Mise en oeuvre de tests unitaires dans un contexte de programmation eXtrème répartie

Ibrahim Lokpo* - Michel Babri* - Gérard Padiou**

* Institut National Polytechnique Félix Houphouet-Boigny
Département Mathématiques et Informatique
Yamoussoukro, Côte d'Ivoire
lokpo@hotmail.com, michel_babri@yahoo.fr

** Institut de Recherche en Informatique de Toulouse, UMR CNRS 5505
ENSEEIHT, BP 7122, 2 rue Camichel, 2
F-31071 Toulouse cedex 7
padiou@enseeiht.fr

RÉSUMÉ. La programmation eXtrème dite XP (eXtreme Programming) est une méthodologie efficace de développement rapide de logiciels qui exige que les membres d'une équipe de développeurs soient physiquement proches les uns des autres. Des travaux de recherche tentent d'étendre les principes de cette métodologie à un contexte réparti. Le défi est alors de préserver les qualités fondamentales de l'approche tout en s'affranchissant de la contrainte de proximité physique des développeurs. Notre travail s'inscrit dans cette logique. De façon plus précise, nous proposons de fonder cette extension de XP à un environnement réparti sur l'assistance à la réalisation des tests unitaires, pierre angulaire de la métodologie XP.

ABSTRACT. eXtreme Programming (XP) is a methodology based on principles and practices for quicklyly developing software. However this approach requires the programmers to be co-located. Many research projects investigate how to extend XP to a distributed environment. However the challenge is to carry out the XP approach without conflicting with the distributed constraints. Our work takes place in this way. More precisely, we propose an extension based on assistance for supporting distributed unit testing one of the key principles of XP methodology.

MOTS-CLÉS : Programmation XP, CSCW , Tests unitaires, XP réparti.

KEYWORDS : eXtreme Programming, CSCW, unit testing, distributed XP.

1. Introduction

La programmation eXtrème ou eXtreme Programming (XP) [2,19] est une méthodologie de production de logiciels qui connaît un large succès depuis quelques années. Avec cette méthodologie, il s'agit en effet de mettre à disposition le plus tôt possible, un produit opérationnel de qualité. Elle repose sur le principe du développement guidé par les tests unitaires et la programmation par paire [3,7]. Dans sa forme originale, cette méthodologie présente toutefois quelques sévères limitations dont les principales sont la localisation centralisée et le nombre limité des développeurs. En effet, XP n'est plus adapté dès lors que l'effectif des équipes de développement devient élevé car méthodologie exige, en particulier, une réelle et importante communication entre les programmeurs. A partir d'un certain seuil, il est difficile à un grand nombre d'interlocuteurs de s'entendre sur l'essentiel au cours d'une discussion. Dans ce cas, fournir des documents, même en grand nombre, devient plus intéressant pour les développeurs. XP nécessite surtout que les membres d'une équipe de développement soient géographiquement proches les uns des autres. Dans certains cas de grands projets menés par différentes équipes à travers le monde, ou des cas de mobilité des membres d'une équipe, ces limitations peuvent constituer un frein au développement du logiciel.

Des travaux de recherche ont été menés pour lever ces limitations. Ces travaux ont permis d'aboutir au concept de Programmation eXtrème répartie (Distributed XP) [4, 11, 14]. Le défi posé est de préserver les qualités fondamentales de la méthodologie tout en s'affranchissant de la contrainte de proximité physique des développeurs. Notre travail s'inscrit dans cette logique. C'est une contribution à la préservation des fondements de XP dans un contexte réparti en se fondant sur l'assistance au test unitaire, principe essentiel de la méthodologie. Dans ce document, nous analysons une mise en œuvre des tests unitaires dans un contexte d'une équipe répartie de développeurs.

Dans une première section nous évaluons les problèmes liés à la mise en pratique de la programmation XP dans un contexte réparti. La section suivante expose quelques solutions envisagées dans différents projets de recherche. Nous exposons ensuite les principes et la mise en œuvre de notre solution. Nous terminons par un cas pratique d'utilisation de la solution proposée.

2. XP et la répartition

La méthodologie DXP [11] se définit comme une extension de la méthodologie XP dans laquelle la contrainte de localisation géographique des membres d'une équipe de développeurs a été levée. En tant que telle, cette approche DXP doit relever de

nouveaux défis notamment dans les pratiques XP fortement liées à cette contrainte de localisation induite par la plupart de ses principes de base tels que la programmation par paire, le jeu de rôle, l'intégration continue. Nous évoquons certains de ces problèmes dans cette section.

