



Réf : ING-GI-2016-19

Rapport de Projet de Fin d'Études

Pour obtenir le

Diplôme d'Ingénieur en Génie Informatique

Option : Systèmes, Réseaux et Sécurité

Présenté et soutenu publiquement le 25 juin 2016

Par

Dhia Eddine SAIDI

Conception, implémentation et intégration d'un processus d'automatisation des tests fonctionnels

Composition du jury

Madame BEN AZZOUZ Lamia **Président**

Monsieur GARALI Wajdi **Rapporteur**

Madame KHELIFI Aicha **Encadrant Entreprise**

Monsieur GHORBEL Khaled **Encadrant ENSIT**

Année universitaire : 2015-2016

Signatures

Encadrants Entreprise



Encadrant ENSIT



Dédicaces

À mes chers parents, pour leur affection, leur patience et leurs prières.

À mes sœurs, pour leurs encouragements qu'elles m'ont apportés au cours de ce projet,

À mes meilleurs amis pour leur aide, leur encouragement et leur assistance,

À l'équipe OOPRO pour leur accueil chaleureux et tout le monde chez SOFRECOM pour leur aide,

À tous ceux qui me sont chers.

Dhia Eddine SADI

Remerciements

C'est parce que nous avons beaucoup estimé tous ceux qui nous ont écouté, conseillé et encadré que nous tenons à leur faire part de toute notre gratitude, et plus particulièrement, nous tenons à remercier à travers ces courtes lignes:

Madame Aycha KHELIFI, Monsieur Amen Allah EL OUAIFI et Madame Asma REZGUI mes encadrants à SOFRECOM, pour la confiance qu'ils ont su m'accorder, la patience qu'ils ont su avoir à mon égard et les conseils précieux qu'ils m'ont prodigués tout au long de ce projet.

Monsieur Khaled GHORBEL, mon encadrant à l'Ecole Nationale Supérieure des Ingénieurs de Tunis (ENSIT), parce qu'il a accepté de me guider et m'assister étape par étape, durant mon projet.

Ma famille dont je ne me permettrai pas d'oublier de la remercier, pour son soutien à la fois moral et matériel durant mon cursus et surtout pendant les moments difficiles.

Mes remerciements et ma reconnaissance s'étendent également à toute l'équipe SOFRECOM, pour leur aide, leurs remarques, leur esprit ouvert et leur respect.

Table des matières

Introduction générale.....	11
Chapitre I : Cadre du projet et étude de l'existant.....	1
Introduction	14
1 Présentation générale de l'entreprise.....	14
1.1 Historique	15
1.2 SOFRECOM Tunisie.....	16
1.3 Organisation	16
1.4 Direction d'accueil	17
2 Présentation du projet	17
2.1 L'OOPRO.....	17
2.2 Le module Web2	19
2.3 Approche Agile	19
3 Etude de l'existant	23
3.1 Migration du module WEB2	23
3.1.1 Le WEB3 est un site web adaptatif.....	23
3.2 Les tests de non-régression IHM du module WEB3	24
3.2.1 Les tests de non-régressions IHM en cours d'optimisation.....	24
3.3 Les tests de non régression IHM réalisés d'une manière manuelle.....	25
4 Problématique.....	25
5 Objectif du projet.....	26
Conclusion.....	26
Introduction	28
1 Le test logiciel	28
1.1 Vérification et Validation (V & V).....	28
1.2 Les méthodes des tests.....	29
1.3 Comparaison entre les méthodes de test.....	30
2 Les niveaux d'un test logiciel.....	31

2.1	Tests fonctionnels	32
2.2	Test unitaire	32
2.3	Test d'intégration	32
2.4	Test système	33
2.5	Test de régression	33
2.6	Test d'acceptation	33
2.7	Test de performance	34
2.8	Test de sécurité	34
3	Documentation du test	35
3.1	Le plan de test	35
3.2	Le Scénario de test	35
3.3	Le Cas de test	35
4	Les Tests dans un processus itératif SCRUM	36
5	L'automatisation des tests	37
5.1	Pyramide de test	37
5.2	Pyramide de test inversé	39
6	Etude Comparative et Choix des outils	39
6.1	Etude comparative des outils	39
6.1.1	Outil d'automatisation	40
6.1.2	Framework d'automatisation	40
6.2	Choix des outils	42
	Conclusion	42
	Introduction	44
1	Analyse des besoins	44
1.1	Besoins fonctionnels	44
1.2	Besoins non fonctionnels	44
2	Spécification des besoins	45
2.1	Identification des acteurs	45
2.2	Diagramme de cas d'utilisation général	45
3	Raffinement des cas d'utilisation	46

3.1	Sous-système « Gestion des tests » QC.....	46
3.2	Sous-système « Editor »	48
3.3	Sous-système «Intégration Continue»	49
4	Le Backlog du produit	51
4.1	Définition des Sprints du projet.....	51
	Conclusion	52
	Introduction.....	54
1	Architecture fonctionnelle	54
1.1	Architecture physique.....	54
1.2	Architecture logique	55
2	Vue statique du système	57
2.1	Diagramme de classe « login ».....	58
2.2	Diagramme de classe « Editor »	59
2.3	Diagramme de classe générale	60
3	Vue dynamique du système.....	61
3.1	Authentification.....	61
3.2	Exécution de script	62
3.2.1	Diagramme d'activité	62
3.2.2	Diagramme de séquence.....	63
3.2.3	Diagramme de séquence détaillé du cas d'utilisation « Exécuter un cas de test »	63
3.3	Mettre à jour les données de la table « keyword » de la BD	65
	Conclusion.....	65
	Introduction	67
1	Framework, technologies et outils utilisées.....	67
1.1	Java 7.0.....	67
1.2	Eclipse Mars.1 (Release 4.5.1)	67
1.3	Vaadin Framework 7.6.4	67
1.4	Apache Tomcat 7.0.....	68
1.5	Mysql 5.6.17.....	68
1.6	Robotframework 2.8.5.....	68

1.7	Selenium Server 2.53.....	68
1.8	StarUML 2.7.0.....	68
1.9	Draw.io	68
1.10	SVN Tortoise 1.9.4.....	68
1.11	Jenkins 1.651.2 LTS (Long Term Support).....	69
2	Fonctionnalités réalisées.....	69
2.1	Authentification.....	69
2.2	Editor	71
2.2.1	Système de fichier du projet	71
2.3	Fonctionnalités	72
2.4	Exécution.....	74
2.5	Consulter le « Log.html » et le « report.html »	74
3	Composant de la solution	75
3.1	Jenkins	76
3.2	HP QC	76
	Conclusion.....	77

Table des figures

Figure 1 Logo de SOFRECOM	14
Figure 2 SOFRECOM dans le monde	15
Figure 3 Organigramme de SOFRECOM Tunisie	16
Figure 4 Schéma de l'architecture d'OOPRO	18
Figure 5 WEB3 Responsive Design	19
Figure 6 Méthode Scrum	22
Figure 7 Devops	22
Figure 8 WEB2.0	23
Figure 9 WEB3.0	24
Figure 10 HP QC	25
Figure 11 Les axes des tests	34
Figure 12 Le test SCRUM	36
Figure 13 Pyramide de test	38
Figure 14 Pyramide de test inversé	39
Figure 15 Puzzle des Keywords	41
Figure 16 Diagramme de cas d'utilisation générale	46
Figure 17 Diagramme de cas d'utilisation générale de sous-système « Gestion des tests »	46
Figure 18 Diagramme de cas d'utilisation générale du sous-système « Editor »	48
Figure 19 Diagramme de cas d'utilisation générale du sous-système « Intégration Continue »	50
Figure 20 Description du scenario de cas d'utilisation « Configurer un Job »	50
Figure 21 Le Backlog du produit	51
Figure 22 Architecture physique	54
Figure 23 Décomposition en couche de l'architecture logique	55
Figure 24 Comparaison entre les design patterns « MVC » et « MVP »	56
Figure 25 Diagramme de classe « Login »	58
Figure 26 Diagramme de classe « Editor »	59
Figure 27 Diagramme de classe générale	60
Figure 28 Diagramme de séquence de scenario d'authentification	61
Figure 29 Diagramme d'activité pour le cas d'utilisation « Exécuter un script de cas de test »	62
Figure 30 Diagramme de séquence de l'interaction du module « Editor » dans le cas d'utilisation « Exécuter un scripts de cas de test»	63
Figure 31 Diagramme de séquence de cas d'utilisation « Exécuter un script de test »	64
Figure 32 Diagramme de séquence de cas d'utilisation « Mettre à jour les données de la table « Keyword» de la BD »	65

Figure 33 Interface de la page d'accueil.....	69
Figure 34 Interface de « Login »	69
Figure 35 Liste déroulante de choix de catégorie	70
Figure 36 Pop-up d'erreur « login ou mot de passe incorrect »	70
Figure 37 Interface de la page principale de l'application « Editor ».....	71
Figure 38 Interface « script sélectionné ».....	72
Figure 39 Les fonctionnalités complémentaires de « Editor »	73
Figure 40 L'autocomplete de « Editor ».....	73
Figure 41 Un script en cours d'exécution.....	74
Figure 42 Consulter le fichier «log.html» depuis Editor	74
Figure 43 Consulter le fichier «report.html» depuis Editor.....	75
Figure 44 Les liens des autres composants de la solution	75
Figure 45 Interface du Composant Jenkins	76
Figure 46 Interface du composant QC.....	76

Liste des Tableau

Tableau 1 Définition des mots clés.....	20
Tableau 2 Définition des rôles au sein d'une équipe.....	21
Tableau 3 Les avantages et les inconvénients de test boite noir.....	29
Tableau 4 Les avantages et les inconvénients de test boite blanche.....	30
Tableau 5 Les avantages et les inconvénients de test boite grise	30
Tableau 6 Comparaison entre les méthodes de tests	31
Tableau 7 Niveaux de tests	31
Tableau 8 Pyramide de test.....	32
Tableau 9 Selenium vs UFT	40
Tableau 10 Robotframework vs Silktest vs QF-test.....	41
Tableau 11 Description du scenario de cas d'utilisation « Ajouter un test dans le test plan de QC ».....	47
Tableau 12 Description du scenario de cas d'utilisation « Ajouter et exécuter un script de test automatique à partir de Editor ».....	49
Tableau 13 Sprint 0	51
Tableau 14 Sprint 1	52
Tableau 15 Sprint 2	52

Introduction générale

Au fil des années, la technologie n'a cessé d'évoluer d'une cadence fulgurante, offrant ainsi, aux industriels des possibilités énormes, afin d'améliorer la qualité et assurer la satisfaction des clients.

Aujourd'hui Les boîtes de développement de logiciels se trouvent face à un grand challenge intégrant les deux exigences suivantes :

- L'optimisation de la qualité des applications.
- Le respect des délais avec des budgets serrés.

Afin de fournir des solutions informatiques gagnantes. Comment certaines entreprises de développement ont-elles répondu aux multiples challenges de livraison de logiciels complexes ? Par l'automatisation répond-t-on généralement. Il y a en effet, d'excellentes raisons à cela puisque un grand nombre d'entreprises de pointe ont démontré la puissance des plates-formes d'automatisation des tests fonctionnels pour accélérer les délais de développement et réduire les coûts de livraison des applications. Cependant, la plupart des entreprises échouent dans cette démarche d'automatisation. Ce taux est symptomatique d'un état d'esprit organisationnel consistant à envisager l'automatisation comme étant une solution rapide et prête à l'emploi, tout en croyant qu'il suffisait d'acquérir le meilleur produit d'automatisation des tests pour résoudre comme par magie des problèmes de manque de temps et de ressources...

Dans ce contexte, s'inscrit notre projet de fin d'études intitulé « Conception et développement d'un processus d'automatisation des tests ».

Le projet consiste alors à développer un processus permettant l'automatisation des tests fonctionnels basé sur l'un des projets de l'entreprise dès la phase de création jusqu'à la phase de clôture.

Structure du rapport

Ce rapport comporte les cinq chapitres suivant :

Chapitre 1 : Cadre du projet et étude de l'existant

Nous présentons dans ce premier chapitre l'entreprise d'accueil, le cadre du travail, l'étude de l'existant et la problématique.

Chapitre 2 : Etat de l'art

Dans une première partie nous entamons une étude théorique dans laquelle nous décrivons la notion du test et ses différents types. Dans une deuxième partie nous nous intéressons à une étude comparative des outils que nous avons choisis comme étant des éléments de réponse à la problématique posée auparavant. A la fin de ce chapitre nous présenterons l'objectif de mon projet de fin d'études.

Chapitre 3 : Spécification des besoins

Nous spécifions la branche technique dont nous présentons certaines technologies et la branche informelle qui comporte les besoins fonctionnels et non fonctionnels de notre solution.
Enfin, nous présentons l'analyse des besoins par des diagrammes du cas d'utilisation.

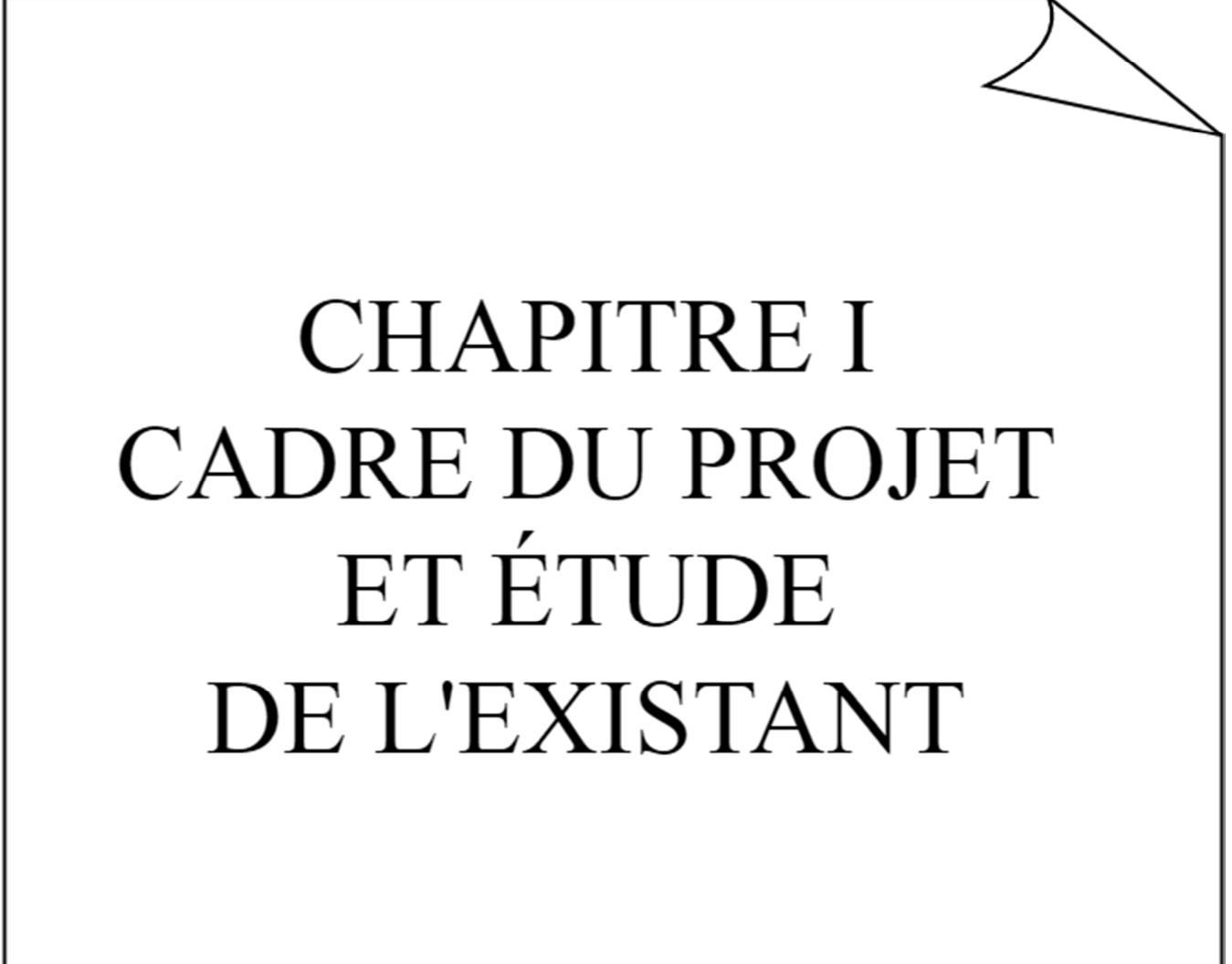
Chapitre 4 : étude conceptuelle

Nous détaillons dans ce chapitre l'analyse et la conception de notre solution.

Chapitre 5 : Réalisation

Nous précisons l'environnement logiciel et matériel utilisé.
Puis nous présentons les plus importantes interfaces de réalisation.

Le rapport de notre projet de fin d'études s'achève avec une conclusion générale qui résume les étapes de la réalisation du projet.



CHAPITRE I

CADRE DU PROJET

ET ÉTUDE DE L'EXISTANT

Cadre du projet et d'étude de l'existant

Introduction

Ce premier chapitre est consacré pour la présentation du contexte générale du projet. Le travail était réalisé au sein de la société SOFRECOM. Nous commencerons donc, par une présentation générale de l'organisme d'accueil. Ensuite, nous entamerons une description de notre projet pour expliquer son contexte, sa problématique ainsi que son objectif.

