UNIVERSITÉ DU QUÉBEC À MONTRÉAL

COMPARAISON DES SOLUTIONS DE TESTS UNITAIRES ET D'INTÉGRATION CONTINUE POUR LE C++

RAPPORT DE PROJET

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN GÉNIE LOGICIEL

PAR

YANN BOURDEAU

FÉVRIER 2014

TABLE DES MATIÈRES

TABLE DES TABLEAUX

TABLE DES FIGURES

1.0 Abrévations

FAQ : Frequently Asked Question. Un document qui contient les questions les plus demandées et leurs réponses sur un sujet donné.

FUSE : Filesystem in Userspace. Permet de monter un système de fichiers sur un système Unix en tant qu'utilisateur normal sans privilèges.

NFS : Network File System. Un système de fichiers accessible à travers un réseau.

SDM : Subscriber Data Management. La gestion des utilisateurs dans un système téléphonique cellulaire 3g ou 4g.

SPR : Subscriber Profile Repository. Partie de la plateforme SDM qui sert à enregistrer les informations des utilisateurs pour l'engin de politiques. C'est-à-dire, enregistrer le profil de l'utilisateur et ses quotas.

SSH : Secure Shell. Permet d'ouvrir une ligne de commande sur un système distant à travers un réseau.

SSHFS : Secure Shell File System. Un système de fichier qui permet de lire à travers un réseau via un lien SSH.

WAR : Web Archive. Un fichier qui contient l'ensemble de fichiers nécessaires pour créer une application sur un serveur applicatif comme Tomcat.

XML : Extensible Markup Language. Un standard codifié comme un document qui contient des données.

XSLT : Extensible Stylesheet Language Transformation. Un langage qui permet de convertir un document XML en autre format ou un autre document XML.

2.0 Remerciements

Je tiens à remercier ma mère, Maryse Laguë, pour son soutien tout au long de mon cheminement universitaire.

Je tiens à remercier mon père, Denis Bourdeau, pour m'avoir donné la chance d'avoir un ordinateur lorsque j'étais un enfant. Il a fait sa maîtrise en informatique de gestion à l'UQAM dans les années 1990. Je suis ses traces en faisant une maîtrise en génie logiciel à l'UQAM.

Je tiens à remercier mon directeur de maîtrise, Normand Séguin, pour son support et ses conseils lors de la réalisation de ce projet.

Je tiens à remercier Tekelec, l'entreprise pour laquelle je travaillais, pour m'avoir permis d'utiliser le code pour bâtir ce projet.

3.0 Résumé

Ce rapport présente la réalisation d'un projet d'évaluation et de sélection d'outils pour l'automatisation des tests unitaires et l'intégration continue du code dans des produits logiciels de télécommunication écrits en C++. Le projet vise à résoudre des problèmes réels de contrôle de qualité en entreprise. Les résultats de cette étude devaient être implantés en production lorsqu'une réorganisation majeure et la vente subséquente de l'entreprise ont empêché leur mise en oeuvre.

Le besoin d'affaire qui justifie ce projet est que beaucoup de problèmes sont découverts dans le produit logiciel après sa mise en production chez le client, alors qu'ils auraient pu être identifiés et résolus bien avant si des processus de contrôle de qualité rigoureux avaient été mis en place.

La solution proposée vise à s'assurer que les tests unitaires soient toujours réalisés pour tous les programmes et librairies du produit logiciel en automatisant le processus de test avec un produit d'intégration continue. On s'assure ainsi que tous les tests seront toujours réalisés, en dépit des ressources limitées et des pressions de la production.

Le projet comporte plusieurs volets :

- Un survol de la littérature a d'abord permis de faire l'inventaire des produits disponibles et identifier les critères de sélection importants pour les auteurs. Le survol de la littérature portait sur les outils de tests unitaires et des outils d'intégration continue pour le C++.
- Deux grilles d'évaluation multicritères pondérés ont ensuite été élaborées, une pour les outils de tests, l'autre pour les outils d'intégration continue. Les critères d'évaluation proviennent d'abord du survol de la littérature et ont été complétés par des critères élaborés à partir des besoins de l'entreprise.
- La pré-sélection de trois outils chacun pour les tests unitaires et l'intégration continue a permis de réduire le nombre d'outils à évaluer à six. Les critères de pré-sélection ont été élaborés à partir des contraintes essentielles à l'entreprise, l'environnement technique ou le statut «open source» , par exemple.
- Une infrastructure d'essai a ensuite été mise en place et configurée. Les produits ont ensuite été installés sur cette infrastructure et des essais ont été conduits pendant plusieurs mois.
- Les essais ont permis d'évaluer les six produits retenus lors de la pré-sélection et de les comparer. Après analyse des résultats, une recommandation finale sur les produits à installer et sur leur configuration a été produite.

Les six outils présélectionnés pour les essais étaitent : Google Test, CPP Unit et Boost Test pour les test unitaires et Cruise Control, Apache Continuum et Hudson, pour les outils d'intégration continue.

L'environnement mis en place pour l'évaluation des solutions comprenait deux serveurs et un poste de travail, Il reproduisait l'environnement technologique de l'entreprise.

Les outils recommandés sont Google Test et Hudson et ce, pour leur facilité d'utilisation, leur impact minimal et leur configuration minimaliste.

4.0 Summary

This project report is the realisation of an evaluation and tools selection project for the automatisation of the unit tests and the continuous integration in a C++ telecommunication software. The project resolve a real problem of an entreprise quality control. The result of this project should have been implemented in the entreprise when an re-organisation did occurs and prevented the application of the project in the entreprise.

The business case justifying this project is that there were many problems were discovered at the customer site when a rigorous quality control process would have avoided this situation.

The proposed solution must force the execution of the units tests for all the binaries of the software product. This is done by a continuous integration process. This should allow the tests to be always done even from the limited resources and the production pressure.

This project contains many components :

- A Literary review allowed to do an inventory of the available products. Also it allowed to indentify selection criteria. The review was on unit tests and continuous integration.
- Two rubrics has been created. One for the unit test and the other one for the continuous integration.
- The pre-selection of the tools has limitied them to three for the unit test and three for the continuous integration.
- An test infrastructure has been created and configured. In this infrastructure the products has been tested for many month.
- The tests resulted on an evaluation of the six selected products. After an analyse, an recommendation has been proposed.

The six tools that were selectioned were: Google Test, CPP Unit et Boost Test for the unit test and Cruise Control, Apache Continuum et Hudson, for continuous integration.

Two servers and an workstation has been used to replicate the entreprise environment.

The selectioned tools are Google Test and Hudson for their user friendly usage, minimal impact and minimal configuration.

5.0 Problématique et Objectifs

5.1 Contexte

Ce projet, présenté comme exigence partielle de la Maîtrise en génie logiciel, a été proposé dans le but de satisfaire les besoins de qualité d'une entreprise qui élabore des logiciels pour les systèmes téléphoniques cellulaires. Ces logiciels sont écrits principalement en C++. L'entreprise compte environ 1500 employés, dont 600 développeurs. Ce projet a été initié dans l'équipe de développement et de support de 2^{e} niveau. L'équipe de 2^{e} niveau est celle qui corrige les bogues lorsque l'équipe de premier niveau a reproduit un problème avec le système. L'équipe de premier niveau interface avec le client et essaie de reproduire le problème.

5.2 Problématique

L'entreprise a remarqué que plusieurs produits livrés aux clients contenaient encore des bogues mineurs. Ces bogues, qui auraient pu être évités, entrainent une baisse de la satisfaction de la clientèle et de sa confiance envers l'entreprise.

Une analyse rapide du problème a démontré que des tests unitaires sont parfois oubliés ou parfois réalisés trop rapidement. Puisqu'il s'agit d'une petite équipe de développement, il n'y a pas assez de ressources assignées aux tests sur les multiples produits de l'entreprise.

C'est une problématique commune à plusieurs PME informatiques. Pourtant, la réalisation de tests unitaires est une pratique reconnue de l'industrie pour réduire le nombre de bogues dans un logiciel comme le SWEBOK le mentionne.

L'entreprise ne possède pas d'outils qui permettraient de mieux supporter la réalisation des tests unitaires. Ces outils permettraient d'automatiser une grande partie des tests et de diminuer la charge de travail requise pour tester les logiciels, tout en s'assurant qu'aucun test ne soit oublié ou exécuté de façon partielle.

Ce projet vise donc à identifier et mettre en place des outils de tests unitaires pour faciliter la tâche des développeurs et augmenter la qualité des produits logiciels.

Pour que les tests unitaires puissent être lancés automatiquement dans l'environnement de l'entreprise, il sera aussi nécessaire d'identifier une solution d'intégration continue. Ce type d'outil surveille un contrôleur de sources pour identifier s'il y a eu modification et lance ensuite automatiquement

la compilation et les tests unitaires des modules. Cela permet donc d'exécuter tous les tests unitaires après chaque modification pour valider que rien n'est brisé.

Une contrainte majeure de la sélection de ce type d'outils est qu'ils doivent être gratuits. Comme c'est le cas souvent dans les PME, il n'y a pas de budget assigné à ce projet par l'entreprise pour sa réalisation ou l'achat d'outils.

5.3 Objectif du projet

L'objectif principal de ce projet est de proposer des solutions automatisées de tests unitaires et d'intégration continue pour un environnement de développement C++, dans le but de réduire le nombre de bogues dans les produits de l'entreprise.

Pour ce faire, l'approche sera de :

- Réaliser un survol de la littérature du domaine pour identifier les produits candidats et construire des grilles d'évaluation des produits.
- Construire un environnement d'essai qui reflète l'environnement réel de l'entreprise.
- Installer, configurer, exécuter et évaluer trois solutions de tests unitaires et trois solutions d'intégration continue à l'aides des grilles d'analyse.
- Analyser les résultats et faire une recommandation.

5.4 Limites du projet

La solution d'intégration continue ne sera utilisée que pour exécuter les tests unitaires. Elle ne sera pas utilisée pour réaliser les tests d'intégration au système, ce qui aurait été une charge de travail trop grande dans le cadre de ce projet de maîtrise.

L'analyse de l'impact organisationnel et le développement de la stratégie d'implantation (organisationnelle) ne font pas partie de ce projet.

De plus, suite à la fermeture du bureau de Montréal et à l'achat subséquent de l'entreprise par Oracle, la solution proposée ne pourra pas être implantée comme prévu. La sélection, la configuration et le test des outils ont quand même pu être réalisés sur du code source réel puisque des composantes ont été transférées dans une nouvelle entreprise en démarrage.

6.0 Survol de la littérature sur les outils

Il existe peu de documentation sur le sujet des solutions de tests unitaires et d'intégration continue en C++. Peu de chercheurs s'y sont consacrés, sauf, peut-être, dans des environnements industriels non documentés. Quelques articles et blogues sont tout de même disponibles sur Internet. Les principaux sont énumérés dans les sections suivantes.

6.1 Outils de tests unitaires en C++

Voici les principaux articles identifiés :

***Exploring the C++ Unit Testing Framework Jungle*[1].** Cet article fait l'inventaire des outils de tests unitaires en C++ et détaille une méthode d'évaluation de ces outils. Cet article contient beaucoup d'informations et est très intéressant à consulter. La dernière mise à jour date de 2010, mais des lecteurs continuent encore à le commenter.

Les autres articles qui traitent de tests unitaires en C++ sont surtout des guides de l'usager pour des produits spécifiques :

- Test-drive development and unit testings with examples in C++ : Cet article décrit comment Ott a réalisé les tests unitaires avec Boost.
- CxxTest User's Guide : Guide d'utilisateur pour CxxTest.
- Primer : Getting started with Google C++ Testing Framework : Introduction à la réalisation de tests avec le Google C++ Testing Framework.
- Unit Testing in C++ and Objective-C just got easier : Cet article discute d'un nouvel outil de test Catch.
- C++ Unit Testing Framework : A Boost Test Tutorial : Un document qui discute comment réaliser des tests avec Boost.

Ce survol de la littérature a permis d'identifier dix outils de tests unitaires en C++. Ce sont : CppUnit, Boost.Test, CppUnitLite, NanoCppUnit, Unit++, CxxTest, Google Test, CGreen, cutee et simplectest. En voici une description sommaire, telle que disponible dans les références :

1. **CPPUnit** est un des premiers outils de tests unitaires qui aie existé en C++. Il est très connu et presque tous les articles sur les tests unitaires en C++ y font référence. Cet outil est toujours supporté et CPPUnit 2 est en développement.

2. **Boost Test** est un outil de tests unitaires assez populaire puisqu'il existe depuis la librairie Boost. Cet outil sert à faire les tests unitaires sur la librairie Boost. Il est toujours supporté.

3. **CppUnitLite** est un outil de tests unitaires écrit par le créateur de CPPUnit. Il a été conçu pour être étendu par l'utilisateur selon ses besoins particuliers de tests et ne contient que le minimum requis pour faire les tests. Il est toujours supporté. La dernière mise à jour date de 2011.

4. **Nano CppUnit** est un outil ultra léger de tests unitaires. Il n'a pas été mis à jour depuis 2005.

5. **Unit++** est un outil de tests unitaires de style JUnit. Il n'est pas très connu et pas beaucoup documenté. Il est toujours supporté. Sa dernière mise à jour date de 2013.

