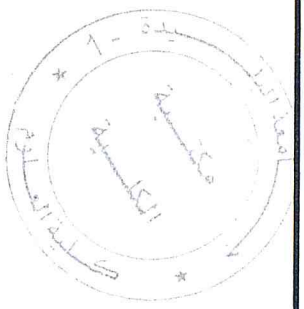
RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE



UNIVERSITÉ SAAD DAHLAB BLIDA

FACULTÉ DES SCIENCES
DÉPARTEMENT INFORMATIQUE

Mémoire de fin d'études
Pour l'obtention d'un diplôme de Master 2 en Informatique
Option : Ingénierie des logiciels

Thème

Conception et réalisation d'une plateforme de tests fonctionnels pour les applications web.

* Réalisé par :
- TAREB Mohamed Amine
- BOUDJEMA Abderrahmane

* Promoteur :
- Mr. KAMECHE Abdallah Hicham

Organisme d'accueil :Département d'Informatique

Jury :

Président : Oukid Lamia

Examinateur : Hadj Henni

Date de Soutenance : 27/06/2018

Promotion : 2017/2018

Remerciement

Grace à dieu, le tout puissant qui nous a donné le courage, la volonté et la patience pour achever notre travail.

Nous tenons à remercier vivement nos chers parents qui nous ont guidés depuis notre enfance vers le chemin du savoir.

Nous exprimons nos vifs remerciements et notre profonde gratitude à notre promoteur qui a bien voulu nous consacrer tout son temps pour nous transmettre un ensemble de connaissances avec une volonté exemplaire, son soutien, son aide, ses conseils et sa bienveillance durant l'élaboration de ce mémoire.

Nous tenons à remercier nos enseignants du département informatique de l'université de blida qui se sont donnés tant de mal pour nous transmettre le savoir.

Merci à nos familles, nos amis(es) et proches qui nous ont soutenus.

Nous remercions également toutes les personnes qui nous ont aidés de près ou de loin, à réaliser ce modeste travail.

Mohamed Amine TAREB

Abderrahmane BOUDJEMA

Dédicace

Je tiens à dédicacer ce modeste travail à tous ceux et celles qui n'ont jamais cessé de m'encourager tout au long de mon cursus universitaire, notamment :

- A mon très cher " Papa-Sidou " qui m'a énormément aidé dans les moments difficiles, encouragé durant toutes mes années d'études et leurs sacrifices. Que dieu me le garde !
- A mon très cher Père à qui je dois tous mes respects et à ma chère Mère qui n'a jamais cessé de prier pour moi, que dieu les protèges, sans eux je n'aurais jamais vu la lumière du jour ni devenu ce que je suis.
- A mon très cher frère "Tarek" que j'aime très fort.
- A ma chère sœur bien aimée "Douaa".
- A mes très chères grand-mères "Yamina & Aicha".
- A mes oncles et mes tantes.
- A toute ma famille.
- A mon binôme "Boudjema Abderrahmane".
- A tous mes amis(es) de l'université où j'ai passé avec eux des moments inoubliables.
- A toutes les personnes qui ont contribué de près ou de loin à l'aboutissement de ce travail.

Mohamed Amine TAREB

Dédicace

— A mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études

— A mes chers frères, Adel et Mohamed, pour leur appui et leur encouragement

— A toute ma famille pour leur soutien tout au long de mon parcours universitaire

— A mon binôme Tareb Mohamed Amine

— A Mes amis(es) qui n'ont cessé de m'encourager

Que ce travail soit l'accomplissement de vos vœux tant allégués, et le fruit de votre soutien infaillible.

Merci d'être toujours là pour moi.

Abderrahmane BOUDJEMA

Résumé

Le but de ce travail est l'automatisation des tests fonctionnels pour les applications web, en utilisant l'approche MBT, tout en donnant une idée sur les tests logiciels ainsi qu'en détaillant les différentes stratégies de tests. L'approche model-based testing permet de concevoir de manière automatique des cas de tests à partir d'un modèle abstrait. Dans notre cas, les cas de tests sont réalisés à partir d'un graphe, l'utilisateur de cette plateforme n'aura pas besoin de connaissance en codage des scripts pour concevoir un test. Grâce au module de drag and drop, l'utilisateur peut créer des scénarios de test plus rapidement. Une telle application permettra de détecter les défaillances d'un système dans un temps rapide et d'économiser les coûts de développements des logiciels.

mots clés— Test logiciel, Test basé sur un modèle, Test de non-régression, Plateforme de test, Test fonctionnels

Abstract

The purpose of this work is the automation of functional testing for the Web applications, by using the approach MBT, while giving an idea about the software testing, as well as by detailing the various strategies of tests.

The approach model-based testing, allows designing tests cases in automated way from an abstract model. In our case, the tests of cases are realized from a graph. The users of this platform will not need to have strong skills in scripting codes to design a test. By using the module of drag and drop, the users can create tests cases more quickly. This application is capable to detect bugs in a system in short lapse of time and economize the cost of software development.

mots clés— Software testing, Model-based testing, Regression testing, Test platform, Functional testing

نبذة مختصرة:

الهدف من هذا العمل هو جعل الاختبارات الوظيفية لتطبيقات الويب اوتوماتكي وذلك باستخدام طريقة لإختبارالقائمعلىنموذج، مع إعطاء فكرة عن اختبار البرامج مع التطرق إلى استراتيجيات الاختبار المختلفة.

يسمح النهج القائم على نموذج الاختبار بتصميم طريقة اختبار تلقائية عبرنموذج معين.

في هذا العمل، لا يحتاج المبرمج إلى معرفة تقنيات البرمجة لتصميم الإختبار. فبفضل وحدة السحب والإسقاط يمكن للمستخدم أن يحظر سيناريوهات الإختبار. مثل هذا التطبيق يساعد في الكشف عن فشل الأنظمة في وقت سريع مما يجعل تكاليف تطوير البرامج رخيص الكلفة.

الكلماتالمفتاحية:اختبارالبرمجيات،اختبارقائمعلىنموذج،اختبارالارجوع،منصةالاختبار،اختباراتوظيفية.

acronyme

- **IEEE** : Institute of Electrical and Electronics Engineers
- **SDLC** : Software Developement Life Cycle
- **HLD** : Whig Level Design
- **LLD** : Low Level Design
- **QA** : Quality Assurance .
- **STLC** : Software Testing Life Cycle.
- **SRC** : Software Requirements Specifications
- **RTM** : Requirement Traceability Matrix
- **IPO** : Input Process Output
- **ECP** : Equivalent Class Partionning
- **ISO** : International Organization for Standardization
- **MBT** : Model Based Testing
- **AETG** : Advanced Efficient Test Generation

- **IPOG** : In Parameter Order General
- **SQL** : Script Query Language
- **LOTOS** : Language of Temporal Ordered Systems
- **IOLTS** : Input Output Labeled Transition System
- **API** : Application Programming Interface
- **IDE** : Integrated Development Environment
- **HTML** : Hyper Text Modeling Language
- **IE6** : Internet Explorer version 6
- **BSD** : Berkeley Software Distribution
- **UML** : Unified Modeling Language
- **UP** : Unified Process

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Les équipes d'assurance qualité de logiciel se voient aujourd'hui projetées sur le devant de la scène. En effet, le contexte de marché très tendu, la course effrénée à l'innovation que se livrent les entreprises, l'accélération de leur compétitivité, mais également la part grandissante des tests de non-régression dans les coûts du développement qui sont entre 30% et 40% selon [Kaner et al.(1999)Kaner, Falk, and Nguyen], génèrent une pression forte sur les équipes de qualification.

Chaque mise à jour de l'application nécessiterait de refaire l'intégralité des tests de non-régression qui est un ensemble de tests d'un programme préalablement testé, après une modification, pour s'assurer que des défauts n'ont pas été introduits ou découverts dans des parties non modifiées du logiciel.

Le délai de mise sur le marché "time to market" et l'estimation qui correspond au marché "juste coût" rendent cette option impossible, même une simple application peut avoir des centaines de combinaisons d'entrées et de sorties, faire le test pour toutes les possibilités est impraticable.
Le test complet d'une application complexe prendrait trop de temps et exige trop de ressources humaines pour qu'il soit économiquement réalisable.
L'entreprise prend alors des risques pouvant aller jusqu'à des régressions sur les fonctions critiques comme les ventes ou les prises de commandes. L'automatisation des tests de non-régression devient donc un enjeu stratégique.
Afin de faciliter la tâche aux développeurs et réduire les coûts de tests manuels, il serait

donc intéressant d'automatiser les tests et rendre leur utilisation plus simple et plus rapide.

L'objectif principal est de concevoir et réaliser une plateforme de tests fonctionnels pour les applications web qui permet de faire l'automatisation des tests de non-régression avec une conception visuelle des tests permettant une représentation graphique des différents parcours de l'utilisateur ,les parcours sont schématisés en cas de tests exécutables sur l'application web . Pour cela le mémoire est organisé comme suit :

- **Chapitre 1 :** Une description détaillée sur les 3 niveaux du test logiciel dont le niveau de détail (cycle de vie), le niveau d'accessibilité et les caractéristiques de ce que l'on veut tester.
- **Chapitre 2 :** Dans ce chapitre, nous nous intéressons à l'automatisation des tests, à savoir les différents outils utilisés dans le marché et sur quelles applications sont appliquées.
- **Chapitre 3 :** Dans ce troisième chapitre nous allons faire une analyse des besoins ainsi qu'une conception du système en utilisant le langage de modélisation UML.
- **Chapitre 4 :** Dans ce dernier chapitre nous allons faire une partie sur l'implémentation du système avec quelques définitions des outils utilisés ainsi qu'une partie sur les résultats.

Chapitre 1: Test Logiciel

1.1 Introduction

Le test constitue une étape importante dans le processus de développement d'un logiciel sûr et fiable, Dans cette partie on va présenter des généralités sur le test logiciel, pour cadrer et bien définir le sujet de notre projet.
Le principal objectif d'une organisation qui réalise des tests, est de vérifier que logiciel, l'application ou le système, satisfait bien les spécifications pour les quelles ont été développé.

1.2 Définitions

- **Étymologie :** du latin. « Testum », vase d'argile. Enveloppe dure qui protège divers êtres vivants. (plaques dermiques de l'oursin, coquille des mollusques, etc.) [petit larousse(2015)]
- **Définition 1 :** La norme IEEE 729 [Jane Radatz(1990)] définit le test comme un processus manuel ou automatique qui vise à établir qu'un système vérifie les propriétés exigées par sa spécification ou à détecter des différences entre les résultats engendrés par le système et ceux qui sont attendus par spécification. [http ://w3.uqo.ca()]
- **Définition 2 :** D'après [Myers and Sandler(2004)] « Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts»

- **Définition 3 :** C'est une activité qui fait partie du processus de développement logiciel. Selon les règles de l'assurance de la qualité, ce processus débute une fois que l'activité de programmation est terminée.[Sommerville(2010)].
- **Définition 4 :** le test est un mot anglais, épreuve permettant d'évaluer les aptitudes de quelqu'un, ou d'explorer sa personnalité : test de niveau. [petit larousse(2015)]

1.3 Cycle de vie du développement logiciel (SDLC)

Il est important de présenter ce qu'un cycle de vie logiciel ainsi que les différentes phases de son développement.

1.3.1 Définitions

- **SDLC** est un acronyme utilisé pour décrire un logiciel ou un cycle de vie du développement des systèmes [Ruparelia(2010)].
- Le cycle de vie du développement logiciel (SDLC) est un cadre définissant les tâches effectuées à chaque étape du processus de développement du logiciel. Le cycle de vie du développement logiciel est une structure suivie d'une équipe de développement au sein d'une organisation.
  Il consiste en un plan détaillé décrivant comment développer, maintenir et remplacer des logiciels spécifiques. Le cycle de vie définit une méthodologie pour améliorer la qualité du logiciel et l'ensemble des processus de développement [Techopedia(2018)].

1.3.2 Les phases de développement d'un cycle de vie

Comme le montre la figure 1.1, dans plusieurs cas le cycle de vie du développement logiciel inclut ses différentes phases :

- **Planification (collecte) et analyse de besoins :** Avant toute activité humaine, il est nécessaire d'organiser et de planifier les activités à exécuter. Ceci s'applique également aux tests de logiciels et de systèmes.

  Les activités de planification des tests comprennent l'organisation des tâches et la coordination avec les autres parties prenantes, telles que les équipes de développement, les équipes de support, les représentants des utilisateurs, la direction, les clients, etc. [Homès(2013)].

  selon [istqb certified tester(2018a)] cette phase est l'objectif principal des gestionnaires de projets et des parties prenantes. Des réunions avec les gestionnaires, les intervenants et les utilisateurs sont tenues afin de déterminer les besoins tels que :

  - Qui va utiliser le système ?
  - Comment vont-ils utiliser le système ?
  - Quelles données doivent être entrées dans le système ?
  - Quelles données doivent être générées par le système ?

  Ce sont des questions générales auxquelles on répond pendant la phase de collection des besoins.

  Après la collecte des besoins, ces exigences sont analysées pour leur validité et sont également étudiées afin d'examiner la possibilité de les incorporer dans le système à développer.

  Enfin, un document de spécification des besoins est créé, qui sert de guide pour la phase suivante du modèle. L'équipe de test suit le cycle de vie du test logiciel et commence la phase de planification du test après l'analyse des besoins .

- **Conception (design) :** Cette partie concerne la conception de logiciel.

  Selon [Lekh and Pooja(2015)] il y'a deux types de conception : HLD « conception de haut niveau » et LLD « conception de bas niveau ».

  HLD nécessite des spécifications exigeantes et l'estimation de l'effort, les bases de données et les diagrammes de séquence, utilisent ce type de conception.

  Dans LLD « conception de bas niveau » les descriptions techniques de conception,

les cas de tests et les scripts de test sont tous préparés, ce type de conception est analysé par les responsables de QA « assurance de qualité » et les responsables des équipes.

- **Implémentation (codage) :** Dans cette étape de SDLC, le développement réel commence ainsi le produit est développé. Si la conception est effectuée d'une manière détaillée et organisée, la génération du code peut être accomplie sans problème. Les développeurs doivent suivre les directives de programmation définies par leurs organisations et les outils de programmation, comme les compilateurs, interpréteurs, etc qui sont utilisés pour la génération du code [tutorialspoint(2017)].
- **Tests et intégration :** Cette phase focalise sur une étude empirique dans lequel les résultats décrivent la qualité du système, le test n'approuve pas les fonctionnalités du système correctement sous toutes les conditions, mais il peut établir qu'il y'a certaines pannes dans quelques conditions, plutôt la panne est détectée dans le cycle de développement de logiciel, autant les couts de maintenances sont réduits. Tester un système démontre qu'il répond aux exigences des utilisateurs, comportant la performance et la sécurité [maryland department of technology(2007)].
- **Déploiement et maintenance :** Après un test réussi, le produit est livré / déployé chez le client pour son utilisation, dès que le produit est donné aux clients, ils vont d'abord faire le test bêta. Si des modifications sont nécessaires ou si des bogues sont détectés, ils les signalent à l'équipe d'ingénierie. Une fois ces modifications effectuées et ces bogues sont corrigés, le logiciel sera prêt pour le déploiement final.

  Une fois que les clients commencent à utiliser le système développé, les problèmes réels surgissent et doivent être résolus. Ce processus où le soin est pris pour le produit développé il est connu sous le nom de maintenance [istqb certified tester(2018a)].

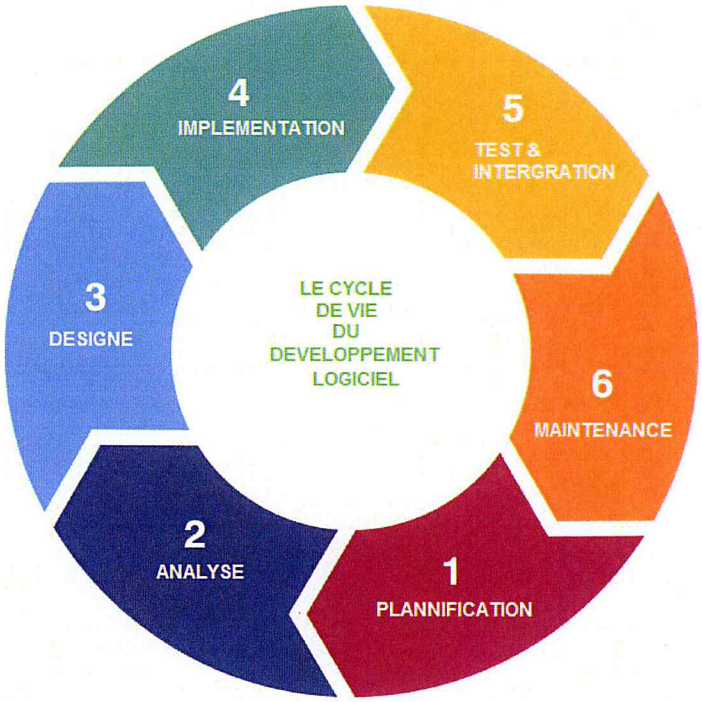


FIGURE 1.1 – Cycle de vie du développement logiciel
[rohit Singh(2016)]

1.4 Cycle de vie du test (STLC)

Tout comme les développeurs suivent le cycle de développement logiciel (SDLC), les testeurs suivent également le cycle de vie des tests logiciels appelé STLC. C'est la séquence des activités menées par l'équipe de test depuis le début du projet jusqu'à la fin du projet. Le cycle de vie de test logiciel est un processus de test qui est exécuté dans une séquence, afin d'atteindre les objectifs de qualité. Il ne s'agit pas d'une seule activité mais, de nombreuses activités différentes comme l'illustre cette figure 1.2 qui sont exécutées pour obtenir un produit de bonne qualité.

