élaboration du cours, des acétates, des projets de laboratoire et des évaluations.

Le plan-cadre a été remis avec le rapport d'étape au début du mois de Mars.

Laboratoire

Énoncer

Quatre énoncés de laboratoire, sur six, ont été produits pour ce projet. Ils couvrent les quatre premier sprints d'un projet ayant 6 sprints, de deux semaines chacun.

Lab01 – Sprint 0

Le but de ce laboratoire est de faire le sprint 0 de la méthodologie scrum. Pour les besoins du cours, ce sprint 0 se compose en l'élaboration de l'équipe de deux personnes et la mise en place de l'infrastructure. Toutes les tâches devront être insérées dans Jira et priorisées. Certaines tâches pourront être mises dans un « backlog » pour être faites plus tard. Par contre, il n'y aura pas de point attribué si les tâches sont faites dans des itérations subséquentes.

Lab02 – Sprint 1

Le but de ce laboratoire sera de créer en un sprint une application fonctionnelle qui portera sur un sujet précis établi au préalable. Le sujet est laissé libre au chargé de TP, mais ce doit être un projet composé de plusieurs requis fonctionnel qui seront relativement simple à produire. Dans un monde idéal, le chargé de TP ou le professeur du cours, aura fait un cadre d'application, ou une interface graphique qui sera le point de départ des étudiants. Notez que le cadre d'application pourra contenir des erreurs, et des irrégularités voulues pour des fins d'apprentissage. Les critères d'évaluation pour ce sprint porteront surtout sur l'utilisation du pseudo-code et des commentaires.

Lab03 – Sprint 2

Une nouvelle liste de requis sera donnée aux étudiants pour implémentation. Ils devront, comme dans tous les sprints, les tâches devront être entrées dans Jira et priorisées. L'évaluation du laboratoire portera sur la journalisation et la gestion d'exception. De plus, les éléments récurant dans tous les laboratoires seront évalués.

Lab04 – Sprint 3

Dans ce sprint, une importance particulière sera portée sur les tests. Les étudiants devront utiliser une méthodologie de Test Driven Developpement, qui dit de faire les tests en premier pour la liste des fonctionnalités. Pour cette itération particulièrement, ils devront faire des branch pour chaque étape de leur développement. Une branch pour prouver que les tests ont été faits et qu'ils ne passent pas et une branch qui montre que les fonctionnalités ont été faites avec les tests, idéalement inchangés, passent.

Élément récurant des laboratoires

Voici une liste des éléments qui seront évalués pour tous les laboratoires. Certains d'entre eux auront une plus ou moins grande pondération selon l'itération pour que le concept soit bien compris par les étudiants.

Ces concepts seront :

- La clarté des commentaires
- La clarté des instructions
- L'entrée et la priorisation des tâches
- De vrais tests qui passent
- L'analyse statique du code

La remise pour tous les laboratoires est la même. Comme les étudiants utiliseront Maven, il devront faire une « release » de leur projet et ils remettront le .jar exécutable de leur projet. Ils devront créer un tag pour le « release » et une branche pour le snapshot créé. Pour le prochain sprint, la version du logiciel devra être incrémentée.

Machine virtuelle

La machine virtuelle a été conçue pour donner un environnement de travail aux étudiants. Ils pourront y accéder par le VPN de l'ÉTS pour travailler de la maison. Cette VM servira pour l'installation des outils choisis pendant les TP par les étudiants. La VM est un serveur d'application qui contiendra entre autres, Jira, SVN, Hudson, Sonar, Nexus et Maven. Une VM par équipe sera attribuée aux étudiants. L'installation et la configuration de la VM seront faites lors du premier laboratoire soit le sprint 0.

Examen final

Ce cours ne comporte pas d'examen intra, seulement un final. Le final posera des questions théoriques sur les concepts vus en classe. Il demandera aux étudiants de justifier certaines parties des problèmes qu'ils ont vécus lors du cours. L'examen testera leur apprentissage, et posera des questions spécifiques sur le cours. L'examen se fera sans note, à livre fermé et testera la compréhension plutôt que de savoir si les étudiants sont capables de réciter, par cœur, les lignes du livre.

Solution

La solution proposée par ce projet est de créer un nouveau cours avec un corpus centré sur l'apprentissage et l'amélioration du code. Un cours qui sera très technique et pratique avec des concepts théoriques qui seront pratiqués à mainte reprise.

Il y a plusieurs concepts qu'on peut apprendre aux étudiants pour les aider à faire du code de meilleures qualités. D'abord les outils qui peuvent être utilisés pour améliorer le code, mais aussi pour bien maîtriser l'environnement du projet. Le but est d'aider les étudiants et à leurs données une bonne boîte d'outil et de l'expérience avec ce qui pourra les aider à produire le code de la plus haute qualité possible.

