GESTION DE L'AUTOMATISATION DES TESTS DES SYSTÉMES ERP EN UTILISANT DES OUTILS COREJET

Tetiana KUSHCHYNSKA

Université nationale d'économie de Kharkiv Simon Kuznets, Ukraine, Université Lumière Lyon 2, France,
e-mail : tetianakushchynska@gmail.com

Dans cet article on a décrit le processus de l'automatisation de tests des système ERP. Pour le développement efficace de logiciels tels que les systèmes ERP contrôle de la qualité est une étape nécessaire. Le volume de test du système ERP est très grand, par conséquent il existe un besoin d'automatiser le test. On a développé ainsi une stratégie générale des tests, on a mis en place l'ensemble de processus de test, on a développé le framework pour les tests automatisés utilisant des outils CoreJet.

Mots clés : *système ERP, automatisation de test, framework de test, intégration continue, reporting des tests.*

1. Introduction

La phase de test est une étape importante dans le développement de logiciels, les logiciels tels que les systèmes ERP. La phase de test est comme une partie intégrante du développement, ainsi que la conception. Le manque de contrôle de la qualité est la cause de la mauvaise qualité du produit final, les plaintes des clients, sérieux manque de grands clients. Tenant compte de l'importance du dépistage, de nombreuses organisations informatiques introduisent une phase de test dans un cycle de développement logiciel. L'organisation de la procédure de test pour un projet existant est un problème complexe et à multiples facettes : le processus d'organisation de test comprend l'élaboration d'un plan de test, le choix des outils pour les tests, la décision sur la nécessité de mettre en œuvre les tests automatisés. Le but de l'introduction de test est d'améliorer la qualité du projet. Pour parvenir cet objectif, il est nécessaire de déterminer la stratégie de test, le plan de test et mettre en œuvre une analyse des résultats.

2. Etude sur les méthodologies et les technologies du processus des tests

2.1. Organisation générale des processus de tests

Le Progiciel de Gestion Intégré (ERP) est une application informatique qui permet de centraliser et de gérer l'ensemble des processus d'une entreprise en intégrant plusieurs fonctions dans un même système : ressources humaines, gestion comptable et financière, aide à la décision, vente, distribution, production, approvisionnement, commerce électronique, etc [3]. Le progiciel de gestion intégré offre une plus grande efficacité opérationnelle, une productivité accrue, une réduction des coûts de gestion de l'information et un meilleur contrôle des fonctions d'arrière guichet.

L'ERP est continuellement en changement que ce soit pour des raisons réglementaires, organisationnelles, d'amélioration de la compétitivité, d'évolution des produits proposés ou pour la modernisation du ERP à la recherche d'une meilleure efficacité [2].

Le couplage des fonctionnalités et des applications et l'accélération de la fréquence des changements dans un ERP intégré augmentent le besoin de tester et retester le ERP considéré comme un système global. Les objectifs du test consistent à détecter les défauts et les manques de conformité, mais aussi à vérifier le bon fonctionnement du système dans son ensemble. Alors que les tests étaient vécus comme un mal nécessaire, la complexification des systèmes les transforme en moyen pour maîtriser les coûts et la qualité des livraisons. D'ailleurs, les métiers du test deviennent spécifiques. Ils consistent à concevoir un système de test permettant de capitaliser et d'optimiser les tests à effectuer pour chaque livraison. De même, ils permettent de réduire les risques et de s'assurer que la mise en production des évolutions d'ERP ne causera pas de dommage critique pour le bon fonctionnement des métiers.

Dans le cycle de développement – intégration – mise en production d'ERP [1], une rupture subsiste dans le processus de tests entre organisation de développement. Ceci renforce le besoin d'homogénéiser les processus et de contrôler la qualité des logiciels livrés. C'est pourquoi il s'avère nécessaire d'aligner les activités de test dans un processus partagé, et de disposer d'applications stabilisées lors de la livraison. Les tests de bout en bout représentent l'ultime barrière de protection pour maîtriser les processus métier et sécuriser la mise en service et la maintenance de ces processus (nouvelles offres, procédures métier, canal de distribution).

Avec l'augmentation du besoin de tests à chaque évolution, il est aussi nécessaire d'améliorer la productivité

des tests :

1. Les tests devenant une activité récurrente, la capacité de construire un patrimoine de tests est un vrai facteur de productivité pour la maîtrise des projets et de la couverture de test des activités d'intégration et de mise en production d'ERP. La réutilisation des tests d'une étape à l'autre ou d'une version à une autre permet un gain sur la préparation des tests et sur la maîtrise des risques afin de maximiser la couverture de test face aux aléas des projets.

2. De même, l'automatisation des tests permet un gain de productivité et de temps sur les tests de non-régression. Cette capacité devient essentielle alors que les modifications et évolutions d'ERP [1] s'accélèrent pour s'adapter aux besoins de l'entreprise face à un marché de plus en plus concurrentiel.

Le processus de test proposé s'appuie sur 5 principes clés structurant sa mise en oeuvre :

1. L'analyse des processus métier critiques et leur modélisation constituent le point d'entrée pour la production des tests. Il s'agit de faire l'inventaire des processus métiers qui doivent faire l'objet du test. La modélisation permet de caractériser précisément les processus concernés et de capitaliser sur leur modélisation pour la génération des tests.

2. L'analyse de risques rend possible la hiérarchisation de ces processus en termes de besoins de test (nature et couverture de test). Cette analyse de risque sur les processus recensés permet de qualifier l'impact potentiel d'une anomalie (donc d'un dysfonctionnement du système) pour l'organisation, et la probabilité d'occurrence évaluée sur la base de l'expérience. Cette phase d'analyse de risque est essentielle pour piloter l'effort de test.

3. La génération automatique de tests à partir de modèles permet de systématiser et accélérer la conception et la maintenance des tests. Les technologies de génération de tests ont émergé ces dernières années pour accélérer et systématiser la production du référentiel de tests. Ces techniques et l'outillage associé permettent de produire les tests fonctionnels complets, pour les tests applicatifs.