6. **CxxTest** est un outil de tests unitaires de style JUnit. Il est bien documenté et il est à jour. Sa dernière mise à jour date de 2013.

7. **Google Test** est un produit relativement récent fait par Google. Il sert à tester leurs produits internes en C++. Il est maintenu et à jour.

8. **CGreen** est un outil de tests unitaires et de «simulation» basé sur le BDD. Il est encore maintenu; la dernière version date de 2013.

9. **Cutee** est un outil de tests unitaires qui cible le minimalisme pour l'utilisateur. Sa dernière mise à jour date de 2011.

10. **Simplectest** est un outil de tests unitaires qui est abandonné puisqu'il n'a pas été mis à jour depuis 2005.

6.2 Survol de la littérature des solutions d'intégration continue en C++

Pour les solutions d'intégration continue en C++, l'information est encore plus restreinte. Il n'y a qu'un seul article, publié dans un blogue intitulé ***Continuous Integration for C++*** par Rick Wargner et qui date de 2009. L'auteur y présente comment il a fait pour que son projet utilise l'intégration continue en C++.

Plusieurs autres articles comparent les solutions d'intégration continue pour d'autres environnements, la plupart pour le Java.

Le survol de la littérature a permis d'identifier huit outils d'intégration continue en C++. Ce sont : Cruise Control, LuntBuild, Continuum, Hudson, AntHillPro, Bamboo, Pulse et Teamcity.

1. **Cruise Control** est l'un des premiers outils d'intégration continue en C++. Il est toujours supporté et il est très utilisé.

2. **LuntBuild** est un automatisateur de construction pour le monde Java. Il permet de faire l'intégration continue. Il n'a pas été mis à jour depuis 2009.

3. **Continuum** vient de la fondation Apache. Il est encore développé et sa dernière mise à jour date de 2013.

4. **Hudson** est un outil d'intégration continue qui est de plus en plus utilisé à cause de sa simplicité de configuration qui se fait par un GUI. Il est encore développé, sa dernière mise à jour date de 2013.

5. **AntHillPro** est produit par UrbanCode et n'est pas un outil «open source». Il continue à être développé. Il faut l'acheter pour pouvoir l'utiliser.

6. **Bamboo** est un outil développé par Atlassian. Il n'est pas «open source» et il faut l'acheter. Il est mis à jour régulièrement.

7. **Pulse** est un outil développé par Zutubi. Il n'est pas «open source» et il faut l'acheter. Il est mis à jour régulièrement.

8. **TeamCity** est un outil développé par JetBrains. Il n'est pas «open source» et il faut l'acheter. Il est mis à jour régulièrement.

7.0 Méthode utilisée

À cause du grand nombre d'outils disponibles, une présélection des outils à évaluer a dû être réalisée afin de réduire le nombre de candidats à trois chacun pour les outils de test unitaires et les outils d'intégration continue. Une grille d'évaluation avec critères pondérés a ensuite été préparée pour chacun des types d'outils. L'expérimentation a permis d'évaluer les outils en fonction de ces critères, pour, éventuellement en arriver à une recommandation sur le choix d'outils de tests unitaires et d'intégration continue pour le C++.

7.1 Critères de présélection des outils à évaluer

La présélection a été faite à l'aide de critères simples, mais critiques pour l'entreprise. Ce sont :

1. L'outil doit être de type «Open Source» puisque l'entreprise n'a aucun budget à consacrer à l'acquisition de logiciels.
2. L'outil doit pouvoir s'intégrer à l'environnement de développement de l'entreprise. Cela implique le respect des caractéristiques technologiques suivantes :
    a. Support de C++
    b. Fonctionnement sur Unix et Linux
    c. Support du contrôleur de source SUBVERSION
    d. Support de la construction de code à l'aide d'un «makefile»
3. L'outil ne doit pas être un clone d'un outil plus générique.
4. L'outil doit être connu et d'usage répandu dans le milieu du développement en C++.
5. L'outil doit être encore maintenu.

Les mêmes critères de présélection ont été appliqués aux outils de tests unitaires et aux outils d'intégration.

7.2 Grille d'évaluation des outils de tests unitaires

Les outils de tests unitaires et les solutions d'intégration continue seront évalués à l'aide de grilles de critères pondérés. Ces critères de sélection ont été élaborés à partir de la littérature sur le sujet et ont été complétés avec des critères basés sur l'expérience de développement en entreprise.

Le poids relatif a été accordé aux critères selon l'échelle suivante, où un critère «souhaitable» est côté à 1 et un critère «très important» à 10. Toutes les cotes intermédiaires sont permises.

La satisfaction de chaque critère pour chaque produit sera ensuite cotée sur une échelle de 0 (Ne rencontre pas l'exigence) à 1 (tout a fait satisfaisant).

L'échelle de 0.1 et 0.9 se décrit comme suit :

0.1 Rencontre l'exigence, mais le fait de manière très complexes. Cela peux être la complexité à faire la configuration. Il y a 40 et plus d'étapes pour faire la configuration

0.2 Rencontre l'exigence, mais le fait avec un peu moins de complexité que le 0.1. Il y a entre 35 et 39 étapes pour faire la configuration.

0.3 Rencontre l'exigence, mais le fait avec un peu moins de complexité que le 0.2. Il y a entre 30 et 34 étapes pour faire la configuration.

0.4 Rencontre l'exigence, mais le fait avec un peu moins de complexité que le 0.3. Il y a entre 25 et 29 étapes de configuration.

0.5 Rencontre l'exigence, mais le fait avec un peu moins de complexité que le 0.4. Il y a entre 20 et 24 étapes de configuration.

0.6 Rencontre l'exigence, mais le fait avec un peu moins de complexité que le 0.5. Il y a entre 15 et 19 étapes de configuration.

0.7 Rencontre l'exigence, mais le fait avec un peu moins de complexité que le 0.6, Il y a entre 10 et 15 étapes de configuration.

0.8 Rencontre l'exigence, mais le fait avec un peu moins de complexité que le 0.7. Il y a entre 5 et 9 étapes de configuration.

0.9 Rencontre l'exigence, mais le fait avec un peu moins de complexité que le 0.8. Il y a moins de 5 étapes de configuration.

En multipliant la cote par le poids relatif, on obtient ensuite la note assignée à ce critère pour un produit.

Voici les critères retenus pour la grille d'évaluation des outils de tests unitaires :

1. **Effort requis**. Selon Noel[2], il est important que le développeur n'aie pas de travail supplémentaire à faire en plus de réaliser les tests eux-mêmes. Les tests unitaires doivent être exécutés à répétition et tout effort supplémentaire requis pour les préparer ou les configurer rend le programmeur moins efficace et le déconcentre de sa tâche.
   Ce facteur n'est cependant pas si important qu'il pourrait faire échouer la solution. Mais c'est un critère important, de nature complexe. Son poids est donc de 10 points.

2. **Initialisation des objets à tester**. Noel[2]souligne aussi qu'il est important de pouvoir créer des objets initialisés à des valeurs prédéfinies pour pouvoir ensuite réaliser les tests unitaires sur ces objets. Cette fonctionnalité est pratique lorsque qu'on adopte l'approche de ne générer qu'une assertion par test pour faire du débogage et que l'objet doit être initialisé en conséquence.

   Ce critère est très utile, mais non critique. Il reçoit donc une pondération 7.

3. **Support d'exceptions et de plantage.** Noel[3] souligne que l'outil doit être en mesure de reporter le plantage de l'application en cas d'erreur dans le code, comme l'accès à une adresse mémoire invalide, par exemple. L'outil doit être en mesure de gérer l'application pour reporter le plantage jusqu'à la fin du test, si possible. L'outil doit aussi produire les mêmes résultats (avoir le même comportement) pour toutes les exécutions du programme pour le même problème de plantage.

   Ce critère est utile puisque sans cette fonctionnalité, les tests vont s'exécuter, mais sans rapporter correctement tous les problèmes de plantage ou d'exception. Sa pondération est de 7.

4. **Méthodes d'assertion**. C'est-à-dire que l'outil doit offrir plusieurs fonctionnalités d'assertions telles que des assertions de presque égalité pour les assertions de points flottants. L'outil doit aussi supporter la détection d'exceptions lorsqu'une assertion est effectuée. Ceci est encore un critère provenant de Noel[4].

   C'est un critère utile qui reçoit une pondération de 7.

5. **Simulateur d'objets**. C'est la capacité de pouvoir créer des objets dans l'environnement de tests qui retournent toujours les mêmes messages à la classe testée, comme un simulateur de message sur un «socket». Ceci permet de créer un environnement de tests stable et complet, produisant toujours la même série de données pour tester. Ce critère est basé sur mon expérience. C'est une fonctionnalité pratique qui permet d'améliorer le diagnostic d'erreurs et de gagner du temps.

   Ce critère est important et reçoit une pondération de 10.

6. **Qualité du rapport de tests**. Ce critère évalue le niveau de clarté et de détail du rapport de tests. Le rapport ne doit pas nécessiter l'utilisation d'un outil tiers pour être visualisé.

Ce critère est basé sur mon expérience. C'est un critère utile qui reçoit une pondération de 5.

7. **Documentation et support gratuits**. Facilité à trouver de la documentation et de l'aide gratuite pour la configuration dans la documentation du produit, les FAQ, les forums de discussion et les listes de distribution de courriels.
   Ce critère est basé sur mon expérience et sa pondération est de 5 points.

8. **Support payant.** Disponibilité du support. Cela peut être pratique d'obtenir du support, payant ou non, si l'on rencontre un problème une fois en production. Ce critère est basé sur mon expérience.
   Ce critère est utile et sa pondération est de 6 points.

Tous ces critères sont réunis dans la grille d'évaluation suivante :

Tableau 1 Grille d'évaluation des outils de test unitaire

Critère	Description	Poids	Cote
1 Effort requis	*L'effort requis pour ajouter un test au programme. Moins il y a de code requis, plus la cote est élevée.*	10	
2 Initialisation d'objets	*Création d'objets initialisés à des valeurs prédéfinies.*	7	
3 Support d'exceptions et de plantage	*L'outil évite le plantage de l'application lorsqu'il y a une erreur dans le code. L'outil doit aussi avoir le même comportement sur plusieurs exécutions rapportant le même problème de plantage.*	7	
4 Méthodes d'assertion	*L'outil devrait supporter plusieurs méthodes de définition d'assertions sur plusieurs types.*	7	
5 Simulateur d'objets	*Capacité de créer des objets comme un simulateur de message sur un «socket».*	10	
6 Rapport de tests	*Rapport de tests clair et bien détaillé. Ne doit pas nécessiter un outil tiers pour le visualiser.*	5	
7 Documentation et support gratuits	*Facilité à trouver de la documentation et de l'aide gratuite pour la configuration (documentation, FAQ, forums, etc.).*	5	
8 Support payant	*L'entreprise ou une autre compagnie offre du support payant en cas de problème.*	6	

Critère	Description	Poids	Cote
	Total	57	

7.3 Grille d'évaluation des outils d'intégration continue

Voici les critères retenus pour la grille d'évaluation des outils d'intégration continue :

1. **Performance de l'outil**. Ce critère vient de Fowler[6]. Le processus de construction devrait s'exécuter rapidement, dans un délai raisonnable.
   Ce critère est souhaitable et sa pondération est de 2 points.

2. **Facilité de configuration**. Ce critère vient de Neokrates[7]. La configuration d'un système de construction à base de «makefile» doit être facile à réaliser dans l'outil.
   Ce critère est très important et sa pondération est de 10.

3. **Historique des constructions**. L'outil doit garder un historique des constructions pour consultation. Ce critère est basé sur mon expérience.
   Ce critère est utile et de basse priorité. Sa pondération est de 3.

4. **Facilité d'utilisation**. Indique si l'outil est facile à utiliser pour les développeurs qui vont interagir avec. Ce critère vient de Neokrates[8].
   Ce critère est important et sa pondération est de 10.

5. **Facilité d'administration de l'outil**. L'outil doit être simple à administrer par l'administrateur du système. Ce critère est basé sur mon expérience.
   Ce critère est souhaitable parce qu'il ne s'applique qu'à l'administrateur du système, qui est plus disponible. Sa pondération est donc de 2 points.

6. **Automatisation**. L'outil doit pouvoir fonctionner de façon normale et autonome au sein de l'entreprise sans nécessiter d'intervention de la part d'un utilisateur. Ce critère est basé sur mon expérience.
   Ce critère est utile et sa pondération est de 5.

7. **Documentation et support gratuits**. Facilité à trouver de la documentation et de l'aide gratuite pour la configuration dans la documentation du produit, les FAQ, les forums de discussion et les listes de distribution de courriels.
   Ce critère est basé sur mon expérience et sa pondération est de 5 points.

9. **Support payant.** Disponibilité du support. Cela peut être pratique d'obtenir du support, payant ou non, si l'on rencontre un problème une fois en production. Ce critère est basé sur mon expérience.
   Ce critère est utile et sa pondération est de 6 points .

