ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

RAPPORT DE PROJET DE 15 CRÉDITS, PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE À L'OBTENTION DE
LA MAITRISE EN INGÉNIERIE (M. ING) LOGICIELLE

PAR

Yves Lionel KEMMOGNE TCHUENTE

ANALYSE DES CADRICIELS DE TESTS AUTOMATISÉS ET RÉALISATION D'UN PROTOTYPE LOGICIEL D'UN ROBOT DE TESTS AUTOMATISÉS

MONTRÉAL, LE 2 AOUT 2016

PRÉSENTATION DU JURY

CE RAPPORT DE PROJET A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Professeur Alain April, directeur de projet
Département de génie logiciel et des Technologies de l'information à l'École de technologie supérieure.

Professeur Sègla Jean-Luc Kpodjedo, jury
Département de génie logiciel et des Technologies de l'information à l'École de technologie supérieure.

REMERCIEMENTS

J’aimerais adresser mes sincères remerciements au professeur Alain April, directeur de ce projet qui m’a patiemment encadré dans cette recherche. Je remercie également M. ing Ricardo Socarras, directeur de développement web, à la société CGI, pour avoir défini ce projet. Il a cru à mon talent et m’a fait confiance dès notre première rencontre. Sa patience, ses conseils, son expérience et son encadrement m’ont permis d’accomplir avec succès ce projet appliqué de recherche.

Je tiens aussi à remercier ma conjointe Marina Barroso, pour son aide psychologique et moral à distance, face aux difficultés que j’ai rencontrées durant mes premiers mois au Canada. Je remercie aussi mon fils Miguel Kemmogne, qui a été une grande source d’inspiration et de persistance vers la réussite.

Je remercie aussi ma famille, principalement ma maman Christine Ndassa, qui a toujours été à l’écoute de mes problèmes, à me soutenir et à me proposer des solutions malgré leurs difficultés financières. Ils méritent un remerciement particulier.
Pour terminer, je remercie mes collègues de CGI qui ont participé à ce projet de recherche, et à mes camarades de classe qui ont été, pour moi, comme une famille au Canada.

ANALYSE DES CADRICIELS DE TESTS AUTOMATISÉS ET RÉALISATION D'UN PROTOTYPE LOGICIEL D'UN ROBOT DE TESTS AUTOMATISÉS

Yves Lionel KEMMOGNE TCHUENTE

RÉSUMÉ

Toute société exerçant dans le domaine du développement logiciel effectue des tests afin de s'assurer de la qualité du produit final. Tel que dicté par les pratiques exemplaires de génie logiciel, les tests devraient débuter avant de commencer le développement du logiciel proprement dit. Les tests logiciels peuvent donc être effectués manuellement et automatiquement, à l'aide de cadriciels et outils de tests. Il existe plusieurs outils de tests disponibles aujourd'hui aussi bien sous la forme de logiciels libres que commerciaux.

De plus en plus, les tests automatisés sont une nécessité et sont indispensables pour diverses raisons : la réduction des coûts, la réduction des délais d'exécution des tests, l'augmentation de la couverture des tests, et la réduction des défauts. Ainsi le centre de service de la paie du bureau de Montréal de la société CGI souscrit à cette logique et souhaite automatiser ses tests.

Le premier objectif de ce projet est d'analyser différents cadriciels de tests automatisés disponibles afin d'établir une liste de ceux retenus. Cette liste servira de référence pour recommander et justifier le choix d'un cadriciel qui serait le mieux adapté pour ce contexte spécifique. Le deuxième objectif est de réaliser, à partir de ce cadriciel, un prototype logiciel d'un robot de tests automatisés offrant diverses fonctions, telles que : 1) l'exécution automatique et périodique des tests; 2) la navigation dans les applications web (c.-à-d. Nethris et EmployeurD); 3) la vérification des fonctionnalités des applications; et 4) la validation de la logique d'affaires. De plus, il a été demandé que cette solution puisse effectuer la capture d'écran, quand une page web contient des défauts. Cette information doit être envoyée, par courriel, au groupe de supports technique de CGI. Finalement, il est requis que le nettoyage des fichiers et des dossiers inutiles dans le serveur soit effectué automatiquement.

Mots-Clés : tests logiciel, tests automatisés, cadriciels de tests, SpecFlow, Selenium, Gherkin, BDD.

ANALYSIS OF AUTOMATED TESTING TOOLS AND PRODUCTION OF A PROTOTYPE OF SOFTWARE TESTING AUTOMATED ROBOT

Yves Lionel KEMMOGNE TCHUENTE

ABSTRACT

Any company operating in the field of software development performs tests of its software to ensure the quality of the final product. As recommended by software engineering best practices, testing should begin before starting to develop software. Software tests may be carried out manually and automatically, using framework and testing tools. Several are available today as free software or commercially.

Increasingly, automated tests are desirable by companies and this for various reasons. This is the case of the payroll service center of Montreal office of the company CGI that wants to automate tests for the following reasons: cost reduction, tests execution times reduction, increasing the test coverage and reduce defect.

The first objective of this project is to analyze different automated testing frameworks available. From this list, make a recommendation and justify the choice of a software framework that would be best suited for that specific environment. The second objective is to achieve from this software framework, a software prototype of an automated testing robot with various functions, such as: 1) automatic and periodic test execution; 2) browse in web applications (ie Nethris and EmployeurD). 3) Verification of application functionality; and 4) validation of the business logic. In addition, it was requested that this solution can perform the screenshot when a webpage contains defects. This information should be sent by e-mail to technical supports group of CGI. Finally, it is required that the cleaning of folders and unnecessary files on the server is done automatically.

Keywords: software testing, automated testing, testing framework, SpecFlow, Selenium, Gherkin, BDD.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

ACRONYMES

IHM…….....Interface Homme-Machine
GUI…….……..Graphical User Interface
TDD……Test Driven Development
BDD…….Behavor Driven Development
ATDD……Acceptance Testing Driven Development
SUT……...System Under Test
DLL……..Dynamic Link Library
IDE……..Integrated Development Environment
UFT……...Unified Fonctional Testing
ROI……...Return On Investment
VS……..Visual Studio
TFB……Team Foundation Build
TFS…….Team Foundation Serveur
ALM……Application Lifecycle Management

INTRODUCTION

Pour effectuer le développement d'un nouveau logiciel, plusieurs étapes et activités de cycle de vie sont nécessaires parmi lesquelles la vérification des fonctionnalités et la validation des règles d'affaires initialement exprimées par la clientèle. Les tests logiciels et l'assurance de la qualité jouent un rôle incontournable dans la réussite et la mise en œuvre de tout logiciel [1]
Le but des tests logiciels est d'identifier le maximum de défauts existant dans le logiciel avant qu'il soit livré. C'est un processus d'évaluation détaillée du système et de ses composants par des moyens automatiques ou manuels, afin de vérifier qu'il satisfait à aux exigences spécifiées en identifiant des différences entre les résultats observés par rapport à ceux attendus.

Les principaux problèmes des tests manuels sont : la longue durée d'exécution des tests; la non-réutilisabilité des tests, ils nécessitent un plus grand effort; et ils sont propices aux erreurs humaines [2] Les tests automatisés complémentent les tests manuels et permettent de résoudre ces problématiques. Le but de l'automatisation des tests est de réduire le nombre de cas de test exécuté manuellement et non pas de les éliminer complètement. L'exécution des tests automatisée réduira l'effort, augmentera la rapidité d'exécution et permettra de sauver des couts. [3]

Aujourd'hui, il existe plusieurs outils d'automatisation des tests ainsi que plusieurs approches pour effectuer sa mise en œuvre. Ce projet de recherche appliquée de 15 crédits vise à expérimenter cette problématique, à l'aide d'une étude de cas, dans une grande société de consultation de logiciel qui souhaite faire le choix d'un outil d'automatisation de tests afin de s'en servir pour tester les changements continus à son application de la gestion de la paie.

CHAPITRE 1

Contexte

1.1 Le système actuel en place

Le système existant, qui fera l'objet d'expérimentation, est une application web de la gestion des paies développée sur la plateforme .Net de Microsoft. C'est une application de grande taille (c.-à-d. ayant environ 2 millions de lignes de code) regroupant plusieurs fonctions. [4]
Actuellement, lors de l'exécution de changements, les développeurs effectuent des tests informellement et de façon manuelle. Cette approche ne permet pas de faire une bonne couverture de test et cause des problèmes de qualité. Dans les étapes d'essais, les tests effectués manuellement sont employés pour la vérification des fonctions et la validation des règles d'affaires de l'application.

1.2 Problématiques

Puisque les tests sont faits de façon manuelle, l'organisme se pose actuellement deux questions en ce qui concerne son désir à migrer vers l'automatisation.

- Back-end : Est-ce qu'il est possible d'automatiser les tests unitaires, dans le code source de notre application de la paie lors de la phase de réalisation afin d'assurer la conformité?
- Front-End : Est-ce qu'il est possible d'automatiser les tests fonctionnels, intégration et acceptation avant la mise en production, afin de vérifier les fonctionnalités et valider la logique des logiciels?

1.3 Approche proposée

Ce projet débute par une étape de démonstration des bénéfices potentiels de l'automatisation des tests logiciels à CGI et à tous les intervenants qui sont impliqués dans la réalisation du logiciel de la paie. Ensuite, deux activités, préalables au développement du robot, sont proposées: 1) l'étude du système existant; et 2) le choix d'un cadriciel de tests (c.-à-d. un outil) le mieux adapté pour ce contexte.

1.4 Objectif visé

Ce projet de recherche appliquée vise premièrement à accompagner la société CGI dans la mise en place de l'automatisation des tests logiciels. Deuxièmement, il vise à choisir et réaliser un prototype logiciel d'un robot de tests automatisés à titre de preuve de concept de l'automatisation des tests.

1.5 Portée

Tel que présenté à la figure 1-1, le cycle de développement logiciel de CGI est constitué de cinq phases. Ce projet basé sur l'automatisation des tests, couvre uniquement les quatre dernières phases du cycle, dont test automatisé en boite blanche (impliqué dans la phase de réalisation) et test automatisé en boite noire (impliqué dans les phases des essaies fonctionnels, intégrations et acceptations).

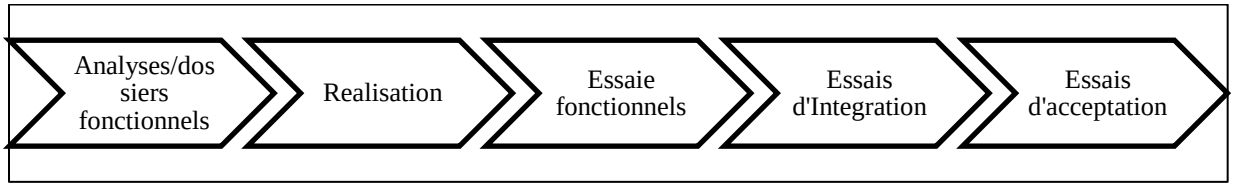


Figure 1-1 : Cycle de développement logiciel du centre de service de la paie de CGI [4]

1.6 Conclusion

Ce chapitre présentait le contexte dans lequel le projet a eu lieu, la problématique qui sera traitée dans le projet, l'approche proposée pour arriver aux objectifs définis et la portée du travail qui sera réalisé.

Le chapitre ci-dessous est une revue littéraire sur la notion d'automatisation des tests.

CHAPITRE 2

Revue de la littérature

Ce chapitre est un survol des notions théoriques de test logiciel de l'origine du domaine jusqu'aujourd'hui. Il présente aussi une synthèse de l'état de l'art concernant l'automatisation des tests, les enjeux de l'automatisation, les facteurs d'évaluation d'outil de tests et quelques outils proposés par la communauté.

2.1 Test logiciel informatique

2.1.1 Définition

Les tests sont reconnus comme étant la pierre angulaire du contrôle de la qualité du logiciel par un grand nombre de développeurs [5]. Un test logiciel est une méthode dynamique de vérifications de la conformité logicielles en s'inspirant des spécifications initiales et visant intentionnellement à trouver le maximum possible des bogues dans un système [6].

2.1.2 Historique

L'origine de techniques de tests logiciels remonte aux années cinquante. Ces techniques ont depuis lors été en constante évolution. Par exemple Glenford J. Mayer est le premier à émettre l'hypothèse qu'on doit faire une distinction entre le débogage : qui signifie l'identification et l'élimination des défauts dans le code du logiciel, et le test : qui signifie l'exécution d'un programme dans l'environnement réel d'opération et la recherche intentionnelle des défauts.

Le débogage est un processus exécuté en deux étapes et qui débute lorsqu'on trouve une erreur à la suite d'un cas de test réussi. La première étape vise la détermination de la nature exacte et la localisation de l'erreur suspectée dans le programme. La deuxième étape consiste à corriger l'erreur.
Le test est le processus d'exécution d'un programme avec l'intention de trouver des défauts. [7]

2.1.3 Évolution

De nos jours, de plus en plus d'outils peuvent aider à produire, appuyer et même enregistrer les cas de tests avec l'objectif de les rejouer automatiquement, et ce, sans intervention humaine. L'industrie a tout intérêt à utiliser cette approche, car les tests demandent actuellement beaucoup d'effort [8]. Cette situation a bien évolué depuis les années cinquante. Voici une récapitulation des grandes étapes des tests depuis ces années :

- Jusqu'en 1956 - Débogage orienté : à cette époque, il n'y avait pas de différence claire entre débogueurs et testée.
- 1957-1978 - Démonstration orientée : à cette époque, il existe déjà la différence entre déboguer et tester. La satisfaction des exigences par le logiciel devient importante.
- 1979-1982 - Destruction orientée : cette époque est marquée par un objectif qui est de trouver les erreurs dans le logiciel.
- 1983-1987 - Évaluation orientée : Dans cette période, l'intention est d'évaluer le produit Fournier et de mesurer la qualité.
- 1988-2000 - Prévention orientée : Marqué par la détection des défauts et la prévention [9].