4. Les tests sont gérés dans un référentiel de tests afin de suivre les campagnes d'exécution et de monitorer les anomalies détectées. La gestion des campagnes de test, de la traçabilité bidirectionnelle entre les tests et les exigences et le suivi des anomalies constituent des points essentiels pour la maîtrise des tests.

5. L'automatisation de l'exécution des tests est réalisée pour le test de non-régression en fonction des évolutions prévues de la chaîne applicative et de la criticité des processus métier. La répétition des exécutions manuelles des tests engendre des coûts en terme d'effort de test et des risques de perte d'attention du testeur lié à la répétitivité des tests. L'automatisation, accélérée par les techniques de génération de scripts, permet de pallier ces problèmes.

La figure 1 propose une illustration du processus de production et de maintenance des tests basé sur les principes qui viennent d'être énoncés.

Ce processus outillé de production des tests s'appuie sur les 5 activités principales suivantes :

1. L'inventaire des processus métier, leur modélisation et l'analyse de risque associé. Cette activité est réalisée par l'analyste métier.

2. La modélisation comportementale associée aux processus métier servant de base aux tests. Elle permettra de générer des tests complets et exécutables, incluant pour chaque pas du test les paramètres d'entrée et les valeurs attendues en sorties.

3. La génération des tests et leur publication au sein du référentiel de test. Cette activité est automatisée par l'outil de génération de tests.

4. Pour les tests devant être automatisés, l'automatisation des mots d'actions associés aux tests retenus.

5. La gestion des campagnes de test, leur exécution et le reporting des anomalies détectées.

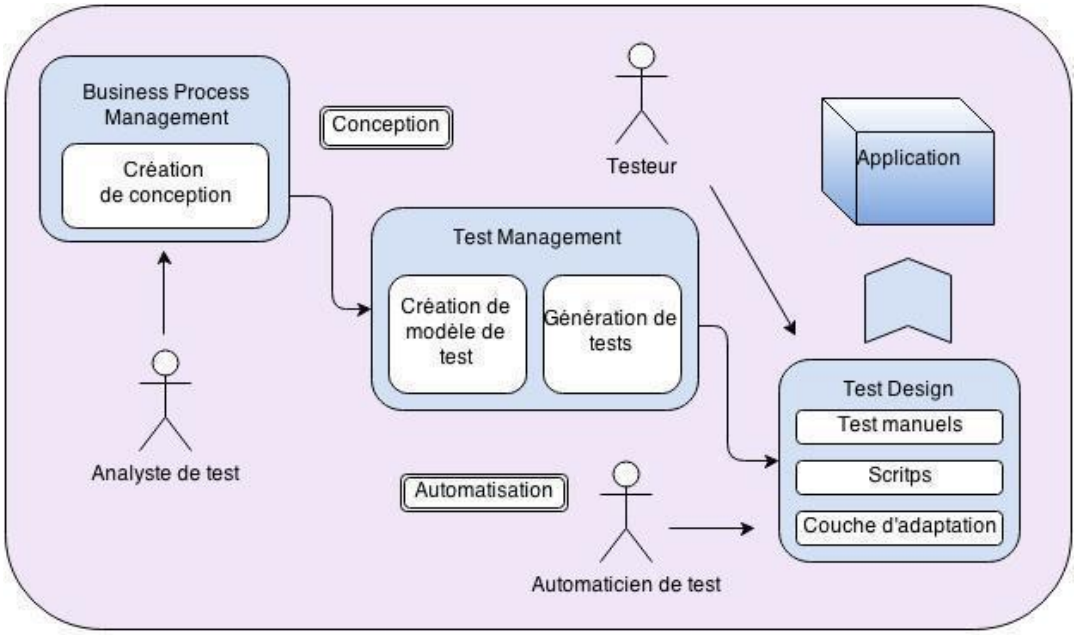

Figure 1. Processus de production des tests

Pour concevoir un test, plusieurs étapes sont nécessaires :

1. Imaginer un scénario de simulation conforme au mode d'utilisation et au fonctionnement du système défini dans les cahiers de charges et spécifications du système.

2. Définir les entrées qui vont faire réagir le système en fonction du scénario.

3. Identifier les sorties et les comportements attendus du système.

4. Lors de l'exécution du test, il faudra prévoir les moyens de vérifier le résultat par rapport au résultat attendu afin d'établir la conformité du test ou au contraire d'identifier le défaut révélé par le test.

La figure 2 propose un schéma de principe de préparation des tests.

