RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS LOG792 PROJET DE FIN D'ÉTUDES EN GÉNIE LOGICIEL

INTÉGRATION CONTINUE POUR LOG240

MAXIME THIBEAULT
THIM20068300

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur superviseur

Alain April

MONTRÉAL, 12 DÉCEMBRE 2011
AUTOMNE 2011

REMERCIEMENTS

Nous aimerions émettre un merci tout spécial à Carlos Teodore Monsalve Artega pour son temps et son appui en tant que chargé de laboratoire.

INTÉGRATION CONTINUE POUR LOG240

MAXIME THIBEAULT
THIM20068300

RÉSUMÉ

Pour le cours de LOG240 : Maintenance et Tests, le chargé de laboratoire ainsi que le professeur responsable du cours désirent ajouter un processus d'intégration continue aux différents laboratoires du cours. Plusieurs critères ont été choisis et évaluer afin de déterminer lequel de « CruiseControl » ou « Hudson » serait le logiciel le plus approprié pour les besoins du cours et pour répondre au besoin du processus d'intégration continue. « Hudson » a été choisi et une procédure générique d'implantation est disponible.
L'intégration continue sera assurée par l'interaction entre « Hudson » et les différents outils que le chargé de laboratoire décidera d'ajouter dans chacun des laboratoires.
Le processus d'intégration continue qui devrait être implanté permettra aux développeurs d'exécuter automatiquement la construction, les tests, et les activités de déploiement permettant d'obtenir un logiciel final.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

IC ou CI	Intégration continue (ou de l'anglais « Continuous Integration »)

INTRODUCTION

Au cours des dernières années, plusieurs nouvelles techniques pour le développement et l'intégration de logiciels ont vu le jour. En particulier, l'intégration continue connait un essor considérable. Souvent considérée à tort comme une panacée à tous les problèmes ou encore une solution ayant de piètres résultats, cette étude cherche à mieux comprendre et appliquer la pratique de l'intégration continue tout en considérant les points de contention.

CHAPITRE 1

DESCRIPTION DU PROJET

1.1 Contexte

Dans notre cas particulier, nous devrons considérer l'application de l'intégration continue selon les besoins des laboratoires du cours LOG240 : Tests et maintenance. Ce cours contient cinq laboratoires distincts portant chacun sur certains points particuliers :

- Laboratoire 1 : Configuration de l'environnement
- Laboratoire 2 : Correction de défauts
- Laboratoire 3 : Nouvelle fonctionnalité et communication avec le client
- Laboratoire 4 : Cas de test
- Laboratoire 5 : Tests unitaires et intégration

Chaque laboratoire utilisera une machine virtuelle contenant tous les outils nécessaires à la réalisation du laboratoire. Aussi, chaque laboratoire sera bâti sur la machine virtuelle du laboratoire précédent.

1.2 Problématique

Afin de combler les besoins des laboratoires, des outils relatifs à l'intégration continue devront être intégrés à l'environnement de développement déjà existant afin de permettre aux étudiants d'appliquer cette pratique, mais aussi de comprendre les objectifs et les limites de l'intégration continue. Pour l'instant, il n'existe aucun processus implémenté dans le cadre du laboratoire en rapport à l'intégration continue.

1.3 Description du projet

Afin de permettre aux étudiants et aux chargés de laboratoire d'utiliser efficacement l'intégration continue, le projet consiste à ajouter des processus relatifs à l'intégration

continue à chacun des laboratoires en utilisant un logiciel d'intégration continue. Il sera important de déterminer le meilleur logiciel à utiliser pour permettre l'intégration continue.

1.4 Objectifs

Le principal objectif de cette étude est de déterminer ce qu'est l'intégration continue ainsi que les implications de l'application de l'intégration continue dans le processus de développement. Ensuite, à partir des résultats des recherches initiales, la pratique de l'intégration continue devra être appliquée aux laboratoires de LOG240. Il sera aussi essentiel de synthétiser les grandes lignes de l'intégration continue afin que les intervenants et les élèves puissent comprendre et utiliser correctement les outils du laboratoire.

Dans le cadre de cette étude, nous nous limiterons à deux logiciels dédiés à l'intégration continue : « CruiseControl » et « Hudson »

Finalement, afin d'obtenir des résultats qui pourront être répliqués, il sera nécessaire de produire les procédures et les instructions nécessaires à l'implantation et à l'utilisation de l'intégration continue. Il sera en outre essentiel d'indiquer les problèmes les plus communs ainsi que leurs solutions. Produire seulement les instructions minimales sans la compréhension du processus amènerait une application biaisée et une attente indue qui doit être enrayée à l'aide de documentation plus complète et d'explication complémentaire.

1.5 Méthodologie

Afin de pouvoir remplir ces objectifs, deux manuels seront créé afin, premièrement, de permettre l'installation des éléments clés du processus d'intégration continue, et, deuxièmement, de permettre une compréhension suffisante du processus afin de permettre une application du processus fidèle aux meilleurs pratiques.
Dans un premier temps, il sera nécessaire de faire une courte revue littéraire afin de nous permettre de déterminer la portée et l'importance de chacune des sources. Ensuite, nous devrons synthétiser ce qu'est l'intégration continue ainsi que les préalables et les

conséquences de son application au processus de développement. Ensuite, il sera nécessaire de déterminer les conditions et les attentes dans l'application de l'intégration continue grâce à des rencontres avec les intervenants. Il sera essentiel par la même occasion de concevoir une liste d'activités ainsi qu'une procédure d'implantation de l'intégration continue au niveau conceptuel.

Dans un deuxième temps, à l'aide des informations obtenues et des guides précédemment créés, les solutions possibles pour chacun des logiciels de cette étude (« CruiseControl » et « Hudson ») seront évaluées et comparées. Une fois ces étapes complétées, un des deux logiciels sera choisi et les instructions d'intégration et d'utilisation spécifiques aux laboratoires seront créées.

1.6 Hypothèses

Plusieurs logiciels seront nécessaires durant l'implantation de l'intégration continue. Pour cette étude, nous supposerons que le seul logiciel qui nécessitera une configuration directe sera le logiciel d'intégration continue. Donc, nous supposerons que chaque outil que le logiciel d'intégration continue utilisera (tel que Maven 3, Subversion, ou QALabs) sera configuré de façon indépendante et donc leurs installations ne seront pas couvertes par cette étude de façon directe.

Aussi, vu la nature changeante des différents laboratoires d'un semestre à l'autre, nous impliquerons que les changements entre les laboratoires soient suffisants pour ne pas pouvoir être prévus. Pour contrer ce phénomène et assurer une plus grande durée de vie utile à la documentation, seule une procédure générique sera proposée pour l'implantation de l'intégration continue.

CHAPITRE 2

REVUE DE LA LITTÉRATURE

2.1 Construction journalière

Daily build and feature development in large distributed projects. **Karlsson, Even-André, Andersson, Lars-Göran et Leion, Per. 2000.** New York : ACM, 2000. 22nd international conference on Software engineering . pp. 649-658. 1-58113-206-9.

L'article de Karlsson et al. démontre l'application de la construction journalière à deux projets différents dans deux entreprises différentes. L'article discute des avantages et des inconvénients de la construction journalière pour ensuite décrire les étapes qui ont été faites pour appliquer la construction journalière, les problèmes rencontrés et les leçons apprises de l'expérience dans chacune des deux entreprises.

McConnell, Steve. 1996. *Rapid Development.* s.l. : Microsoft Press, 1996. 978-1-55615-900-8.

Le chapitre sur la construction journalière dans le livre de McConnell décrit les avantages de la construction journalière et ensuite décrit comment implanter le processus ainsi que les problèmes principaux qui peuvent survenir.

2.2 Intégration continue

Duvall, Paul. 2007. Automation for the people: Continuous Integration anti-patterns. *developerWorks.* [En ligne] 4 Décembre 2007. [Citation : 29 Octobre 2011.] http://www.ibm.com/developerworks/java/library/j-ap11297/index.html.

Duvall décrit plusieurs mauvaises pratiques souvent utilisées lors de l'implantation continue ou encore plusieurs mauvaises pratiques qui peuvent s'installer avec le temps. Pour chaque problème souligné, Duvall explique rapidement la raison du problème et suggère des avenues de solution pour résoudre les problèmes.

Duvall, Paul M., Matyas, Steve et Glover, Andrew. 2007. *Continuous Integration: Improving Software Quality and Reducing Risk.* s.l. : Addison-Wesley Professional, 2007. ISBN-13: 978-0-321-33638-5.

Duvall et al. propose à travers de ce livre un recueil complet sur tous les aspects de l'intégration continue jumelant la théorie et la pratique à une description ordonnée des différents sous-processus de l'intégration continue.

Fowler, Martin. 2006. Continuous Integration. *Martin Fowler.* [En ligne] 01 Mai 2006. [Citation : 29 Septembre 2011.] http://martinfowler.com/articles/continuousIntegration.html.

Cet article de Martin Fowler est une réédition d'un article qu'il a publié en 2000 et qui a amorcé un mouvement de l'intégration continue parmi les masses. Dans cet article, Fowler décrit le processus de l'intégration continue et suggère rapidement une implémentation possible du processus.

Humble, Jez et Farley, David. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* s.l. : Addison-Wesley Professional, 2010. 978-0-321-67025-0.

Pouvant être considérés une suite du livre de Duvall, Continuous Integration, Humble et Farley décrivent les processus et les particularités permettant de pousser encore plus loin l'intégration continue. Leur idée principale est de permettre d'établir un processus où le logiciel résultant d'une construction est toujours prêt à être déployé après avoir passé tous les tests nécessaires.

Ostermeier, Daniel et Sankey, Jason. 2009. Continuous Integration Myth: A New Practice. *a little madness.* [En ligne] 9 Janvier 2009. [Citation : 11 Décembre 2011.] http://www.alittlemadness.com/2009/01/09/continuous-integration-myth-a-new-practice/.

