ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

RAPPORT DE PROJET PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
Maitrise en Génie Logiciel

PAR
Guypacome GBELAI

ÉTUDE DE CAS DU DÉPLOIEMENT D'UNE APPLICATION .NET (PACIQ) À L'HÔPITAL SAINTE-JUSTINE

MONTRÉAL, LE 17 AOÛT 2016

PRÉSENTATION DU JURY

CE RAPPORT DE PROJET A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Professeur Alain April, directeur de projet,
Département de génie logiciel et TI à l'École de technologie supérieure

Professeur Alain Abran, jury
Département de génie logiciel et TI à l'École de technologie supérieure

REMERCIEMENTS

Je tiens à remercier tout d'abord le professeur Alain April pour sa disponibilité et ses conseils pertinents qui m'ont permis de mener à bien ce projet.

J'aimerai aussi remercier M. Patrice Dion du département de génie logiciel et des TI de l'ÉTS, ainsi que les membres de l'équipe de l'hôpital Sainte-Justine, notamment Madame Isabelle Olivier et Monsieur Michel Lemay.

Je remercie aussi les étudiants : Ulrich GHOMSI KAMGUEM, Nicolas Brousseau et Moulay Youssef Tariq qui ont initié ce projet et qui ont répondu à mes questions.

Enfin, je remercie toute ma famille pour leur soutien et leur encouragement tout au long de mon parcours dans ma maitrise.

ÉTUDE DE CAS DU DEPLOIEMENT DE L'APPLICATION .NET PACIQ À L'HÔPITAL SAINTE-JUSTINE

GUYPACOME GBELAI

RÉSUMÉ

Le projet consistait à livrer le logiciel PACIQ au client, en occurrence l'hôpital Ste-Justine. Le logiciel PACIQ a été réalisé par une équipe d'étudiants en génie logiciel de l'école de technologie supérieure (ÉTS). Il est basé des technologies .Net et permettra le suivi des projets d'amélioration au sein de l'établissement hospitalier. Cette application étant déjà réalisée au tout début de ce projet de recherche, l'objectif de ce projet de recherche appliquée de 15 crédits, est de mettre en œuvre l'environnement pour les tests d'acceptation, les stratégies de test, la livraison et les moyens de déploiement de l'application PACIQ à l'hôpital Ste-Justine.

Ce rapport traite de la gestion de l'activité de tests dans un cycle de développement de logiciel. Il aborde les différents éléments que l'on retrouve dans les activités de tests. Le but du test d'acceptation est de vérifier les exigences logicielles dans le but d'atteindre un niveau élevé de confiance qu'il rencontre les besoins du client avant sa mise en service opérationnelle. Il couvre aussi le suivi des anomalies qui permet une meilleure gestion de cette étape en leur affectant une criticité.

Mots clés : mise en production, essais de réception, suivi des anomalies

ÉTUDE DE CAS DU DEPLOIEMENT DE L'APPLICATION .NET PACIQ À L'HÔPITAL SAINTE-JUSTINE

Guypacome Gbelai

ABSTRACT

The main goal of this project is to deliver the PACIQ software to the customer at Sainte-Justine hospital. This software was built by previous students from école de technologie supérieure (ÉTS). PACIQ will allow the tracking of quality improvement projects within the hospital to help with its certification efforts. PACIQ was developed using Microsoft .Net technologies. This application prototype was already built at the beginning of this 15 credits applied research project. The objective of this project is to set up a user acceptance environment to conduct the acceptance testing with the end users, prepare the production environment and its deployment on the hospital servers.

This report presents the acceptance test planning. Different aspects of the acceptance test activities will be presented. The final goal of the acceptance tests is to verify the functional software requirements to ensure they meet the customer requirement before the deployment. This report also covers the need to track anomalies, which allow for a better management of this process by identifying their criticality.

Keywords: transition to production, user acceptance testing, defect tracking

TABLE DES MATIÈRES

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

BI: Business Intelligence
CHUSJ : Centre hospitalier de l'Université de Montréal de Sainte-Justine
ÉTS : École de technologie supérieure
ISO : Organisation internationale de normalisation
KPI: Key Performance Indicators
PACIQ : Programme d'amélioration continue et intégratif de la qualité
SQL: Structured Query Language
SSAS: SQL Server Analysis Services
SSIS: SQL Server Integration Services
SSRS: SQL Server Reporting Services

INTRODUCTION

Dans le domaine du génie logiciel, les tests ont toujours eu une place importante. Dans le cycle de vie de développement des logiciels, les tests font partie intégrante des différentes phases. Aujourd'hui les logiciels sont présents partout autour de nous. Ils ont tous des rôles et des criticités différentes. Dès leur conception jusqu'à leur livraison, les logiciels peuvent contenir des défauts. Les tests servent à détecter le plus grand nombre de défauts et de les corriger avant la livraison finale du produit. Plus ils sont détectés tôt, dans le cycle de vie de développement, et moins ils auront un impact sur le re-travail, les coûts et les délais de livraison.

Dans ce contexte, l'objectif principal de ce projet de maîtrise appliquée, de 15 crédits, en génie logiciel est de planifier, mettre en place l'environnement et effectuer les essais d'acceptation du logiciel PACIQ, développé pour l'hôpital Sainte-Justine.

Coordonner les essais d'acceptation d'un logiciel nécessite aussi d'identifier et fournir de l'accompagnement à l'utilisateur final, tel que de l'aide à l'apprentissage du logiciel, de l'assistance pour régler des défauts et de coordonner la mise en production avec son service informatique. La phase des tests d'acceptation, effectuée avec l'aide de l'utilisatrice principale, est effectuée juste avant le déploiement de l'application en production. Une plateforme de gestion des problèmes a été mise en place afin de s'assurer d'une meilleure gestion des défauts et des demandes de changements (c.à-d. les billets de support/maintenance) pendant cette étape.

Ce rapport présente, dans le chapitre 1, le contexte du projet, ses enjeux ainsi que les objectifs de l'étape d'acceptation du projet PACIQ. Par la suite, le chapitre 2 décrit les activités de tests d'acceptation du prototype logiciel. Finalement, le dernier chapitre décrit l'expérimentation ainsi que les commentaires de l'utilisatrice ainsi que des recommandations pour des travaux futurs.

CHAPITRE 1

Mise en contexte

1.1 Contexte

Ce projet de recherche appliquée poursuit le travail de développement du prototype PACIQ des étudiants de l'ÉTS. Suite au développement, aux tests unitaires et aux tests de système, il est maintenant temps d'effectuer les tests d'acceptation et la mise en production du prototype logiciel afin qu'il soit utilisé pour la première fois par sa clientèle. En effet, l'étape actuelle vise à planifier et effectuer la conduite des tests d'acceptation incluant la mise en place d'une plateforme pour le suivi des défauts et des demandes de changements. Ce logiciel de suivi des défauts sera d'une grande utilité quand le logiciel sera en production et que les utilisateurs voudront rapporter des problèmes et des demandes de changement.

PACIQ est un prototype web qui a été conçue afin de faciliter la gestion du programme d'amélioration de la qualité de l'hôpital Sainte-Justine. Il est destiné à gérer un grand nombre de plans d'action concurremment. Cette solution commandée par la Direction Qualité Sécurité et Risques du CHU Sainte-Justine a été réalisée par des étudiants en génie logiciel de l'École de Technologie Supérieure (ÉTS). Le logiciel PACIQ, comporte deux volets technologiques qui ont chacun été réalisés en 2015 :

1. La conception d'une application Web permettant la gestion des plans d'amélioration ;
2. La conception d'un module analytique d'intelligence d'affaires.

Depuis la fin de la phase de réalisation du prototype logiciel, il s'est écoulé plusieurs mois avant la reprise du projet avec les clients. Cela est normal, car il n'y a pas toujours des étudiants disponibles pour y travailler. Il fallait donc rétablir le contact avec la représentante de l'hôpital Sainte-Justine et relancer les activités du projet PACIQ en vue de son acceptation finale et de sa mise en service. Pour ce faire, dans un premier temps, il s'agissait de comprendre et maitriser le prototype logiciel réalisé par les étudiants, qui maintenant avaient quitté l'ÉTS suite à la

réalisation du logiciel. L'implication des membres de la Direction Qualité Sécurité et Risques du CHU Sainte-Justine a été grandement appréciée dans cette premier phase de ce projet.

1.2 Enjeux

Les enjeux les plus importants identifiés pour ce projet sont :

- La collaboration avec l'utilisatrice principale à l'hôpital Sainte-Justine ;
- La gestion des attentes de la cliente ;
- La gestion du temps, des communications et des réunions ;
- La recherche et l'installation d'un logiciel libre pour la gestion des défauts qui doit être compatible avec l'environnement de tests à l'ÉTS ;
- La compréhension de l'architecture et des détails des deux parties du prototype PACIQ.