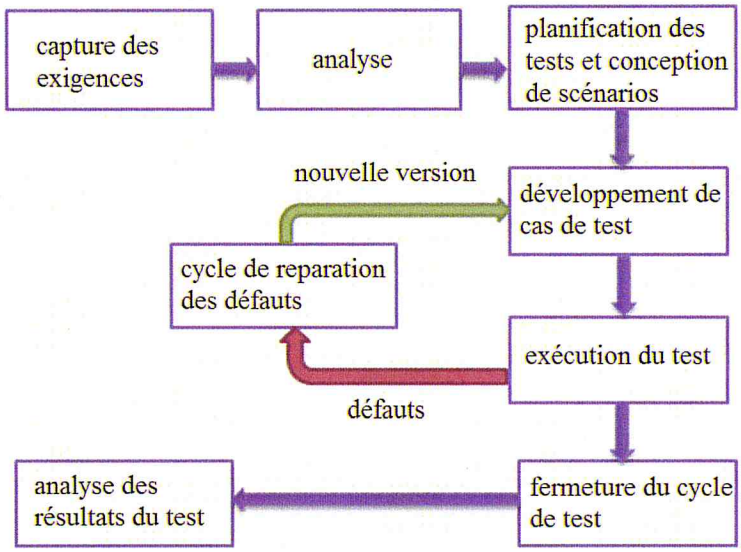


FIGURE 1.2 – Cycle de vie des tests logiciels (STLC)
[Mrs.A.Vanitha Katherine(2012)]

1.4.1 Les phases du cycle de vie du test Logiciel

- **L'analyse des besoins :** L'analyse des besoins est la première phase du STLC. Le critère d'entrée de cette phase est la fourniture de SRS (Software RequirementSpécification) [software testing class(2018a)] .

  Il est également recommandé que l'architecture de l'application soit pratique. Dans cette phase, l'équipe d'AQ « assurance de qualité » [Russell and Division(2012)] analyse, à un niveau supérieur, ce qu'il faut tester et comment tester.

  L'équipe d'assurance de qualité assure le suivi auprès de diverses parties prenantes telles que le Business Analyste, Architecture du système, Client, Test manager / Lead dans le cas où une requête ou une clarification est requise pour comprendre l'exigence.

  Les exigences peuvent être fonctionnelles ou non fonctionnelles comme la performance, la sécurité, la convivialité, etc.

  Le critère de sortie de cette phase consiste à compléter le document RTM « requirement traceability matrix » [Guru99(2018)] si nécessaire.

Le rapport de faisabilité de l'automatisation et une liste de questions pour être plus précis sur les exigences [tutorialspoint(2017)].

- **Planification du plan de test :** Le plan de test est le plus important dans le cycle de vie du test de logiciel. Cette phase est appelée aussi phase de stratégie de test. Dans cette étape, le gestionnaire de test est généralement impliqué pour déterminer l'effort et le cout de projet entier. Cette phase est lancée dès que la phase de collection des exigences est accomplie et en se basant sur l'analyse des exigences que la préparation de plan de test commence, le résultat de la phase de planification de test serait le plan de test. Une fois cette phase est terminée, l'équipe d'assurance de qualité peut commencer l'activité de développement des cas de tests [software testing class(2018b)].
- **Développement des cas de test :** Dans cette étape l'équipe d'assurance de qualité écrit des cas de test, et des scripts pour l'automatisation des tests s'ils ont exigés. La création de cas de test des données est faite dans cette phase [istqb certified tester(2018a)].
- **Configuration de l'environnement de test :** Cette étape est précédée par plusieurs activités mises en place, l'environnement de test comporte les techniques de test qui inclut des ressources supplémentaires comme l'installation des hardwares et softwares qui supportent l'automatisation des tests et son exécution.
  La préparation opérationnelle globale du test doit être bien analysée, Le hardware qui supporte la conception d'environnement d'automatisation des tests aussi doit prendre en considération la charge des scénarios de test [Ashish Shah(2017)].
- **Test d'exécution :** Cette étape commence dès que les cas de tests sont développés, et l'environnement de test est proprement mis en place.
  L'ingénieur de test n'exécute pas que les cas de test, mais aussi il peut détecter les erreurs et les bogues dans le logiciel, si un cas de test échoue, les bogues qui lui correspondent seront rapportés à l'équipe de développement par un système de traçabilité [ProfessionalQA.com(2017)].
- **Fin du cycle de test :** Une vérification par rapport aux critères de sortie de test

est essentielle pour affirmer que le test est maintenant terminé. Avant de mettre fin au processus de test, la qualité du produit est mesurée par rapport aux critères d'achèvement du test. Le critère d'entrée de cette phase est que l'exécution du scénario de test est terminée, les résultats des tests sont disponibles et le rapport de défauts est prêt [tutorialspoint(2017)].

- **Critères d'entrée et de sortie** En général, l'équipe d'assurance qualité ne procède pas à la phase suivante tant que les critères de sortie de la phase actuelle ne sont pas satisfaits. Les critères d'entrée doivent inclure l'achèvement des critères de sortie de la phase précédente.

  En temps réel, il n'est pas possible d'attendre la phase suivante avant que le critère de sortie ne soit satisfait.

  Maintenant, la phase suivante peut être lancée, si les critiques de livraison de la phase précédente ont été terminés. Dans chaque phase de STLC, les critères d'entrée et de sortie doivent être définis [tutorialspoint(2017)].

1.4.2 Importance du test

Le test logiciel est important pour deux raisons :
Premièrement, selon les enquêtes du gouvernement des États-Unis, on n'estime que 59,5 milliards de dollars dans les pertes d'affaires depuis 2000 en raison de logiciels de mauvaise qualité [Everett and McLeod(2007)] il existe plusieurs exemples.

- En Avril 2015, le terminal Bloomberg à Londres s'est écrasé en raison d'un problème du logiciel qui a affecté plus de 300 000 commerçants sur les marchés financiers. Il a forcé le gouvernement à retarder une vente de dette de 3 milliards de livres [Nathaniel Popper(2015)].
- Selon [Isidore(2014)] la firme Nissan a rappelé plus de 1 million de voitures du marché en raison d'une défaillance du logiciel dans les détecteurs sensoriels de leur airbag. Il a été signalé que deux accidents se sont produits à cause de cet échec du logiciel.

- Starbucks a été contraint de fermer environ 60% des magasins aux États-Unis et au Canada en raison de défaillance logicielle dans son système de leur point de vente. À un moment donné les magasins ont servi du café gratuitement car ils étaient incapables de traiter la transaction.[fortune.com(2015)]
- En avril 1999, un bug logiciel a causé l'échec de lancement d'un satellite militaire de 1,2 milliard de dollars, l'accident le plus coûteux de l'histoire [Ruparelia(2010)].

Deuxièmement, sur la base de l'incapacité des auteurs à trouver des testeurs de logiciels expérimentés pour une opportunité de test de 22,2 milliards de dollars, le nombre actuel des testeurs de logiciels expérimentés est déjà employé.

1.4.3 Le rôle du test dans le cycle de vie (SDLC)

On voit bien que dans le cycle de vie logiciel, le test à un rôle important dans son développement, il doit être présent tout au long du cycle de vie à partir du début de sa conception jusqu'à la fin de sa maintenance.
Les tests fournissent également des informations qui permettent aux gestionnaires de prendre des décisions éclairées, avec une meilleure compréhension au niveau de la qualité et de l'impact de leurs décisions.

1.4.4 Coût des tests

Le logiciel passe par un cycle d'étapes de développement. Un produit est imaginé, créé, évalué, fixé, utilisé. L'activité complète, à partir de la réflexion initiale à l'utilisation finale, est appelée le cycle de vie du logiciel.
Selon [Kaner et al.(1999)Kaner, Falk, and Nguyen] le cycle de vie du logiciel comporte de nombreuses étapes mais, il peut se résumer en cinq étapes de base, qui sont : la planification, la conception, le codage et la documentation, les tests et la correction, la maintenance et l'amélioration après-vente. Les coûts relatifs de chaque étape, selon [Kaner et al.(1999)Kaner, Falk, and Nguyen], peuvent être résumés, comme cité dans les tableaux 1.1 et 1.2 ci-dessous :

TABLE 1.1 – Le coût de chaque étape en pourcentage du coût total et coût du développement

[Kaner et al.(1999)Kaner, Falk, and Nguyen]

Stage	Coût total	Coût du développement
Analyse des besoins	3%	9%
Spécification	3%	9%
Design et Conception	5%	15%
Codage	7%	21%
Tests	15%	45%

TABLE 1.2 – Le coût de fonctionnement et de maintenance en phase de production en pourcentage

Opération et Maintenance	67%

Les tableaux ci-dessus montrent que la maintenance est le principal élément de coût du logiciel.

Le test est la deuxième activité la plus coûteuse, représentant 45% du coût de développement initial d'un produit. Les tests représentent aussi, une grande partie du coût de maintenance, car les modifications de code lors de la maintenance doivent également être testées.

Tester, trouver et corriger les erreurs dans les programmes peuvent être faites à n'importe quelle étape du cycle de vie et peuvent être estimées entre 40% et 80% du coût total du développement.

Cependant, le coût de la recherche et de la correction des erreurs augmente considérablement au fur-et-à-mesure que le développement progresse. La figure 1.3 montre que plus une erreur est trouvée tard, plus il en coûte pour la réparer.

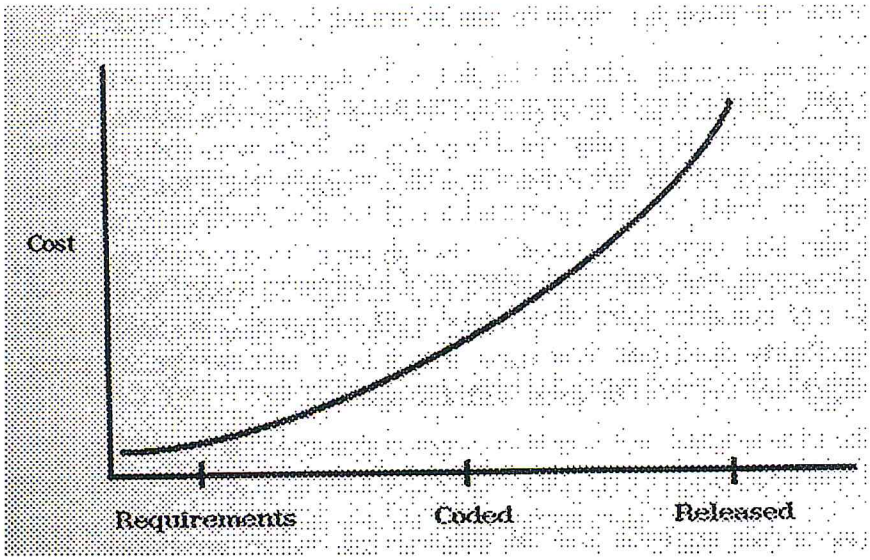


FIGURE 1.3 – Coût de la recherche et de la correction des erreurs de logiciels [Kaner et al.(1999)Kaner, Falk, and Nguyen]

Changer un document d'exigences avant que le code soit écrit coûtera moins cher que de le changer après que le code a été écrit, puisque le code doit être réécrit. La correction des défauts est beaucoup moins chère lorsque les programmeurs trouvent leurs propres défauts. Ils n'ont pas à expliquer le défaut à quelqu'un d'autre, et il n'y aura aucun coût de communication. Corriger un défaut avant de lancer un programme, est également moins cher que d'envoyer un technicien à chaque client.

Les différentes stratégies de tests

La procédure de test est très grande et il existe une infinité de tests, Tout dépend de la combinaison des données d'entrées ainsi que les aspects vérifiables (conformité aux spécifications, aux manuels d'utilisation et d'exploitation, aux exigences de robustesse, de performance d'ergonomie, du comportement en cas de défaillance, etc.)
Par conséquent, l'effort du test doit être adapté aux enjeux, Ce qui nécessite une stratégie et un but de test.
Il s'agit de définir les objectifs à même de permettre de tester en couvrant les enjeux qu'on s'est fixés, donc focaliser le test sur les points les plus fréquents et les plus importants, là où on estime que le risque est élevé. Dès lors le choix reste à définir.
Une stratégie de test doit par conséquent être mise en place, il faut alors préparer un plan de test.
La figure 1.4 montre bien les différents aspects de la gestion des tests. Aussi, nous allons nous baser sur cette classification pour décrire les différents éléments pris en compte lorsque qu'on veut bien mener une campagne de tests.

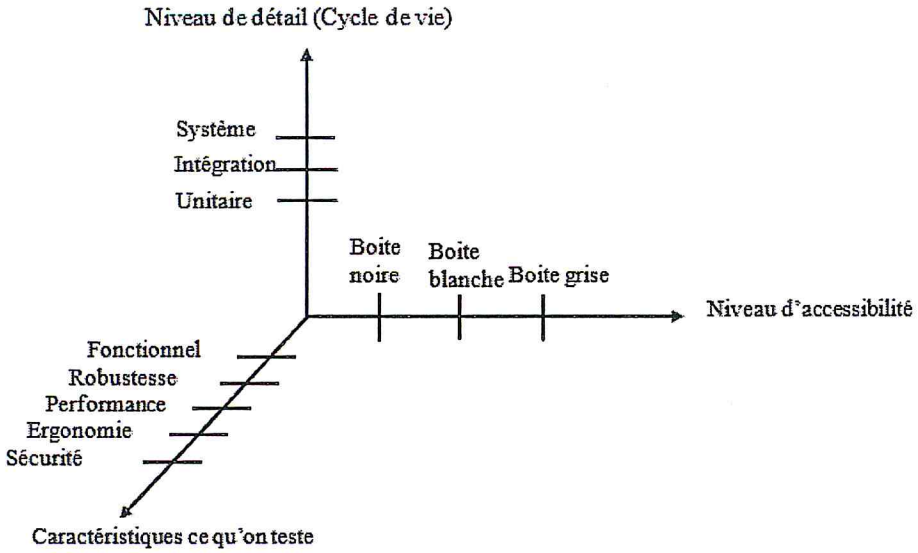


FIGURE 1.4 – Les différentes configurations possibles
[Tretmans(1996)]

1.5 Niveau de détail

1.5.1 Test unitaire

Appelé aussi test de composants, il consiste à vérifier qu'une unité de code ne comporte pas d'erreur de programmation et qu'il correspond aux exigences et respecte bien les spécifications fonctionnelles des unités qui sont définies par les programmeurs comme le montre la figure 1.5.

Dans un langage procédural, une unité peut être une fonction. Dans d'autres langages comme le langage orienté objet c'est une classe.

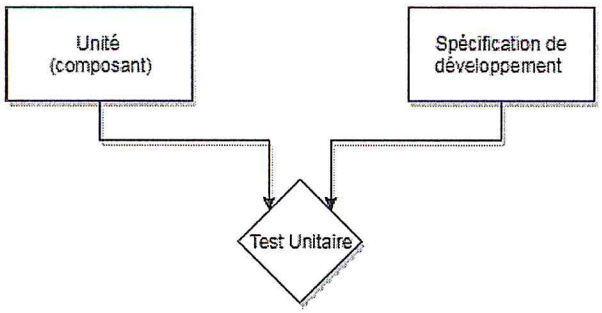


FIGURE 1.5 – le test unitaire

1.5.2 Test d'intégration

Les tests d'intégration sont les tests appliqués lorsque tous les modules sont combinés pour former un programme de travail. Les tests sont effectués au niveau du module, plutôt qu'au niveau de l'instruction, comme dans le test unitaire. Les tests d'intégration mettent l'accent sur les interactions entre les modules et leurs interfaces [G. J. Myers(1976)]. Bien que de nombreuses stratégies d'intégration aient été décrites, quelques-uns donnent des directives pour générer des cas de test.
Les tests d'intégration utilisent plusieurs unités qui ont été combinées pour former un module, un sous-système ou un système comme l'illustre la figure 1.6.

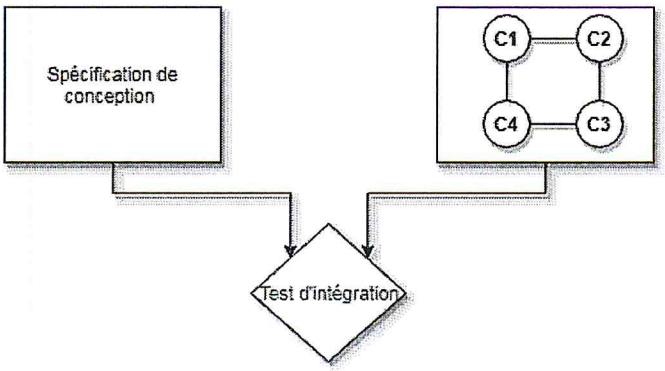


FIGURE 1.6 – Test d'intégration

1.5.3 Test système

Le test système concerne le comportement de l'ensemble du système défini par la portée d'un projet de développement ou d'un produit. Cela peut inclure des tests basés sur des spécifications de risques ou d'exigences (voir figure 1.7), des processus métier, des cas d'utilisation ou d'autres descriptions de haut niveau du comportement du système, des interactions avec le système d'exploitation et des ressources système.
Le test système est souvent le test final effectué dans le développement pour vérifier que le système est prêt à être délivré et répond aux spécifications. La plupart du temps, il est réalisé par des testeurs spécialistes qui forment une équipe spécifique, qui est parfois indépendante au sein du développement, rendant compte au responsable du développement ou au chef de projet. Les tests système doivent examiner les exigences fonctionnelles et non fonctionnelles du système.
Les testeurs peuvent également avoir besoin de répondre à des exigences incomplètes ou non documentées. Dans le test système, les exigences fonctionnelles commence par l'utilisation des techniques basées sur des méthodes de spécifications appropriées (boîte noire) pour l'aspect du système à tester [Graham et al.(2006)Graham, Black, Van Veenendaal, and Evans].