Dans cet article, Ostermeier et Sankey discutent des débuts de l'intégration continue mettant un accent sur une tentative de découvrir les racines de l'intégration continue.

Walls, Don. 1999. Integrate Often. *Extreme Programming.* [En ligne] 1999. [Citation : 29 Septembre 2011.] http://www.extremeprogramming.org/rules/integrateoften.html.

La programmation extrême est un processus de développement nécessitant plusieurs règles. « Integrate Often » est une de ces règles et une des premières apparitions de l'intégration continue.

2.3 Outils d'intégration continue

John, Smart Ferguson. 2008. *Java Power Tools.* s.l. : O'Reilly Media, Inc., 2008. 978-0-596-52793-8.

Ferguson présente dans ce livre plusieurs des outils qui seront utilisés dans ce projet incluant entre autres Maven, CruiseControl et Hudson. L'installation et l'utilisation de chacun des outils sont décrites en détail.

ThoughtWorks. 2011. CI Feature Matrix. *ThoughtWorks.* [En ligne] 27 Mai 2011. [Citation : 29 Novembre 2011.] http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix.

Le site de ThoughtWorks contient une liste des fonctionnalités pour un grand nombre de serveurs d'intégration continue incluant « CruiseControl » et « Hudson ».

CHAPITRE 3

INTÉGRATION CONTINUE

3.1 Description

Le terme « intégration continue » a été publicisé particulièrement par un article de Martin Fowler (Fowler, 2006) ainsi que par l'utilisation de l'intégration continue comme une des douze pratiques du processus de développement « Extreme Programming » (Walls, 1999).
Martin Fowler décrit l'intégration continue comme suit :

> ***« Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily – leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. »***[1]

L'intégration continue est une évolution de la méthode de construction journalière (Duvall, et al., 2007). Les processus de l'intégration continue constituent un complément aux processus existants d'autres méthodes de développement (Duvall, et al., 2007).
Duvall suggère que l'intégration continue ne s'arrête pas seulement à la construction continue et à l'exécution des tests à chaque construction, mais suggère aussi l'utilisation d'analyse statique ainsi que l'implantation d'un pipeline de construction automatisé permettant le déploiement de chaque version du logiciel.

[1] (Fowler, 2006)

3.2 Processus

De façon générale, l'intégration continue est utilisée plus particulièrement en conjonction avec des processus itératifs bien qu'il soit possible de l'inclure dans d'autres types de processus de développement. Par exemple, comme mentionnée dans la section précédente0, l'intégration continue est utilisée explicitement dans la méthode de développement « Extreme Programming ».

L'intégration continue permet de supporter plusieurs aspects fondamentaux de la construction logicielle telle que décrite par le « Software Body of Knowledge » (SWEBOK). En particulier, l'intégration continue permet :

- de diminuer la complexité grâce à l'utilisation d'outils d'analyse statique;
- d'anticiper les changements grâce à une version des logiciels en développement qui peut être intégré et déployé à tout moment dans les environnements requis;
- de construire pour la vérification grâce encore une fois à des outils d'analyse statique, mais aussi grâce à des tests automatisés.

Les étapes typiques d'un processus d'intégration continue se résument à ceci[2] :

1. Un développeur soumet du code
2. Le serveur d'intégration continue détecte un changement
    a. Il télécharge le code le plus à jour
    b. Il exécute les scripts de construction
3. Le serveur d'intégration continue génère une rétroaction.
4. Retour à 1.

Toutefois, ces étapes ne sont que des points de départ. Chacune de ces étapes peut avoir des particularités ainsi que des modifications pour mieux s'adapter à un processus déjà existant ou tout simplement pour minimiser certains risques reliés au logiciel en développement.

---

[2] (Duvall, et al., 2007), p. 5

3.2.1 Bénéfice

La raison d'être de l'intégration continue est de s'assurer que l'application en développement s'intègre correctement en tout temps[3]; que l'application en développement puisse être construite afin d'obtenir une version utilisable en tout temps. Pour ce faire, les processus de l'intégration continue tentent de pallier quatre risques principaux mentionnés par Duvall :

- Manque de version utilisable du logiciel en développement[4]
- Découverte tardive des défauts[5]
- Manque de visibilité de l'avancement du projet[6]
- Mauvaise qualité du logiciel[7]

Le premier risque, le manque de version utilisable du logiciel en développement, provient souvent de l'impossibilité de construire régulièrement une version qui peut être déployée du logiciel. Plusieurs raisons telles que le manque de construction globale régulière, le manque d'automatisation du processus de construction, ou encore une construction nécessitant un environnement de développement graphique (IDE), voir un processus manuel de construction peut entraver le bon déroulement d'un processus d'intégration ajoutant des heures de travail souvent à des moments clés du développement où aucun réel développement n'est fait puisque tous les développeurs tentent de faire fonctionner tous les modules et les fonctionnalités ensemble souvent pour la première fois. L'utilisation de processus d'intégration continue permet dans ces cas d'avoir à tout moment une version complète et utilisable du logiciel.

---

[3] (John, 2008), p.268

[4] (Duvall, et al., 2007), p. 49-53

[5] Op. cit., p. 53-55

[6] Op. cit., p. 55-57

[7] Op. cit., p. 57-61

Le deuxième risque, la découverte tardive des défauts, se produit généralement dans des circonstances similaires, mais peut aussi se déclarer lorsque les tests effectués sur le logiciel ne sont pas complets et que certaines fonctionnalités ou certaines corrections de problèmes entraînent d'autres défauts dans des sections du code qui semblait sans risque. L'utilisation de processus d'intégration continue permet une meilleure adhérence aux standards de programmation du projet, des tests effectués régulièrement de façon automatisée, ainsi qu'un historique des problèmes dans le temps.

Le troisième risque, le manque de visibilité de l'avancement du projet, se produit particulièrement lorsque le suivi des versions de l'application en développement est inadéquat. Il est aussi possible d'avoir un manque de visibilité lorsque la documentation relative à un projet est obsolète. L'utilisation de processus d'intégration continue permet un suivi centralisé des versions de l'application en développement ainsi que l'intégration d'outils permettant la génération automatique de documentation incluant des manuels utilisateurs, de la documentation technique telle que « javadoc », ainsi que des diagrammes uml du système.

Finalement, le quatrième risque, la mauvaise qualité du logiciel, peut se produire lorsque des défauts potentiels s'introduisent dans le logiciel dû à un mauvais design du logiciel, aux standards du projet qui sont mal suivis, ou encore à un logiciel trop difficile à maintenir. L'utilisation de l'intégration continue permet d'automatiser les vérifications quant à la qualité grâce à des outils d'analyse statique ainsi que des outils d'analyse du code. Ceux-ci étant exécuté de façon automatique à chaque construction du logiciel permettent de plus rapidement repérer ces défauts potentiels.

3.2.2 Inconvénients

Avant d'appliquer les processus relatifs à l'intégration continue, il est nécessaire de prendre en compte certaines particularités que demandent les processus.

Tout d'abord, les processus demandent un certain niveau de discipline de la part des développeurs. En ce sens, chaque développeur utilisant l'intégration continue doit suivre les sept règles suivantes[8] :

- Mettre à jour le code fréquemment (au moins une fois par jour)
- Ne pas mettre à jour le code s'il ne fonctionne pas correctement (qui passe tous les tests)
- Réparer les constructions brisées immédiatement
- Écrire des tests automatisés et les inclure dans le processus de construction.
- Tous les tests et les inspections doivent passer en tout temps.
- Construire le logiciel complet de façon locale (en plus du serveur d'intégration continue) afin de réduire les mises à jour de code non fonctionnel.
- Éviter d'obtenir une version du logiciel qui ne fonctionne pas correctement (pour mettre à jour le code local ou entreprendre une nouvelle fonctionnalité ou régler un nouveau problème).

Chacune de ces règles demande que les développeurs se responsabilisent quant à la qualité du code introduit à chaque mise à jour.

3.3 Constructions automatisées

L'étape principale du processus d'intégration continue consiste à détecter les mises à jour récentes du code de l'application en développement afin de pouvoir exécuter le script de construction nécessaire.

---

[8] (Duvall, et al., 2007), p. 39, 44.

La première étape dans l'exécution du script sera de construire l'application en développement. Il est important à ce stade d'utiliser des outils en ligne de commande afin qu'il soit possible d'automatiser le processus.
Bien que la simple construction de l'application en développement soit assez d'étape pour exécuter l'application et en faire une version pouvant être déployée, l'intégration continue ne s'arrête pas ici. En fait, Duvall spécifie que l'utilisation d'un processus de construction automatisé seulement n'amènera pas les bénéfices associés à l'intégration continue[9].

3.4 Tests continus

La deuxième étape dans l'exécution du script de construction sera d'exécuter tous les tests automatisés. Considérant que certains systèmes peuvent contenir des milliers de tests, il est important de considérer les différentes catégories de tests, ainsi que leurs implications sur les performances du processus. La section 3.8.1 discute ces implications ainsi que de solutions à des problèmes de performance par rapport aux tests. La différentiation des différents types de tests est importante afin de permettre d'exécuter les tests rapides en premier.
Afin de pouvoir avoir un processus d'intégration continue stable, il est important que chacun des tests soit stable et qu'il retourne le même résultat à chaque exécution, peu importe la condition du système.

3.4.1 Tests unitaires

Duvall définir les tests unitaires comme des tests permettant de vérifier le bon fonctionnement d'un petit élément, généralement une classe, dans le logiciel en développement[10]. Un test unitaire ne devrait pas avoir de dépendance extérieure telle qu'un fichier de configuration ou une base de données afin de réduire le temps d'exécution au

[9] (Duvall, et al., 2007), p. 35

[10] Op. cit., p. 132

minimum. Duvall suggère d'exécuter les tests unitaires toutes les fois où le logiciel est construit sans exception[11].