Les tests sont généralement découpés selon les types (c.-à-d. unitaires, fonctionnels, d'intégration, d'acceptation, etc.) et par catégories (c.-à-d. statique et dynamique, boite blanche et boite noire).

2.1.4 Catégories et types de tests

A. Types de tests

Le tableau suivant présente quelques types de tests et leurs descriptions.

Tableau 2.1 : Organisation des tests par phase de développement

TYPES DE TESTS	DESCRIPTIONS
Tests unitaires	consistent à tester individuellement les composants de l'application
Tests fonctionnels	ont pour but de vérifier la conformité de l'application développée en se basant sur les spécifications fonctionnelles et techniques.
Tests de non-régressions	permettent de vérifier que des modifications n'ont pas altéré le fonctionnement de l'application.
Tests de validation IHM	ont pour but de vérifier que la charte graphique a été respectée tout au long du développement.
Tests d'intégration	Pour assurer le bon fonctionnement des différents composants ou modules lorsqu'intégré ensemble.
Tests d'acceptation	Généralement effectués par le client du produit logiciel, les tests d'acceptation permettent de s'assurer que le produit logiciel est conforme aux spécifications initiales.

On retrouve aussi certains autres types de tests tels que :

- Tests de configurations : une application doit pouvoir s'adapter au renouvèlement de plus en plus fréquent des ordinateurs. Il s'avère donc indispensable d'étudier l'impact des environnements d'exploitation sur son fonctionnement.
- Tests de performance : le but principal des tests de performance est de valider la capacité qu'ont les serveurs et les réseaux à supporter des charges d'accès importantes.

Il faut notamment vérifier que les temps de réponse restent raisonnables lorsqu'un nombre important d'utilisateurs sont simultanément connectés à la base de données de l'application.

- Tests d’installation : une fois l'application validée, il est nécessaire de contrôler les aspects liés à la documentation et à l'installation. Les procédures d'installation doivent être testées intégralement, car elles garantissent la fiabilité de l'application dans la phase de démarrage. Bien sûr, il faudra aussi vérifier que les supports d'installation ne contiennent pas de virus.

B. Catégories de tests

On peut diviser les tests selon plusieurs catégories dont en voici quelques :

- Tests statiques vs dynamiques : les tests statiques sont généralement les tests basés sur la vérification telle que, les revues, les inspections, et, etc. Tandis que les tests dynamiques sont les tests de validation. Ils sont exécutés lorsque le logiciel est complètement terminé et déployé.
- Tests en boite blanche vs tests en boite noire : les tests en boite blanche sont les tests qui sont effectués à l’intérieur du système, back-end, sur le code source. Par contre, les tests en boite noire sont effectués à l’extérieur du système, font-end, sur l’interface utilisateur personne-machine.
- Tests manuels vs tests automatisés : les tests manuels sont exécutés par les êtres humains. Cette pratique consisté à assigner à plusieurs personnes des cas de tests dont ils vont les exécuter attentivement et minutieusement cas par cas dans l’intention de retrouver des défauts dans l’application. Quant aux tests automatisés, c’est exactement la même pratique, à l’exception que les tests seront exécutés de façon automatique grâce aux scripts de tests écrire par les développeurs à base d’un ou plusieurs cadriciels de tests.

2.2 État de l’art : Automatisation des tests

2.2.1 Ce que les auteurs pensent de l’automatisation des tests

Les techniques de tests logiciels s’améliorent constamment, si bien que l’automatisation est déjà chose courante dans les organisations [10]. On peut définir l’automatisation des tests comme étant : l'utilisation d'un cadriciel ou d’un outil spécialisé afin de contrôler et faciliter l'exécution des tests et la comparaison des résultats obtenus avec les résultats prévus. Plusieurs de ces cadriciels et outils rendent possibles l’exécution périodique et la génération automatique des rapports de tests [11]. Pour Hildreth, l'automatisation peut réduire significativement le rôle des tests manuels, et ainsi raccourcir le délai de mise en production des applications [12]. Dustin, appuie cet énoncé, en décrivant ses résultats qui démontrent que l’utilisation de ces outils d’automatisation réduit l'effort de test [13].

Berner et ses collègues décrivent, dans leur article, que l’automatisation ne peut pas totalement remplacer les activités de tests manuels. Ils expliquent que toutes les tâches ne peuvent pas être facilement automatisées, en particulier celles qui nécessitent des connaissances approfondies d’un domaine spécifique [14].

Mayer, de son côté identifie une relation entre les tests et l'assurance de la qualité du logiciel. Il met en évidence que l'utilisation d'outils de tests automatisés améliore la qualité du logiciel [15].

Shea met l'accent sur l'importance de l’utilisation de ces outils et discute de l'adoption croissante des tests automatisés par les entreprises. Il fournit également une estimation du taux de pénétration des outils d’automatisation dans les organisations, et propose des facteurs qui influencent l'adoption de solutions de tests automatisés [16].

Pour Bashir et Banuri, le processus d'automatisation de test a besoin de temps pour murir. Cela est dû à la nécessité de créer une infrastructure pour automatiser les tests. Cela exige du temps et des efforts, d'où la nécessité d’avoir atteint une certaine maturité avant de pouvoir faire de l'automatisation [17].

2.2.2 Ce que disent les auteurs concernant l'acquisition d'un outil de tests

Vail stipule que le facteur technique le plus important, considéré avant l'acquisition d'un outil de test automatisé, est sa compatibilité avec l'application logicielle à tester. Ensuite, un autre critère important est la part de marché occupée par l'outil, la situation financière du fournisseur de l'outil et la disponibilité d'un service d'assistance technique [18].

Selon Bach, l'essentiel, pour le succès de ce genre de projet, n'est pas seulement d'acquérir un outil de tests, mais aussi la manière dont cet outil sera utilisé pour concevoir des scripts de tests, la façon dont les scripts sont conçus et le profil du développeur qui sera assigné à cette tâche [19].

Le choix d'un outil d'automatisation de test se fait donc en fonction du type d'application à tester, du budget disponible et des fonctionnalités requises [11]. Schwaber et ses collègues citent quatre principaux facteurs qui contribuent au succès du projet de mise en œuvre d'un outil de test de logiciel [20]:

- Offre d'un soutien technique solide ;
- Suit le niveau de compétences des utilisateurs finaux destinés ;
- Intégration d'outils internet du cycle de vie de développement de logiciels (SDLC) ;
- Intégration d'outils internet de la gestion du cycle de vie des applications (ALM).

2.2.3 Les outils de tests populaires

L'automatisation des tests requiert d'utiliser un ou plusieurs outils dédiés. Actuellement, un grand nombre d'outils sont disponibles sur le marché. Parmi ces outils, certains ne peuvent effectuer que des types de tests spécifiques et se restreignent à certains langages de programmation spécifiques. D'autres prennent en charge un large éventail d'applications et offrent plus de fonctionnalités. [10]

Harpreet fait l'analyse des trois outils de tests : QTP, Selenium et TestComplete. Pour son cas d'utilisation, il obtient le tableau de synthèse ci-dessous.

Tableau 2.2 : Synthèse des outils de tests par Harpreet [21]

FEATURES	SELENIUM	QUCIK TEST PROFESSIONAL	TESTCOMPLETE
Licensing Cost	It is open source. So, there‟s no licensing or renewal cost for this tool. It‟s free of cost.	Licensed and very Expensive, Ten user license costs approx. 60L.	$2K Enterprise Seat License
Application support	Web Applications only it supports addition of plugins to achieve desired results that are not provided by Selenium Core. Since, selenium is open source, plug-ins are also available free of cost.	A client server application Only. It also supports add-ons, but user needs to purchase license for them	All of this included right out of the box there are no plug-ins or add-ons to buy. You can install Test Complete and immediately create any test against any application.
Object Oriented Language support and Scalability	Supports Java, .Net, Perl, PHP, Python and Ruby.	Scripts can be developed only in VBScript or JavaScript.	Test Complete supports scripting in VBScript, JSScript, DelphiScript, C++Script and C#Script, so you can create scripts in the language
Support for operating system/platforms	Supports Windows PC/MAC/UNIXPlatforms.	QTP supports only Windows XP.	Windows 7, Windows Vista, Windows Server 2008 or later operating systems.

Tableau 2.3 : Synthèse des outils de tests par Harpreet - Suite [21]

FEATURES	SELENIUM	QUCIK TEST PROFESSIONAL	TESTCOMPLETE
Programming skills	For using Selenium one needs to have programming skills.	QTP is quite easy to use. It is quite easy to edit the script, parameterize, navigate, playback and validate the results.	TC is good for both web based and desktop application.
Usage	Selenium needs quite a bit of expertise	QTP is quite easy to learn in a short time.	Support for all 32-bit and 64-bit window application.
Database applications	With Selenium one needs to exert hard to do the same job.	QTP works very well with database applications	TC works very well database application.
Platform dependency	With Selenium these tasks can be easily accomplished.	It is difficult to deploy smoke tests for web applications using QTP especially with Windows7.	It is difficult to deploy application using.
Report Generation	Selenium users don‟t enjoy such luxury as enjoyed.	With QTP we can easily generate most comprehensive reports due to the availability of an efficient online help.	Report generation is an easy-to-use utility that is support along with TC and lets you generate dump files.

C’est aussi le cas des analyses comparatives faites par la compagnie Aspire Systems, Inc. sur les outils UFT et Selenium qui conclut que le choix dépend du contexte d’utilisation et de l’entreprise. L’argument principal de cette conclusion est qu’il est aussi simple entre l'achat d'une voiture chez un concessionnaire (faisant allusion à l’outil UFT) et d'assembler la voiture soi-même (faisant allusion à l’outil Selenium). La voiture idéale est livrée avec des garanties et des services de maintenance alors que dans un kit, vous devez prendre soin de tout faire vous-même et le résultat est votre propre responsabilité [22].

Quant aux analyses comparatives effectuées par la compagnie SmarBear, le tableau de synthèse ci-dessous a été ressorti afin de conclure que TestComplete est le meilleur outil.

Tableau 2.4 : Synthèse des outils de tests par SmartBear [23].

Features	**TestComplete**	**Selenium**	**UFT**
Record and Playback	Yes	Limited	Yes
Scripting Languages	Python, VBScript, JScript, C++, C#, and Delphi	Java, C#, Ruby, Python, Perl, PHP , Javascript	VBScript
IDE Integrations	VisualStudio and RADStudio	Intellij, Eclipse, Visual Studio	None
Unit Testing Support	PyUnit, Ruby, PHPUnit, JUnit, NUnit, and TestNG	Java, C#, Ruby, Python,Perl, PHP , Javascript	VBScript
BDD/TDD Support	Yes	Manual	None
Data-Driven Testing	Yes	Manual	Yes
Source Control Management	Git, Subversion, Visual SourceSafe, CVS, Team Coherence	Manual	Git and Subversion

Une autre analyse comparative effectuée par Gouri parmi les outils : Selenium, QTP et Code UI, indique les résultats dans un tableau de synthèse [24] ou il affirme que Coded UI est le meilleur outil d'automatisation du à :

- La facilité d'utilisation de son IDE;
- Le support des objets;
- La possibilité de tester divers types d'applications (web, Windows et mobile);
- L'intégration d'un ALM.

Yogesh affirme que Selenium WebDriver (également connu sous le nom de Selenium 2.0) est le meilleur outil d'automatisation de test. Il permet de sauver des couts, car il est gratuit, mais aussi il permet de sauver du temps de programmation à l'aide de ses scripts facile [25].
Vinita, quant à elle, effectue une étude comparative d'un total de neuf outils d'automatisation de tests en fonction du type d'application à tester, du budget et de l'efficacité. Elle insinue que « si vos besoins d'automatisation de test sont remplis avec Testcomplete alors, il n'y a pas de raison d'opter pour QTP à un coût plus élevé, car, ces deux outils résolvent les mêmes problèmes, à la différence que QTP est un outil polyvalent pour une demande critique et plus risquée de l'application sous tests (AUT). Selenium peut également être utilisé si vous ne voulez pas dépenser en outil de test. En conclusion, QTP est le meilleur outil parmi tous » [26].

2.3 Conclusion

Cette revue littéraire a présenté une synthèse du domaine des tests, la notion de tests automatisés suivis de ce que les auteurs pensent de l'automatisation des tests. Il a aussi abordé les facteurs d'évaluation d'outil de tests d'après certains auteurs et quelques outils proposés par la communauté. À partir de cette revue, la remarque est que le choix d'un cadriciel ou d'un outil de tests dépend toujours de certains facteurs spécifiques de la compagnie et du logiciel qui sera testé. Dans le chapitre suivant, une analyse des meilleurs cadriciels d'automatisation des tests sera faite selon les critères définis, et le choix du mieux adapté suivra.