1 Présentation générale de l'entreprise



Figure 1 Logo de SOFRECOM

SOFRECOM, filiale du groupe France Télécom - Orange, apporte son expertise à l'international sur les marchés en forte croissance. Son savoir-faire et ses offres créatrices de valeurs dédiées aux opérateurs télécoms lui permettent d'intervenir sur l'ensemble des domaines de compétences nécessaires à leur développement. Grâce à sa connaissance du métier d'opérateur, des technologies de l'information et de la communication, ses nombreuses références et son réseau mondial de partenaires, SOFRECOM, aujourd'hui, est un acteur incontournable non seulement dans le secteur de la télécommunication mais également auprès des gouvernements, des investisseurs et des bailleurs de fonds internationaux.

SOFRECOM est mondialement réputée grâce à ses réalisations ainsi qu'à ses implantations locales à travers le monde :

Algérie, Argentine, E.A.U., France (siège), Indonésie, Jordanie, Maroc, Pologne, Thaïlande, Tunisie et Vietnam.

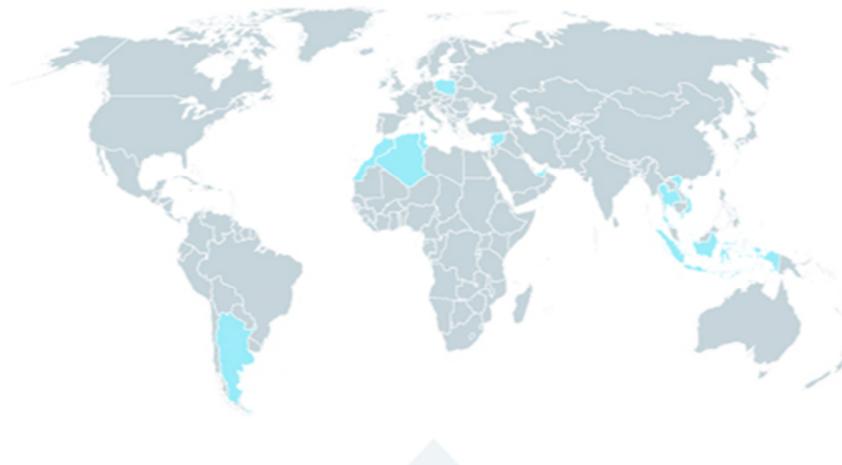


Figure 2 SOFRECOM dans le monde

- Plus de 1000 consultants experts
- Une culture internationale avec plus de 30 nationalités représentées
- Certifications ISO 9001 version 2008 et CMMI niveau 2
- Certification Ethic Intelligence 2011

Riche de ses références dans plus de 100 pays, d'une véritable culture opérateur et de la force du groupe France Télécom-Orange, SOFRECOM associe à sa parfaite connaissance du métier d'opérateur une expertise focalisée sur l'ensemble des enjeux des technologies de l'information et de la communication. SOFRECOM aborde chaque projet en mettant en perspective l'évolution des marchés, les problématiques métiers, l'intensification de la concurrence et les technologies d'avenir.

C'est dans cet esprit que son activité a été organisée en quatre secteurs :

- Marketing & distribution
- Organisation & process
- Systèmes d'information
- Réseaux & services

1.1 Historique

Depuis plus de 50 ans, SOFRECOM a développé aux quatre coins du monde un savoir-faire et des expertises uniques qui en font aujourd'hui un leader du conseil et de l'ingénierie télécoms. Son expérience à l'international et son appartenance au groupe France Télécom-Orange ont façonné son histoire.

SOFRECOM, a été fondée en 1966 sous l'impulsion de l'État français, et sous couvert de la Direction Générale des Télécommunications, le futur opérateur historique France Télécom.

CHAPITRE I : Cadre du projet et étude de l'existant

Dès les années 80, les consultants et les ingénieurs de SOFRECOM sont présents dans une quarantaine de pays, en Afrique, en Asie, au Moyen-Orient, en Océanie. Dans le même temps, l'historique Direction Générale des Télécommunications devient France Télécom.

En 1985, SOFRECOM devient une filiale à 100% de France Télécom, qui souhaite structurer son savoir-faire à l'international. SOFRECOM joue un rôle actif dans la mise en œuvre des licences mobiles acquises par France Télécom dans les années 90 en Europe et en Afrique. Ils travaillent également en Amérique Latine, où ils créent leur première filiale, SOFRECOM Argentine, en 1992.

The Know-How Network Fort de son histoire et de son réseau international d'experts SOFRECOM rentre dans une nouvelle dynamique. Groupe international, ils mènent une politique de développement local ambitieuse pour s'intégrer parfaitement dans l'écosystème de leurs clients. Son identité de marque évolue, avec un nouveau logo et une nouvelle signature « The Know-How Network » pour répéter plus simplement ce qu'elle représente.

1.2 SOFRECOM Tunisie

SOFRECOM Tunisie est la plus jeune filiale de SOFRECOM. Lancée en novembre 2011, SOFRECOM Tunisie étend la présence de SOFRECOM sur la zone Afrique du Nord et Moyen Orient afin de répondre à une demande croissante de solutions dédiées et de proposer à ses clients des offres adaptées et compétitives en tant que Centre de développement et d'expertise pour les plates-formes de services, SOFRECOM Tunisie s'inscrit en complémentarité des domaines d'activité des autres implantations de SOFRECOM dans la région et permettra aux clients du groupe SOFRECOM de bénéficier des synergies entre les filiales.

1.3 Organisation

La figure suivante nous donne une idée sur l'organigramme de SOFRECOM Tunisie

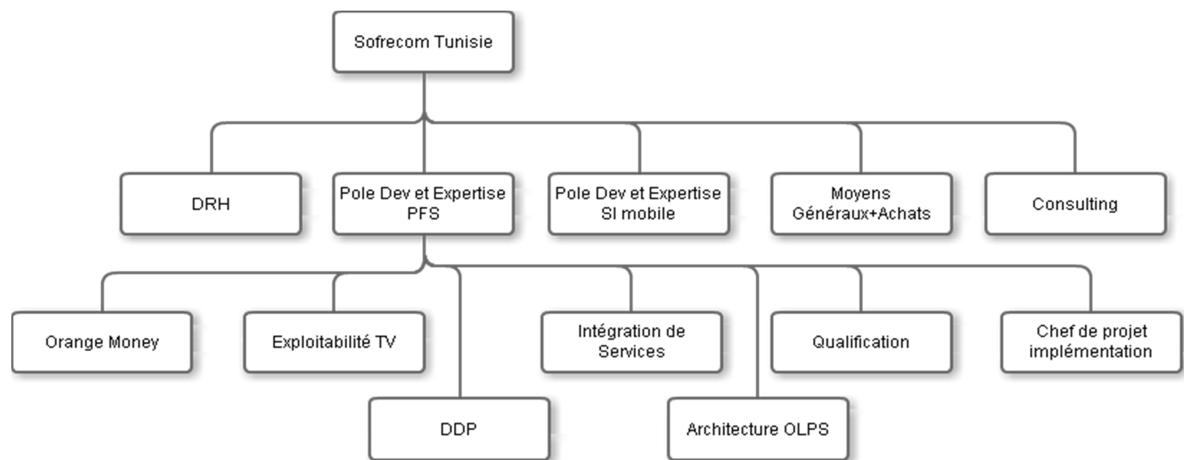


Figure 3 Organigramme de SOFRECOM Tunisie

CHAPITRE I : Cadre du projet et étude de l'existant

L'entreprise comporte maintenant un pôle principal qui est le pôle « Développement et expertise DFS ». Ce pôle regroupe les activités principales de la filiale :

- DDP : département de développement
- Orange Money
- Intégration de Services
- Architecture OLPS Tunisie : regroupe les architectes transverses, architectes et experts Zebra, Architectes et experts OM/MMM et les IN Architectes.
- Exploitabilité TV
- Qualification

1.4 Direction d'accueil

Mon stage est effectué au sein du département « OLPS : Orange Labs Products and Services » qui est constitué d'environ 150 collaborateurs, et précisément au sein de l'équipe OOPRO Tunis qui prend en charge le développement, l'intégration et la qualification du produit OOPRO. C'est une équipe constituée de 16 personnes :

- un manager
- un chef projet
- 14 personnes Techniques : Installation, intégration, packaging, tests techniques, support aux exploitants, Automatisation des tests, qualification fonctionnelle.
- Un gourou

L'équipe traite toutes les aspects de développement, intégration et qualification du produit OOPRO et ce en étroite collaboration avec le reste de l'équipe basée à Grenoble.

2 Présentation du projet

Dans cette section, nous allons présenter le projet OOPRO, le module qui nous intéresse et la méthode de travail adopté au sein de ce projet.

2.1 L'OOPRO

L'OOPRO « Office Orange Pro » est un produit orange destiné au marché des petits professionnels. Il s'agit d'une suite de communication :

- mail
- calendrier
- carnet d'adresses
- tâches

- transfert de fichiers
- stockage
- envoi de sms/mms/fax

C'est une offre incluse dans l'offre internet pro mais avec des options qui sont payantes.

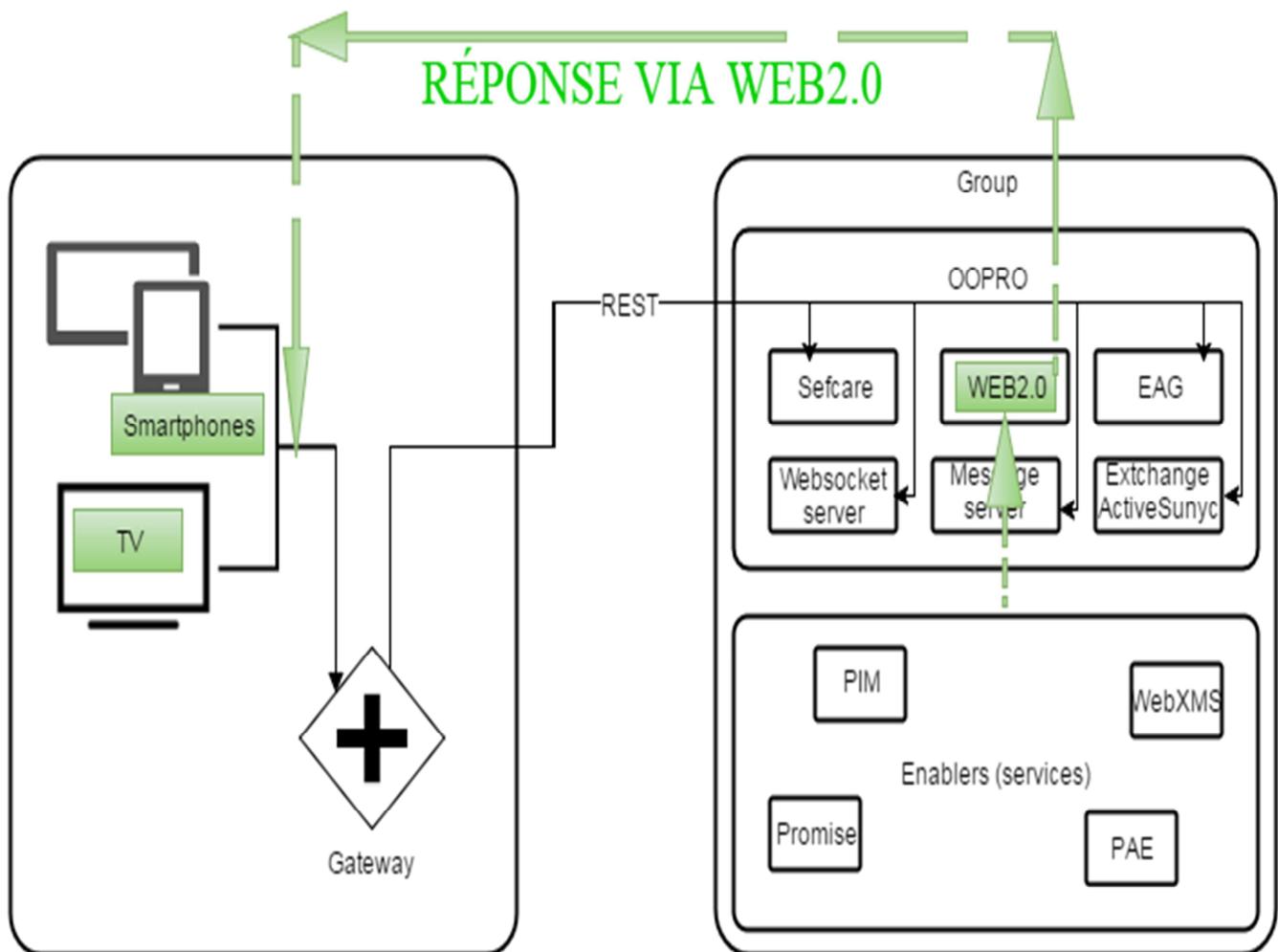


Figure 4 Schéma de l'architecture d'OOPRO

OOPRO, c'est où ? :

- Dix pays :
 - France, Belgique, Jordanie, Bahreïn, Maroc, Sénégal, Côte d'Ivoire, Madagascar, Ouganda, Niger.
- Quatre langues :
 - Français, anglais, néerlandais, arabe.

OOPRO, c'est pour qui ? :

- France : 1,2 Millions clients
- 2,5 Millions BAL dont 70% utilisées

40.000 utilisateurs de fax

40.000 utilisateurs de stockage

16.000 utilisateurs de SMS/MMS

330.000 connexions par jour

- BU monde :

2.000 clients et 4.800 utilisateurs

OOPRO, c'est fait par qui ? :

Environ 100 personnes travaillent sur cette offre (marketing, maîtrise d'œuvre, développement, qualification, production ...).

2.2 Le module Web2

Le WEB2.0 est le module messagerie Pro, il s'agit d'une application Web mail « Responsive Design » s'adaptant à tous les terminaux. Cette interface est en cours de migration vers la version 3.0 où l'ergonomie est remarquablement améliorée.



Figure 5 WEB3 Responsive Design

2.3 Approche Agile

Les méthodes agiles sont des groupes de pratique pouvant s'appliquer à divers types de projets, mais actuellement elles sont plutôt dédiées à la gestion de projets informatiques. Elles reposent sur des cycles de développement itératifs et adaptatifs en fonction des besoins évolutifs du client. Elles permettent notamment d'impliquer l'ensemble des collaborateurs ainsi que le client dans le développement du projet. [0]

Ces méthodes permettent généralement de mieux répondre aux attentes du client en un temps limité (en partie grâce à l'implication de celui-ci) tout en faisant monter les collaborateurs en compétences.

CHAPITRE I : Cadre du projet et étude de l'existant

Ces méthodes constituent donc un gain en productivité ainsi qu'un avantage compétitif tant du côté client que du côté du fournisseur.

L'équipe OOPRO adopte les méthodes SCRUM et Devops.

- **SCRUM**

Scrum est une méthode agile de gestion de projets utilisée pour organiser les équipes et améliorer la productivité avec une meilleure qualité [1]. Elle est une méthode itérative et incrémentale permettant la réalisation d'un ensemble de fonctionnalités dans une itération d'une durée entre 2 à 4 semaines. Dans le jargon SCRUM une itération est planifiée dans un Sprint. Dans cette méthode le client participe activement dans le cycle de développement de projet, ce qui permet :

- La définition des fonctionnalités prioritaires.
- L'ajout de fonctionnalités en cours de projet (pas pendant un sprint).

Afin de comprendre cette méthode agile, il est nécessaire de définir un ensemble de mots ainsi qu'un ensemble de rôles à partager au sein de l'équipe.

Le tableau présente les mots clés et leurs définitions.

Mot clé	Description
User Story	Description littérale (non technique) d'une fonctionnalité donnée.
Produit BackLog	Définition des besoins fonctionnels sous forme de « User Story ».
Sprint	C'est une liste des « User Story ». Chaque itération définit un objectif à atteindre, en lui associant une liste d'éléments du « Product BackLog » à réaliser.
Task	Un « User Story » se décompose en un ensemble de tâches élémentaires. La durée maximale d'une tâche ne dépasse pas (obligatoirement) deux jours. Dans les cas contraires la tâche est décomposée en autres tâches.
Vélocité	Vitesse du progrès : Une vitesse réelle (Temps réel d'atteinte), et la Vitesse estimée (Temps au bout duquel l'atteinte est estimée).
Story Point	Estimation du travail nécessaire pour accomplir un « User Story ».
Burndown Chart	Graphique de ré-estimation du « Reste A faire ».

Tableau 1 Définition des mots clés

Le tableau suivant présente les différents rôles dans la méthode SCRUM ainsi que leurs descriptions.

Rôle	Description
Product Owner	C'est le directeur de produit. Il est le représentant des clients et utilisateurs. C'est lui qui définit l'ordre dans lequel les fonctionnalités seront développées et qui prend les décisions importantes concernant l'orientation du projet. C'est lui aussi qui formalise le backlog du produit.
SCRUM Master	Il travaille de façon rapprochée avec le Product Owner et l'équipe. Il joue le rôle d'un facilitateur et d'un coach plus que le rôle d'un superviseur.
Development Team	Représente l'équipe. Elle ne comporte pas de rôles prédefinis, elle est autogérée : les membres de l'équipe prennent les décisions ensemble et personne ne donne l'ordre à l'équipe sur sa façon de procéder.
Stakeholders	Ce sont les intervenants : les personnes qui souhaitent avoir une vue sur le projet sans réellement s'investir dedans.

Tableau 2 Définition des rôles au sein d'une équipe

Pour la planification, SCRUM propose une planification opérationnelle à trois niveaux :

release/projet, sprint et quotidien.