3.4.2 Tests de composantes

Les tests de composante testent l'interaction entre certaines composantes spécifiques. Les tests de composantes devraient garder au minimum les dépendances extérieures afin de garder le temps d'exécution au minimum. Les tests d'intégration sont généralement des tests de composantes.

3.4.3 Tests système

Les tests système requièrent généralement toutes les dépendances du système et testent l'interaction des interfaces utilisateurs ainsi que l'interaction avec les différents systèmes dont le logiciel en développement dépend. Ces tests doivent en général tester tout le système dans son entier. Les tests de performance et de stabilité entrent dans cette catégorie.

3.4.4 Tests fonctionnels

Les tests fonctionnels sont semblables aux tests systèmes, par contre, ils sont exécutés du point de vue d'un utilisateur, en essayant d'imiter ce qu'il ferait pour accéder au système. Les tests fonctionnels inclus entre autres les tests d'acceptation.

---

[11] Op. cit., p. 141

3.5 Inspections continues

Plusieurs méthodes d'inspections existent, particulièrement, la revue de code par les paires est une méthode très connue et très utilisée. Duvall émet cependant une mise en garde par rapport à ce genre de méthode. En effet, puisque l'être humain a une tendance à être émotionnel dans sa prise de décision, il s'avère que certains commentaires peuvent être omis pour diverses raisons[12].

Pour plusieurs raisons, Duvall suggère l'utilisation d'outils d'analyse statique afin de pallier ces problèmes[13]. Dans les prochaines sections, plusieurs de ces outils et leurs applications possibles seront décrits.

3.5.1 Complexité du code

Plusieurs indicateurs peuvent aider à déterminer la complexité du code d'une application en développement. Plus particulièrement, Duvall suggère l'utilisation de « Cyclomatic Complexity Number »[14]. Le « Cyclomatic Complexity Number » représente le nombre de points de sortie à travers une méthode. Duvall suggère qu'une valeur plus grande à 10 indique un problème dans la méthode en inspection[15]. Il suggère de surcroit que le nombre de cas de test soit égal au nombre indiqué par le « Cyclomatic Complexity Number »[16].

---

[12] (Duvall, et al., 2007), p. 162

[13] Op. cit., p. 163.

[14] Op. cit., p. 167

[15] Ibib.

[16] Op. cit., p.168-169.

3.5.2 Dépendance des modules

Afin de déterminer la stabilité d'un objet lorsque des objets desquels ils dépendent sont modifiés, Duvall suggère de compter le nombre de liens entre les différents objets. Les indicateurs « Afferent coupling » (aussi appelé « Fan In ») et « Efferent Coupling » (aussi appelé « Fan Out ») permettent de déterminer la stabilité de l'objet grâce à la formule suivante :

$$\text{Instabilité} = \frac{\text{« Efferent Coupling »}}{\text{« Efferent Coupling »} + \text{« Afferent Coupling »}}$$

3.5.3 Duplication du code

La duplication du code amène son lot de problème. De façon générale, la duplication de code nécessite la mise à jour conjointe de toutes les parties dupliquées ce qui entraîne une augmentation du coût de maintenance. Une incertitude envers le code peut aussi être une conséquence de la duplication puisqu'il peut être difficile lors d'un changement de ne pas involontairement laisser de côté une partie de code dupliquée.
La découverte rapide de code dupliquée ainsi que l'éradication des duplicatas permet ainsi d'augmenter la confiance envers le système ainsi que de réduire les coûts de maintenance.

3.5.4 Couverture du code

La couverture du code existe sous deux formes principales. La première forme est l'analyse des lignes couvertes. Dans ce cas, l'outil de couverture du code analyse les lignes qui sont exécutées lors des tests afin de déterminer quelles lignes de code ne sont jamais testées.
La deuxième forme est l'analyse des branchements. Dans ce cas, l'outil de couverture du code vérifiera quel branchement (bloc conditionnel) a été activé durant l'exécution des tests.

Duvall suggère de séparer l'analyse de couverture du code en fonction des différents types de tests et de créer un rapport pour chacun[17].
Puisque les tests doivent contenir du code particulier afin de déterminer les sections qui sont ou pas exécutés, l'exécution des tests durant l'analyse de couverture du code sera plus lente que la normale, ainsi il serait essentiel d'éviter d'exécuter des tests de performance ou des tests de charges durant l'analyse.

3.6 Déploiement continu

Duvall spécifie que sans un déploiement réussi du logiciel, il n'existe pas réellement[18]. Ainsi, le déploiement continu est l'étape finale de la construction d'un logiciel en développement.

3.6.1 Nom de version

La première étape du déploiement continue consiste à déterminer un nom pour la nouvelle version du logiciel en développement. Ce nom permettra de repérer l'historique de la version construite du logiciel et ainsi de repérer facilement l'ordre chronologique de création ainsi que les fonctionnalités et les problèmes présents dans cette version spécifique du logiciel.
Une fois le nom choisi automatiquement par le système de construction, une version du code source sera sauvegardée sous le nom choisi. De façon générale, lorsqu'un serveur de source est utilisé, une branche en lecture seule sera créée dans le serveur de source.

---

[17] (Duvall, et al., 2007), p. 184.

[18] (Duvall, et al., 2007), p. 189.

3.6.2 Utilisation et installation

Pour terminer le déploiement, le logiciel en développement sera installé sur une copie d'une machine contenant un environnement de production puis tous les tests seront exécutés de nouveau.

3.7 Rétroaction

La rétroaction du processus à l'usager est une des parties les plus importantes du processus d'intégration continue. Trop d'information et l'utilisateur n'y porteront bientôt plus aucune attention; pas assez et l'utilisateur n'aura aucune idée de l'état du système ce qui rendrait inutile tout le processus d'intégration continue.
Dans tous les cas, un point important à considérer est qu'il est primordial que le système de construction envoie les rétroactions d'échec le plus rapidement.

3.7.1 Information transmise

Pour toutes les tentatives de construction d'un logiciel en développement, tous les fichiers contenant les comptes-rendus des différents outils devraient être disponibles. Évidemment, il serait un peu contraignant de transmettre toute l'information nécessaire à chaque développeur pour chaque rétroaction. C'est pour cette raison que des outils tels que « CruiseControl » et « Hudson » existent. L'utilisation de ces outils permet de transmettre l'information nécessaire tout en prenant en compte la méthode de transmission. Aussi, la centralisation de l'information permet de garder un historique des constructions passées ce qui permet de faciliter le suivi du projet.

3.7.2 Moyen de transmission

Plusieurs méthodes peuvent être utilisées pour transmettre une rétroaction aux différents partis impliqués. Les méthodes plus conventionnelles telles que les courriels et les messages

à des téléphones mobiles peuvent être utilisées. Duvall suggère aussi l'utilisation de moyens moins traditionnels tels que des boules de lumières (« Ambient Orb »)[19].

3.7.3 Intérêt des parties responsables

Durant la mise en place des processus de rétroaction, il est nécessaire de considérer les différents acteurs et leur interaction avec le système incluant la nécessité ou non de recevoir des messages de rétroaction.

De façon générale, l'envoi de messages à tous les acteurs d'un projet toutes les fois où le système de construction exécute une construction peut être excessif. Duvall note qu'il est important que les acteurs d'un projet n'apprennent pas à ignorer les messages du processus d'intégration continue[20].

D'un autre côté, chaque acteur doit avoir l'information nécessaire à la détection rapide des défauts potentiels. En effet, Duvall précise que la réduction du temps requis entre l'introduction d'un défaut et sa résolution permet d'économiser temps et argent[21].

3.8 Résolution de problème

Bien qu'un processus d'intégration continue puisse réduire nombre de risques et faciliter la transmission d'informations aux partis concernés, il est important de mentionner que certains problèmes peuvent survenir et que, à cause de ces problèmes, le processus d'intégration continue peut devenir en soi un problème.

Dans cette section, plusieurs des problèmes les plus communs seront abordés, proposant des solutions et ciblant les causes les plus communes.

---

[19] (Duvall, et al., 2007), p. 214.

[20] (Duvall, et al., 2007), p. 207.

[21] Op.cit., p. 208.

3.8.1 Temps d'exécution des constructions

Dans certains cas, il est possible que le processus complet de l'intégration continue demande un investissement en temps d'exécution trop grand. En considérant l'organisation et le nombre des unités de compilation, le temps d'exécution et le nombre de tests, le temps d'exécution des différents outils d'analyse statique et le temps d'empaquetage et de déploiement nécessaire, il est facile de comprendre que le processus d'intégration continue peut rapidement augmenter.

3.8.1.1 Performance de la machine de construction

La première option à considérer pour réduire le temps d'intégration est de changer la machine servant à l'intégration continue. La nouvelle machine devrait être totalement dédiée à l'intégration continue. Aussi, idéalement, la nouvelle machine devrait être la plus performante sur le marché. Duvall démontre d'ailleurs les avantages d'avoir une machine puissante dans le cadre d'une entreprise[22].

3.8.1.2 Performance des tests

La seconde option consiste à garder le temps d'exécution des tests unitaires et des tests de composantes très bas. En effet, en prenant par exemple mille tests et en les réduisant en moyenne d'une seconde, il serait possible de réduire le temps d'exécution total de 16 minutes 40 secondes. Ainsi, pour certains types de tests, l'utilisation d'une limite de temps d'exécution peut s'avérer un avantage considérable. Évidemment, certains tests nécessitent plus de temps afin d'être exécutés correctement. Dans ce cas, il serait possible d'utiliser une

[22] (Duvall, et al., 2007), p.100.

technique de construction en étape comme mentionnée dans la section 3.8.1.5 ou encore d'exécuter des scripts de construction spécifiques à des moments précis (par exemple durant la nuit à chaque jour) plutôt qu'à toutes les constructions.