1.3 Objectifs

1.3.1 Étape de certification/acceptation technique de PACIQ

Cette première étape vise, avec l'aide d'un représentant du groupe informatique de l'hôpital Sainte-Justine, de planifier la future installation du prototype logiciel PACIQ sur les serveurs de l'hôpital Sainte-Justine afin que le représentant utilisateur puisse effectuer ses essais dans un environnement réel. Étant donné que ce sera le service informatique de l'hôpital Sainte-Justine qui va supporter l'utilisateurs quotidiennement, il est nécessaire de planifier que le processus de gestion des versions subséquentes, entre l'ÉTS et l'hôpital, est bien défini, testé et fonctionne bien. Il s'agit, dans cette étape, de planifier avec ces intervenants, la mise en place de l'environnement de production pour PACIQ et d'effectuer les essais de réception. Pour ce faire, il est nécessaire de concevoir, réviser et faire approuver un plan de test par cet intervenant. Ce plan de test contient :

- Les objectifs des tests (fonctionnels, opérationnels) ;
- Le calendrier détaillé des activités (avant et pendant les tests) ;
- La liste des éléments de configuration ;

- Une description du processus de test ;
- La préparation, par le représentant utilisateur, des cas de tests ;
- La mise en place du processus/logiciel des rapports de défauts et demandes de changements ;
- La gestion de la communication avec le représentant utilisateur et le représentant informatique ;
- La planification des réunions de pilotage des tests ;
- La certification de l'environnement de test.

1.3.2 Exécution du plan de test

Cette étape concerne l'exécution même des tests ainsi que le rapportage des anomalies dans le système de gestion de problèmes. Les tests, avant le déploiement, se feront dans un environnement technique situé à l'ÉTS :

- L'exécution des tests ;
- La création/traitement des rapports de défauts et demandes de changements ;
- La conclusion des tests ;
- La création de la version 1.0 pour futur déploiement en version béta ;
- S'assurer d'avoir un environnement de test, à l'ÉTS, avec l'image de la version 1.0 accompagnée de données de test.

1.4 Conclusion

Ce premier chapitre a présenté le contexte, les enjeux et les objectifs principaux de ce projet. Le prochain chapitre présente la revue de la littérature qui porte sur les tests en génie logiciel.

CHAPITRE 2

Revue littéraire

2.1 Introduction

Ce chapitre présente une synthèse concernant les tests de logiciel. Elle portera plus particulièrement sur les techniques et stratégies de tests dans le domaine du génie logiciel.

2.2 Les tests logiciels dans les hôpitaux

Existe-t-il des différences pour tester les logiciels liés au milieu hospitalier ? La réponse est surement qu'il y a des similarités avec les autres domaines et des différences propres au milieu hospitalier. D'une manière générale, il n'existe pas de grandes différences, car les logiciels de gestion en milieu hospitalier suivent mêmes principes de développement et de tests que les autres logiciels de gestion des autres domaines. Un aspect important de ce domaine est la notion de sécurité des données des patients. Dans certains cas, il y'a certaines précautions à prendre en compte pour certains logiciels destinés au milieu de la santé. En effet certains tests spéciaux sont effectués pour veiller à ce que les logiciels soient surs lors de leur utilisation sur les humains dans le cas de diagnostique ou de thérapie. Les tests effectués afin de mitiger les risques «Health Risk Mitigation test» *(IEC 62304 :2006)* pour la santé et la sécurité. Les risques sont les suivants :

- Augmentation de l'exposition aux radiations et chocs électriques ;
- Extrême contacts de température ;
- erreur de diagnostic clinique ;
- une exposition accrue à la puissance acoustique.

Le prototype logiciel PACIQ, que l'ÉTS et l'hôpital Sainte-Justine vont mettre en place, n'est pas un logiciel thérapeutique et ne permet pas non plus de poser un diagnostic patient. C'est un logiciel de gestion qui permet de faire le suivi de projets d'amélioration. À ce titre, ce logiciel, comme tous les autres logiciels du même type, se doit d'être sécure, de donner des informations exactes, un bon niveau de service et il doit répondre aux exigences de l'utilisatrice principale telles que spécifiées dans le document de vision approuvé préalablement.

2.3 Qu’est-ce que tester un logiciel ?

2.3.1 Définition

Les tests sont un ensemble d'activités menées pour faciliter la découverte et/ou l'évaluation des propriétés d'un ou plusieurs éléments de test *(IEEE 29119-1, 2013)*. Tester, c’est aussi exécuter le logiciel avec l’intention d’y détecter des anomalies ou des défauts *(Myers, Badgett, Sandler, 2012)*. Réaliser des tests sur un logiciel, consiste à valider et vérifier ce logiciel. Ce processus permet donc de valider que le logiciel testé répond non seulement à son document d’exigences, mais qu’il effectue bien les tâches qu’on s’attend de lui. Les tests permettent aussi de soulever des erreurs, défauts et des anomalies le plus tôt possible afin de les réparer avant sa mise en service.

2.3.2 Les objectifs des tests d’un logiciel

Les tests ont pour objectifs : 1) d’évaluer le logiciel dans son environnement réel ; et 2) de valider son comportement afin de réduire les risques d'apparition de défaillances suite à sa mise en service. Le processus nécessite de concevoir des scénarios de tests qui seront en mesure de découvrir des défauts (c.-à-d. non encore rencontrés et cachés dans le logiciel). Un défaut est l'apparition d’une erreur de programmation dans le code source du logiciel. Lorsqu'un défaut est exécuté, il provoque une défaillance, qui est une manifestation de l'exécution du défaut. Un seul défaut peut provoquer une variété de pannes, en fonction de l'état du programme lors de son exécution. Pour mieux atteindre l’objectif de découvrir le plus grand nombre de défauts, il est recommandé que ce ne soit pas la même personne qui développe et qui teste le logiciel *(Myers, Badgett, Sandler, 2012)*.

Principes des tests logiciel - The Art of Software Testing
1. La partie essentielle d'un cas de test est la définition du résultat attendu
2. Un programmeur devrait éviter de tester son propre code
3. Une entreprise de développement de logiciels devrait éviter de tester ses propres programmes
4. Vérifier minutieusement les résultats de chaque test
5. Les cas de tests doivent être écrits pour des conditions d'entrées invalides ou inattendues, aussi bien que pour celles qui sont valides et attendues
6. Examiner un programme pour voir s'il ne fait pas ce qui devrait ne constitue que la moitié de la bataille ; l'autre moitié consiste à voir si le programme fait ce qu'il n'est pas supposé faire.
7. Eviter de jeter les cas de tests, sauf si le programme est lui-même jeté.
8. Ne pas planifier des tests en faisant implicitement l'hypothèse qu'aucune erreur ne sera trouvée
9. Le risque de trouver une erreur dans une partie du code est proportionnel au nombre d'erreurs déjà trouvées dans cette partie.
10. Tester est une tâche très créative et proposant d'intéressants défis intellectuels

Tableau 1- principes de tests logiciels (Myers, 2012)

2.4 La nature des tests d'un logiciel

La nature des tests de logiciels peut varier selon l'objectif du test, mais aussi le moment où ils interviennent dans le cycle de vie de développement du logiciel. Les tests peuvent être définis par trois concepts *(Padrat-Peyre, Printz, 2009)*:

1. L'objet du test ;
2. Le niveau de tests ;
3. Les types de tests.

Le concept d'objet de test précise la raison du test. S'agit-il d'un test fonctionnel, non-fonctionnel ou encore d'un test de régression ? Voici une explication de chacune de ces perspectives différentes de tests :

- Les tests fonctionnels : qui vérifient que le logiciel exécute les fonctionnalités décrites dans le dossier de spécifications fonctionnelles (c.-à-d. le document de vision) ;

- Les tests non-fonctionnels : qui vérifient que le logiciel rencontre les exigences d'attributs de qualité tels que définis dans le document de vision et précisés dans la norme *(ISO/IEC 9126, 2001)* ;
- Les tests de régression : qui vérifient qu'une correction d'erreur ou une amélioration n'a pas affecté le reste du logiciel.

Le deuxième concept est le niveau de test. L'exécution de tests du logiciel, spécialement pour les grands logiciels, est typiquement effectuée par niveaux. Dans la plupart des entreprises il y aura quatre à cinq niveaux de tests : unitaire, intégration, système, acceptation et opérationnel *(April, 2016)*. Dans les normes ISO, de l'IEEE et même dans le SWEBOK, le nombre de niveaux est de trois : unitaire, intégration et systèmes *(SWEBOK, 2014)*. Au niveau unitaire, les tests visent à identifier des défauts fonctionnels et structurels. Au niveau de l'intégration, les tests visent à découvrir les problèmes de communication entre les unités et s'assurer du bon fonctionnement et de la structure du groupe. Au niveau du système, la fonctionnalité est encore une fois testée, mais ce niveau de test va contenir des types de tests variés avec des objectifs spécifiques au logiciel testé. Au niveau de l'acceptation, le test est effectué par le client pour bien s'assurer que le logiciel fait ce qu'il désire. Ce niveau de tests vise à s'assurer que toutes les fonctions attendues du logiciel sont implémentées et fonctionnent correctement (c.-à-d. du point de vue des utilisateurs finaux). Finalement le test de niveau opérationnel s'assure que le logiciel, une fois accepté par le client, est bien installé dans l'environnement de production.