- Sprint :

Scrum est un processus itératif : les itérations sont appelées des sprints et durent en théorie 30 jours calendaires. En pratique, les itérations durent généralement entre 2 et 4 semaines. Chaque sprint possède un but et on lui associe une liste d'items de Backlog de produit (fonctionnalités) à réaliser. Ces items sont décomposés par l'équipe en tâches élémentaires de quelques heures, les items de Backlog de sprint.

- Release :

Pour améliorer la lisibilité du projet, on regroupe généralement des itérations en releases. En effet, comme chaque sprint aboutit à la livraison d'un produit partiel, un release permet de marquer la livraison d'une version aboutie, susceptible d'être mise en exploitation.

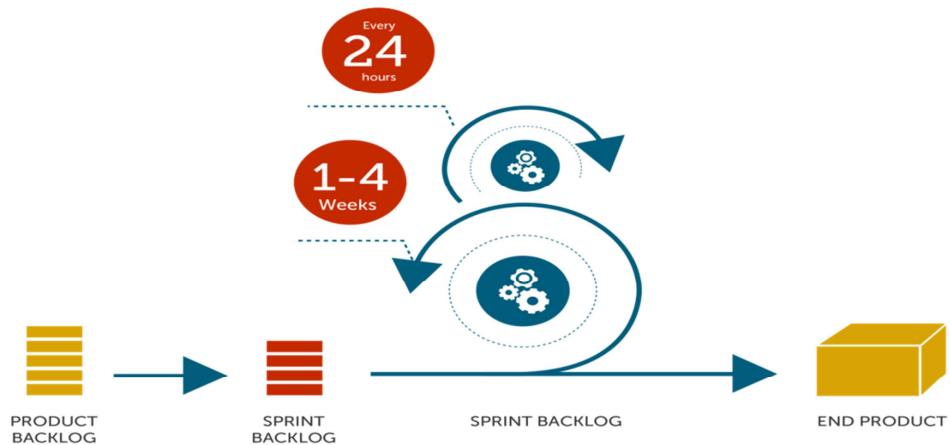


Figure 6 Méthode Scrum

- Devops

Le "DevOps" est un mouvement qui vise à réconcilier et synchroniser le développement (agile) et la mise en production (immédiate) à travers un ensemble de bonnes pratiques. Son émergence est motivée par une évolution profonde des demandes des métiers, qui veulent accélérer les changements pour coller au plus près aux exigences du business et du client. Pour s'adapter, les développeurs doivent faire plus de modifications de taille plus petite. C'était déjà le sens des méthodes agiles qui ont permis de produire rapidement des versions de logiciels correspondantes, de façon optimale, à l'exigence métier. Mais l'infrastructure n'était pas forcément organisée pour suivre le rythme. Le DevOps vise à étendre les cycles courts jusqu'à la mise en production.

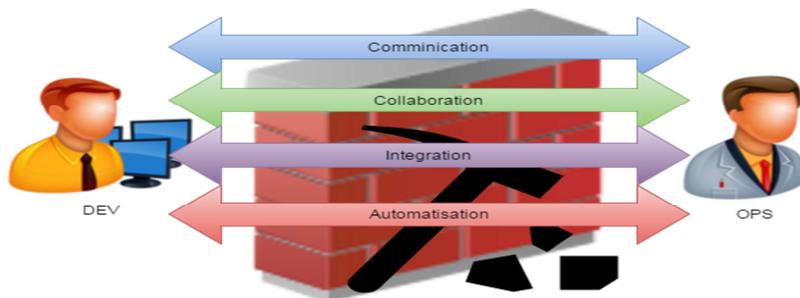


Figure 7 Devops

Le Devops vise à rassembler les développeurs, les exploitants métiers .Comme il n'y a plus de déroulé linéaire (expression des besoins, développement et mise en production), tous les acteurs doivent être présents dès le début. Il s'agit donc de regrouper les équipes de développement et d'exploitation, sans pour autant les installer physiquement sur un même plateau, car l'externalisation d'une partie du développement ou de l'exploitation doit être possible.

3 Etude de l'existant

Dans cette partie, nous allons présenter une étude de l'existant au sein de l'entreprise d'accueil pour pouvoir bien dégager les besoins et la problématique.

3.1 Migration du module WEB2

Comme nous venons de le présenter, le WEB3.0 est le module messagerie Pro du produit OOPRO de Orange. Ce module est considéré comme la partie la plus importante du produit car il représente son interface homme-machine qui va être commercialisé. Cette dénomination de X.0 s'est découlée du fait que ce module-là est en cours de migration d'une version 2.0 vers une nouvelle qui est 3.0.

La version existante sur le marché aujourd'hui est la WEB2.0, elle n'est pas conçue et adaptée pour la nouvelle génération des supports physiques (Smartphone, Tablet, etc.) C'est-à-dire qu'elle n'est pas « Responsive Design ».

3.1.1 Le WEB3 est un site web adaptatif

Un site web adaptatif (en anglais : responsive web design) est un site web dont la conception vise à offrir une expérience de consultation confortable pour des supports différents. L'utilisateur peut ainsi consulter le même site web à travers une large gamme d'appareils (moniteurs d'ordinateur, Smartphones, tablettes, TV, etc..) avec le même confort visuel sans pour autant avoir recours au défilement horizontal ou au zoom avant et zoom arrière, notamment sur les appareils tactiles.

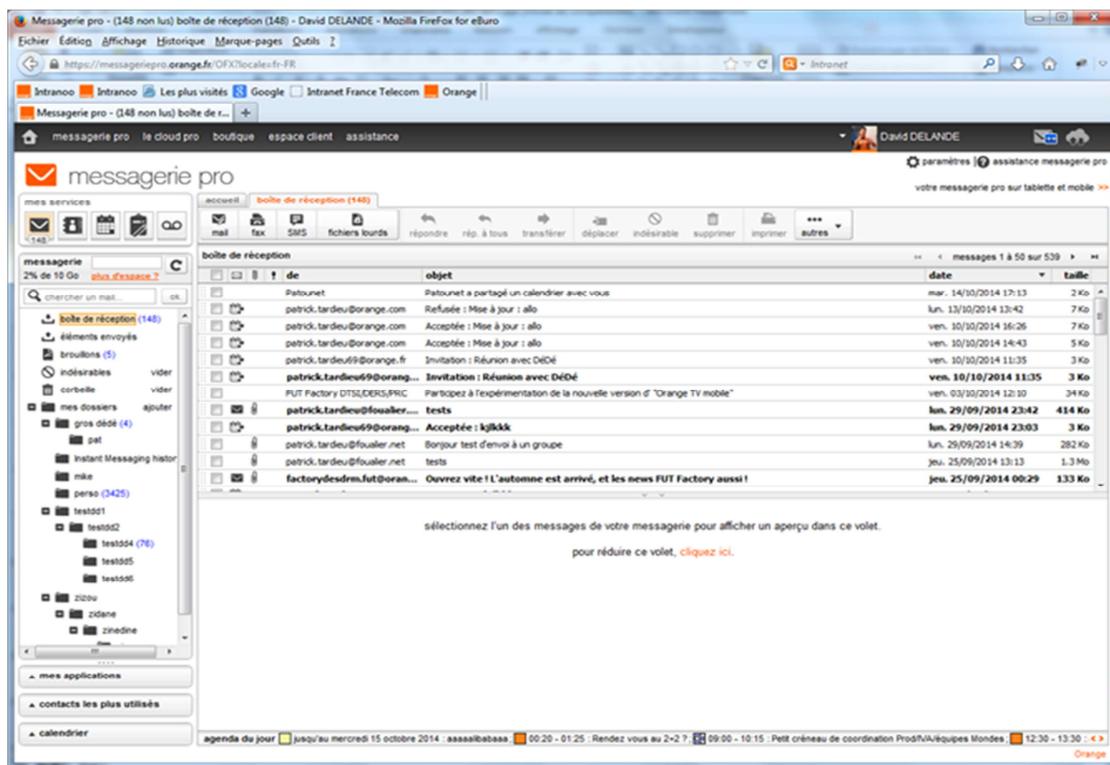


Figure 8 WEB2.0

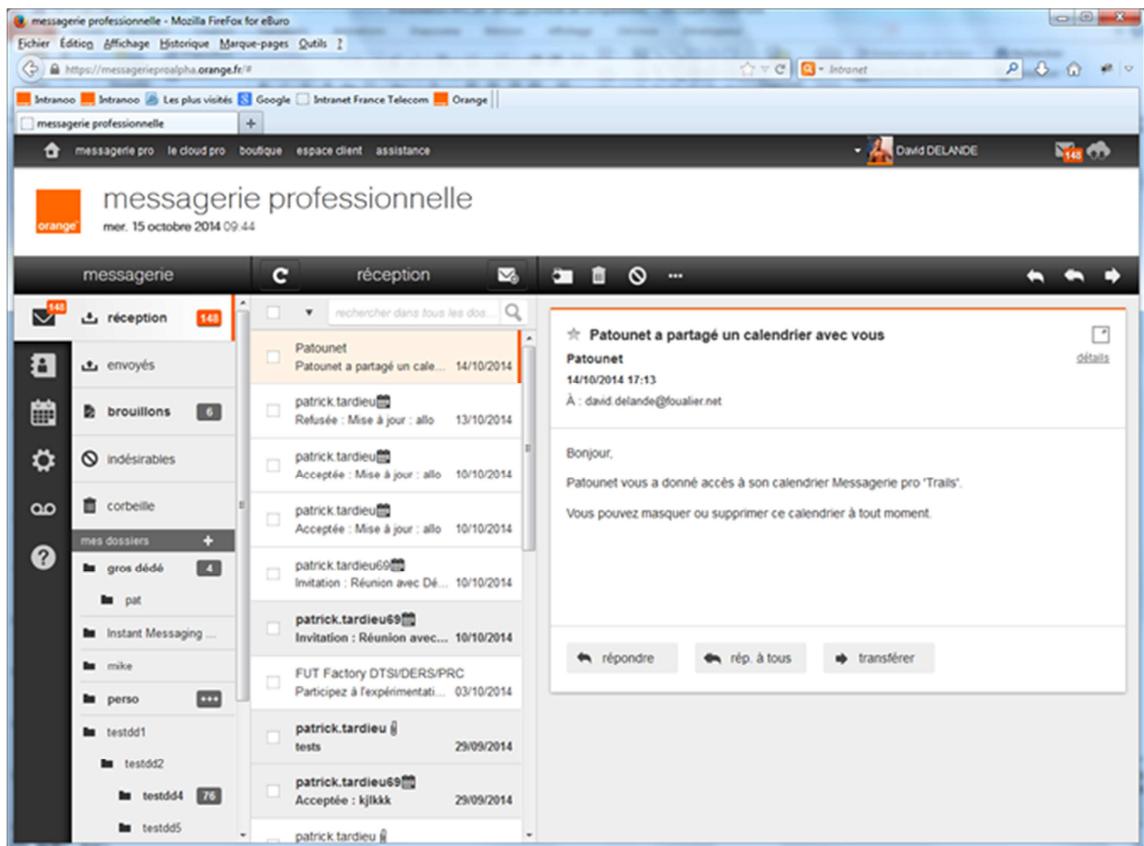


Figure 9 WEB3.0

3.2 Les tests de non-régression IHM du module WEB3

Beaucoup de régressions sont apparues lors du développement de la nouvelle interface et des nouvelles fonctionnalités ajoutées. Ainsi le besoin des tests de non régression IHM est inévitable. L'équipe OOPRO se base sur les tests de non-régression IHM. Pour clôturer le cycle du test de bout en bout, ils disposent d'un arsenal de test constitué de 7000 cas de tests à jouer pour pouvoir valider/invalider chaque release (version admissible). Cette activité consiste à vérifier « ce que l'on ne voulait pas changer fonctionne toujours comme avant ». Cela représente en pratique 30 % du volume des tests réalisés sur le WEB3.0. Les tests de non-régression ne peuvent être conduits qu'une fois l'application soit livrée et ainsi, ils se trouvent sur le chemin critique du projet.

3.2.1 Les tests de non-régressions IHM en cours d'optimisation

L'équipe OOPRO opte pour optimiser les Tests de Non-Régression IHM à trois niveaux :

- « en amont » au moment où on les conçoit.
- « pendant » lorsqu'on les exécute.
- « après ».

En amont, la première étape d'optimisation vise à gérer les exigences qui sont de bon niveau et qui permettent d'optimiser les analyses d'impacts. En effet, lorsqu'on a une mauvaise analyse d'impacts,

on crée des problèmes de non-régression ayant des anomalies produites sur des zones qu'on ne doit pas toucher.

La deuxième source d'optimisation concerne la structuration du patrimoine des tests, dans lequel on cherche à avoir des attributs de tri qui permettent de sélectionner les scénarios en fonction des risques pris vis-à-vis des évolutions et de la non-régression. Par exemple, l'équipe OOPRO utilise une structuration des scénarios sur deux volés :

- Des scénarios très détaillés « NonRegFull » : qui touchent en détails toutes les fonctionnalités de l'application, ce qui semble beaucoup à un Brut-force des fonctionnalités de l'application.
- Des scénarios moins détaillés « NonRegLight » : qui ne touche que les fonctionnalités de base de l'application.

3.3 Les tests de non régression IHM réalisés d'une manière manuelle

L'équipe OOPRO utilise maintenant l'outil QC « Quality Center » de gestion des tests pour assurer la qualité des tests d'où de l'application.

« HP Quality » Center est une plateforme de gestion de qualité logicielle offerte de la division « HP de Hewlett-Packard » avec beaucoup de capacité acquise de « Mercury Interactive Corporation ». « HP Quality Center » offre l'assurance de la qualité des logiciels y compris la gestion des exigences, gestion des tests pour l'informatique et les environnements d'application. « HP Quality Center » est un composant de la « HP application Lifecycle Management » qui est un ensemble de solutions logicielles.



Figure 10 HP QC

Mais malheureusement cela ne suffit pas, car on exécute, encore, manuellement les cas du test. C'est une procédure très lente et couteuse qui fait prolonger la durée du Sprint.

4 Problématique

L'équipe de qualification d'OOPRO se voit aujourd'hui projetée sur le devant de la scène. En effet, le contexte du marché très tendu, l'accélération de leur compétitivité, mais également la part grandissante des tests de non régression dans les coûts du développement (entre 30% et 40%) génèrent une forte

pression sur l'équipe. De plus les Tests de non régression sont importants en termes de délai et de time-to-Market. L'automatisation des tests de non régression devient donc un enjeu stratégique.

Comment peut-on optimiser et automatiser les tests de régression IHM en phase d'exécution ?

5 Objectif du projet

Mon projet s'intéresse à l'étude et à la création d'un processus d'automatisation des tests de non régression IHM, ainsi que le développement d'une interface qui permet de simplifier la manipulation d'un tel processus afin d'aller beaucoup plus vite qu'en mode manuel. Cependant quelques prérequis sont nécessaires en termes de faisabilité technique et de stabilité pour éviter trop de maintenance.

Conclusion

A travers ce chapitre, nous avons décrit le contexte du projet et nous avons présenté le projet OOPRO et la méthodologie respecté par l'équipe OOPRO ainsi qu'une étude de l'existant. Nous avons aussi présenté la problématique et l'objectif de mon projet de fin d'études. Dans le chapitre suivant, nous allons présenter une étude théorique du métier informatique dont lequel notre solution va s'introduire, ainsi nous décrivons les notions des tests et leurs différents types, nous allons présenter une étude comparative pour bien choisir les outils qui vont être les éléments de réponse pour la problématique



CHAPITRE II

ÉTAT DE L'ART

État de l'art

Introduction

Avant de nous lancer dans notre projet, une étude théorique, spécifique et rigoureuse s'impose. Ensuite nous nous focaliserons sur l'étude du processus existant et nous poserons la problématique à laquelle nous allons répondre dans les chapitres qui suivent.

1 Le test logiciel

En informatique, un test désigne une procédure de vérification partielle d'un système. Son objectif principal est d'identifier un nombre maximal de comportements problématiques du logiciel afin d'en augmenter la qualité (si les problèmes identifiés lors des tests sont corrigés). Néanmoins, le test peut aussi avoir pour objectif d'apporter des informations quant à cette qualité afin de permettre la prise de décisions.

Les tests de vérification ou de validation visent, respectivement, à vérifier que le système considéré réagit comme prévue par ses développeurs (spécifications) ou qu'il soit conforme aux besoins du client.

1.1 Vérification et Validation (V & V)

La vérification et la validation (les tests) sont deux disciplines de la qualité logicielle. Dans le cas où ces techniques, soient adoptées et appliquées rigoureusement et avec intelligence, dès le début d'un projet, ils peuvent améliorer significativement la qualité logicielle.

- Vérification : est-ce que le logiciel fonctionne correctement ?
- Validation : est-ce que le logiciel fait ce que le client veut ?

Mais pourquoi les logiciels échouent en termes de qualité ?

- Fausses exigences (pas ce que le client veut ou exigences manquantes)
- Exigence impossible à mettre en œuvre
- Conception défectueuse
- Code défectueux

Les objectifs :

- Identification des défauts : quelle faute a causé l'échec?
- La correction des défauts : changer le système
- Élimination des défauts : enlever la faute

1.2 Les méthodes des tests

- **Test boite noir**

Cette méthode où le test se déroule sans avoir aucune connaissance du fonctionnement intérieur de l'application, est appelée test boîte noire. Le testeur est inconscient de l'architecture interne du système et n'a pas accès au code source. Typiquement, en effectuant un test de boîte noire, un testeur va interagir avec l'interface du système en fournissant des entrées et en examinant les sorties sans savoir comment les entrées sont traitées.