CHAPITRE 3

Canalise et choix du cadriciel de tests le mieux adaptés

Bien qu'il existe plusieurs cadriciels et outils d'automatisation des tests logiciels, son choix est une étape importante qui aura un effet sur tout le reste du projet. Ce chapitre présente une synthèse de la recherche, de l'analyse et du choix des outils d'automatisation des tests les plus populaires, en faisant ressortir les avantages et les inconvénients de chacun, pour le contexte spécifique à l'étude ce cas de la compagnie CGI, dans le but de choisir celui qui sera le mieux adapté à son logiciel de gestion de la paie.

3.1 Recherche et analyse des cadriciels de tests automatisés

Afin de mieux encadrer la recherche d'outils potentiels, il est important de préciser la plateforme de développement et le type d'application qui sera testé lors de cette recherche.

- Plateforme : le développement du système de la gestion de la paie s'effectue actuellement sur la plateforme .Net, sur Visual Studio professionnel 2010 et Visual Studio prémium 2013. Les langages de programmation utilisés sont le C# et le VB.
- Type d'application à tester: application web s'exécutant sur tout type de navigateurs et tous les systèmes d'exploitation populaires (c.-à-d. Windows, Mac OS, Linux…).

Suite à une recherche initiale, les outils suivants ont été listés à titre d'outils potentiels: Ms Tests, CODE UI, Nunit, Selenium, WatiN, Mbunit, Owin Testing, Test Architect, Automation Anywhere, TestComplete, Team System web Test, UFP et Squish GUI Tester.
Avec autant de choix, lesquels éliminer, présélectionner, expérimenter et évaluer? Un premier filtrage est effectué en s'inspirant des critères suivants :

- Suggestions des partenaires de la compagnie : Outil recommandé UFT;
- Suggestion de mon superviseur académique à ÉTS : outil recommandé Selenium;
- Faire ressortir, de suggestions de conférences internationales des tests automatisés en ligne [27]: outils recommandés CODE UI, Selenium;

- Synthétiser les recommandations incluses dans les présentations PowerPoint sur le sujet qui sont publiées sur [28]:outil recommandé CODE UI, Selenium;
- Résultats les plus présents après recherche sur Google : outil trouver UFT et Test Complete.

À l'aide de ces critères, quatre outils ont été présélectionnés : Code UI, Selenium, Testcomplete et UFT. La prochaine section présente chaque outil.

3.1.1 L'outil d'automatisation de test Coded UI

Édité par l'entreprise Microsoft, l'outil d'automatisation de tests Coded UI est relativement nouveau sur le marché. Il est devenu disponible dans le cadre de la mise à jour de Visual Studio en 2010. Le produit a subi de nombreuses améliorations depuis, et une nouvelle version a été publiée dans le cadre de la mise à jour de Visual Studio en 2013. Actuellement, le code source peut être facilement examiné dans Visual Studio. Il y a aussi une fonction de complétion de code (c.-à-d. la fonction IntelliSense) qui contribue à générer du code source plus rapidement. Cet outil d'automatisation de l'interface utilisateur, est soutenu par les langages de programmation C # et Visual Basic .NET. L'outil Coded UI est intégré dans les versions prémium et ultimate 2013 de Visual Studio et enterprise 2015. En plus de permettre d'automatiser les tests des applications web, il peut être utilisé pour tester les applications Windows (client-serveur) et mobile [29].
La figure 3-1, ci-dessous, présente les fonctionnalités de l'IDE de Coded UI, des différents fichiers générer et leur description ainsi qu'une vue du code source généré.

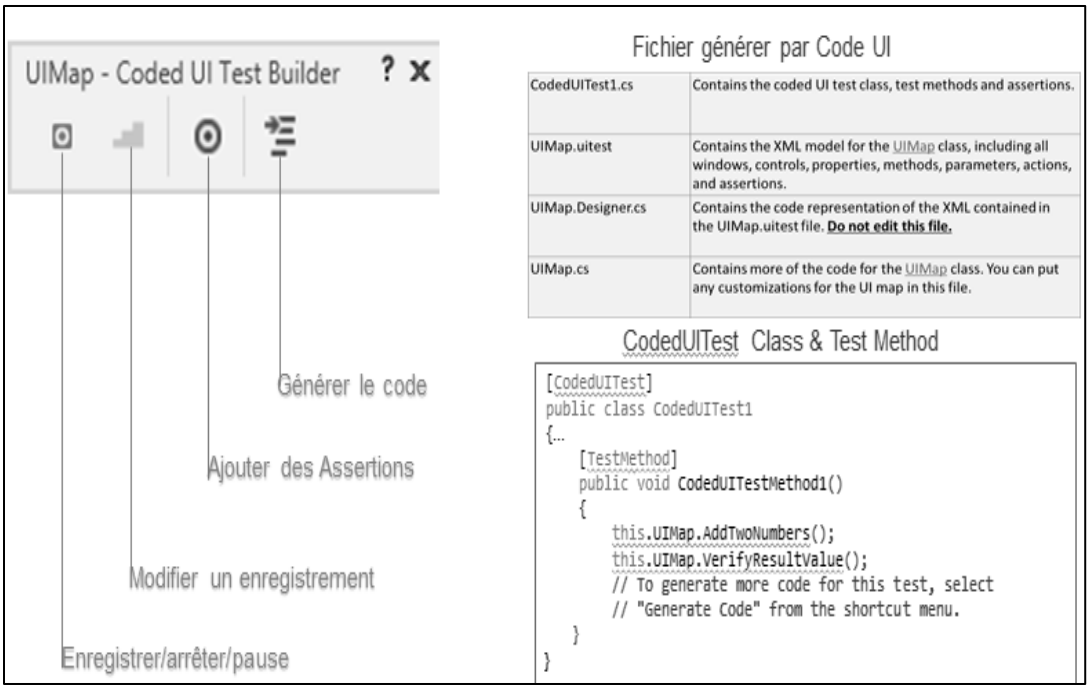


Figure 3-1 : Présentation de CODE UI [29]

Les capacités robustes de Visual Studio et de son Team Foundation Server (TFS) ont fait d'eux des favoris parmi les environnements de développement de logiciels modernes. Les développeurs utilisent ces deux environnements de travail pour créer des applications logicielles d'entreprise. L'utilisation combinée de TFS, Visual Studio et de ses outils de test appuie un processus de développement agile. Voici quelques raisons pour lesquelles l'outil d'interface utilisateur Coded est un choix préféré pour les testeurs de logiciels [30]:

- Coded UI permet le développement d'une gamme complète de test et effectuée des tests dans des environnements locaux;
- Testeurs et développeurs logiciels peuvent travailler en utilisant les mêmes outils / langue, ce qui leur permet de collaborer efficacement;
- Coded prend en charge les projets web et Windows et mobile, et les langages C # et VB.NET connus pour leur robustesse ;
- Les tests automatisés peuvent être exécutés sur des machines distantes avec l'aide de « tests Agents »;
- Coded prend en charge les contrôles AJAX.

- Une des plaintes concernant cet outil est sa lenteur et sa lourdeur, ainsi que la grande quantité de code source, avec une structure assez complexe qui est générée par l'outil [30].

Les figures A-I-1 et A-I-2, à l'annexe I, présentent respectivement la procédure de création d'un projet de test à l'aide de Coded UI dans Visual Studio ainsi que le code source qui est généré suite à l'automatisation d'un simple cas de test qui test la navigation sur une page web.

Un tutoriel bien détaillé et expliquant sommairement le fonctionnement de cet outil de test se trouve à cette référence [31].

3.1.2 L'outil d'automatisation de test Selenium

Selenium est constitué d'un ensemble de composants (c.-à-d. IDE, Iwebdriver, Grid) utilisés pour tester les interfaces graphiques des applications web. Selenium est un logiciel libre et gratuit. Il offre un grand nombre de commandes prédéfinies et supporte plusieurs langages de programmation tels que : Java, c#, vb, Ruby, Python, Perl et Php. Il permet de tester des applications web fonctionnant sur plusieurs navigateurs, tels que : Firefox, Internet Explorer, Safari, Chrome, et, etc. Il peut donc fonctionner sur plusieurs systèmes d'exploitation.

La figure 3-2, ci-dessous, décrit le fonctionnement d'ensemble de Selenium.

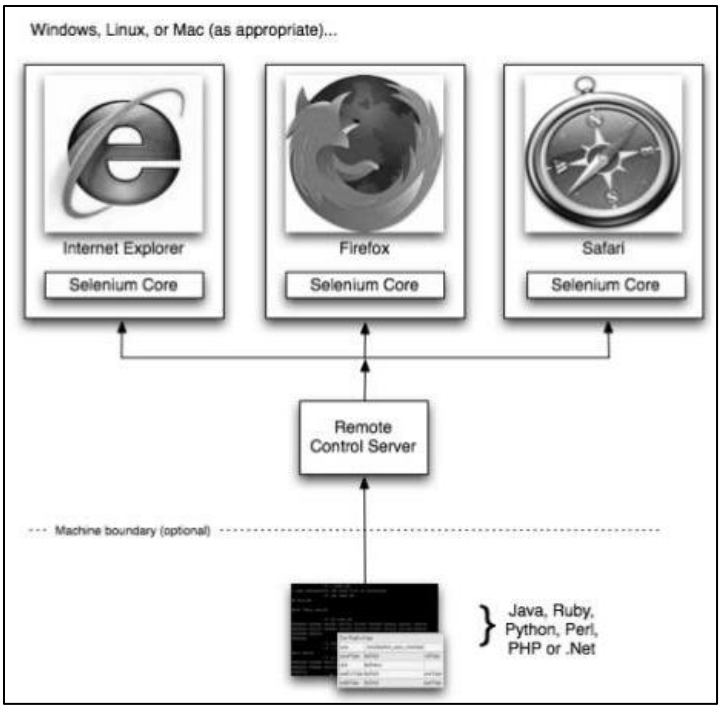


Figure 3-2 : Fonctionnement des composants Selenium [32]

Le composant Selenium IDE est facile à utiliser, flexible, et une expérience poussée de la programmation n'est pas nécessaire pour son utilisation. Il enregistre les tests sous le format HTML et possède aussi la capacité de convertir les scripts résultants vers différents langages de programmation. À l'aide de cet IDE, l'utilisateur peut également chercher les défauts et définir des points d'arrêt. Il peut aussi planifier les périodes d'exécution des tests.

Le plus grand inconvénient de cet outil est qu'il a été conçu comme un « Plugin Firefox », donc son soutien est limité à Firefox uniquement.

La figure 3-3, ci-dessous, décrit l'IDE de Selenium version 2.3.0.

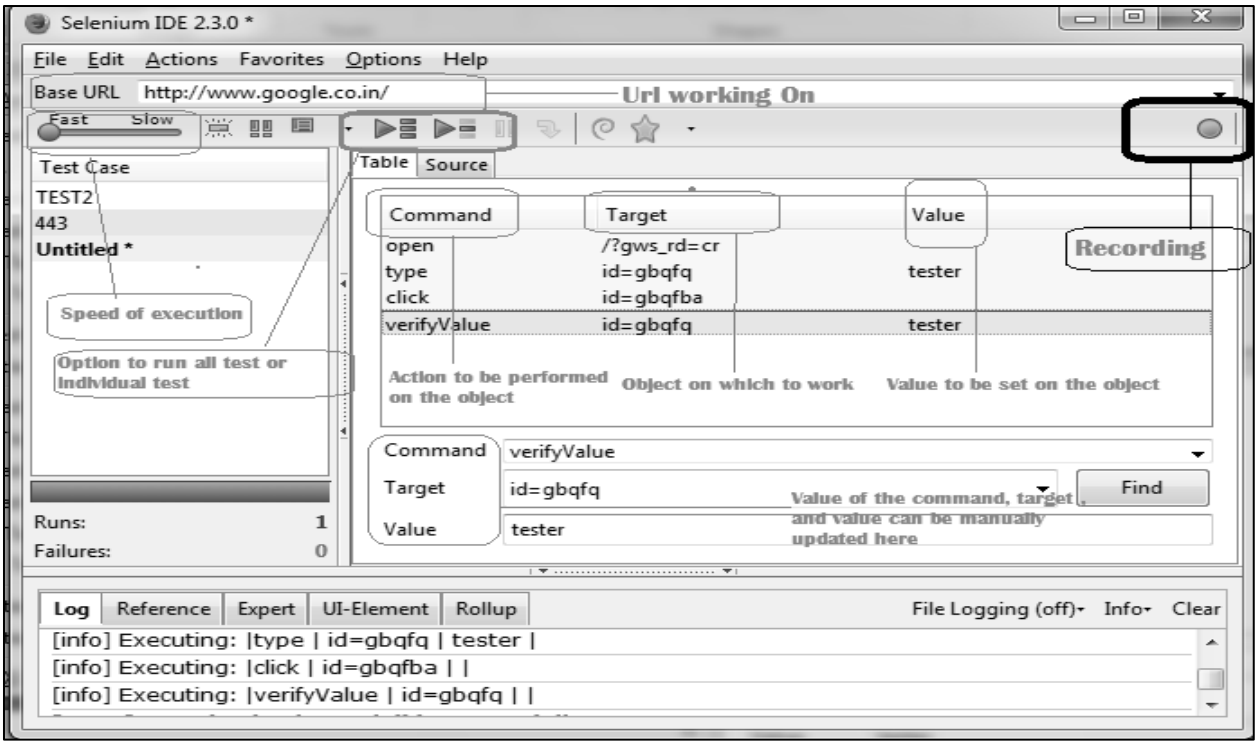


Figure 3-3 : Présentation de l'outil Selenium IDE [33]

Un tutoriel bien détaillé et expliquant sommairement le fonctionnement de cet outil de test se trouve à cette référence [34].