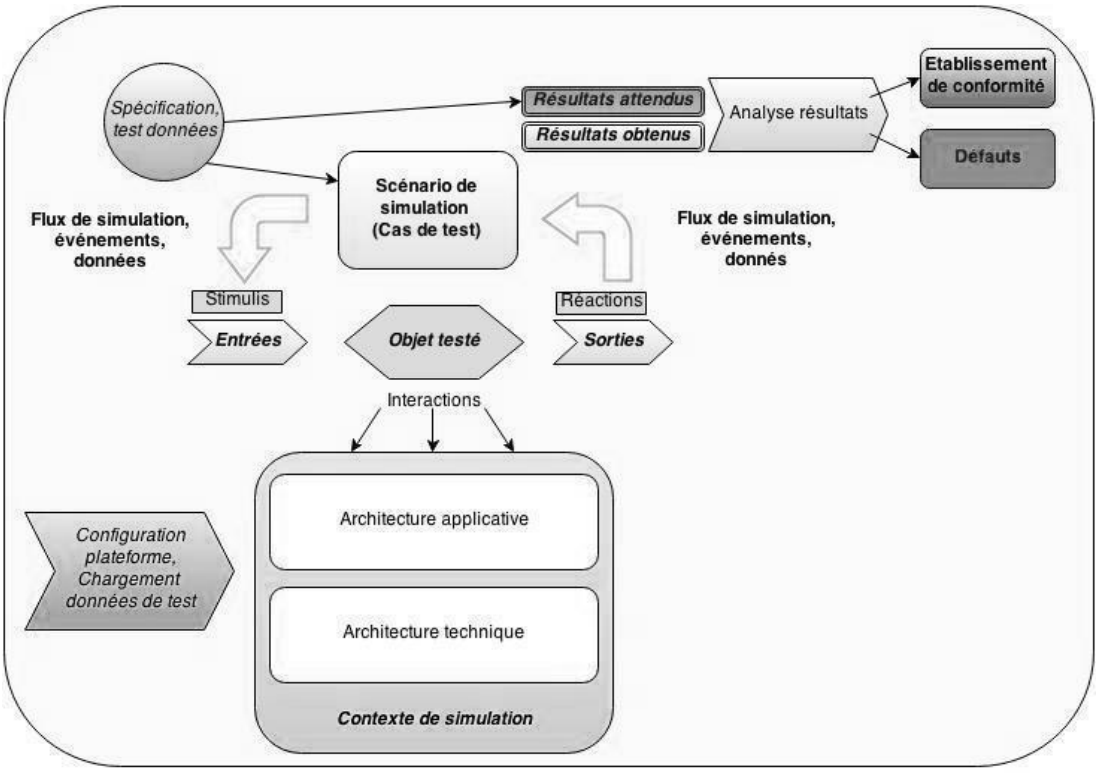

Figure 2. Schéma de principe de préparation des tests

Avant de pouvoir tester, il faut constituer le système de test et définir un contexte de simulation cohérent qui

permettra le fonctionnement de l'objet testé :

1. Ce système de test est constitué de la configuration des applications avoisinantes et des bases de données associées, ainsi qu'un certain nombre de moyens de test permettant d'injecter des données, d'observer les réactions du système ou de simuler des parties non présentes du contexte de test (bouchon). Il faut aussi prévoir si besoin des moyens pour simuler les chronologies et les traitements de masse périodiques du système.

2. Les données du contexte de test devront être définies de manière cohérente avec les besoins des scénarios qui seront exécutés au cours d'une campagne de test. Pour un test, le jeu de données est constitué des entrées du test, des données du contexte de tests nécessaires et les données attendues qui permettront de statuer sur le résultat du test.

Les activités d'un processus de test sont décrites dans le schéma suivant (Figure 3), elle se focalise sur les activités opérationnelles de tests.

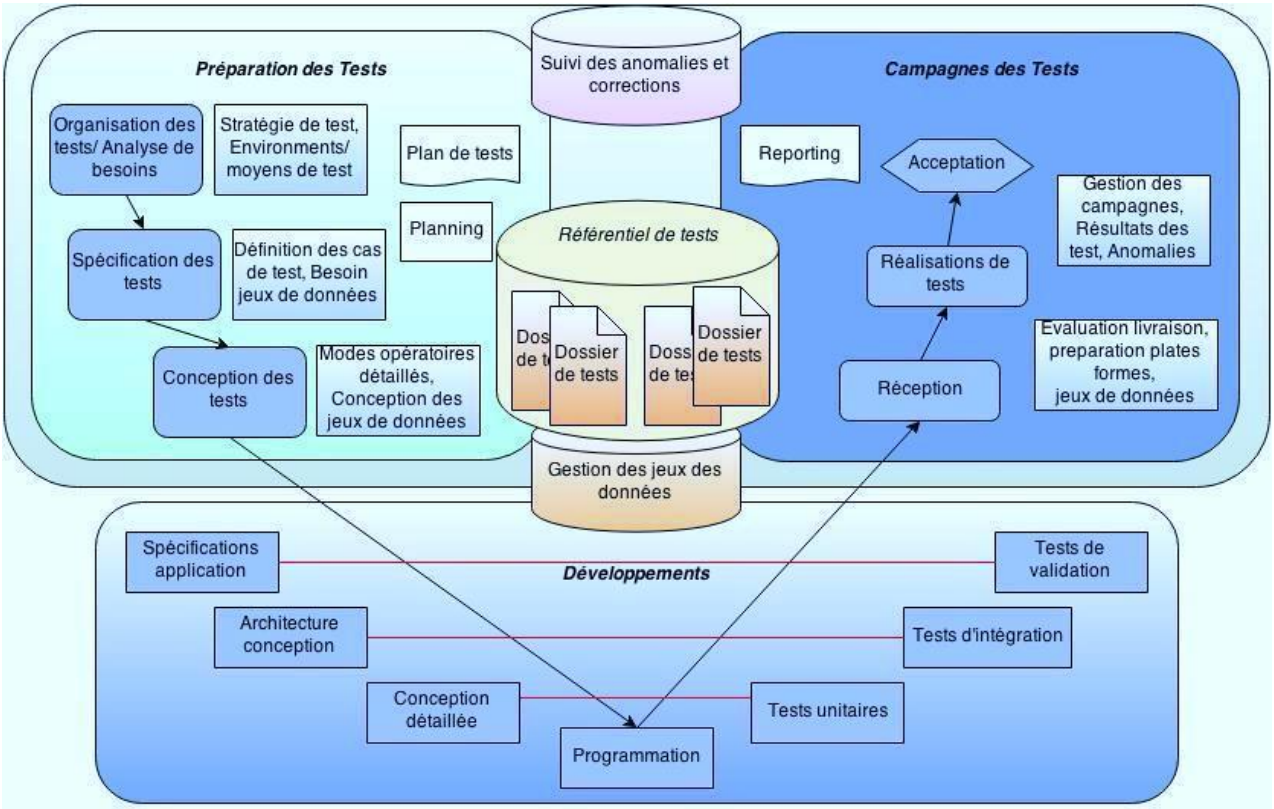

Figure 3. Activités des tests

Le principe consiste à préparer les tests dès que possible pour mettre en place une stratégie de tests qui garantira la meilleure couverture de test possible en fonction des contraintes du projet (budget, délais, moyens). Et cette stratégie permettra de s'adapter au mieux aux événements survenant en cours de projet (changement de spécification, de périmètre, dérive de planning).