2.1. La programmation par paire

La programmation par paire est une des principales pratiques XP dans laquelle deux programmeurs travaillent sur une même machine. Les deux sont actifs en même temps. Pendant que l'un pense et écrit le code, l'autre suit et cherche à savoir si l'approche adoptée est correcte, ou à trouver éventuellement une autre solution. Ainsi, les binômes peuvent apporter des changements à tout endroit du code. Dans un binôme, les rôles sont interchangeables dans la même journée ; ceci amène le binôme à une parfaite maîtrise de son sujet. Une des questions que l'on peut se poser est quel serait le comportement d'une paire de programmeurs si ceux-ci ne sont plus sur la même machine, voire s'il sont géographiquement distants. C'est un des défis à relever pour pratiquer la programmation XP en réparti.

2.2. Les tests unitaires et de recette

On distingue deux types de tests:

- le test fonctionnel dans lequel l'on ne vérifie que, compte tenu de données d'entrée, les résultats en sortie. Le module testé est considéré comme une boîte noire. Ni la modification des paramètres ni la gestion des erreurs ne sont prises en compte. En fait, avec un tel test, il s'agit de s'assurer que des paramètres corrects donnent toujours des résultats corrects.

- le second type de test consiste à tenir compte du fonctionnement interne du module et des liens entre modules. Le test donnera également des informations sur l'état interne du module ou de la classe.

Les programmeurs sont amenés à écrire toute une batterie de tests automatiques des classes ou modules de l'application, et ceci avant qu'ils n'écrivent les différents codes d'implantation correspondants. Lorsque ces tests de non régression sont satisfaits, les développeurs sont assurés que leurs modifications n'ont pas introduit d'erreurs dans les jeux de tests précédents.

Le comportement d'ensemble de l'application sera, quant à lui, pris en charge par des tests dits de recette. Ceux-ci permettent de capter l'état d'avancement de l'ensemble d'un projet. Les tests de recette sont généralement effectués par le représentant des utilisateurs finaux qui doit être présent sur le site de développement. Etendre les tests à un environnement réparti est un défi difficile pour au moins deux raisons :

- les problèmes liés au déploiement des composants logiciels et à la topologie du réseau sous adjacent ;

- la difficulté liée à la synchronisation de l'exécution des tests.

La solution technique que nous proposons permet principalement aux paires de programmeurs de réaliser facilement leurs tests de non régression dans un environnement réparti. Notre solution contribue aussi à faciliter la réalisation de test de recette car elle permet de rassembler toutes les classes de tests d'un projet en une seule classe de tests.

2.3. Intégration continue et partage de responsabilité

Ces deux propriétés sont étroitement liées. Le partage de responsabilité indique que toute paire de programmeurs peut modifier et incorporer tout ou partie du code. Par intégration continue, XP exige que la suite de tests soit tout de suite exécutée. De cette façon, les éventuels problèmes induits apparaissent immédiatement. Toutefois, l'intégration continue ne peut se faire qu'à des instants précis. Ceci devient primordial dès lors que les binômes sont distants.

3. Solutions pour appliquer XP dans un contexte réparti

Le développement des télécommunications et plus particulièrement d'Internet a permis à de nombreuses équipes dispersées dans le monde de collaborer à un même projet. De telles équipes partagent le code sans pour autant être obligées de se rencontrer physiquement. La contrainte de proximité géographique imposée par XP, aux équipes de développement peut donc être levée pour permettre à des équipes virtuelles d'appliquer les autres recommandations XP. Dans un tel environnement, les problèmes de communication et de fiabilité des informations sont fondamentaux. Par exemple, lorsqu'une version de tout ou partie d'une application est disponible, les équipes utilisatrices doivent en être informées.

Plusieurs travaux de recherche ont proposé des solutions permettant de s'affranchir des principales limitations initiales de XP tout en conservant son efficacité [1]. La programmation par paire telle que prescrite par XP oblige les membres du binôme à travailler sur la même machine et donc dans un même bureau. Or, de nombreuses applications telles que le télé-enseignement, peuvent être organisées en binômes. Il peut arriver que les membres d'un binôme ne soient pas sur le même campus ou aient des disponibilités quelques fois incompatibles.