Tous ces critères sont réunis dans la grille d'évaluation suivante :

Tableau 2 Grille d'évaluation des outils d'intégration continue

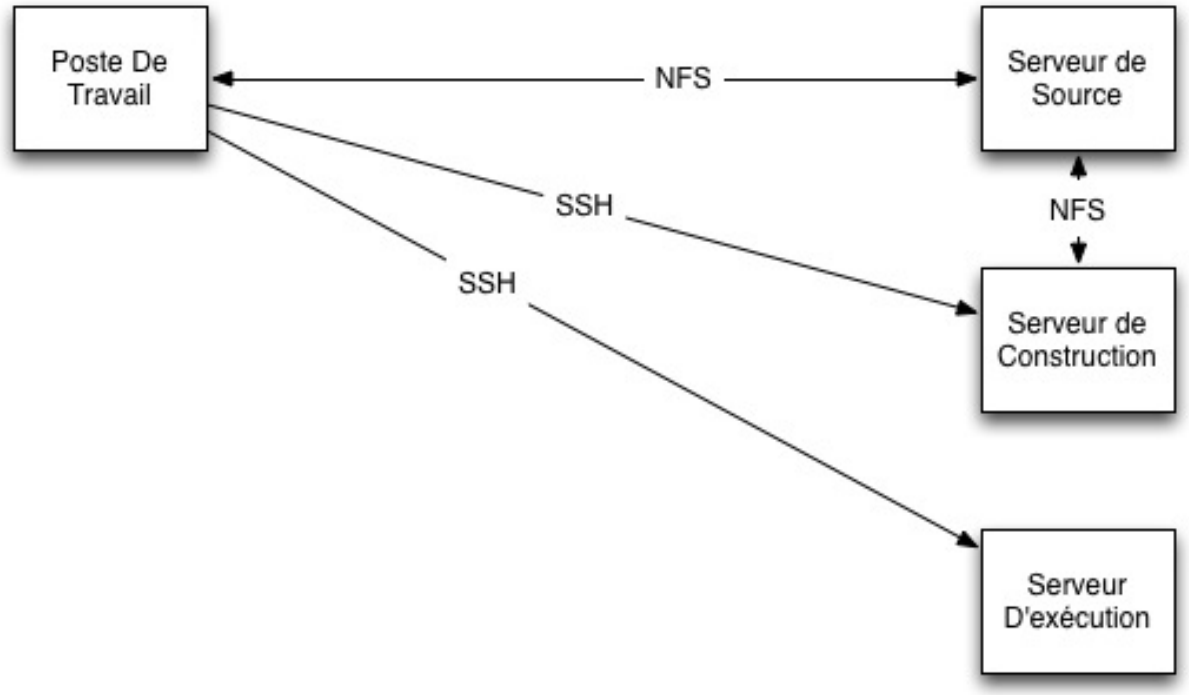
Critère	Description	Poids	Cote
1. Performance	*Le processus de construction doit s'exécuter rapidement, dans un délai raisonnable.*	2	
2. Facilité de configuration	*La configuration d'un système de construction à base de «makefile» doit être facile à réaliser.*	10	
3. Historique	*L'outil doit conserver un historique des constructions réalisées.*	3	
4. Facilité d'utilisation	*Indique si l'outil est facile à utiliser pour les développeurs qui vont interagir avec.*	10	
5. Administration de l'outil	*L'outil doit être simple à administrer par l'administrateur du système.*	2	
6. Automatisation	*L'outil doit pouvoir fonctionner de façon normale au sein de l'entreprise, sans nécessiter d'intervention de la part d'un utilisateur.*	5	
7. Documentation et support gratuits	*Facilité à trouver de la documentation et de l'aide gratuite pour la configuration (documentation, FAQ, forums, etc.).*	5	
8. Support payant	*L'entreprise ou une autre compagnie offre du support payant en cas de problème.*	5	
	Total	43	

8.0 Environnement mis en place pour l'évaluation des produits

8.1 Configuration matérielle et réseau

La configuration d'un environnement de test ou de développement est la suivante : Le poste de travail fonctionne sous Linux et les sources sont contenus sur un serveur NFS que chaque développeur accède. Ensuite, le développeur se connecte sur un serveur de construction pour bâtir l'exécutable. L'exécutable est fait dans un paquet qui est possible d'installer sur un serveur d'exécution.

Figure 1 Configuration compilation



Pour évaluer les outils, j'ai fait l'installation sur ma machine Mac OS X puisqu'il fallait simuler le développement à deux machines comme dans l'entreprise.

La plateforme de la machine de construction et d'exécution Linux est souvent mise à jour. Pour éviter de toujours avoir à réinstaller les outils d'intégration continue au rythme des mises à jour, ces outils ont été installés sur un autre serveur dédié seulement à cela. Pour que cette approche fonctionne, la construction et les tests unitaires doivent être réalisés à distance sur une autre machine Linux.

Pour résoudre le problème de compilation à distance, j'ai fait deux choses. La première étape est de partager le répertoire de compilation de la machine de construction vers la machine qui contient le serveur d'intégration continue. Pour ce faire, j'ai utilisé l'outil SSHFS qui permet de monter un répertoire sur une machine distante. SSHFS utilise FUSE qui permet de monter un répertoire distant via le protocole SSH. FUSE est fait pour monter des « file system » pour un utilisateur Unix et non toute la machine comme un «mount» NFS. SSHFS est moins performant que NFS, mais il est moins difficile à configurer. Je voulais éviter de mettre de la complexité, dans ma configuration à la maison, de mettre un service NFS entre mes deux machines. Dans l'entreprise Tekelec, il y a un système NFS par «auto-mount» qui permet d'avoir accès à un répertoire qui contient les sources. J'ai trouvé cela trop complexe et hors sujet de mon projet pour reproduire cela. Donc, j'ai monté sur la machine du serveur d'intégration continue le répertoire de la machine construction. Ce qui permet à l'outil d'intégration continue de sortir les sources d'un contrôleur de sources et les mettre dans le répertoire partagé. La deuxième étape a été de faire un script Shell qui permet d'exécuter les commandes de compilation à distance via SSH. Le script relaie les codes d'erreur de compilation.

8.2 Configuration logicielle pour les tests unitaires

8.2.1 Choix du module à tester

Un seul module du produit d'entreprise sera utilisé pour évaluer les solutions de tests unitaires. Ce module s'appelle «Policy» et il gère les données XML décrivant un utilisateur dans le produit. C'est un petit module qui ne contient que sept sources. Il est représentatif de l'ensemble du code à tester dans le projet. «Policy» est un excellent candidat pour évaluer les solutions de tests unitaires.

8.2.2 Utilisation de «stubs»

Pour exécuter les sources du module «Policy» sans le reste du produit, trente-cinq «stubs» ont été créés pour simuler la fonctionnalité des modules qui interfacent avec «Policy». Les «stubs» retournent toujours une réponse valide, toujours avec les mêmes résultats, lorsqu'il est nécessaire de retourner un résultat. Cette approche permet de tester le module Policy sans avoir accès à tous les autres modules du produit.

Ceci est d'autant plus nécessaire que l'initialisation de tous les modules du produit ne peut être réalisée que sur le serveur d'exécution et non sur le serveur de construction. Cette situation rend impossible la réalisation des tests unitaires lors de la compilation, à moins de créer des «stubs».

8.2.3 Nom des «stubs» et des tests unitaires

L'extension des sources C++ dans la hiérarchie de modules du produit est «.cpp». Le système de «makefile» est fait pour compiler tous les sources, dont l'extension est «.cpp» et de les lier dans le module créé dans le répertoire. Puisque les modules créés par le système de modules ne doivent pas contenir des «stubs» et des tests unitaires, ces derniers ont donc reçu l'extension «.cxx».

8.2.4 Les rapports d'exécution

Il n'y a que Google Test qui formate les rapports d'exécution au format JUnit utilisé par les outils d'intégration continue. Pour les deux autres outils, un outil XSLT a dû être utilisé pour transformer leurs rapports du format XML vers le format JUnit.

Un XSLT permet de transformer un document XML dans un autre format ou en XML selon un langage codifié. L'outil XSLT que j'ai utilisé est Saxon. Pour les tests unitaires Boost Test et CPP Unit j'ai appelé Saxon dans le code du test unitaire dans les «hooks» pour formatage de rapport des tests unitaires.

Google test, générait un rapport XML compris par les outils d'intégration continue. Aucune conversion n'a été requise. C'est le même format que JUnit.

8.2.5 Le «makefile»

J'ai modifié le système de «makefile» pour ajouter l'exécution des tests unitaires à la cible «exec» du «makefile» du module de Policy. Donc, quand la commande «make exec» est lancée à la racine de la hiérarchie du projet, le module Policy va compiler les «stubs», les tests unitaires des trois outils et lancer l'exécution des tests unitaires. Pour les besoins de l'évaluation, les tests unitaires sont toujours lancés pour les trois outils. Cela permet d'avoir un système de «makefile» plus simple que d'avoir faire exécuter chaque outil de test unitaire individuellement.

8.2.6 Emplacement des rapports de tests unitaires

Les rapports des tests unitaires des trois outils sont tous placés dans un répertoire au-dessus du répertoire racine de la hiérarchie de modules. Cela permet d'éviter de polluer les répertoires avec les rapports. Cela simplifie aussi le travail pour les outils d'intégration continue qui n'ont à consulter qu'un seul répertoire pour avoir tous les résultats des tests unitaires.

Cette approche a comme limite d'imposer un nom unique à chaque rapport, ce qui peut devenir compliqué quand un nom de source est utilisé par plusieurs modules différents. Pour résoudre ce

problème, il suffit d'ajouter le nom du module au nom de source pour définir le nom du rapport de tests unitaires.

8.3 Configuration logicielle pour les solutions d'intégration continue

Les rapports des outils de tests unitaires sont créés dans le répertoire partagé dans un même répertoire. Ce qui permet aux outils d'intégration continue d'aller les chercher et les utiliser dans leurs rapports.

Pour limiter le nombre de machines virtuelles, j'ai décidé d'exécuter les outils d'intégration continue sur mon ordinateur Mac OS X. Puisque les outils sont en Java, ils sont multiplateformes. Il n'y a pas de problème à les rouler sous Linux ou Mac OS X.

Puisque je parle de performance dans ma grille d'évaluation, je vais décrire les machines que j'utilise pour faire exécuter les outils d'intégration continue et la compilation. Pour les outils d'intégration continue une machine Mac OS X Montain Lion avec un processeur Intel i5 2.8Ghz et 16 gigaoctets de mémoire. Pour la compilation, une machine virtuelle Linux avec seulement deux coeurs et 4 gigaoctets de mémoire qui roule sur la machine Mac OS X.

9.0 Résultats de l'évaluation – Outils de Tests Unitaires

9.1 Présélection

9.1.1 Outils de tests unitaires

Dix outils de tests unitaires en C++ ont été identifiés à l'aide du survol de la littérature. Les produits dérivés de CPPUnit ont d'abord été éliminés. CPPUnit étant représentatif de la classe d'outils dérivés de JUnit, l'outil de tests unitaires du monde Java, les autres outils de cette classe ont aussi été éliminés. Ensuite, ce sont les outils qui n'ont pas été mis à jour depuis plus de deux ans qui ont été éliminés. À ce stade, il ne reste que cinq outils en liste pour l'évaluation. Ce sont : CppUnit, Boost.Test, Google Test, CGreen et Cutee.

Parmi les cinq outils restants, les trois suivants ont été présélectionnés pour l'évaluation des tests unitaires :

1. **CppUnit** a été choisi parce que c'est un des outils de tests unitaires C++ le plus connu et qu'il est un port de JUnit du monde Java. CPPUnit est bien documenté et plusieurs exemples d'utilisation sont disponibles. Les autres produits dérivés de CppUnit ou de JUnit ont donc été éliminés.
2. **Boost Test** a été choisi parce qu'il est bien établi et qu'il est maintenu à jour par les développeurs de la librairie Boost. Il est bien documenté et des exemples d'utilisation sont disponibles.
3. **Google Test** a été choisi parce que c'est un nouveau produit dans le monde des tests unitaires C++. Ce n'est pas une copie de JUnit et il est bien documenté. Il a été développé et est maintenu par Google.

Les deux suivants ont été rejetés :

1. **CGreen** : Il a été rejeté puisqu'il est moins populaire que les autres.
2. **Cutee** : Il a été rejeté puisqu'il est moins populaire que les autres.

9.2 Évaluation de Google Test

La configuration des options dans Google Test se fait par des variables d'environnement, ce qui simplifie sa configuration puisqu'on n'a pas à modifier le code source pour changer le comportement de l'outil. Les rapports des tests unitaires sont produits dans le format requis par les outils d'intégration continue.

Pour l'utiliser, il suffit d'inclure un fichier d'en-tête (.h) dans le source du test unitaire. Ce fichier définit les macros qui doivent être utilisées pour les tests. Il n'est pas nécessaire de définir une procédure main. Il suffit de déclarer une méthode de test avec la macro TEST et le test va être exécuté lors du lancement du binaire du test unitaire. Cela laisse toute la place aux tests unitaires dans le code source.

Voici un exemple de définition d'un test unitaire :

```
TEST(HostnameTestCase,TestParse)
{
      HostName hn;
      std::string nai;
      nai="bob";
      EXPECT_EQ(hn.isValidNAI(nai),true);
…
}
```

La macro TEST prend deux arguments en paramètres, soit le nom du test unitaire et le nom de la suite de tests. Il suffit de lier l'exécutable avec la librairie de Google Test et les tests unitaires sont prêts à être exécutés.

La variable d'environnement GTEST_OUTPUT a été initialisée avec le nom du répertoire devant contenir les rapports des tests unitaires.

Le tableau suivant présente les résultats de l'évaluation de Google Test en fonction des critères retenus plus haut.