3.1.3 L'outil d'automatisation de test TestComplete

Réalisé et commercialisé par l'entreprise Smart Baer, l'outil TestComplete possède une architecture ouverte et flexible qui permet de créer, maintenir et exécuter des tests automatisés pour les logiciels client-serveur, web et mobile. TestComplete offre aux testeurs la possibilité de créer des tests automatisés pour les applications Microsoft Windows, Web, Android et iOS. Les tests peuvent être enregistrés, scriptés ou manuellement créés avec des opérations axées sur le mot-clé (c.-à-d. « Checkpoints »). Cet outil est caractérisé par ses fonctionnalités puissantes qui démontrent sa souplesse et sa facilité d'utilisation, des tests basés sur les mots-clés, test scriptés, tests enregistrement et lecture. [35]

La figure 3-4, ci-dessous, présente l'interface graphique de l'outil TestComplete et ces fonctionnalités.

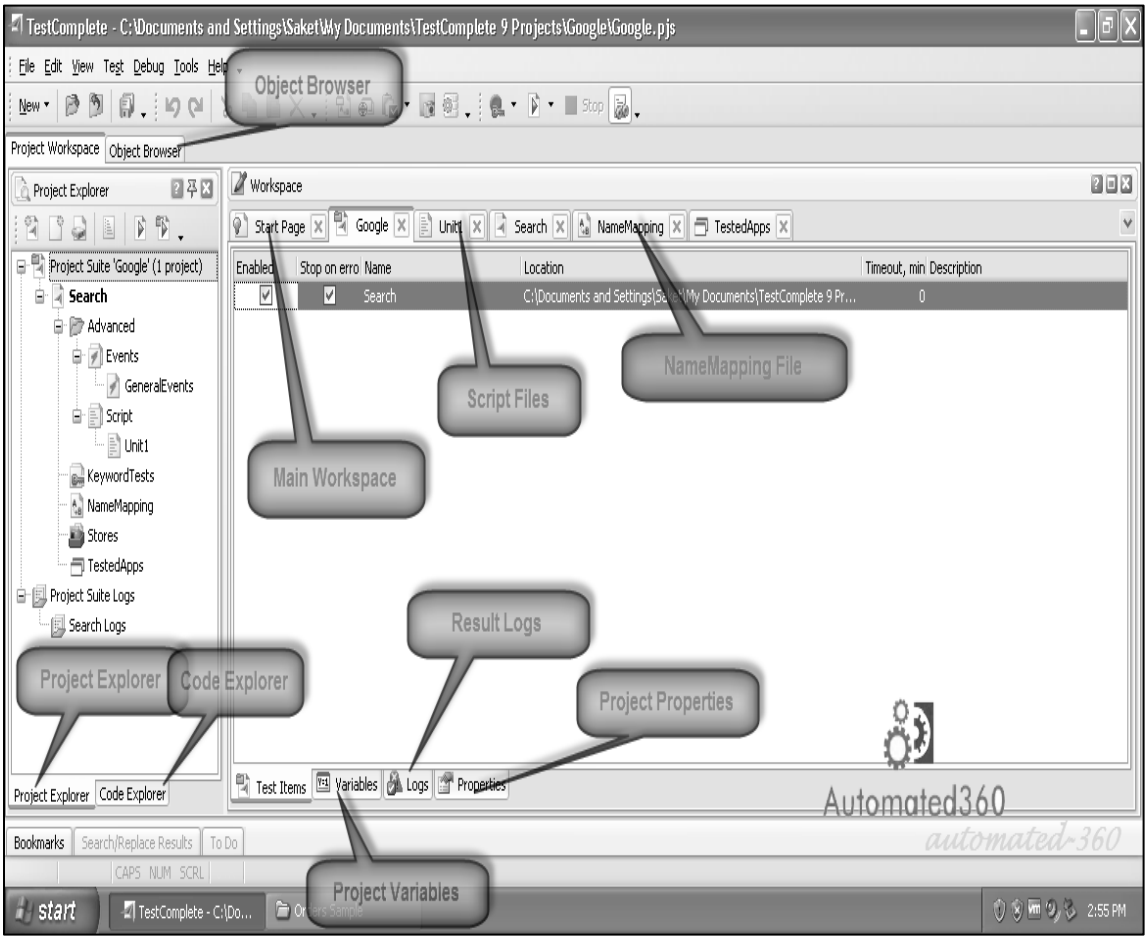


Figure 3-4 : Présentation de TestComplete [36]

Une brève présentation de la procédure de création d'un nouveau projet de test automatisée, comment enregistrer un test et les résultats de tests sont présentés dans les figures A-II-1 à A-II-2 de l'annexe II.

3.1.4 L'outil d'automatisation de tests UFT

Réalisé par la société Hewlett Packard (HP) et anciennement nommé QTP (Quick Test Professional), UFT (Unified Functional Testing) est le meilleur outil de test fonctionnel et de

régression de logiciel [37]. L'environnement de développement intégré (c.-à-d. son IDE) offre diverses fonctionnalités qui justifient cette mention. Il utilise le langage de programmation VB Script pour exécuter le script de test. UFT 12.0 prend en charge Safari sur Mac, Internet Explorer et tous les navigateurs supportés par Windows [37].

Il est très facile à utiliser, même pour un néophyte en programmation. Tout comme les autres outils, il effectue les enregistrements et la relecture des automatiques des scripts de tests. Il dispose aussi de la capacité d'éditer les scripts après enregistrement.

La figure 3-5, ci-dessous, présente l'interface graphique de l'outil UFT et ces fonctionnalités.

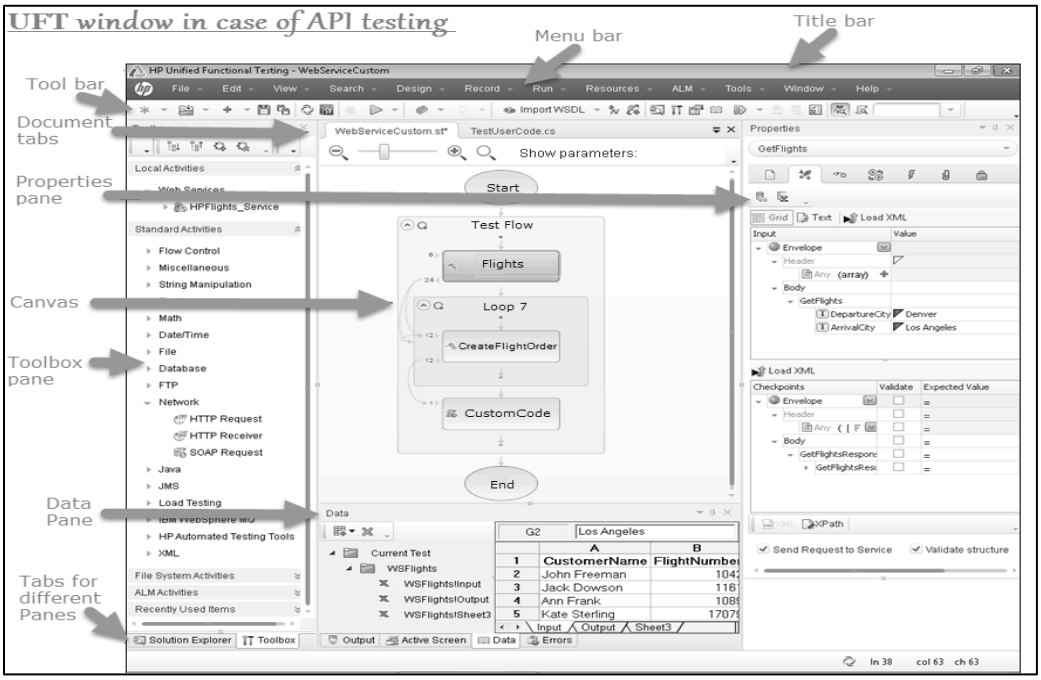


Figure 3-5 : Présentation UFT [38]

Le fonctionnement de l'outil UFT n'est pas très différent de celui de TestComplete (précédemment présenté), mais dispose de plus de fonctionnalité. Tous les deux sont utilisent la logique de vérification à l'aide de « Checkpoints ». L'éditeur du logiciel UFT étant l'entreprise HP, ceci mène à un produit de grande qualité et d'une grande fiabilité.

Un tutoriel bien détaillé et expliquant sommairement le fonctionnement de cet outil de test se trouve à cette référence [39].

3.2 Synthèse et choix d'outil pour le projet

Les quatre outils présélectionnés, et présentés ci-dessus ont été installés sur un poste local et testé. Une application web, de la même technologie que le logiciel de gestion de la paie a été utilisé pour effectuer des essais préliminaires afin de les évaluer.

3.2.1 Critères d'évaluation

L'analyse de chacun des outils est effectuée selon les critères suivants :

- Efforts d'automatisation: temps mis pour écrire les scripts de tests, complexité du langage utilisé;
- Effort d'apprentissage**:** niveau de compréhension de l'outil, de la syntaxe du code source, de l'exécution des tests, de l'environnement;
- Disponibilité des ressources: Support technique, service après-vente, accès aux informations sur l'outil, tutoriels, cours, forum, et, etc.;
- Performance: temps d'exécution, qualités des résultats après exécution des tests, tailles et types d'application pouvant être automatisée;
- Recommandés: conseillé par les partenaires, testeurs et autres internautes;
- Couts: Prix des outils.

3.2.2 Résumé de l'analyse comparative

Après essais d'utilisation et analyse de chacun de ces outils, le résumé à retenir est présenté dans le tableau ci-dessous tout en décrivant les avantages et inconvénients de chacun par rapport au contexte de la compagnie.

Tableau 3.1 : Résumé des analyses des outils de tests

Outils de Tests	Avantages	Inconvénients
TestComplete	- L'outil possède d'énormes fonctionnalités facilitantes bien de chose (ordonnancement d'exécution des tests, effectue les tests sur la base de données et, etc.). - Possibilité de gagner en temps d'automatisation. - Utilisable par un non-programmeur. - Léger et rapide. - Possibilité de définir les variables et les paramètres lors des tests.	- Ressource et support limités - Outil payant - Langage de programmation JBScript, Jscript, DelphiScrip - Complexité de compréhension de l'outil.
Selenium	- Ressource et support assez disponibles. - Utilisé par une très large communauté. - Langage de programmation C# - Logiciel libre et gratuit - Facilité t'intégration dans Visual studio. - Implémente une interface IWebDriver basée sur le JavaScript et facile à comprendre.	- Pas facile pour un non-programmeur. - Investissement en temps pour écriture et maintenance des scripts.

Tableau 3.2 : Résumé des analyses des outils de tests - Suite

Outils de Tests	**Avantages**	**Inconvénients**
UFT	- Fiabilité due au géant éditeur qui est HP - Possède plus de fonctionnalités que tout autre outil de tests. - Possibilité de gagner en temps d'automatisation. - Possibilité d'utilisation par un non-programmeur. Possibilité de définir les variables et les paramètres lors des tests.	- Ressources non disponibles - Outils très chers - Langage de programmation VBScript - Outil complexe Exécution lente
Coded UI	- Ressource et support assez disponibles - Langage de programmation C# ou VB, - Intégrer dans Visual studio - Très efficace - Possibilité d'exécution sur Microsoft test manager et Team Foundation Server - Dispose un nombre élevé de namespace et de classe. - Génération des tests de façon très organisés.	- Très lourd - Très lent - Génère une grande quantité de code source pour un simple cas. - Exécute les tests uniquement sur le navigateur Internet Explorer.

3.2.3 Choix de l'outil le mieux adapté

Le tableau suivant représente enfin une évaluation estimative des outils selon les critères initialement identifiés. Les cotes y sont également estimées et mentionnées pour chacun. Ceci est calculé sur une base de :

- Faible = 0
- Moyen = 10
- Élevé = 20

Tableau 3.3 : Évaluation des outils de tests

Outils d'automatisation	**Critères de choix**						**Cotes (sur 120)**
	Temps d'automatisation	Temps d'apprentissage	Disponibilité des ressources	Performances	Recommandés	Couts	
Coded UI	**Moyen**	**Élevé**	**Élevé**	**Élevé**	**Moyen**	Intègre dans Visual studio (Ultimate et premium) (**Moyen**)	**90**
Selenium	**Moyen**	**Élevé**	**Élevé**	**Moyen**	**Élevé**	Gratuit (**élevé**)	**100**
TestComplete	**Élevé**	**Moyen**	**Moyen**	**Élevé**	**Faible**	2 500 USD/ Licence (**Moyen**)	**70**
UFT	**Élevé**	**Moyen**	**Moyen**	**Élevé**	**Moyen**	À partir de 600 USD / 3mois (**faible**)	**70**

3.3 Conclusion

Après avoir fait des recherches sur les cadriciels et outils d'automatisations de tests existants et filtrés préliminairement 4 entre eux, ils ont été pratiqués afin de déterminer le niveau de complexité d'utilisation. Par la suite, leurs analyses ont été basées sur certains critères définis par la compagnie. Il en résulte donc que le mieux adapté pour le contexte de la compagnie est Selenium.

Dans la suite de ce projet, un prototype de robot de tests automatisé sera réalisé en s'inspirant de Selenium.

CHAPITRE 4

Réalisation d'un prototype logiciel d'un robot de tests automatisés

Le prototype logiciel du robot d'automatisation conçu lors de cette recherche se nomme « *Robot-Tests-NethrisEmployeurD* » [4]. Il aura comme premier objectif de détecter rapidement les défauts, en production, et de les rapporter au groupe d'assistance technique de la paie de CGI. Ainsi il sera plus facile de localiser rapidement les défauts et d'effectuer leur correction rapide.