Dans ces conditions, il paraît intéressant de leur donner tout de même les moyens de développer par paire (programmation par paire répartie). Internet offre actuellement de

nombreux outils qui peuvent être mis à profit pour faire de la programmation par paire répartie (DPP) [18]. Il s'agit essentiellement de l'utilisation de la messagerie électronique, de l'audio et de la vidéo conférence, des tableaux blancs et des forums de causette. Des logiciels de gestion de versions concurrentes ainsi que des environnements de développement intégré sont également utilisés. En plus des avantages classiques de la programmation par paire, l'étude de [18] relève les points suivants :

- les développeurs ont une lisibilité plus forte car chacun dispose d'un écran,
- les déplacements ne sont pas nécessaires,
- les binômes ont la possibilité de revenir en arrière pour voir une version précédente, car ils conservent une copie de leur travail,
- l'ordinateur est le moyen de communication, et les membres n'ont pas le temps de parler d'autre chose que de leur développement.

Les participants à cette étude ont toutefois fait ressortir quelques problèmes parmi lesquels on peut citer :

- l'arrêt de l'un des membres entraîne celui de l'autre,
- l'éloignement physique nécessite beaucoup de temps d'explication verbale.

4. Un système de tests unitaires répartis

Les résultats de différents travaux de recherche permettent de s'affranchir des contraintes initiales de XP sans perte de son efficacité. Par exemple [11, 14, 17] ont proposé la notion d'équipes virtuelles pour garantir les fonctionnalités de CSCW (Computer-Supported Cooperative Work) dans un contexte réparti, notamment la programmation par paire. De telles approches nécessitent l'existence d'un réseau fiable. Nous proposons une approche qui tient compte d'un contexte particulier dans lequel le réseau de connexion n'offre pas un haut débit. Nous nous intéressons aux tests unitaires dans un environnement réparti, comme base d'une application de XP dans un tel contexte. Notre outil s'adresse particulièrement aux équipes de développement dispersées ou mobiles qui souhaitent toutefois adopter XP comme méthodologie. Cet outil est un moyen destiné à des paires de programmeurs distantes qui veulent écrire des tests et les modules associés dans un projet auquel elles collaborent.

4.1. Choix d'un environnement de programmation

Cette expérimentation a été faite en s'appuyant sur l'environnement de programmation BlueJ(http://bluej.org). Il s'agit d'un environnement de développement intégré pour Java. C'est un outil initialement prévu pour l'enseignement des concepts

orientés « objet » et qui intègre maintenant un outil d'assistance aux tests unitaires, Junit(http://www.junit.org), développé dans le cadre de la programmation eXtrème [12, 15].

Nous avons également utilisé JML [5, 6, 13], un langage de spécification de modules Java permettant de faciliter le développement des tests par génération automatique. De plus, la gestion des versions concurrentes des modules à développer a nécessité l'utilisation d'un logiciel adéquat. Notre choix s'est porté sur CVS (http://www.cvshome.org). CVS permet de suivre toutes les modifications opérées sur tout ou partie du code source d'un projet. Toutefois, CVS est basé sur la notion de copie centrale, de telle sorte que les développeurs ne travaillent que sur une copie locale. Une commande spécifique permet ensuite de fusionner de façon intelligente toutes les modifications. De plus, ce logiciel permet d'avoir accès à n'importe quelle version du projet.

Nous avons intégré l'ensemble de ces outils pour construire un environnement unique appelé JUTE. La section suivante décrit de façon plus détaillée cet environnement.