Tableau 3 Évaluation de Google Test

Critère	Description	Poids	Cote
1 Effort requis	*L'effort requis pour ajouter un test au programme. Moins il y a de code requis, plus la cote est élevée.* La solution Google Test ne nécessite que de définir des fonctions de tests unitaires avec la macro TEST. Cela est très simple à faire.	10	1.0
2 Initialisation d'objets	*Création d'objets initialisés à des valeurs prédéfinies.* Oui, il est possible de créer des objets initialisés à des	7	1.0

	Critère	Description	Poids	Cote
		valeurs prédéfinies pour tester. J'ai dû le faire pour réaliser les tests unitaires.		
3	Support d'exceptions et de plantage	*L'outil évite le plantage de l'application lorsqu'il y a une erreur dans le code. L'outil doit aussi avoir le même comportement sur plusieurs exécutions rapportant le même problème de plantage.* Oui, cela est géré par Google Test.	7	1.0
4	Méthodes d'assertion	*L'outil devrait supporter plusieurs méthodes de définition d'assertion sur plusieurs types.* L'outil a permis de créer toutes les assertions requises par les essais.	7	1.0
5	Simulateur d'objets	*Capacité de créer des objets comme un simulateur de message sur un «socket».* Il n'y a pas d'outils de simulation d'objets dans Google Test. Google fournit cependant un «framework» qui permet de faire de la simulation d'objets en C++. Google Test ne satisfait pas ce critère, puisque le simulateur n'est pas disponible dans l'outil.	10	0
6	Rapport de test	*Rapport de tests clair et bien détaillé. Ne doit pas nécessiter un outil tiers pour le visualiser.* Le rapport de test est fait en XML et en format JUnit. Le format JUnit est compris directement par les outils d'intégration continue, sans transformation.	5	1.0
7	Documentation et support gratuits	*Facilité à trouver de la documentation et de l'aide gratuite pour la configuration (documentation, FAQ, forums, etc.).*	5	1.0

Critère	Description	Poids	Cote
	Il y a des guides d'utilisation pour l'utilisation des tests unitaires.		
8 Support payant	*L'entreprise ou une autre compagnie offre du support payant en cas de problème.* Aucun support n'est offert sur le produit, à part via le forum de développement.	6	0
	Total	57	41

9.3 Évaluation de Boost Test

Pour faire des tests avec Boost Test, il faut définir le test dans une macro. Il faut aussi définir une «fixture» pour la sortie de rapport pour pouvoir faire exécuter le XSLT avec Saxon pour transformer les rapports XML au format requis par les outils d'intégration continue.

Les options d'exécution des tests sont passées à la ligne de commande. Dans les tests, on a spécifié de générer un rapport avec ses options à la ligne de commande. Voici une exemple de ligne de commande :

./TestDataBoost --output_format=XML --log_level=test_suite --report_level=no

Voici un exemple de test unitaire :

```
BOOST_AUTO_TEST_SUITE( test_hostname_suite )
BOOST_AUTO_TEST_CASE(test_hostname_parse)
{
      HostName hn;
      std::string nai;
      nai="bob";
      BOOST_CHECK_EQUAL(hn.isValidNAI(nai),true);
…
}
BOOST_AUTO_TEST_SUITE_END()
```

La fonction « init » permet d'initialiser des objets avant que les tests unitaires ne soient appelés.

Le tableau suivant présente les résultats de l'évaluation de Boost Test en fonction des critères retenus plus haut.

Tableau 4 Résultats de l'évaluation de Boost Test

Critère	Description	Poids	Cote
1 Effort requis	*L'effort requis pour ajouter un test au programme. Moins il y a de code requis, plus la cote est élevée.* La solution nécessite de définir des fonctions de test unitaire avec la macro BOOST_AUTO_TEST_CASE. Par contre, il faut définir une «fixture» pour avoir des rapports XML.	10	1.0
2 Initialisation d'objets	*Création d'objets initialisés à des valeurs prédéfinies.* Oui, il est possible de créer des objets initialisés à des valeurs prédéfinies pour tester. J'ai dû le faire pour réaliser les tests unitaires	7	1.0
3 Support d'exceptions et de plantage	*L'outil évite le plantage de l'application lorsqu'il y a une erreur dans le code. L'outil doit aussi avoir le même comportement sur plusieurs exécutions rapportant le même problème de plantage.* Oui, cela est géré par Boost Test.	7	1.0
4 Méthodes d'assertion	*L'outil devrait supporter plusieurs méthodes de définition d'assertions sur plusieurs types.* Il n'y a pas autant d'assertions disponibles qu'avec les autres outils. J'ai utilisé BOOST_CHECK_EQUAL et l'assertion générique BOOST_CHECK.	7	0.8

Critère	Description	Poids	Cote
	BOOST_CHECK prend une expression pour évaluer. Puisqu'il faut utiliser la fonction générique dans plusieurs cas, j'enlève deux points à cette catégorie.		
5 Simulateur d'objets	*Capacité de créer des objets comme un simulateur de message sur un «socket».* Il n'y a pas d'outils de simulation d'objets dans Boost Test.	10	0
6 Rapport de tests	*Rapport de tests clair et bien détaillé. Ne doit pas nécessiter un outil tiers pour le visualiser.* Il est possible de générer des rapports de tests en XML. Par contre, les rapports ne sont pas en format JUnit et ils ne sont pas compris sans transformation XSLT par les outils de test d'intégration continue.	5	0.4
7 Documentation et support gratuits	*Facilité à trouver de la documentation et de l'aide gratuite pour la configuration (documentation, FAQ, forums, etc.).* Il est très facile d'obtenir de l'aide dans la liste de courriel. Aussi le guide est bien fait.	5	1.0
8 Support payant	*L'entreprise ou une autre compagnie offre du support payant en cas de problème.* Il n'y a plus de support commercial pour Boost.	6	0
	Total	57	36.6

9.4 Évaluation de CPP Unit

Les tests faits avec CPP Unit exigent plus de code. Il faut définir une classe dans un .h et le test dans un fichier .cpp. Cela est plus compliqué que les deux autres solutions qui utilisent seulement des macros pour générer les tests.

Pour le rapport XML des tests unitaires, il faut définir un XmlOutputter qui sert à générer le rapport. Ceci est fait dans la fonction «main» du test. J'ai créé une fonction «main» pour chaque source de tests unitaires. Cela me semblait plus modulaire que de tout mettre dans une fonction «main» commune à tous les sources de tests unitaires.

Voici un exemple de test unitaire :

```
{
        std::string entity;
        std::string content("<allo/>");
        std::string result("&lt;allo/&gt;");
        blueslice::policy::xml::xmlEntityEscape(entity,content);
        CPPUNIT_ASSERT_EQUAL(result,entity);
    }
```

Le tableau suivant présente les résultats de l'évaluation de CPP Unit en fonction des critères retenus plus haut.

Tableau 5 Résultats de l'évaluation de CPP Unit

	Critère	Description	Poids	Cote
1	Effort requis	*L'effort requis pour ajouter un test au programme. Moins il y a de code requis, plus la cote est élevée.* La solution nécessite de créer un fichier «header» pour définir une classe et les tests unitaires dans un autre fichier .cpp. Il faut aussi définir une fonction «main» qui sert à enregistrer les tests unitaires à l'exécuteur de tests.	10	0.3
2	Initialisation d'objets	*Création d'objets initialisés à des valeurs prédéfinies.* Oui, il est possible de créer des objets initialisés à des	7	1.0

	Critère	Description	Poids	Cote
		valeurs prédéfinies pour tester.		
3	Support d'exception et de plantage	*L'outil évite le plantage de l'application lorsqu'il y a une erreur dans le code. L'outil doit aussi avoir le même comportement sur plusieurs exécutions rapportant le même problème de plantage.* Oui, cela est géré par CPP Unit.	7	1.0
4	Méthodes d'assertion	*L'outil devrait supporter plusieurs méthodes de définition d'assertion sur plusieurs types.* L'outil supporte une bonne suite d'assertions. Elles prennent une évaluation en paramètre plutôt que ce soit l'assertion qui définisse le type d'évaluation.	7	1.0
5	Simulateur d'objets	*Capacité de créer des objets comme un simulateur de message sur un «socket».* Il n'y a pas d'outils de simulation d'objets dans CPPUnit.	10	0
6	Rapport de tests	*Rapport de tests clair et bien détaillé. Ne doit pas nécessiter un outil tiers pour le visualiser.* Il est possible de générer des rapports de tests en XML. Par contre, les rapports ne sont pas en format JUnit et ils ne sont pas compris sans transformation XSLT par les outils de test d'intégration continue.	5	0.4
7	Documentation et support gratuits	*Facilité à trouver de la documentation et de l'aide gratuite pour la configuration (documentation, FAQ, forums, etc.).* Il y a des guides d'utilisation pour l'utilisation des tests	5	1.0

Critère	Description	Poids	Cote
	unitaires.		
8 Support payant	*L'entreprise ou une autre compagnie offre du support payant en cas de problème.* Il n'y a plus de support commercial pour CPP Unit.	6	0
	Total	57	31

9.5 Comparatif des résultats pour l'évaluation des outils de test unitaire

Tableau **6** Tableau comparatif des résultats d'évaluation des tests unitaires

Critère	Google Test	Boost Test	CPP Unit
1. Effort requis	10	10	3
2. Initialisation d'objets	7	7	7
3. Support d'exceptions et de plantage.	7	7	7
4. Méthodes d'assertion.	7	5,6	7
5. Simulateur d'objets	0	0	0
6. Rapport de tests	5	2	2
7. Documentation et support gratuits	5	5	5
8. Support payant	0	0	0
Total	41	36,6	31

10.0 Résultats de l'évaluation – Outils d'intégration continue

10.1 Présélection

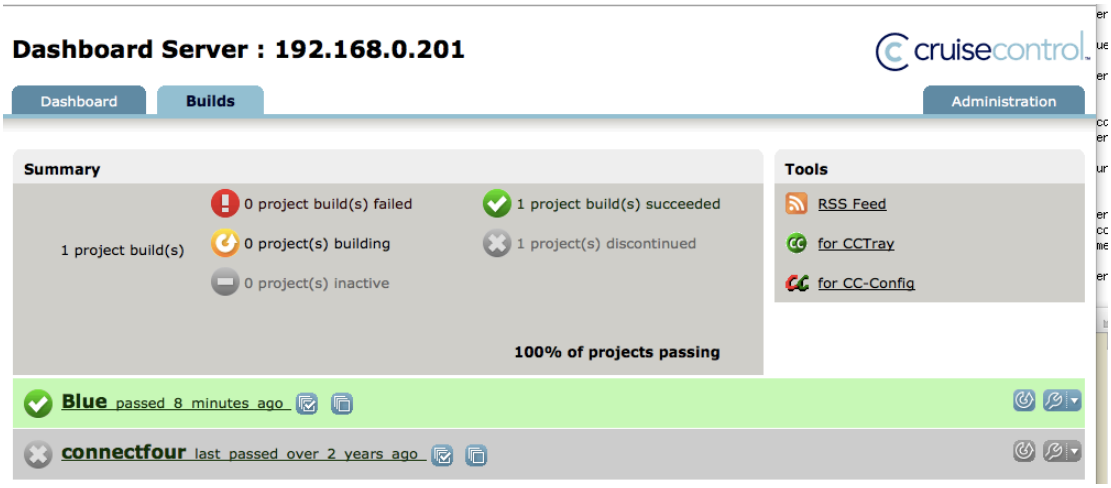
Huit outils d'intégration continue en C++ ont été identifiés dans le survol de la littérature. Après avoir éliminé les outils qui ne sont pas «Open Source» et ceux qui ne sont plus maintenus, il ne reste que trois candidats pour l'évaluation. Ce sont : **CruiseControl**, **Continuum** et **Hudson**.

10.2 Évaluation de CruiseControl

CruiseControl est un outil Java qui fonctionne sans serveur applicatif comme Tomcat. C'est une application Java autonome. La configuration de l'outil se fait dans un fichier XML. Il n'y a pas d'interface utilisateur pour faire la configuration. À chaque changement de configuration, il faut relancer CruiseControl. Il y a une interface utilisateur pour lancer des constructions et regarder les rapports des constructions. Il est aussi possible de suivre la sortie de la ligne de commande de la construction en cours.

Voici la page principale de Cruise Control.