Ce chapitre décrit tout d'abord les fonctionnalités attendues du prototype. Par la suite, il présente la synthèse des activités de réalisation c.-à-d. les technologies qui ont été utilisées et comment il a été conçu (c.-à-d. l'implémentation et les tests). Il décrit par la suite le processus de déploiement sur le serveur du prototype et, finalement, la synthèse de l'expérimentation c.-à-d. les caractéristiques, les avantages et les inconvénients discutés avec les intervenants.

4.1 Fonctionnalités attendues

Les fonctionnalités attendues du robot Robot-Tests-NethrisEmployeurD sont :

- Exécution automatique et périodique : Il s'exécute automatiquement après chaque 1 h de temps;
- Navigation dans les applications : Il ouvre automatiquement le fureteur et navigue dans les applications dans le but de vérifier le bon fonctionnement et de valider les logiques d'affaires;
- Capture-écran : Lorsqu'un comportement inattendu d'une page de l'application est perçu, il effectue une capture-écran de la page en question en enregistre en .png;
- Envoi des courriels : Il envoie deux types de courriel. Courriel d'échec lorsqu'un bogue est trouvé et courriel d'avertissement lorsqu'il est impossible de procéder un scénario de test pour diverse raison (c.-à-d. La base de données est vide);

- Génération des rapports de tests : un rapport au format .html est généré à la fin de chaque suite de test, décrivant l'état (succès, suspendu ou échec) de chaque scénario de tests ainsi que le temps mis pour l'exécuter;
- Ordonnancement des résultats de tests : sous la forme « Fonctionnalité –> Scénarios –> étapes ».

Les figures A-III-1 à A-III-2 de l'annexe III présente un exemple illustratif de comment le prototype fait fonctionne.

4.2 Synthèse des activités de réalisation du robot de tests

4.2.1 Nécessité de plus d'un outil

Étant donné toutes les fonctionnalités citées ci-dessus, l'outil Selenium (c.-à-d. Iwebdriver) à lui seul, ne sera pas suffisant pour la réalisation du robot de test, car, il ne permet pas, par exemple, la génération des rapports de tests, l'exécution automatique et la spécification fonctionnelle des tests. C'est donc la raison pour laquelle, il sera nécessaire d'utiliser d'autres outils complémentaires afin d'atteindre les objectifs de ce projet.

4.2.2 Liste des outils choisis pour le projet

Pour la conception du prototype du robot de test, les éléments suivants ont été utilisés :
Outils de Test :

- Selenium IDE: Facilitateur, enregistrement des scénarios utilisateurs et générations du code source en langage C#.
- Visual studio (toute version): Éditeur, Édition du code source et exécution des tests à travers MS Test (intégré a VS).

Cadriciel de Test :

- SpecFlow: TechTalk.SpecFlow.
- MS Test: Microsoft.VisualStudio.TestTools.UnitTesting.
- Selenium WebDriver: OpenQA.Selenium.

Langages utilisés:

- Gherkin: Pour les spécifications des scénarios de tests utilisateurs.
- Webdriver : Pour l'écriture du code source qui va interagir avec le navigateur.
- C#: Pour la complétude du code source Webdriver.
- XML: Pour les fichiers de configurations
- Batch Windows : Pour ordonner l'exécution du programme par le gestionnaire de tâche Windows.

En gros, le projet de test est constitué des fichiers « .features » dans lesquels sont écrit les fonctionnalités à tester, leurs scénarios découlant et les étapes de chaque scénario. Ceci en langage Gherkin. Également, chaque fichier « .features » possède son fichier « .cs » qui, dans lequel sont situé les méthodes « steps definition » de chaque étape. Ces méthodes seront donc complétées avec du code Selenium webdriver afin de favoriser la communication avec les navigateurs, et les vérifications « assert » sont faites grâce à l'outil Microsoft Test intégré dans Visual Studio. La figure 4-1 ci-dessous est une représentation de cette explication.

Figure 4-1 : Technologies utilisées [40]

4.2.3 Ressources humaines impliquées

Pour la réalisation du robot de test, un certain nombre d'experts ont dû être impliqués :

- Analystes fonctionnels : Collaborer avec les analystes en assurance qualité lors de la rédaction des cas de tests.
- Architectes : Collaborer avec les analystes en assurance qualité lors de la structuration des cas de tests.
- Analystes en assurance qualité : écriture des cas de tests en langage Gherkin, contrôler l'exécution et interpréter les résultats.
- Développeur des tests : Écriture des scripts de tests automatisés en langage de programmation.

4.2.4 Architecture du robot

La structure logicielle du robot est fondée sur la même structure proposée des « Hooks » de la technologie SpecFlow [41].

- Test : C'est la logique qui va s'exécuter avant n'importe quelle partie du programme de test et après l'exécution complète de tous les tests. Elle est constituée de deux attributs :
    1. [BeforeTestRun] : Ce blogue reçoit la logique qui initialisé toutes les variables et apprêter tous les éléments nécessaires à l'exécution de tous les tests.
    2. [AfterTestRun] : Ce blogue clôture le test. Il détruit tous les variables et éléments inutiles.

```
[Binding]
    class TestRun
    {
        public static string dates = DateTime.Now.ToString("dd-MMMM-yyyy hh-mm-ss");
        public static string path = "C:\\Robot de Tests\\";
        public static string TestResultspath = path+"TestResults\\";
        static MethodesComplementaire call = new MethodesComplementaire();
        public static bool anyTestFail = false;
        public static string listeTotal="";

        [BeforeTestRun]
        public static void Before()
        {
            call.TestSetup();

        }

        [AfterTestRun]
        public static void After()
        {
            call.TestTearDown();
        }
    }

}
```

Figure 4-2: Exemple de Hook TestRun de SpecFlow

- Features: C’est la logique du programme qui s’exécute avant et après chaque fonctionnalité du test. Elle est constituée de deux attributs:
    1. [BeforeFeature] : Ce blogue initialise toutes les variables avant et après qu'une fonctionnalité a besoin pour s’exécuter.
    2. [AfterFeature] :C’est le blogue de fermeture de chaque fonctionnalité.

```
[BeforeFeature]
public static void Before()
{
   featureFail = false;
   Scenarios.listeScenario = "";
}

[AfterFeature]
public static void After()
{
    if (featureFail == true)
    {
       TestRun.anyTestFail = true;
       listeFeature = listeFeature + "<b>+ Fonctionnalité: " +
FeatureContext.Current.FeatureInfo.Title.ToString() + ".</b> <br/>" +
Scenarios.listeScenario + "<br/><br/>";
     }
            Scenarios.driver.Quit();
}
```

Figure 4-3 : Exemple de Hook Feature de SpecFlow

- Scénarios : C'est la logique qui s'exécute avant et après chaque scénario d'une fonctionnalité. Elle est constituée de deux attributs :
  1. [BeforeScenario] : ce blogue initialise chaque scénario et prépare tous les éléments nécessaires à sa bonne exécution.
  2. [AfterScenario] : blogue qui s'exécute à la fin de chaque scénario afin de mettre jour la liste des erreurs rencontrée lors des tests.

```
[BeforeScenario]
public void BeforeScenario()
{
    errorpresent = false;
    stepsScenario = "";

    if (conexionerror == true)
    {
        ScenarioContext.Current.Pending();
    }
}

[AfterScenario]
public void AfterScenario()
{
     if (errorpresent == true)
     {
         Features.featureFail = true;
         listeScenario = listeScenario + "<b>- Scénario: " +
ScenarioContext.Current.ScenarioInfo.Title.ToString() + ".</b><br/>" +
stepsScenario;
      }
}
```

Figure 4-4 : Exemple de Hook Scenario de SpecFlow

- Steps: C'est la logique qui s'exécute avant et après chaque étape d'un scénario. Tout comme les autres, elle est constituée de deux attributs :
  1. [BeforeStep] :S'exécute avant que chaque étape d'un scénario s'exécute. Exemple, pour faire attendre 500 millisecondes avant l'exécution de chaque étape le code ci-dessous peut être utilisé.
  2. [AfterStep] : S'exécute après que chaque étape ai finis sont exécution.

```
[BeforeStep]
public static void Before()
{
    Thread.Sleep(500);
}
[AfterStep]
public static void After()
{
}
```

Figure 4-5 : Exemple de Hook Step de SpecFlow

L'architecture technique globale du programme peut donc être représentée comme ci-dessous.

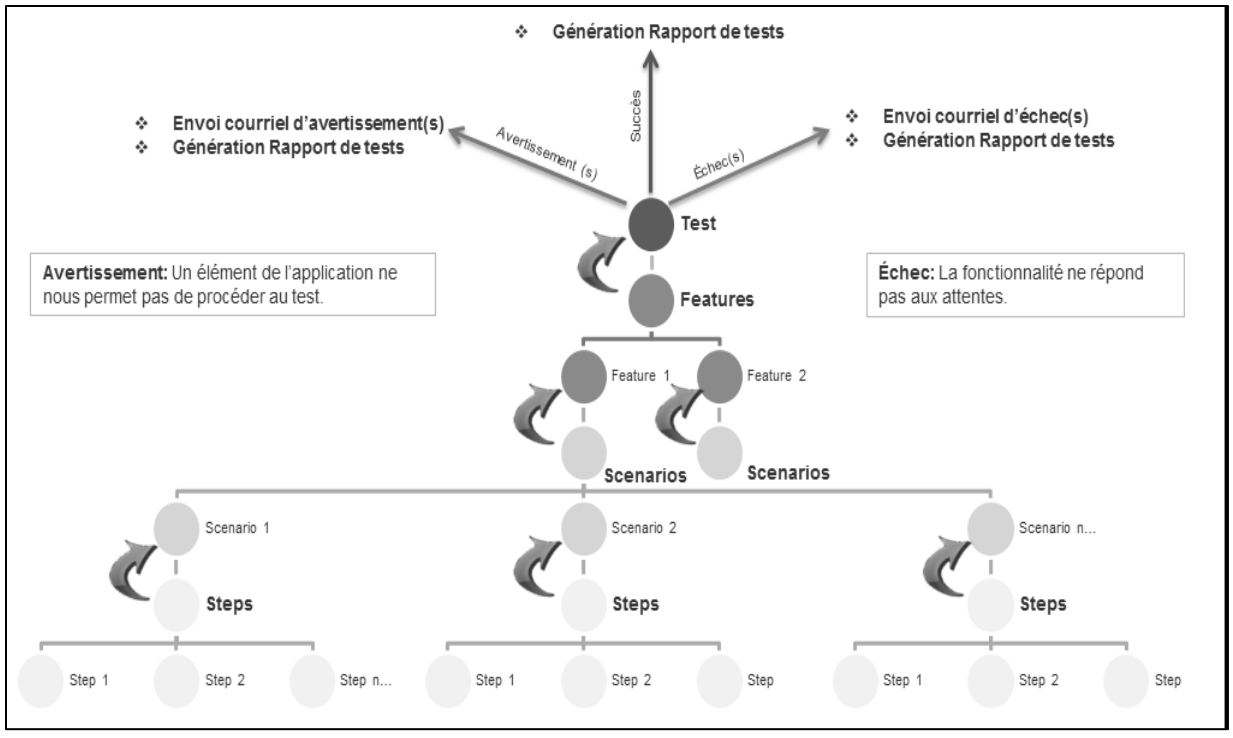


Figure 4-6 : Architecture technique de prototype de robot d'automatisation

4.2.5 Implémentation (procédure)

Maintenant que les outils utilisés sont précisés, l'architecture logicielle du prototype conçu, il ne reste plus qu'à implémenter les composants du robot. Cette phase décrit donc comment le prototype du robot a été mis en œuvre. Avant de débuter cette description, il faut d'abord préparer un environnement de développement pour le projet. La première étape de la préparation de l'environnement est d'installer Visual Studio 2013 premium. Par la suite, à l'aide du menu « outils » installé l'outil SpecFlow. Les figures A-IV-1 et A-IV-2 de l'annexe IV décrivent les étapes de cette installation.

Par la suite il est nécessaire d'aller dans le gestionnaire des packages « NuGet » de l'environnement afin d'installer les cadriciels nécessairent, soit Selenium Iwebdriver et SpecFlow. Les figures A-IV-3 à A-IV-5 de l'annexe IV disposent de plus d'images descriptives de cette installation.

La prochaine étape vise l'installation de Selenium IDE qui est tout simplement un « Plug-in » de Firefox et qui disponible gratuitement dans le site web officiel de Selenium. Il existe plusieurs sites web qui expliquent cette installation [42].

Suite à cette étape, il faut démarrer le « Selenium IDE » et activer le bouton « record » qui va permettre l'enregistrement des actions effectuées sur le navigateur. Effectuer tous les scénarios dont nous avons besoin d'automatiser (exemple : ouvrir la page des feuilles de temps, cliquer sur les liens « temps, soumission et génération des transactions de paie »). Les actions enregistrées par IDE vont être exportées dans Visual Studio en langage de programmation C#. Le code Selenium Webdriver généré ressemble donc à ceci :

```
Scenarios.driver.FindElement(By.LinkText("Temps")).Click();
Scenarios.driver.FindElement(By.LinkText("Soumission")).Click();
Scenarios.driver.FindElement(By.LinkText("Génération des transactions de paie")).Click();
```

Figure 4-7 : Code source généré par Selenium IDE

Une fois le code Selenium Webdriver générer, l'implémentation de chaque fonctionnalité de tests est constituée suit. Ceci est constitué de 3 principales étapes :

- Création d'une nouvelle fonctionnalité à tester : une fois qu'un nouveau projet de test a été créé sur VS, faire un clic droit sur ce projet et choisir ajouter un nouvel item, puis dans la liste, choisir « une nouvelle fonctionnalité SpecFlow ».