4.2. JUTE : un Environnement de Tests Répartis pour Java

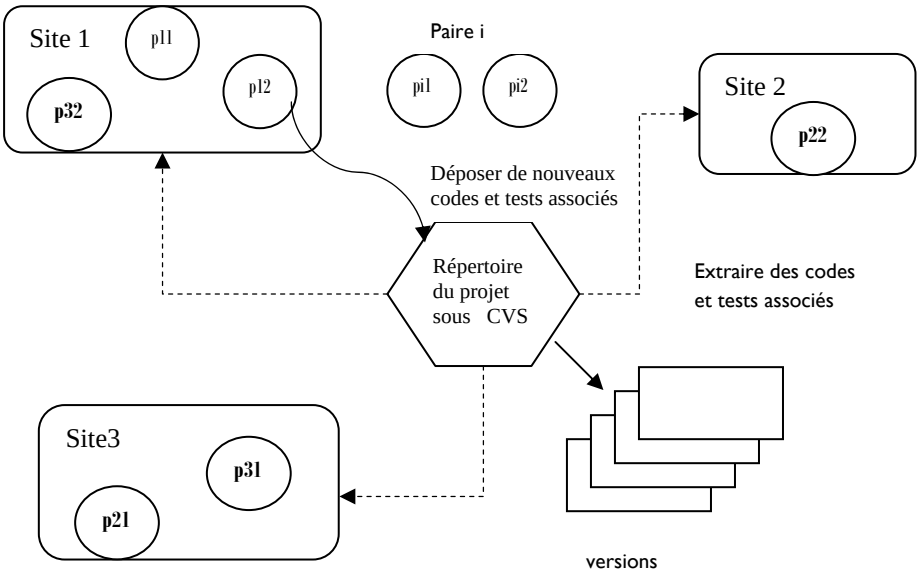
JUTE (Java Unit Testing Environnement) est un outil qui permet aux paires de programmeurs d'appliquer un nombre important de tests unitaires sans se préoccuper ni de l'organisation de ceux-ci, ni de la manière dont ils s'exécutent. Des outils tels que Junit et BlueJ [12, 15] ont été introduits pour les tests unitaires [10] au cours d'un développement XP. Cependant, ces outils non seulement imposent une méthodologie très stricte, mais nécessitent surtout que les programmeurs écrivent eux-mêmes les tests.

Le projet JML (Java Modelling Language) permet de spécifier formellement des propriétés des classes d'un programme Java : pré et post conditions, invariants, etc. Ces spécifications sont insérées dans les classes sources sous la forme de commentaires spéciaux qui sont interprétés par les outils de JML. Pour pouvoir assurer la vérification de ces propriétés à l'exécution, ces classes doivent être instrumentées en utilisant un tisseur jmlc qui instrumente les classes engendrées.

Pour faciliter la réalisation des tests unitaires, la composante jmlunit de JML engendre une suite de tests au format Junit à partir des spécifications introduites dans les classes. En effet, pour un fichier source donné C.java, l'outil engendre deux fichiers : C_JML_Test.java qui décrit la liste des tests à effectuer et C_JML_Testdata.java qui contient les données de test. En fonction des types de données qui serviront à faire le test, le programmeur doit compléter certaines méthodes du fichier de données.

Nous pensons qu'intégrer les outils jmlc et jmlunit dans l'environnement de développement intégré Bluej, constitue un apport appréciable dans la production des tests. En programmation eXtrème, les tests unitaires doivent être exécutés de façon la plus systématique possible [8]. Si ceux-ci passent avec succès, les développeurs sont sûrs d'avoir obtenu une nouvelle version de leur application globale. Il est donc nécessaire de construire une suite globale de tests pour l'ensemble du projet. Dans cette optique, nous avons écrit un générateur automatique qui explore la hiérarchie de paquetages du projet afin d'engendrer une suite globale de tests. Le programme ainsi obtenu sera compilé et exécuté par Junit.

Une pratique importante de XP est le réajustement continu qui peut être obtenu grâce à BlueJ qui fournit un environnement convivial de développement de programmes. Par ailleurs, le critère de proximité géographique de XP peut être relâché en utilisant un gestionnaire de versions concurrentes tel que CVS (http://www.cvshome.org). La figure 1 montre la conception générale de notre environnement. Les paires de programmeurs (Pi1, Pi2) peuvent être soit localisées sur une même machine (cas de la paire 1), soit localisées sur des machines distantes. Grâce à un outil de partage d'accès à distance d'une machine tel que VNC (Virtual Network Computing)[16], une paire de programmeurs distants peut construire un module commun cohérent. Chaque paire envoie son code après l'avoir testé; or le test n'est rien d'autre que du code en plus; celui-ci est donc également envoyé au dépôt (copie centrale du test). Chaque paire de programmeurs récupère ce dont elle a besoin (code et tests associés) à partir du dépôt central. Grâce à l'outil mis en place, la paire reconstruit sa suite de tests et l'exécute.



Lors de **figure1. *Structure générale du système de tests répartis*** t provenir de deux sortes d'erreurs :

-un comportement marginal n'a peut-être pas été prévu ou des interactions entre modules n'ont pas été prises en compte ;

- une erreur due au nouveau code qui vient d'être écrit.