3.8.1.3 Modules

La troisième option consiste à diviser pour conquérir. Au niveau de la compilation, chaque module indépendant peut être divisé en plusieurs sous-modules indépendants ce qui permet de recompiler seulement les sous-modules nécessaires. La majorité des outils d'intégration continue permettent de gérer les dépendances entre différents modules.

3.8.1.4 Construction en parallèle

La quatrième option consiste à exécuter certaines parties de la construction sur des machines séparées. Évidemment, cette option a son lot de problème, et Duvall suggère d'utiliser cette option seulement en dernier recours[23].

3.8.1.5 Construction en étape

Dans certains cas, malgré toutes les stratégies mentionnées précédemment, il est possible que la construction du logiciel en développement s'avère trop longue. Dans ce cas, la construction par étape peut s'avérer la seule solution viable.
Afin d'utiliser la construction par étape, il est nécessaire de cibler combien de temps chaque processus prend et d'ensuite créer deux ou plusieurs constructions en série chacune contenant un certain nombre d'étapes. Ensuite, chaque étape sera exécutée à tour de rôle jusqu'à ce que la construction fonctionne ou qu'une des étapes produise des erreurs.

---

[23] (Duvall, et al., 2007), p. 96

L'idée de la construction par étape est d'exécuté les tâches permettant de cibler le plus grand nombre d'erreurs en premier pour avoir une rétroaction rapide à l'usager et d'ensuite lancer les processus plus longs afin de vérifier qu'aucune erreur ne se serait glissée dans l'application que les processus initiaux n'auraient pas repérée.
Duvall suggère d'utiliser la règle du 80/20 afin de maximiser les étapes. Pour se faire, il suggère d'utiliser 20% du temps de construction pour découvrir 80% des erreurs et d'ensuite lancer une seconde construction pour découvrir le 20% d'erreurs restantes.[24]

3.8.2 Implantation dans un projet existant

Dans le cas où l'intégration continue est implantée dans un projet existant déjà, la première étape sera d'automatiser le processus de construction. Dans certains cas, il sera possible d'intégrer les tests automatisés qui existent déjà, par contre, dans le cas où aucun test automatisé n'existe, il n'y a pas de solution miracle. La solution idéale sera d'ajouter progressivement des tests pour les nouvelles fonctionnalités en développement ou pour la résolution de problèmes. En effet, en ajoutant graduellement des tests, le processus d'intégration pourra détecter certains problèmes et la charge de travail ajoutée pour l'implantation de l'intégration continue restera relativement basse.

3.8.3 Support de plusieurs versions grâce à des branches

Pour le développement de certaines fonctionnalités ou pour d'autres raisons telles que le support de version spécifique à un client, il peut être décidé qu'une branche soit faite. Duvall note que l'intégration continue est exécutée sur la version principale du logiciel en développement[25]. Ainsi, même lors de la création de branches, toutes les modifications

---

[24] (Duvall, 2007)

[25] (Duvall, et al., 2007), p. 100

devraient être rapatriées dans la branche principale, dans la version principale, du logiciel en développement et non pas seulement mettre à jour la branche en développement sans mettre à jour la branche principale.

CHAPITRE 4

OUTILS D'INTÉGRATION CONTINUE

4.1 Serveur d'intégration continue

L'implantation de l'intégration continue demande qu'un logiciel serveur reste toujours actif et qu'il intercepte les ajouts de nouveau code au serveur de gestion de code. Bien qu'il soit possible de créer de toutes pièces des scripts permettant d'effectuer toutes les tâches reliées à l'intégration continue, plusieurs logiciels existent déjà sur le marché permettant d'effectuer toutes les tâches nécessaires. Dans notre cas, nous évaluerons deux logiciels en particulier : « CruiseControl » et « Hudson ».

4.2 Critères d'évaluation

Afin de déterminer correctement quels logiciels doivent être utilisés lors de l'implantation d'un processus d'intégration continue, il est impératif d'être en mesure d'évaluer de façon quantitative les différents candidats. Dans le cas d'une étude pour un projet en particulier où des processus existent déjà, il est aussi essentiel de considérer les outils et les processus courant lors de l'évaluation de nouveau logiciel relié directement à l'intégration continue.

4.2.1 Hypothèses

Les hypothèses déjà mentionnées dans la section 1.6 nous permettent d'émettre certaines restrictions quant aux critères d'évaluation qui devront être inclus. En effet, l'utilisation des logiciels particuliers indiquée à la section mentionnée ci-dessus demandera que les outils sélectionnés soient compatibles et s'intègrent bien entre eux. Dans le cas présent, il sera nécessaire de considérer les logiciels suivants : Maven 3, Subversion, JUnit, Jira, QALab.
Nous prendrons aussi comme hypothèse qu'il n'existe qu'un seul serveur de gestion de source par serveur d'intégration continue. Aussi, nous assumerons que l'intégration de Jira est facultative au bon fonctionnement de l'intégration continue. En fait, quoique reliés au

processus d'intégration continue, nous considérerons ici que la fermeture de bogue devrait être un acte réservé aux personnes associées aux projets plutôt que la responsabilité du serveur d'intégration continue.
Finalement, nous prendrons comme hypothèse que les analyses effectuées à l'aide de JUnit et QALab seront lancées de Maven 3 et que le serveur d'intégration continue ne sera responsable que de l'agrégation des résultats.

4.2.2 Code source et licence

L'utilisation d'un logiciel ayant un code source libre de droits permet la modification du code dans le cas où l'outil nécessiterait des fonctionnalités qui ne sont pas offertes directement. En outre, l'utilisation d'un logiciel ayant un code source libre de droits a plus de chance d'avoir les extensions nécessaires à coup moindre qu'un logiciel propriétaire.
Aussi, il est plus simple d'utiliser un logiciel ayant le même langage de programmation que les projets qu'il devra gérer. En effet, l'utilisation du même langage permet de plus facilement modifier le logiciel si nécessaire, mais aussi, cela permet de visualiser le code en cas de comportements inattendus afin de pouvoir soit les contournés, soit trouver une solution.

4.2.3 Logiciels supportés ou requis

Afin de pouvoir utiliser un processus d'intégration continue adapté au projet en cours, il est nécessaire que l'outil à utiliser supporte tous les outils nécessaires. Cette demande disqualifierait probablement d'emblée la majeure partie des logiciels d'intégration continue. Pour répondre à cette demande, les logiciels d'intégration continue permettent normalement l'exécution de commandes arbitraires ce qui permet l'intégration d'outils qui seraient normalement non supportés. Pour cette raison, même si un logiciel n'est pas supporté, si l'exécution de commandes arbitraires est supportée, le logiciel ne sera pas éliminé de la comparaison.

4.2.4 Communauté et espérance de vie

Lors de l'évaluation d'un outil d'intégration continue, il est essentiel de prendre en compte les aspects temporels. Ainsi, idéalement, un outil d'intégration continue devrait avoir un historique relativement long ainsi qu'une communauté active et des développeurs actifs. Dans le cas de l'historique, nous voulons nous assurer que le logiciel utilisé ne sera pas qu'un accident de parcours[26] et l'activité de la communauté et des développeurs est un indicatif de la facilité ou la difficulté à recevoir du support ainsi que la rapidité à s'adapter à des situations ou des besoins particuliers[27]. En outre, nous devons considérer la maturité des outils utilisés afin de diminuer les risques pour l'utilisation à long terme que nous pourrions en faire.

Afin de pallier en partie ce risque, il est aussi possible de considérer le langage d'implémentation de l'outil d'intégration continue dans le cas des logiciels distribués avec des licences à source libre de droits comme mentionné à la section 4.2.2. En effet, en utilisant un logiciel avec le même langage, il est plus facile pour les développeurs de modifier le logiciel selon leur besoin.

4.2.5 Serveur de source

Le serveur de source est le point de départ du processus d'intégration continue. En effet, le serveur d'intégration continue utilise le serveur de source pour déterminer les moments où une nouvelle construction doit être lancée. Afin de mieux préciser les conditions démarrant une construction, la plupart des serveurs d'intégration permettent d'effectuer un filtrage des informations du serveur de source plutôt que de démarrer dès qu'un nouveau changement est découvert.

---

[26] TODO : Continuous Integration, Evaluating CI tools

[27] TODO : Continuous Integration, Evaluating CI tools.

Aussi, certains serveurs d'intégration continue permettent de supporter plusieurs serveurs de source différents de façon concurrente ce qui peut être pratique lorsque plusieurs projets dans plusieurs départements différents utilisent chacun leur propre serveur de source, mais centralise les services d'intégration continue.

4.2.6 Construction du code source

Les serveurs d'intégration continue peuvent offrir certaines options pour la construction de code source qui peuvent aider grandement le processus d'intégration continue.
Pour améliorer la vitesse d'exécution de différents projets, le serveur d'intégration continue peut supporter l'exécution simultanée de la construction de plusieurs projets. Sans cette option de construction en parallèle, les projets sont généralement construits un après l'autre.
Une autre option disponible dans certains serveurs d'intégration continue est la possibilité d'exécuter des constructions distribuées sur plusieurs machines. Généralement, ce genre de construction nécessite des outils spécialisés même lorsque le serveur d'intégration continue supporte la construction distribuée.
Outre la vitesse d'exécution, il est aussi important de prendre en compte les moments où le processus d'intégration aura lieu. En général, la majorité des serveurs d'intégration continue prend en compte les modifications à partir d'un serveur de code source. Toutefois, certains serveurs d'intégration continue supportent aussi la possibilité de permettre à l'utilisateur de forcer la construction d'un projet même lorsqu'aucune modification n'a été effectuée au niveau du code source.
Aussi, la majorité des serveurs d'intégration continue permettent de prévoir un moment où certaines constructions seront effectuées. Grâce à cette fonctionnalité, il est possible d'exécuter des constructions, généralement avec des paramètres différents d'exécution, de façon journalière ou encore de façon hebdomadaire.