La figure A-IV-1 de l'annexe IV présente comment, ajouter ce fichier.

- La spécification fonctionnelle : Écriture des attentes fonctionnelles en langage Gherkin inspirer du langage courant humain avec des mots-clés « Feature, Scenario, Given, When et Then ».
    - Feature: Indique le nom de la fonctionnalité et ça description.
    - Scenario : Indique le nom du scénario
    - Given: Est le préalable pour tester un scénario (Entrée).
    - When: marque l'action à tester.
    - Then: C'est la postcondition qu'on veut obtenir après que l'action soit terminée avec succès (sortie).

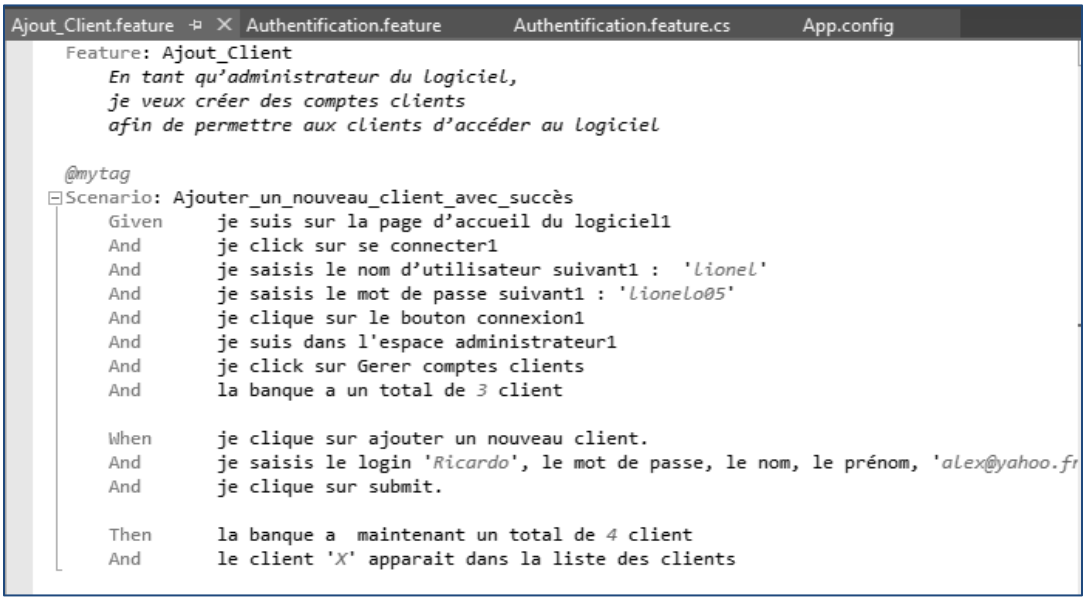


Figure 4-8 : Exemple de spécification fonctionnelle

- La génération des « steps definition » : Lorsqu'on fait un clic droit sur la spécification précédemment écrite en langage Gherkin dans Visual Studio et choisis « generate step definition », un fichier est généré et constitue des méthodes portant comme attributs les étapes du scénario précédemment écrit.

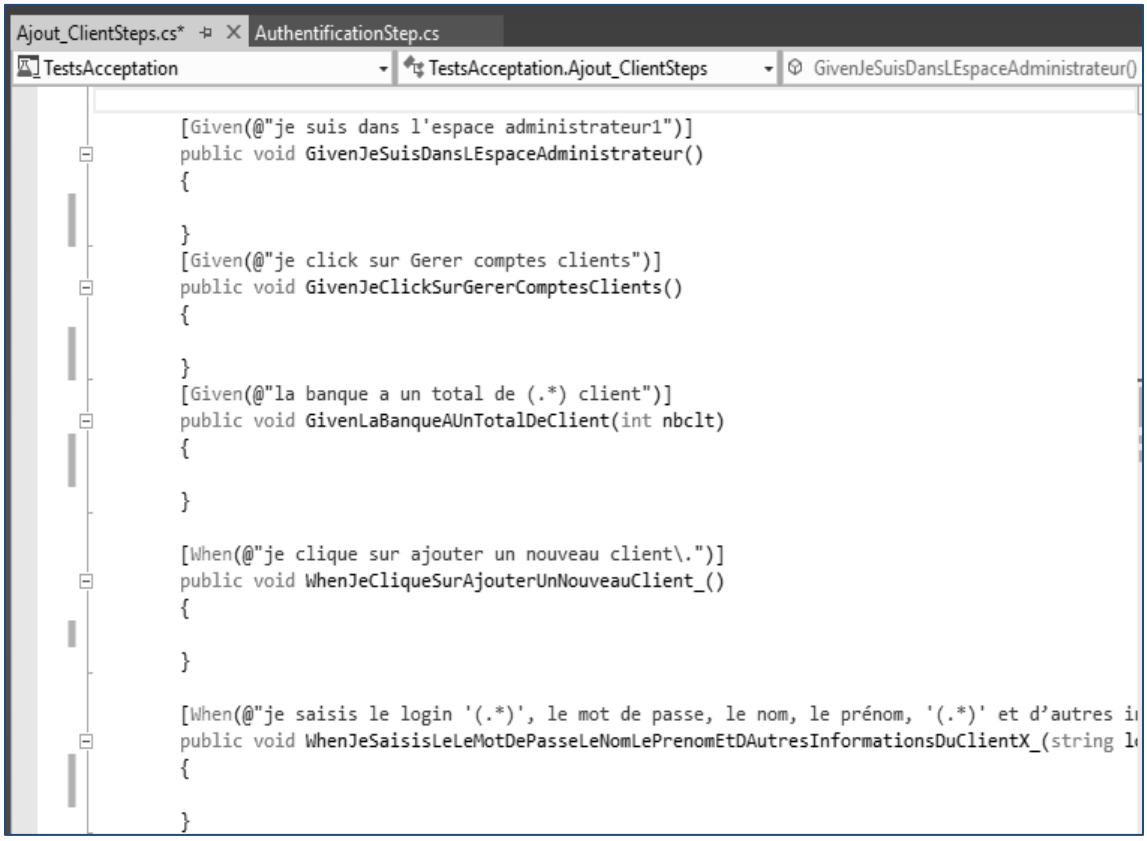


Figure 4-9 : Exemple des « steps definition » générés (vide)

- L'implémentation des « steps definition » : Une fois les méthodes vides ci-dessus générées, la prochaine et dernière étape est de compléter avec du code source Selenium Webdriver initialement générer, le contenu de chacune des méthodes « steps ».

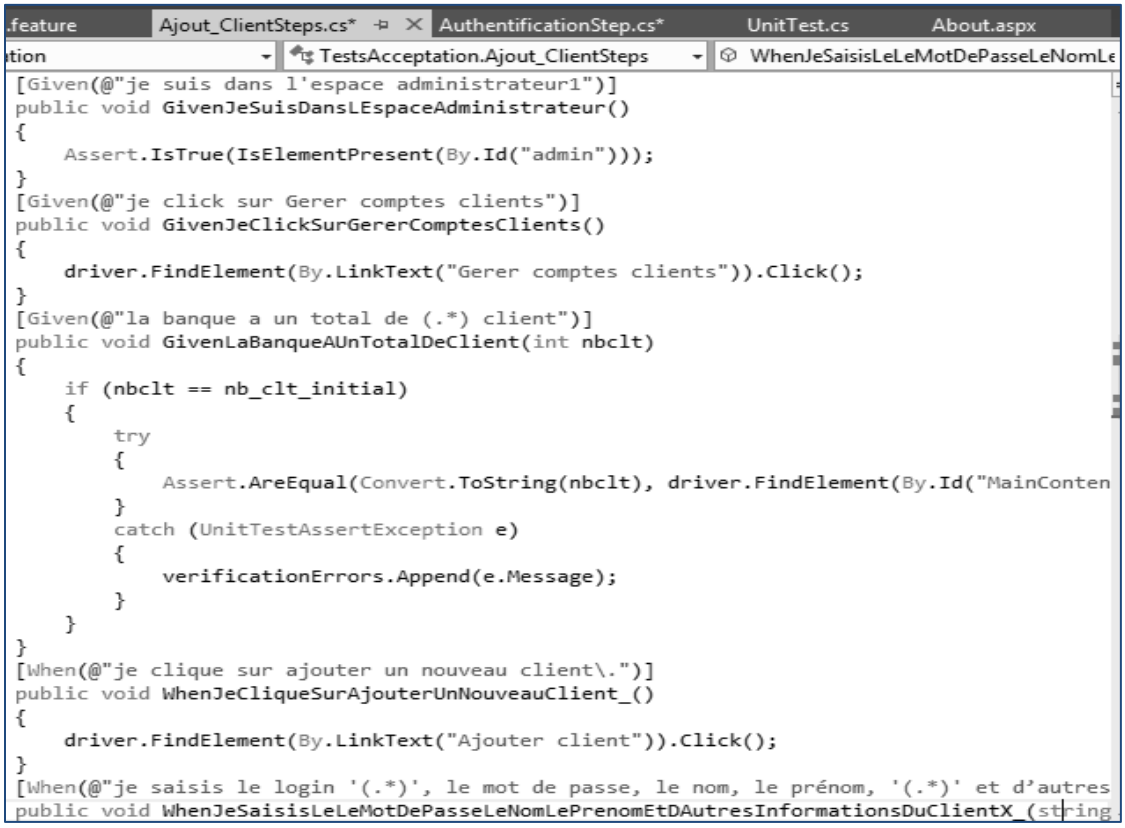


Figure 4-10 : Exemple « step definition » complété avec du code Webdriver

Ces trois étapes vont être répétées pour chacune des fonctionnalités à tester jusqu'à ce que tous les scripts déficits dans le plan de test soit couvert.

Cette implémentation se limite juste à la fonctionnalité de navigation dans les pages web et la vérification puis validation. Pour compléter les autres fonctionnalités telles que la capture d'écran en cas d'échec, l'envoi des courriels d'échecs et avertissements et, etc. une classe appelée méthode complémentaire est créée, et ces fonctionnalités vont être implémenté en langage C# et appelé par les « Hooks » lorsque nécessaires. Si une erreur est rencontrée dans une étape lors des tests, le Hook « step » va appeler la méthode qui enregistre l'erreur et capture la page contenant l'erreur. À la fin des tests, le Hook « test » dans son attribut « AfterTestRun » va appeler la méthode d'envoi des courriels.

L'exécution automatique et périodique est faite par le gestionnaire de tâche de Windows.

4.2.6 Tests

Dans cette phase, le prototype de robot de tests est activé et roule en local sur un poste de la compagnie. Ceci pendant une période de 1 semaine. Son comportement est contrôlé et les rapports qu'il générer sont interprétés et revérifier durant toute cette période afin de s'assurer qu'il fonctionne comme prévu.

4.3 Déploiement au serveur

4.3.1 Identification du serveur

Étant donné que ce prototype en mode exécutable a une taille totale de 70 Mo, il n'y a pas d'exigence en termes de caractéristique pour le serveur et ne consomme pas assez de mémoire. Le serveur qui a été identifié pour ce prototype et qui le supporte actuellement aux les caractéristiques suivantes :

- Système d'exploitation : Windows server 2008 R2
- La mémoire RAM : 1 Go
- Capacité du disque dur : 50 Go

4.3.2 Composants nécessaires pour le déploiement

Le déploiement sur le serveur nécessite les composants tels que :

- Le navigateur Firefox 33.1 : Car, il est compatible a Selenium Iwebdriver 2.2 avec lequel a été développé le robot.
- MS Test Agent 2013 : Étant donné que le robot est développé sur VS 2013 et serra compilé par MS Test, il est requis d'installer le MS Test Agent 2013.

4.3.3 Installation et configuration

La procédure d'installation du robot est la suivante :

1. Télécharger et installer Firefox 33.1
2. Télécharger et installer MS Test Agent 2013
3. Créer un dossier « Robot-Tests-NethrisEmployeurD » sur la racine C : du serveur
4. Créer dans le dossier précédent deux sous-dossiers « Pakages » et « Trx »
5. Placer également le fichier « Robot-Tests-NethrisEmployeurD V0.bat » dans le dossier précédent, et créer un raccourci vers le bureau.
6. Après avoir compilé en mode release le programme du robot en local, aller dans le dossier Release et copier tout le contenu, puis coller dans le dossier « Pakages » du serveur.

Et le prototype du robot est installé avec succès. Il ne reste plus qu'à programme sa planification d'exécution.