Lorsqu'un test est concluant, son résultat apparaît dans la hiérarchie des classes du projet. Dès lors, les principales fonctions, entre autres celles de CVS peuvent être appelées. Un exemple de scénario de la figure 1 se présente comme suit : supposons que la paire P3 de programmeurs (P31, P32) désire écrire un module M3 qui fait appel aux modules M1 de (P11, P12) et M2 de (P21, P22). JUTE permet à P3 d'extraire du dépôt central (Test repository) M1 et M2 ainsi que les tests associés et de construire de façon automatique les tests associés à M3.

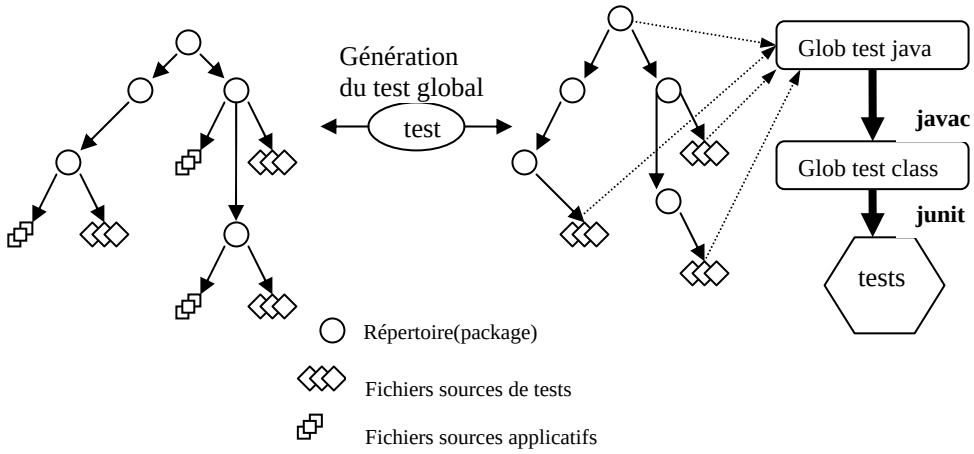


Figure 2. *Commande test : exécution d'un ensemble de tests unitaires*

4.3. Etat actuel du projet

Dans la version actuelle du projet, les trois extensions à savoir la génération automatique de suite de tests d'un module de projet, la génération et l'exécution automatiques de la suite globale des tests de tout un projet et la gestion des versions concurrentes, ont été intégrées entièrement à l'environnement BlueJ. La figure 3 (Menu de base) montre cette version étendue qui fait donc apparaître un menu à trois options :

- l'option test concerne la génération d'une suite globale de tests qui sera par la suite compilée et exécutée par JUnit. Comme l'illustre la figure 2, cette commande collecte systématiquement l'ensemble des classes de tests du projet courant (engendrées automatiquement par JML ou écrites manuellement par le programmeur) et engendre un fichier global de test à la racine du projet.

-L'option jml affiche la liste de toutes les classes du projet courant (au sens BlueJ). Cette option met à disposition deux commandes. Pour la classe sélectionnée, la commande jmlc assure l'instrumentation de cette classe et la commande jmlunit engendre une suite de tests unitaires et un squelette de jeu de données pour cette classe.

- La troisième option, cvs, présente à l'utilisateur également une liste des sources des classes et la liste des commandes CVS disponibles applicables aux modules sélectionnés.

Ceci constitue une première étape d'assistance au test unitaire dans un contexte réparti indispensable à la pratique de la programmation eXtrème. Une autre caractéristique à développer est la programmation par paire.