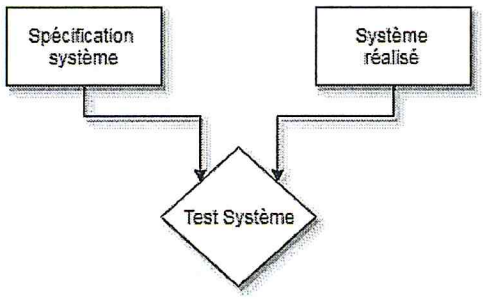


FIGURE 1.7 – Test système

1.5.4 Test d'acceptation :

C'est le processus de comparaison du produit final aux besoins actuels de ses utilisateurs finaux. Le système est testé avec les données fournies par l'acheteur du système plutôt que par des données de test simulées. Généralement, il est effectué par le client

ou l'utilisateur final. Le système est vérifié par rapport à la description des besoins du client.

1.6 Niveau d'accessibilité

1.6.1 Test en boite noire

Le terme (test en boîte noire) est utilisé pour décrire les tests qui sont dérivés principalement de la spécification d'un programme. En principe, le code source du programme interne n'est pas pris en considération comme le montre la figure 1.8. Les données de test dérivées de la spécification sont utilisées pour tester systématiquement le comportement du programme (entrées/sorties) [Murnane and Reed(2001)].

L'objectif est de générer un ensemble de tests qui ne teste que les exigences fonctionnelles du programme. Les types de tests dans cette catégorie comprennent :

- Les tests de classe d'équivalence
- L'analyse des valeurs aux limites
- La représentation graphique de cause-à-effet
- Test d'état transition

En utilisant l'approche de la boîte noire, un testeur considère le logiciel sous test comme une boîte opaque. Il n'a aucune connaissance de sa structure interne (c'est-à-dire, comment le logiciel fonctionne). Le testeur a seulement la connaissance de ce que le logiciel fait, mais, il n'a pas idée sur comment il le fait.

Cette approche peut varier d'un simple module, d'un nombre de fonctions ou d'un groupe d'objets à un sous-système ou à un système logiciel.

La description du comportement ou la fonctionnalité de logiciel sous test peut provenir d'une spécification formelle, d'un diagramme de processus entrée et sortie (IPO), ou un ensemble bien défini de pré-conditions.

Une autre source d'information, c'est le document de spécification des exigences (SRS) qui décrit généralement la fonctionnalité du logiciel à tester, ses entrées et les résultats

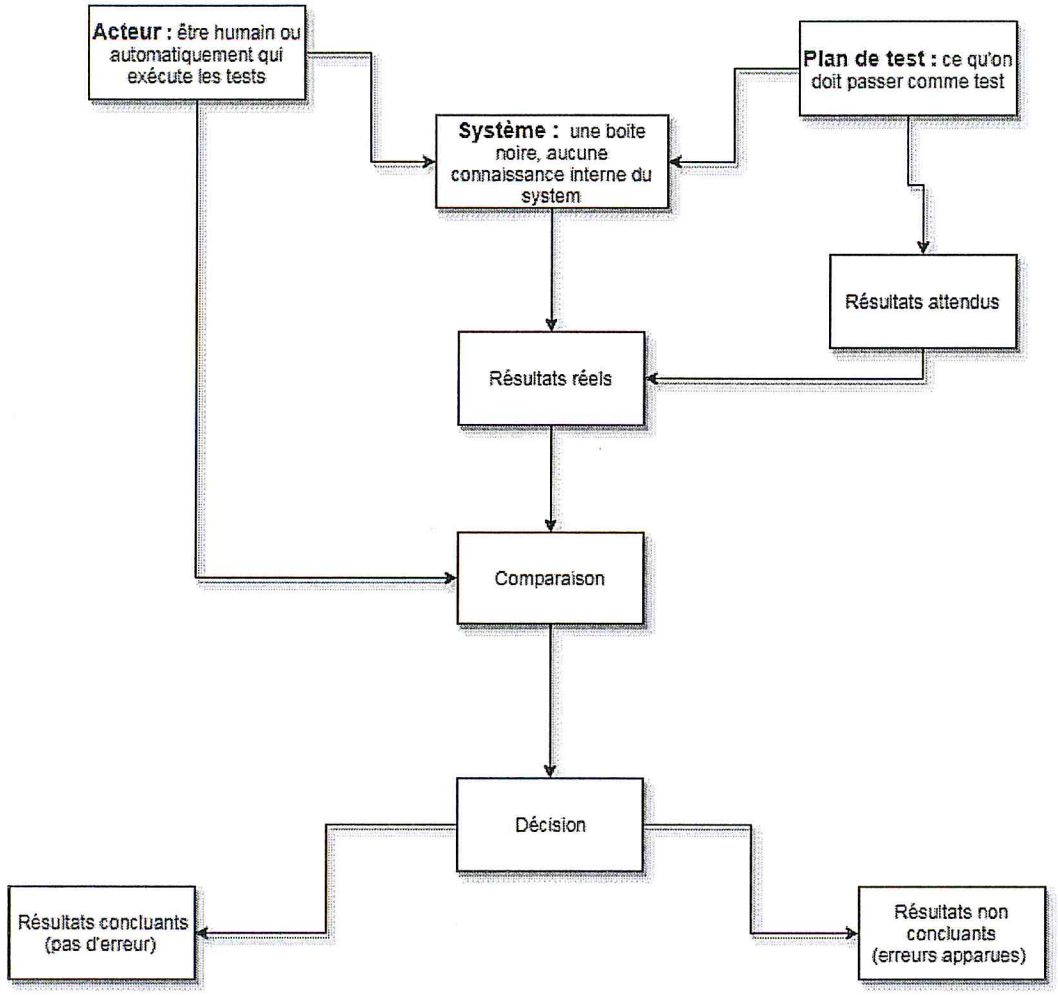


FIGURE 1.8 – Test en boite noire

attendus. Le testeur fournit les entrées spécifiées au logiciel, exécute le test et détermine si les sorties sont équivalentes à celles de la spécification mentionnée dans le document. L'approche en boîte noire ne considère donc, que le comportement et la fonctionnalité du logiciel, appelé souvent un test fonctionnel basé sur des spécifications.

Cette approche est particulièrement utile pour révéler les exigences et les défauts de fonctionnalités [Murnane and Reed(2001)].

1.6.2 Approche de test en boite noire

1.6.2.1 Tests aléatoires

Chaque composant d'un logiciel ou un système possède un domaine d'entrée à partir duquel les tests des données d'entrée sont sélectionnées. Si un testeur sélectionne aléatoirement des entrées du domaine, on appelle ça le test aléatoire [Murnane and Reed(2001)].

1.6.2.2 Partitionnement des classes d'équivalence

Le parionntitionnement d'équivalence ou le partitionnement de classe d'équivalence (ECP) est une technique de test de logiciel qui divise les données d'entrée d'une unité logicielle en partitions de données équivalentes à partir desquelles des cas de test peuvent être dérivés.
En principe, les cas de test sont conçus pour couvrir chaque partition au moins une fois. Cette technique tente de définir des cas de test qui permettent de découvrir des classes d'erreurs, réduisant ainsi le nombre total de cas de test à développer [Burnstein(2003)]. En utilisant le partitionnement de classe d'équivalence, une valeur de test dans une classe particulière est équivalente à une valeur de test de tout autre membre de cette classe. Si un cas de test dans une classe d'équivalence particulière révèle un défaut, alors, tous les autres cas de tests basés sur cette classe devraient révéler le même défaut.
Si un test élémentaire dans une classe d'équivalence donnée n'a pas détecté un type particulier de défaut, alors aucun autre cas de test basé sur cette classe ne détecte le défaut (à moins qu'un sous-ensemble de la classe d'équivalence se trouve dans une autre classe d'équivalence, car, les classes peuvent se chevaucher dans certains cas) [Burnstein(2003)]. L'utilisation de cette technique de test a les avantages suivants :

- Eliminer le besoin de tests exhaustifs.
- Guider un testeur dans la sélection d'un sous-ensemble d'entrées de test qui a une grande probabilité de détecter une faille dans le logiciel.
- Permettre au testeur de couvrir un domaine d'entrées / sorties plus grand avec un sous-ensemble plus petit, sélectionné à partir d'une classe d'équivalence.

1.6.2.3 Analyse des valeurs aux limites

Le partitionnement de classe d'équivalence donne au testeur un outil nécessaire avec lequel il développe des tests basés sur la boîte noire. La méthode nécessite qu'un testeur ait accès à une spécification de comportement d'entrée / sortie de logiciel à tester.
Les cas de test développés et basés sur le partitionnement des classes d'équivalence, qui peuvent être renforcés par l'utilisation d'une autre technique appelée analyse de la valeur aux limites.
Avec l'expérience, les testeurs réalisent rapidement que de nombreux défauts se produisent au-dessus et en dessous des limites des classes d'équivalence et dans les classes. Les cas de test qui considèrent ces limites sur les deux espaces (d'entrée / sortie) sont souvent utiles pour révéler des défauts [Burnstein(2003)].

1.6.2.4 la représentation graphique de cause-effet

Le point faible de partitionnement des classes d'équivalence est qu'il ne permette pas aux testeurs de combiner les conditions. Les combinaisons peuvent être couvertes dans certains cas par des tests générés à partir des classes.
La représentation graphique de cause-effet est une technique qui peut être utilisée pour combiner les conditions et dériver un ensemble de cas de test pouvant révéler des incohérences dans une spécification de composant. Cependant, la spécification doit être transformée en graphe, cela ressemble à un circuit logique numérique. Le testeur n'est pas obligé d'avoir un fond de connaissance électronique, mais, il devrait avoir une connaissance de logique booléenne [Burnstein(2003)].

1.6.2.5 Test d'état transition

Le test d'état transition est utile pour le développement en procédural ou en orienté objet. Il est basé sur les concepts d'états et de machines à états finis, et permet au testeur de visualiser le logiciel en développement, en fonction de ses états, des transitions entre états et des entrées d'événements qui déclenchent les changements d'états comme l'illustre cette figure. 1.9 [Burnstein(2003)].

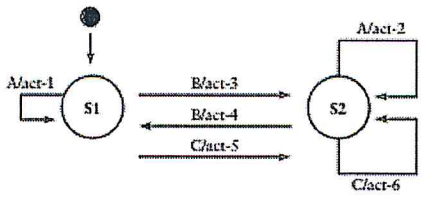


FIGURE 1.9 – Graphe d'état transition

Cette vue offre au testeur une opportunité supplémentaire de développer des cas de tests pour détecter les défauts qui peuvent ne pas être révélés en utilisant la condition d'entrée/sortie ainsi que par les vues de cause à effet présentées par partitionnement de classe d'équivalence et graphes de cause à effet.

1.6.3 Test en boite blanche

Le test en boite blanche est un procédé qui consiste à construire un ensemble de cas de test sur la base de la structure interne du produit à tester.
[Myers and Sandler(2004)] a suggéré que même un bon test en boîte noire ne peut tester que 50% à 70% du code [Farrell-Vinay(2007)]. Il faut donc compléter les tests en boîte noire en choisissant un cas de test qui assure que l'unité a entièrement subit aux tests à un certain niveau. C'est le but de la technique de test en boîte blanche. La situation idéale serait de tester chaque chemin d'entrée-sortie dans l'unité. Voici donc les couvertures de test qu'il faut prendre :

- **La couverture de ligne :**La couverture de ligne : Les tests de couverture de ligne nécessitent de concevoir un ensemble des cas de test, cela implique l'exécution de chaque ligne déclarée dans le programme. Certains tests peuvent provenir des techniques de la boîte noire. L'objectif est de compléter ces cas de tests avec d'autres cas de tests pour assurer que chaque déclaration est exécutée complètement, plutôt qu'environ 50% seulement.
- **Couverture de décision :** Couverture de décision : Cela oblige une conception des cas de tests pour que chaque décision ait un résultat vrai ou faux au moins

une fois [Farrell-Vinay(2007)].

- **Couverture des conditions :** Une condition est une expression booléenne élémentaire dans une décision nécessitant des tests pour chaque valeur possible d'une condition apparemment plus complexe que la couverture de décision, mais ne l'englobe pas [Mitra et al.(2011)Mitra, Chatterjee, and Ali].

1.6.4 Différences entre les tests en boite noire et boite blanche

Les tests en boite noire vérifient ce que le programme est censé faire tandis que le test en boite blanche vérifie ce qu'il fait réellement. Et voici le tableau 1.3 comparatif qui explique la différence entre les deux types de test.

TABLE 1.3 – La différence entre les tests en boite noir et boite blanche [Ehmer and Khan(2012)]

Test en boite noire	Test en boite blanche
Analyse fondamentale des aspects	Connaissance complète des mécanismes internes du logiciel
Niveau de granularité très bas	Niveau de granularité élevé
Effectué par les utilisateurs finaux et aussi les testeurs et les développeurs (test d'acceptation d'utilisateur)	Effectué par les testeurs et les développeurs
Le test est basé sur l'exception externe, avec négligence le comportement interne du programme	Le test est basé sur les exceptions internes du logiciel
Moins Exhaustif et le coût en terme de temps est petit par rapport au test en boite blanche	Plus exhaustif et le coût en terme de de temps est plus grand
Basé sur la méthode des tests et erreurs	Meilleur pour le test des domaines de données et les limites internes peuvent être mieux testés
Il n'est pas convenable pour le test algorithmique	Convenable pour le test algorithmique

1.7 Caractéristiques de qualité

En termes de sélection de logiciels et de systèmes fonctionnels et non fonctionnels, les caractéristiques sont également importantes, et leurs évaluations à travers des tests, permettent leurs mesures.

La norme internationale ISO 9126 propose un modèle de qualité pour les logiciels et les systèmes. Ce modèle est organisé autour de six grandes caractéristiques (fonctionnalité, fiabilité, rentabilité, efficacité, maintenabilité et portabilité). Ces caractéristiques

possèdent aussi des sous-caractéristiques comme l'illustre cette figure 1.10. L'objectif du test fonctionnel est de valider le comportement du logiciel contre les fonctionnalités métier documentées dans les exigences et les caractéristiques du logiciel. La fonctionnalité métier est généralement définie comme les activités qui soutiennent les processus métier quotidiens. Les tests fonctionnels sont réalisés par une série de test exercée sur des logiciels qui permettent directement aux utilisateurs d'accomplir ce processus métier quotidien [Homès(2013)].

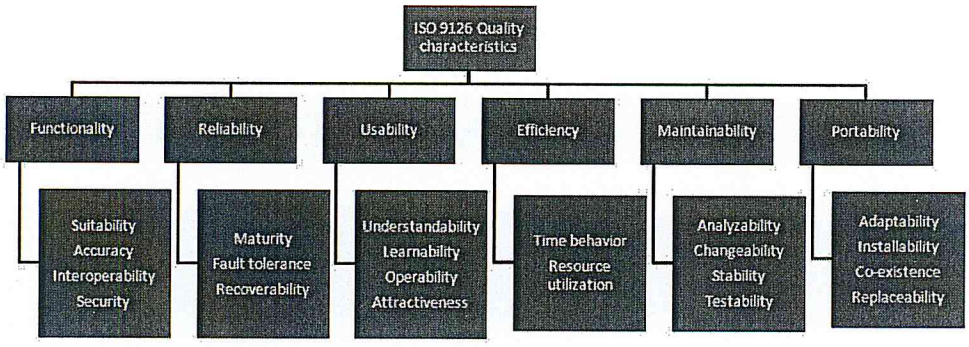


FIGURE 1.10 – Caractéristiques de qualité selon ISO 9126 [Abran(2010)]

Chacune de ces caractéristiques et sous-caractéristiques peuvent être soumises à des tests.

Souvent ces caractéristiques sont divisées comme suit :

- les caractéristiques fonctionnelles : qui comprennent les fonctionnalités fournies par le système.
- les caractéristiques non fonctionnelles : qui comprennent d'autres caractéristiques telles que la fiabilité, la convivialité, l'efficacité la maintenabilité et la portabilité.

Ces caractéristiques, fonctionnelles ou non, peuvent être testées à chaque niveau de test, avec différentes techniques de test [Homès(2013)].

1.7.1 Test fonctionnel :

permet de vérifier les fonctionnalités d'un système, qu'une méthode fonctionne correctement et qu'elle retourne les résultats attendus, c'est la conformité vis-à-vis des spécifications du système [Homès(2013)].

1.7.2 Test de performance :

Les tests de performance sont souvent exécutés lorsque le système est achevé et fonctionne correctement, ce qui est très proche de la date de livraison prévue.
Il permet de vérifier certains paramètres du système comme la taille mémoire, les délais de transfert de données, le temps d'exécution de ses fonctionnalités [Homès(2013)].

1.7.3 Test d'ergonomie :

appelé aussi test d'accessibilité, c'est tester à quel degré le système est facile à utiliser et correspond aux exigences des utilisateurs [Homès(2013)].
Le test d'ergonomie révèle si le client se sent confortable avec le logiciel, selon différents paramètres « vitesse d'exécution des taches, contenu, navigation rapide, etc » [istqb certified tester(2018b)].

1.7.4 Test de sécurité :

Ce type de test évalue les caractéristiques du système qui sont associées à l'intégrité et à la confidentialité des données du système. Les clients doivent être encouragés pour dévoiler leurs impératifs de sécurité, pour que les problèmes de sécurité soient traités par les développeurs et les testeurs du système [Homès(2013)].

1.8 Test de non régression

Ce type de test consiste à exécuter une batterie prédéfinie de tests sur des versions successives d'une application pour vérifier que les bogues sont corrigés et que les fonc-

tionnalités qui fonctionnaient dans la version précédente n'ont pas été endommagées. Les tests de non-régression sont une partie essentielle des tests, mais, ils sont très répétitifs et peuvent devenir fastidieux lorsqu'ils sont exécutés manuellement après chaque correction de bogue. Ce type de test qui devient de plus en plus important doit être fait tout au long du cycle de développement, comme l'illustre la figure 1.11.