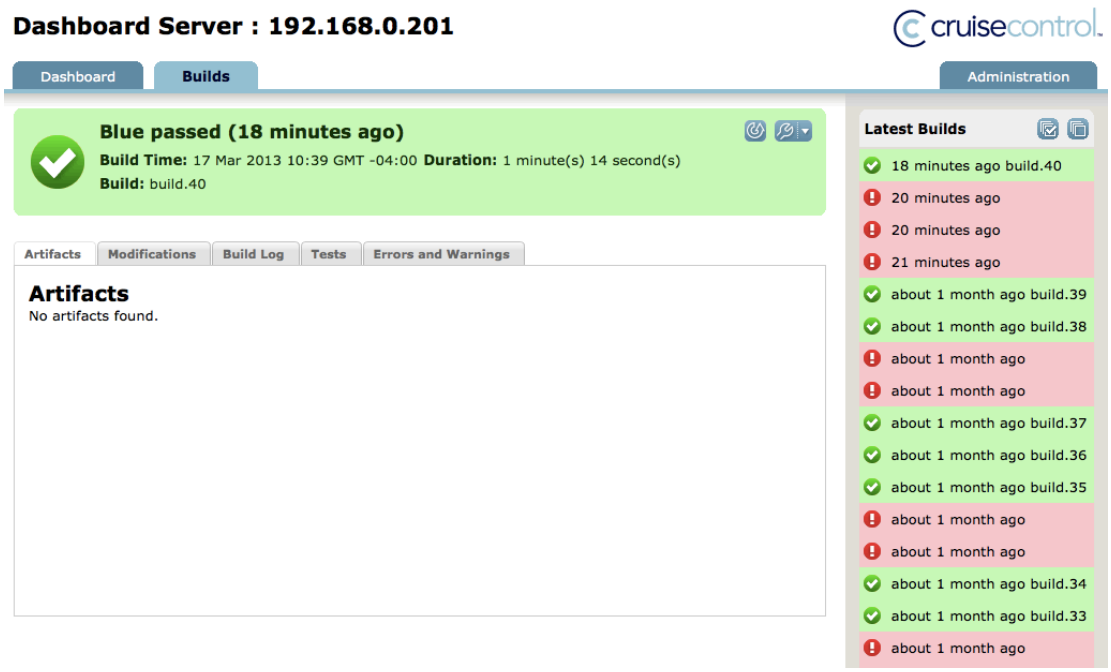
Figure 2 Panneau principal Cruise Control



À partir de cette page, il est possible de lancer la construction et de consulter l'historique de la construction. Pour consulter, l'historique de la construction il suffit de cliquer sur le nom du projet. Le nom que j'ai utilisé pour le projet est Blue.

Voici l'écran de l'historique de construction.

Figure 3 Historique de construction



Il est possible dans l'historique d'avoir les détails d'une construction. La liste des constructions est à droite et il suffit de cliquer sur un élément de la liste pour avoir les détails.

Pour faire la construction, j'ai déclaré un projet de type «build» qui est exécuté aux 30 secondes. Ce projet exécute un interpréteur de commande et reçoit en paramètre le script à exécuter. Le script lance la mise à jour des répertoires sources et lance la compilation à distance. Je n'ai pas fait sortir les sources du contrôleur de sources par CruiseControl puisque c'était plus simple de le faire dans le script. C'est-à-dire qu'il suffit de lancer la commande SVN pour sortir les sources, ce qui est plus simple que de configurer Cruise Control pour sortir les sources dans le répertoire partagé de la compilation.

Le script ne sort pas tous les sources sur un répertoire vide, mais plutôt fait une mise à jour des sources pour éviter de tout recompiler ce qui prendrait plus d'une heure au lieu de quelques secondes.

Tableau 7 Évaluation de CruiseControl

Critère	Description	Poids	Cote
1. Performance	*Le processus de construction doit s'exécuter rapidement, dans un délai raisonnable.* La construction s'effectue dans un délai raisonnable.	2	1.0
2. Facilité de configuration	*La configuration d'un système de construction à base de «makefile» doit être facile à réaliser.* C'est la plus difficile à configurer des trois solutions. La configuration se fait dans un fichier XML.	10	0.2
3. Historiques	*L'outil doit conserver un historique des constructions réalisées.* Oui, il y a un historique des constructions effectuées.	3	1.0
4. Facilité d'utilisation	*Indique si l'outil est facile à utiliser pour les développeurs qui vont interagir avec.* Une fois, configuré CruiseControl est facile d'utilisation.	10	1.0
5. Administration de l'outil	*L'outil doit être simple à administrer par l'administrateur du système.* Il n'y a pas beaucoup d'administration à faire.	2	1.0
6. Automatisation	*L'outil doit pouvoir fonctionner de façon normale au sein de l'entreprise, sans nécessiter d'intervention de la part d'un utilisateur.* CruiseControl fonctionne de façon autonome une fois configuré.	5	1.0

Critère	Description	Poids	Cote
7. Documentation et support gratuits	*Facilité à trouver de la documentation et de l'aide gratuite pour la configuration (documentation, FAQ, forums, etc.).* Oui, il y a une liste de discussion.	5	1.0
8. Support payant	*L'entreprise ou une autre compagnie offre du support payant en cas de problème.* Aucun support payant n'est disponible.	5	0
	Total	42	29

10.3 Évaluation de Apache Continuum

Je n'ai pas réussi à faire fonctionner l'outil pour faire une construction avec un script Shell. L'option est pourtant supportée, mais j'avais des erreurs internes dans le log lorsque la commande était lancée. Je n'ai pas évalué l'outil puisqu'il ne répondait pas au besoin de l'entreprise pour faire la construction.

Apache Continuum est écrit en Java et a besoin d'un serveur applicatif comme Tomcat pour fonctionner. Il peut être lancé de façon autonome. C'est de cette façon que je l'ai utilisé.

Je donne la note de 0 puisque je n'ai pas été en mesure de le faire fonctionner pour qu'il réponde au besoin.

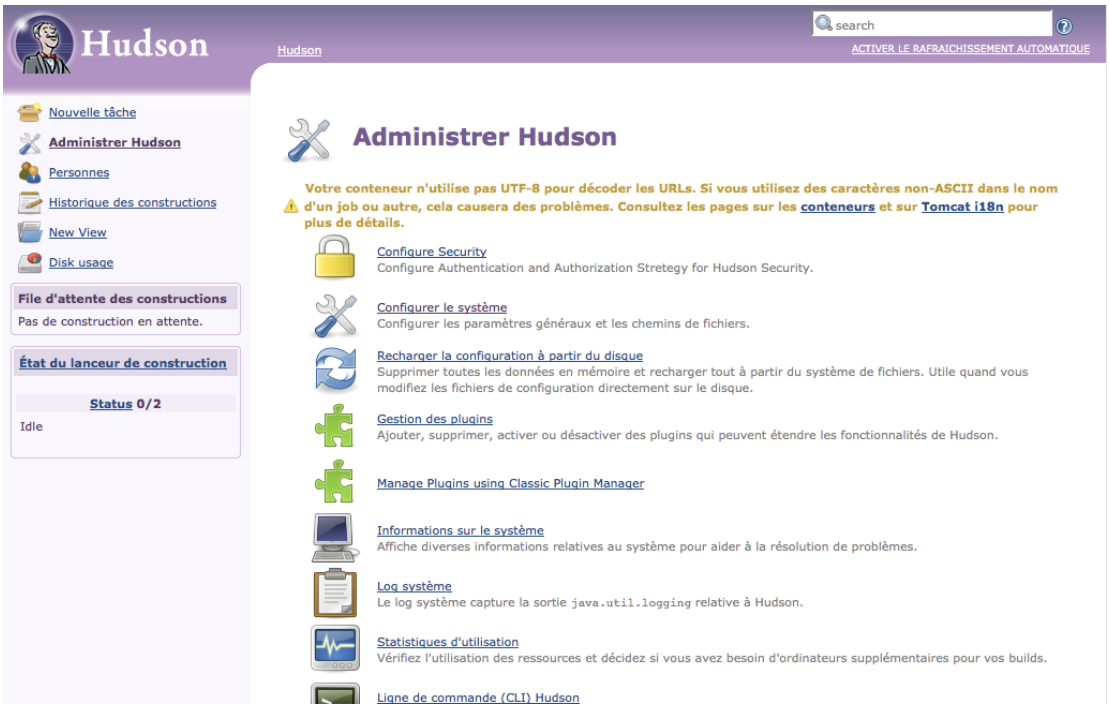
10.4 Évaluation de Hudson

Hudson est écrit en Java et a besoin d'un serveur applicatif pour fonctionner. J'ai utilisé Tomcat comme serveur applicatif. Pour lancer Hudson, il suffit de placer la distribution WAR dans le répertoire webapps de Tomcat et il se fait déployer.

La configuration se fait à travers l'interface usager et est très conviviale. La simplicité de la configuration m'a surpris. Il y a une configuration globale pour les paramètres qui s'appliquent à tous les projets de construction et une configuration par construction.

Voici une partie de l'interface de configuration de Hudson. Cela va vous donner une idée de tout ce qui est configurable. Hudson est très paramétrable.

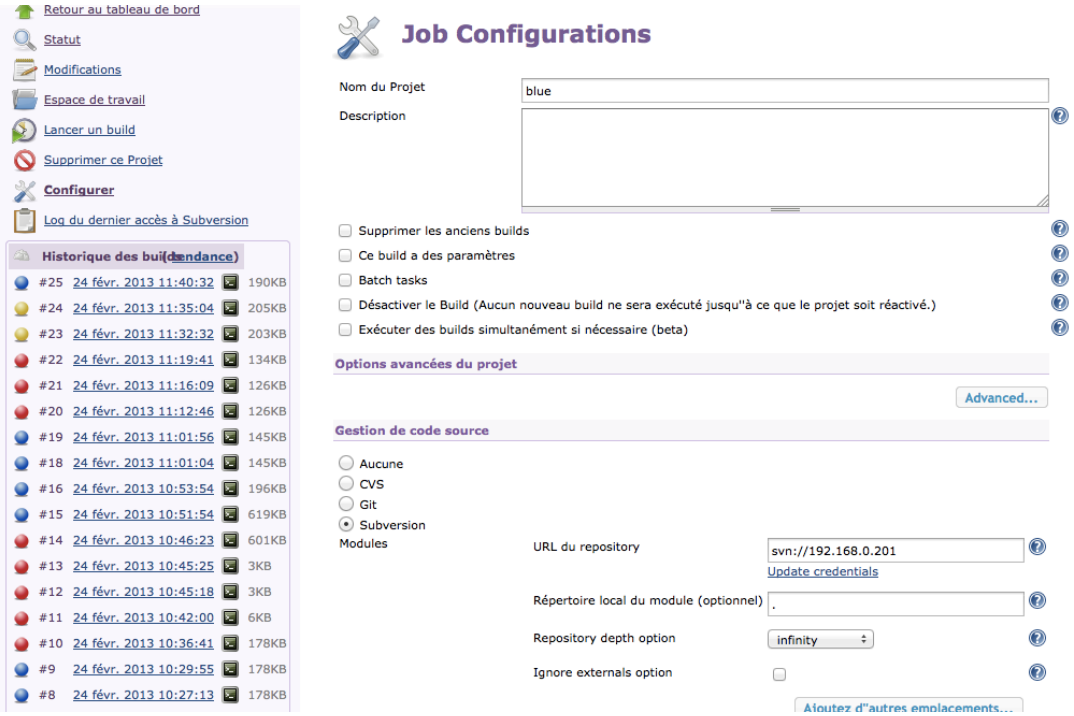
Figure 4 Menu de configuration de Hudson



Hudson fonctionne avec des modules d'extension pour ajouter de la fonctionnalité. Il suffit d'installer et de configurer le module d'extension pour avoir la fonctionnalité voulue. Il y a un vaste choix de modules d'extension.

La configuration d'un projet de construction se fait à travers une interface utilisateur. Il suffit de choisir le type de projet de construction et de le configurer. Pour l'évaluation, j'ai créé un projet script Shell. Voici l'interface de configuration d'un projet de construction.

Figure 5 configuration d'une construction



Pour faire la construction, j'ai configuré un répertoire spécifique pour les sources qui est le répertoire sur la machine Linux via SSHFS. Lorsque Hudson détecte une modification dans les sources, il sort les sources du contrôleur de source et lance le script de construction qui fait la compilation à distance grâce à des commandes SSH.

Lorsqu'une construction est en court, il est possible d'avoir la sortie de la ligne de commande de la construction pour la suivre. Cela peut être pratique pour roder le processus de construction.

Comme on peut le voir dans la capture d'écran, il y a un historique des constructions. Une construction avec un cercle bleu a fonctionné comme il faut. Avec un cercle rouge, la compilation a été en erreur. Et finalement, avec le cercle jaune, la construction est instable puisqu'il y a des tests unitaires qui n'ont pas fonctionné.

Tableau 8 Évaluation de Hudson

Critère	Description	Poids	Cote
1. Performance	*Le processus de construction doit s'exécuter rapidement, dans un délai raisonnable.* La construction s'effectue dans un délai raisonnable.	2	1.0
2. Facilité de configuration	*La configuration d'un système de construction à base de «makefile» doit être facile à réaliser.* Hudson est très facile à configurer.	10	1.0
3. Historiques	*L'outil doit conserver un historique des constructions réalisées.* Oui, il y a un historique des constructions effectuées.	3	1.0
4. Facilité d'utilisation	*Indique si l'outil est facile à utiliser pour les développeurs qui vont interagir avec.* Hudson est très facile à utiliser.	10	1.0
5. Administration de l'outil	*L'outil doit être simple à administrer par l'administrateur du système.* Des utilisateurs peuvent être configurés comme administrateur pour gérer l'outil.	2	1.0
6. Automatisation	*L'outil doit pouvoir fonctionner de façon normale au sein de l'entreprise, sans nécessiter d'intervention de la part d'un utilisateur.* Hudson est autonome.	5	1.0
7. Documentation et	*Facilité à trouver de la documentation et de l'aide gratuite*	5	1.0

Critère	Description	Poids	Cote
support gratuits	*pour la configuration (documentation, FAQs, forums, etc.).* Il y a un forum de discussion et une liste de discussion.		
8. Support payant	*L'entreprise ou une autre compagnie offre du support payant en cas de problème.* Oracle ne semble pas offrir de support payant pour ce produit.	5	0
	Total	42	37

10.5 Comparatif des résultats pour l'évaluation des outils d'intégration continue

Tableau 7 Tableau comparatif des résultats des outils d'intégration continue

Critère	CruiseControl	Hudson
1. Performance	2	2
2. Facilité de configuration	2	10
3. Historique	3	3
4. Facilité d'utilisation	10	10
5. Administration de l'outil	2	2
6. Automatisation	5	5
7. Documentation et support gratuits	5	5
8. Support payant	0	0
Total	29	37

11.0 Interprétation des résultats

11.1 Outils de tests unitaires

11.1.1 Facilité d'utilisation

À l'usage, Google Test se révèle être l'outil le plus simple d'utilisation, suivi de Boost Test et finalement de CPP Unit.