Le troisième concept de tests est le type de test qui définit toutes sortes de tests qui ont un objectif spécifique. Ces tests sont exécutés sur des composants ou sur le logiciel et même le système (c.-à-d. le logiciel opérant sur les équipements) au complet. Voici des exemples de types de tests *(April, 2016)*:

- Installation/désinstallation – Type de tests d'un logiciel, sur différentes plateformes cibles, qui vise à s'assurer qu'il s'installe, fonctionne et se désinstalle de façon satisfaisante ;
- Alpha et bêta – Avant qu'un logiciel de masse soit commercialisé, il est parfois distribué à un ensemble restreint et représentatif d'utilisateurs potentiels pour des essais préliminaires, soit une première fois en interne (test alpha) ou à l'externe pour utilisation restreinte (test bêta). Ces utilisateurs signaleront des défauts avec le produit ;
- Fiabilité – Ce type de test évalue les endroits critiques qui peuvent influencer son bon fonctionnement (le temps de service entre deux pannes) ;

- Régression – Ce type de tests consiste à effectuer des essais sur les parties (composants) qui ne sont pas touchées directement par une modification. Il vise à s'assurer qu'il n'y a pas eu d'effets secondaires imprévus lors des modifications. La définition officielle est : 'Des tests sélectifs d'un système ou composants pour vérifier que les modifications n'ont pas entrainé des effets inattendus" ;
- Performance – Type de tests qui s'assure que le temps de réponse d'une transaction, telle que perçue par l'utilisateur final, reste à l'intérieur de limites spécifiées acceptables. Ce test est spécifiquement destiné à vérifier que le logiciel répond aux exigences spécifiques de performance, par exemple, la capacité et le temps de réponse. Un genre spécifique de tests de performance est le test de volume.
- Sécurité – Type de tests qui s'assure d'identifier des vulnérabilités contre les accès accidentels, non autorisés, les menaces des virus et les logiciels espions ;
- Stress – Les simulations de comportement du logiciel à sa charge maximale (tel que spécifié lors de sa construction), ainsi qu'au-delà de cette charge. Dans certains cas, ce type de tests cherchera à rendre certaines ressources non disponibles pour voir le comportement du logiciel ;
- Dos à dos – Dans ce type de tests, le même jeu d'essais est effectué sur deux versions différentes d'un même produit logiciel, et les résultats sont comparés ;
- Recouvrement (aussi nommé test de reprise) – l'objectif de ce type de tests est de s'assurer que le logiciel peut récupérer facilement et rapidement suite à une défaillance ;
- Interface – Type de tests qui s'assurent que les interfaces inter systèmes offrent bel et bien les services demandés ainsi que les contrôles associés ;
- Configuration – Tests effectués pour le cas où le logiciel est construit pour desservir différents utilisateurs. Le test de configuration analyse le logiciel sous diverses configurations spécifiées.
- Utilisabilité – Type de tests du logiciel, pour différents types d'utilisation, qui s'assure que l'interface utilisateur s'acquitte, de manière satisfaisante, des exigences d'utilisabilité spécifiées. On vise ici à évaluer si l'utilisateur peut faire son travail efficacement en suivant la séquence des opérations dans l'interface. Ce test a pour objectif d'évaluer la facilité pour les utilisateurs finaux à utiliser et à apprendre le logiciel et d'évaluer l'efficacité des fonctions à soutenir les tâches des utilisateurs. Un genre spécifique de tests d'utilisabilité est le test de la documentation des utilisateurs ;

- Documentation utilisateur – Ce type de tests vise à s'assurer que les messages d'erreurs et la documentation s'arriment parfaitement avec le comportement du logiciel ;
- Bout en bout – Dans ce type de tests, les transactions d'affaires sont initiées et suivies tout au long de leurs transformations jusqu'à complétude. Par exemple, dans un dossier d'historique de l'employé, un employé se joint à l'entreprise ; est ensuite promu; est ensuite rétrogradé, des augmentations de salaire sont accordées, quelquefois des diminutions de salaire peuvent être effectuées; un dossier peut être gardé en suspens; un employé transféré, puis partir à la retraite, mis à pied ou décéder. Ce type de test s'assure que la transaction d'affaires est fonctionnelle ;
- Cohérence (« Sanity Test ») – Ce type de tests effectue un survol rapide de l'ensemble des composants pour s'assurer que le « build » contiendra les versions appropriées et complètes des composants appropriés pour la version désirée ;
- Charge – Le test de charge est un type de tests populaire pour les applications Web et les applications multiutilisateurs. Un grand nombre d'utilisateurs sont connectés et essaient d'utiliser le logiciel d'une manière aléatoire. L'objectif est de voir si le logiciel permet de bien gérer plusieurs demandes de toutes sortes. Ce type de test fait ressortir des problématiques liées à la bande passante, les bases de données, la suffisance de mémoire, les accès aux disques, etc. Un genre spécifique de tests de charge est le test de concurrence ;
- Concurrence – Le type de tests concurrent requiert aussi un certain nombre d'utilisateurs. Mais cette fois-ci ils utilisent tous la même fonction simultanément en saisissant les mêmes données. Ce type de test vise à faire ressortir la capacité du système à exécuter simultanément la même transaction et en préservant sa performance et surtout l'intégrité des données. Par exemple, prenez le scénario de réservation des billets, il y a un seul siège et il est actuellement disponible. Au moment de confirmer l'achat, le système devrait l'assigner à une seule personne et rejeter les demandes des autres ;
- Parallèle – Ce type de tests permet de reproduire exactement l'environnement de la production. Le test parallèle est utile, car il permet de simuler exactement ce qui va se passer en production si on mettait le composant en service ;
- Déploiement (aussi nommé d'opérabilité) – Ce type de tests vise à simuler toutes les étapes de mise en production et de retrait en cas de problème. C'est une préparation consciencieuse afin de s'assurer que le déploiement spécifié est approprié et fonctionnera sans problèmes ;

- Maintenabilité – Type de tests du logiciel pour s'assurer que sa maintenance sera efficace. La maintenabilité dans ce contexte signifie: 1) qu'il est facile d'identifier l'endroit ou faire une modification ; 2) effectuer une modification se fait facilement; 3) il est simple de tester la modification et d'effectuer le test de régression sur le reste du logiciel et 4) de faciliter son déploiement suite à une modification ;
- Au niveau opérationnel, les tests visent à identifier s'il n'y a pas dégradation des services suite au changement ou à l'ajout de logiciel.

Chaque niveau et type de tests découvrira des défauts. Dans la prochaine section présentera les méthodes de classification des défauts spécialement utiles pour le spécialiste en AQL.

2.5 Les stratégies de tests

Il existe deux grandes familles de techniques de tests de logiciels : les stratégies « statiques » et les stratégies « dynamiques ». Les techniques statiques se basent sur les sources du logiciel à tester sans le faire fonctionner et les techniques dynamiques s'appuient sur l'exécution du logiciel.

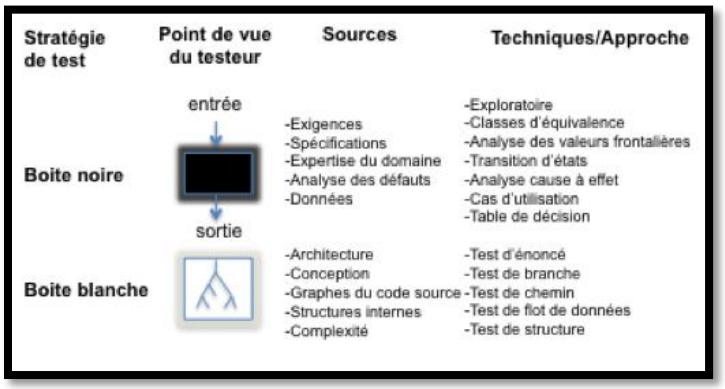

Figure 1 - Les stratégies de test boite noire et boite blanche *(Burnstein, 2003)*

Les sections qui suivent décrivent ces stratégies.

2.5.1 Tests dynamiques

Dans le cadre de tests dynamiques, le testeur va exécuter le logiciel dont le comportement dépend de son interaction avec son environnement. Il exécute le programme avec des valeurs en entrée et on observe le comportement. Le rôle du testeur est de trouver les « bonnes » valeurs d'exécution, soit les valeurs d'environnement qui auront le plus de chance de mettre en défaut le logiciel. Il existe trois principales catégories de techniques de tests dynamiques : les techniques basées sur les spécifications dites de « boite noire », techniques basées sur la structure dite de « boite blanche », techniques basées sur l'expérience dite « aléatoire ».

2.5.1.1 Technique de boite noire

La technique boite noire est une stratégie de test basée sur les spécifications. Elle consiste à considérer le logiciel comme une boite noire avec des entrées et des sorties. En effet, le but est de vérifier les principales fonctions du logiciel. Plusieurs méthodes peuvent être utilisées pour un test « boite noire » :

- Partition d'entrée : l'idée est de diviser le domaine des entrées en un nombre fini de classes tel que le programme réagit pareil pour toutes les valeurs de la classe ;
- Analyse des valeurs limites : le principe est de tester les valeurs aux limites des domaines ou des classes d'équivalence. Les erreurs se nichent dans les cas limites ;
- Tables de décision.

2.5.1.2 Technique de boite blanche

La technique de la boite blanche est une stratégie de test basée sur la structure interne du programme. Dans ce type de test, les testeurs accèdent au code source ce qui leur permet de possibilité de fixer des valeurs d'entrées afin d'assurer que :

- Toutes les conditions d'arrêt de boucle ont été vérifiées ;
- Toutes les branches d'une instruction conditionnelle ont été testées ;
- Les structures de données internes ont été testées (pour assurer la validité).