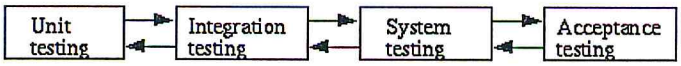


FIGURE 1.11 – Application du test de non régression dans le processus de test

Toutes ces phases doivent être effectuées aussi souvent que possible. La relation entre le test de non régression et les autres types de tests mentionnés ci-dessus sont illustrés dans la figure 1.11. Les flèches en haut des cases indiquent la séquence normale de test tandis que les flèches revenantes à la case précédente indiquent que les étapes de test précédentes peuvent être répétées, c'est-à-dire faire un test de non régression qui peut également conduire à modifier des cas de test existants ou à créer de nouveaux cas de test.

Deux types de test de non régression peuvent être identifiés basés sur la modification possible de spécification, selon [Leung and White(1989)] :

- **Test de non régression progressive :** n'impliquant que la spécification modifiée. Chaque fois que de nouvelles améliorations ou de nouvelles exigences de données sont incorporées dans un système, la spécification sera modifiée pour refléter ces ajouts. Dans la plupart des cas, de nouveaux modules seront ajoutés au logiciel avec pour conséquence que le processus de test de régression consiste à tester un programme modifié par rapport à une spécification modifiée.
- **Tests de régression corrective :** c'est quand la spécification ne change pas. Seules certaines instructions du programme et éventuellement certaines décisions de conception sont modifiées.

Ces procédures sont utiles pour :

- Tester rapidement les bogues.

- Voir et attendre ce qui va se passer après avoir résolu les bogues, c'est-à-dire quand on résout un bogue, d'autres bogues peuvent surgir.
- Avoir un code stable

1.9 Conclusion

Les stratégies mentionnées dans le chapitre précédent sont indispensables dans le test de cycle de vie du logiciel, elles réduisent le temps et les prix de maintenance d'un produit.

Ces tests ont un impact sur la sécurité de la vie quotidienne et les risques économiques dus aux erreurs faites par un programme ou un logiciel, Ils décrivent également comment les failles trouvées, liées aux produits, sont corrigées au niveau du test. Ils sont créés à partir des documents qui contiennent des exigences et des spécifications de logiciel à tester et, à partir de ces derniers, il est possible de tester le comportement du logiciel, tout en vérifiant si les sorties obtenues de la part des entrées données par les testeurs correspondent aux sorties notées dans les documents. Tout ça en passant par de multiples processus et types de tests , et c'est au testeur de voir quel est le test à appliquer sur le logiciel (sécurité, performance, etc).

Chapitre 2: Automatisation des tests

2.1 Introduction

Ce chapitre est un état d'art des outils d'automatisation de test logiciels et de ces approches, tout d'abord nous allons parler d'automatisation des tests en général, ensuite on va donner quelques approches de tests et éventuellement quelques recherches académiques, enfin nous allons présenter quelques outils d'automatisation et faire une comparaison entre ces derniers.

2.2 Définition

L'automatisation est un terme technologique général qui est utilisé pour décrire chaque processus automatisé avec l'utilisation des ordinateurs et des logiciels [Vangie and Beal()]. Le test automatique est une méthode dans le test logiciel qui fait usage des outils de test spécifique pour contrôler l'exécution de test ,pour ensuite comparer les résultats actuels avec les résultats attendus . Tout cela est fait automatiquement sans intervention humaine, ou presque, car cela doit en effet passer par une phase de préparation de test [Janssen()].

2.3 Les Approches de tests

2.3.1 L'approche MBT model based testing

2.3.1.1 Définition

Selon le groupe ISTQB le Model-Based Testing (MBT) est une approche de test s'appuyant sur la modélisation. Cette approche étend les techniques classiques de conception de tests telles que le partitionnement des classes d'équivalence, l'analyse des valeurs aux limites, le test par tables de décision, et le test à partir de diagrammes d'états-transitions ou de cas d'utilisation[Stephan Christmann(2016)].
L'idée essentielle de l'approche MBT est d'améliorer la qualité et l'efficacité des pratiques d'analyse, de conception et d'implémentation des tests par :

- la mise en œuvre outillée de modélisation pour le test permettant de satisfaire les objectifs de test du projet.
- la mise en œuvre de la modélisation MBT comme spécification de conception des cas de test. Les modèles MBT incluent des informations suffisamment précises pour permettre la génération automatique des cas de test directement à partir du modèle 2.1.

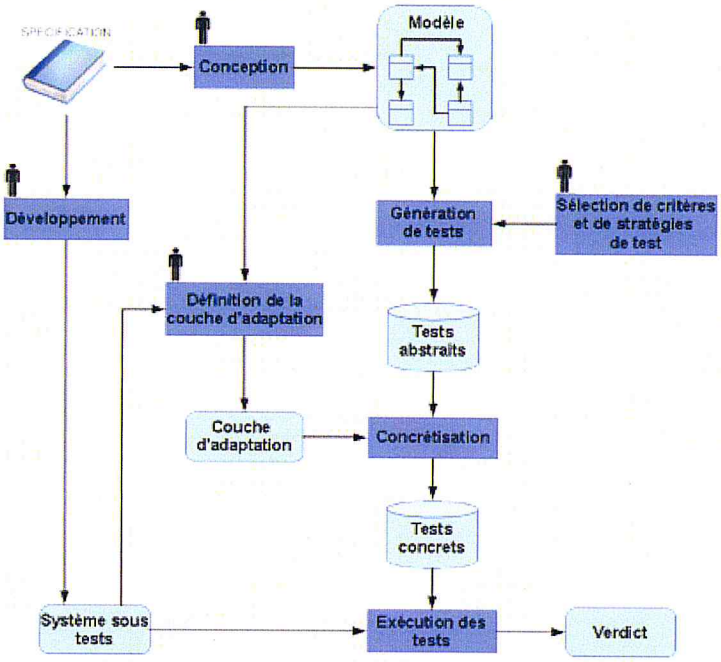


FIGURE 2.1 – Processus classique du model-based testing . [Castillos(2013)]

2.3.1.2 Motivations principales du MBT

Il y a deux aspects principaux dans les activités du test logiciel [Stephan Christmann(2016)] qui expliquent les motivations principales du MBT et qui expliquent comment le MBT contribue à l'amélioration des pratiques de test :

- **L'amélioration de l'efficacité :**
  - La modélisation MBT renforce la communication entre les partie-prenantes du projet.
  - L'amélioration de la communication permet une compréhension partagée des exigences et facilite la détection de mécompréhension potentielle dans ces exigences
  - Dans le cas de modèles MBT graphiques, les participants au projet (tels que les Analystes Métier) peuvent être impliqués plus facilement.

- La modélisation MBT facilite une augmentation continue de la compétence des testeurs sur le domaine Métier.
- Le niveau d'abstraction des modèles MBT rend plus facile l'identification des parties du système où se poseront le plus probablement des problèmes.
- La génération et l'analyse des cas de test est possible avant que le système soit réellement implémenté.

- **L'amélioration de l'efficience :**
  - Les artefacts MBT issus de précédents projets peuvent être réutilisés et adaptés pour de nouveaux projets.
  - Le MBT peut être utilisé pour différents objectifs de test et pour couvrir différents niveaux et types de test.
  - Le MBT permet de réduire les coûts de maintenance lorsque les exigences évoluent car le modèle MBT crée un point de maintenance unique.

2.3.1.3 Activités et artefacts du MBT dans le processus de test

— **Activités spécifiques du MBT :** Lorsque le MBT est déployé sur un projet, le processus de test est adapté avec les activités spécifiques du MBT, telles que :

– les activités de modélisation MBT (définition des règles de modélisation, développement et maintenance des modèles, administration des modèles, mise en place des outils).

– La génération d'artefacts (par exemple des cas et scripts de test, de la matrice de traçabilité entre les exigences et les tests) à partir des modèles MBT et des critères de sélection de tests.

Au minimum, le MBT modifie les phases d'analyse, de conception et d'implémentation des tests. En fonction des objectifs de test du projet, le MBT peut influencer la totalité des phases du processus de test.

Cela concerne en particulier :

- La planification et le suivi des tests qui doivent prendre en compte les activités spécifiques du MBT avec des métriques spécifiques.
- L'analyse et la conception des tests qui s'appuient sur la modélisation MBT, la mise en œuvre de critères de sélection de tests et de métriques spécifiques de couverture des tests.
- L'implémentation et l'exécution qui s'appuient sur la génération automatique des tests avec les outils MBT et l'adaptation des tests.
- L'évaluation des critères d'arrêt des tests et le reporting qui utilisent des métriques de couverture spécifiques (par exemple fondées sur la couverture du modèle MBT) et une analyse d'impact supportée par les modèles.
- La phase de clôture des tests peut inclure la mise en place de bibliothèques de modèles MBT pour une réutilisation ultérieure.

  Le MBT contribue à automatiser le processus de test et la production des artefacts de test (tels que les cas et scripts de test et la matrice de traçabilité entre les exigences et les tests). Le MBT favorise un déplacement du test vers les phases en amont du cycle de vie du logiciel comparé aux techniques de conception classique de tests. Cela contribue à une vérification/validation en amont des exigences et améliore la communication, en particulier dans le cas de modèles graphiques. Les tests MBT viennent souvent renforcer des tests existants créés manuellement. Cela permet à l'équipe de test de vérifier la consistance entre les deux types de test.

— **Principaux artefacts du MBT (en entré et en sortie) :** Les modèles MBT peuvent provenir soit, de modélisations spécifiques au test soit, d'une réutilisation de modèles développés lors de la conception du logiciel.

La modélisation MBT peut être réalisée à différents niveaux d'abstraction et intégrer différentes informations pour le test. Les artefacts générés à partir des modèles MBT reflètent les choix de niveau d'abstraction.

En fonction des informations de test prises en compte et du niveau d'abstraction,

le MBT s'intègre au processus de test au travers de différents artefacts d'entrée et de sortie.

Artefacts en entrée des activités du MBT :

- Stratégie de test.
- Bases de tests, en particulier les exigences, cibles de test, conditions de test, informations orales et modélisations ou éléments de conception existants.
- Rapports d'anomalies et d'incidents, registres de test, rapports d'exécution des tests issus de tests précédents.
- Guides méthodologiques décrivant le processus de test, documentations des outils.

Artefacts de sortie produits pas les activités du MBT :

- Modèles MBT.
- Des parties du plan de test (configuration de l'environnement de test), ordonnancement des tests, métriques de test.
- Scénarios de test, suites de test, ordonnancement de l'exécution des tests, spécifications de conception des tests.
- Cas de test, spécifications des procédures de test, données de test, scripts de test, couche d'adaptation des tests (spécifications et code).
- Matrice bidirectionnelle de traçabilité entre les tests générés et les bases de test, en particulier les exigences, et les rapports d'anomalies

Le Model-Based Testing, ses activités et ses artefacts sont totalement intégrés aux processus, méthodes, environnements, techniques et outils du cycle de vie du logiciel.

2.3.2 Le système AETG (Advanced Efficient Tests Generation)

Le système AETG basé sur l'approche de conception combinatoire, il utilise de nouveaux algorithmes combinatoires pour générer des ensembles de tests qui couvrent toutes les combinaisons n-nœuds valides. La taille d'un ensemble de tests AETG augmente logarithmiquement, dépendant du nombre de paramètres de test. Cela permet aux testeurs

de définir des modèles de tests avec des dizaines de paramètres. Le système AETG est utilisé dans une variété d'applications pour l'unité, le système et tests d'interopérabilité. Il génère à la fois des plans de test de haut niveau et des cas de test détaillés. Dans plusieurs applications, il réduit considérablement le coût du développement d'un plan de test [Cohen et al.(1997)Cohen, Dalal, Fredman, and Patton].

2.3.3 La stratégie d'IPOG

La stratégie d'IPOG est expliquée en deux étapes :
Les chercheurs voulaient développer une stratégie de test qui peut être appliquée aux applications logicielles de manière générale. Ainsi, la stratégie ne devrait pas mettre de restrictions sur la configuration du système en cours de test. Cette considération favorise les approches computationnelles par rapport aux approches algébriques.
Deuxièmement, les tests de t-way généraux ont une demande plus stricte sur les exigences de temps et d'espace que les tests par paires.
C'est parce que le nombre de combinaisons augmente exponentiellement à mesure que la force de la couverture augmente. Cette considération favorise la stratégie IPO par rapport à d'autres stratégies telles que 'AETG' et techniques de recherche heuristiques.
A noter que la stratégie IPO est déterministe, c'est-à-dire qu'elle produit le même ensemble de test pour la même configuration de système.
Le cadre de la stratégie IPOG peut être décrit comme suit :
Pour un système avec 't' paramètre ou plus, la stratégie d'IPOG construit un test de t-way pour les 't' premiers paramètres, étend l'ensemble de test pour construire un test de t-way pour les 't+1' premiers paramètres, puis continuer à étendre le test jusqu'à ce qu'il construit un ensemble de test t-way pour tous les paramètres. Les paramètres peuvent être dans un ordre arbitraire.
L'extension d'un ensemble de test t-way existant pour un paramètre supplémentaire est faite en deux étapes [Lei et al.(2007)Lei, Kacker, Kuhn, Okun, and Lawrence] :

- **croissance horizontale :** qui étend chaque test existant en ajoutant une valeur

pour le nouveau paramètre.

- **croissance verticale :** qui ajoute de nouveaux tests, si nécessaire, à l'ensemble des tests produits par la croissance horizontale.

Avantages :

- Ils peuvent être appliqués à une configuration de système arbitraire, car il n'y a pas de restriction sur le nombre de paramètres et le nombre de valeurs que chaque paramètre peut prendre.
- Ils peuvent être facilement adaptés pour la priorisation des tests et le traitement des contraintes [Nie and Leung(2011)].

Inconvénients :

Parmi les inconvénients des algorithmes basés sur les approches computationnelles :

- Ils impliquent explicitement l'énumération de toutes les combinaisons possibles à couvrir. Lorsque le nombre de combinaisons est grand, l'énumération explicite peut être prohibitive en termes d'espace pour stocker ces combinaisons et le temps nécessaire pour les énumérer.
- Ils sont typiquement gourmands, dans le sens où ils construisent des tests localement optimisés, et qui ne conduisent pas nécessairement à un ensemble de tests globalement optimisés. Ainsi, les ensembles de tests générés à partir d'approches de calcul sont souvent non minimaux [Nie and Leung(2011)].

2.3.4 Méthode de test basé sur les risques

La méthode de test basé sur les risques priorise les caractéristiques, les modules, les fonctions d'un logiciel sous le test basé sur les effets et la probabilité de défaillance, elle entraine l'analyse des risques en basant sur la complexité et la fréquence de l'utilisation [guru99(2018b)].

Le risque est l'occurrence des événements incertains avec un effet négatif ou positif sur les critères de réussite d'un projet. Ce risque peut être des événements qui se sont produits

dans le passé, ou des événements actuels, ou bien qui vont être produits. Ces événements incertains peuvent avoir un impact sur le coût du projet et de sa qualité[guru99(2018b)].

Les risquent se divisent en deux catégories :

- **Risques positifs :** qui sont référés comme étant des opportunités et aide la stabilité de métier.
- **Risques négatifs :** qui sont référés comme étant des menaces et des recommandations qui doivent être éliminés ou minimisés [guru99(2018b)].

2.3.4.1 Les motivations d'utilisation de la méthode de test basé sur des risques

- Des projets où les analyses basés sur les risques peuvent être utilisés pour détecter les vulnérabilités (comme SQL injection).
- Test de sécurité dans l'environnement de l'informatique en nuage. (cloud computing).
- Des nouveaux projets qui ont des facteurs de risques plus élevés, comme le manque de l'expérience dans la technologie utilisée.

2.3.5 Approches par modèles de fautes

Ces approches reposent sur l'introduction volontaire d'erreurs dans le modèle afin de déterminer les objectifs de test qui permettent la génération de tests dédiés à leur détection.

Dans [Aichernig and Delgado(2006)], les auteurs génèrent des mutants pour des modèles LOTOS. Ces modèles sont ensuite traduits en tant qu'IOLTS puis simplifiés par une relation d'équivalence appelée Safety Equivalence. Ensuite, les deux modèles sont utilisés pour déterminer s'il existe une relation de bi-simulation forte entre les deux. Si les IOLTS sont équivalents, alors il s'agit peut-être d'un mutant équivalent. Dans le cas contraire, un contre-exemple est trouvé. Il est ensuite complété par des transitions supplémentaires permettant d'observer la faute. Enfin, l'objectif de test est formé de la

trace du contre-exemple et de la séquence supplémentaire pour observer la faute.

Dans [Aichernig et al.(2008)Aichernig, Weiglhofer, and Wotawa], les auteurs utilisent des règles de mutation sur le modèle pour produire des mutants. Ces mutations permettent de modifier une des transitions du système qui est alors marquée comme telle dans le modèle original.
Ensuite, les auteurs font appel à un model-checker pour générer une trace permettant d'atteindre la transition mutante. Un graphe de test complet est créé pour le modèle original et pour le mutant, puis ces automates sont ensuite comparés selon la relation de conformité ioco (Input/Output Conformance [Tretmans(1996)]). Enfin, si un contre-exemple est trouvé, celui-ci est considéré comme un test permettant de détecter la mutation..

2.4 Outils open source pour l'automatisation des tests web

Il existe plusieurs outils d'automatisation notamment pour les différentes applications que ce soit web, mobile ou application de bureau. Ces outils peuvent être d'accès libre (open sources) ou propriétaire avec une licence. On va s'intéresser aux outils open sources, les définirent et ainsi, avec un tableau comparatif on va monter les caractéristiques de chaque outil proposé. Cependant, on a décidé d'introduire un outil propriétaire dans le tableau pour voir en général la complétude d'un outil propriétaire par rapport aux autres outils de tests open source.