Le tableau suivant présente les avantages et les inconvénients des tests de boîte noire :

Avantages	Inconvénients
bien adapté et efficace pour les grands segments de code.	La couverture des tests est limitée, puisque seul un certain nombre de scénarios de test est effectivement réalisé.
l'accès au code source du logiciel n'est pas nécessaire.	les tests sont inefficaces, puisque le testeur ne peut pas cibler un segment de code spécifique pour tester son fonctionnement.
N'importe quel testeur qualifié peut tester l'application sans avoir des connaissances du langage de programmation du logiciel ni le système d'exploitation sur lequel il fonctionne.	

Tableau 3 Les avantages et les inconvénients de test boîte noir

- **Test boîte blanche**

Cette méthode vise à analyser un programme informatique dont on connaît la logique interne. Afin d'effectuer des tests boîte blanche sur un programme, le testeur a besoin de savoir le fonctionnement interne du code. Le testeur a besoin d'analyser le code source pour savoir quelle unité de code se comporte d'une façon inappropriée.

Le tableau suivant présente les avantages et les inconvénients des tests boîte blanche :

Avantages	Inconvénients
Comme le testeur a connaissance du code source, il devient très facile à savoir quel type des données peut lui aider à tester l'application efficacement.	Du fait qu'un testeur qualifié est nécessaire pour effectuer les tests boîte blanche, les coûts augmentent.
Cette méthode permet d'optimiser le code source du programme.	Parfois, il est difficile d'analyser le code pour trouver les erreurs qui peuvent créer problèmes.
Comme le testeur connaît très bien le code source, il peut couvrir le maximum des cas de tests lors de l'écriture d'un scénario de test.	

Tableau 4 Les avantages et les inconvénients de test boîte blanche

- **Test boîte grise**

Cette méthode du test requiert quelques connaissances sur le fonctionnement interne et la plateforme du programme informatique testé. Maîtriser le domaine du système donne toujours au testeur un avantage lors de sa mission. Contrairement aux tests boîte noire où le testeur ne teste que l'interface du système, dans les tests boîte grise le testeur a accès aux documents de conception et à la base de données. Ayant cette connaissance, un testeur peut préparer les meilleures données et scénarios des tests lorsqu'il vise à établir leurs plans.

Le tableau suivant présente les avantages et les inconvénients des tests boîte grise :

Avantages	inconvénients
Offre les avantages du test boîte blanche et du test boîte noire.	Vu que l'accès au code source est indisponible, la capacité de couvrir tous les cas de tests est limitée.
Le testeur ne compte pas sur le code source, au contraire, il compte sur la spécification fonctionnelle du programme.	Les tests peuvent être redondants (chevauchement avec d'autres cas de tests réalisées par d'autres testeurs)
Sur la base des informations disponibles, le testeur peut concevoir scénarios de test efficaces.	

Tableau 5 Les avantages et les inconvénients de test boîte grise

1.3 Comparaison entre les méthodes de test

Le tableau suivant présente la liste des différences entre les tests boîte noire, les tests boîte grise et les tests boîte blanche.

Test boîte noire	Test boîte grise	Test boîte blanche
Le fonctionnement interne du programme devrait être inconnu.	Le testeur a une connaissance limité du fonctionnement interne du programme.	Le testeur devrait avoir une connaissance totale du fonctionnement interne.
Peut être effectué par les utilisateurs finaux et les testeurs.	Peut être effectué par les utilisateurs finaux, les testeurs et les développeurs.	Peut être effectué par les testeurs et les développeurs.
Simple à réaliser et consomme peu de temps.	Plus ou moins simple et consomme plus de temps que le test en boîte noire.	C'est la méthode de test la plus complexe à effectuer et il consomme beaucoup de temps.

Tableau 6 Comparaison entre les méthodes de tests

2 Les niveaux d'un test logiciel

Le tableau suivant présente les différents niveaux d'un test logiciel.

Type de test	Spécification	Portée générale	Opacité	Qui le fait ?
Unitaire	La conception de bas-niveau & code réel	Classes	Boite blanche	Programmeur
Intégration	La conception de bas-niveau & La conception de haut-niveau	Plusieurs Classes	Boite blanche & Boite noir	Programmeur
Fonctionnel	La conception de haut-niveau	Produit entier	Boite noir	Testeur indépendant
System	Analyse d'exigences	Tout le produit dans son environnement	Boite noir	Testeur indépendant
Acceptation	Analyse d'exigences	Tout le produit dans son environnement	Boite noir	Client
Régression	Documentation modifiée & La conception de haut-niveau	Aucun de ce qui précède	Boite noir & Boite blanche	Programmeurs ou Testeurs indépendants

Tableau 7 Niveaux de tests

2.1 Tests fonctionnels

Les tests fonctionnels vérifient que le comportement d'un système en boîte noire soit conforme à ses spécifications. L'application est testée en introduisant des données comme entrées puis en examinant les résultats et en vérifiant que ces derniers sont conformes à la fonctionnalité qu'on vise à tester. Les tests fonctionnels sont effectués généralement sur un système ou toutes ses fonctionnalités sont complètes pour évaluer sa conformité avec ses exigences spécifiées. Il y a cinq étapes qui sont impliquées lorsqu'on teste un système fonctionnellement :

Le tableau suivant présente les différentes étapes du test fonctionnel.

étape	description
1	La détermination de la fonctionnalité que le système, à tester, doit réaliser.
2	La détermination des données du test en fonction des spécifications du système à tester.
3	Les résultats sont basés sur les données du test et sur les spécifications du système.
4	L'écriture de scénarios du test et l'exécution des cas du test.
5	La comparaison des résultats réels et prévus, est basée sur les cas du test exécutés.

Tableau 8 Pyramide de test

2.2 Test unitaire

Ce type de test permet au développeur de s'assurer qu'une unité de code ne comporte pas d'erreur de programmation. Les tests unitaires sont effectués par les développeurs avant que le code soit remis à l'équipe du test pour exécuter formellement les cas du test. Le but des tests unitaires est d'isoler chaque partie du programme et de montrer que les unités de code sont correctes en termes d'exigences et de fonctionnalités.

2.3 Test d'intégration

Les tests d'intégrations sont définis comme étant le test des parties combinées d'une application pour déterminer si elles fonctionnent correctement. Les tests d'intégration peuvent se faire en deux manières : soit d'une manière ascendante ou bien d'une manière descendante.

En choisissant la manière ascendante pour faire les tests d'intégrations, on doit commencer par faire les tests unitaires suivie par des combinaisons progressives de ces tests appelés des modules (à chaque fois on combine les tests précédents pour tester les modules résultants). En choisissant la manière descendante pour faire les tests d'intégrations, on teste les modules de plus hauts niveaux suivie par les tests des niveaux plus bas jusqu'à atteindre des modules unitaires. Dans un environnement de

développement logiciel, les tests ascendants sont habituellement faits en premier, suivie par les tests descendants.

2.4 Test système

Les tests système de logiciel testent le système en son ensemble. Une fois que tous les composants du système sont intégrés, on peut tester ce dernier pour évaluer sa conformité aux exigences spécifiées. Les tests système appartiennent à la classe des tests de type boîte noire puisqu'ils n'exigent aucune connaissance du fonctionnement interne du système. Ce type de test est effectué par une équipe de test spécialisée.

Les tests système sont importants pour les raisons suivantes :

- Le système est soigneusement testé pour vérifier qu'il répond à ses spécifications fonctionnelles techniques.
- Le système est testé dans un environnement qui est très proche de son environnement de production où il sera déployé.
- Les tests système nous permettent de tester, de vérifier et de valider le système à la fois sur le plan fonctionnel ainsi que sur le plan architectural.

2.5 Test de régression

Chaque fois qu'un changement dans un programme informatique est fait, il est possible que d'autres domaines au sein de l'application (ajout de nouvelles fonctionnalités, modification de fonctionnalités existantes ou tout composant qui intervient dans le fonctionnement du système) aient été touchés par ce changement. Les Tests de régression sont effectués pour vérifier qu'un changement au niveau d'une fonctionnalité n'a pas eu lieu dans une autre. Le but des tests de régression est d'assurer qu'un changement, comme une correction de bug, ne devrait pas mener à des défauts dans les parties non modifiées du programme. Ces tests sont effectués quand le logiciel ou son environnement est modifié.

2.6 Test d'acceptation

Ce test est mené par l'équipe d'assurance qualité qui permet d'évaluer si le système répond à ses spécifications et s'il répond aux exigences du client. L'équipe d'assurance qualité dispose d'un ensemble de scénarios pré-écrits et des cas de test qui sera utilisé pour tester l'application. Les tests d'acceptation ne sont pas destinés seulement, à signaler les erreurs d'orthographe simples ou les lacunes de l'interface, mais aussi à signaler les bugs du système qui se traduira par le plantage du système ou des erreurs majeures dans ce dernier.

2.7 Test de performance

Il est principalement utilisé pour identifier les goulots d'étranglement et les problèmes de performance plutôt que trouver des bugs dans un logiciel. Il y a différentes causes qui contribuent à abaisser l'exécution d'un programme dont on peut citer :

- Le traitement des transactions côté base de données
- L'équilibrage de charge entre les serveurs

Le test de performance est considéré comme l'un des tests obligatoires pour tester le programme au niveau des aspects suivants :

- vitesse (temps de réponse, l'accès aux données...)
- capacité
- stabilité

2.8 Test de sécurité

Le test de sécurité consiste à tester un programme afin d'identifier les défauts et les lacunes du point de vue sécurité et vulnérabilité. Les principaux aspects que le test de sécurité doit assurer sont :

- Confidentialité
- Intégrité
- Authentification
- Disponibilité
- Autorisation
- La non-répudiation

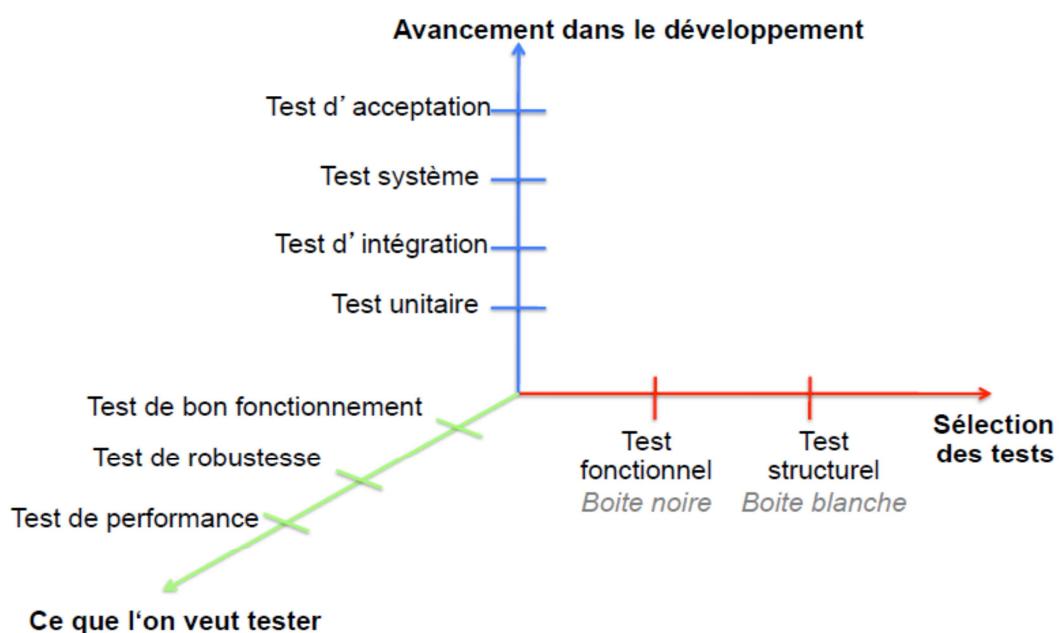


Figure 11 Les axes des tests

3 Documentation du test

La documentation des tests logiciels aide à estimer l'effort nécessaire pour le test, la couverture du test et le suivi des besoins d'un test. Cette section décrit certains documents couramment utilisés dans les tests logiciels.

3.1 Le plan de test

Un plan de tests décrit la stratégie qui sera utilisée pour tester un programme, les ressources qui seront utilisées, l'environnement de test dans lequel les tests seront réalisés, et les limites des tests et le calendrier de l'activité de test. Typiquement, le chef d'équipe d'assurance qualité sera responsable de la rédaction d'un plan de test.

Un plan de test comprend les éléments suivants :

- Introduction au document de plan de test.
- Hypothèses pris lors de l'activité de test.
- La liste des cas de test inclus dans l'activité de test.
- La liste des fonctionnalités à tester.
- Quelle approche utiliser pour tester le programme.
- Liste des livrables qui doivent être testés.
- Les ressources allouées pour tester le programme.
- Les risques encourus au cours du processus de test.
- Un calendrier des tâches et les étapes à accomplir.

3.2 Le Scénario de test

C'est la procédure à suivre par le testeur pour exécuter le cas de test : manipulations à effectuer, dialogue homme/machine etc...

Un scénario de test répond à la question suivante : "Que vais-je tester ?".

En développant des scénarios de test, vous définissez les éléments que vous devez valider afin que le système fonctionne correctement et offre un niveau de qualité optimale.

3.3 Le Cas de test

C'est le chemin fonctionnel à mettre en œuvre pour atteindre un objectif de test. Un cas de test se définit par le jeu de test (données en entrée d'un cas de test) à mettre en œuvre, le scénario de test à exécuter et les résultats attendus. Le but principal de cette activité est de vérifier si un programme passe ou échoue en termes de fonctionnalité et d'autres aspects (aspects techniques, aspects de performance et de sécurité

etc...). Il y a de nombreux types de cas de test tels que fonctionnel, négatif, erreur, cas de tests logiques, les cas de tests physiques et des cas de test de l'interface utilisateur etc...

Les cas de test sont écrits pour garder la trace de la couverture de test d'un logiciel. Généralement il n'y a pas de modèles formels qui peuvent être utilisés lors de l'écriture de cas de tests. Cependant, les composants suivants sont toujours inclus dans tous les cas de test :

- L'identifiant du cas de test
- Le module du produit à tester
- La version du produit à tester
- Objectif
- Hypothèses
- Conditions préalables
- Résultat attendu
- Résultat réel
- Post-conditions

4 Les Tests dans un processus itératif SCRUM

SCRUM est un processus itératif qui concentre dans des itérations courtes (2 à 3 semaines en général) toutes les activités de la production logiciel. Si on se concentre sur les activités des tests, on remarque que tous les types de test sont passés lors de chaque itération. De ce fait, les tests ne sont pas relégués à la toute fin de processus, mais intégrés dans celui-ci. De plus comme le logiciel est livré au Product-Owner à chaque itération il peut (et même doit) tester lui aussi la livraison pour pouvoir faire ses retours le plus rapidement possible. [2]

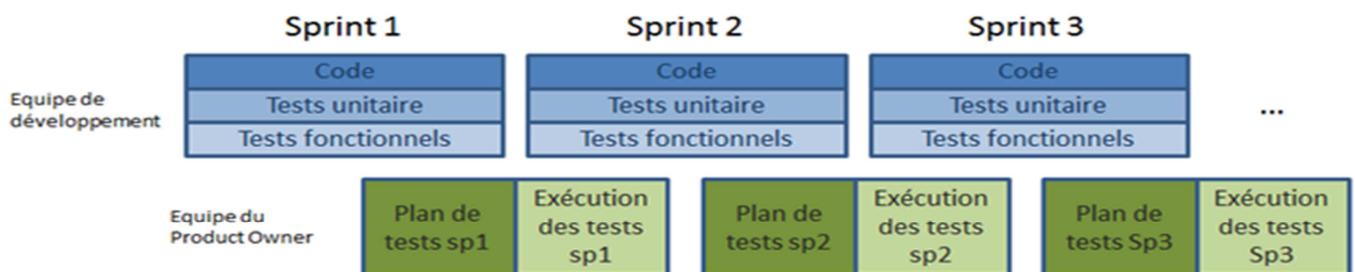


Figure 12 Le test SCRUM

Quand on dit que l'Agilité permet de réduire l'effet tunnel, on pense souvent au côté fonctionnel, mais l'effet tunnel affecte aussi, le niveau de qualité de la production, et là encore, l'Agilité apporte une solution véritable. Une autre particularité de SCRUM est que les développements sont priorisés par rapport à leur valeur métier, et non pas par rapport aux priorités techniques. Cette particularité a un impact sur les tests. En effet, cette façon d'ordonner les développements oblige les développeurs à

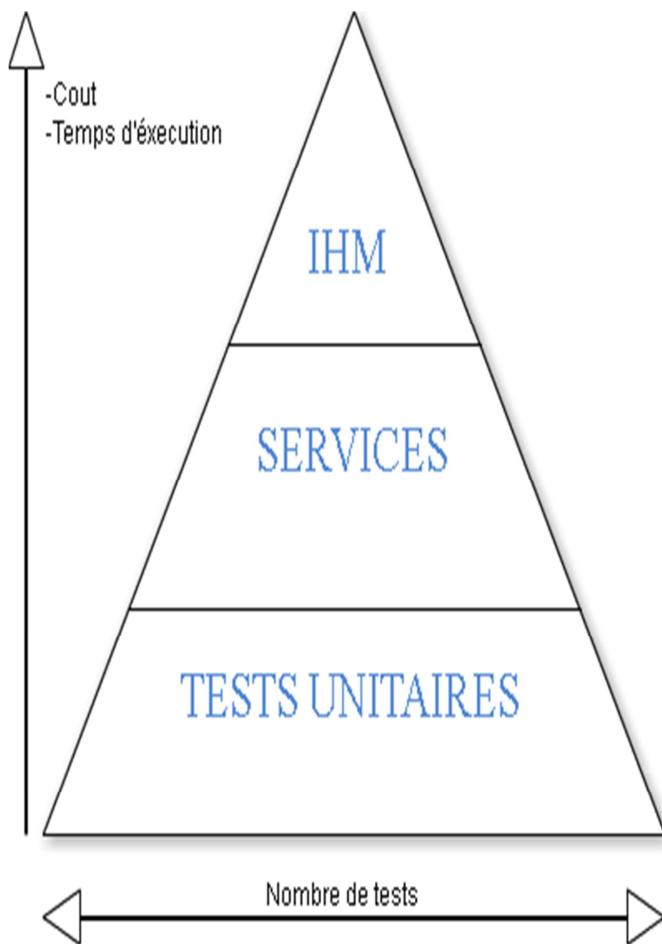
revenir sans cesse sur le code déjà testé pour y rajouter des fonctionnalités. Dans ces conditions il est important de pouvoir vérifier rapidement qu'on n'a pas introduit des régressions. C'est pour cette raison que les processus Agiles mettent l'accent sur les tests unitaires automatisés, et les chaînes d'intégration continues. Les tests unitaires, lorsqu'ils sont réellement unitaires sont faciles à maintenir et peuvent être exécutés à chaque compilation. De plus, comme ce type de test identifie les problèmes au plus proche de la réalisation, les corrections sont peu coûteuses. [2]

5 L'automatisation des tests

Avant même l'ascendant des méthodologies agiles comme SCRUM, nous savions qu'on devrait automatiser nos tests. Mais nous ne l'avons pas fait. Les tests automatisés ont été considérés comme coûteux, on met souvent des mois voire même, dans certains cas, des années pour les écrire, après l'implémentation d'une fonctionnalité. Une des raisons qui fait du mal à écrire des tests plus vite c'est qu'on les automatise au mauvais niveau.