Outils

Dans le cadre de ce projet, les outils sont définis comme des logiciels externes pouvant aider au développement d'application. Ils ont différente portée et utilité dans le projet. Certains d'entre eux sont dédiés à la qualité du code, d'autre au bon fonctionnement d'un projet. Par contre, ils sont tous importants pour l'évolution d'un projet.

Intégration continue

L'intégration continue est définie comme un processus dans lequel le logiciel va être compilé et vérifié après chaque changement. Dans le cas présent, un outil tel que Hudson sera utilisé. L'importance de l'intégration continue est pour contrer les mauvaises surprises lors de changement au code. Tous les tests unitaires, idéalement, devraient être roulés après chaque changement pour s'assurer de l'intégrité de celui-ci.

Les étudiants apprendront donc à se servir d'un outil populaire pour l'intégration continue avec Hudson. Ils apprendront à faire la configuration et l'installation du système, à comprendre comment créer des tâches et à maintenir le serveur d'intégration continue. Ils verront aussi les concepts théoriques reliés à l'intégration continue, les avantages, les inconvénients et être capable de comprendre pourquoi l'outil est rendu aussi important dans le domaine logiciel.

Gestion de dépendance

Une des plaies d'écrire du code est de faire la gestion de toutes les dépendances pouvant vivre dans un projet donné. Dans cette optique, Maven est l'outil qui doit être utilisé pour contrer ce problème. Le plus gros problème de Maven est qu'il est centré vers les technologies Java. Une version dans le monde Microsoft existe, par contre elle n'est pas aussi étoffée. Même si l'outil est orienté vers une technologie particulière, il a été choisi pour faire partie de la suite d'outil que les étudiants doivent utiliser.

Maven est le meilleur outil pour faire comprendre l'importance de la gestion de dépendances aux étudiants. Un projet est en général composé de plusieurs dépendances vers des libraires externes. Chaque librairie possède sa propre version, et ce peut être complexe gérer cet aspect qui est pourtant d'une importance capitale.

Maven est aussi utilisé pour aider à construire le logiciel. Il fournit des utilitaires pour faire cette tâche, ainsi que de faire rouler des tests. De plus, Maven étant un projet très prisé par la communauté du logiciel libre Java, obtiens beaucoup de support et beaucoup de plugiciels sont disponibles. Parmi ces plugiciels, certains d'entre eux sont faits uniquement pour faire de l'analyse de code.

De plus, un autre outil intéressant dans le monde de la gestion de dépendance, et allant de concert avec Maven est Nexus. Cet outil est préconisé pour conserver les dépendances à l'interne d'une organisation. Il est aussi utilisé pour conserver les versions des logiciels déployés en version de production.

Système de contrôle de version

Un des outils les plus importants dans le monde du logiciel. Cet outil, dans le cas de ce projet SVN, sert essentiellement d'entrepôt de code source. Cet outil est un essentiel aujourd'hui, non seulement il y a un historique sur les changements, mais il permet de créer des « Tags » et des « Branches » pour les différentes versions.

Les étudiants doivent apprendre à se servir d'un tel outil parce qu'il est rendu un standard en entreprise. Si ce n'est pas le cas, un nouvel ingénieur doit insister pour que l'entreprise utilise un système de contrôle de version parce qu'il est la base de l'amélioration du code. Sans cet outil, la traçabilité du code est quasi impossible à faire.

Outil d'analyse statique de code

L'utilisation d'un tel outil est l'essence même de la qualité du code. Sonar, qui est l'outil à la mode, et qui présente les informations de plusieurs outils d'analyse de code, est le choix de prédilection. Il est facile d'utilisation, s'intègre bien à Maven et moyennant quelques plugiciels peut-être adaptés pour faire de l'analyse de code avec d'autres langages de programmation que Java.

Sonar présente beaucoup de statistique sur le code et la qualité. Il est facile de voir qu'est qui cloche et pourquoi. Plusieurs règles d'analyse sont intégrées dans logiciel, il est facile de faire des profils pour les différentes analyses voulues.

Gestion de tâche

Être capable de bien gérer son temps est primordial dans n'importe quel projet. Il est aussi un facteur de la qualité du code. Si le temps est mal géré est qu'on arrive à un point ou on doit tout faire rapidement, la qualité du code et du projet diminuera. Il est donc important de voir comment utiliser un logiciel pour la gestion de tâche.