2.5.2 Techniques statiques

Les techniques statiques n'exécutent pas le code et sont généralement utilisées avant que tout test ne soit réalisé sur le logiciel. Elles peuvent servir à tester le code source, les documents de désign, des spécifications fonctionnelles ou encore des requis. Les types d'erreurs qui sont facilement détectables grâce aux tests statiques (ou revues) sont les suivants : requis manquants, erreurs de désign, divergence par rapport aux standards, code non soutenable ou évolutif, etc. Contrairement aux tests dynamiques, cette stratégie de test permet de trouver des défauts plutôt que des défaillances du logiciel. Les techniques statiques de tests ont de nombreux avantages entre autres :

- Une validation précoce des requis des usagers et une rétroaction rapide sur les problèmes de qualité ;

- Un échange d'information constructif entre les participants du projet (développeurs, clients/usagers, testeurs) ;
- Une amélioration de la productivité de développement – en effet, les erreurs étant trouvées très tôt dans le cycle de développement, les coûts de reprise de code sont relativement réduits.

Par contre, celles-ci ne pourront être exécutées que par des testeurs ayant une bonne connaissance du langage de programmation utilise. En effet, ces derniers devront livre et interpréter le code manuellement, et comprendre les messages produits par les outils utilisés.

Comme stratégies de tests statiques, on distingue les procédés de revues : revue de code, revue par pairs, revue technique, et une analyse automatique (c.-à-d. vérification de propriétés, règles de codage, etc.).

2.6 Conclusion

Ce chapitre a présenté une vue générale des tests. Les tests doivent être faits dans un objectif bien précis. Ainsi selon l'évolution du logiciel dans le cycle de développement il faut déterminer tant la nature que la stratégie des tests qui sont applicables au logiciel. Dans le cadre de ce projet qui porte sur les tests d'acceptation du logiciel, la stratégie qui va être adoptée est celle dite dynamique. En effet les tests d'acceptation seront basés sur la technique de la boite noire. Tel que présenté ci-dessus, cette technique repose sur l'analyse des spécifications tant bien fonctionnel et non-fonctionnel sans tenir compte de la structure interne du logiciel. Les tests valideront donc la présentation de l'interface client et valider que toutes les fonctionnalités d'ajouts, de modifications et de suppression de données se réaliser comme il faut mais aussi la relation entre les données lorsque cela est prévu par l'analyse.

CHAPITRE 3

Réalisation du projet

3.1 Introduction

Ce projet de recherche appliquée consistait à définir la planification et l'exécution d'un plan de test d'acceptation ainsi que d'accompagner l'utilisateur final dans ce processus. Il vise aussi à mettre en place un outil de gestion des problèmes.

3.2 Activités réalisées

La contribution au projet s'est faite sur plusieurs axes :

- Apprentissage du prototype existant (PACIQ) ;
- Formation ;
- Correction d'erreurs ;
- Stratégie de test et aide aux tests d'acceptation ;
- Mise en place d'un outil de gestions des anomalies.

L'objectif premier de ce projet était de mener à bien les activités de test en vue de la mise en production du prototype. Pour donc réaliser cette tâche, il a fallu apprendre le fonctionnement de l'application tant au niveau fonctionnel que technique.

3.2.1 Apprentissage du prototype existant (PACIQ)

Le prototype logiciel PACIQ est une application web qui comporte deux volets. Le premier volet est une application web basé sur la technologie de Microsoft ASP.net *(Grahan, Van Veenendaal, Evans, Black, 2009)* et le second volet est un module d'intelligence d'affaire (et de tableaux de bords) qui a été conçu à l'aide de la technologie SSRS et SSAS de Microsoft *(Grahan, Van Veenendaal, Evans, Black, 2009)*. Il a donc été nécessaire de comprendre, techniquement, la structure des différents programmes qui avait été programmés afin de bien

les maitriser et de devenir familier avec leurs fonctionnements. De plus, pour cet apprentissage, il était incontournable de comprendre les besoins et règles d'affaires. Pour ces deux aspects, des échanges avec des étudiants, prédécesseurs à ce projet de recherche, ont permis de bien comprendre le prototype autant au niveau fonctionnel qu'au niveau technique : c'est à dire l'architecture du logiciel qui a été mise en place pour réalisation ce premier prototype logiciel. De plus, les rencontres avec la représentante utilisatrice, Mme Isabelle Olivier, de l'hôpital Sainte-Justine, ont permis de confirmer les fonctionnalités. Le tableau qui suit liste les rencontres réalisées lors de cette étape du projet.

Date	Lieu	Participant	Objectif	Heures
20/07/2015	St -Justine	Alain April Isabelle Olivier Michel Lemay Guypacome Gbelai	Reprise du projet, Introduction de Guypacome à l'équipe de st-Justine	1h
/08/2015	St -Justine	Isabelle Olivier Guypacome Gbelai	Configuration environnement	1h
08/09/2015	St -Justine	Isabelle Olivier	Configuration environnement + test	1h30
29/09/2015	St -Justine	Guypacome Gbelai	Test	1h
09/10 2015	St -Justine	Isabelle Olivier	Test	1h
/12/2015	St -Justine	Guypacome Gbelai	Test	1h
5/01/2016	St -Justine	Isabelle Olivier	Test	1h
/02/2016	St -Justine	Guypacome Gbelai	Test	1h

Tableau 2 - tableau de rencontre

3.2.2 Formation

Le projet PACIQ était arrêté temporairement depuis quelques mois faute de la disponibilité du personnel de l'hôpital et aussi d'étudiants de l'ÉTS. Il était donc important de faire une mise à niveau, avec la représentante/utilisatrice, pour qu'elle se refamiliarise sur ce qui avait été

réalisé par le passé. Même si la représentante avait une bonne compréhension du besoin initial et de ce que PACIQ faisait, ces séances ont servi à redécouvrir l'application en parcourant les menus ainsi que les différents interfaces utilisateurs et formulaires de l'application. Elle a donc pu se familiariser à nouveau concernant la navigation dans l'application et a repris possession de l'application. Tester une application n'est pas une activité simple. Pour que les tests soient efficaces, une formation initiale sur comment tester et comment prioriser les demandes de changements ainsi que les défauts éventuels est nécessaire.

La classification et priorisation des requêtes permet de pouvoir distinguer les anomalies des demandes de changements prioritaires versus celles qui sont moins urgentes. Cette classification est d'autant plus importante, car elle permet de déterminer les anomalies qu'il faut corriger impérativement avant la mise en production et celles qui peuvent être corrigées lors d'une itération subséquente.

En plus de la formation de la représentante, Mme Isabelle Olivier de Sainte-Justine, une nouvelle étudiante a joint le projet lors de la session d'hiver 2016. Afin de faciliter son intégration au projet, il a fallu l'informer sur ce qui avait été fait, lui expliquer le logiciel ainsi que les attentes de la cliente. De plus, il a fallu l'aider à mettre en place un environnement local de développement sur son propre poste afin qu'elle puisse approfondir ses connaissances de l'application PACIQ.

3.2.3 Tests

Les tests ont été effectués avec l'aide de la technique boite noire, c'est-à-dire des tests fonctionnels. Ces tests sont purement une représentation des spécifications développées sans se préoccuper de l'implémentation. Lors de cette phase, les tests sont exécutés dans un environnement similaire à celui de la production avec tous les composants du système. Ces tests, qui visent une perspective de la cliente, permettent de faire la validation du logiciel afin de déterminer s'il répond vraiment aux spécifications fonctionnelles et non fonctionnelles telles que décrites dans le document de vision.

Pour effectuer ces tests, il a fallu préparer l'environnement de test afin que les données utilisées reflètent le plus près possible la réalité du client. Pour ce faire, la cliente a fourni des données pour toutes les tables qui contiennent les données maitresses. Il a été nécessaire de charger ces données dans la base de données SQL serveur de l'environnement de test. Les données maitresses sont des données essentielles au système PACIQ qui permettent de pouvoir utiliser le système et entrer des projets d'amélioration.

Le défi de cette activité était de connaitre les relations entre les tables de PACIQ afin de pouvoir supprimer les données issues des phases de tests précédentes et ensuite charger ces nouvelles données. Pour insérer les nouvelles informations dans la base de données il a fallu développer un script qui exploite la fonctionnalité de « *bulk copy* » du SQL Server afin de faciliter ces insertions.

Des tests fonctionnels ont par la suite été réalisés. Ces premiers tests ont porté plus sur la configuration et la saisie de données de gestion de PACIQ. La représentante sera responsable de la mise à jour de toutes les données maitresses de PACIQ. Il est donc essentiel que tous les menus de gestion de ces données soient testés. Les séances de tests se sont déroulées dans les bureaux du département de la qualité à Sainte-Justine avec la représentante.

Les séances avec la représentante ont permis de parcourir, de manière systématique, les différents menus de l'application et ainsi de valider les données rentrées préalablement de manière automatique (c.-à-d. les données maitresses du chargement initial), mais aussi de valider l'interface graphique. Toutes les données de configuration initiale (données maitresses) ont été rentrées dans l'application via des scripts SQL.