5.1 Pyramide de test

Une stratégie d'automatisation de test efficace appelle à les diviser sur trois différents niveaux, comme le montre la figure suivante, c'est la pyramide de l'automatisation des tests. A la base de la pyramide on trouve les tests unitaires. Les tests unitaires devraient être le fondement d'une stratégie solide d'automatisation des tests. Les tests unitaires automatisés sont merveilleux parce qu'ils donnent des données spécifiques à un programmeur. Par exemple : il y a un bug et il est sur la ligne 47. Les programmeurs ont appris que le bug peut vraiment être sur la ligne 51 ou 42, mais il est beaucoup plus agréable d'avoir un test unitaire automatisé pour le dégager qu'un testeur dit: « Il y a un bug dans la façon dont vous récupérez les dossiers des membres de la base de données » qui pourrait représenter 100 lignes de code ou plus. Aussi, parce que les tests unitaires sont généralement écrits dans le même langage que le système, les programmeurs sont souvent plus à l'aise à les écrire. [3]

**Figure 13 Pyramide de test**

Les tests de niveau de services sont dédiés à tester les services de l'application séparée de son interface utilisateur. Mais il est plus crédible de tester ces services en simulant l'expérience du client utilisateur. Malheureusement les tests IHM sont :

- Fragile

Un petit changement dans l'interface utilisateur peut briser de nombreux tests. Lorsque cela est répété plusieurs fois au cours d'un projet, les équipes abandonnent et arrêtent la correction des tests.

- Cher à écrire

Une approche rapide de capture par l'enregistrement et de relecture des tests d'interface utilisateur peut fonctionner, mais les tests enregistrés de cette manière sont généralement les plus fragiles. La rédaction d'un bon test de l'interface utilisateur qui restera utile et valide prend du temps.

- Long

Les tests effectués par le biais de l'interface utilisateur prennent souvent beaucoup de temps d'exécution.

5.2 Pyramide de test inversé

Trop souvent, les cas de test qui ne peuvent pas être couverts par les développeurs dans les tests unitaires, sont directement automatisés au niveau de l'interface utilisateur, ce qui entraîne de grands lots des tests automatisés au niveau de l'interface utilisateur, qui prennent des éons pour exécuter et maintenir. Ce phénomène est représenté par une pyramide d'automatisation de test inversé.

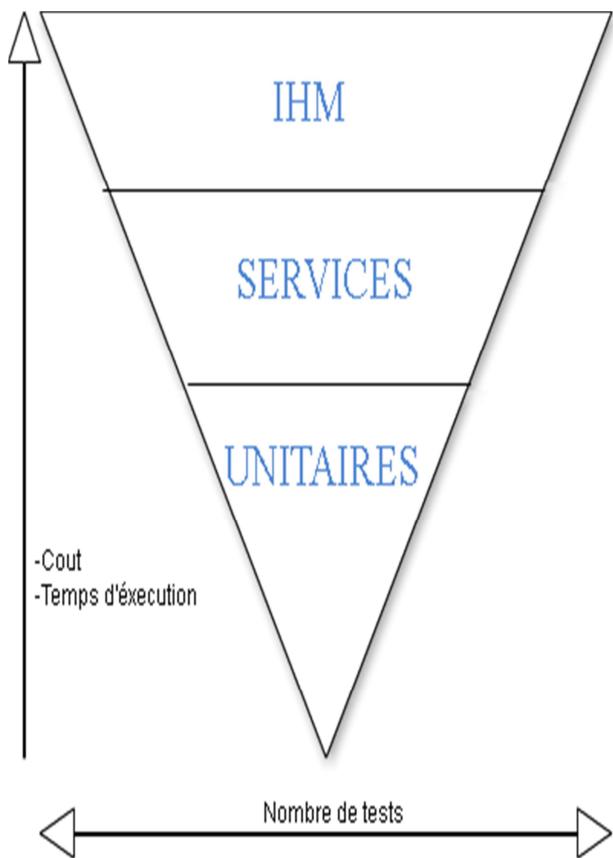


Figure 14 Pyramide de test inversé

Malheureusement c'est le cas pour le WEB3. Donc nous sommes appelés à bien choisir une stratégie qui sera bien garnie d'outils élus pertinemment.

6 Etude Comparative et Choix des outils

Après avoir spécifié notre objectif nous allons maintenant chercher les outils à utiliser dans ce projet et qui vont être les composants de notre solution.

6.1 Etude comparative des outils

Pour bien choisir un outil il faut toujours se méfier de ces inconvénients. Chaque outil comporte ses propres avantages et inconvénients, il faut avoir toujours recours à une comparaison objective.

6.1.1 Outil d'automatisation

Comme indique son nom WEB3 est le composant web de l'OOPRO. Ainsi, en cherchant les outils d'automatisation les plus répandus pour les applications web nous avons trouvé deux outils qui peuvent répondre à nos besoins le SELENIUM et le HP UFT/QTP.

Le tableau suivant présente une comparaison entre les fonctionnalités des deux outils

Fonctionnalité	SELENIUM	UFT/QTP
Langage supporté	Java, C#, Ruby, Python, Perl PHP, JavaScript	VB Script
Navigateur supporté	Google Chrome, Internet Explorer, Firefox, Opera, HtmlUnit	Google Chrome (jusqu'à ver 23) Internet Explorer, Firefox (ver 21)
Environnement supporté	Windows, Linux, Solaris OS X, autre (si navigateur & JVM ou JavaScript support existent)	Windows seulement
Mobile (Smartphones et tablette) supporté	Tous	Tous
Framework	Selenium + Eclipse + Maven / ANT+ Jenkins / Hudson et ses plugins / Cruise Control + TestNG + SVN	HP QC (Quality Center), HP ALM
Intégration continu	Possible via Jenkins / Hudson / Cruise Control	Possible via Quality Center / ALM or Jenkins
Reconnaissance des objets	ID de l'élément ou plusieurs attributs	ID/CSS/XPATH/ DOM/ LINK/JQUERY
Cout	License & frais annuel de maintenance	Aucun
Ressource (RAM &CPU)	Beaucoup	Peu
Documentation	Riche, détaillé, disponible et gratuite	payante

Tableau 9 Selenium vs UFT

6.1.2 Framework d'automatisation

Le Framework d'automatisation est un système intégré qui établit les règles d'automatisation. Ce système intègre les bibliothèques des fonctions, les sources de données de test, les détails de l'objet et de

divers modules réutilisables. Ces composants agissent comme de petits blocs de construction qui doivent être assemblés pour représenter un processus métier. Le Framework fournit la base et simplifie l'effort d'automatisation.



Figure 15 Puzzle des Keywords

Les principaux avantages d'un Framework, sont les concepts et les outils qui fournissent un soutien pour les tests de logiciel automatisé et le faible coût d'entretien. Le tableau suivant présente une comparaison entre les critères des trois outils élus après une recherche pertinente.

critères	Robotframework	SilkTest	QF-Test
Système d'exploitation	tout	Windows	Windows, Linux
Système sous test	tout	Windows, Linux, application Web	Java, Swing, SWT, JavaFX, applications Web
Entreprise	Open source	Micro Focus	Quality First Software GmbH
License	Apache	Propriétaire	Propriétaire
IHM test	Oui	Oui	Oui
Automatisation	Oui	Oui	Oui
Version courante	3.0	16.0	4.0.11
Statut	Active		Active
Documentation	Riche, détaillé, disponible et gratuite	payante	payante

Tableau 10 Robotframework vs Silktest vs QF-test

6.2 Choix des outils

Après avoir cité les caractéristiques de chaque outil nous adoptons la solution combinée de « RobotFramework » et « Selenium » pour les raisons suivantes :

- Documentation riche : communautée très active
- Cout : gratuit
- Multi-langage, multi-utilisation et multiplateforme.

L'équipe OOPRO dispose de l'outil « Jenkins » pour l'intégration continue et une solution « SVN » pour la gestion des versions donc nous allons les utiliser.

Nous allons voir par la suite, dans le chapitre réalisation, comment intégrer ces outils pour aboutir à notre solution.

Conclusion

Après avoir défini d'une manière générale les notions fondamentales d'un test et de l'automatisation, nous avons choisi une combinaison d'outils qui seront les composants de notre solution. Nous allons spécifier la branche informelle qui comporte les besoins fonctionnels et non fonctionnels de notre solution qui sera l'objectif du prochain chapitre qui va mettre en œuvre les fonctionnalités que devra offrir notre solution.



CHAPITRE III

SPÉCIFICATION

DES BESOINS

Spécification des besoins

Introduction

Une étape primordiale de tout projet consiste à effectuer une étude d'analyse des besoins et des spécifications. Cette étude consiste à examiner le système auquel nous allons créer des améliorations et remédier aux objectifs auxquels il doit répondre. Ce chapitre se consacre à clarifier les objectifs à atteindre et à analyser les besoins des utilisateurs et les contraintes à satisfaire.

1 Analyse des besoins

Le projet consiste à concevoir et réaliser un processus pour faciliter et automatiser l'opération du test de non régression via l'IHM du composant Web3 ainsi que le développement d'une interface qui permet de simplifier la manipulation d'un tel processus afin d'aller beaucoup plus vite qu'en mode manuel.

1.1 Besoins fonctionnels

Les besoins fonctionnels doivent répondre aux exigences du système en termes de fonctionnalités. Ils constituent une sorte de contrat par rapport au comportement du système cible. Ces exigences sont citées ci-dessous :

- L'édition des scripts des tests automatiques.
- La gestion des scripts.
- La gestion des keywords (mots clé) des scripts.
- L'exécution programmé et non programmé des scripts des tests automatiques.
- La génération des rapports bien détaillés.
- La consultation des Log et des rapports d'exécutions.
- La planification des scenarios des tests.
- La gestion des versions.

1.2 Besoins non fonctionnels

Parmi les considérations et les contraintes dont nous devons tenir compte lors de la réalisation du projet nous pouvons citer :

- Contrainte technique : La solution ne doit exiger aucune connaissance approfondie des langages de programmation pour le développeur des scripts des tests automatiques et surtout pas le Java car on a choisi le « Selenium » comme outil d'automatisation web.

- Contrainte ergonomique : Le système à développer doit présenter des interfaces utilisateurs conviviales bien structurées du point de vue contenu informationnel.
- Contrainte d'utilisation : Le système doit fournir une interface simple et facile à utiliser. Le système doit aussi aider à être utilisé.
- Contrainte de sécurité : Le système doit appliquer une stratégie d'authentification.

2 Spécification des besoins

Lors de l'analyse des besoins, nous avons cité les besoins fonctionnels ainsi que les besoins techniques et non fonctionnels qui doivent figurer dans la solution proposée. A cette étape, il est important de connaître les interactions entre le système et ses utilisateurs. Le concept des cas d'utilisation permet d'écrire ces interactions. Il faut alors définir l'acteur et les cas d'utilisation du système.

2.1 Identification des acteurs

Un acteur est une personne ou un système qui interagit avec le système étudié en échangeant de l'information en entrée et en sortie. Les acteurs humains pour notre application sont les suivants :

- Analyste du test : Cet acteur a comme mission l'identification des fonctionnalités et services à tester, la définition et l'introduction des scénarios des tests (les étapes) dans le système (QC), cet acteur définit le plan de test.
- Développeur : Cet utilisateur a comme métier d'implémenter les scripts des cas de test en définissant les keywords nécessaires.

2.2 Diagramme de cas d'utilisation général

Suite à la définition de nos acteurs, nous décrivons leurs interactions avec notre système à travers un diagramme de cas d'utilisation général qui reflète les fonctionnalités générales.

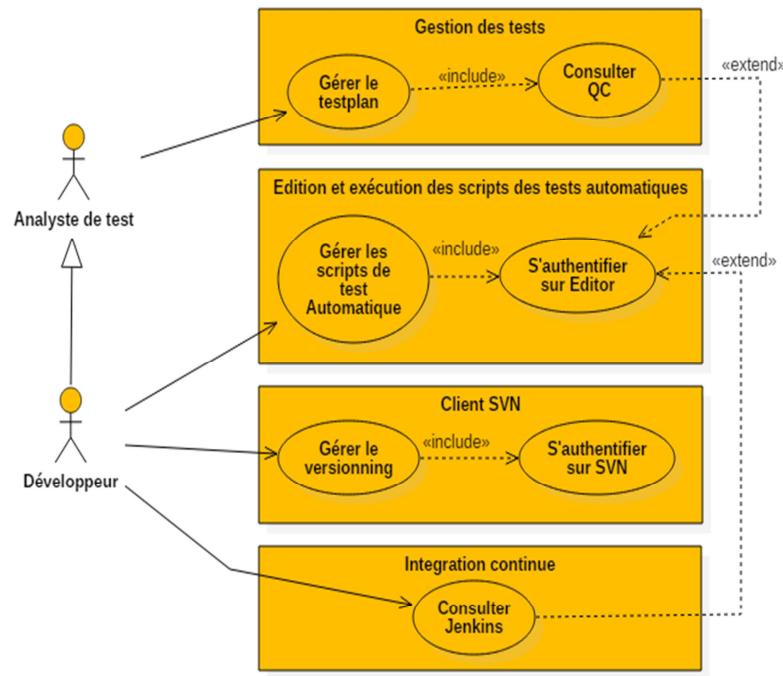


Figure 16 Diagramme de cas d'utilisation générale

3 Raffinement des cas d'utilisation

3.1 Sous-système « Gestion des tests » QC

La gestion des tests assurés par HP QC est dédiée à l'analyste des tests car c'est lui qui va dégager les fonctionnalités et services à tester. Il est le plus compétant du côté fonctionnel. Ce système lui permet l'ajout, la suppression, la modification d'un ou de plusieurs tests. Il lui permet aussi de détailler les étapes des cas de test.

La figure suivante illustre le raffinement du sous-système « Gestion des tests » et ses cas d'utilisation qui nous intéressent dans notre solution.

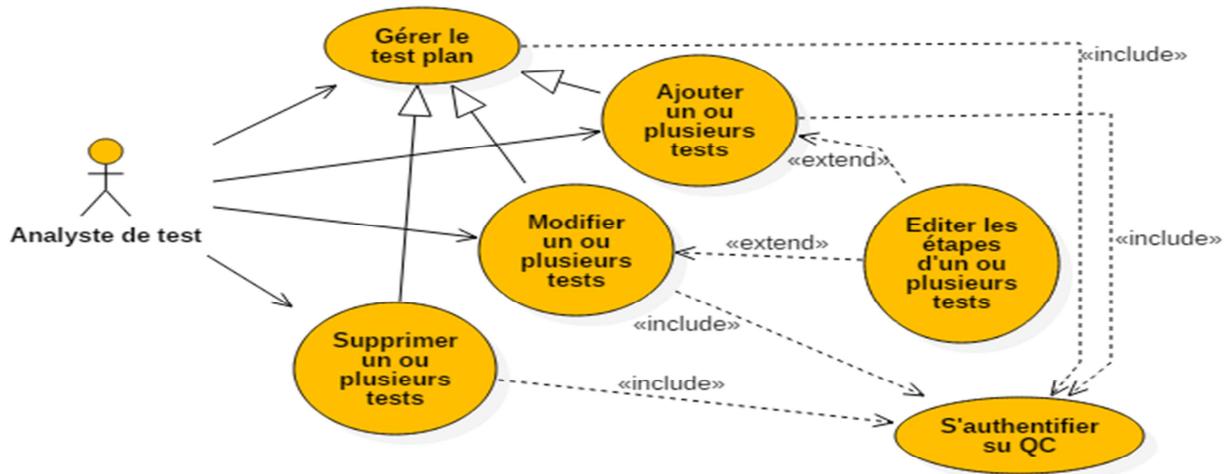


Figure 17 Diagramme de cas d'utilisation générale de sous-système « Gestion des tests »

CHAPITRE III : Spécification des besoins

La description scénariste du cas d'utilisation d'ajout d'un test dans le test plan de QC est représentée par le tableau suivant

Titre	Ajouter un test
Acteur	Analyste de test
Pré condition	Authentification sur QC
Post condition	Cas de test ajouté
Scénario nominal	<p>L'utilisateur accède au menu d'ajout d'un nouveau cas de test sous le répertoire souhaité via un clic droit puis il choisit « New Test »</p> <p>Le système affiche un formulaire à remplir</p> <p>L'utilisateur remplit les champs du formulaire avec les informations qui conviennent.</p> <p>Le système affiche un message affirmant que l'opération d'ajout a été effectuée avec succès.</p>
Scénario alternatif	<p>Champs obligatoires non renseignés :</p> <p>Le système affiche un message d'erreur</p>
Exception	En cas d'erreur le système affiche un message d'erreur.