Cette approche intègre les problématiques d'industrialisation des tests afin de tirer profit des solutions de test du marché. La méthode décrit une démarche systématique pour dériver à partir des objectifs de tests, la description des tests à effectuer et des jeux de données de test associés :

1. Une particularité de l'approche est de prendre en charge les problématiques de gestion des jeux de données qui sont essentielles à la maîtrise des activités de test.

2. Une autre particularité tient dans la différentiation pour la préparation des tests d'une étape de spécification des tests et d'une étape de conception des tests, qui permet d'anticiper au plus tôt la prise en charge des tests et la revue des spécifications alors que tous les documents ne sont pas disponibles. La spécification des tests permet aussi une meilleure gestion de la couverture des tests en fonction de la stratégie de test et de la maîtrise des risques choisie.

Le processus de test détaille les activités de prise en charge d'un projet de tests :

1. Définition de la stratégie de test pour prendre en charge les enjeux et risques applicatifs et métier, planification du projet de test, processus de maîtrise des risques du projet.

2. Préparation des tests avec une différentiation des étapes de spécification des tests et de conception des tests avec la gestion des jeux de données associés.

3. La démarche de constitution d'un référentiel de tests et de dérivation des tests en fonction des enjeux et contraintes du projet. Le référentiel de tests pourra être réutilisé pour les phases ultérieures du projet ou pour de nouvelles versions des applications.

4. La gestion des campagnes de test.

5. Et le suivi des anomalies et des livraisons.

Les tests sont essentiels et permettent de mettre en évidence la valeur métier des évolutions d'ERP ainsi que de sécuriser les mises en service de nouveaux produits ou de changement de processus de l'entreprise. Cependant, les entreprises soient très prometteuses et apportent une étape supplémentaire dans l'industrialisation des tests.

2.2. Intégration du système des tests

La qualité du logiciel est devenu un enjeu majeur de tout développement informatique. Si aujourd'hui des normes et des bonnes pratiques existent, elles ne sont pas toujours connues par les testeurs, ni par leur encadrement. Le test n'est pas limité à l'exécution du logiciel dans le but d'identifier des défaillances : il est aussi nécessaire de planifier, définir des objectifs, concevoir des conditions de tests, prévoir des données de tests, des critères de début et d'arrêt, des environnements de tests et bien sûr de contrôler tout cela.

Pour l'intégration du système des tests on a choisi une méthodologie Behavior driven testing (BDT). L'objectif du BDT est une lisibilité et un langage spécifique qui vous permet de décrire un comportement d'un système sans mettre de détails sur la façon dont le comportement est développé. Dans BDT, des tests sous forme de texte simple proposent des descriptions des scénarios généralement écrites avant toute autre chose et contrôlées par les actionnaires non-techniques.

Le framework de test pour ce projet a été élaboré comme suit (voir figure 4 ci-dessous) :

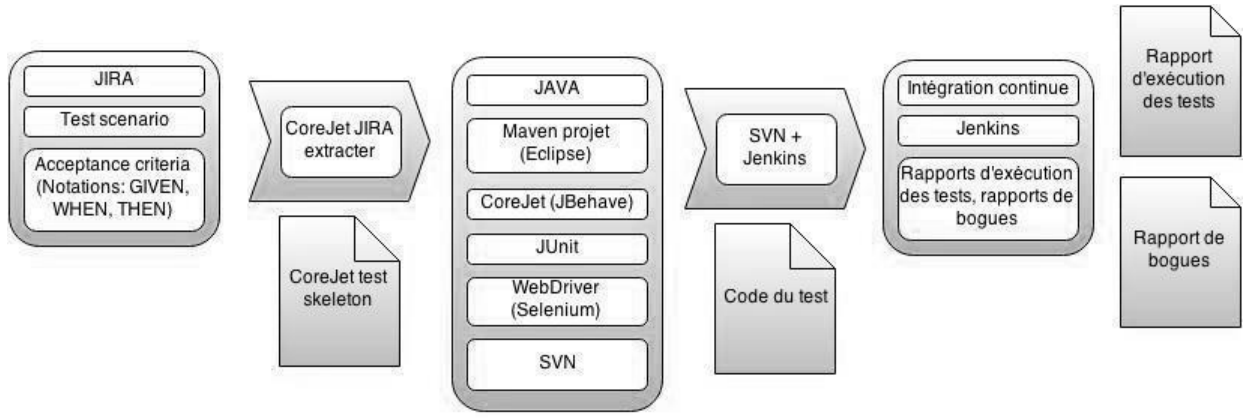

Figure 4. Structure du framework de test

Organisation d'un framework de test est une tâche complexe et multiforme. Ce procédé se compose de plusieurs étapes ; dans la première étape on doit décider comment stocker des scénarios de test. Dans le projet le système JIRA a été utilisé actuellement – pour l'organisation du développement du projet Fitnet Manager dans l'ensemble. Pour la mise en œuvre des tests il est configuré comme suit :

1. Ajout du type de l'issue «test» et «epic» : dans l'issue avec type «epic» on ajoute test suite ; dans l'issue avec type «test» on ajoute scénario de test ; les issues avec type «test» sont combinés dans les «epics».

2. Ajout des champs «Acceptance Criteria» et «Story Points» : dans le champ «Acceptance Criteria» on doit écrire un scénario de test selon les annotations de la langue Gherkin ; dans le champ «Story Points» on doit mettre nombre correspondant à la complexité du scénario de test.

3.Création un filtre personnalisé pour les issues avec les scénarios de test.

Après la création d'un scénario de test dans JIRA il faut créer CoreJet test skeleton pour la mise en œuvre de test. Il est créé à l'aide de plug-in CoreJet Jira extracter (qui est installé dans le navigateur Google Chrome). Plug-in CoreJet Jira extracter permet de créer CoreJet test skeleton dans deux langues : Python et Java. On a créé CoreJet test skeleton en Java et ensuite on commence la réalisation de test dans Eclipse (outil de développement Java).