4.2.7 Rétroaction et publication

Le support de multiple façon de faire parvenir une rétroaction à l'utilisateur est essentiel au bon fonctionnement d'un processus d'intégration continue. La majorité des serveurs

d'intégration continue supporte des rétroactions à l'aide de courriel, par contre, d'autres options sont disponibles telles que les messages textes. La variété des moyens de rétroaction permet de mettre plus de nuances dans la rétroaction aux différents utilisateurs.

4.2.8 Interface utilisateur et configuration

La configuration d'un serveur d'intégration continue peut s'avérer complexe. Dans un environnement où de nouveaux projets sont ajoutés ou enlevés de façon régulière, il peut être important d'avoir accès à une interface utilisateur solide pour effectuer les modifications. Évidemment, dans le cas contraire, il est essentiel qu'un système de fichier de configuration stable puisse être utilisé..

4.3 Candidats

Comme mentionnés dans la section above, deux logiciels en particulier seront évalués soit : « CruiseControl » et « Hudson »

4.3.1 CruiseControl

Table 1: Information à propos de CruiseControl

Nom	CruiseControl
Site Web	http://cruisecontrol.sourceforge.net/
Distribution du code source	Source libre de droits sous licence BSD
Langage de l'implémentation	Java

Plusieurs logiciels se sont démarqués au cours de la dernière décennie depuis la parution initiale de l'article de Martin Fowler sur l'intégration continue[28]. CruiseControl a été un des premiers disponible avec un code source libre de droits. En fait, encore aujourd'hui, le code source de CruiseControl est distribué sous la licence BSD ce qui permet aux entreprises privées de modifier le logiciel selon leur besoin.

Le développement de CruiseControl a débuté aux alentours des années 2000[29]. Les premiers messages de la liste de distribution commencent cependant en octobre 2001. Durant les années suivantes, la liste de distribution fut particulièrement active jusqu'en 2008 après quoi les activités sur la liste ont diminué pour se stabiliser.

Malheureusement, CruiseControl ne supporte pas d'emblée Maven 3. Les versions antérieures sont supportées, mais pas la dernière. Il serait possible de modifier le code source de CruiseControl pour prendre en compte Maven 3 en prenant comme modèle Maven 2, mais dans tous les cas, il sera nécessaire de prendre en compte ce problème lors de l'évaluation finale.

---

[28] (Fowler, 2006)

[29] (Ostermeier, et al., 2009)

4.3.2 Hudson

Table 2: Information à propos de Hudson

Nom	Hudson
Site Web	http://www.hudson-ci.org/
Distribution du code source	Source libre de droits sous licence MIT
Langage de l'implémentation	Java

Lors des années de développement de Sun Microsystem, plusieurs projets sont nés afin de supporter le développement de Java. L'un de ceux-ci fut Hudson. Hudson est développé en Java et son code source est disponible sous licence MIT. Bien que le code source soit disponible en version libre, le nom ainsi que les logos ont récemment été remis en cause à la suite de l'achat de Sun Microsystem par Oracle. Après consultation et délibération avec les développeurs et les membres de la communauté, une nouvelle version fut démarrée à partir de Hudson qui s'appelle Jenkins. À ce jour, les retombés de ces actions sont inconnus.
Hudson supporte Maven 3 et Subversion à l'aide d'extension déjà incluse dans le logiciel.

4.4 Évaluation des candidats

Pour déterminer le meilleur candidat, il est nécessaire d'évaluer l'adhérence aux critères d'évaluation mentionnés précédemment. Pour ce faire, nous utiliserons un tableau où chaque critère sera évalué selon son importance. Selon le niveau d'importance, un nombre de points sera attribué. Un critère d'importance haut aura 5 points d'attribués tandis qu'un critère importance moyenne aura 3 et un critère d'importance bas aura 1.

Table 3: Comparaison de CruiseControl et Hudson[30]

Critère	Importance	CruiseControl		Hudson	
Code source et licence					
Licence du code source libre de droits	Moyen	X	3	X	3
Utilise Java pour son code source	Moyen	X	3	X	3
Logiciels supportés ou requis					
S'exécute sous Linux	Haut	X	5	X	5
Ne requiers pas de logiciel supplémentaire en surplus du JDK	Moyen	X	3	X	3
Support d'exécutable arbitraire	Haut	X	5	X	5
Support de Subversion	Haut	X	5	X	5
Support de Maven 3	Haut		0	X	5
Support d'Eclipse	Haut	X	5	X	5
Support de QALab	Moyen		0		0
Support de JIRA	Moyen		0	X	3
Support de JUnit	Haut	X	5	X	5
Support de Trac	Haut	X	5	X	5
Support de cobertura	Haut		0		0
Support de uispec4j	Haut		0		0
Communauté et espérance de vie					
Plusieurs développeurs impliqués	Moyen	X	3	X	3
Future du projet stable	Haut	X	5		0
Serveur de source					
Filtrage des informations du serveur de	Moyen	X	3	X	3

[30] (ThoughtWorks, 2011)

source					
Support pour multiple serveur de source	Bas	X	1		0
Construction du code source					
Construction en parallèle de projets	Bas	X	1	X	1
Construction distribuée de projets	Bas	X	1	X	1
Permets de forcer la construction d'un projet à l'aide d'une interface utilisateur	Moyen	X	3	X	3
Prévoir des constructions dans le temps (journalière, hebdomadaire, etc.)	Haut	X	5	X	5
Rétroaction et publication					
Publication de la rétroaction par courriel	Haut	X	5	X	5
Publication de la rétroaction par exécutable	Moyen	X	3	X	3
Publication de la rétroaction par FTP	Bas	X	1	X	1
Publication de la rétroaction par Jabber	Bas	X	1	X	1
Publication de la rétroaction sur le bureau Windows	Bas	X	1	X	1
Publication de la rétroaction par Yahoo Messenger	Bas	X	1		0
Publication de la rétroaction par MSN Messenger	Bas		0		0
Publication vers un site Maven 3	Haut		0		0
Interface utilisateur et configuration					
Ajouter/Modifier/Supprimer des projets avec une interface utilisateur	Haut		0	X	5
Configuration de l'outil à l'aide de XML	Haut	X	5	X	5
			78		84

4.5 Résultats de l'évaluation

Comme nous pouvons le constater dans les résultats de cette évaluation côte à côte des logiciels, « Hudson » se classe au-dessus de « CruiseControl » par quelques points de différence. Cette différence est principalement attribuable à l'interface utilisateur présentée par « Hudson » afin d'ajouter, de modifier, et de supprimer des projets, ainsi qu'à son support de « JIRA » et de « Maven 3 ». Il est important de noter que « Hudson » et « CruiseControl » supporte tous deux l'exécution de commande arbitraire ce qui permettrait théoriquement de pallier n'importe quelle lacune grâce à des scripts spécialisés.

Ainsi, dans le cas du projet d'implantation, il serait recommandé d'utiliser Hudson. Par contre, il serait intéressant de refaire cette comparaison en utilisant Jenkins plutôt que Hudson.

CHAPITRE 5

IMPLANTATION DE L'INTÉGRATION CONTINUE

5.1 Hypothèse et contexte du système

Afin d'effectuer correctement l'implantation de processus d'intégration continue. Il est nécessaire de prendre en compte les différentes configurations déjà présentes dans le système.

Premièrement, le nom projet que les étudiants développent pour leur laboratoire s'appelle « FinanceJ ». Aussi, le projet contient du code source Java ainsi que les fichiers nécessaires à la construction du projet à l'aide de Maven 3.

Deuxièmement, le serveur de code source, Subversion, sera déjà installé sur les machines virtuelles. L'étudiant devrait avoir utilisé « /opt/svn » comme chemin pour accéder au serveur de code source pour le projet. Une organisation typique serait utilisée pour Subversion ce qui veut dire que trois répertoires seront utilisés soit « branches », « tags », et « trunk ». Le répertoire « trunk » contiendra la version principale du logiciel tandis que le répertoire « branches » contiendra dans ses sous-répertoires des embranchements du logiciel et le répertoire « tags » contiendra les marqueurs de version par rapport à la version principale.

Troisièmement, puisque la liste des outils qui sont utilisés pour la mise en place des tests et de l'inspection semble encore contenir certaines instabilités, l'installation de chacune des extensions ne sera pas décrite. Nous assumons que l'explication des procédures d'installation d'extension sera suffisante afin que l'étudiant ou le chargé de laboratoire puissent déterminer lui-même les requis de ses outils et les installer en conséquence.

Quatrièmement, nous supposerons que les fichiers utilisateurs pour Hudson seront installés dans « /opt/hudson » afin de poursuivre une certaine consistance avec les autres outils déjà installés.

Finalement, nous assumons que les logiciels suivants sont déjà installés et prêts à être utilisé :

- Ubuntu 10.04 LTS,
- Subversion 1.6,

- Maven 3,
- Httpd (Apache2) 2.2,
- Tomcat6,
- Sun JDK 6.

5.2 Hudson

L'installation et la configuration de Hudson nécessitent plusieurs éléments. La première étape sera d'acquérir les binaires de Hudson disponible à :
http://hudson-ci.org/download/war/2.2.0/hudson.war
Une fois l'acquisition des binaires, il sera possible d'exécuter directement Hudson à l'aide de la ligne de commande (« java –jar hudson.war »). Cette option permet le déploiement rapide de Hudson, mais ne permet pas d'assurer son exécution ni de l'utiliser à travers Httpd. L'utilisation de la ligne de commande ouvre un serveur http sur le port 8080 par défaut.