Tableau 11 Description du scenario de cas d'utilisation « Ajouter un test dans le test plan de QC »

3.2 Sous-système « Editor »

L'implémentation et l'exécution des scripts des tests automatiques sont dédiées au développeur.

Afin de faciliter l'ajout, la suppression, la modification et l'exécution des scripts des tests automatiques, nous allons implémenter une nouvelle application que nous appelons «Editor».

«Editor» comportera des fonctionnalités importantes comme l'autocomplete, la détection des erreurs et d'autres qui ne sont pas très fonctionnels.

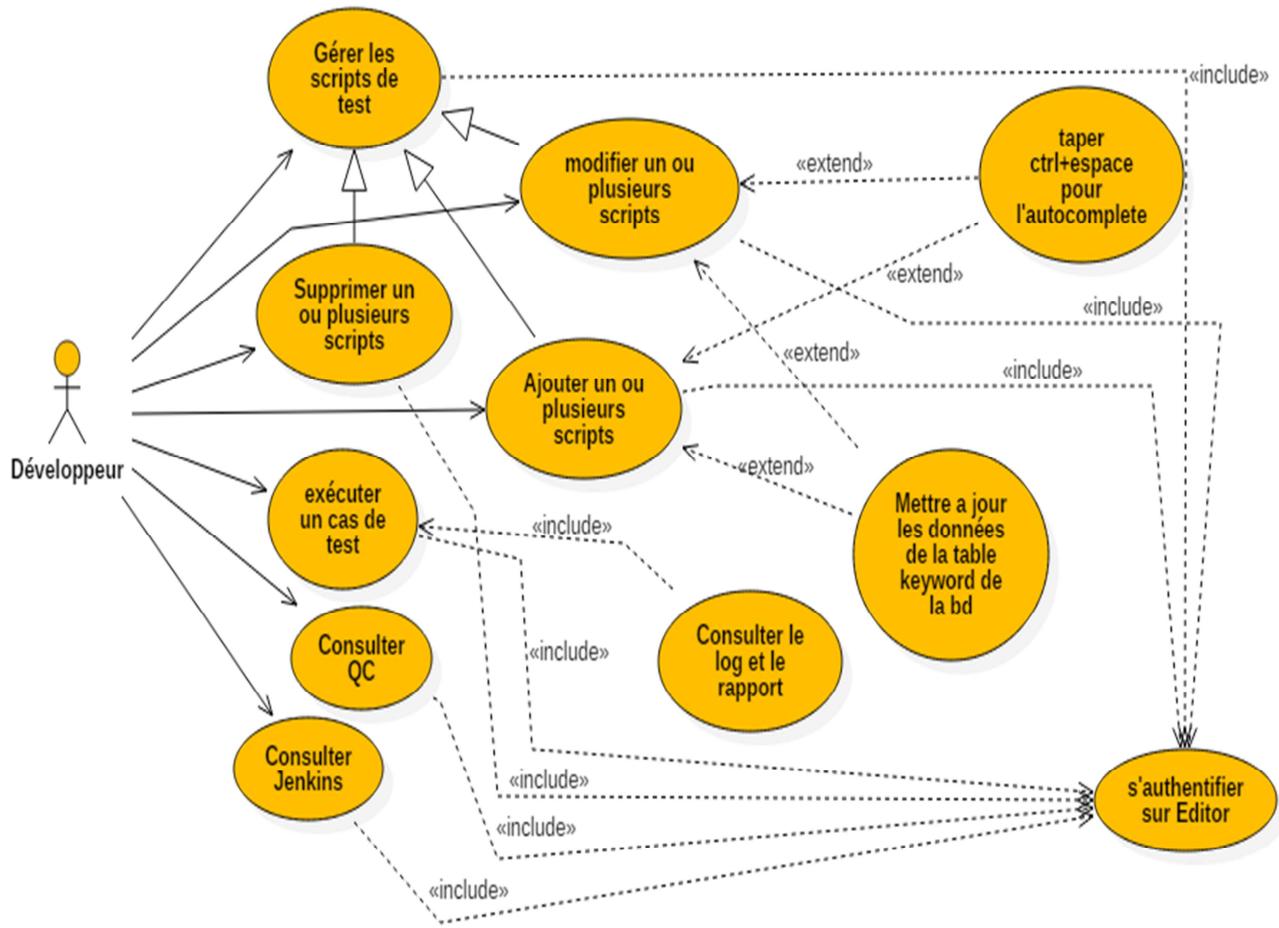


Figure 18 Diagramme de cas d'utilisation générale du sous-système « Editor »

CHAPITRE III : Spécification des besoins

La description scénariste du cas d'utilisation d'ajout et d'exécution d'un script de test automatique à partir de l' «Editor » est représentée par le tableau suivant

Titre	Ajouter un test
Acteur	Développeur
Pré condition	Authentification sur Editor
Post condition	Cas de test ajouté et exécuté
Scénario nominal	<p>L'utilisateur fait un clic droit sur le dossier où il veut ajouter son nouveau script</p> <p>Le système affiche un champ input texte vide pour y introduire le nom du script</p> <p>L'utilisateur remplit le champ</p> <p>Le système donne la main à l'utilisateur pour écrire le script.</p> <p>L'utilisateur tape CTRL+ espace</p> <p>Le Système expose la liste des keywords qui existe dans la base de données en la triant par ordre alphabétique et en éliminant les keywords selon les lettres tapées par l'utilisateur (autocomplete)</p> <p>L'utilisateur termine l'implémentation de son script et tape le bouton « Run ».</p> <p>Le système exécute le script en ouvrant un navigateur et en appliquant toutes les étapes implémentées dans le script.</p> <p>Le système génère deux fichiers : le « log.html » et le « rapport.html » et affiche un résumé de log dans la partie console.</p> <p>L'utilisateur peut maintenant consulter ces fichiers et il peut aussi consulter QC pour ajouter l'état « PASSED » ou « FAILED » du cas de test.</p>
Scénario alternatif	<p>Une erreur d'exécution est survenue :</p> <p>Le système affiche un message d'erreur.</p> <p>L'utilisateur introduit un login ou un mot de passe incorrect.</p> <p>Le système affiche un pop-up pour informer l'utilisateur qu'il doit réessayer.</p>
Exception	En cas d'erreur le système affiche un message d'erreur.

Tableau 12 Description du scenario de cas d'utilisation « Ajouter et exécuter un script de test automatique à partir de Editor »

3.3 Sous-système «Intégration Continue»

L'intégration continue consiste, dans notre situation, à l'exécution automatique quotidienne de la dernière version des tests. Cette fonctionnalité est assurée par l'outil d'intégration continue Jenkins. La

manipulation de cet outil est dédiée au développeur qui doit veiller sur le bon fonctionnement de ses tests automatiques.

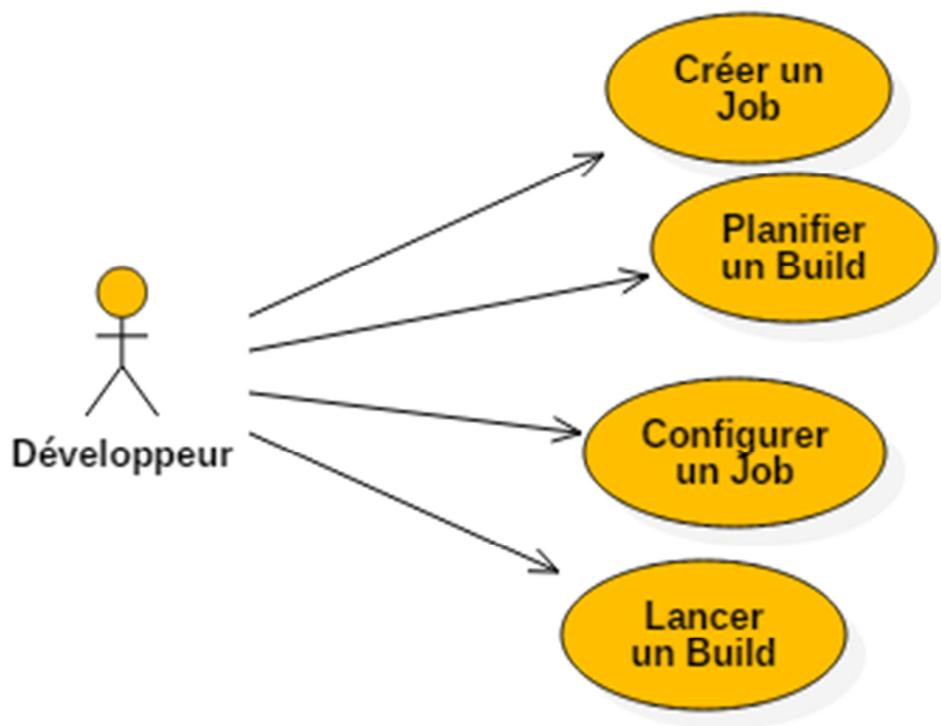


Figure 19 Diagramme de cas d'utilisation générale du sous-système « Intégration Continue »

La description scénariste du cas d'utilisation « Configurer un Job » et d'exécution d'un script de test automatiques à partir de « «Editor » est représentée par le tableau suivant

Titre	Ajouter un test
Acteur	Développeur
Pré condition	Accéder à Job dans Jenkins
Post condition	Job bien configuré
Scénario nominal	<p>L'utilisateur clique sur la rubrique « configurer » sous le job souhaité</p> <p>Le système affiche un formulaire de configuration à remplir</p> <p>L'utilisateur remplit les champs du formulaire et surtout les champs de «Gestion de code source » et « Build » puis clique sur le bouton sauver.</p> <p>Le système sauvegarde la configuration du Job.</p>
Scénario alternatif	<p>Champs obligatoires non renseignés :</p> <p>Le système affiche un message d'erreur</p>
Exception	En cas d'erreur le système affiche un message d'erreur.

Figure 20 Description du scenario de cas d'utilisation « Configurer un Job »

4 Le Backlog du produit

Comme nous avons adopté la méthodologie SCRUM tout au long de ce projet, on a pris recours à présenter notre Backlog du produit. Nous avons reformulé dans ce Backlog les fonctionnalités attendues de l'application. Après la collection des fonctions essentielles de notre solution nous avons parvenu à la décomposition de modules suivants :

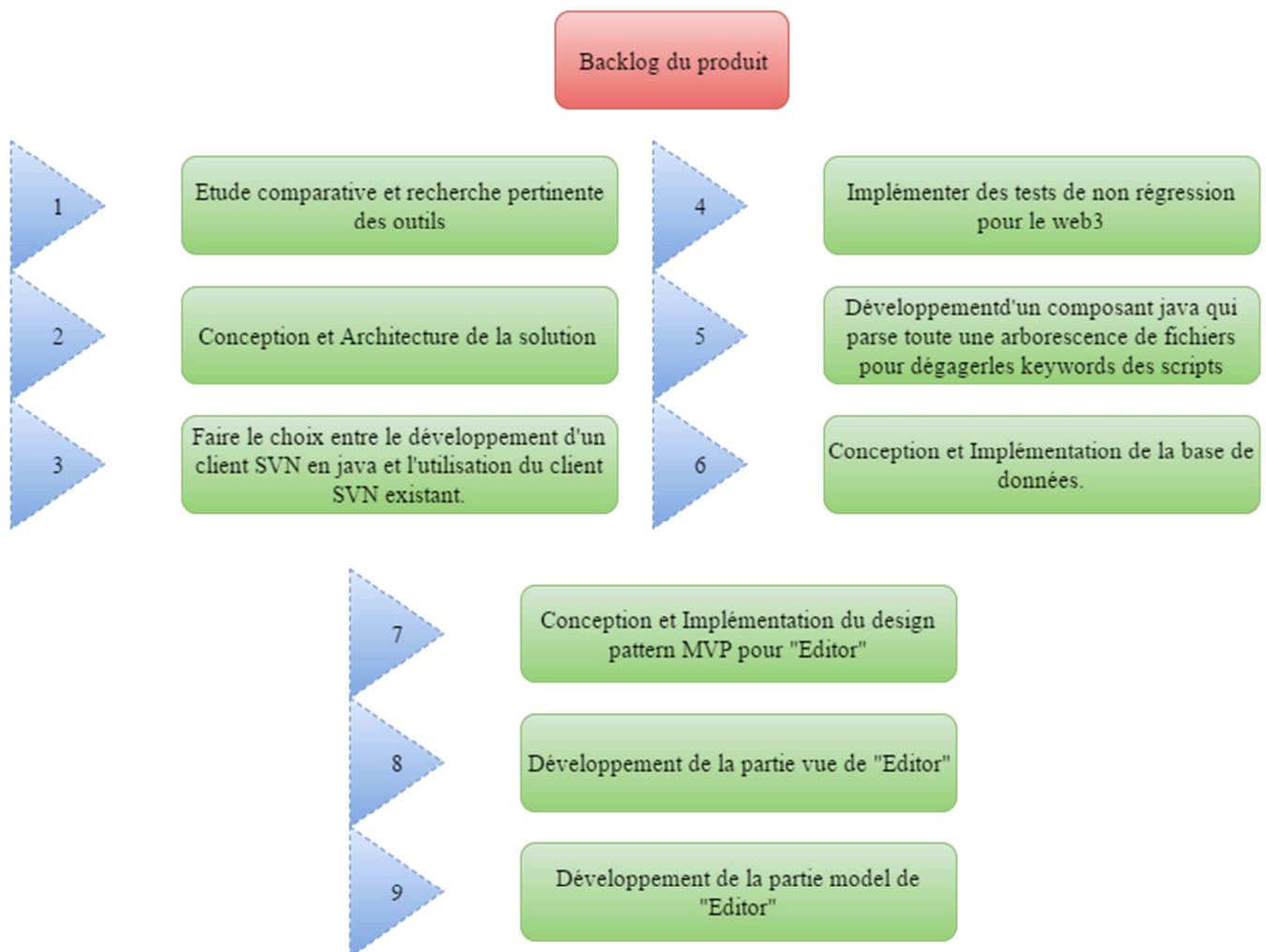


Figure 21 Le Backlog du produit

4.1 Définition des Sprints du projet

A présent, nous procédons à la définition des User Stories en les regroupant en même temps en Sprint.

User Story	Description	Estimation
1	Etude comparative et recherche pertinente des outils	
2	Conception et architecture de la solution	
3	Faire le choix entre le développement d'un client SVN en java et l'utilisation du client SVN existant.	

Tableau 13 Sprint 0

User Story	Description	Estimation
1	Implémenter des tests de non régression pour le web 3	
2	Développement d'un composant java qui analyse toute une arborescence de fichiers pour dégager les keywords des scripts.	
3	Conception et implémentation de la base de données.	

Tableau 14 Sprint 1

User Story	Description	Estimation
1	Conception et implémentation du design pattern MVP	
2	Développement de la partie vue de «Editor »	
3	Développement de la partie model de «Editor »	

Tableau 15 Sprint 2

Conclusion

Ce chapitre nous a permis d'analyser et de spécifier les besoins de la solution à développer. Ceci nous a permis d'avoir une vue simplifiée, abstraite et détaillée des fonctionnalités que doit offrir notre solution. À travers cette étape fondamentale, nous avons pu fixer les exigences de l'application. Une fois les besoins fixés, nous commencerons à prévoir la manière qui permettra une mise au point optimale et fiable du futur système.



CHAPITRE IV

ÉTUDE

CONCEPTUELLE

Étude conceptuelle

Introduction

Ce chapitre sera consacré à la conception de notre solution. Nous entamerons tout d'abord la description de l'architecture fonctionnelle de la solution proposée. Par la suite, nous présenterons l'aspect orienté objet de notre solution. Enfin, nous détaillerons l'enchaînement dynamique de quelques scénarios avec les diagrammes des séquences.

1 Architecture fonctionnelle

1.1 Architecture physique

Notre solution s'agit de la combinaison des outils déjà mentionnés dans les chapitres qui précèdent donc le choix de l'architecture physique influence la qualité notre système. Notre solution doit assurer une bonne qualité en termes de flexibilité et de performance. C'est pour ces raisons-là, nous avons choisi l'architecture physique illustrée par la figure suivante.