2.4.1 Selenium

Selenium est un ensemble d'outils logiciels différents ayant chacun une Selenium est un ensemble d'outils logiciels différents ayant chacun une approche différente pour la prise en charge de l'automatisation des tests. La plupart des ingénieurs AQ de Selenium

se concentrent sur les outils qui répondent le mieux aux besoins de leur projet. Cependant, l'apprentissage de tous ces outils nous donnera de nombreuses options différentes pour aborder différents problèmes d'automatisation des tests. La suite complète d'outils se traduit par un ensemble complet de fonctions de test spécifiquement adaptées aux besoins de test des applications Web de tous types.
Ces opérations sont très flexibles, ce qui nous donne de nombreuses options pour localiser les éléments de l'interface utilisateur et comparer les résultats de test attendus avec le comportement réel de l'application. L'une des principales caractéristiques de Selenium est la prise en charge de l'exécution de ses tests sur plusieurs plates-formes et navigateurs [Dave Hunt(2018)].
Selenium est composé comme suit :

- **Selenium IDE :** c'est une extension de Firefox et récemment de chrome, qui permet d'enregistrer une suite d'actions, qu'il sera possible de les ré-exécuter d'un simple clic (play).
- **Selenium Web Driver :** il s'agit cette fois d'une API, disponible pour plusieurs langages notamment dans java, python et C# permettant ainsi, de programmer des actions et éventuellement des tests sur l'interface, et de vérifier les réponses. Les actions à réaliser peuvent être exportées depuis Selenium IDE (click, double click etc).

2.4.2 Katalon Studio

Katalon Studio est une solution d'automatisation simple et puissante conçue par KMS Technology pour les testeurs. C'est un puissant ensemble d'outils d'automatisation pour les tests d'applications Web et mobiles. Katalon Studio révolutionne la façon dont les testeurs de logiciels utilisent Selenium et Appium avec un cadre d'automatisation de test complet qui permet aux testeurs de configurer, créer, exécuter, rapporter et maintenir rapidement leurs tests automatisés. Katalon Studio est disponible pour les testeurs gratuitement [team QATestingTools(2017a)].

2.4.3 Sahi

Sahi est un outil gratuit et open source pour l'automatisation des tests d'applications web. Sahi est très facile à tester et permet l'automatisation facile d'applications web 2.0 complexes avec beaucoup de contenu AJAX. Doté d'un excellent enregistreur, d'une identification intelligente des objets, de scripts simples, d'attentes automatiques et de rapports intégrés, Sahi offre au testeur un outil puissant et simple pour effectuer des tests sur diverses combinaisons de navigateurs et de systèmes d'exploitation.
Sahi travaille sur Internet Explorer, Firefox, Chrome, Safari, Opera, etc. sur Windows, Mac et Linux. (Sahi fonctionne sur n'importe quel navigateur qui supporte un proxy et exécute Javascript, ce qui signifie qu'il prend en charge tous les navigateurs depuis IE6) [team QATestingTools(2017b)].

2.4.4 Watir

Watir est une famille de bibliothèques Ruby open-source (BSD) pour l'automatisation du test sur les navigateurs Web. Il permet de d'écrire des tests faciles à lire et à maintenir. Son fonctionnement est simple : Il clique, relie, remplit les formulaires, appuie sur les boutons. Watir vérifie également les résultats, par exemple si le texte attendu apparaît sur la page. Bien que watir est une bibliothèque Ruby, mais il prend en charge l'application à tester quelle que soit la technologie utilisée.
Watir travaille sur Internet Explorer sous Windows, Watir-WebDrivers prend en charge Chrome, Firefox, Internet Explorer, Opera et fonctionne également en mode navigateur sans tête (HTMLUnit) [team QATestingTools(2017c)].

2.5 Comparaison entre outils de test open source et payant

Dans le tableau 2.1 suivant nous allons présenter les outils d'automatisation de tests les plus connus et les plus utilisés

TABLE 2.1 – Comparaison entre les différents outils d'automatisation

Caractéristique	Selenium	KatalonStudio	Sahi	Watir	TestComplete
Disponible depuis	2004	2015	2005	2011	1999
Open source	Oui	Oui	Oui	Oui	Non, Licence fixe,Licence flottante
Méthode de tests	Tests en boite noire, Test fonctionnels	Tests automatisés, Test fonctionnels, Tests de régression, Tests traditionnels	Tests agile, Tests automatisés, Test fonctionnels	Tests automatisés, Testfonctionnels, Tests unitaires	Tests d'acceptation, Test automatique, Données de couverture de code, Tests pilotés, Tests distribués, Test fonctionnels, Test manuel, Les tests de régression
Méthodologie de test (approches de test)	Script-based	Tests pilotés par mots-clés (Keyword-driven)	Tests pilotés par les données (Data Driven testing)	/	Tests pilotés par des objets (Object-Driven Testing), Tests pilotés par les données (Data Driven testing)
Contraintes d'utilisation	Application web	Web (UI et API), application mobile	Application web	Application web	Application web (UI et API), mobile, application de bureau
System d'exploitation	Windows, Mac et Linux	Windows, Mac et Linux	Windows, Mac et Linux	Windows, Mac et Linux	Windows ServerWindows
Plateforme d'exécution	Firefox, Chrome	Firefox, Chrome, safari , IE, Edge, navigateur sans tête	Explorer, Firefox, Chrome, Safari, Opera	Chrome, FireFox, IE,Opera	Explorer, Firefox, Chrome, Safari, Opera
Langage de script	C // JavaSelenium API	KDTLocal Scripting	JavaScript	Ruby	C# Script, C++ Script, Delphi Script, Jscript, Local Scripting, VBScript, Visual Design (Scriptless)
Résultat des tests	Fichier XML	Journaux externes HTML, Rapport HTML, Rapports internes, Affichage résumé des résultats	Rapport HTML	Archives de la base de données, Tableau de bord graphique	Tableau de bord graphique, Partagerles rapports Résumé Affichage des résultats
Technologies supportées	Java Mobile, Applications Android, Apps iPhone, Apps Web, Apps Perl, App PHP, RubyOnRails	Web Windows GUI/Forms	AJAX, Web, Web2.0	RubyOnRailsWeb	C/C++, Dot.NET, AppJava AppMicrosoft, AppWeb, ASP, HTML, Windows,GUI/Forms
En développement	En évolution	Continuellement mise à jour	Continuellement mise à jour	En évolution	Continuellement mise à jour
Les perspectives de son développement	Sélénium IDE, Sélénium remote control, Sélénium Grid	/	Sahi pro	Nerodia, Watir classic, watir WebDriver, watirspec	Test Execute, LoadComplete, QAComplete, AQtime

Ce tableau est loin d'être exhaustif mais, il propose une vue globale des outils existant dans le domaine du test logiciel. La plupart de ces outils sont open source. Cependant, on a décidé de présenter un outil sous licence « Testcomplete » pour voir la complétude des outils payants. Chaque outil apporte de nouvelles fonctionnalités, ils sont continuellement mis à jour par leurs entreprises, c'est le cas pour les logiciels payants. En ce qui concerne les outils open source, c'est la communauté qui se charge de leurs évolutions.
Selenium reste l'outil open source le plus utilisé par les testeurs dans le monde, il est le premier outil libre pour l'automatisation des tests comme le montre le tableau 2.1 ci-dessus, la plupart des solutions de test (plateforme de tests, logiciels, framework) se basent sur cet outil pour faire fonctionner leurs produits.
C'est pour cela que nous avons choisi selenium plus précisément le selenium web-driver pour notre solution d'automatisation de test.

Pourquoi choisir selenium ?

On a choisi l'outil selenium parmi tant d'autres bien qu'il existe des outils récents, c'est parce qu'il est basé sur les méthodes des tests fonctionnels et de l'accessibilité en boîte noire, ce qui est nécessaire pour notre travail.
Contrairement aux autres outils qui ne nous permettent pas de faire ça, une autre caractéristique importante de selenium est l'approche de test script-based, ce qui nous permet de traduire nos éventuels graphes en script code selenium.

Remarque

Les outils open source ne sont pas complets. En effet, comme on a pu le voir dans le tableau comparatif tableau 2.1, l'outil TestComplet est un outil propriétaire qui est plus complet et permettant une large couverture de tests. Par contre, les outils libres restent toujours en évolution et susceptibles d'être améliorés.

2.6 Conclusion

L'automatisation des tests est essentielle à la création d'un environnement de test continu. Une équipe ne peut pas tester efficacement son système sans avoir mis en place

des tests unitaire automatisés, d'API et d'interface graphique. Chaque couche d'automatisation des tests fournit une orientation et un avantage spécifique qui permettent au système d'évoluer rapidement et de répondre aux demandes du client. De plus, l'automatisation des tests aide à créer un cycle de feedback (Actualisation des informations) rapide qui permet de tester le produit de manière continue. Ces boucles de feedback rapides permettent à l'équipe de développement de corriger rapidement et efficacement les bogues et d'implémenter de nouvelles fonctionnalités.

Chapitre 3: Analyse de besoins et conception

3.1 Introduction

Nous proposons dans ce travail un système qui est une plateforme d'automatisation des tests fonctionnels appelés aussi, tests de non-régression qui a pour but d'économiser du temps et de minimiser les coûts de développement. Pour cela, on va faire en premier lieu une architecture du système qui est illustré sur la figure 3.1, ensuite, on va aborder la partie qui traite l'analyse des besoins accompagner d'une partie de conception qui identifie les diagrammes de cas d'utilisation, le diagramme de classes ainsi que les diagrammes de séquence.

3.2 Architecture du système

La figure 3.1 ci-dessous représente l'architecture globale de notre système.

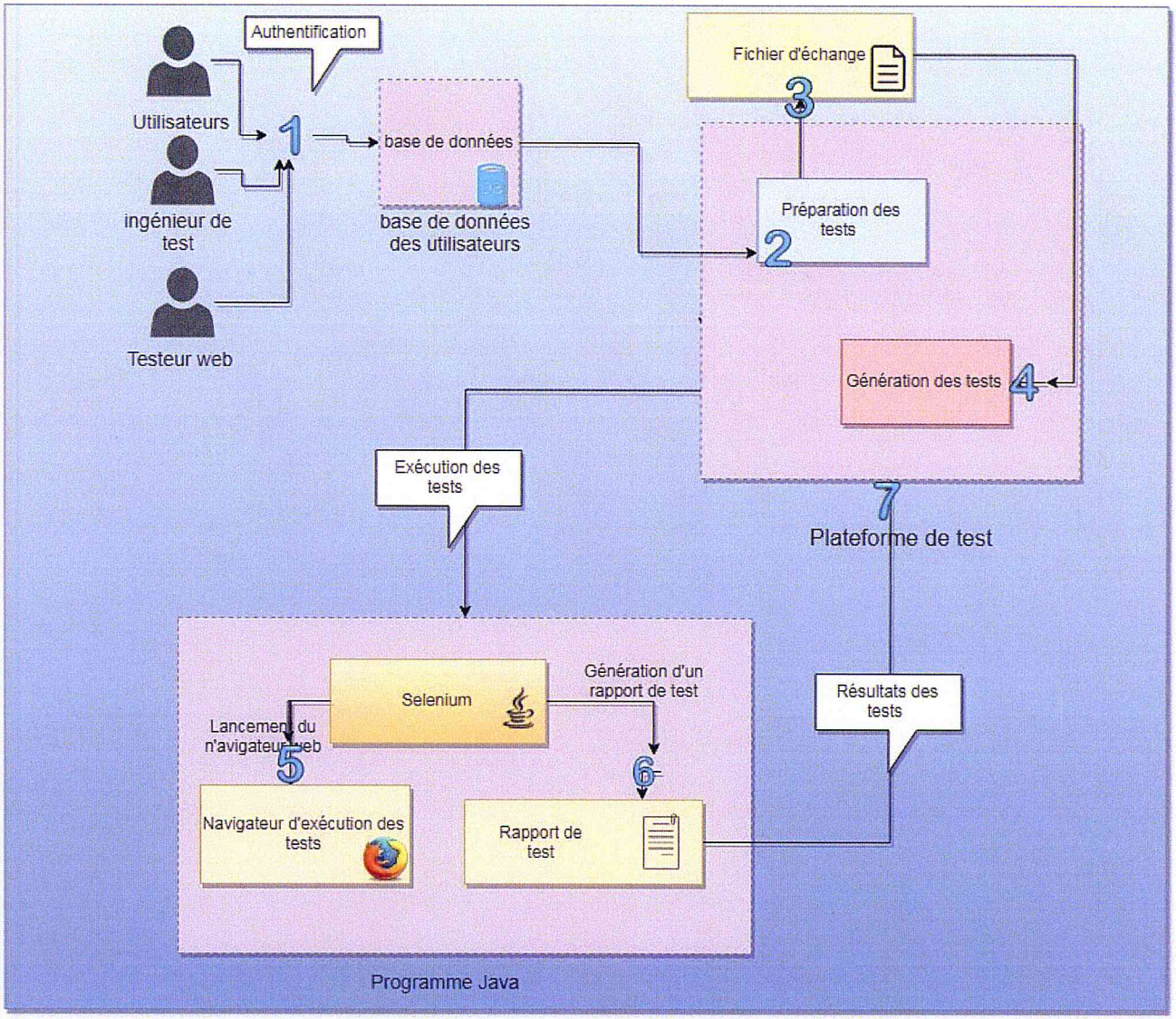


FIGURE 3.1 – Architecture globale du système

3.2.1 Description de l'architecture

- **Authentification :** Chaque utilisateur doit s'authentifier pour accéder au système.
- **Base de données :** Contient les utilisateurs du système.
- **Plateforme de test :** C'est ici que l'utilisateur prépare les tests dans la partie préparation des tests, les données de tests préparés sont stockées dans des fichiers

« JSON » qu'on appelle ici un fichier d'échange. Les cas de tests sont préparés à partir du fichier qui contient les informations nécessaires pour faire ce cas de tests. Les cas de tests sont stockés quant à eux dans un autre fichier. À partir de ce fichier on génère les tests.

- **Générateur de test :** C'est ici que vient le rôle du selenium Web-Driver, lors de l'exécution des tests à partir de la phase génération de tests, le selenium ouvre le navigateur web et commence à exécuter les cas de tests. Quand selenium termine l'exécution de tous les cas de tests, il génère un fichier XML qui contient les résultats des tests, ces résultats sont affichés dans la plateforme.

3.3 Préparation des tests

L'approche utilisée dans la préparation des tests est le test basé sur un modèle « model based testing ». Dans notre cas Figure 3.2, nous avons choisi de mettre en place un modèle de graphe « drag and drop » pour faciliter la tâche aux testeurs qui n'ont pas une grande connaissance sur la programmation et le codage des scripts de test. Ce modèle permet de préparer des tests par de simples clics sur l'écran. Ainsi, le scénario de test s'affiche sur le côté de l'écran, permettant une visualisation des parcours de test. En effet, le testeur doit tester chaque cas de tests, et un cas de tests contient plusieurs scénarios. un scénario de test permet d'utiliser un même script de test pour tester plusieurs configurations . Un scénario est représenté par deux nœuds reliés par un lien entre eux, le premier nœud appelé nœud de départ qui contient un écran de départ, et un nœud d'arrivée qui contient un écran d'arrivée.

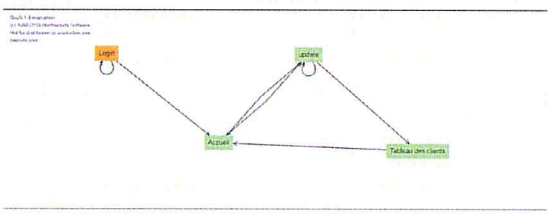


FIGURE 3.2 – Drag and drop

3.4 Génération des parcours

La génération des parcours se fait selon le pseudo-algorithme suivant :

Pseudo-algorithme
@BeforeTest
Lancer geckodriver
@Test
Pour chaque cas_de_test dans List faire :
Lire (fichier JSON)
Pour chaque scénario dans List faire :
Lire (fichier JSON)
Lire (id_écran_départ)
Pour chaque condition_d'entré faire :
Lire (type Condition d'entré)
Lire (valeur Condition_d'entré)
Exécuter commande selenium spécifique à cette condition
Fait
Pour chaque condition_sortie faire :
Lire (type condition de sortie)
Lire (valeur condition de sortie)
Exécuter commande selenium pour récupérer valeur attendu
Si (valeur condition de sortie = valeur attendu)
Test_scénario = pass
Else
Test_scénario = faill
Fait
fait
fait
fin
@AfterTest
Fermer geckodriver

3.5 Les besoins fonctionnels

les besoins fonctionnels nécessaires qui doivent être réalisés dans notre système sont les suivants :

- L'authentification, où chaque utilisateur doit être authentifié pour accéder au système.

- Chaque utilisateur a un rôle spécifique pour réaliser un ensemble précis d'activités. Dans la phase de préparation des tests, l'ingénieur a la possibilité d'effectuer les tâches suivantes :

  Préparation des composants :

  - Ajouter un écran
  - Supprimer un écran
  - Modifier un écran
  - Attribuer à chaque écran un composant ou un ensemble de composants
  - Supprimer un ou plusieurs composants
  - Modifier un ou plusieurs composants

  préparation des chemins :

  - Ajouter un scénario
  - Supprimer un scénario
  - Modifier un scénario

  préparation des cas des tests :

  - Ajouter un cas de test
  - Supprimer un cas de test
  - Modifier un cas de test

- L'ingénieur de test a la possibilité de créer un scénario de test, avec une représentation graphique « Drag & Drop »
- Lancer un test
- Annuler un test
- Créer un référentiel de test
- Modifier un référentiel de test
- Supprimer un référentiel de test
- Exporter un référentiel de test

3.6 Les besoins non fonctionnels

L'application doit avoir une IHM intuitive permettant une navigation facile et de faire un travail rapide.
L'application nécessite d'être compatible avec les navigateurs web les plus utilisés (Google chrome, Mozilla Firefox, Opera).

3.7 Processus de développement

Nous avons utilisé le processus unifié UP (unified process) qui est un processus de développement logiciel construit sur UML. Il est itératif, centré sur l'architecture, piloté par des cas d'utilisation et orienté vers la diminution des risques. Il regroupe les activités à mener pour transformer les besoins d'un utilisateur en système logiciel.
C'est un patron de processus pouvant être adapté à une large classe de systèmes logiciels, à différents domaines d'application, à différents types d'entreprises, à différents niveaux de compétences et à différentes tailles de l'entreprise [Roques(2008)].