Elles ont aussi permis d'effectuer la saisie et la modification de données dans les différents formulaires proposés par l'application. Ces données contrairement aux données maitresses étaient rentrées de façon manuelle par la représentante. Les validations sur les fonctionnalités de suppressions et d'ajouts de données ont pu être aussi testées dans l'application PACIQ. Ces tests ont également permis de valider que les données présentes dans le système étaient disponibles dans toute l'application PACIQ. Une revue de chaque table de données maitresses a permis, en plus, de confirmer que les relations entre les données étaient bien correctes et correspondait bien aux besoins exprimés dans le document de vision du projet.

Une matrice de capture d'anomalies *Tableau 3 - matrice des anomalies*, situé à l'annexe 1, a été mise en place pour que les testeurs puissent rentrer le résultat de leurs tests. Le fichier de la matrice a été créé pour récupérer les résultats des tests puisse que le système de suivi de tickets n'était pas encore fonctionnel.

3.2.4 Système de suivi de tickets/défauts

Pour une meilleure gestion et suivi de cette étape de test d'acceptation, au niveau des défauts et des demandes de changements, un logiciel de gestion de tickets a été installé sur le serveur PACIQ de l'ÉTS,

Le critère principal pour la sélection du logiciel de gestion de ticket était qu'il devait être un logiciel libre pour ne pas engendrer de coût. De plus ce logiciel devait être compatible avec l'environnement technique du projet PACIQ, c'est-à-dire que ce logiciel soit compatible avec :

- SQL SERVER 2008 R2 ;

- WINDOWS SERVER 2008 ;
- IIS.

Un premier choix s'est arrêté sur l'application Mantis. MANTIS qui est l'un des plus populaires logiciels libres de suivi des défauts. Il peut être utilisé pour suivre plusieurs projets à la fois. Paru en 2000, il a été développé en langage de programmation PHP. Selon sa documentation, ce logiciel était compatible avec nos requis. Cependant l'installation de plusieurs composants tels que les extensions PHP étaient requises pour pouvoir faire fonctionner ce logiciel correctement sur un serveur Windows. Suite à plusieurs tentatives, le logiciel n'a jamais pu être s'installe comme voulu.

Category	Package	Minimum Version	Recommended	Comments
	MS SQL Server	2005	2005 or above	PHP extension: mssql or sqlsrv
	Oracle	8i	11gR2	PHP extension: oci8
Web Server	Apache	1.3.x	2.2.x	
	lighttpd	1.4.x	1.4.x	
	IIS	6.0	6.0	
PHP	PHP	5.1.x	5.2.x or above	See above for PHP extensions

Figure 2 – Requis Mantis pour l'environnement technique PACIQ

Suite à ces essais non-fructueux, un autre choix de logiciel libre qui répondra plus adéquatement aux critères énoncés à la figure 2 a été fait. L'application BugNet a donc été choisie pour un deuxième essai. Ce logiciel sera présenté plus en détail au chapitre 4. La liste des défauts relevés par l'utilisateur principal sur cette plateforme se trouve en annexe *ANNEXE II*39.

3.2.5 Conclusion

Cette section à démontrée l'apport au projet tant au niveau de la coordination des tests avec l'utilisatrice principale en lui donnant une formation sur le logiciel, la manière de l'utiliser et les tests qui devraient être réalisés. Il a aussi été nécessaire de l'assister à ces tests. De plus, une formation a été effectuée à l'étudiante qui a pris la relève sur ce projet. Un autre point important couvert par ce chapitre concerne la recherche et l'installation d'un outil libre, afin de capturer les défauts identifiés par l'utilisatrice. Cet outil est présenté dans le prochain chapitre.

CHAPITRE 4

BugNet – Système de gestion des tickets

4.1 Introduction

Bugnet est une solution libre de gestion de suivi des tickets pour l projets logiciels conçu sur la plateforme .Net (c.-à-d. C # et ASP.NET). Il fonctionne à l'aide d'une base de données Microsoft SQL. Bugnet a plusieurs fonctionnalités intéressantes qui permettent d'appuyer la gestion de projet, les tests et le suivi des tickets de problèmes et de demandes de changements. Voici ces fonctions principales :

- Les notifications par courriel : envoie des courriels de mises à jour, des commentaires, des résolutions des parties prenantes concernées ;
- Contrôle accès : permet contrôler l'accès des utilisateurs au niveau du projet ;
- Personnaliser : permet de facilement personnaliser Bugnet ;
- Offre l'intégration avec d'autres services, tels que :
    - gestionnaire de version de code.

La prochaine section décrit brièvement les différentes fonctionnalités du logiciel Bugnet.

4.2 Fonctionnalités

4.2.1 Multi -projets

Bugnet permet de gérer plusieurs projets de manière simultanée. Pour ce faire il suffit juste créer les projets et de les configurer selon les besoins.

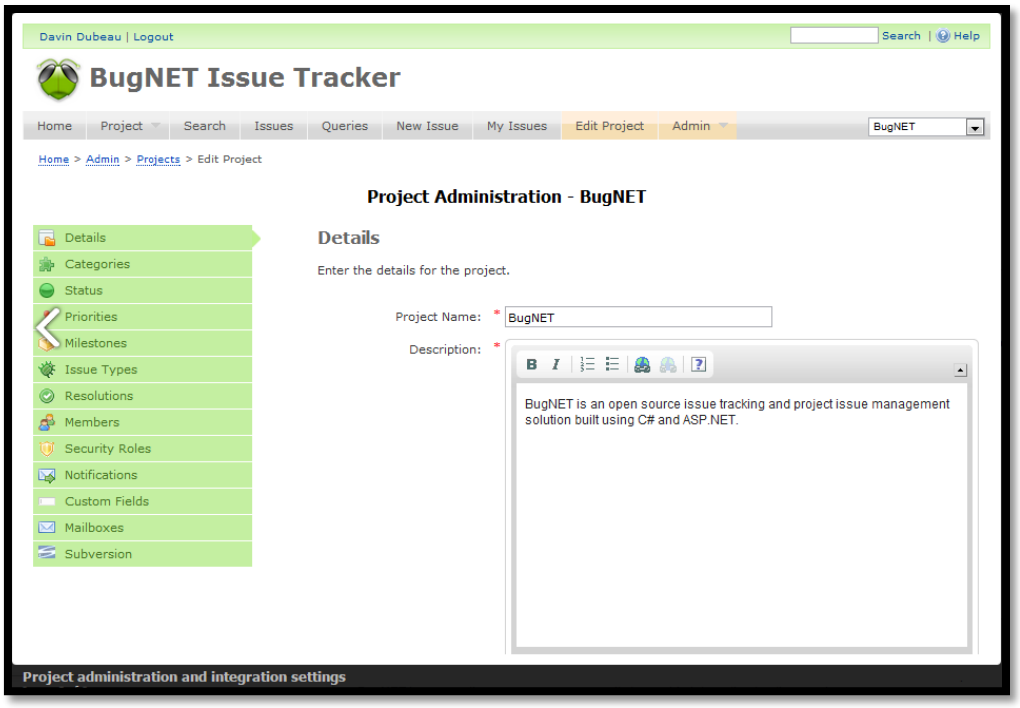

Figure 3- Bugnet - Création de projet

Les sections suivantes sont totalement configurables par projet. Pour ce faire il faut accéder au menu administration.

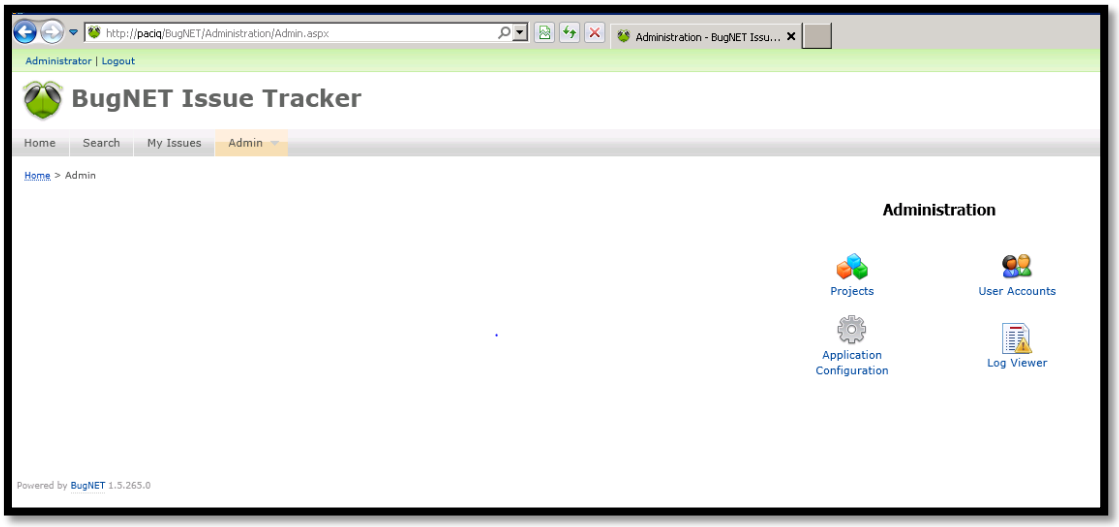

Figure 4 - Bugnet – Administration

4.2.1.1 Catégorie

Ce logiciel permet aussi de créer des catégories permettant ainsi de catégoriser les problèmes, par exemple :

- Code – le problème catégorisé sous ce type signifie que le problème est relié au code ;
- Configuration – le problème catégorisé sous ce type signifie que le problème est relié à la configuration.