Pour cet aspect, Jira a été choisi. Contrairement à l'ensemble des outils présenté jusqu'ici, Jira n'est pas un produit libre. Il est développé par Atlassian et est vendu. Il y a plusieurs produit qui aurait pu être utilisé, tel que Redmine, Bugzilla… mais Jira est celui qui est le plus étoffé et le plus complet. Il s'intègre très bien avec les outils précédents, est très utilisé dans la communauté et a un excellent plugiciel pour les méthodologies agiles avec Greenhopper. Un autre outil qui est absolument nécessaire dans une entreprise.

Concept de programmation

Les outils ne peuvent qu'aider le développeur à produire du meilleur code. Par contre, des principes solides de programmation, des connaissances et de la pratique de concept vont grandement aider à écrire et maintenir le code pour qu'il soit de qualité supérieure.

Pseudo-code et commentaire

Le pseudo-code est souvent mal utilisé et ses bienfaits sont mal connus. Écrire le pseudo-code est parfois révélateur, surtout pour les bouts de code complexe. Il permet de savoir exactement ce que le développeur veut, ou ce qui doit être fait. Comme c'est fait en langage naturel, le pseudo-code n'est pas un travail perdu puisque par la suite il peut être utilisé comme commentaire pour le même bout de code qui vient d'être réalisé.

Pour ce qui est des commentaires, il est important de les utiliser à bon escient. Il ne faut pas documenter toutes les lignes, méthode, classes, mais bien ce qui est significatif et où le commentaire ajoute de la valeur au code.

Par la suite, il y a la Javadoc, qui elle doit être utilisée pour décrire les méthodes et les classes. Il est possible de négliger les accesseurs/mutateurs s'il n'y a pas de traitement fait dans ceux si. C'est la seule exception qui doit être faite cependant. Une description simple de la méthode et la classe est normalement suffisante. De plus, la Javadoc sert comme documentation au projet et son importance augmente considérablement lorsqu'il s'agit d'une librairie ou d'un cadre d'application.

Variable

Tous les programmeurs savent ce qu'est une variable. C'est un des premiers principes de programmations enseignés aux étudiants. Il est important de faire des rappels dans le cas des variables. Elles doivent avoir un nom significatif et un type significatif. Elles doivent avoir la bonne portée et la bonne visibilité.

Les variables sont importantes dans un programme parce que c'est avec elles que le code est produit, et vient qu'à vouloir dire quelque chose. Elles sont utilisées pour tout sortant d'opérations, que ce soit des additions mathématiques, pour stoker du texte, ou bien des objets complexes qui seront utilisés dans l'exécution du programme.

Il est donc primordial que les variables soient toujours bien nommées, significatives, utiles et surtout du bon type. Si elle ne respecte pas ces critères, les variables peuvent devenir une source de confusion et être dur à comprendre, ce qui occasionnera des problèmes dans la maintenance du logiciel.

Les instructions

Les instructions représentent tout ce qui donne un logiciel fonctionnel. Quelque millier d'instructions pour produire des logiciels de bonne, ou mauvaise qualité, des jeux vidéo, au logiciel mobile. Les instructions sont le corps et l'âme même d'un logiciel. C'est pourquoi il est important d'en prendre soins et de bien les écrire.

Savoir comment utiliser les bonnes instructions de contrôle, quand les utiliser et pourquoi les utiliser est primordial. Tous les ingénieurs comprennent ce que fait un « if », mais bien peut savent comment bien utiliser les itérateurs, les types énumérés (enum), quand utiliser un for, un while ou un for each. Pourquoi est-ce qu'un goto est si mauvaise. Ce sont des choses qui sont apprises, ou bien il est présumé que cette connaissance existe déjà.

D'autres types d'instruction reste un peu plus dans l'ombre, comme l'utilisation de la récursivité ou bien l'utilisation de liste et tableau dans les méthodes.

Garder les choses simples est la clé de la réussite avec les instructions. Une méthode ne doit jamais être trop complexe, contenir plusieurs comportements. Il est important de revoir c'est principe est de les faire pratiquer aux étudiants qui seront de nouveaux développeurs.

Gestion d'erreur et journalisation

Un aspect un peu obscur du développement est la journalisation et la gestion d'erreur. La pensée populaire d'un universitaire est qu'on ne doit jamais avoir d'erreur exposée dans un programme. Un « stack trace » est considéré comme une abomination et ne devrait jamais arriver.

La triste réalité, c'est que faire un logiciel commercial qui répond à ces critères est virtuellement impossible. Il y aura toujours ces fameux « stack trace » et des erreurs arriveront, et ce, même sur du logiciel en production.

Cependant, ce qui doit être fait est une gestion élégante de ces erreurs. Quoi faire quand elles arrivent. Comment les « loger », les utiliser pour améliorer le programme et surtout à les rendre significatives.