La prochaine étape de développement de test est une implémentation directe de code de test. Le langage de programmation utilisé pour développer des tests est JAVA. Pour créer un projet avec un framework de test on utilise l'outil de développement Eclipse. Type de projet est Maven. Apache Maven est un outil pour la gestion et

l'automatisation de production des projets logiciels Java en général et Java EE en particulier. Maven utilise un paradigme connu sous le nom de Project Object Model (POM) afin de décrire un projet logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production. Il est livré avec un grand nombre de tâches pré-définies, comme la compilation de code Java ou encore sa modularisation. Comme système de gestion de versions on a utilisé SVN. Subversion (en abrégé SVN) est un logiciel de gestion de versions, distribué sous licence Apache et BSD. Puisqu'il aide les sources à converger vers la même destination, on dira que SVN fait la gestion concurrente de versions ou de la gestion de versions concurrentes. Il peut aussi bien fonctionner en mode ligne de commande qu'à travers une interface graphique. Il se compose de modules clients et d'un ou plusieurs modules serveur pour les zones d'échanges.

Pour l'organisation de l'automatisation des tests comme un tout on a été choisi méthodologie BDT. JBehave est optimisée pour les BDT, peut exécuter des spécifications fonctionnelles simple texte comme des tests automatisés [6]. Aujourd'hui, CoreJet outil de test d'automatisation est de plus populaire dans le marché en raison de leurs capacités d'automatisation pour les applications web. L'utilisateur peut écrire des tests ou des caractéristiques au-dessus du cadre de BDT avec l'aide de JBehave. Framework CoreJet permet de mettre en œuvre le test selon les BDD scénarios ; pour mettre en œuvre directement les tests eux-mêmes on a a utilisé des outils tels que Selenium WebDriver et JUnit. Selenium WebDriver est une API, disponible pour plusieurs langages, permettant de programmer des actions sur l'interface, et à vérifier les réponses [4]. JUnit est un framework de test unitaire pour le langage de programmation Java. JUnit est intégré par défaut dans les environnements de développement intégré Java tels que BlueJ, Eclipse et NetBeans [5].

La prochaine étape de développement est l'organisation d'intégration continue : l'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée. Utilisation d'outils SVN et Jenkins on permet d'organiser l'intégration continue de haute qualité et rapide [8] :

le code source soit partagé (en utilisant des logiciels de gestion de versions tels que SVN) ;

les développeurs intègrent (commit) quotidiennement (au moins) leurs modifications ;

les tests d'intégration soient développés pour valider l'application (JUnit).

Ensuite, il faut un outil d'intégration continue tel que Jenkins (anciennement Hudson) pour le langage Java. Les principaux avantages d'une telle technique de développement sont :

le test immédiat des unités modifiées ;

la prévention rapide en cas de code incompatible ou manquant ;

les problèmes d'intégration sont détectés et réparés de façon continue, évitant les problèmes de dernière minute ;

une version est toujours disponible pour un test, une démonstration ou une distribution.

Le résultat est un rapport qualitatif à l'exécution des tests (HTML-report) : ce rapport fournit des informations sur le temps, le succès de l'exécution des tests, l'inscription des raisons de l'échec des tests, l'information sur les défauts. Selon les rapports nous tenons des statistiques qui montrent l'évolution de la qualité du produit logiciel. Les résultats des tests permettent de déterminer rapidement les effets indésirables, qui réduit les coûts pour corriger ces défauts. Une bonne organisation des tests peut réduire le coût d'élimination des défauts : le plus tôt un défaut est détecté, moins il en coûte de son élimination.

3. Implémentation des tests automatisés

3.1. Configuration de l'environnement de test

Automatisation des tests représente l'utilisation de logiciels pour exécuter ou supporter des activités de tests, p.ex. gestion des tests, conception des tests, exécution des tests ou vérification des résultats. Dans le cadre du développement d'une application, quelle qu'elle soit, les tests sont indispensables, et prennent une part non négligeable du développement. Il en existe plusieurs types : unitaires, intégration, fonctionnels, qualification, etc. Aujourd'hui, la plupart sont automatisés, ce qui permet un gain de temps substantiel, ainsi qu'une plus grande fiabilité.

Le test manuel est effectué par une personne, vérifiant attentivement les applications, essayant des combinaisons variées de l'usage et de l'input, comparant les résultats aux attentes et rédigeant des observations. Un outil de test automatisé est capable de refaire des tests pré-enregistrés et prédéfinis, comparer les résultats avec les comportements attendus et signaler le succès ou l'échec des tests. Une fois crées, les tests automatisés peuvent être facilement répétés et ils peuvent être étendus afin d'exécuter des tâches impossibles avec le test manuel. Donc, l'automatisation d'un test n'a de sens que si le test répond à un certain nombre de critères :

1. Le test est systématique : il doit être exécuté à chaque nouvelle version de l'application.

2. Le test est répétitif : il est présent dans de nombreux scénarios de test.

3. Le test est automatisable : il est possible techniquement de faire jouer le test par un automate.

L'automatisation des tests logiciel économise du temps et de l'argent. Les tests logiciel doivent être répétés souvent lors du cycle de développement afin d'assurer la qualité du produit. Chaque fois que le code source est changé, des tests logiciel devraient être répétés. Faire ceci manuellement prend plus de temps est rend l'activité de test plus chère.

L'automatisation des tests permet d'augmenter la profondeur et le périmètre des tests afin d'améliorer la qualité logiciel. Les processus de tests longs, qui sont souvent évités pendant le test manuel, peuvent être exécutés sans surveillance. Ils peuvent même être exécutés sur plusieurs ordinateurs avec des configurations différentes. Les tests automatisés peuvent exécuter des milliers des cas de tests complexes et différents pendant chaque exécution, fournissant une couverture impossible d'obtenir avec des tests manuels.