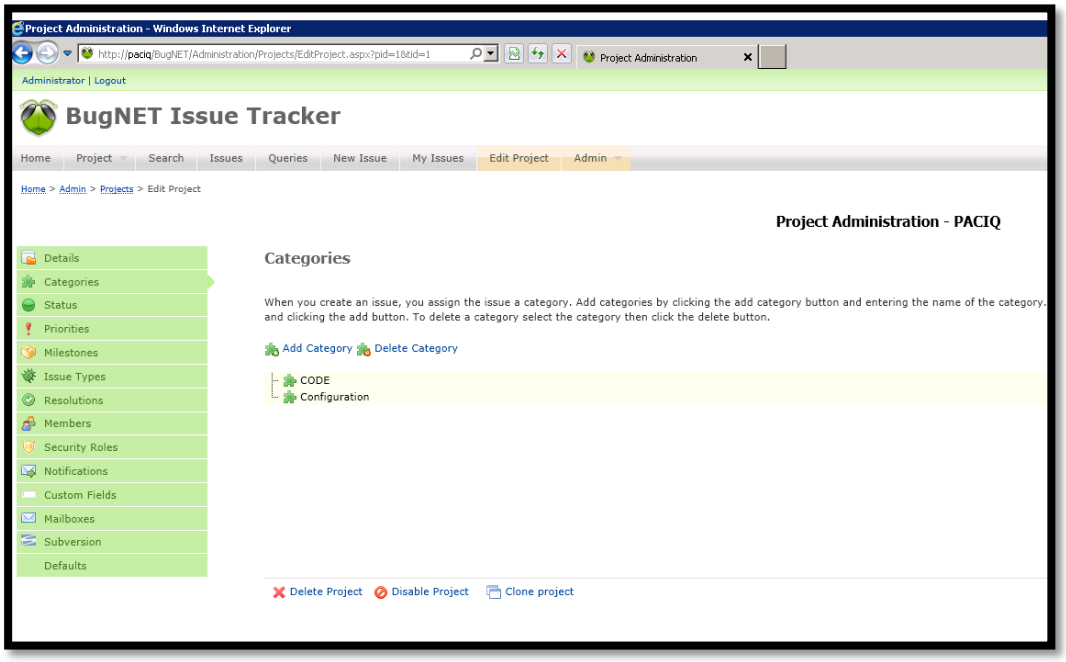

Figure 5 - Bugnet - administration Catégorie

4.2.1.2 Statut

Le statut, d'un billet, permet de connaître l'état d'un billet dans son cycle de vie de sa résolution.

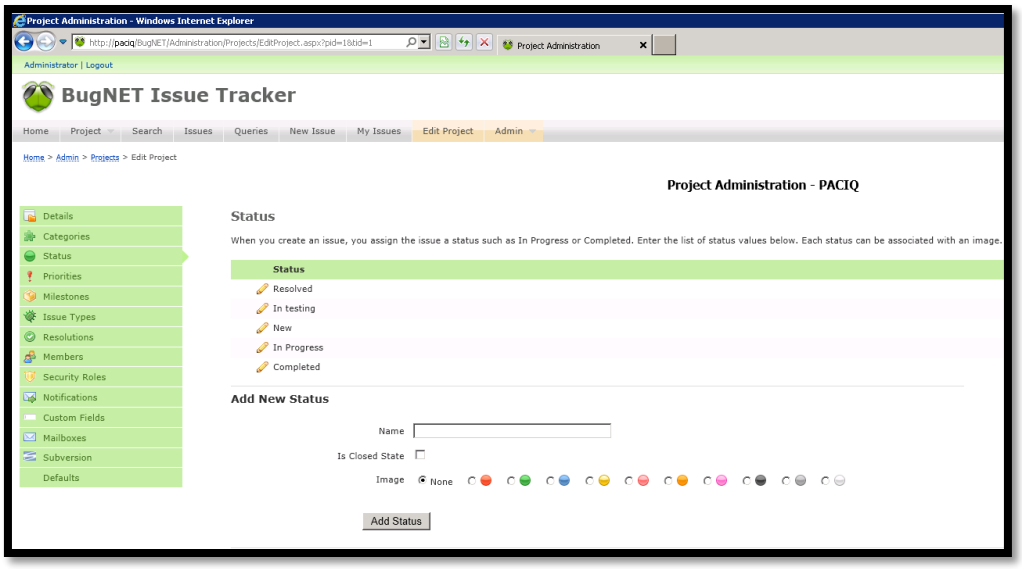

Figure 6 - Bugnet - Administration des statuts

4.2.1.3 Priorités

Il est aussi possible d'assigner une priorité aux billets. Cette fonctionnalité va permettre de pouvoir classifier le travail et permettre à l'utilisatrice principale d'identifier les problèmes et les changements les plus prioritaires.

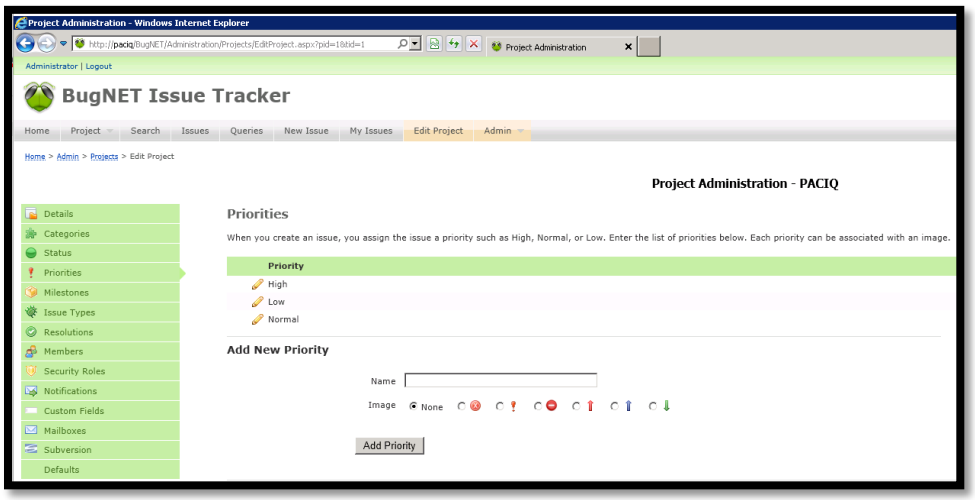

Figure 7 – Bugnet - Administration des priorités

4.2.1.4 Type de problème

Les billets que l'on soulève dans ce logiciel peuvent être de différents types, par exemple :

- Bug (problème à corriger) ;
- Task (demande de changement / amélioration).

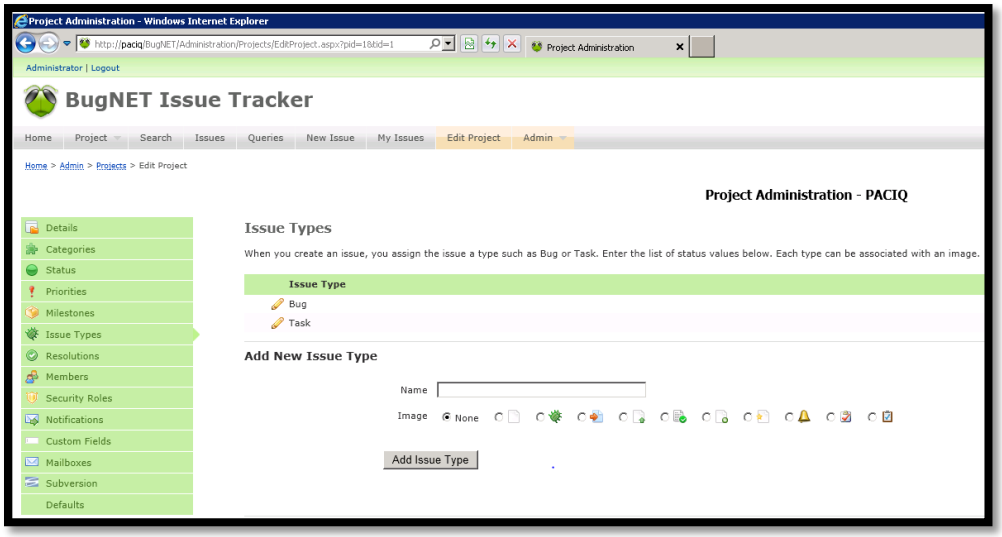

Figure 8 - Bugnet - Administration des types de problèmes

4.2.1.5 Membres

Il est aussi possible d'identifier les intervenants qui peuvent accéder aux billets de PACIQ.