3.8 Conception

3.8.1 Diagramme de cas d'utilisation globale

Le diagramme ci-dessous représente les cas d'utilisation globale de notre système.

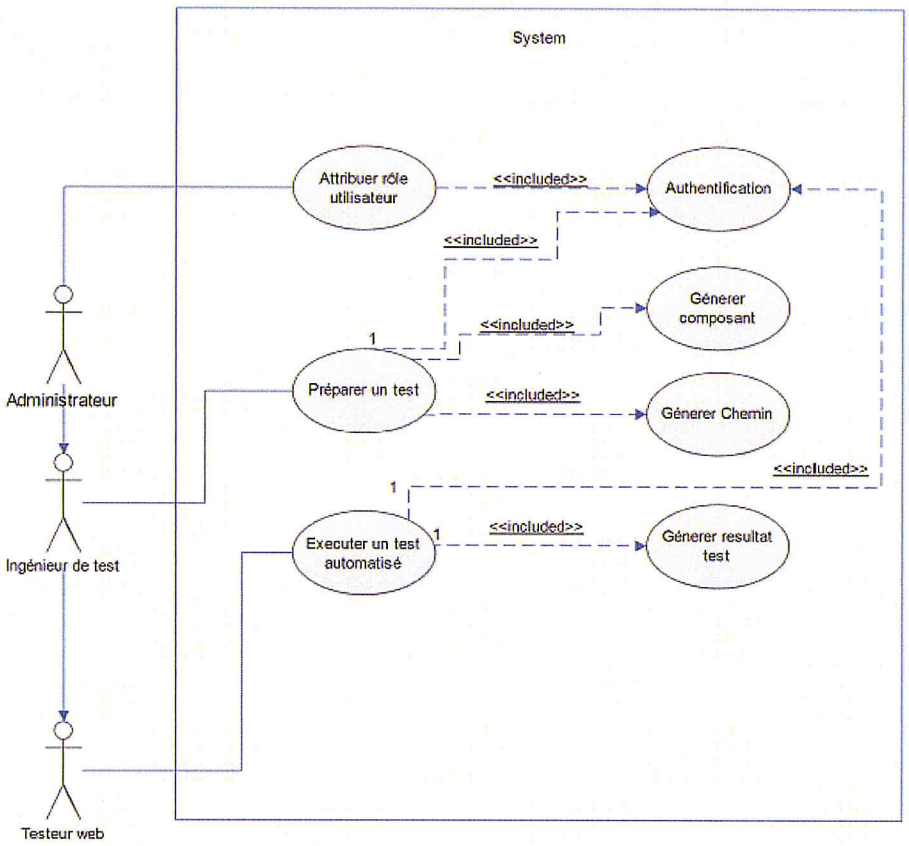


FIGURE 3.3 – Diagramme de cas d'utilisation globale du système

Description textuelle de cas d'utilisation globale du système :

Nom : Authentification

Acteur : Administrateur

Objectif : Attribuer un rôle a un utilisateur

Précondition : L'utilisateur doit s'authentifier en tant qu'administrateur

Postcondition : Rôle attribué pour chaque utilisateur

Scénario nominal :

— Le système afficher un tableau qui contient les utilisateurs

— L'administrateur sélectionne un utilisateur et lui attribue un rôle

Scénario alternatif :

— L'administrateur décide de quitter la consultation du tableau des utilisateurs.

Scénario d'exception :

— Authentification erronée

— Utilisateur non enregistré

Nom : Préparer un test

Acteur : Ingénieur de test

Objectif : Préparation des tests

Précondition : L'utilisateur doit s'authentifier en tant qu'ingénieur de test

Postcondition : Les tests sont préparés

Scénario nominal :

— Le système afficher une page pour la préparation des tests

— L'utilisateur ajoute un écran et relie chaque écran à ses composants

Scénario alternatif :

— L'utilisateur décide de quitter la page de préparation des tests

Scénario d'exception :

— Erreur d'authentification

— Écran inexistant lors de l'attribution des composants

Nom : Exécution du test automatisé

Acteur : Testeur web

Objectif : Lancer un Test

Précondition :

— L'authentification en tant que testeur web

— Cas de test préparé

Postcondition :

- L'authentification en tant que testeur web
- rapport de test généré

Scénario nominal :

- Le système lance des tests automatiques lorsque le testeur décide
- Le système génère un rapport de test
- L'utilisateur consulte le rapport de test généré

Scénario alternatif :

- L'utilisateur décide d'annuler l'exécution du test

Scénario d'exception :

- Cas de test non préparés
- Le système n'arrive pas à localiser les écrans ou les composants lors de l'exécution du test automatisé
- Échec de connexion entre la plateforme de test et l'application sur laquelle s'effectue les tests.

3.8.2 Diagramme de cas d'utilisation : Préparation des tests

Le diagramme ci-dessous représente les cas d'utilisation de préparation de tests.

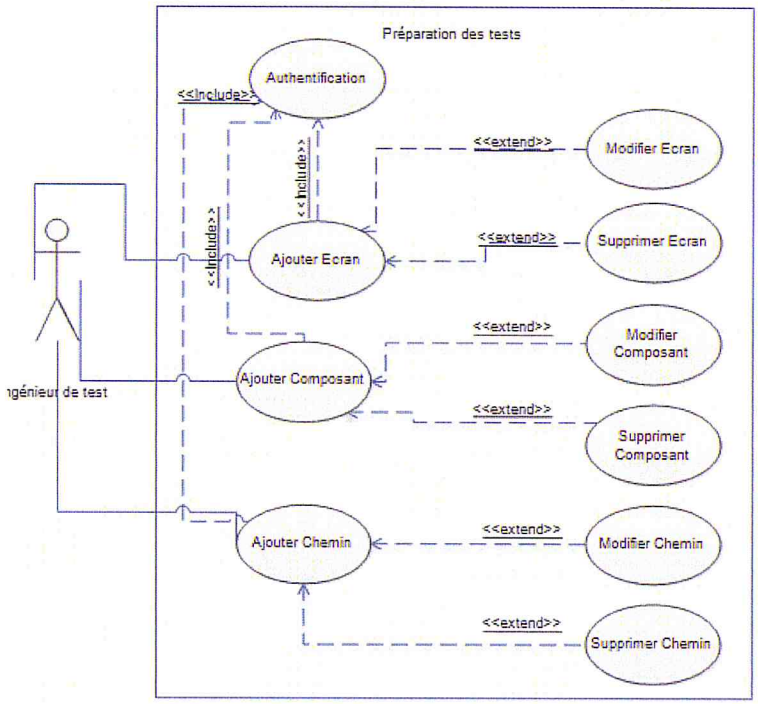


FIGURE 3.4 – Diagramme de cas d'utilisation de la préparation de tests

Description textuelle des cas d'utilisations : Préparation de tests

Nom : Préparer un test

Acteur : Ingénieur de test

Objectif : l'ajouter d'un écran

Précondition : Authentification en tant qu'ingénieur de test

Postcondition : Écran ajouté

Scénario nominal :

— Le système afficher la page de préparation de test pour l'ajout des écrans

— L'utilisateur ajoute un écran et l'écran sera enregistré

Scénario alternatif :

— L'utilisateur annule l'opération de l'ajout

— L'utilisateur supprime un écran ajouté

Scénario d'exception :

— Erreur d'authentification

— Si deux écrans du même nom existent une exception est déclenchée

Nom : Ajouter un composant

Acteur : Ingénieur de test

Objectif : L'ajout d'un composant

Précondition : Authentification en tant qu'ingénieur de test

Postcondition : Composant ajouté

Scénario nominal :

— Le système afficher la page de préparation de test pour l'ajout des composants

— L'utilisateur ajoute un composant

Scénario alternatif :

— L'utilisateur annule l'opération de l'ajout du composant

— L'utilisateur supprime un composant

Scénario d'exception :

— Erreur d'authentification

Nom : Ajouter un scénario

Acteur : Ingénieur de test

Objectif : L'ajouter d'un scénario

Précondition : Authentification en tant qu'ingénieur de test

Postcondition : Scénario ajouté

Scénario nominal :

— Le système afficher la page chemins pour la création d'un scénario

— L'utilisateur ajoute un scénario

Scénario alternatif :

— L'utilisateur annule l'opération de l'ajout d'un scénario

— L'utilisateur supprime un scénario

Scénario d'exception :

— Erreur d'authentification

3.8.3 Diagramme de cas d'utilisation : Exécution des tests

Le diagramme ci-dessous représente les cas d'utilisation d'exécution des tests.

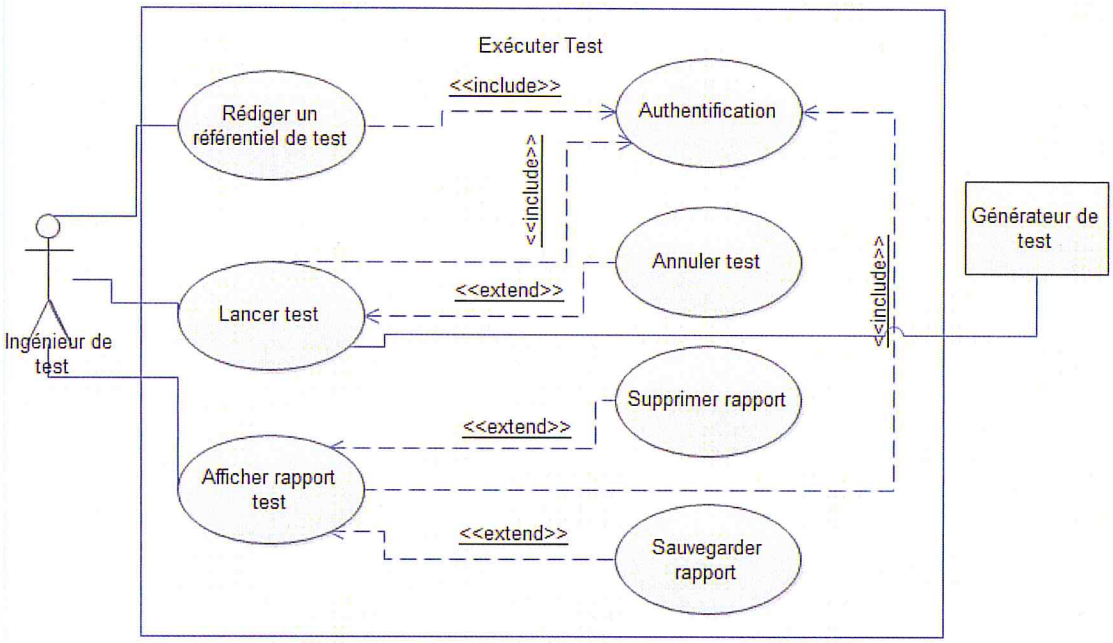


FIGURE 3.5 – Diagramme de cas d'utilisation de l'exécution des tests

Description textuelle de cas d'utilisation : Exécution de tests

Nom : Exécuter un test

Acteur : Testeur web

Objectif : Lancer l'exécution automatique d'un test

Précondition :

— L'authentification en tant que testeur web

— Cas de test préparé

Postcondition : Génération d'un rapport de test

Scénario nominal :

— Le système lance des tests automatiques lorsque le testeur décide

— Le système génère un rapport de test

— L'utilisateur consulte le rapport de test généré

Scénario alternatif :

— L'utilisateur décide l'annulation de l'exécution du test

Scénario d'exception :

— Cas de test non préparés

— Le système n'arrive pas à localiser les écrans ou les composants lors de l'exécution du test automatisé

— Coupure de connexion entre la plateforme de test et l'application sur la quel s'effectue les tests.

3.8.4 Diagramme de cas d'utilisation : Gestion des utilisateurs

Le diagramme ci-dessous représente les cas d'utilisation de la gestion des utilisateurs.

Description textuelle de cas d'utilisation : Gestion des utilisateurs

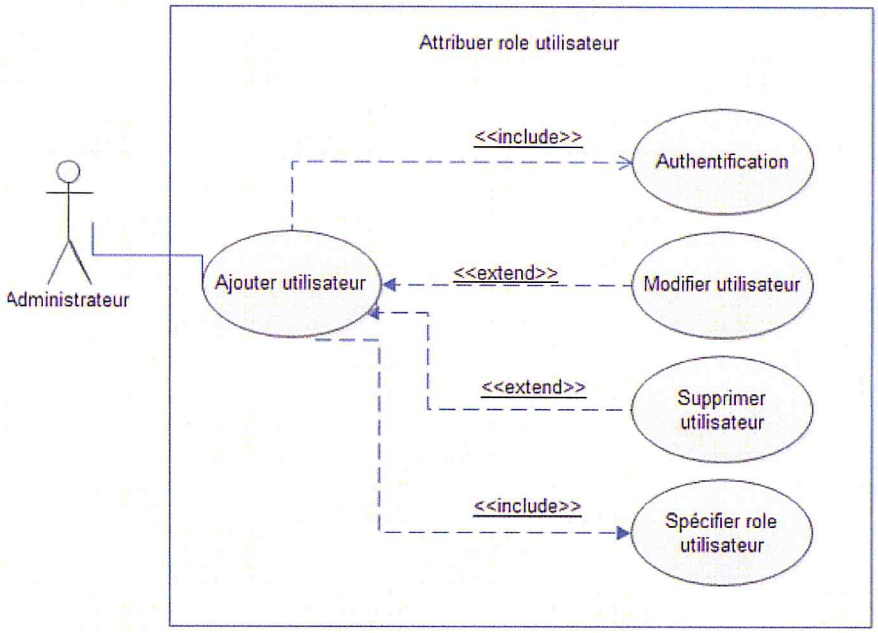


FIGURE 3.6 – Diagramme de cas d'utilisation pour la gestion des utilisateurs

Nom : Gérer des utilisateurs

Acteur : Administrateur.

Objectif : L'ajout d'un nouveau utilisateur et lui attribuer un rôle.

Précondition : Authentification au système en tant qu'administrateur

Postcondition : Utilisateur ajouté avec son rôle

Scénario nominal :

- L'administrateur accède à la page de gestion des utilisateurs
- Une interface apparaît contenant une liste des utilisateurs
- L'administrateur clique sur ajouter un utilisateur, une fenêtre apparait pour remplir les informations liées à ce dernier
- Si un administrateur veut modifier les informations d'un utilisateur, il clique sur

le bouton modifier utilisateur.

— Si l'administrateur désire supprimer un utilisateur de la base de données, il clique sur supprimer utilisateur.

— Un bouton « spécifier un rôle » a pour but d'attribuer un rôle à l'utilisateur

Scénario alternatif :

— L'ajout d'un même utilisateur peut déclencher une exception.

Scénario d'exception :

— L'ajout d'un même utilisateur peut déclencher une exception.

3.8.5 Diagrammes de séquence

Les diagrammes de séquence présentent la vue dynamique du système. Leur objectif est de représenter les interactions entre les objets en indiquant la chronologie des échanges. Les diagrammes de séquences qui vont suivre représentent les cas d'utilisation cités précédemment, pour les autres diagrammes de séquences détaillés [voir annexe B].

3.8.6 Diagramme de séquence : Gérer un scénario

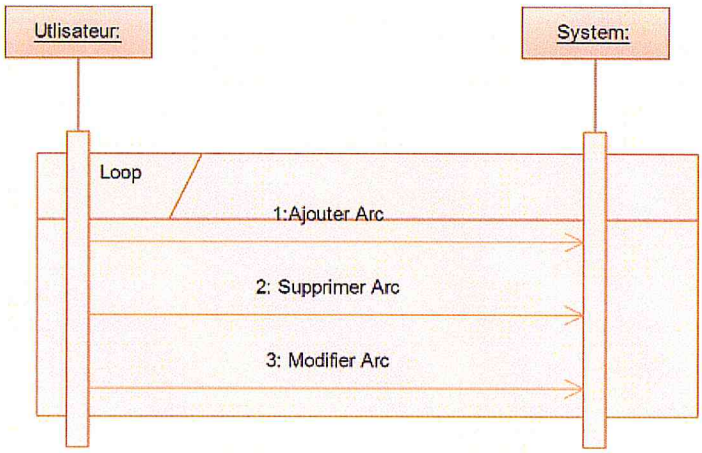


FIGURE 3.7 – Diagramme de séquence représentant les opérations effectuées sur un scénario

Ce diagramme représente l'interaction entre le « user » et notre « système » afin d'ajouter, modifier ou supprimer un scénario.

3.8.7 Diagramme de séquence : Gérer un écran

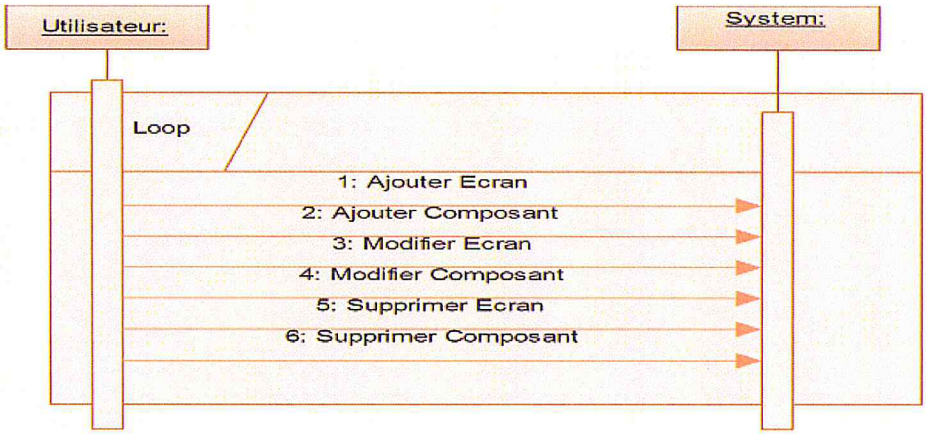


FIGURE 3.8 – Diagramme de séquence représentant les opérations effectuées sur un écran

Ce diagramme représente l'interaction entre le « user » et notre « système » afin d'ajouter, modifier ou supprimer un écran. Éventuellement, ajouter, modifier ou supprimer un composant.