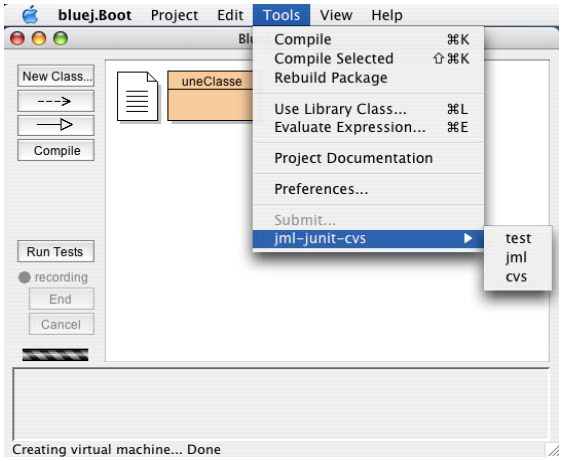


Figure 3 : Menu de base de l'extension

4.4. Un exemple de génération de tests

Selon la méthodologie XP de base, la paire de programmeurs localisée sur le même site, doit d'abord commencer par produire une série de tests avant même d'écrire le code dont les fonctionnalités seront testées. Avec notre solution, ce principe de « test-first development » reste toujours valable. Mais en plus, nous donnons la possibilité de produire les tests dans un contexte réparti et nous facilitons également la production de ces tests. En effet, l'intégration de l'outil JML dans notre solution technique permet d'automatiser l'écriture de tests fonctionnels.

Nous proposons donc de décrire dans cette section un exemple très simple mais suffisant pour illustrer ce principe de génération automatique des tests. Notre exemple s'appuie sur une classe écrite en Java. Il s'agit d'une classe nommée ***Calculette*** comprenant les méthodes ***add***, ***sub, mul*** et ***div*** qui permettent respectivement d'effectuer l'addition, la soustraction, la multiplication et la division de deux nombres. Dans notre exemple, nous voulons nous assurer que la méthode ***add*** réalise effectivement l'addition. Pour cela nous allons vérifier que cette méthode ***add*** respecte la propriété « *0 est un élément neutre de l'addition* ». Le reste de la section indique, étape par étape, comment procéder lors de l'utilisation de l'outil JUTE.

Etape 1

Pour des développeurs respectant la méthodologie XP, la première étape consiste normalement à écrire en Java une classe de tests pour vérifier que la méthode add (codée après l'écriture de la classe de tests) respecte la propriété « *0 est un élément neutre de l'addition : (pour tout X, X+0 = X ; pour tout Y, 0+Y = Y)* ». Mais ici, dans l'environnement JUTE, on va simplement se contenter de spécifier cette propriété en utilisant le langage de spécification JML. L'expression de cette propriété pour la méthode add(X,Y) donne :

ensures Y= = 0 ➔ \result = = X ;
ensures X= = 0 ➔ \result = = Y ;

Etape 2

On introduit l'expression des propriétés ainsi définies dans le code de la méthode sous la forme de commentaires comme l'illustre les lignes ci-après.

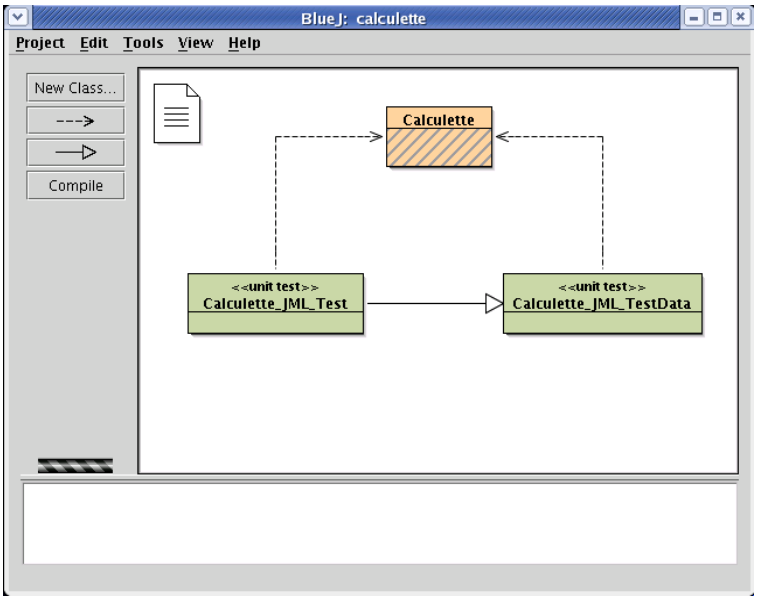
```
public class Calculette
{
        private String name ;
        public Calculette(String nom)
        {       Name = nom ;
        }
        //@ ensures Y= = 0 ➔ \result = = X
```

```
        //@ ensures X= = 0 ➔ \result = = Y
        public int add(int X, int Y)
        {       int somme = (X+Y) ;
                return somme ;
        }
}
```

À ce stade, si nous visualisons le projet dans l'environnement BlueJ, on ne voit seulement que la classe ***Calculette***. Cette classe sera compilée avec l'outil JUTE.