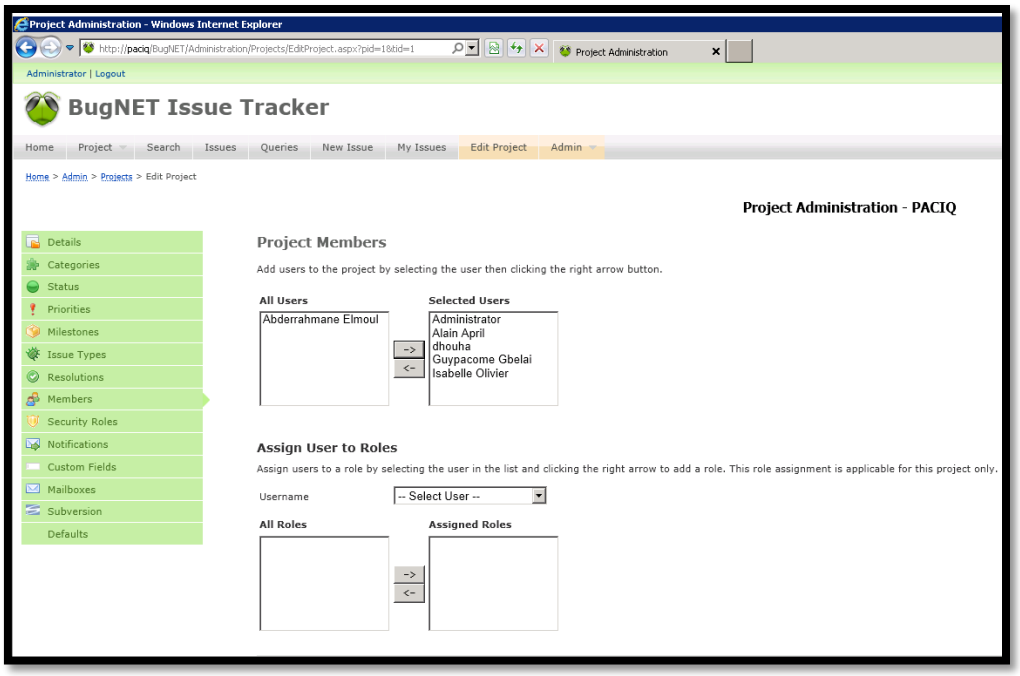

Figure 9 - Bugnet – Administration gestion des membres

4.2.1.6 Sécurité

A chaque membre associé à un projet, des droits d'accès peuvent être configurés en fonction de leurs responsabilités.

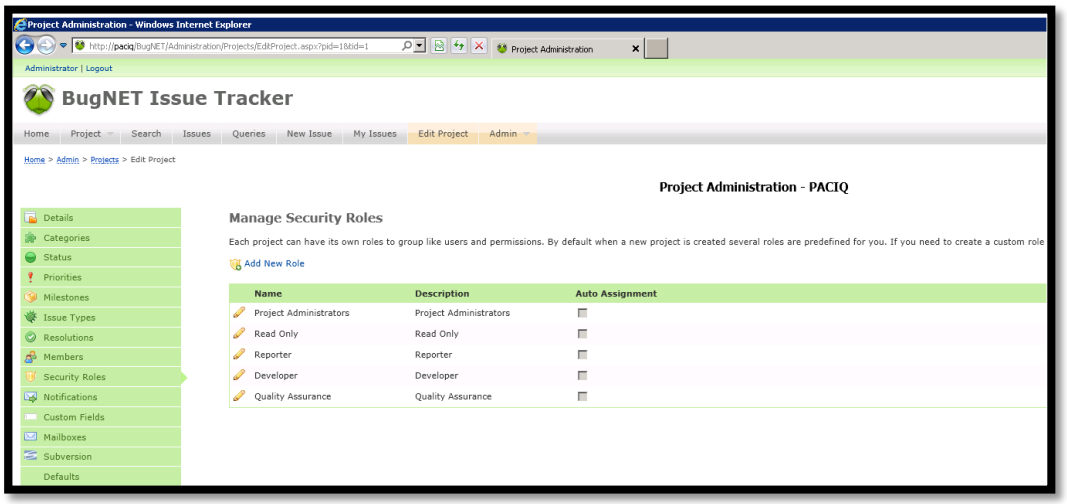

Figure 10 - Bugnet – Administration Gestion de la sécurité

4.2.2 Suivi des problèmes

Ce logiciel de billet offre une interface Web qui permet, de manière très simple et intuitive, de créer, gérer et signaler des problèmes et des demandes de changements. L'interface offre aussi la possibilité d'assigner une date limite à un billet et de permettre une estimation de l'effort requis. Les clients et les utilisateurs peuvent soumettre des questions par courriel, télécharger des captures d'écran ou des pièces jointes et les adjoindre au billet. De plus, l'utilisateur peut créer des champs personnalisés pour la saisie d'autres données. Avec toutes ces possibilités : commentaires, pièces jointes, les notifications, suivi du temps et de l'historique des billets, Bugnet permet aussi un suivi complet des tickets pour l'étape de tests d'acceptation et même pour les étapes subséquentes de maintenance/support.

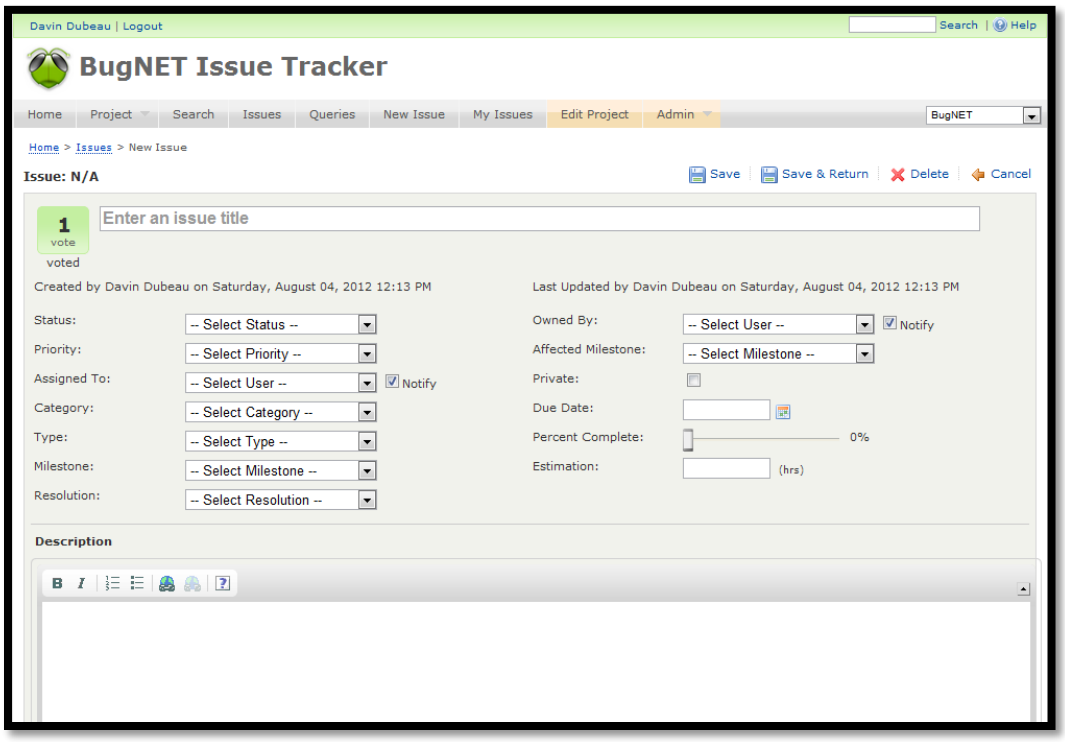

Figure 11 – Bugnet – saisir de problème

4.2.3 Suivi de projet

Bugnet s'adapte aussi facilement aux processus et aux flux de travail existant. Il permet de personnaliser les différents attributs de projets tel que : les types d'erreur, les statuts de projet, la priorité et plusieurs autres champs afin de rester personnalisable et de permettre de suivre les évolutions des processus d'entreprises qui évoluent. Les fonctionnalités suivantes sont disponibles et permettent un meilleur suivi des projets :

- Sommaire du projet, cette option montre une vue d'ensemble " en bref " du projet ;
- Des requêtes personnalisables peuvent être conçus et enregistrés pour fournir une analyse détaillée ;
- Les listes des bogues sont exportables vers Excel ou RSS ;
- Calendrier montrant les bogues ainsi que leur date limite par étapes ;
- Début d'itération et de fin avec le journal des modifications et des vues feuille de route.