3.8.8 Diagramme de séquence : Gérer un composant

Ce diagramme représente l'interaction entre le « user » et notre « système » afin d'ajouter, modifier ou supprimer un composant.

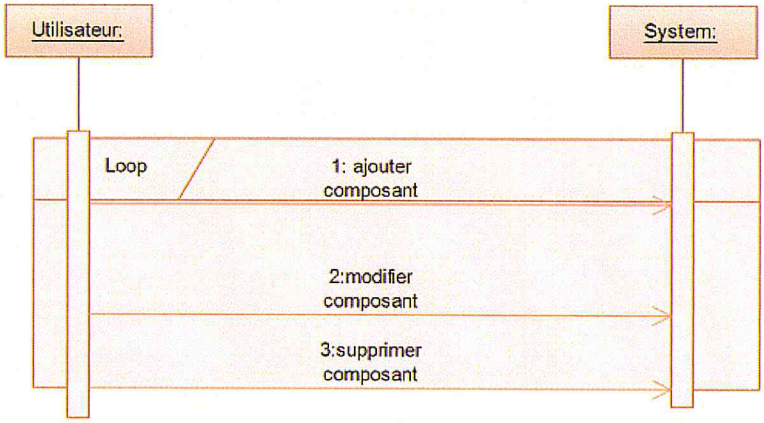


FIGURE 3.9 – Diagramme de séquence représentant les opérations appliquées sur un composant

3.8.9 Diagramme de séquence globale du système

Ce diagramme représente l'interaction entre le « user » et notre « système » pour l'utilisation globale du système.

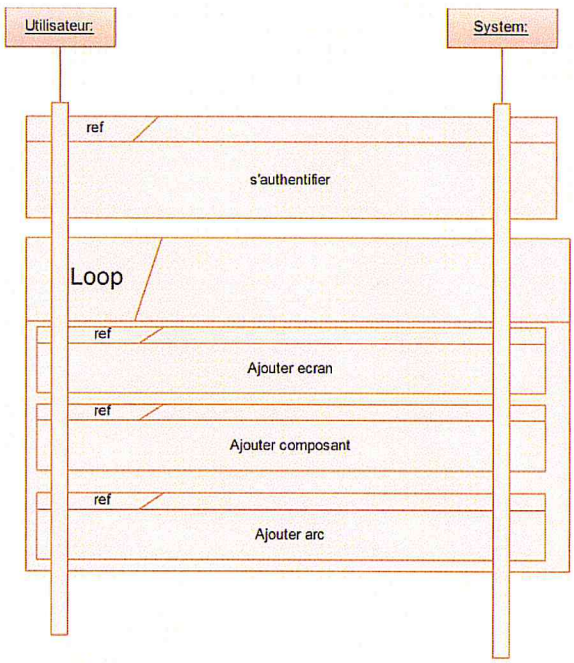


FIGURE 3.10 – Diagramme de séquence globale du système

3.8.10 Diagramme de classe

Ce diagramme exprime l'aspect structurel des données à l'aide de classes et d'associations entre ces classes.
Une classe UML décrit de manière abstraite un ensemble d'objets du système qui possèdent une sémantique commune. Cette sémantique est définie alors à partir des attributs et des opérations de la classe.

— **Les attributs** permettent de décrire les données de la classe. Un attribut possède un type, une visibilité et peut éventuellement être caractérisé par une valeur par défaut.

— **Les opérations** décrivent l'aspect comportemental de la classe. Il s'agit de fonctions qui peuvent prendre des valeurs d'entrées (paramètres d'entrées), modifier les attributs et / ou produire des résultats.

3.8.11 Diagramme de classe du système

Le diagramme ci-dessous permet de voir en détail les différentes classes utilisées dans notre système et les liens entre eux.

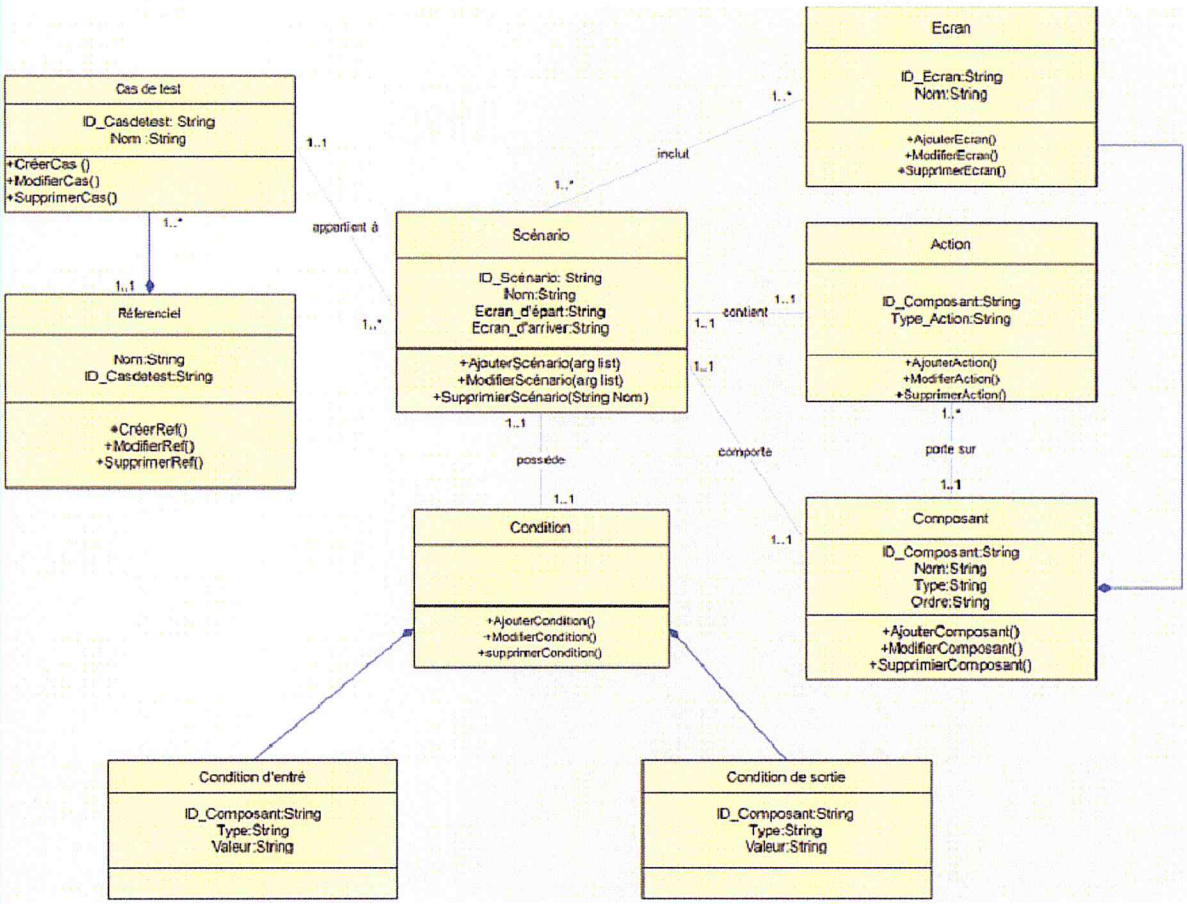


FIGURE 3.11 – Diagramme de classe globale du système

3.9 Conclusion

Après avoir définit les rôles de chaque utilisateur ainsi que le fonctionnement détaillé de notre système, on va vous présenter notre plateforme avec quelques captures d'écran et son environnement de développement.

Chapitre 4: Implémentation et résultats

4.1 Introduction

Pour mener une démarche scientifique rigoureuse, il est nécessaire d'effectuer un ensemble de tests sur l'approche suggérée, afin d'obtenir des résultats et voir s'ils conviennent aux résultats attendus.
Dans les deux chapitres précédents, nous avons présenté l'approche d'automatisation des tests fonctionnels basée sur un modèle ainsi qu'une conception du système.
Dans ce chapitre, nous allons d'abord parler sur les différents outils utilisés pour élaboré notre travail. Ensuite, nous nous sommes focalisés sur l'implémentation de notre système en justifiant nos choix techniques. Enfin, nous allons présenter quelques tests effectués par notre système.

4.2 Présentation des outils de travail

Toutes les expérimentations ont été exécutées et réalisées à l'aide d'un ordinateur muni d'un processeur de type Intel (R) Core (TM) i5-5200U CPU @ 2.20GHz 2.20 GHz avec une mémoire RAM de taille 8 Go.

4.2.1 Plateformes de développement

- **Angular** Angular est un Framework JavaScript développé par Google libre et open source qui permet d'améliorer la syntaxe de JavaScript et d'augmenter la productivité de développement. Angular s'appuie sur la logique Modèle-Vue-Contrôleurs (MVC). Créé en 2009, il est devenu incontournable pour le développement web. La version d'Angular utilisée dans ce projet est Angular 5.0.

  Les avantages du framework Angular 5.0 :

  — **La cohérence :** Le Framework est basé sur des composants et des services.

  — **Le Typescript :** Il utilise le Typescript comme un langage de programmation qui est plus élaboré que le javascript et plus facile à utiliser. Il permet également, une maintenance plus facile et une détection de bogues plus rapide.

  — **Une seule page :** Possibilité de développer des applications (single page) ou (one page application), c'est-à-dire donner à l'utilisateur l'impression d'utiliser tout le site comme étant une seule page.

  — **Cross plateformes** Permets aux développeurs d'apprendre une seule façon de créer des applications avec Angular. Ensuite, le code est réutilisé. Il a la capacité de créer des applications pour n'importe quelle cible de déploiement. Pour le web, le web mobile, le mobile natif et le bureau natif.

  — **Développé par Google :** Du prototype au déploiement global, Angular fournit la productivité et l'infrastructure évolutive qui prend en charge les plus grandes applications de Google.

- **Java :** Java est un langage de programmation orienté objet, créé par James Gosling et Patrick Naughton. Nous avons utilisé le langage java pour différentes raisons, notamment :

  — La popularité de java

  — Il est utilisé par de nombreux programmeurs et développeurs de grandes sociétés, ce qui permet de trouver facilement de l'aide.

  — La disponibilité de la documentation (javadoc).

— Nous avons déjà travaillé avec java au cours de notre cursus universitaire, ce qui nous facilite grandement la tâche.

- **TestNG :** C'est est un framework pour effectuer des tests pour le langage de programmation Java, inspiré par JUnit et NUnit. L'objectif de TestNG étant de couvrir un large spectre de catégories de tests unitaires [voir annexe A].
- **JSON :** Le format de données JSON (JavaScript Object Notation) constitue le standard actuel pour les échanges de données sur le Web. Il s'agit d'une syntaxe pour décrire des informations structurées sous une forme proche des objets JavaScript.

4.2.2 Gestion du projet avec l'outil Trello

Trello est un outil de gestion de projet en ligne, il est basé sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. La version de base de trello est gratuite permettant de faire une gestion de projets magnifiques. Tandis que la version payante permet d'obtenir des services supplémentaires.
Dans notre cas, on a utilisé la version gratuite qui nous a suffit largement et nous a bien aidé à accomplir les tâches demandées par notre promoteur. Notre travail est organisé en 'sprint' où chaque sprint consiste à terminer une partie bien précise du projet.
Voici un exemple, sur comment on a organisé notre travail :

Voici un exemple d'un sprint sur Trello :

FIGURE 4.1 – Organisation du travail sur Trello

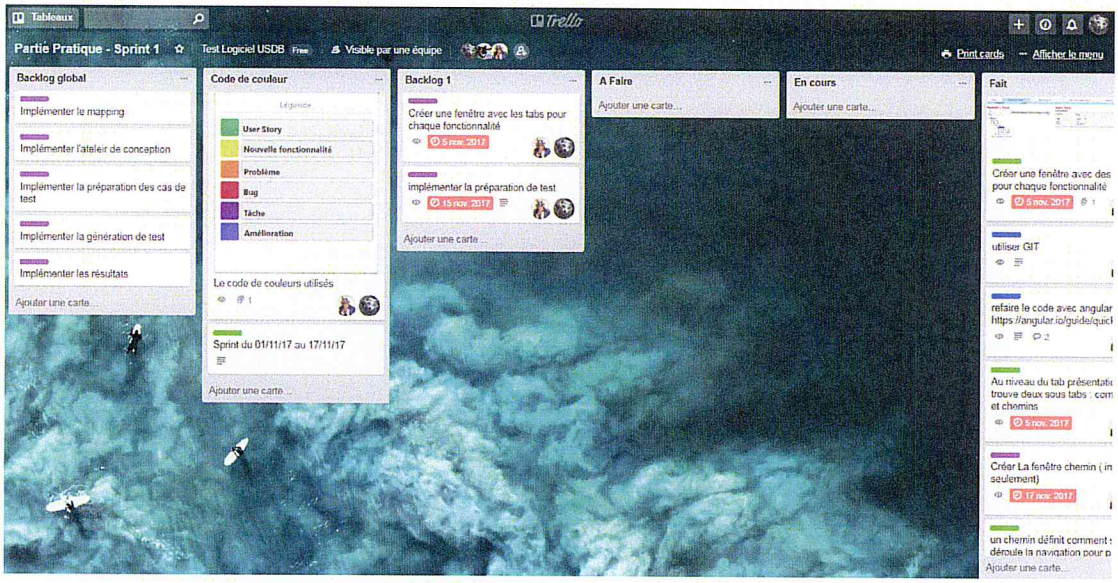


FIGURE 4.2 – Exemple d'un sprint sur Trello

4.2.3 Benchmark de test

Le test de benchmark est un type de test effectué pour fournir un ensemble reproductible de résultats quantifiables à partir desquels les fonctionnalités spécifiques des versions logicielles actuelles et futures peuvent être référencées et comparées.

Dans cette étape de test, nous allons utiliser le benchmark guru99 Bank axé sur sélénium

[guru99(2018a)] vu qu'il offre de multiples cas de test avec la possibilité de réaliser les tests de non-régressions, par exemple, Guru bank v1 guru bank v2 jusqu'à v4.
Ces versions permettent de tester la non-régressions dans le fait où, si les tests effectués sur la version v1 ont été corrects (erreurs non trouvées) ou éventuellement les erreurs trouvées ont été corrigées, alors les tests sur les versions (v2, v3 etc.) doivent être sans erreur.

4.2.4 Principales fonctionnalités de la plate-forme de test

Le plan de test est conçu pour prescrire la portée, l'approche, les ressources et le calendrier de toutes les activités de test du projet Guru99 Bank.
Le plan identifie les éléments à tester, les caractéristiques à tester, les types de tests à effectuer, le personnel responsable des tests, les ressources et le calendrier requis pour effectuer les tests, ainsi que les risques associés au plan.
Voici quelques cas de tests parmi les cas de test qui existent sur guru99 Bank 4.1, toutes les fonctionnalités du site Web Guru99 Bank qui ont été définies dans les spécifications des exigences du logiciel ont besoins d'être tester.

TABLE 4.1: Description d'un plan de test du benchmark GuruBank

Nom du module	Rôles	Description
Demande de solde de compte	Manager Client	Client : un client peut avoir plusieurs comptes bancaires. Il peut voir le solde de ses comptes seulement. Manager : un manager peut voir le solde de tous les clients qui relèvent de sa supervision.
Transfert de fonds	Manager Client	Client : Un client peut transférer des fonds de son propre compte vers n'importe quel compte de destination. Manager : Un manager peut transférer des fonds de n'importe quel compte bancaire source vers le compte de destination.
Relevé de compte	Manager Client	Un relevé de compte affichera les 5 dernières transactions d'un compte. Client : Un client peut voir le relevé de compte de ses propres comptes seulement. Un manager peut voir le relevé de compte de n'importe quel compte.

Relevé personnalisé	Manager Client	Un relevé personnalisé vous permet de filtrer et d'afficher les transactions dans un compte en fonction de la date et de la valeur de la transaction. Client : un client peut voir une déclaration personnalisée de ses seuls comptes "propres" Manager : un manager peut voir une déclaration personnalisé de n'importe quel compte.
Changer un mot de passe	Manager Client	Client : Un client peut changer le mot de passe de son compte uniquement. Manager : Un manager peut changer le mot de passe de son compte uniquement. Il ne peut pas changer les mots de passe de ses clients.
Ajouter un nouveau client	Manager	Manager : un manager peut ajouter un nouveau client.
	Manager	Manager : un manager peut modifier des détails comme l'adresse, l'adresse e-mail, le numéro de téléphone d'un client

Ajouter un nouveau compte	Manager	Actuellement, le système fournit 2 types de comptes . Enregistrement . Actuel Un client peut avoir plusieurs comptes d'épargne (un à son nom, un autre à un nom commun, etc.). il peut avoir plusieurs comptes courants pour différentes entreprises qu'il possède. Ou il peut avoir plusieurs comptes courants et d'épargne. Manager : un manager peut ajouter un nouveau compte pour un client existant.
Modifier un compte	Manager	Manager : un manager peut ajouter des informations de compte d'édition pour un compte existant.
Supprimer un compte	Manager	Manager : un manager peut ajouter un compte supprimé pour un client.
Supprimer un client	Manager	Un client ne peut être supprimé que s'il n'a pas de compte courant ou de compte d'épargne. Manager : un manager peut supprimer un client.

Dépôt	Manager	Manager : un manager peut déposer de l'argent dans n'importe quel compte. Habituellement effectué lorsque l'argent est déposé dans une succursale bancaire.
Retrait	Manager	Manager : un manager peut retirer de l'argent de n'importe quel compte. Habituellement fait lorsque l'argent est retiré à une succursale bancaire.