Etape 3

À partir de cette classe ainsi annotée, l'outil JUTE va utiliser JML et plus spécifiquement la commande *jmljunit*, pour automatiquement engendrer deux autres classes comme le montre la figure ci-après. La classe ***Calculette_JML_Test*** (de test) permet de vérifier si la méthode ***add*** de la classe ***Calculette*** respecte la contrainte spécifiée. La classe ***Calculette_JML_TestData*** permet de fournir les données à utiliser au cours des tests. Cette deuxième classe pourra être manuellement modifiée si on désire fournir des données spécifiques ou supplémentaires pour le test. Les endroits où il faut introduire ces données dans la classe ***Calculette_JML_TestData*** sont indiqués sous la forme de commentaires dans le code de cette classe.



Etape 4

Ces deux nouvelles classes sont compilées avec le compilateur normal de Java. On obtient ainsi des tests prêts à être lancés.

L'outil JUTE permet, grâce à CVS qu'il intègre, à plusieurs paires de programmeurs d'effectuer ces quatre étapes de façon répartie et en parallèle. La commande test de JUTE se chargera de parcourir toute l'arborescence du projet et de rassembler tous les tests pour lancer leur exécution.

5. Conclusion

Dans ce papier, nous avons décrit et présenté notre approche pour la pratique de tests unitaires dans un environnement réparti. L'environnement JUTE permet en effet de construire une suite de tests qui sont exécutés de façon quasi automatique même par des équipes distantes. Il s'appuie sur trois extensions apportées à l'environnement de développement Bluej. La première permet de générer puis d'exécuter une suite globale de tests, la seconde (jmlc) engendre les sources des classes et les suites de tests unitaires. La troisième incorpore l'utilisation de CVS, un gestionnaire de versions concurrentes. La contribution de JUTE est d'une part, d'engendrer de façon automatique la suite globale de tests à partir de la hiérarchie des paquetages d'un projet en cours de développement et d'autre part, de permettre de relâcher la contrainte de proximité physique imposée aux programmeurs, grâce à l'utilisation d'un gestionnaire de versions concurrentes comme CVS.

Les choix des différents environnements et logiciels utilisés se justifient essentiellement par leur caractère "open source" et, d'un point de vue performance, par l'absence de moyen de communication à haut débit. D'autres approches complémentaires sont en cours de développement, en particulier, le projet XPWeb (http://xpweb.sourceforge.net/) de serveur Web offrant des outils de base de gestion de la coopération (CSCW) dans un développement XP et le projet associé EcliWPWeb (http://sourceforge.net/projects/eclixpweb/) permettant une connexion sous forme de plug-in avec l'environnement de développement intégré Eclipse (http:/www.eclipse.org). Par rapport à ces travaux en cours, notre approche s'en distingue par sa focalisation sur le test avec en particulier l'intégration de JML. Cependant, selon la même démarche que pour l'environnement Eclipse, l'utilisation du service Web de type CSCW offert par XPWeb pourrait être intégré en tant qu'extension de l'environnent BlueJ.

Notre projet est mené dans le cadre d'une collaboration entre l'Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et de

Télécommunication (ENSEEIHT) de Toulouse (France), via son laboratoire de recherche (IRIT) et l'Institut National Polytechnique Félix Houphouët-Boigny (INP-HB) de Yamoussoukro (Côte d'Ivoire). La prochaine étape sera l'évaluation du serveur XPWeb pour l'assistance à la programmation par paire à partir de l'environnement BlueJ étendu actuel.

6. Bibliographie et biographie

6.1 Bibliographie

[1] BAHETI P., GEHRINGER E., And STOTTS D. "Exploring the efficacy of distributed pair programming", http://rockfish-cs.cs.unc.edu/pubs/XPU2002DXP.pdf, 2001.

[2] BECK K. "Extreme Programming Explained : Embrace Change." The XP Series. Addison Wesley Publishing Company, 2000.

[3] BECK K. "Test-driven Development by example." The Addison Wesley Signature Series. Addison Wesley Publishing Company, 2001.

[4] BOWEN S. and MAURER F. "Using Peer-to-Peer Technology to Support Global Software Development – Some Initial Thoughs." In Proceedings of the Workshop on Cooperative Supports for Distributed Software Engineering Processes. (26-29th August 2002) 26th IEEE Annual International Computer Software and Application Conference (COMPSAC), 2002.