Google Test et Boost Test ne nécessitent pas beaucoup de code supplémentaire pour effectuer les tests unitaires. L'approche est simple, il suffit d'utiliser une macro pour définir une fonction et le tour est joué.

Par contre, c'est beaucoup plus lourd pour CPP Unit. Il faut déclarer une classe de test unitaire et une fonction «main» où il faut enregistrer les tests unitaires à l'exécuteur de la classe de test. CPP Unit est un grand perdant à cause de cela. Le code à réaliser est plutôt verbeux. Le fait d'avoir à créer du code autre que celui des tests unitaires ajoute inutilement à la charge de travail du développeur de logiciel, ce qui n'est pas nécessaire avec les deux autres outils de tests unitaires.

Google Test l'emporte sur ce critère pour sa facilité d'utilisation dans l'ensemble.

11.1.2 Assertions

Les trois outils n'offrent pas de contrainte pour faire des assertions. Même si Boost Test n'offre pas beaucoup de fonctionnalités pour les assertions, cela fonctionnait quand même. Ce critère ne permet pas de les distinguer.

Les trois outils ne se distinguent pas beaucoup puisqu'ils satisfont tous à la tâche des tests unitaires, à part pour la simplicité d'utilisation et les rapports XML. Tous fonctionnent bien aucun bogue n'a été remarqué.

11.1.3 Rapports de tests

Seul Google Test génère des rapports XML dans un format directement assimilable par les outils d'intégration continue. Pour Boost Test et CPPUnit un XSLT a dû être utilisé pour les transformer dans le bon format. Google Test l'emporte sur ce point.

11.1.4 Configuration des options d'exécution des tests

Pour Google Test, la configuration de l'exécution se fait à l'aide de variables d'environnement. Boost Test, lui, exige que les options soient passées à la ligne de commande qui lance les programmes de

tests unitaires. Alors que pour CPP Unit, il faut spécifier les options dans le code, qu'il faut recompiler à chaque changement dans les options.

CPP Unit est le grand perdant de cette section, alors que les deux autres produits sont ex aequo puisqu'ils passent les options en paramètres.

11.1.5 Conclusion

Le meilleur outil de test unitaire parmi les trois est Google Test. Les trois outils font bien la tâche de réaliser des tests unitaires et ils sont assez semblables sur bien des points, mais Google Test est le plus simple à utiliser et il ne nécessite aucune modification des rapports XML. Le pointage sur les grilles d'évaluation souligne ce choix.

11.2 Outils d'intégration continue

Puisque je n'ai pas été en mesure de faire fonctionner Apache Continuum, je vais faire la comparaison seulement avec Cruise Control et Hudson. Les deux outils ne se démarquent pas beaucoup l'un de l'autre puisque les deux font le travail demandé. Par contre, Hudson me semble plus élégant et complet que Cruise Control. La raison est peut-être qu'il est plus récent et mieux conçu que Cruise Control.

Un grand avantage de Hudson par rapport à Cruise Control est la simplicité de configuration. La configuration se fait à partir d'une interface utilisateur et non d'un fichier XML. Hudson supporte plus d'options de configuration que Cruise Control grâce à ses modules d'extension. Il y a une grande gamme de modules d'extension qui permette de personnaliser la fonctionnalité d'Hudson. Hudson est beaucoup plus convivial à configurer et d'une simplicité frappante.

Hudson supporte en plus des groupes d'utilisateurs et d'administrateurs qui peuvent être définis pour contrôler leur accès au système. Je n'ai pas tenu compte de cela dans mon évaluation puisque cela n'était pas requis pour le projet au sein de l'entreprise, mais je trouve que c'est un avantage pour Hudson. Si cette fonctionnalité devenait nécessaire, Hudson le supporte déjà.

Les deux outils permettent de lancer des scripts Shell. Cela est nécessaire pour pouvoir exécuter à distance le «makefile». Je n'ai pas eu de problème à configurer les deux outils le lancement de scripts. Quoique Hudson était plus convivial avec son interface utilisateur.

En conclusion, je recommande Hudson comme outil d'intégration continue. Le pointage sur les grilles d'évaluation supporte ce choix. Je crois que Hudson va remplir pleinement le besoin et en plus, si

la fonctionnalité devient nécessaire dans le futur, il peut gérer des groupes. Aussi, la convivialité de la configuration dans des interfaces utilisateurs graphiques rend son utilisation simple et agréable.

12.0 Conclusion

La démarche de sélection et d'essai a permis d'atteindre l'objectif principal du projet qui était de «… proposer des solutions automatisées de tests unitaires et d'intégration continue pour un environnement de développement C++, dans le but de réduire le nombre de bogues dans les produits de l'entreprise».

Les deux outils sélectionnés, Google Test pour les tests unitaires et Hudson pour l'intégration continue, ont été intégrés à un environnement d'essai qui reproduisait l'environnement de développement réel de l'entreprise. Les deux produits ont donc été intégrés au processus de construction des produits de l'entreprise. Les deux produits ont démontré leurs fonctionnalités et répondent bien aux critères du projet.

Google Test et Hudson permettront d'automatiser le processus de tests unitaires, ce qui devrait permettre à l'entreprise d'atteindre ses objectifs en terme de qualité.

La prochaine étape consistera à élaborer une stratégie d'implantation qui prendra en compte l'impact sur les processus organisationnels, la formation des ressources, la justification des coûts pour les nouveaux serveurs et leur installation, l'installation et la configuration des logiciels et la conversion au nouvel environnement (création de tous les tests unitaires requis).

J'ai atteint les objectifs de ce travail. Si l'entreprise implémentait les solutions proposées, la qualité des produits et la satisfaction des clients seraient à la hausse. Je crois donc que l'entreprise bénéficierait de la mise en place de la solution.

Les résultats de l'étude ont permis de distinguer deux gagnants. Google Test pour les tests unitaires et Hudson pour l'outil d'intégration continue.

La fonctionnalité de base a été satisfaite par les trois outils, mais la complexité d'utilisation n'était pas la même pour tous les outils. Google Test était de loin le plus simple à utiliser. Aussi, pour la génération des rapports en XML, qui sont utilisés par les outils d'intégration continue, il y avait que Google Test qui était dans le bon format. Les deux autres ont dû être convertis avec un script XSLT.

Le gagnant pour les solutions d'intégration continue a été Hudson pour sa facilité d'utilisation et de configuration. Je n'ai pas été en mesure de tester Apache Continuum puisque l'exécution de script ne fonctionnait pas pour lancer la compilation en C++. Donc, je n'en tiens pas compte dans mes comparaisons. Cruise Control, l'autre outil d'intégration continue délivrait la fonctionnalité de base, mais

il était beaucoup plus difficile à configurer puisque sa configuration se fait à travers d'un fichier de configuration XML. L'évaluation démontre que Hudson est le gagnant puisque sa configuration est plus simple à travers d'une interface graphique.

Les tests d'intégration n'ont pas été inclus dans le projet pour limiter la complexité et la durée du projet d'évaluation. Normalement, il faudrait aussi avoir des tests d'intégration. Ceci est une limite de l'étude. Sinon, il n'y a pas eu d'autre limite à l'étude à part cela.

Les bénéfices pour l'entreprise sont nombreux. Premièrement, une façon d'automatiser l'exécution des tests unitaires. Cela va permettre de mieux contrôler la qualité du produit lors de la modification du code source. Aussi, cela va augmenter la satisfaction des clients qui vont recevoir un produit contenant moins de bogues. Cela ne va pas empêcher certains bogues d'être découverts par le client, mais cela va les réduire d'une grande partie. Deuxièmement, il y aurait amélioration du processus de développement qui est assez chaotique. Cela pourrait paver la route à mettre d'autre processus d'amélioration de la qualité comme des tests d'intégration.

13.0 Réflexion

Je crois que l'objectif principal du projet, soit d'identifier comment améliorer la qualité des produits de l'entreprise, a été atteint. J'ai été en mesure d'ajouter dans le processus de construction du produit des outils de tests unitaires et d'intégration continue. Cela va permettre d'augmenter la qualité des produits développés par l'entreprise.

Les deux outils sélectionnés, soit Google Test pour les tests unitaires et Hudson pour l'intégration continue, sont fonctionnels et répondent bien aux besoins de l'entreprise. Ils devraient être en mesure d'aider l'entreprise à atteindre ses objectifs en termes de qualité.

La prochaine étape relève de la gestion du changement organisationnel et ne fait pas partie de la portée ce projet. Les processus de travail des développeurs devront être modifiés et ils devront créer des tests unitaires pour leurs produits. Rien ne garantit l'acceptation de ces changements par les développeurs. Certains sont très ouverts à ce type de changements, alors que d'autres pourraient résister.

Il faut aussi tenir compte de l'effort initial requis pour créer tous les tests unitaires pour tous les produits et toutes les librairies. C'est un effort non négligeable, qui peut exiger beaucoup de temps-ressource. Afin de minimiser l'impact sur les développeurs, on pourrait peut-être engager des stagiaires en informatique pour réaliser le plus gros du travail initial.

L'utilisation de tests unitaires et de processus d'intégration continue sont des pratiques reconnues du génie logiciel. La mise en place de ces outils pourra améliorer la qualité et la fiabilité des produits de l'entreprise. L'entreprise pourrait rapidement réaliser une analyse coûts vs bénéfices et une stratégie d'implantation détaillée devra être produite.

Sur le plan personnel, j'ai appris plusieurs nouvelles technologies en réalisant ce projet.

Il y a évidemment les nouvelles connaissances techniques que j'ai acquises sur les outils de tests unitaires et d'intégration continue en C++, mais il y en a aussi d'autres. Par exemple, je ne connaissais pas SSHFS avant d'en avoir entendu parler sur l'internet lors de la réalisation de l'environnement d'essai. SSHFS est beaucoup plus simple à configurer que NFS et je compte l'utiliser à nouveau.

J'ai aussi appris à utiliser la technologie XSLT que je n'avais jamais utilisée et que j'ai dû apprivoiser pour faire la conversion de rapport XML de tests unitaires. C'est beaucoup plus efficace d'utiliser XSLT que de faire les conversions avec un script de conversion de fichiers. N'importe qui peut

maintenant faire référence à la documentation existante pour mettre à jour ou modifier les règles de conversion.

Finalement, je me suis aussi familiarisé avec l'installation du serveur applicatif Tomcat et des serveurs d'intégration continue Hudson et Continuum.

Parce que j'ai réalisé ce projet sur mon temps personnel, l'entreprise ne m'a imposé aucune contrainte dans sa réalisation, tout en me permettant d'utiliser du code réel pour les essais et une partie de leur infrastructure. On m'a donné carte blanche pour la réalisation du projet.

Cela m'a permis de mettre en pratique plusieurs apprentissages acquis tout au long de ma maîtrise et d'en faire de nouveaux. Ces nouvelles connaissances me seront utiles dans le milieu du travail.

De façon plus spécifique, dans le cadre de ce projet j'ai appris à :

- définir l'objectif d'un projet à partir des intentions vagues de l'entreprise,
- comprendre le problème à l'étude, faire un survol de la littérature sur le sujet, en comprendre les composants principaux et les adapter aux besoins du projet,
- développer et appliquer une méthode pour la sélection des produits (grilles d'évaluation à critères pondérés),
- définir et structurer un ensemble viable de solutions à évaluer,
- définir et mettre en place une infrastructure d'essai pour les solutions,
- installer, configurer, exécuter et évaluer les solutions proposées,
- choisir et proposer une solution finale.

Les méthodes et outils de travail que j'ai développés dans ce projet vont me permettre de proposer des solutions aux problèmes de génie logiciel qui se posent lors de la réalisation de projets. Cette nouvelle connaissance va me permettre de me différencier de mes collègues, par une meilleure efficacité dans le travail et par une meilleure qualité des résultats produits.

Je suis maintenant en mesure de prendre en main un projet complexe qui implante plusieurs technologies, de les évaluer et de les utiliser pour atteindre les objectifs du projet.

Ce projet m'a aussi permis de démontrer mon autonomie et ma capacité d'identifier et de structurer des solutions informatiques viables et immédiatement réalisables.

Il n’y a pas eu de difficultés que je n’ai su surmonter en utilisant des nouvelles technologies que je ne connaissais pas auparavant. Les principales difficultés du projet étaient le partage de répertoire entre machines Unix et le formatage de rapport XML des outils de tests unitaires CPP Unit et Boost Test. Pour cela j’ai utilisé SSHFS pour partager les répertoires et XSLT pour le formatage de rapport.