5.2.1 Configuration de Tomcat

La configuration de Tomcat nécessite de modifier le fichier de configuration :
« /etc/tomcat6/server.xml ».
La ligne « <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" /> » devrait être en commentaire. Afin de pouvoir utiliser ce connecteur lors de la configuration d'Httpd, il est nécessaire d'activer cette commande en retirant les marqueurs de commentaires « <!-- » et « --> » avant et après l'instruction
Une fois les modifications effectuées, un redémarrage de tomcat6 sera nécessaire :
« sudo service tomcat6 restart ».

5.2.2 Configuration de Httpd (Apache2)

Afin de configurer adéquatement Httpd, il sera nécessaire d'acquérir un lien adéquat entre tomcat6 et Httpd. En tant que connecteur entre Tomcat6 et Httpd, nous utiliserons le module « libapache2-mod-jk » qui pourra être installé à l'aide de :

« sudo apt-get install libapache2-mod-jk ».

La configuration de ce module nécessite la copie du fichier d'exemple de configuration et de modifier les valeurs pour les adapter à la machine virtuelle. La copie s'effectue à l'aide de la commande suivante :

« sudo cp /usr/share/doc/libapache2-mod-jk/httpd_example_apache2.conf /etc/apache2/mods-available/jk.conf ».

Ensuite, l'édition nécessitera de modifier le fichier jk.conf (« sudo vim /etc/apache2/mods-available/jk.conf ») afin que les répertoires pour Tomcat6 et Java pointent au bon endroit.

Afin d'activer le connecteur, il sera nécessaire de l'ajouter à Httpd en utilisant : « sudo a2enmod jk ». La prochaine étape nécessitera l'installation de l'archive contenant Hudson (hudson.war) dans Tomcat6 : « sudo rm -R /var/lib/tomcat6/webapps/hudson ; sudo cp ~/hudson.war /var/lib/tomcat6/webapps ». L'utilisation de la dernière commande supprimera une ancienne version de Hudson pour la remplacer par la version que nous avons acquise précédemment.

L'exécution de Hudson nécessite toutefois la modification de « /etc/default/tomcat6 » afin d'ajouter le répertoire d'où Hudson sera exécuté. La ligne à ajouter est la suivante :

« JAVA_OPTS="${JAVA_OPTS} –DHUDSON_HOME=/opt/hudson ». Évidemment, un répertoire devra être créé afin que Hudson puisse utiliser le répertoire :

« sudo mkdir /opt/hudson ; sudo chown -R www-data:tomcat6 /opt/hudson ; sudo chmod -R ug+rwX /opt/hudson ».

Maintenant que la configuration de base a été effectuée, il ne reste qu'à ajouter un lien vers Hudson à l'intérieur des configurations d'Httpd. Pour ce faire, nous modifierons le fichier « /etc/apache2/sites-available/default » en ajoutant « JkMount /hudson* ajp13_worker » juste avant la dernière ligne (avant « </VirtualHost> »).

Il ne reste maintenant qu'à redémarrer Httpd pour que Hudson puisse être utilisé : « sudo service apache2 restart ».

5.2.3 Configuration de Hudson

À ce moment, Hudson devrait être disponible à http://127.0.0.1/hudson. À partir de ce moment, aucune autre configuration en ligne de commande ne devrait être nécessaire.

5.2.3.1 Configuration du système

La première étape pour la configuration de Hudson consiste à configurer les différents modules disponibles sur le système afin que Hudson puisse les utiliser.

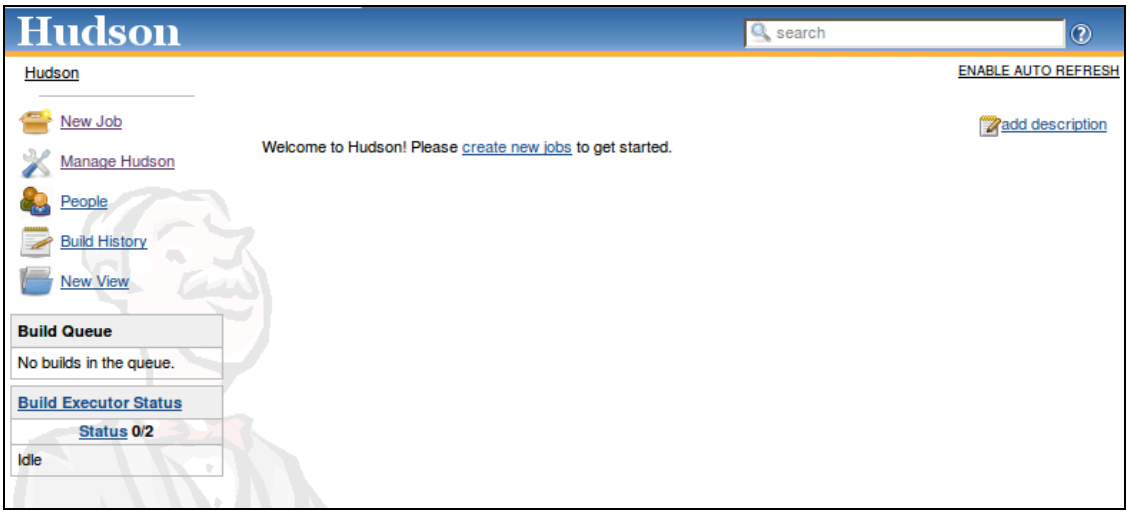

Figure 1: Page d'accueil de Hudson

Afin d'accéder aux configurations système, l'utilisateur doit cliquer sur l'option « Manage Hudson ».

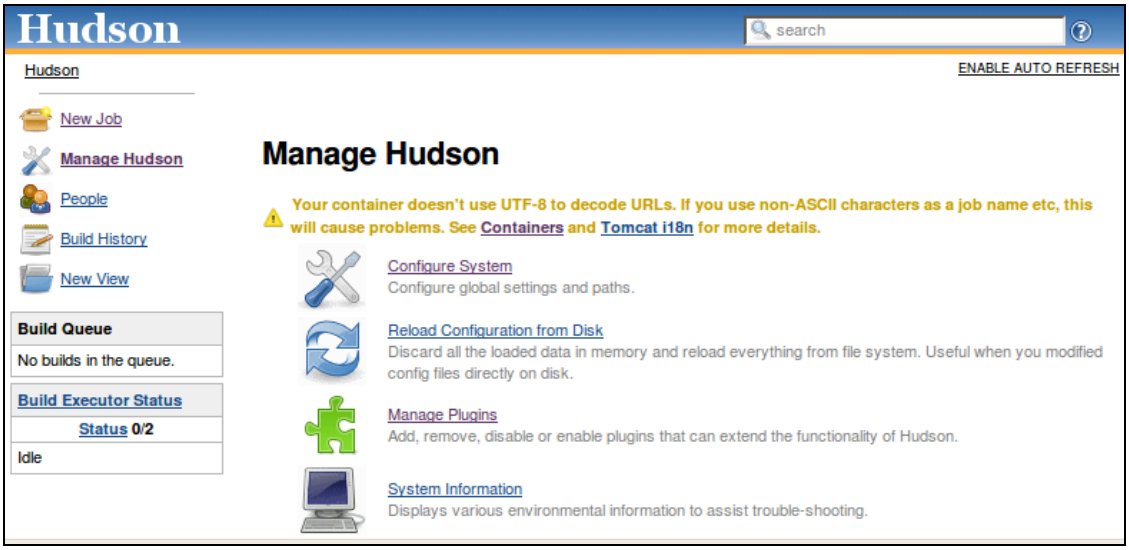

Figure 2: Configuration système de Hudson

À ce stade, l'utilisateur devra choisir « Configure System » afin de finalement accéder aux options système.

Chacun des modules disponibles sur le système et reconnus par Hudson devra être configuré à ce moment. Plus particulièrement, il sera nécessaire de :

- spécifier le répertoire d'installation de Maven 3;
- vérifier la version de Subversion utilisée (devrait être 1.6);
- mettre à jour la configuration des notifications courriel (E-mail Notification).

Le chargé de laboratoire devrait être en mesure de fournir les répertoires des différents modules nécessaires à la configuration.

5.2.3.2 Configuration des extensions

Une fois la configuration initiale du système effectué, l'utilisateur pourra retourner à la page de configuration système de Hudson et sélectionner la configuration des extensions « Manage Plugins ». La page suivante montrera quatre onglets : « Updates », « Available », « Installed », et « Advanced ». La liste des onglets disponibles (« Available ») permet de cocher les extensions qui doivent être installées. Comme mentionnées précédemment, les extensions nécessaires seront différentes toutes en fonction des besoins des laboratoires. En ce sens, le chargé de laboratoire sélectionnera les extensions à utiliser.

5.3 Configuration d'un projet

Afin de configurer un nouveau projet, l'utilisateur doit retourner à la page d'accueil et sélectionner « New Job ».