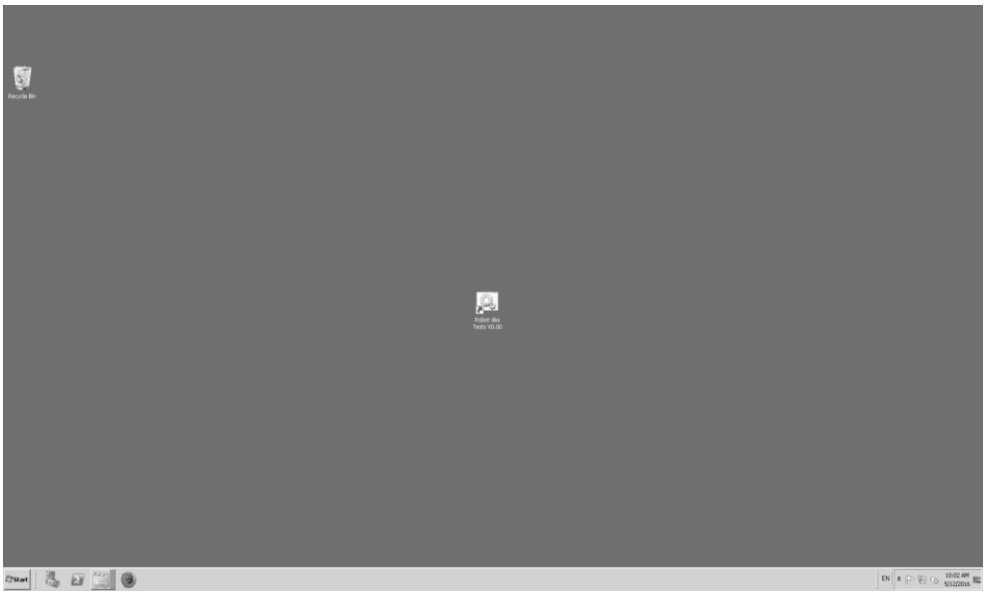


Figure 4-11 : Prototype de robot de tests installé au serveur

7. Ouvrir le planificateur de tache Windows et programmer l'exécution pour chaque 1h de temps
8. Activer la tache

Le tout est joué, le robot commencera son exécution dans 1heure, et le répètera chaque heure pour toujours.

4.4 Synthèse de cette réalisation

4.4.1 Caractéristiques du prototype

Ce prototype de robot des tests automatisés a diverses caractéristiques parmi lesquels, les principaux :

- Répétabilité : les tests sont répétable autant et autant de fois possible et nécessaire.
- Couverture : Le niveau de couverture des tests automatisés pour chaque cas est raisonnable
- Fiabilité : fournir des résultats vrai et fiable. Offre également la possibilité de retracer les étapes d'exécution et de voir les résultats bien détaillés et clairs pour chaque cas de tests.
- Réutilisabilité : Il est modulaire et réutilisable. Le script de chaque cas de test est entièrement indépendant des autres cas.
- Portabilité : Les scripts de tests sont facilement transportés et exécutables sur tout autre type de plateforme.
- Performance : Il est robuste et a un temps de réponse très rapide.

4.4.2 Avantage du prototype

Le prototype de robot identifié possède les avantages suivant, au niveau de développement, le niveau de l'assurance qualité et le niveau de l'application.

Niveau du développement:

- Cohérence de développements
- Facilite la localisation des erreurs
- Réduction de la majeure partie des erreurs de développements.
- Facilite la maintenance

Niveau du service d'assurance qualité :

- Réduction des couts.
- Réduction des temps d'exécutions des tests.
- Augmentation de la capacité des tests.

Niveau de l'application :

- Diminuer les bogues en production
- Favoriser le refactoring et l'optimisation
- Améliorer la qualité.

4.4.3 Inconvenant du prototype

Bien que les tests automatisés soient avantageux, ils disposent aussi de quelques ambigüités telles que :

- Apprentissage d'utilisation des outils et cadriciels de tests (Investissement en temps).
- Investissement en temps pour la conception et le développement des cas de tests.

- Maintenance des scripts de tests car, ils évoluent avec le SUT.

4.5 Conclusion

Ce chapitre présente en détail les phases de réalisation du prototype de robot d'automatisation des tests à savoir, l'analyse, la conception, l'implémentation, le test et le déploiement au serveur.

Pour atteindre les objectifs de cette réalisation, le cadriciel « Selenium » choisi à lui seul était insuffisant. D'autres éléments complémentaires ont été utilisés afin de le renforcer.

Le chapitre qui suit est une synthèse de ce projet de synthèse, constitué d'un rappel des taches réalisées, difficultés rencontrées, solutions de contournement et les choses apprises.

CHAPITRE 5

Synthèses du projet

Ce chapitre qui est le dernier est, en bref un résume, du projet. Il cite les tâches qui ont été réalisées, les difficultés rencontrées lors du projet et les solutions de contournement adoptés. Il décrit aussi les choses apprises telles que l'expérience professionnelle, expériences humaines et autres bénéfices.

5.1 Taches réalisées

Comme taches réalisées, les deux objectifs principaux du projet ont été atteints avec succès, et la compagnie a été satisfaite. Plus en détail, ce qui a été réalisé est :

1) l'analyse des outils et cadriciels de tests automatisés existants et choix du mieux adaptés : un premier filtrage a été effectué selon certains critères, et quatre outils ont été présélectionnés à savoir les outils UFT, TestComplete, Code UI et Selenium. Par la suite, l'installation de ces outils sur un poste de la compagnie et pratique a été faite. Et enfin, une analyse et évaluation selon des critères définis par la compagnie de chacun des quatre outils a permis de choisir l'outil Selenium comme étant le mieux adapté pour le contexte.
2) La réalisation d'un prototype de robot de tests automatisés : après la définition des fonctionnalités attendues de ce prototype par la compagnie, il a été impossible de les réaliser uniquement avec l'outil Selenium, d'où d'autres outils complémentaires tels que SpecFlow et MS Test, gestionnaire de tâche Windows vont intervenir de façon complémentaire afin de permettre à Selenium de satisfaire à toutes les fonctions initialement requises.

5.2 Difficultés rencontrées

Plusieurs difficultés ont été rencontrées lors de ce projet :

1) Difficultés techniques :
   - Pas évident de compréhension des outils TestComplete et UFT : IDE peut complexes, ne disposes pas assez de ressources (tutoriels, forums, groupes et, etc.) sur internet.
   - Outils TestComplete et UFT sont payants et très chers, la compagnie n'était pas prête à dépenser de l'argent pour les acheter, juste pour les évaluer. Leurs versions d'évaluation sont incomplètes.
2) Difficultés logiques :
   - Il a été difficile de comprendre la logique de l'application sous test : pour tester, il faut d'abord comprendre ce qui a été demandé (spécification fonctionnelle) ensuite, ce qui a été réalisé (SUT).
   - Le prototype de robot utilise le même code de connexion qu'un testeur manuel de la compagnie : ceci va entrainer l'échec de tests chaque fois que le robot essaie de se connecter quand ce testeur manuel est connecteur, dû à une redondance de connexion.

5.3 Solutions de contournement adoptées

Pour les difficultés techniques, la solution était d'entrer en contact direct avec les éditeurs des deux outils payants et de leur faire part du projet et des difficultés. D'où ils ont accordé à la compagnie, la grâce d'utiliser la version payante pour une durée de quinze jours avant de procéder à l'évaluation des outils.

En ce qui concerne les difficultés logiques, pour la première, des séances de formations vont être programmées afin de présenter les spécifications fonctionnelles initialement définies de

l’application et aussi d’expliquer ce qui a été réalisé. Et pour la deuxième difficulté, des codes d’accès personnels aux prototypes de robot vont être créés.

5.4 Leçons apprises

Premièrement, le fait de travailler avec une telle grande compagnie était une énorme richesse au niveau de la gestion des relations humaines.
Le fait de planifier un projet, et de l’exécuter du début jusqu'à la fin tout en contournant des difficultés et des risques a permis d’acquérir plus d’expérience en gestion de projet informatique.
L’analyse de plusieurs outils d’automatisation des tests logiciels a permis de beaucoup apprendre et être capable de facilement recommander un outil sans doute.
La réalisation du prototype de robot avec succès a permis d’acquérir d’énorme expérience technique. Surtout lors de la combinaison des différents outils ensemble dans le but de satisfaire toutes les fonctions requises par la compagnie.
Sans oublier aussi l’explication des spécifiques fonctionnels du produit logiciel attendu et la formation sur l’application qui a permis d’enrichir les connaissances dans le domaine de la paie et de gestion des ressources humaines.

5.5 Conclusion

Ce chapitre avait pour objectif de présenter une synthèse de ce qui a été fait dans ce projet, les difficultés qui ont été rencontrées et les solutions qui ont été adoptées face à ces difficultés. Il décrivait aussi les choses apprises lors de la réalisation du projet.
La section qui suit une conclusion du projet.

CONCLUSION

C’est dans le but de satisfaire à une des exigences de réussite au programme de maitrise en génie logiciel à l’École de technologie supérieure, dont ce projet qui a été réalisé. Dirigé par le professeur Alain April, il était question ici premièrement d’analyser les différents cadriciels et outils d’automatisation de tests logiciels existant sous forme de logiciels libres, gratuits ou commercialisés. Et deuxièmement de réaliser un prototype de robot de tests automatisés à partir de ce cadriciel ou outils choisis.
Le fait de planifier un projet, de le concevoir, de l’exécuter et d’obtenir une sortie qui marque le succès total a été une meilleure façon de mettre en pratique la synthèse des enseignements obtenus dans la formation de maitrise en génie logiciel et une source d’acquisition des expériences professionnelles.

En ce qui concerne le projet réalisé, ce qui reste à retenir est que, l'automatisation des tests logiciels permet de faciliter certaines tâches répétitives, mais nécessaires dans un processus de test formalisé déjà en place, ou d'effectuer des tests supplémentaires qui seraient difficiles à faire manuellement. L'automatisation des tests est essentielle pour la livraison continue et le contrôle continu.

RECOMMANDATIONS

Le domaine de l'automatisation des tests évolue de plus en plus, et des nouvelles idées, des façons de s'en servir des tests d'automatisés apparaissent également. La compagnie utilise les tests automatisés sous forme comme d'un robot hébergé sur un serveur et qui va s'exécuter périodiquement pour vérifier les applications. Aujourd'hui, l'intégration prend aussi de l'ampleur et les serveurs d'intégration continue contiennent des outils d'exécution des tests tels que Team Foundation Build dans le serveur Team Foundation Server.

Il serait une bonne idée si la compagnie intégrée les tests automatisés dans le serveur d'intégration continuent de telle sorte qu'ils peuvent vérifier le fonctionnement et les régressions dans l'application avant le déploiement en production.

Un projet de tests automatisés comporte des scripts tout comme un projet de développements logiciels. Ce qui nécessite la maintenance des scripts, et la gestion des versions.
L'utilisation d'un outil de gestion de version pour l'écriture des scripts de tests automatisés pourrait éviter beaucoup de dégât futur et épargner des regrets dans la compagnie.

ANNEXE I

Création d'un projet de test avec l'outil de test Coded UI et génération de code source après enregistrement.

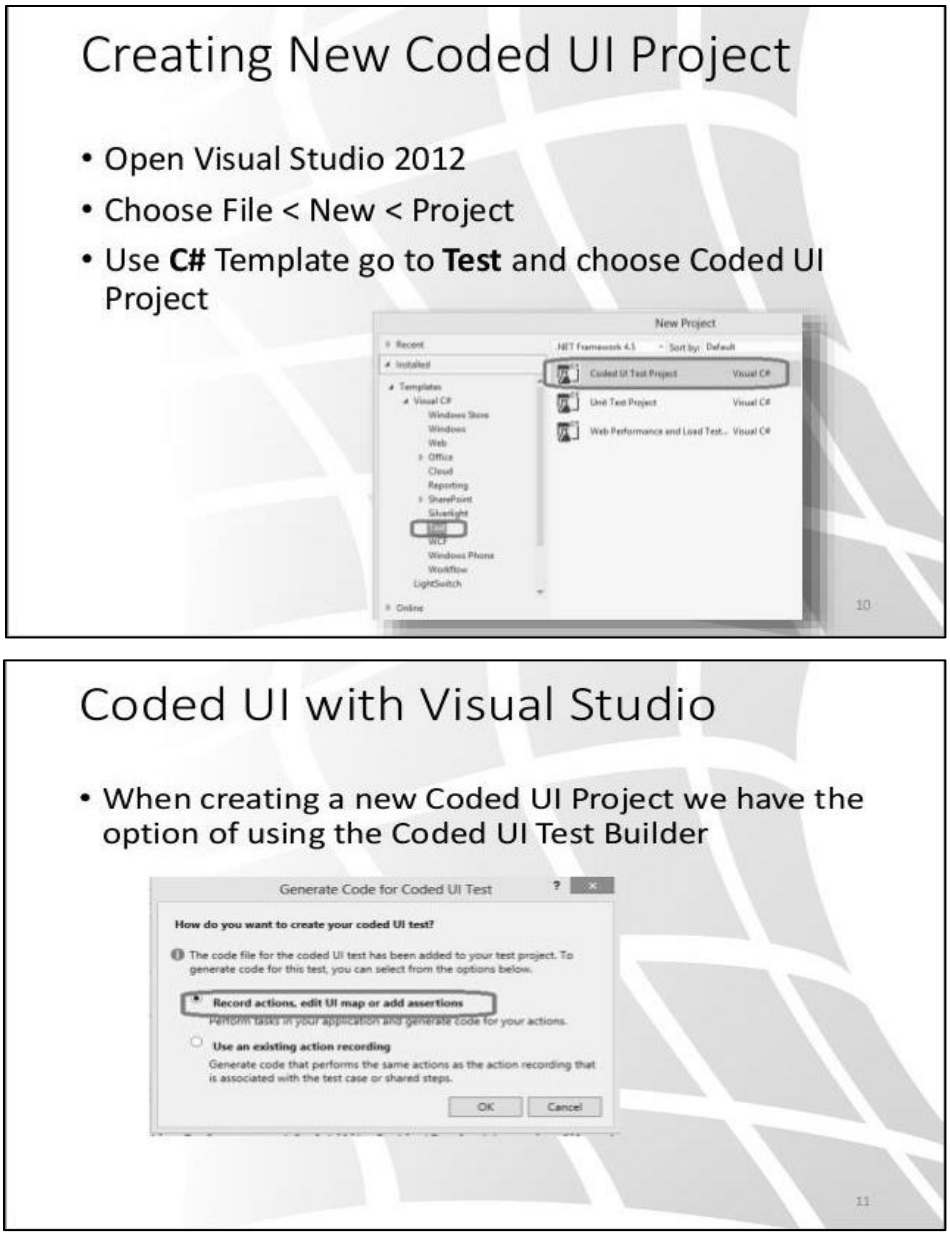


Figure-A I-1 : Création d'un nouveau projet de test avec Coded UI [43]