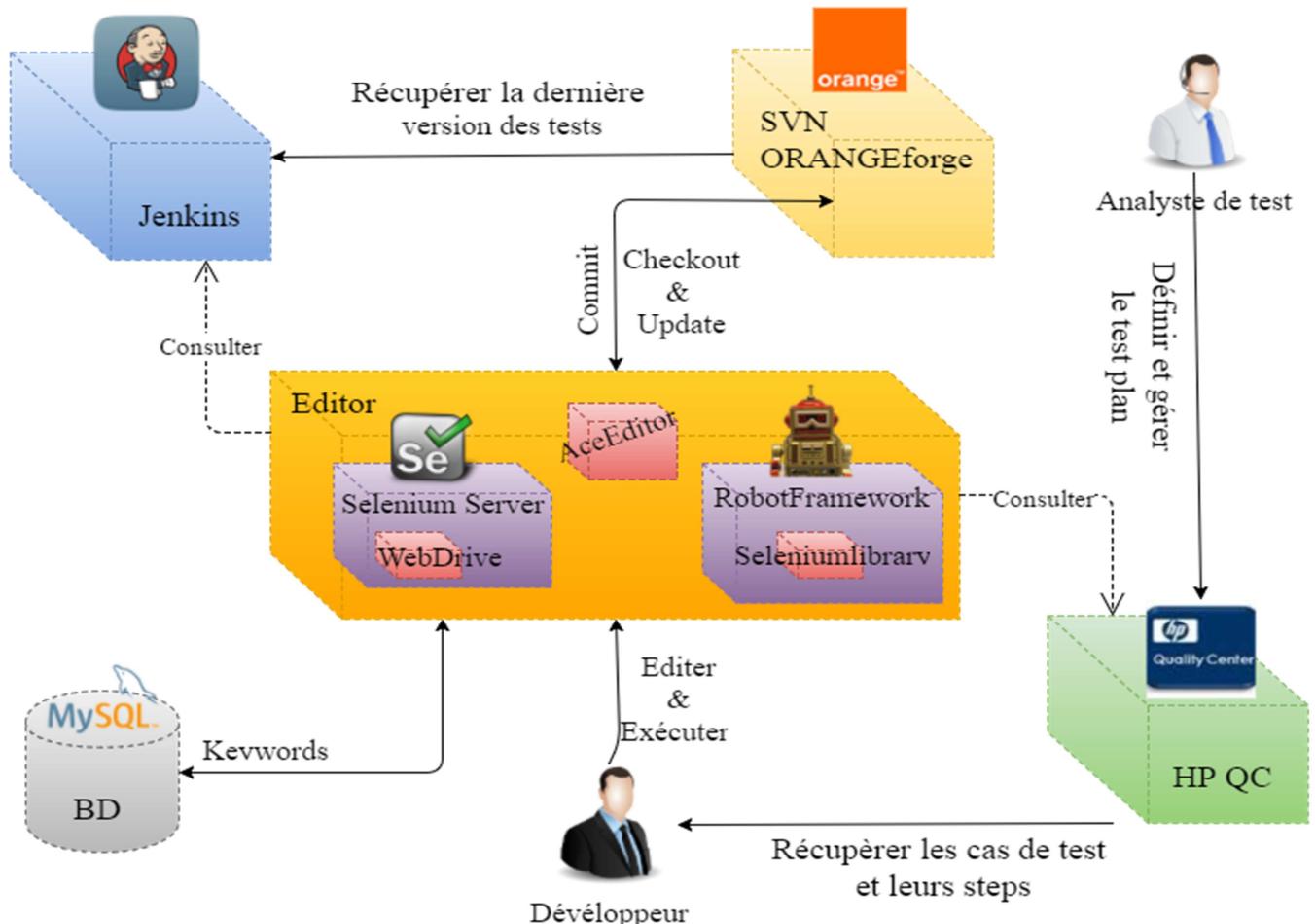


Figure 22 Architecture physique

1.2 Architecture logique

Cette section s'intéressera à l'étude de l'«Editor» le module principal de notre solution. Dans le développement logiciel, la division en couches est une technique répandue pour décomposer logiquement un système complexe. En effet, elle revient à modéliser un système en un arrangement en couches. Chaque couche constitue l'une des parties du système. La communication entre les couches s'effectue en établissant des contrats via des interfaces. Ainsi, la notion de dépendance, quant à elle, se caractérise par des instructions d'import de packages. Dans notre application, nous avons choisi une division en trois couches distinctes qui sont :

- Couche présentation UI (User Interface) : elle gère l'affichage des données et les interactions du système avec l'utilisateur. Outre, la séparation de cette couche permet de présenter pour une même application plusieurs présentations.
- Couche métier : implémentation de règles de gestion. Cette couche se rapporte aux tâches à réaliser par le système sur les données ainsi que les traitements nécessaires suite à une action venant de l'utilisateur (implémentation des règles fonctionnelles, vérification d'authentification, calculs divers, etc.).
- Couche accès aux données DAO (Data Access Object) : cette couche regroupe le stockage et les mécanismes d'accès aux données de façon qu'elles soient utilisables par l'application au niveau traitement.

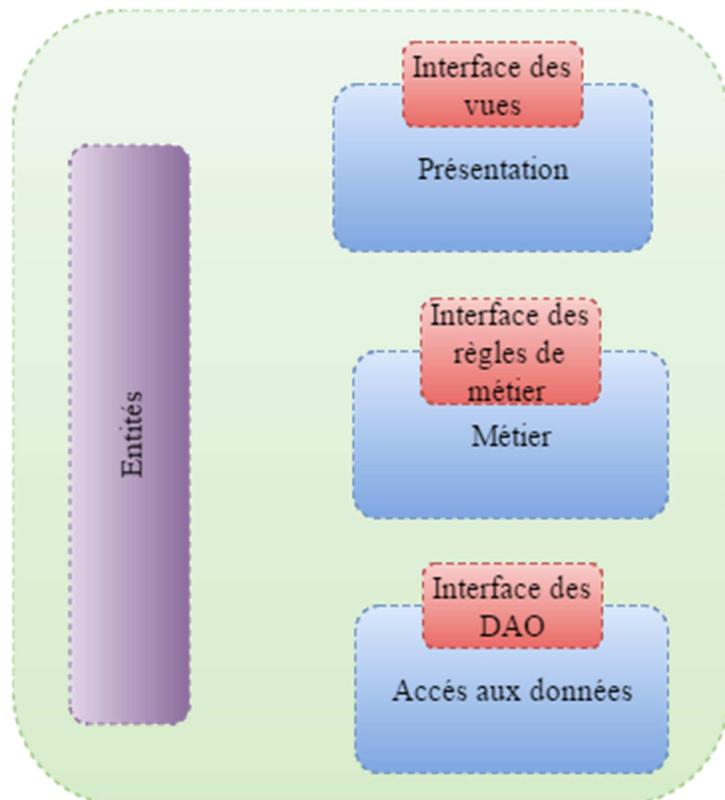


Figure 23 Décomposition en couche de l'architecture logique

Nous avons choisi la décomposition en couches puisqu'elle présente de nombreux avantages à savoir :

- Faible couplage.
- Possibilité d'isoler une couche afin de faciliter sa compréhension et son développement.
- Très bonne gestion des dépendances.
- Possibilité de substituer les implémentations de couches.
- Réutilisation facilitée.
- Testabilité favorisée (grâce à l'isolement et à la possibilité de substituer les implémentations).

Avec cette architecture logique, nous avons utilisé le patron de conception MVP. Le modèle-vue-présentation (en abrégé MVP, de l'anglais model-view-presenter) est un patron de conception, considéré comme un dérivé du patron de conception MVC (modèle-vue-contrôleur).

Dans le MVP, le Présenter endosse le rôle du cerveau (comme le contrôleur dans le MVC), le modèle porte les informations métier et la vue représente l'interface graphique.

On compare souvent le pattern MVP au pattern MVC (Model-View-Controller), dont les points communs sont importants.

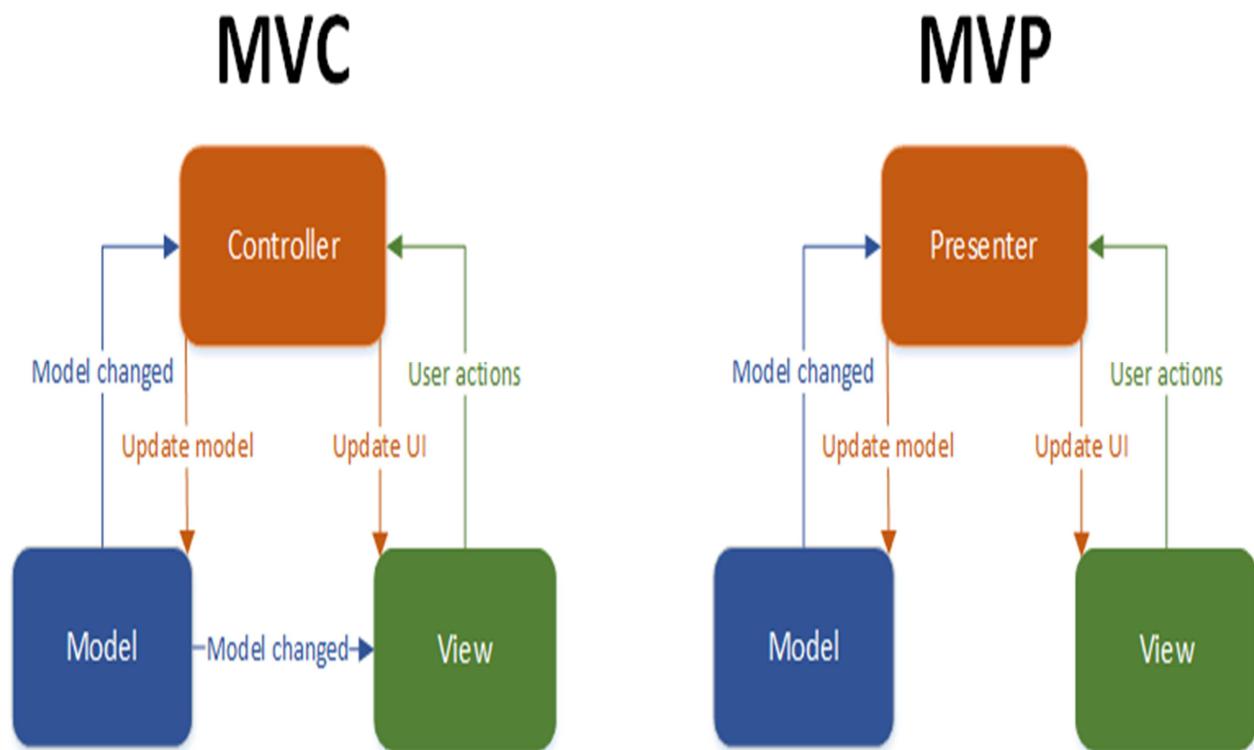


Figure 24 Comparaison entre les design patterns « MVC » et « MVP »

On peut relever quelques particularités du MVP qui le différencie du MVC. En premier lieu, la séparation stricte des responsabilités de la logique dans le Présenter et de l'affichage dans la vue. On peut noter également que la vue, implantée comme étant interface, peut dépendre ou non du modèle. Ce n'est pas le cas dans le MVC où la vue dépend systématiquement du modèle.

- Le Modèle :

Il représente les informations que l'utilisateur souhaite visualiser et ou manipuler dans les vues de l'application. C'est un acteur passif du modèle MVP. Il est manipulé par le Présenter et également la vue si on s'inscrit dans le cadre du MVP Présenter Superviseur (Le pattern MVP peut être implémenté de différentes façons). Le modèle, lui, n'utilise ni l'un ni l'autre.

- La Vue :

Elle affiche à l'écran les informations nécessaires au cas d'utilisation dont elle est liée et répond également aux actions de l'utilisateur. Elle travaille en coopération avec le Présenter. Dans son rôle actif, la Vue lui délègue les actions utilisateur qu'elle reçoit. Dans son rôle passif, la Vue se met à la disposition du Présenter pour mettre à jour les informations qu'elle affiche, ainsi que pour récupérer ce que l'utilisateur a saisi.

- Le Présenter :

Contrairement au modèle et à la vue, le Présenter n'est pas visible pour l'utilisateur de l'application. C'est un composant de contrôle qui permet d'orchestrer le fonctionnement de la couche web de notre application. Il a en commun avec le contrôleur du MVC d'être le cerveau des 3 éléments et d'être l'homme du milieu entre l'interface graphique et le modèle. Il assure deux fonctions principales :

- Répondre aux sollicitations de la Vue qui lui délègue les événements utilisateurs qu'elle reçoit. Pour ce faire, le Présenter peut lui-même réaliser les traitements ou bien collaborer avec des objets ayant davantage la connaissance pour réaliser ces traitements.
- S'assurer que la Vue affiche des informations conformes aux résultats des requêtes utilisateurs qui lui ont été transmises par la Vue. A cet effet, le Présenter met à jour les informations de la Vue à partir des méthodes que l'interface de la Vue expose.

2 **Vue statique du système**

En UML, la vue statique du système se traduit par le diagramme de classes. Il fournit une représentation abstraite des objets du système qui vont interagir ensemble pour réaliser les cas d'utilisations. Les diagrammes de classes sont les bases de l'analyse orientée objet. Ils montrent les classes du système, leurs interactions (y compris l'héritage, l'agrégation et l'association), ainsi que les opérations et les attributs des classes. Les diagrammes de classes sont utilisés pour une grande variété de fins. Ils montrent la structure du système conçu au niveau des classes et des interfaces, leurs caractéristiques, les contraintes et les relations ainsi que les associations, les généralisations et les dépendances. [4]

2.1 Diagramme de classe « login »

Ce diagramme de classe met en valeur les classes concernées pendant l'authentification. Il présente la séparation en couche dont nous avons parlé auparavant. Dans ce diagramme de classe on trouve une utilisation concrète de l'MVP.

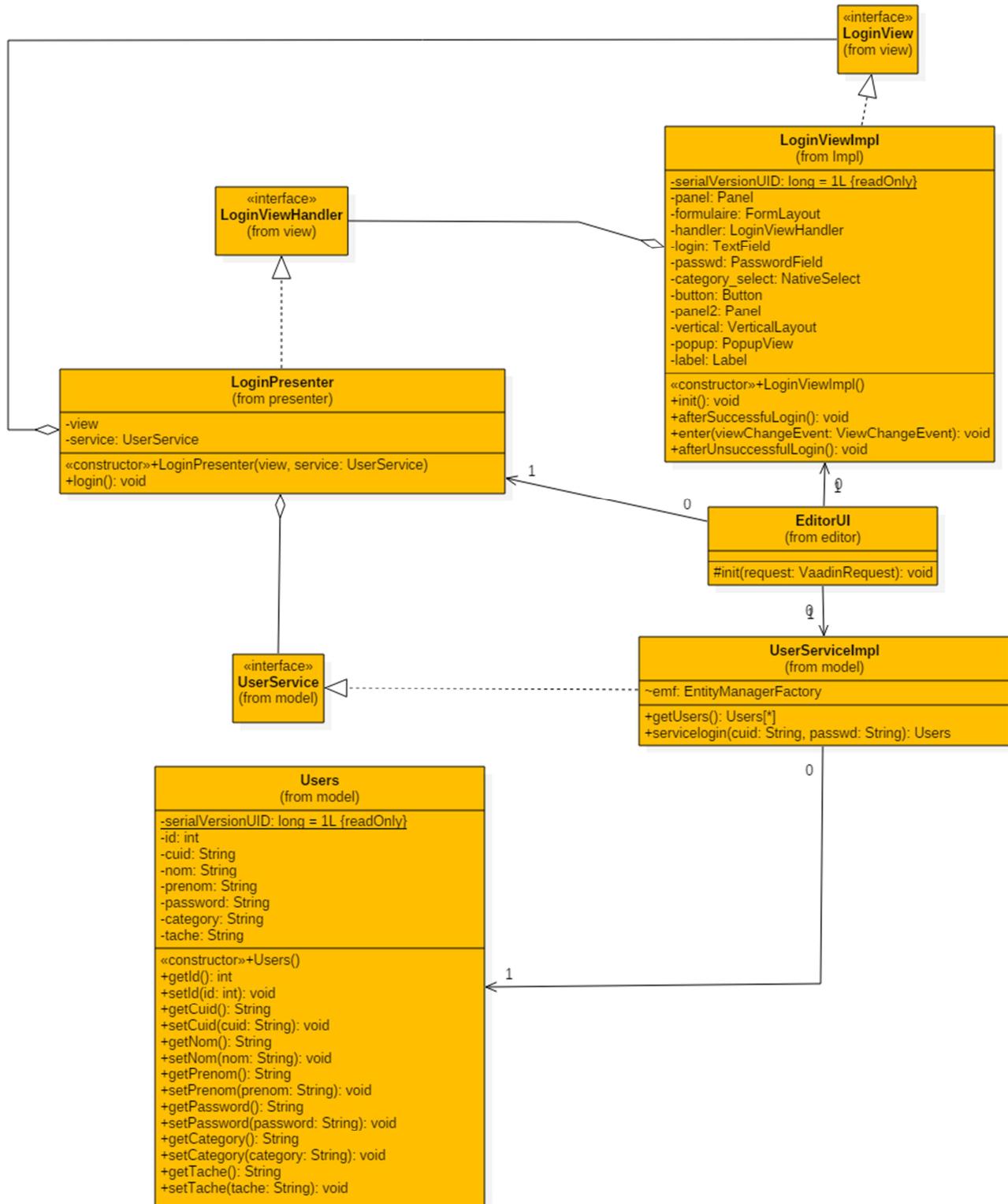


Figure 25 Diagramme de classe « Login »

2.2 Diagramme de classe « Editor »

Ce diagramme de classe met en valeur les classes concernées pendant la manipulation du module « Editor » pour l'édition, l'exécution des scripts des tests et même pour la consultation des rapports générés. Dans ce diagramme de classe on trouve une utilisation concrète de l'MVP.