Suivant l'organisation de la société, l'automatisation est confiée à un ingénieur de test fonctionnel ayant des compétences informatiques, notamment en scripting. Le principal sera que la personne pense à la maintenance des scripts, c'est à dire avoir des scripts réutilisables par composant et avec un certain niveau d'abstraction.

La mise en place d'un framework de test remédie au principal facteur d'échec des projets d'automatisation, à savoir la maintenabilité des scripts. Un framework de test est un ensemble d'hypothèses, de concepts et d'outils qui supporteront l'automatisation des tests. Pour implémenter le framework de test on a utilisé de nombreux outils différents pour les différentes étapes d'implémentation. Pour implémenter la partie du programmation on a utilisé les outils tels que Selenium WebDriver, JBehave et CoreJet.

Selenium est une suite d'outils permettant de faire des tests fonctionnels d'une application web (et uniquement web). Selenium est un outil qui fonctionne sur de nombreuses plateformes de développement Java, .Net. Selenium WebDriver est propre, rapide framework pour les tests automatisés d'applications Web. Selenium un framework de test populaire et bien établi est un merveilleux outil qui fournit une interface pratique unifiée qui fonctionne avec un grand nombre de navigateurs. Caractéristiques de Selenium WebDriver [4] :

Selenium WebDriver supporte plusieurs langues par exemple Java, Python, C #, Ruby, Perl, PHP, Java script.

Pas besoin de démarrer le serveur de sélénium.

Selenium WebDriver permet de tester l'iPhone et Android.

Vous pouvez trouver les coordonnées d'un objet à l'aide WebDriver.

Vous pouvez facilement simuler le clic sur le devant et sur le bouton retour de navigateur.

WebDriver utilise l'automatisation natif et ne pas avoir les contraintes de sandbox de Selenium-RC. Il est un peu plus rapide et ne nécessite pas un serveur.

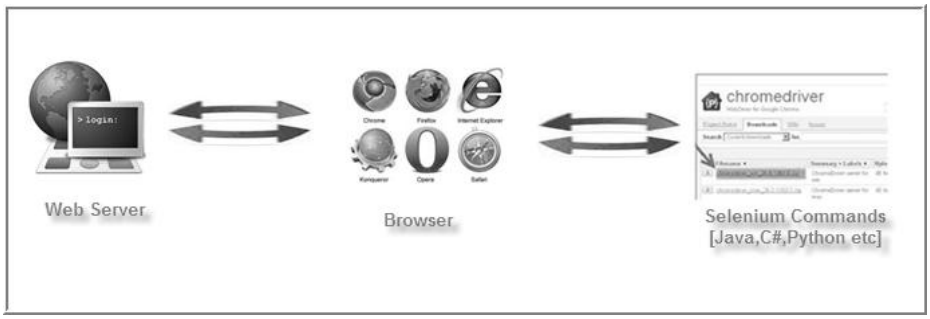

Figure 5. Système du fonctionnement de Selenium WebDriver

3.2. Framework Jbehave

JBehave est un framework pour le développement du comportement-Driven (BDD). BDD est une évolution du développement piloté par les tests (TDD) et la conception d'acceptation-piloté par les tests, et vise à rendre ces pratiques plus accessible et intuitive pour les nouveaux arrivants et les experts. Il déplace le vocabulaire de l'être au axées sur le comportement, et les positions en fonction de test lui-même comme une philosophie de conception [6].

JBehave soutient la rédaction du cahier dans environ 30 langues parlées. JBehave travaille avec Ruby, Java, NET, Flex ou des applications web écrites dans n'importe quel langage.

JBehave encourage à commencer à définir les histoires via des scénarios qui expriment le comportement désiré dans un format textuel. Le scénario textuelle devrait utiliser la langue du domaine des affaires et protéger l'écart autant que possible les détails de la mise en œuvre technique. En outre, il devrait être donné un nom qui est l'expression de la fonctionnalité qui est en cours de vérification.

Le framework JBehave utilise langage Gherkin [6] pour définir les scénarios des tests. Le scenario est une succession d'étapes (Step) permettant :

soit de définir et de construire le contexte dans lequel le scénario va se dérouler Given ;

soit de provoquer des évènements ou des actions sollicitant le système When ;

soit de vérifier que le comportement attendu a bien eu lieu Then ; c'est généralement à ces étapes que l'on retrouvera les assertions.

3.3. Outil CoreJet

Pour l'étape de la création d'une implémentation logicielle de test, on utilise un outil CoreJet. CoreJet est un outil qui s'intègre parfaitement à l'outil de suivi de projet populaire JIRA [9] afin de mettre les principes de comportement conduit le développement en action [7].

Comment fonctionne cet outil :

1. Les comportements qui sont requis par l'entreprise sont répartis en Epics, Stories and Scenarios et stockées dans JIRA par Business Analysts.
2. Les scénarios sont écrits dans le format BDD : Given, When, Then.
3. Utiliser le plugin Google Chrome pour CoreJet, Stories and Scenarios convertis en squelettes Java.
4. Pour UI tests, chaque Story/Scenario a un certain nombre des Page Objects à l'aide de Selenium. 2.0 qui permettent à l'appareil de contrôle d'interagir avec l'application.
5. JUnit Runner personnalisé est utilisé pour exécuter des stories que les tests.
6. La sortie est sous la forme de fichiers JUnit et CoreJet XML.
7. CoreJet XML file se traduit par un rapport interactif HTML5 Report et affiché dans le navigateur.

Pour la mise en œuvre réussie et l'utilisation de l'outil CoreJet on a besoin des éléments suivants [7] :

Une instance JIRA ;

Google Chrome ;

Plugin Google Chrome pour CoreJet ;

Selenium 2.0 ;

Jenkins ;

Rapports CoreJet ;

Un groupe de Business Analysts fraîches ;

Quelques développeurs mûres.