14.0 Citations

[1] N.U. Noel Unknown, Exploring the C++ Unit Testing Framework Jungle, 2004 (mis à jour 2010) [En ligne]. Disponible: http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle. Consulté le 28 janvier 2012.

[2] N.U. Noel Unknown, Exploring the C++ Unit Testing Framework Jungle, 2004 (mis à jour 2010) [En ligne]. Disponible: http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle. Consulté le 28 janvier 2012. p. 2.

[3] N.U. Noel Unknown, Exploring the C++ Unit Testing Framework Jungle, 2004 (mis à jour 2010) [En ligne]. Disponible: http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle. Consulté le 28 janvier 2012. p. 2.

[4] N.U. Noel Unknown, Exploring the C++ Unit Testing Framework Jungle, 2004 (mis à jour 2010) [En ligne]. Disponible: http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle. Consulté le 28 janvier 2012. p. 3.

[5] N.U. Noel Unknown, Exploring the C++ Unit Testing Framework Jungle, 2004 (mis à jour 2010) [En ligne]. Disponible: http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle. Consulté le 28 janvier 2012. p. 3.

[6] M.F. Martin Fowler, Continuous Integration, 2007 [En ligne]. Disponible: http://martinfowler.com/articles/continuousIntegration.html. Consulté le 16 octobre 2011. p. 8-9

[7] N.K. Neo Krates, Choosing continuous integration (CI) tool. Comparison : CruiseControl, Hudson, Continuu, vs. TeamCity, 2010 [En ligne]. Disponible: http://www.thinkplexx.com/learn/article/build-chain/ci/choosing-continuous-integrationci-tool-opensource-vs-professional-cruisecontrol-hudson-continuum-vs-teamcity. Consulté le 16 octobre 2011. p. 2.

[8] N.K. Neo Krates, Choosing continuous integration (CI) tool. Comparison : CruiseControl, Hudson, Continuu, vs. TeamCity, 2010 [En ligne]. Disponible: http://www.thinkplexx.com/learn/article/build-chain/ci/choosing-continuous-integrationci-tool-opensource-vs-professional-cruisecontrol-hudson-continuum-vs-teamcity. Consulté le 16 octobre 2011. p. 1.

15.0 Références

A.O. Alex Ott, Test-driven development and unit testing with examples in C++, 2007 [En Ligne]. Disponible: http://alexott.net/en/cpp/CppTestingIntro.html. Consulté le 16 octobre 2011.

A.H. Aslak Hellesoy, Continuous Integration Server Feature Matrix, 2008 [En ligne]. Disponible: http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix. Consulté le 16 octobre 2011

C.R. Chris Read, Quick comparison of TeamCity 1.2, Bamboo 1.0 and CruiseControl 2.6, 2007 [En ligne]. Disponible: http://blog.chris-read.net/2007/02/21/quick-comparison-of-teamcity-12-bamboo-10-and-cruisecontrol-26. Consulté le 16 octobre 2011.

C.R. Chris Read, Continuous Integration Server Comparison 2008, 2008 [En ligne]. Disponible: http://blog.chris-read.net/2008/09/22/continuous-integration-server-comparison-2008/. Consulté le 16 octobre 2011.

E.V. Erez Volk, CxxTest User's Guide, 2010 [En ligne]. Disponible: http://www.tigris.org/files/documents/6421/43284/cxxtest-guide-3.10.1.pdf. Consulté le 22 avril 2012.

G Google, Primer: Getting started with Google C++ Testing Framework, 2010 [En ligne]. Disponible: http://code.google.com/p/googletest/wiki/Primer. Consulté le 22 avril 2012.

M.F. Martin Fowler, Continuous Integration, 2007 [En ligne]. Disponible: http://martinfowler.com/articles/continuousIntegration.html. Consulté le 16 octobre 2011.

N.K. Neo Krates, Choosing continuous integration (CI) tool. Comparison : CruiseControl, Hudson, Continuu, vs. TeamCity, 2010 [En ligne]. Disponible: http://www.thinkplexx.com/learn/article/build-chain/ci/choosing-continuous-integrationci-tool-opensource-vs-professional-cruisecontrol-hudson-continuum-vs-teamcity. Consulté le 16 octobre 2011.

N.U. Noel Unknown, Exploring the C++ Unit Testing Framework Jungle, 2004 (mis à jour 2010) [En ligne]. Disponible: http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle. Consulté le 28 janvier 2012.

P.F. Peter Franza, Comparison of continuous integration servers, 2008 [En ligne]. Disponible: http://www.peterfranza.com/2008/09/26/comparison-of-continuous-integration-servers/. Consulté le 16 octobre 2011.

P.N. Phil Nash, Unit Testing in C++ and Objective-C just got easier, 2010 [En ligne]. Disponible: http://www.levelofindirection.com/journal/2010/12/28/unit-testing-in-c-and-objective-c-just-got-easier.html. Consulté le 16 octobre 2011.

W.B. Warner Beroux, C++ Unit Testing Framework: A Boost Test Tutorial, 2009 [En ligne]. Disponible: http://www.beroux.com/english/articles/boost_unit_testing/. Consulté le 22 avril 2012.

R.W. Rick wagner, Continuous Integration for C++, 2009 [En ligne]. Disponible: http://rickwagner.blogspot.ca/2009/04/continuous-integration-for-c.html. Consulté le 16 octobre 2011.

Annexe

Exemple d'un test unitaire Google Test

TestDataGTest.cxx

```
#include "Data.h"
#include "gtest/gtest.h"
TEST(TestDataSuite, TestParse)
{
	using namespace blueslice::policy::data;
	std::string name, data;
	// Test positif
	std::string xmlToParse=
	"<?xml version=\"1.0\" encoding=\"UTF-8\"?><subscriber><data
name=\"quota\"><![CDATA[[<?xml version=\"1.0\" encoding=\"UTF-
8\"?><usage>"
	"<version>1</version><quota
name=\"q1\"><cid>92233720368547758O7</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
	"<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-
05:00</nextresettime></quota></usage>]]></data></subscriber>";
	// Test si le parsing est bon
	EXPECT_EQ(parse(xmlToParse,name,data,"/subscriber/data"),true);
	// test si le resultat est bon
	std::string dataResult="<?xml version=\"1.0\" encoding=\"UTF-
8\"?><usage>"
	"<version>1</version><quota
name=\"q1\"><cid>92233720368547758O7</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
	"<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
	EXPECT_EQ(data,dataResult);

	std::string nameResult="quota";
	EXPECT_EQ(name,nameResult);
	//test negatif
	std::string xmlToParse2=
	"<?xml version=\"1.0\" encoding=\"UTF-8\"?><subscriber<data
name=\"quota\"><![CDATA[[<?xml version=\"1.0\" encoding=\"UTF-
8\"?><usage>"
	"<version>1</version><quota
name=\"q1\"><cid>92233720368547758O7</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
	"<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-
05:00</nextresettime></quota></usage>]]></data></subscriber>";
	EXPECT_EQ(parse(xmlToParse2,data,name,"/subscriber/data"),false);
```

```
}

TEST(TestDataSuite,TestValidation)
{
	using namespace blueslice::policy::data;
	std::string name, data;
	// Test positif
	std::string xmlToParse=
	"<?xml version=\"1.0\" encoding=\"UTF-8\"?><usage>"
	"<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
	"<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
	// Test si le parsing est bon
	EXPECT_EQ(validation(xmlToParse,"quota"),true);

	// Test negatif
	// Test si le xml est pas bon
	std::string xmlToParse2=

	"<?xml version=\"1.0\" encoding=\"UTF-8\"?><usage>"
	"<version1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
	"<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
	EXPECT_EQ(validation(xmlToParse2,"quota"),false);
}

TEST(TestDataSuite,TestGenerate)
{
	using namespace blueslice::policy::data;
	std::string entity;
	std::string name="quota";
	// Test positif
	std::string data=
	"<?xml version=\"1.0\" encoding=\"UTF-8\"?><usage>"
	"<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
	"<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
	// Test si le parsing est bon
	generate(entity,name,data,true);
	std::cout << entity << std::endl;
	std::string nothing="";
	EXPECT_EQ(entity,nothing);
}
```

Exemple d'un de tests unitaire Boost

TestDataBoost.cxx

```
#ifndef BOOST_TEST_DYN_LINK
#define BOOST_TEST_DYN_LINK
#endif
#include "Data.h"
#define BOOST_TEST_MODULE TestDataBoost
#include <boost/test/unit_test.hpp>
#include <boost/bind.hpp>
bool gUseClassicLogging = true;
#include <string>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>


BOOST_AUTO_TEST_SUITE( test_data_suite )

BOOST_AUTO_TEST_CASE(test_data_parse)
{
  using namespace blueslice::policy::data;
  std::string name, data;
  // Test positif
  std::string xmlToParse=
  "<?xml version=\"1.0\" encoding=\"UTF-8\"?><subscriber><data
name=\"quota\"><![CDATA[[<?xml version=\"1.0\" encoding=\"UTF-
8\"?><usage>"
  "<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
  "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-
05:00</nextresettime></quota></usage>]]></data></subscriber>";
  // Test si le parsing est bon
  BOOST_CHECK_EQUAL(parse(xmlToParse,name,data,"/subscriber/data"),true
);
  // test si le resultat est bon
  std::string dataResult="<?xml version=\"1.0\" encoding=\"UTF-
8\"?><usage>"
  "<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
  "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
  BOOST_CHECK_EQUAL(data,dataResult);

  std::string nameResult="quota";
  BOOST_CHECK_EQUAL(name,nameResult);
  //test negatif
```

```
        std::string xmlToParse2=
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><subscriber<data
name=\"quota\"><![CDATA[[<?xml version=\"1.0\" encoding=\"UTF-
8\"?><usage>"
        "<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
        "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-
05:00</nextresettime></quota></usage>]]></data></subscriber>";
        BOOST_CHECK_EQUAL(parse(xmlToParse2,data,name,"/subscriber/data"),fal
se);

      }

      BOOST_AUTO_TEST_CASE(test_data_validation)
      {
        using namespace blueslice::policy::data;
        std::string name, data;
        // Test positif
        std::string xmlToParse=
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><usage>"
        "<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
        "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
        // Test si le parsing est bon
        BOOST_CHECK_EQUAL(validation(xmlToParse,"quota"),true);

        // Test negatif
        // Test si le xml est pas bon
        std::string xmlToParse2=

        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><usage>"
        "<version1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
        "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
        BOOST_CHECK_EQUAL(validation(xmlToParse2,"quota"),false);
      }


      BOOST_AUTO_TEST_CASE(test_data_generate)
      {
        using namespace blueslice::policy::data;
        std::string entity;
        std::string name="quota";
        // Test positif
        std::string data=
```

```
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><usage>"
        "<version>1</version><quota
name=\"q1\"><cid>92233720368547758O7</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
        "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
        // Test si le parsing est bon
        generate(entity,name,data,true);
        std::cout << entity << std::endl;
        std::string nothing="";
        BOOST_CHECK_EQUAL(entity,nothing);
      }
      struct LogToFile
      {
         LogToFile()
         {
            std::string
logFileName(boost::unit_test::framework::master_test_suite().p_name);
            logFileName.append(".xml");
               std::string lfName("../../../../resboost/");
               lfName.append(logFileName);
            logFile.open(lfName.c_str());
               logFile << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" ;
            boost::unit_test::unit_test_log.set_stream(logFile);
         }
         ~LogToFile()
         {
            logFile << "</TestLog>" << std::flush;
            logFile.close();
            boost::unit_test::unit_test_log.set_stream(std::cout);
            // convert the report.
               std::string
logFileName(boost::unit_test::framework::master_test_suite().p_name);
           logFileName.append(".xml");
               std::string lfName("../../../../resboost/");
               lfName.append(logFileName);

               std::string
logConvFileName(boost::unit_test::framework::master_test_suite().p_name);
               logConvFileName.append(".xml");
               std::string lfConvName("../../../../res/");
               lfConvName.append(logConvFileName);

               std::string cmd("java -jar /home/ybourdea/saxon/saxon9he.jar -s:");
               cmd+=lfName;
               cmd+=" -xsl:boost.xsl -o:";
               cmd+=lfConvName;
               std::cout << "cmd:" << cmd << std::endl;
               system(cmd.c_str());
         }
         std::ofstream logFile;
      };
```

```
BOOST_GLOBAL_FIXTURE(LogToFile);

BOOST_AUTO_TEST_SUITE_END()

bool init_function()
{

   // do your own initialization here
   // if it successful return true

   // But, you CAN'T use testing tools here

   return true;
}
```

Exemple d'un test unitaire CPP Unit

TestData.h

```
#ifndef TestData_H__
#define TestData_H__

#include <iostream>
#include <string>

#include <cppunit/TestCase.h>
#include "cppunit/TestSuite.h"
#include "cppunit/TestCaller.h"
#include "cppunit/TestRunner.h"

#include "Data.h"
using namespace CppUnit;
class DataTestCase : public TestCase
{
public:
  // Constructor
  DataTestCase(std::string name): TestCase(name) {}
  DataTestCase() {}

  //Method to test
  void testParse();
  void testValidation();
  void testGenerate();
  static Test *suite();
};
#endif
```