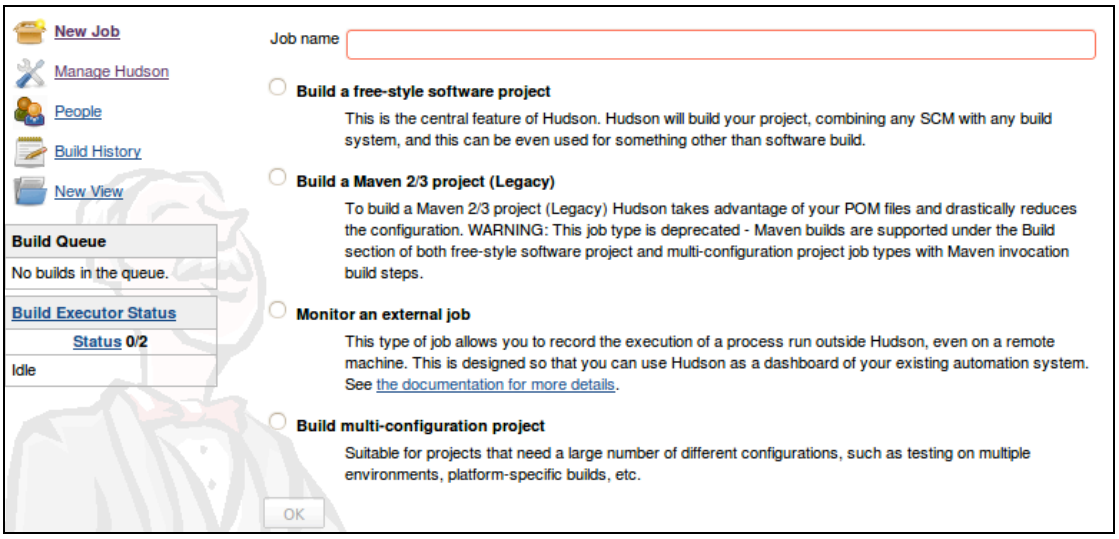

Figure 3: Nouvelle tâche pour Hudson

La première étape dans la création d'une nouvelle tâche nécessite la sélection d'un type de projet. Dans la majorité des cas, la première option sera la bonne (« Build a free-style software project »). La tâche nécessitera aussi un nom : « FinanceJ ».

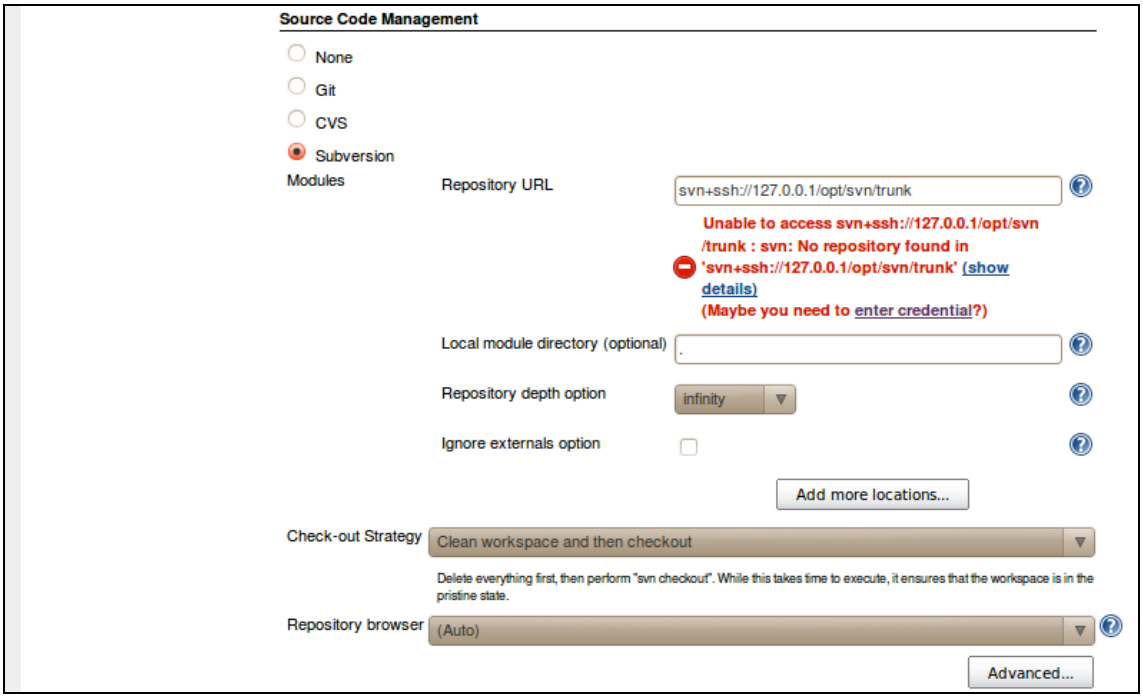

Figure 4: Nouvelle tâche: Serveur de code source

La configuration de Subversion nécessite d'entrer un mot de passe grâce au lien « enter credentials ». Les options lors de la configuration devraient être semblables à celles présentées à la Figure 4.

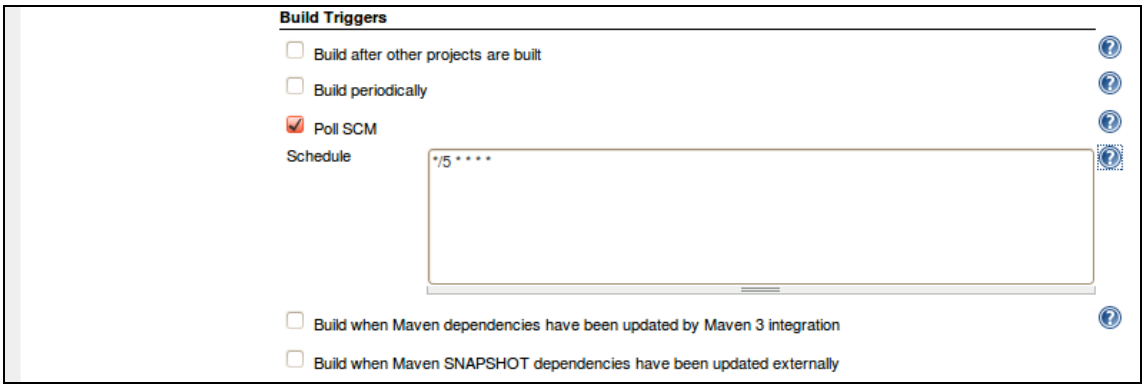

Figure 5: Nouvelle tâche: Déclencheur de construction

Le déclenchement des constructions nécessite une configuration de base permettant une vérification perpétuelle du serveur de code source afin de découvrir les changements récents. Dans la Figure 5, Hudson est configuré pour vérifier le serveur de source toutes les 5 minutes.

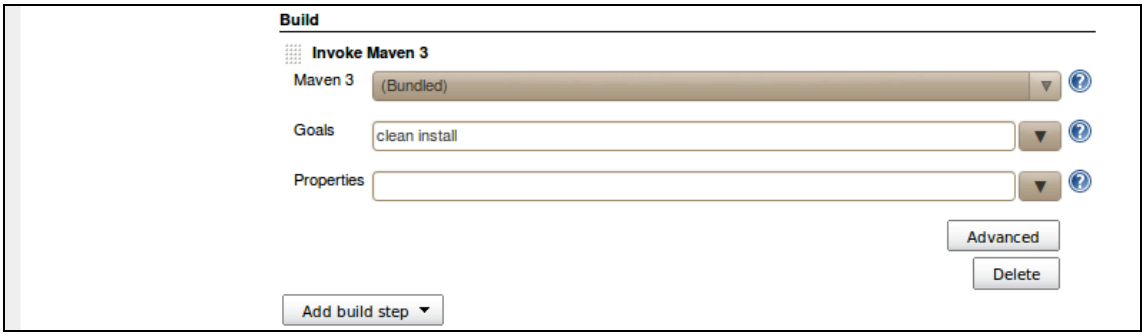

Figure 6: Nouvelle tâche: Construction

Jusqu'à maintenant, la nouvelle tâche n'exécutait pas de construction. Pour remédier à ce problème, il est nécessaire d'ajouter une étape de construction et de spécifier Maven 3 (puisque notre projet utilise Maven 3). Il est possible ici de modifier les buts de la construction par exemple en ajoutant « site » pour construire un site Maven 3.

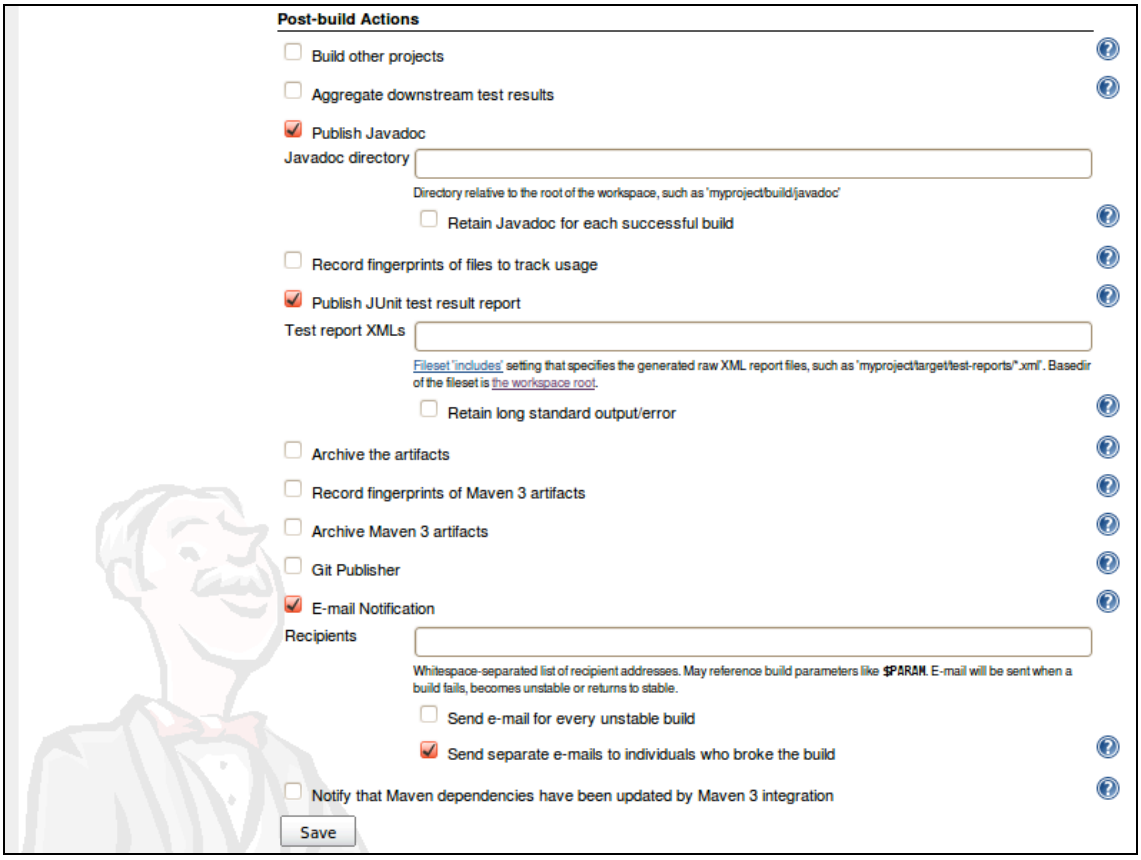

Figure 7: Nouvelle tâche: Action après construction

La dernière étape de configuration d'un nouveau projet est la configuration des actions à effectuer après la construction du logiciel. Tout dépendant des décisions de la part des

étudiants, l'endroit où se trouve la javadoc et les tests unitaires JUnit ne sera pas toujours constant et ainsi les espaces réservés à ces paramètres devront être remplis. Aussi, il sera essentiel de configurer la rétroaction courriel (« E-mail Notification »).