4.2.5 Implémentations

Dans cette partie, nous allons parler des fonctionnalités de notre application.
La figure 4.3 représente la phase de préparation des tests, c'est la phase initiale où l'utilisateur enregistre les différentes informations d'écrans et de leurs composants.
L'utilisateur clique sur "Préparation des tests" ensuite, il clique sur le bouton "Composants", une fenêtre s'affiche et lui donnera la main pour ajouter un écran et ses composants.

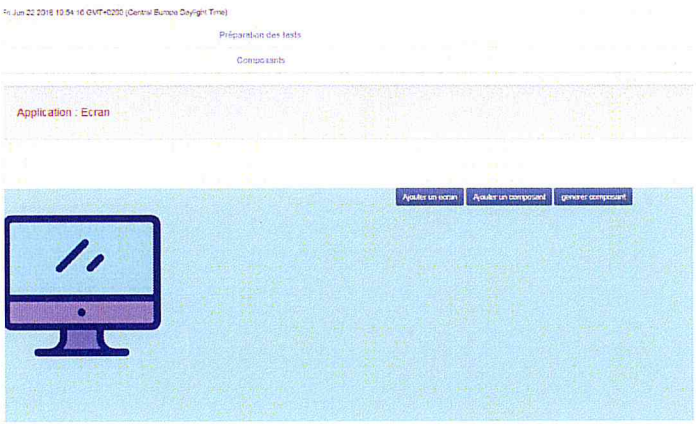


FIGURE 4.3 – Préparation des tests

L'utilisateur clique sur le bouton "ajouter écran" qui lui permettra d'insérer toutes les informations souhaitées, voir figure 4.4.

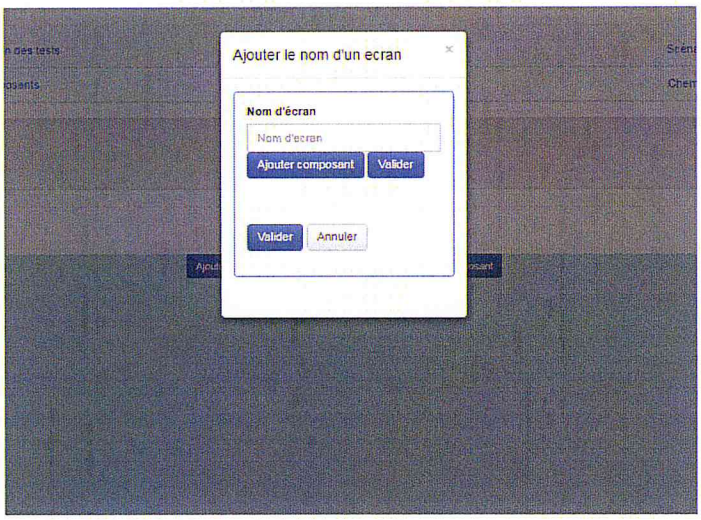


FIGURE 4.4 – Fenêtre d'ajout des écrans

.

Ensuite, en cliquant sur le bouton "ajouter composant", il attribut aux écrans leurs composants et leurs informations, voir Figure. 4.5

FIGURE 4.5 – Formulaire qui associe un composant à son écran

.

Après la création des écrans et leurs composants, la phase qui suit, c'est la création des chemins pour créer un scénario de test.
Cette phase est importante pour construire le scénario d'un cas de test.

La figure 4.6 représente un formulaire qui doit être remplis pour créer un scénario de test.

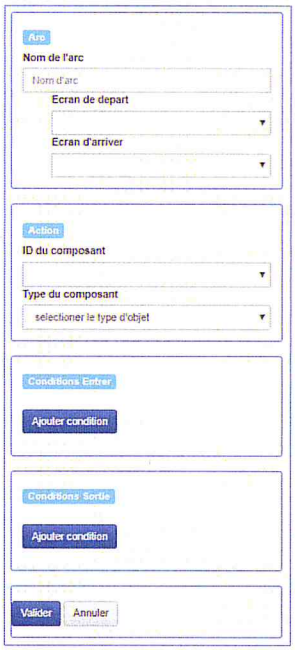


FIGURE 4.6 – Formulaire pour ajouter un scénario

4.3 Conclusion

Dans ce chapitre nous avons présenté l'environnement matériel et logiciel sur lesquels nous avons travaillé, ainsi que les captures d'écran de notre plateforme.

Conclusion générale

Avant de commencer ce projet, nous avons toujours considéré que l'étape des tests peut être évitée et qu'on pouvait se passer d'elle, mais, après avoir réalisé ce travail, nous nous sommes rendus à l'évidence que le test est une étape très importante, à lui seul il constituait toute une problématique.

Le test relève du domaine du génie logiciel. En effet, il permet de faire un bon logiciel et une maintenance de son après-vente qui ne coûte pas chère.

Le but fixé tout au début de notre travail était la conception et la réalisation d'une plateforme d'automatisation des tests fonctionnels.

Après avoir fait une étude théorique sur les tests logiciels, une étude détaillée sur les outils d'automatisation des tests et une analyse des besoins, nous avons entamé la phase de conception de notre projet, en utilisant le langage de modélisation UML avec le processus de développement UP.

A la fin, nous sommes arrivés à un résultat acceptable. Cependant, on a dû remplacer l'idée de faire un drag and drop avec des formulaires pour remplir les scénarios et les cas de tests.

Les perspectives susceptibles d'être ajoutées :

- Inclure la fonctionnalité de drag and drop dans l'application.
- Inclure un pilote statistique « dashboard » qui rend l'utilisation de la plateforme plus simple.
- Créer des sessions pour différents utilisateurs avec les logs.

Bibliographie

[Abran(2010)] Alain Abran. *Software Metrics and Software Metrology.* Wiley-IEEE Computer Society Pr, 2010. ISBN 0470597208, 9780470597200.

[Aichernig and Delgado(2006)] Bernhard K. Aichernig and Carlo Corrales Delgado. From faults via test purposes to test cases : On the fault-based testing of concurrent systems. In *Fundamental Approaches to Software Engineering*, pages 324–338. Springer Berlin Heidelberg, 2006. doi : 10.1007/11693017_24. URL `https://doi.org/10.1007%2F11693017_24`.

[Aichernig et al.(2008)Aichernig, Weiglhofer, and Wotawa] Bernhard K. Aichernig, Martin Weiglhofer, and Franz Wotawa. Improving fault-based conformance testing. *Electronic Notes in Theoretical Computer Science*, 220 (1) :63–77, dec 2008. doi : 10.1016/j.entcs.2008.11.006. URL `https://doi.org/10.1016%2Fj.entcs.2008.11.006`.

[Ashish Shah(2017)] Devesh Hingorani Zohair Hasan Ashish Shah, Ashwin Megha. Test automation across the phases of test cycle, 2017. URL `https://www.forgeahead.io/blogs/test-automation-across-the-phases-of-test-cycle-2`.

[Burnstein(2003)] Ilene Burnstein. *Practical Software Testing.* Springer-Verlag, 1st edition, 2003. ISBN 0-387-95131-8, N 0-387-95131-8.

[Castillos(2013)] Kalou Cabrera Castillos. Generation automatique de scenarios de tests à partir de proprietes temporelles et de modèles comportementaux, 11 2013.

[Cohen et al.(1997)Cohen, Dalal, Fredman, and Patton] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton. The AETG system : an approach to testing based on

combinatorial design. *IEEE Transactions on Software Engineering*, 23(7) :437–444, jul 1997. doi : 10.1109/32.605761. URL https://doi.org/10.1109%2F32.605761.

[Dave Hunt(2018)] Mary Ann May-Pumphrey Noah Sussman Paul Grandjean Peter Newhook Santiago Suarez-Ordonez Simon Stewart Tarun Kumar Dave Hunt, Luke Inman-Semerau. Test automation for web applications, 2018. URL https://www.seleniumhq.org/docs/01_introducing_selenium.jsp#test-automation-for-web-applications.

[Ehmer and Khan(2012)] Mohd Ehmer and Farmeena Khan. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Computer Science and Applications*, 3(6), 2012. URL https://doi.org/10.14569%2Fijacsa.2012.030603.

[Everett and McLeod(2007)] G.D. Everett and R. McLeod. *Software Testing : Testing Across the Entire Software Development Life Cycle.* Wiley, 2007. ISBN 9780470146347. URL https://books.google.dz/books?id=dmG_Dc7QFWUC.

[Farrell-Vinay(2007)] Peter Farrell-Vinay. *Manage Software Testing.* Auerbach Publications, Boston, MA, USA, 2007. ISBN 0849393833, 9780849393839.

[fortune.com(2015)] fortune.com. starbucks systems outage. april 2015. URL http://fortune.com/2015/04/24/starbucks-systems-outage.

[G. J. Myers(1976)] Wiley G. J. Myers. *Software reliability : Principles and practices.* John Wiley & Sons, Inc., New York, NY, USA, 1976.

[Graham et al.(2006)Graham, Black, Van Veenendaal, and Evans] D. Graham, R. Black, E. Van Veenendaal, and I. Evans. *Foundations of Software Testing : ISTQB Certification.* Thomson, 2006. ISBN 9781844803552. URL https://books.google.dz/books?id=_4tUPgAACAAJ.

[Guru99(2018)] Guru99. How to create requirements traceability matrix (rtm), 2018. URL https://www.guru99.com/traceability-matrix.html.

[guru99(2018a)] guru99. benchmark gurubank, 2018a. URL https://www.guru99.com.

[guru99(2018b)] guru99. risk based testing, 2018b. URL https://www.guru99.com/risk-based-testing.html.

[Homès(2013)] B. Homès. *Fundamentals of Software Testing.* ISTE. Wiley, 2013. ISBN 9781118603093. URL `https://books.google.dz/books?id=z1XDNlSceFkC`.

[http ://w3.uqo.ca()] http ://w3.uqo.ca. Développement des systèmes informatiques.

[Isidore(2014)] Chris Isidore. Nissan recalls 1 million vehicles due to airbag flaw. March 2014. URL `http://money.cnn.com/2014/03/26/autos/nissan-recall`.

[istqb certified tester(2018a)] istqb certified tester. What are the software development life cycle (sdlc) phases ?, 2018a. URL `http://istqbexamcertification.com/what-are-the-software-development-life-cycle-sdlc-phases`.

[istqb certified tester(2018b)] istqb certified tester. What is usability testing in software and it's benefits to end user ?, 2018b. URL `http://istqbexamcertification.com/what-is-usability-testing-in-software-and-its-benifits-to-end-user/`.

[Jane Radatz(1990)] Chairpersons Jane Radatz. Ieee standard glossary of software engineering terminology. 1990.

[Janssen()] Dle Janssen. Automation. URL `https://www.techopedia.com/definition/32099/automation`.

[Kaner et al.(1999)Kaner, Falk, and Nguyen] Cem Kaner, Jack L. Falk, and Hung Quoc Nguyen. *Testing Computer Software, Second Edition.* John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 1999. ISBN 0471358460.

[Lei et al.(2007)Lei, Kacker, Kuhn, Okun, and Lawrence] Yu Lei, Raghu Kacker, D. Richard Kuhn, Vadim Okun, and James Lawrence. *IPOG : A General Strategy for T-Way Software Testing.* IEEE, mar 2007. doi : 10.1109/ecbs.2007.47. URL `https://doi.org/10.1109%2Fecbs.2007.47`.

[Lekh and Pooja(2015)] Rachna Lekh and Pooja. Exhaustive study of sdlc phases and their best praxctices to create cdp model for process improvement. *2015 International Conference on Advances in Computer Engineering and Applications*, pages 997–1003, 2015.

[Leung and White(1989)] H. K. N. Leung and L. White. Insights into regression testing [software testing]. In *Proceedings. Conference on Software Maintenance - 1989*, pages 60–69, Oct 1989. doi : 10.1109/ICSM.1989.65194.

[maryland department of technology(2007)] maryland department of technology. Sdlc phase integration and test phase multiple hardware, 2007. URL http://doit.maryland.gov/SDLC/Documents/SDLC%20Phase%2007%20Integration%20and%20Test%20Phase%20Multiple%20Hardware.pdf.

[Mitra et al.(2011)Mitra, Chatterjee, and Ali] P. Mitra, S. Chatterjee, and N. Ali. Graphical analysis of mc/dc using automated software testing. In *2011 3rd International Conference on Electronics Computer Technology*, volume 3, pages 145–149, April 2011. doi : 10.1109/ICECTECH.2011.5941819.

[Mrs.A.Vanitha Katherine(2012)] Dr. K. Alagarsamy Mrs.A.Vanitha Katherine. Conventional software testing vs cloud testing. *International journal of scientific and engineering research*, 2012.

[Murnane and Reed(2001)] Tafline Murnane and Karl Reed. On the effectiveness of mutation analysis as a black box testing technique. pages 12–20, 01 2001.

[Myers and Sandler(2004)] Glenford J. Myers and Corey Sandler. *The Art of Software Testing.* John Wiley & Sons, Inc., USA, 2004. ISBN 0471469122.

[Nathaniel Popper(2015)] Neil Gough Nathaniel Popper. bloomberg terminals outage. april 2015. URL https://www.nytimes.com/2015/04/18/business/dealbook/bloomberg-terminals-outage.

[Nie and Leung(2011)] Changhai Nie and Hareton Leung. A survey of combinatorial testing. *ACM Computing Surveys*, 43(2) :1–29, jan 2011. doi : 10.1145/1883612.1883618. URL https://doi.org/10.1145%2F1883612.1883618.

[petit larousse(2015)] petit larousse. *petit larousse illustré.* larousse, 2015.

[ProfessionalQA.com(2017)] ProfessionalQA.com. Test execution cycle, 2017. URL https://www.forgeahead.io/blogs/test-automation-across-the-phases-of-test-cycle-2.

[rohit Singh(2016)] rohit Singh. Easy way to understand sdlc, 2016. URL http://www.dignitasdigital.com/blog/easy-way-to-understand-sdlc/.

[Roques(2008)] P. Roques. *UML 2 : Modéliser une application web.* Les cahiers du programmeur. Eyrolles, 2008. ISBN 9782212852189. URL `https://books.google.dz/books?id=D3OCYHZGTT4C`.

[Ruparelia(2010)] Nayan B. Ruparelia. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, 35(3) :8–13, May 2010. ISSN 0163-5948. doi : 10.1145/1764810.1764814. URL `http://doi.acm.org/10.1145/1764810.1764814`.

[Russell and Division(2012)] J.P. Russell and A.S.Q.Q.A. Division. *The ASQ Auditing Handbook, Fourth Edition.* ASQ Quality Press, 2012. ISBN 9780873898478. URL `https://books.google.dz/books?id=XZUv5HjPAS8C`.

[software testing class(2018a)] software testing class. Software requirement specification (srs), 2018a. URL `https://www.softwaretestingclass.com/software-requirement-specification-srs`.

[software testing class(2018b)] software testing class. Software testing life cycle stlc, 2018b. URL `https://www.softwaretestingclass.com/software-testing-life-cycle-stlc/`.

[Sommerville(2010)] Ian Sommerville. *Software Engineering.* Addison-Wesley Publishing Company, USA, 9th edition, 2010. ISBN 0137035152, 9780137035151.

[Stephan Christmann(2016)] Bruno Legeard Armin Metzger-Natasa Micuda Thomas Mueller Stephan Schulz Stephan Christmann, Anne Kramer. *International Software Testing Qualifications Board*, 2016.

[team QATestingTools(2017a)] team QATestingTools. Katalon studio, 2017a. URL `http://www.qatestingtools.com/testing-tool/katalon_studio`.

[team QATestingTools(2017b)] team QATestingTools. Katalon studio, 2017b. URL `http://www.qatestingtools.com/testing-tool/sahi-opensource`.

[team QATestingTools(2017c)] team QATestingTools. Katalon studio, 2017c. URL `http://www.qatestingtools.com/testing-tool/watir`.

[Techopedia(2018)] Techopedia. Software development life cycle (sdlc), 2018. URL `https://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc`.

[Tretmans(1996)] Jan Tretmans. Test generation with inputs, outputs, and quiescence. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 127–146. Springer Berlin Heidelberg, 1996. doi : 10.1007/3-540-61042-1_42. URL `https://doi.org/10.1007%2F3-540-61042-1_42`.

[tutorialspoint(2017)] tutorialspoint. Sdlc tutorialspoint, 2017. URL `https://www.tutorialspoint.com/sdlc/sdlc_tutorial.pdf`.

[Vangie and Beal()] Vangie and Beal. Process automation. URL `https://www.webopedia.com/TERM/P/process_automation.htmlByVangieBeal`.

[vineet Kumar(2017)] vineet Kumar. Architecture of the testng framework, 2017. URL `http://www.seleniumwebdriver.in/2017/02/architecture-of-testng-framework.html`.

Première partie

Annexe

Annexe A

Outil de test unitaire

Ici on va essayer de décrire ce qui fait en pratique dans les tests unitaires.
Voici un exemple d'un framework de test « TestNG » inspiré de « JUnit » très connu dans java, utilisé pour couvrir les tests unitaires ainsi que d'autres tests tels que le test fonctionnel, test d'intégration etc.

Présentation de TestNG :

Le framework est développé en utilisant TestNG, POM et Exel libraty. Il s'agit d'une combinaisonde framework de Data-Driven et Methode-Driver, que nous appelons un Hybrid Framework.

L'exécution est contrôlée par le fichier TestNG suite, qui contient la liste des classes TestNG à exécuter.

Chaque classe TestNG a une méthode de test qui s'étend également de la classe BaseTest qui a @BeforeMethod et @AfterMethod.

— Premièrement @BeforeMethod est exécutée, ouvre le navigateur et entre l'URL. Exemple :
Pilote WebDriver = nouveau FirefoxDriver ();
driver.get ("");

— Après l'exécution de @BeforeMethod, l'exécution de testmethod commence. Le testmethod prend les données de la feuille Excel et effectue l'action en appelant la méthode présente dans la classe POM. Exemple : String un=Excel.getCellData(xlpath,sheet,1,0) ;