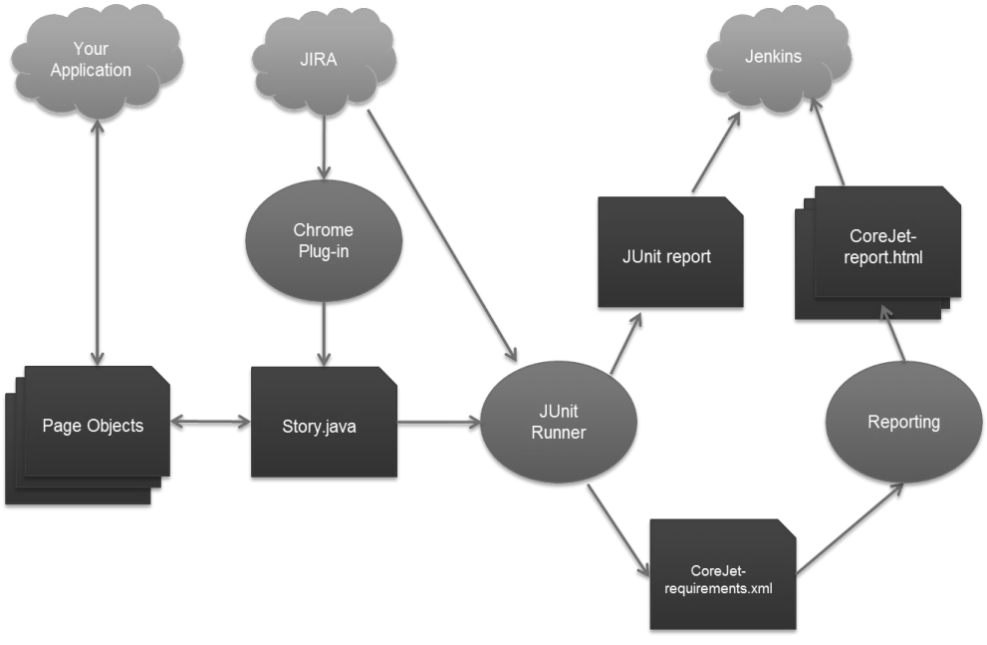

Figure 6. Schéma de fonctionnement de l'outil CoreJet

Comment fonctionne CoreJet, quelles sont les parties à inclure et comment ils se rapportent : application qui est testé est décrit par Page Objects (framework Java, Eclipse, Maven) ; scénarios de test sont décrites vers tickets JIRA (type «issue» - test) en format de langage Gherkin ; par l'intermédiaire plugin Google Chrome pour CoreJet on a crée squelettes de test (Java) ; test est réalisé en utilisant Page Objects (base - squelette généré par plugin) ; JUnit Runner permet d'exécuter des tests ; après le test on obtient JUnit report et CoreJet requirements (requirements sont utilisés pour générer CoreJet report) ; les tests sont exécutés à l'aide Jenkins ; après l'exécution des tests on peut voir les résultats des tests sous forme de HTML-report.

Simplicité, accessibilité, possibilité de modifier, facilité d'utilisation sont des avantages importants d'outil CoreJet qui le distinguent parmi d'autres outils. L'intégration avec des systèmes comme JIRA et Jenkins permet de mettre en œuvre framework de test de haute qualité, qui peuvent utiliser des groupes différentes d'utilisateurs, qui peut être facilement étendu.

3.4. L' intégration continue d'un projet avec Jenkins

L'intégration continue est un ensemble de bonnes pratiques utilisées en génie logiciel. Ces bonnes pratiques visent à vérifier qu'une modification de code source n'entraîne pas de régression de l'application en cours de développement. Cette vérification est en générale effectuée sur une autre machine que la machine de développement (serveur d'intégration) ; et cette vérification est effectuée assez fréquemment, d'où le nom d'Intégration Continue.

On a choisi d'intégrer l'outil d'intégration continue Hudson/Jenkins, qui est l'un des meilleurs outils actuellement. Jenkins (anciennement Hudson) est un serveur Open Source d'intégration continue permettant d'automatiser les tâches répétitives comme le build, la génération de documentation, les déploiements [8]. Pour l'organisation l'IC trois jobs avec les tests ont été créés : TestFitnet, TestFitnetSequence et SuppressionTest.

Figure 7. Outil Jenkins (jobs avec les tests)

A la fin de l'exécution des jobs on évalue les résultats et l'analyse des statistiques. Selon les résultats, nous pouvons juger de la qualité du produit à cette étape. L'analyse des statistiques révèle des tendances dans la qualité des produits – l'amélioration ou l'aggravation.

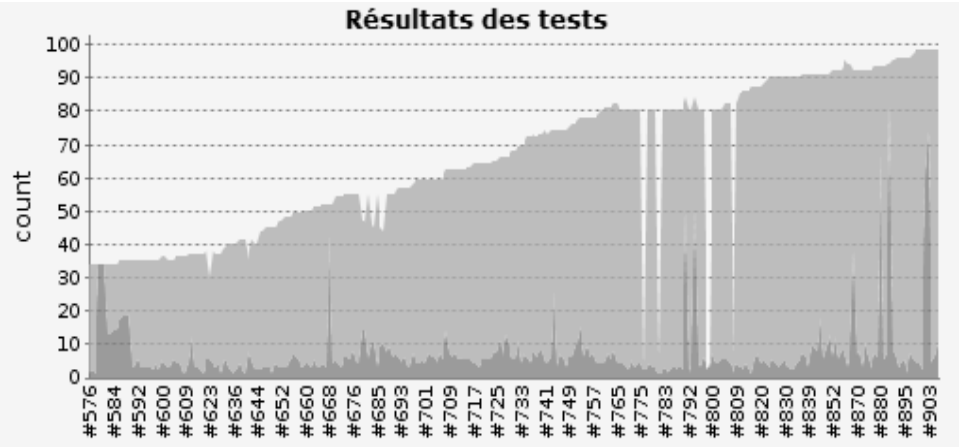

Figure 8. Outil Jenkins (Statistiques sur job TestFitnet)

L'utilisation de l'intégration continue permet d'accélérer le processus de tests de régression, de libérer des ressources humaines : ainsi le contrôle constant de la qualité actuelle du projet a été assuré.