TestData.cxx

```
#include "TestData.h"
#include <cppunit/CompilerOutputter.h>
#include <cppunit/TestResult.h>
#include <cppunit/TestResultCollector.h>
#include <cppunit/TestRunner.h>
#include <cppunit/TextTestProgressListener.h>
#include <cppunit/BriefTestProgressListener.h>
#include <cppunit/XmlOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <stdexcept>
#include <fstream>
#include <string>
#include <stdio.h>
#include <stdlib.h>

using namespace CppUnit;
void DataTestCase::testParse()
{
  using namespace blueslice::policy::data;
  std::string name, data;
  // Test positif
  std::string xmlToParse=
  "<?xml version=\"1.0\" encoding=\"UTF-8\"?><subscriber><data
name=\"quota\"><![CDATA[[<?xml version=\"1.0\" encoding=\"UTF-
8\"?><usage>"
  "<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
  "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-
05:00</nextresettime></quota></usage>]]></data></subscriber>";
  // Test si le parsing est bon
  CPPUNIT_ASSERT_EQUAL(parse(xmlToParse,name,data,"/subscriber/data"),tr
ue);

  // test si le resultat est bon
  std::string dataResult="<?xml version=\"1.0\" encoding=\"UTF-
8\"?><usage>"
  "<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
  "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
  CPPUNIT_ASSERT_EQUAL(data,dataResult);

  std::string nameResult="quota";
  CPPUNIT_ASSERT_EQUAL(name,nameResult);
  //test negatif
  std::string xmlToParse2=
```

```
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><subscriber<data
name=\"quota\"><![CDATA[[<?xml version=\"1.0\" encoding=\"UTF-
8\"?><usage>"
        "<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
        "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-
05:00</nextresettime></quota></usage>]]></data></subscriber>";
        CPPUNIT_ASSERT_EQUAL(parse(xmlToParse2,data,name,"/subscriber/data"),
false);
      }
      //test
      void DataTestCase::testValidation()
      {
        using namespace blueslice::policy::data;
        std::string name, data;
        // Test positif
        std::string xmlToParse=
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><usage>"
        "<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
        "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
        // Test si le parsing est bon
        CPPUNIT_ASSERT_EQUAL(validation(xmlToParse,"quota"),true);

        // Test negatif
        // Test si le xml est pas bon
        std::string xmlToParse2=

        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><usage>"
        "<version1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
        "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
        CPPUNIT_ASSERT_EQUAL(validation(xmlToParse2,"quota"),false);

      }
      void DataTestCase::testGenerate()
      {
        using namespace blueslice::policy::data;
        std::string entity;
        std::string name="quota";
        // Test positif
        std::string data=
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><usage>"
        "<version>1</version><quota
name=\"q1\"><cid>9223372036854775807</cid><time>1234</time><totalVolu
```

```
me>99999999999</totalVolume><inputvolume>5000</inputvolume><outputvolu
me>15000</outputvolume>"
        "<servicespecific>12</servicespecific><nextresettime>2010-05-
12T16:00:00-05:00</nextresettime></quota></usage>";
        // Test si le parsing est bon
        generate(entity,name,data,true);
        std::cout << entity << std::endl;
        std::string nothing="";
        CPPUNIT_ASSERT_EQUAL(entity,nothing);
      }

      Test *DataTestCase::suite()
      {
        TestSuite *testSuite = new TestSuite ("DataTestCase");
        Test          *t=new          TestCaller          <DataTestCase          >
("testParse",&DataTestCase::testParse);
        testSuite->addTest (t);
        testSuite->addTest       (new       TestCaller       <DataTestCase       >
("testValidation",&DataTestCase::testValidation));
        testSuite->addTest         (new         TestCaller         <DataTestCase>
("testGenerate",&DataTestCase::testGenerate));
        return testSuite;
      }

      int main(int argc, char *argv[])
      {

        std::string testPath = (argc > 1) ? std::string(argv[1]) : std::string("");

        // Create the event manager and test controller
        CPPUNIT_NS::TestResult controller;

        // Add a listener that colllects test result
        CPPUNIT_NS::TestResultCollector result;
        controller.addListener( &result );

        // Add a listener that print dots as test run.
        CPPUNIT_NS::BriefTestProgressListener progress;
        controller.addListener( &progress );

        // Add the top suite to the test runner
        CPPUNIT_NS::TestRunner runner;
        runner.addTest( DataTestCase::suite() );
        try
        {
          CPPUNIT_NS::stdCOut() << "Running "  <<  testPath;
          runner.run( controller, testPath );

          CPPUNIT_NS::stdCOut() << "\n";

          // Print test in a compiler compatible format.
          CPPUNIT_NS::CompilerOutputter                outputter(                &result,
CPPUNIT_NS::stdCOut() );
          outputter.write();
```

```
// Uncomment this for XML output
    std::ofstream file( "../../../../rescppunit/TestData.xml" );
    CPPUNIT_NS::XmlOutputter xml( &result, file );
    xml.write();
    file.close();
  std::string cmd("java -jar /home/ybourdea/saxon/saxon9he.jar -s:");
  cmd+="../../../../rescppunit/TestData.xml";
  cmd+=" -xsl:cppunit.xsl -o:";
  cmd+="../../../../res/TestData.xml";
  std::cout << "cmd:" << cmd << std::endl;
  system(cmd.c_str());
  }
  catch ( std::invalid_argument &e )  // Test path not resolved
  {
    CPPUNIT_NS::stdCOut()  <<  "\n"
                       <<  "ERROR: "  <<  e.what()
                       << "\n";
    return 0;
  }

  return result.wasSuccessful() ? 0 : 1;

  return 0;
}
```

Configuration de Cruise Control

```
<cruisecontrol>
   <project name="Blue">

      <listeners>
         <currentbuildstatuslistener file="/Users/yannbourdeau/cc-
svndir/${project.name}/status.txt"/>
      </listeners>

      <bootstrappers>
         <svnbootstrapper localWorkingCopy="/Users/yannbourdeau/cc-
svndir" />
      </bootstrappers>

      <modificationset quietperiod="30">
         <svn RepositoryLocation="svn://192.168.0.201"
username="yannbourdeau" password="Bombastic"/>
      </modificationset>

      <schedule interval="30">
         <exec command="/bin/bash" args="/Users/yannbourdeau/cc-
svndir/192.168.0.201/cc-buildblue.sh"/>
      </schedule>

      <log>
         <merge dir="/Users/yannbourdeau/cc-svndir/res"/>
      </log>

      <publishers>


       <htmlemail mailhost="relais.videotron.ca"
returnaddress="yann.bourdeau@mac.com" skipusers="false"
spamwhilebroken="true">
          <map alias="yannbourdeau" address="yann.bourdeau@mac.com"/>
             <always address="yannbourdeau@mac.com"/>
       </htmlemail>

     </publishers>
   </project>
</cruisecontrol>
```

Fichier XSLT pour Boost

```
<?xml version="1.0"?>
<!-- addition-1_0.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xsl:output method="xml" cdata-section-elements="failure"/>

<xsl:template match="/TestLog/TestSuite">
  <xsl:element name ="testsuites">
        <xsl:attribute name="tests">
        <xsl:value-of
select="count(/TestLog/TestSuite/TestSuite/TestCase)"/>
        </xsl:attribute>
        <xsl:attribute name="failures">
              <xsl:value-of
select="count(/TestLog/TestSuite/TestSuite/TestCase/Error)"/>
        </xsl:attribute>
        <xsl:attribute name="disabled">
              <xsl:text>0</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="errors">
              <xsl:text>0</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="time">
              <xsl:value-of
select="sum(/TestLog/TestSuite/TestSuite/TestCase/TesttimeTime)"/>
        </xsl:attribute>

        <xsl:attribute name ="name">
        <xsl:value-of select="/TestLog/TestSuite/@name"/>
        </xsl:attribute>
        <xsl:element name="testsuite">
              <xsl:attribute name="tests">
              <xsl:value-of
select="count(/TestLog/TestSuite/TestSuite/TestCase)"/>
              </xsl:attribute>
              <xsl:attribute name="failures">
                    <xsl:value-of
select="count(/TestLog/TestSuite/TestSuite/TestCase/Error)"/>
              </xsl:attribute>
              <xsl:attribute name="disabled">
                    <xsl:text>0</xsl:text>
              </xsl:attribute>
              <xsl:attribute name="errors">
                    <xsl:text>0</xsl:text>
              </xsl:attribute>
              <xsl:attribute name="time">
                    <xsl:value-of
select="sum(/TestLog/TestSuite/TestSuite/TestCase/TesttimeTime)"/>
              </xsl:attribute>
```

```
                    <xsl:for-each select="/TestLog/TestSuite/TestSuite/TestCase">
                        <xsl:element name="testcase">
                            <xsl:attribute name="name">
                                <xsl:value-of select="@name"/>
                            </xsl:attribute>
                            <xsl:attribute name="status">
                                <xml:text>run</xml:text>
                            </xsl:attribute>
                            <xsl:attribute name="time">
                                <xsl:value-of select="TestingTime"/>
                            </xsl:attribute>
                            <xsl:attribute name="classname">
                                <xsl:value-of
select="/TestLog/TestSuite/@name"/>
                            </xsl:attribute>
                            <xsl:for-each select="Error">

                                <xsl:element name="failure">
                                    <xsl:attribute name="message">
                                        <xsl:value-of select="."/>
                                    </xsl:attribute>
                                    <xsl:attribute name="type"/>
                                        <xsl:value-of select="."/>

                                </xsl:element>
                            </xsl:for-each>
                        </xsl:element>
                    </xsl:for-each>
                </xsl:element>
            </xsl:element>
        </xsl:template>
        </xsl:stylesheet>
```

Fichier XSLT pour CPP Unit

```
<?xml version="1.0"?>
<!-- addition-1_0.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xsl:output method="xml" cdata-section-elements="failure"/>
<xsl:template match="/TestRun">
  <xsl:element name ="testsuites">
        <xsl:attribute name="tests">
              <xsl:value-of select="Statistics/Tests"/>
        </xsl:attribute>
        <xsl:attribute name="failures">
              <xsl:value-of select="Statistics/Failures"/>
        </xsl:attribute>
        <xsl:attribute name="disabled">
              <xsl:text>0</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="Errors">
              <xsl:value-of select="Statistics/Errors"/>
        </xsl:attribute>
        <xsl:attribute name="name">
              <xsl:text>AllTests</xsl:text>
        </xsl:attribute>
        <xsl:element name="testsuite">
              <xsl:attribute name="name">
                    <xsl:text>AllTestsSuite</xsl:text>
              </xsl:attribute>

              <xsl:attribute name="tests">
                    <xsl:value-of select="Statistics/Tests"/>
              </xsl:attribute>
              <xsl:attribute name="failures">
                    <xsl:value-of select="Statistics/Failures"/>
              </xsl:attribute>
              <xsl:attribute name="disabled">
                    <xsl:text>0</xsl:text>
              </xsl:attribute>
              <xsl:attribute name="Errors">
                    <xsl:value-of select="Statistics/Errors"/>
              </xsl:attribute>
              <xsl:for-each select="SuccessfulTests/Test">
                    <xsl:element name="testcase">
                          <xsl:attribute name="name">
                                <xsl:value-of select="Name"/>
                          </xsl:attribute>
                          <xsl:attribute name="status">
                                <xml:text>run</xml:text>
                          </xsl:attribute>
                          <xsl:attribute name="time">
                                <xsl:text>0</xsl:text>
                          </xsl:attribute>
```

```
                        </xsl:element>
                    </xsl:for-each>
                    <xsl:for-each select="FailedTests/FailedTest">
                        <xsl:element name="testcase">
                            <xsl:attribute name="name">
                                <xsl:value-of select="Name"/>
                            </xsl:attribute>
                            <xsl:attribute name="status">
                                <xml:text>run</xml:text>
                            </xsl:attribute>
                            <xsl:attribute name="time">
                                <xsl:text>0</xsl:text>
                            </xsl:attribute>
                            <xsl:element name="failure">
                                <xsl:attribute name="message">
                                    <xsl:value-of
select="Message"/>
                                </xsl:attribute>
                                <xsl:value-of select="Location/File"/>
                                <xsl:value-of select="Location/Line"/>
                                <xsl:value-of select="Message"/>
                            </xsl:element>
                        </xsl:element>
                    </xsl:for-each>
                </xsl:element>
            </xsl:element>
        </xsl:template>
        </xsl:stylesheet>
```

Script pour lancer le build Cruise Control

```
cd /Users/yannbourdeau/cc-svndir/res
rm -f *
cd /Users/yannbourdeau/cc-svndir/resboost
rm -f *
cd /Users/yannbourdeau/cc-svndir/rescppunit
rm -f *
cd /Users/yannbourdeau/cc-svndir/192.168.0.201
svn up
ssh ybourdea@192.168.0.210 'export GTEST_OUTPUT=xml:/home/ybourdea/cc-
svndir/res/;cd cc-svndir/192.168.0.201/Blue; make exec'
```

Script pour lancer le build Hudson

```
#!/bin/sh
set -e
cd /Users/yannbourdeau/.hudson/jobs/blue/workspace
ssh ybourdea@192.168.0.210 'export
GTEST_OUTPUT=xml:/home/ybourdea/cc-svndir/res/;cd cc-svndir/res; rm * ; cd
../192.168.0.201/Blue; svn up ; make exec'
scp ybourdea@192.168.0.210:cc-svndir/res/* res/.
```