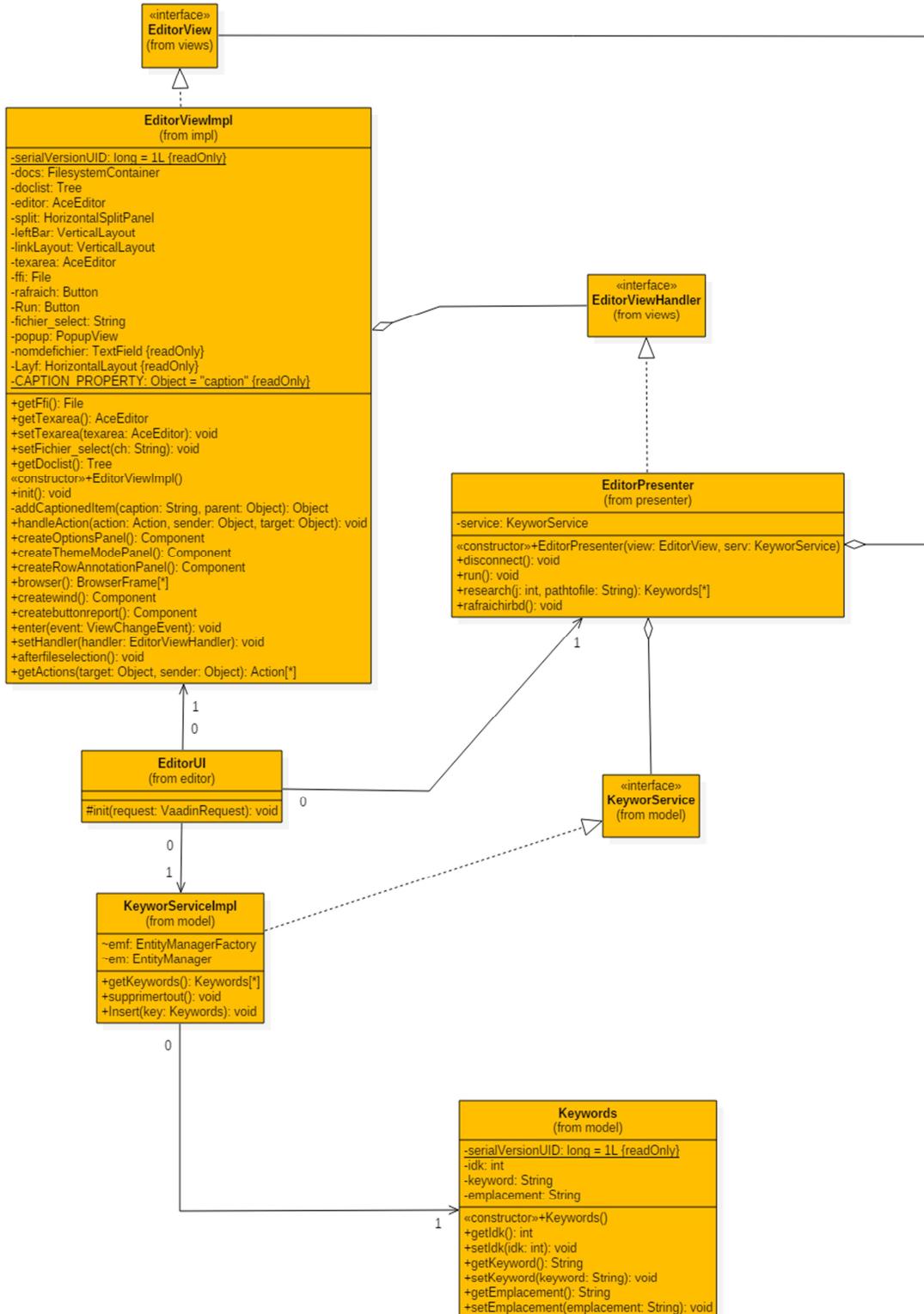


Figure 26 Diagramme de classe « Editor »

2.3 Diagramme de classe générale

Ce diagramme de classes comporte toutes les classes de l'application sans avoir détaillé les dépendances (les jars).

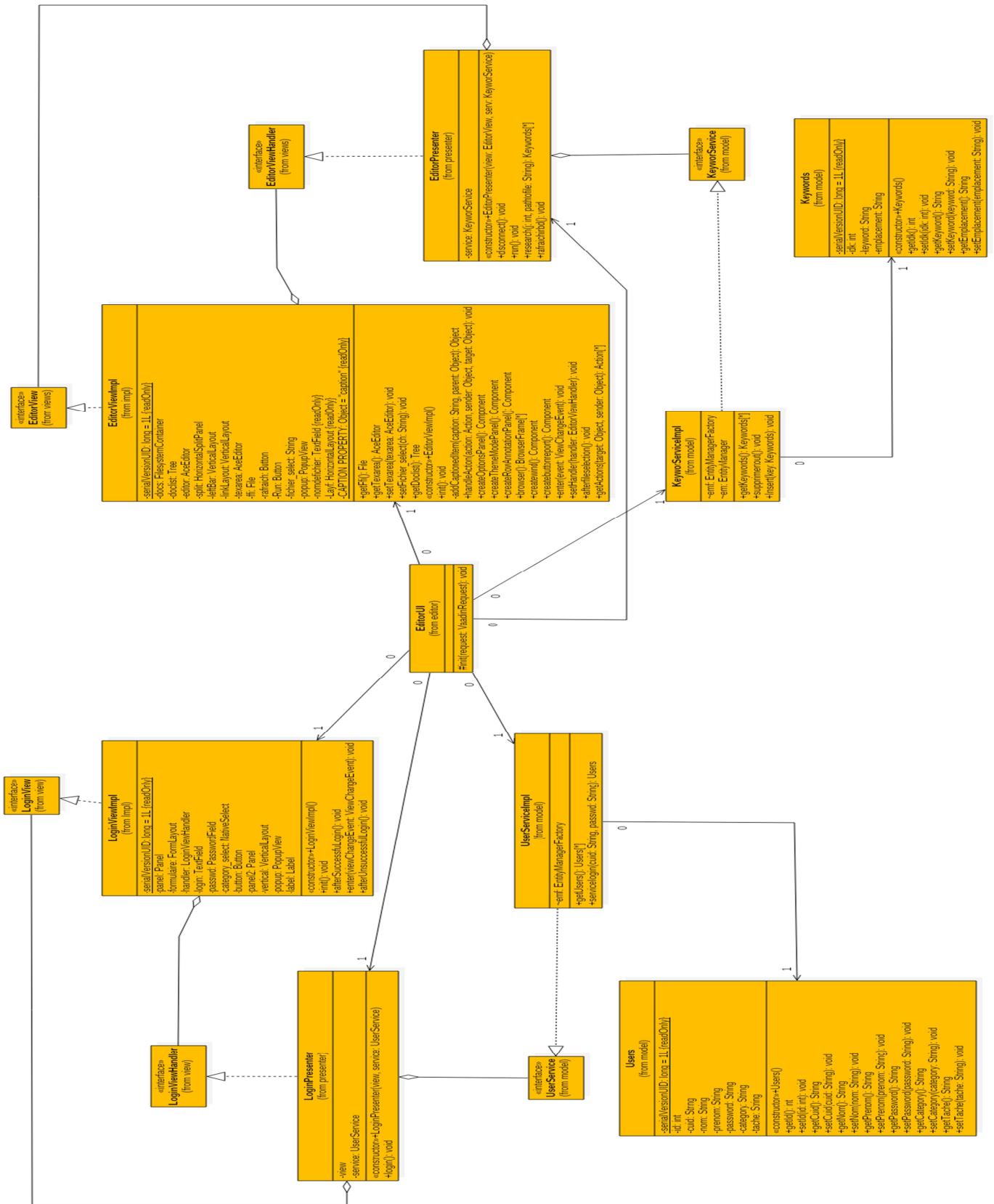


Figure 27 Diagramme de classe générale

3 Vue dynamique du système

Pour modéliser les scénarios, nous allons modéliser leurs enchaînements dynamiques via des diagrammes de séquence. Leur objectif est assurer une représentation plus complète des interactions entre les objets selon un point de vue temporel. Le diagramme de séquence UML modélise le flux de la logique au sein du système d'une manière visuelle, ce qui permet à la fois de documenter et de valider la logique. Il est couramment utilisé pour l'analyse et la conception. Le diagramme de séquence est l'artefact le plus populaire pour la modélisation dynamique qui met l'accent sur l'identification du comportement au sein du système. [5]

3.1 Authentification

L'authentification est le cas d'utilisation nécessaire pour la plupart des autres cas. Le diagramme suivant met en valeur la séquence réalisée durant cette manipulation en mentionnant les classes et les attributs concernés.

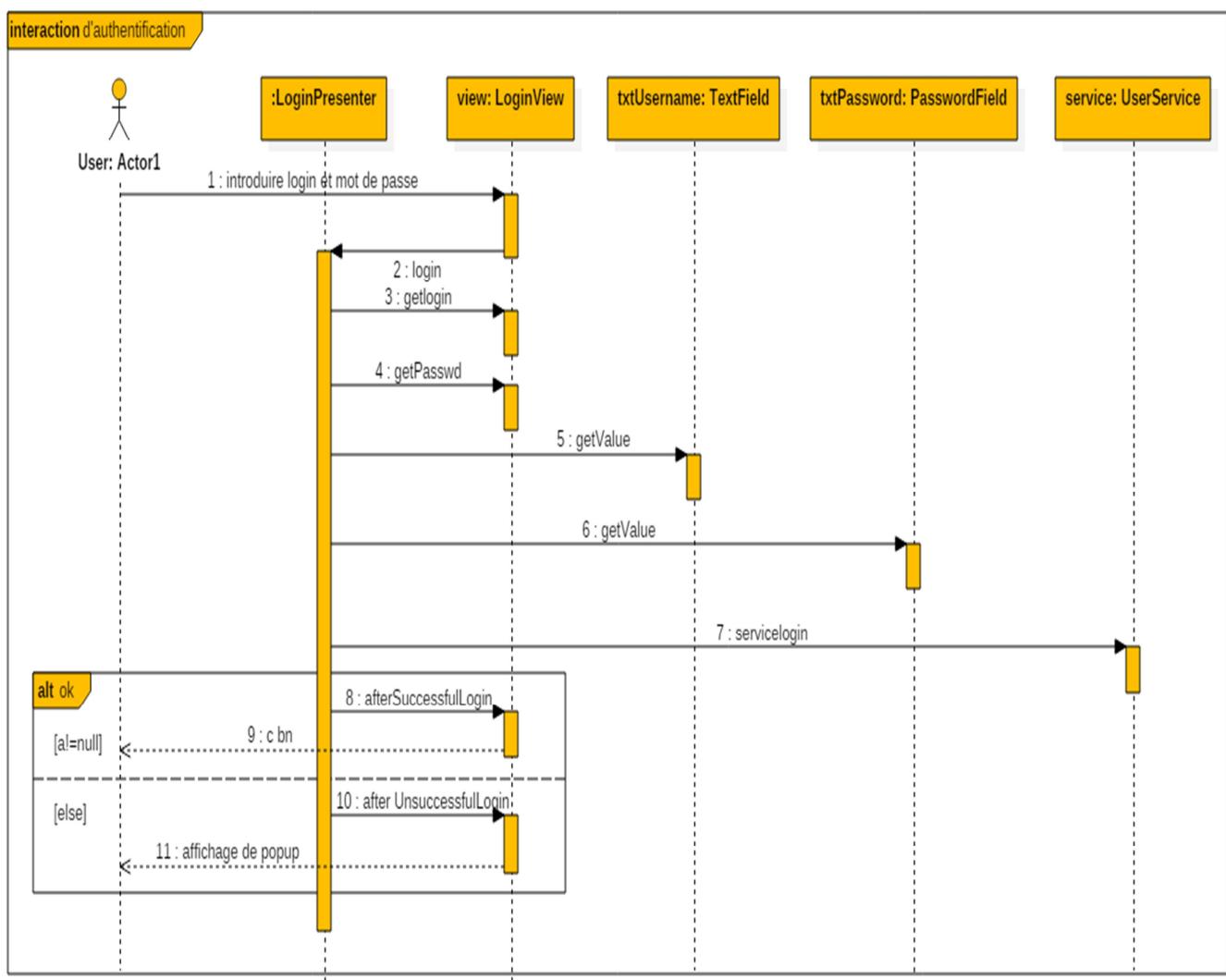


Figure 28 Diagramme de séquence de scenario d'authentification

3.2 Exécution de script

L'exécution des scripts des tests est la fonctionnalité principale de notre solution, elle présente aussi une bonne partie de notre objectif. On va passer par une description détaillée de ce cas d'utilisation via un diagramme d'activité avant de dégager sa séquence.

3.2.1 Diagramme d'activité

Ce diagramme nous permet de représenter graphiquement le comportement général du système pour le cas d'utilisation «Exécuter un cas de test ».

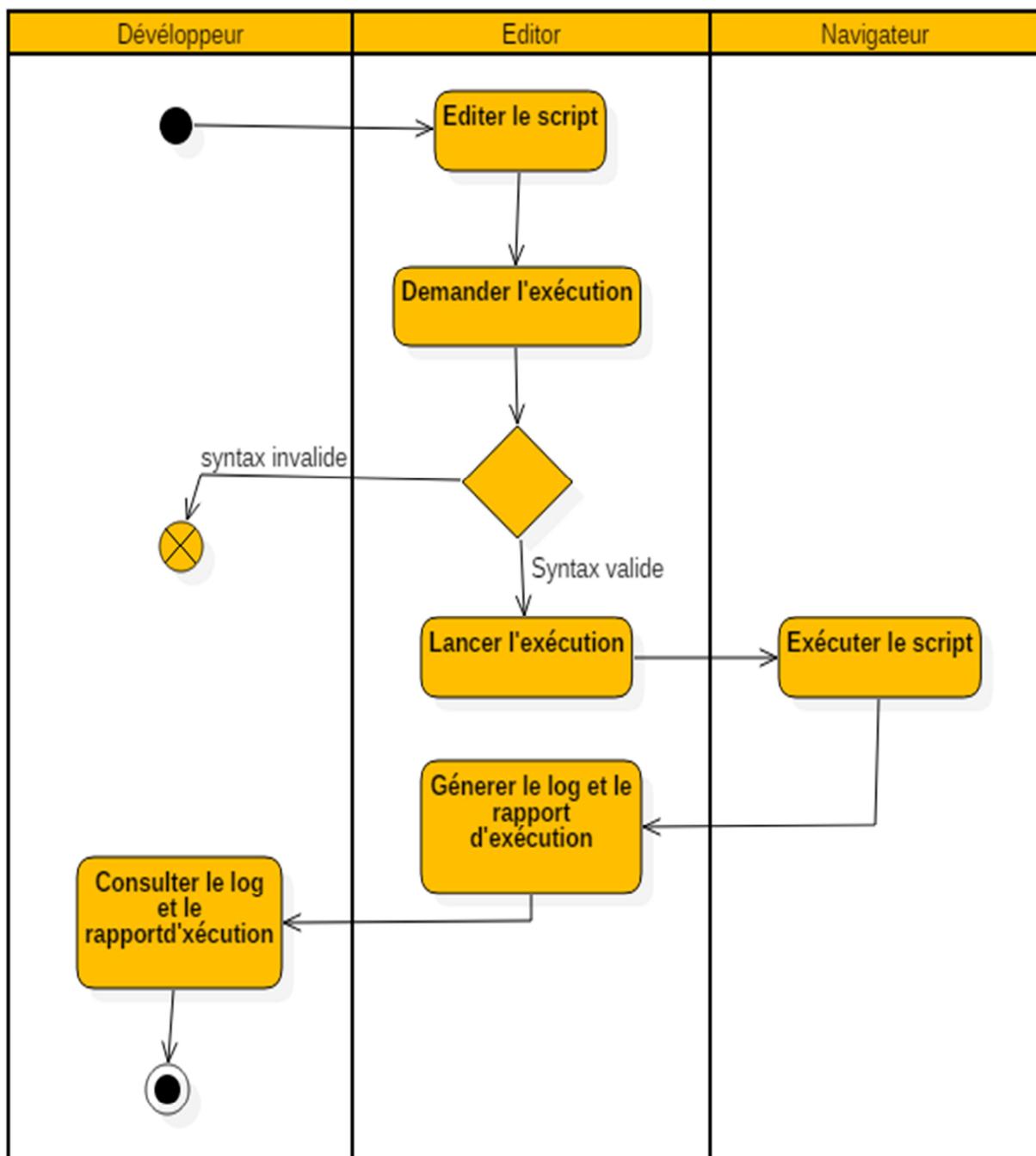


Figure 29 Diagramme d'activité pour le cas d'utilisation «Exécuter un script de cas de test »

Résumé:

Cet ouvrage est un rapport de projet de fin d'étude. Il s'agit de la documentation nécessaire pour concevoir, implémenter et intégrer un processus d'automatisation des tests fonctionnels. Ce processus comporte beaucoup de composants et outils.

Editor, le composant principal de cette solution, est le fruit de ce projet. Il s'agit d'un IDE pour éditer et exécuter les scripts des tests automatiques.

Abstract:

This book is a draft report for the final year study project. This is the documentation needed to design, implement and integrate an automation process of functional tests.

This process includes many components and tools.

Editor, the main component of this solution, is the fruit of this project. It is an IDE to edit and execute automatic test scripts.

الملخص

هذا الكتاب عبارة عن تقرير لمشروع ختم الدراسات والتخرج . هذه هي الوثائق الازمة لتصميم وتنفيذ و دمج عملية جعل الاختبارات الوظيفية الية . وتشمل هذه العملية العديد من العناصر والأدوات. المكون الرئيسي لهذا العمل هو خلاصة هذا المشروع . و هو بيئة تطوير متكاملة لتحرير و تنفيذ البرامج النصية للاختبارات.