3.5. Reporting des tests

Le reporting est une étape clé du test logiciel. Le test est une activité dont les résultats intéressent de nombreuses parties prenantes :

les testeurs, pour évaluer leurs propres prestations ;

les développeurs, pour évaluer la qualité de leurs développements et la charge de travail restante, que ce soit en termes de corrections à effectuer ou de développements à livrer ;

les responsables qualité pour déterminer les améliorations à entreprendre dans les processus, que ce soit lors des phases d'identification d'exigences, de revue, de conception ou de tests ;

les clients, les utilisateurs finaux ou le marketing, pour savoir quand le logiciel ou système sera disponible et commercialisable ;

la hiérarchie, afin d'évaluer les charges à prévoir et la rentabilité des activités ayant eu lieu jusqu'à ce jour.

Ces parties prenantes doivent être informées, par le biais de rapports d'avancement, de statistiques et de graphiques, pour répondre à leurs interrogations, et leur permettre de prendre des décisions en ayant les informations adéquates. Les activités de reporting se basent sur les données de suivi et de contrôle fournies par chacune des activités de tests.

Pour reporting des tests dans le projet Fitnet Manager nous avons utilisé CoreJet ingrédient - CoreJet Reports. CoreJet Reports nous ont permis dans une forme commode d'enregistrer les résultats des tests.

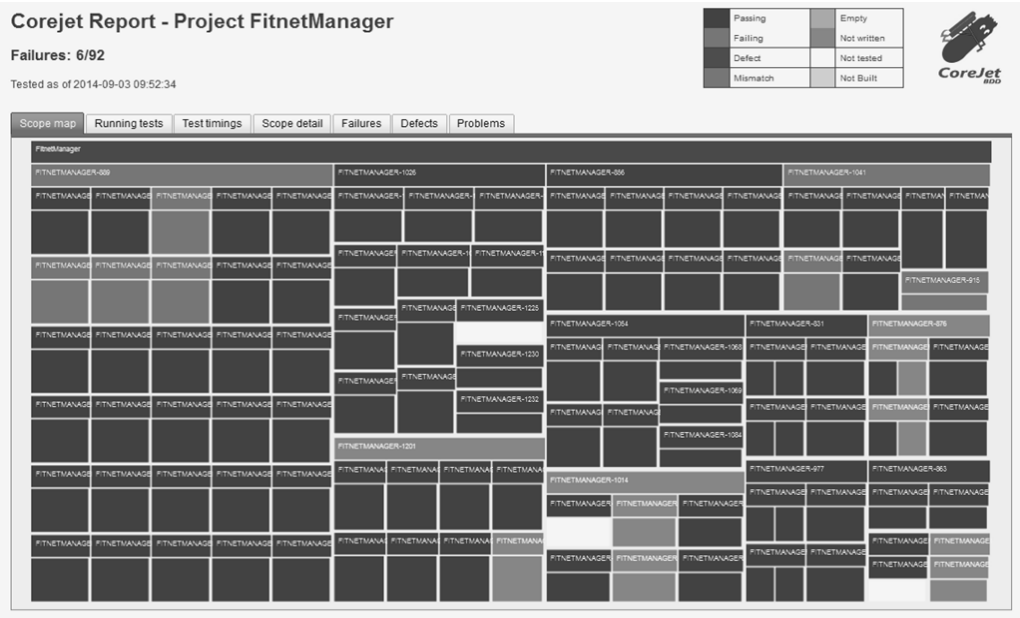

Figure 9. Outil Jenkins (Exemple du test rapport)

4. Conclusions

ERP système est un logiciel qui est requis de haute qualité. Pour assurer la qualité du logiciel on a besoin de test de la qualité au cours de la phase de développement. Compte tenu le volume de test il est nécessaire d'introduire une automatisation des tests.

L'organisation du processus de test et la mise en place de tests automatisés pour les applications Fitnet Manager permettra d'améliorer la qualité du produit logiciel - des tests approfondis de nouvelles fonctionnalités, les tests de régression (vérification du bon fonctionnement de la fonctionnalité existante), l'identification et l'élimination des défauts dans la phase de développement. Poursuite de l'expansion du framework de test permettra de réduire le nombre de défauts, maintenir la haute qualité du projet, ainsi que pour augmenter la stabilité du projet.

Références

1. ERP orienté gestion de projets et facturation par affaire [Ressource électronique]. – L'accès à la ressource : http ://www.fitnetmanager.com.

2. Blog Fitnet Manager [Ressource électronique]. – L'accès à la ressource : http ://fitnet-leblog.com/.

3. O'Leary D. Enterprise resource planning systems : systems, life cycle, electronic commerce, and risk // Cambridge University Press [Ressource électronique]. – L'accès à la ressource : http://assets.cambridge.org/052179/1529/sample/0521791529ws.pdf.

4. SeleniumHQ [Ressource électronique]. – L'accès à la ressource : http ://www.seleniumhq.org/.

5. JUnit. Project Documentation [Ressource électronique]. – L'accès à la ressource : http ://junit.org/.
6. JBehave [Ressource électronique]. – L'accès à la ressource : http ://jbehave.org/.
7. CoreJet. A Behaviour Driven Development Framework [Ressource électronique]. – L'accès à la ressource : http ://corejet.org//.
8. Jenkins [Ressource électronique]. – L'accès à la ressource : https ://wiki.jenkins-ci.org/display/JENKINS/Home.
9. JIRA [Ressource électronique]. – L'accès à la ressource : https ://www.atlassian.com/software/jira/.

Sous la supervision de (Під керівництвом) :

Jérôme Darmont (PhD, HdR, professeur, ERIC lab)

Сидоренко І.Г. (к. т. н., доцент кафедри інформаційних систем, с.н.с.)

Керівник з іноземної мови ст. викладач кафедри іноземних мов та перекладу Безугла І.В.