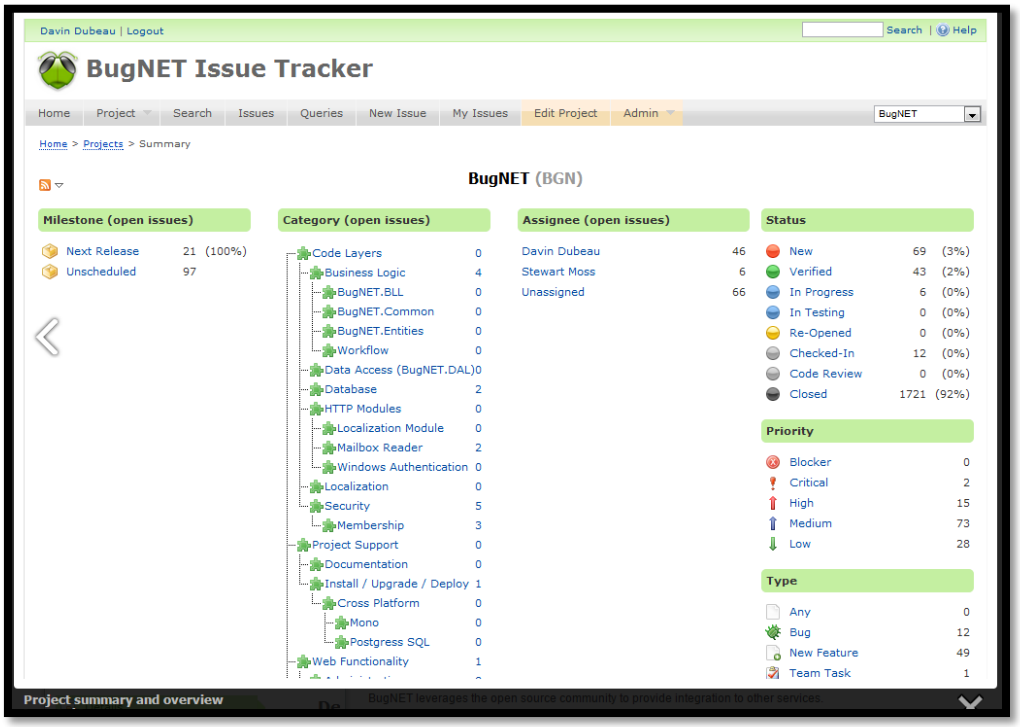

Figure 12- Bugnet – Suivi de projet

CONCLUSION

La mise en service de l'application PACIQ, avait pour but de permettre à la DQSR du CHUSJ d'appuyer son processus d'affaire d'amélioration de la qualité des services de santé avec un logiciel.

Le but premier de ce projet de recherche appliquée était d'obtenir, à la suite des tests d'acceptation, une validation du produit afin qu'il puisse être déployé en production. Plusieurs sessions de tests ont été réalisées et même si l'utilisatrice pilote n'a pas encore donné son aval pour la mise en production d'une manière formelle, on peut constater, par la nature des défauts identifiés, que le logiciel répond bien au besoin du CHUSJ et qu'il pourra être déployé sous peu.

Ce projet a permis de pouvoir mettre en place un outil de gestion des défauts qui pourra être réutilisé pour le suivi et les améliorations de l'application Paciq tout au long de son évolution. Ce projet de maitrise appliquée, de 15 crédits, a permis de mettre en pratique plusieurs connaissances acquises lors de du cursus universitaire de la maîtrise en génie logiciel de l'ÉTS et d'expérimenter une étude de cas réelle, c'est-à-dire, le travail en équipe avec la cliente et une collègue.

RECOMMANDATIONS

Cette section décrit les recommandations afin que l'application puisse être déployée. Ces recommandations tiennent en trois points :

- La disponibilité de l'utilisateur pilote ;
- L'implication de plus d'une personne du côté du CHUSJ ;
- Pour les étudiants de l'ÉTS, maitriser les outils de développement qui ont été utilisés pour bien comprendre le logiciel.

Avec ces trois recommandations, le logiciel PACIQ pourra être déployé très prochainement à l'hôpital Sainte Justine. Finalement, un plan de déploiement, qui est en cours de conception, aidera grandement à passer l'étape du déploiement final par le groupe informatique de l'hôpital.

ANNEXE I

Matrice des anomalies

Matrice des anomalies

No anomalie	Description anomalies	Opération	Section	Priorité	Fréquence
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					

Tableau 3 - matrice des anomalies

matriceAnomalie.xlsx

Tableau 4 - matrice des anomalies complètes

ANNEXE II

Résultat des tests et bugs ouverts

		Id	Title	Project	Votes	Category	Creator	Owner	Assigned	Type	Milestone
FALSE	Private	PACIQ-26	changer le mot de passe		1	Configuration	Isabelle Olivier	Isabelle Olivier	Abderrahmane Elmoul	Bug	UAT
FALSE	Private	PACIQ-24	Aucun plan amélioration n'apparait dans la liste des plans amélioration. de plus comment allons-nous faire la sélection des plans selon les directions services etc....		1	Configuration	Isabelle Olivier	Isabelle Olivier	Abderrahmane Elmoul	Bug	UAT
FALSE	Private	PACIQ-23	Indicateur pouvoir inscrire un indicateur pas seulement un menu déroulant		1	Configuration	Isabelle Olivier	Isabelle Olivier	Abderrahmane Elmoul	Bug	UAT
FALSE	Private	PACIQ-22	Création de plusieurs activités d'amélioration pour un objectif.		1	Configuration	Isabelle Olivier	Isabelle Olivier	Abderrahmane Elmoul	Bug	UAT
FALSE	Private	PACIQ-21	Ajout de responsable pour l'objectif		1	Configuration	Isabelle Olivier	Isabelle Olivier	Abderrahmane Elmoul	Task	UAT
FALSE	Private	PACIQ-19	plan amélioration, création, zone de texte pour plan stratégique qui est plus long que la zone de visionnement		1	Configuration	Isabelle Olivier	Isabelle Olivier	Abderrahmane Elmoul	Bug	UAT
FALSE	Private	PACIQ-18	relation entre les critères pouvoir partir d'un programme d'accréditation. pas seulement celui agrément canada		1	Configuration	Isabelle Olivier	Isabelle Olivier	dhouha	Bug	UAT

FALSE	Private	PACIQ-16	Ajuster les champs écriture dans donnée de référence critère		1	Configuration	Isabelle Olivier	Isabelle Olivier	dhouha	Task	UAT
FALSE	Private	PACIQ-15	Données référence: normes: afficher les normes par programme accréditation et cahier de normes		1	Configuration	Isabelle Olivier	Isabelle Olivier	dhouha	Bug	UAT
FALSE	Private	PACIQ-14	Données référence: normes: afficher les normes par programme accréditation et cahier de normes		1	Configuration	Isabelle Olivier	Isabelle Olivier	dhouha	Bug	UAT
FALSE	Private	PACIQ-13	Modifier le libellé du cahier de normes.		1	Configuration	Isabelle Olivier	Isabelle Olivier	dhouha	Bug	UAT
FALSE	Private	PACIQ-12	Modifier la page accueil		1	Configuration	Isabelle Olivier	Isabelle Olivier	dhouha	Task	UAT
FALSE	Private	PACIQ-11	Modifier la page accueil		1	Configuration	Isabelle Olivier	Isabelle Olivier	dhouha	Task	UAT

Tableau 5 - Résultat des tests

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

[1] ISO/IEC/IEEE 2013. «ISO 29119-1:2013, Software and systems engineering - Software testing - Part 1: Concepts and definitions, International Organization for Standardisation».

[2] Glenford, J. Myers, Corey Sandler et Tom Badgett. 2012. *The Art of Software Testing*, Third Edition. New Jersey (USA): Wiley, 240 p. En ligne < it-ebooks.info/book/2899/ >. Consulté le 17 Février 2016

[3] ISO/IEC. 2011. « ISO/IEC 25010:2011 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models ».

[4] IEC. 2006. «IEC 62304: 2006 Medical device software -- Software life cycle processes».

[5] K, Devi. 2012. « Testing in the healthcare domain ». En ligne. < http://fr.slideshare.net/QAIOffical/testing-in-the-healthcare-domain >. Consulté le 17 Février 2016

[6] Padrat-Peyre, Jean-François et Printz Jacques. 2009. *Pratique des Tests Logiciels - Concevoir et mettre en œuvre une stratégie de tests - Préparation à la certification ISTQB.* Paris(FR) : Dunod, 186 p.

[7] Ilene, Burnstein. 2003. *Pratical Software Testing: A process -oriented approach*. New York (USA): Springer, 709 p.

[8] Graham, Dorothy, van Veenendaal Erik, Evans Isabel et Black Rex. 2009. *Foundations of Software testing, ISTQB Certification*. Andover (UK): Course Technology - Cengage Learning, 258 p.

[9] Moulay-Youssef, TARIQ. 2014. « Rapport projet PACIQ, modélisation d'un programme d'amélioration continue et intégratif de la qualité au CHU Sainte-Justine ». Montréal, ETS, 186 p. En ligne. < http://publicationslist.org/data/a.april/ref-471/RAPPORT%20-%20PROJET%20PACIQ%20CHUSJ%20V5.3%20FINAL.pdf >. Consulté le 21 Avril 2015.

[10] KAMGUEM, ULRICH GHOMSI. 2014. « Rapport projet PACIQ, INTELLIGENCE Intelligence d'affaires pour le programme d'amélioration continue et intégratif de la qualité au CHU sainte-Justine ». Montréal, ETS, 107 p. En ligne. < http://publicationslist.org/data/a.april/ref-477/Rapport_Maitrise_Paciq_Ghomsi_public.pdf >. Consulté le 19 Avril 2015.

[11] GitHub, Inc. [s.d.]. « Wiki ». En ligne. < https://github.com/dubeaud/bugnet/wiki >. Consulté le 19 Avril 2015.

[12] Bourque, Pierre et Fairley Richard E. (Dick) 2014. Guide to the Software Engineering Body of Knowledge - SWEBOK, V3 edition. IEEE Computer Society, 335 p.