La journalisation est un art qui s'acquiert seulement avec de l'expérience et une connaissance approfondie du logiciel sur lequel on travaille. Savoir quoi garder, qu'est qui ajoute de la valeur, ou ce doit allez, dans un fichier, sur le système out, est-ce que si ce type d'erreur arrive c'est vraiment grave à envoyer un courriel, SMS ou autre alerte directement à l'équipe de support, ce sont tous des petites choses, qui ne sont pas vues.

Les tests

Les tests sont déjà couverts par un autre cours, soit LOG240. Par contre, il est toujours important de tester un logiciel et écrire des tests est une forme d'amélioration au code. Être en mesure de rendre on code testable avec diverse technique, comme l'injection de dépendance ou bien utiliser des « mock object » par exemple.

Même si un autre cours le couvre il est de mise de revenir sur les concepts parce qu'ils sont importants et que les étudiants doivent les maîtriser.

Discussion

Il est important de comprendre que l'auteur ne veut écorcher de cours, d'égo, ou quoi que ce soit avec ce projet. Ce doit être vu comme une suggestion pour l'amélioration au corpus de cours en génie logiciel et aux technologies de l'information de l'École de technologie supérieure. Les cours qui y sont déjà donnés sont enrichissants, quelques améliorations sont de mises, et je crois que le programma pourrait bénéficier d'un cours extrêmement technique comme celui-ci.

Au moment d'écrire ces lignes, je travaille à temps plein dans une entreprise, 8D Technologies depuis 8 mois, ou j'ai un rôle de développeur. Je crois qu'il est erroné de penser que tous les ingénieurs logiciels deviendront des gestionnaires de projet, chef d'équipe, analyste, ou autre tâche dans le monde logiciel qui ne requiert pas de programmer. Chez 8D, sur sept finissants de l'ÉTS six sont des développeurs à temps plein. C'est un bien petit échantillon, j'en conviens, mais je crois que même pour des emplois qui sont plus tournés vers la gestion ou l'analyse, il est important d'apprendre ces concepts. Rien n'est plus enrichissant que d'avoir une discussion avec les chefs de projet et de pouvoir parlé technique et que tout le monde se comprend et qu'ils puissent apporter des solutions aux problèmes exposés.

Amélioration potentielle

Je suis un néophyte dans la création des cours, je n'ai aucune expérience en pédagogie. Mon expérience se situe dans le développement Java. Les séances de cours et de travaux pratiques, s'ils étaient donnés par moi, seraient biaisées vers le monde Java. Je n'ai pas d'expérience significative dans le monde Microsoft. Par contre, je crois que les concepts de ce cours sont agnostiques des technologies. De la journalisation faite en Java où en C#, Python, Ruby, C++… reste de la journalisation, la seule différence notable sera la librairie qui sera utilisée pour l'exercice.

La plupart des concepts présentés sont intemporels. Un if sera toujours un if, peu importe le langage. Tout le monde le sait, la technologie est un monde qui évolue rapidement. Des nouveaux principes arrivent chaque année dans notre domaine. Par exemple, je n'ai jamais mentionné de programmation orientée Aspect ou des annotations, qui soyons franc, rendre la vie d'un développeur plus facile. Il serait possible de miser sur de tels concepts pour améliorer le cours. Et ce ne sont que des exemples, car il y a beaucoup de technologie qui s'ouvre dans le domaine en ce moment avec le « boom » mobile.

Conclusion

Ce projet m'a permis de toucher à une lacune du programme de génie logiciel et des TI de l'ÉTS. Bien que le programme soit bien fait et bien enseigné, il faut toujours trouver des moyens pour le rendre meilleurs et je crois réellement que d'ajouter un cours comme celui-ci le rendra meilleur.

Les économies de coût sont considérables pour les entreprises, et le fait d'avoir des leaders techniques dans une équipe aide grandement à la prise de décision. Rappelons cependant que ce n'est pas parfait et que la technologie évolue rapidement. Un cours comme celui-ci devra être donné par un enseignant avare de connaissance, de technologie et il devra être à l'affût des derniers développements.

Référence

Cours LOG 792 Projets de fin d'études en génie 2011. Site du projet de fin d'études en génie. http://cours.logti.etsmtl.ca/log792.
Consulté le 22 février 2011.

MCCONNELL S., Code Complete: A Practical Handbook of Software Construction. Microsoft Press; 2nd édition, 2004, 960 p.

Annexes

Information de la VM

- Nom d'usager : PFE
- Mot de passé : root

Version

- Jira 4.3
- Hudson 1.398
- SVN 1.6.2.12
- Nexus 1.9
- Sonar 2.6