[5] CHEON Y. and LEAVENS G.T. "A Runtime assertion checker for Java Modeling Language (JML)." In H.R. Arabnia and editors Y. Mun, editors, International Conference on Software Engineering Research and Practice (SERP '02), pages 322–328. CSREA Press, Las Vegas, 2002.

[6] CHEON Y. and LEAVENS G.T. "A simple and practical approach to unit testing : The JML and JUnit way." In Hern´andez Juan (Eds.), editor, 16th European Conference ECOOP 2002 Workshops and Posters, volume 2548 of Lecture Notes in Computer science. Springer Verlag, 2002.

[7] CROS T. "Maîtriser les projets avec l'eXtreme programming : pilotage par les tests clients" CEPADUES EDITIONS, 111 rue Vauquelin, 31100 Toulouse, 2004

[8] GAMMA E. and BECK K. "Test infected : Programmers love writing tests". Java Report, 3,7 (July 1998).

[9] GAMMA E. and BECK K. "Eclipse : Principes, patterns et plug-in" Réference Campus Press, 2004.

[10] GASSMANN P. "Unit testing in a Java project. In Kent Beck, editor, Extreme Programming Examined, The XP Series, pages 249–269. Addison-Wesley Publishing Company, 2001.

[11] KIRCHER M., JAIN P., CORSARO A. and LEVINE D. “Distributed extreme programming.” In XP2001 - eXtreme Programming and Flexible Processes in Software Engineering. Villasimius, Sardinia, Italy, May 2002.

[12] KÖLLING M., QUIG B., PATTERSON A. and ROSENBERG J. “The BlueJ system and its pedagogy.” Journal of Computer Science Education, 13,4, (December 2003.)

[13] LEAVENS G. T., RUSTAN K, LEINO M., POLL E., RUBY C., and JACOBS B. “JML : notations and tools supporting detailed design in java.” In ACM Conference on Object-oriented Programming, Systems, languages, and applications (OOPSLA’00) Companion, pages 105–106, August 2000.

[14] MAURER F. “Supporting distributed extreme programming.” In Don Wells and Laurie A. Williams, editors, XP/Agile Universe 2002, Second XP Universe and First Agile Universe Conference Chicago, IL, USA, August 4-7, 2002, ProceedingsXP/Agile Universe, volume 2418 of Lecture Notes in Computer Science, pages 13–22. Springer, 2002.

[15] PATTERSON A., KÖLLING M., and ROSENBERG J. “Introducing Unit Testing with Bluej.” In Proceedings of the 8th conference on Information Technology in Computer Science Education (ITiCSE 2003), 2003.

[16] RICHARDSON T., STAFFORD-FRASER Q., WOOD K. R., and HOPPER A. “Virtual Network Computing.” IEEE Internet Computing 2,1 (1998), 33-38.

[17] SKAF-MOLLI H., MOLLI P., OSTER G., GODART C., RAY P., and RABHI F. “Toxic farm: A cooperative management platform for virtual teams and enterprises.” In 5th International Conference on Enterprise Information Systems ICEIS03. Angers, France, April 2003.

[18] STOTTS D., WILLIAMS L. et AL. “Virtual teaming: Experiments and Experiences with Distributed Pair Programming”, tech. Rep. TR03-033, Department in Computer Science, University of North Carolina, Chapell Hill, NC 27695, USA.

[19] SUCCI G. and MARCHESI M. “Extreme Programming Examined.” The XP Series. Addison Wesley Publishing Company, 2001.

6.2 Biographie

Ibrahim LOKPO est enseignant-chercheur en Informatique à l’Institut National Polytechnique Félix Houphouët-Boigny de Yamoussoukro (Côte d’Ivoire). Ses centres d’intérêt sont entre autres les systèmes d’exploitation, les systèmes répartis et les applications de travail coopératif réparti.

Michel BABRI est enseignant-chercheur en Informatique à l’Institut National Polytechnique Félix Houphouët-Boigny de Yamoussoukro (Côte d’Ivoire). Ses centres d’intérêt sont l’algorithmique, les systèmes d’exploitation et les approches de travail coopératif appliqués à la programmation.

Gérard PADIOU est enseignant à l'ENSEEIHT et chercheur à l'IRIT. Ses centres d'intérêts sont l'algorithmique répartie et la vérification de programmes.