Plusieurs options pourront s'ajouter à chacune de ses parties à cause de l'ajout d'extensions spécifiques. Chaque extension devra être configurée de la même façon.

5.4 Intégration continue d'un projet

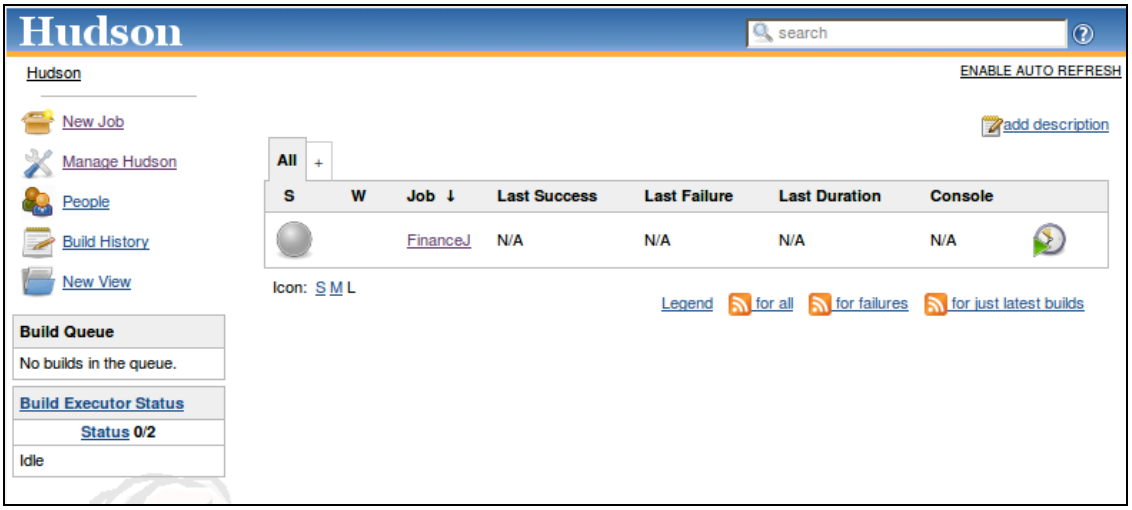

Figure 8: Page d'accueil de Hudson avec le projet FinanceJ

Suite à la création d'un nouveau projet dans Hudson, la page d'accueil ressemblera à la Figure 8. À ce point, Hudson n'a pas encore tenté de construire FinanceJ et c'est pour cette raison que l'orbe est gris.

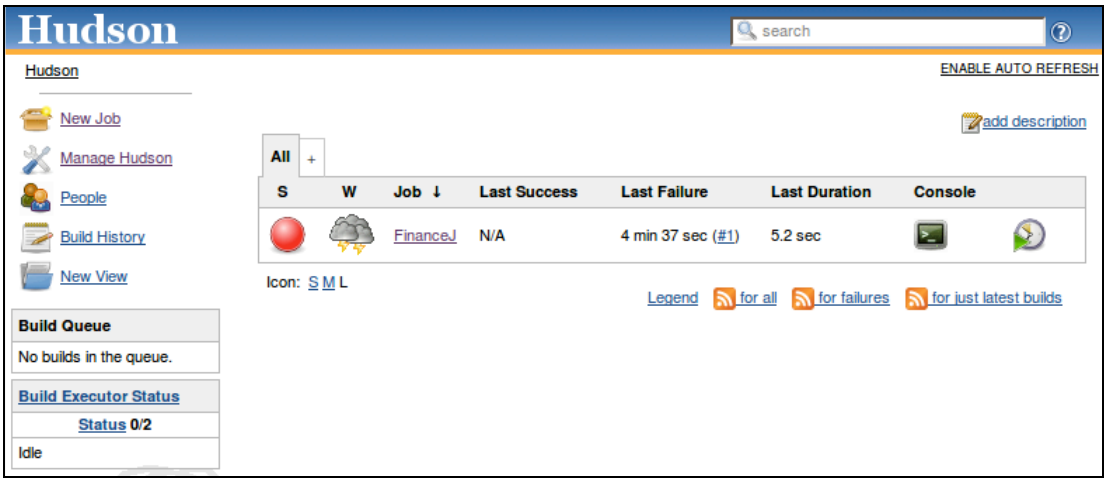

Figure 9: Page d'accueil après la première construction

Après la première construction, si la construction s'est bien passée, l'orbe deviendra vert. Dans le cas contraire, comme le montre la Figure 9, un orbe rouge sera montré. La colonne « W » montre la tendance du projet grâce à un système de climat : plus le temps est orageux, plus les constructions ont tendance à ne pas fonctionner; plus le temps est ensoleillé, plus les constructions ont tendance à fonctionner du premier coup.

Il est possible de vérifier pour chaque construction, quels étaient les résultats obtenus dans la console lors de la construction en cliquant sur l'icône de console.

CONCLUSION

Comparativement à plusieurs autres systèmes et processus, l'intégration continue peut aisément être ajoutée à un système existant sans que celui-ci nécessite des modifications extensives. Les différents outils permettant l'intégration continue ont tous leurs propres problèmes et solutions. Dans notre cas, nous avions à déterminer quel était le meilleur candidat entre « CruiseControl » et « Hudson ». Suite à une comparaison point par point et à un système de pointage, nous avons été en mesure de déterminer lequel des deux logiciels est le mieux adapté au projet des laboratoires du cours de LOG240 : Maintenance et tests. L'implantation d'un processus d'intégration continue était alors possible.

RECOMMANDATIONS

Suite à l'évaluation à l'aide de système de pointage, nous avons déterminé que « Hudson » était le logiciel qui répondait le mieux à nos exigences. Malheureusement, certaines disputes au niveau corporatif ont miné les progrès et le futur de « Hudson ». Plus particulièrement, l'acquisition de Sun Microsystem par Oracle a permis à Oracle d'empêcher l'utilisation du nom « Hudson » par les développeurs de « Hudson ». Après délibération, les développeurs de « Hudson » ont décidé par un vote majoritaire de se dissocier de « Hudson » et de créer un clone qu'ils pourraient modifier à leur guise qui s'appelle « Jenkins ». Ainsi, nous recommandons le passage à « Jenkins » immédiatement plutôt que l'utilisation de « Hudson ». Heureusement, dans le cadre de ce projet, la seule conséquence et le remplacement systématique du nom « Hudson » par le nom « Jenkins ».

BIBLIOGRAPHIE

Daily build and feature development in large distributed projects. **Karlsson, Even-André, Andersson, Lars-Göran et Leion, Per. 2000.** New York : ACM, 2000. 22nd international conference on Software engineering . pp. 649-658. 1-58113-206-9.

Duvall, Paul. 2007. Automation for the people: Continuous Integration anti-patterns. *developerWorks.* [En ligne] 4 Décembre 2007. [Citation : 29 Octobre 2011.] http://www.ibm.com/developerworks/java/library/j-ap11297/index.html.

Duvall, Paul M., Matyas, Steve et Glover, Andrew. 2007. *Continuous Integration: Improving Software Quality and Reducing Risk.* s.l. : Addison-Wesley Professional, 2007. ISBN-13: 978-0-321-33638-5.

Fowler, Martin. 2006. Continuous Integration. *Martin Fowler.* [En ligne] 01 Mai 2006. [Citation : 29 Septembre 2011.] http://martinfowler.com/articles/continuousIntegration.html.

Humble, Jez et Farley, David. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* s.l. : Addison-Wesley Professional, 2010. 978-0-321-67025-0.

John, Smart Ferguson. 2008. *Java Power Tools.* s.l. : O'Reilly Media, Inc., 2008. 978-0-596-52793-8.

McConnell, Steve. 1996. *Rapid Development.* s.l. : Microsoft Press, 1996. 978-1-55615-900-8.

Ostermeier, Daniel et Sankey, Jason. 2009. Continuous Integration Myth: A New Practice. *a little madness.* [En ligne] 9 Janvier 2009. [Citation : 11 Décembre 2011.] http://www.alittlemadness.com/2009/01/09/continuous-integration-myth-a-new-practice/.

ThoughtWorks. 2011. CI Feature Matrix. *ThoughtWorks.* [En ligne] 27 Mai 2011. [Citation : 29 Novembre 2011.] http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix.

Walls, Don. 1999. Integrate Often. *Extreme Programming.* [En ligne] 1999. [Citation : 29 Septembre 2011.] http://www.extremeprogramming.org/rules/integrateoften.html.