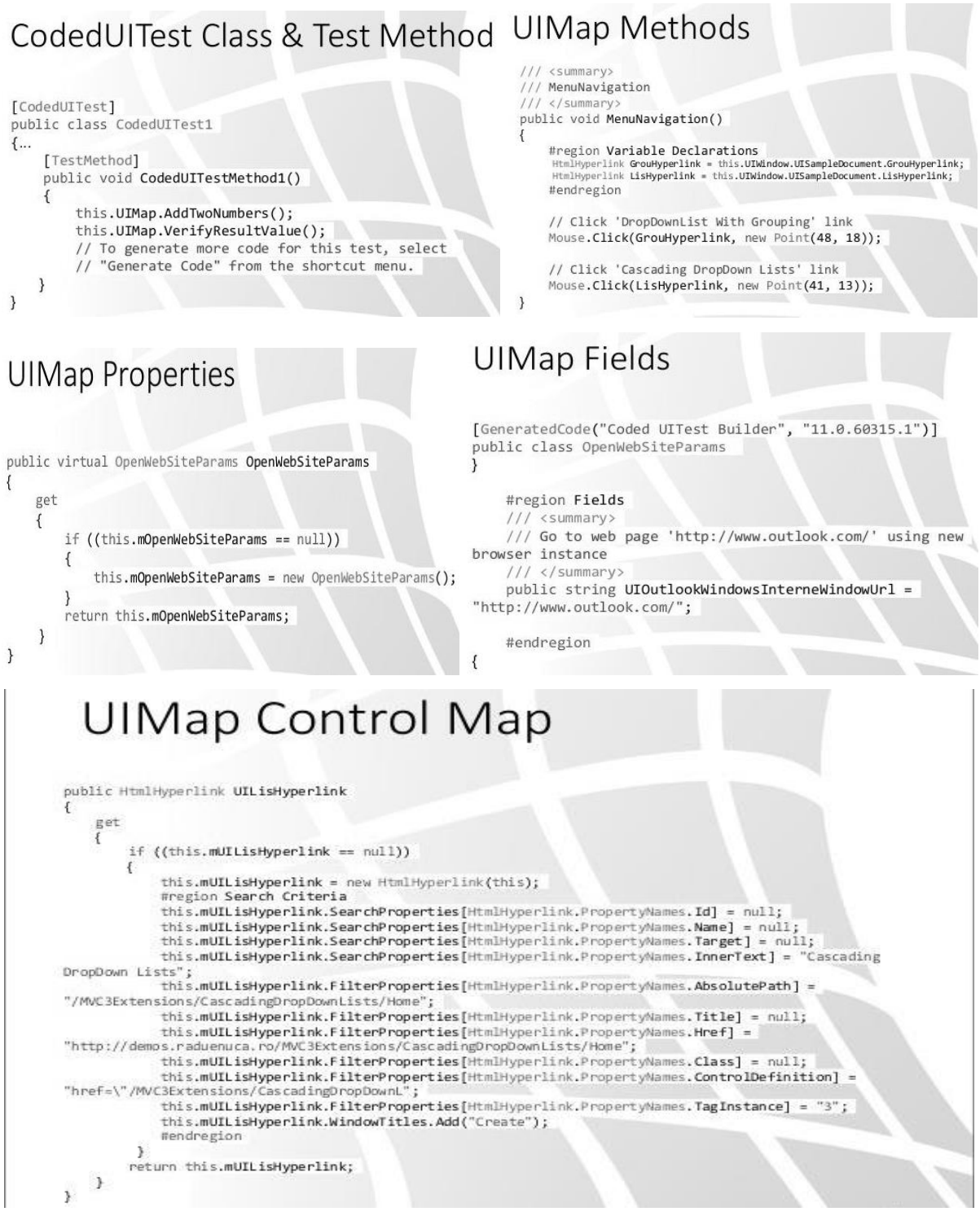


Figure -A I-2 : Code source généré après enregistrement de l'action de navigation sur la page www.outlook.com [43]

ANNEXE II

Présentation de la création d'un projet, des tests et du résultat avec TestComplete.

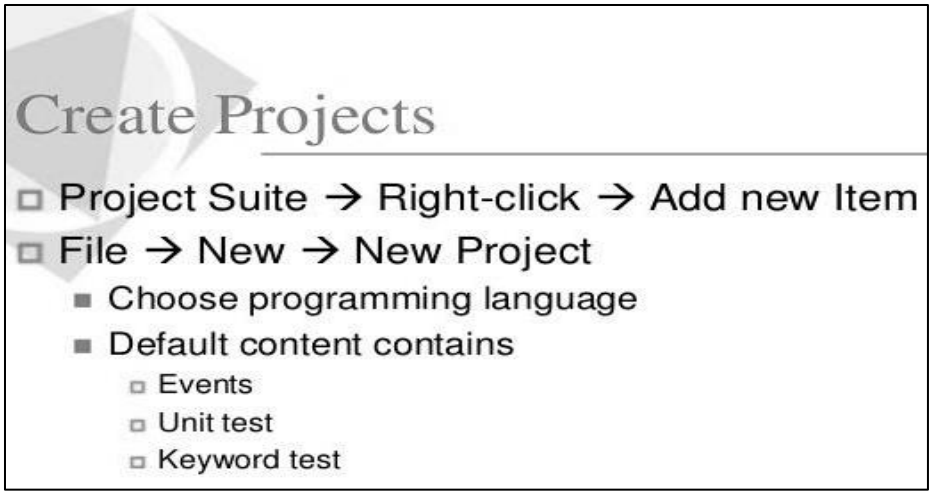


Figure -A II-1 : Création d'un nouveau projet avec TestComplete [44]

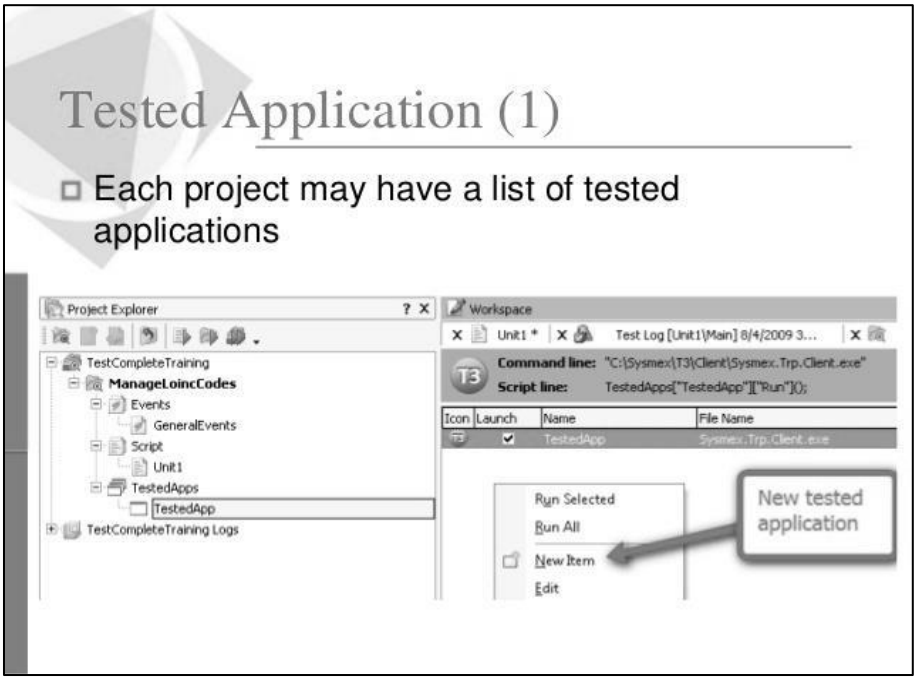


Figure -A II-2 : Création d'un nouvel item pour l'application sous test [44]

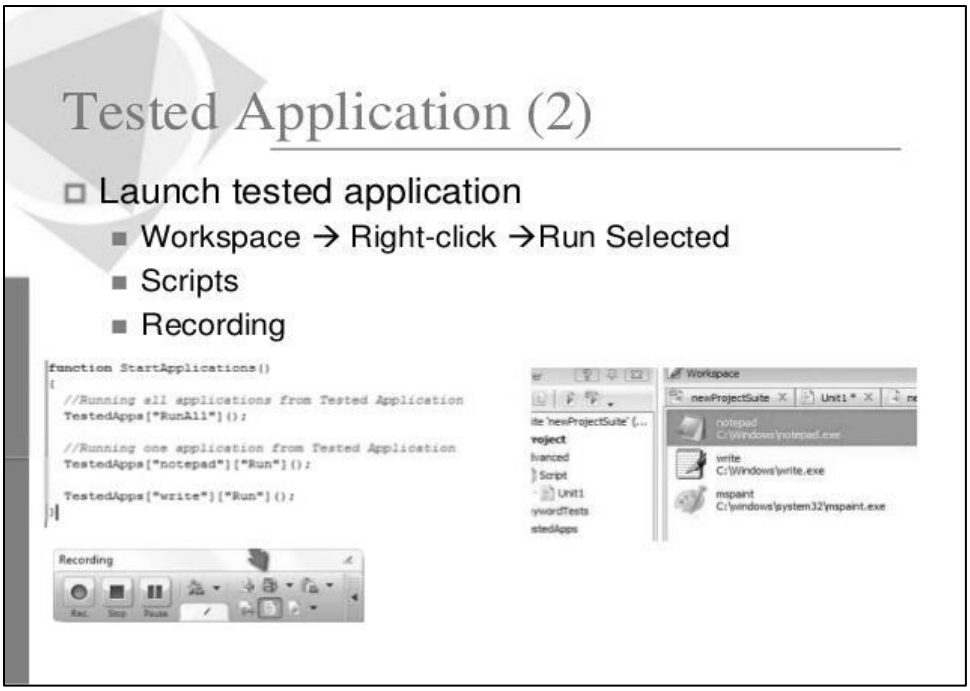


Figure -A II-3 : déclenché l'enregistrement des actions sur le navigateur (génération des scripts simultanément dans l'IDE) [44]

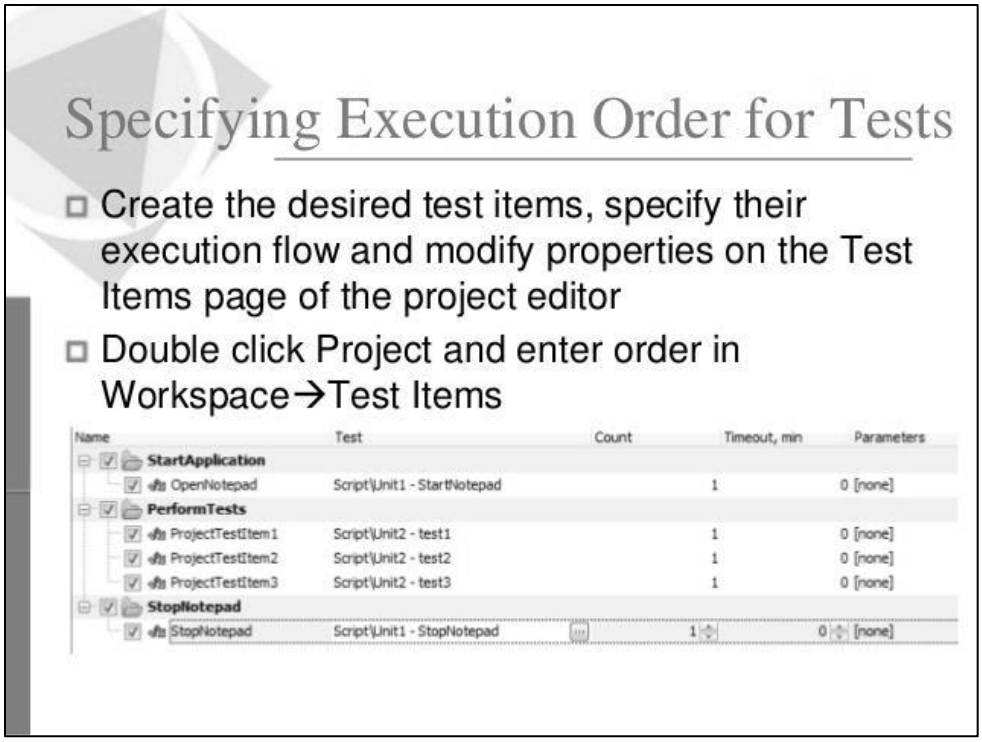


Figure -A II-4 : Spécification de l'ordre d'exécution des tests [44].

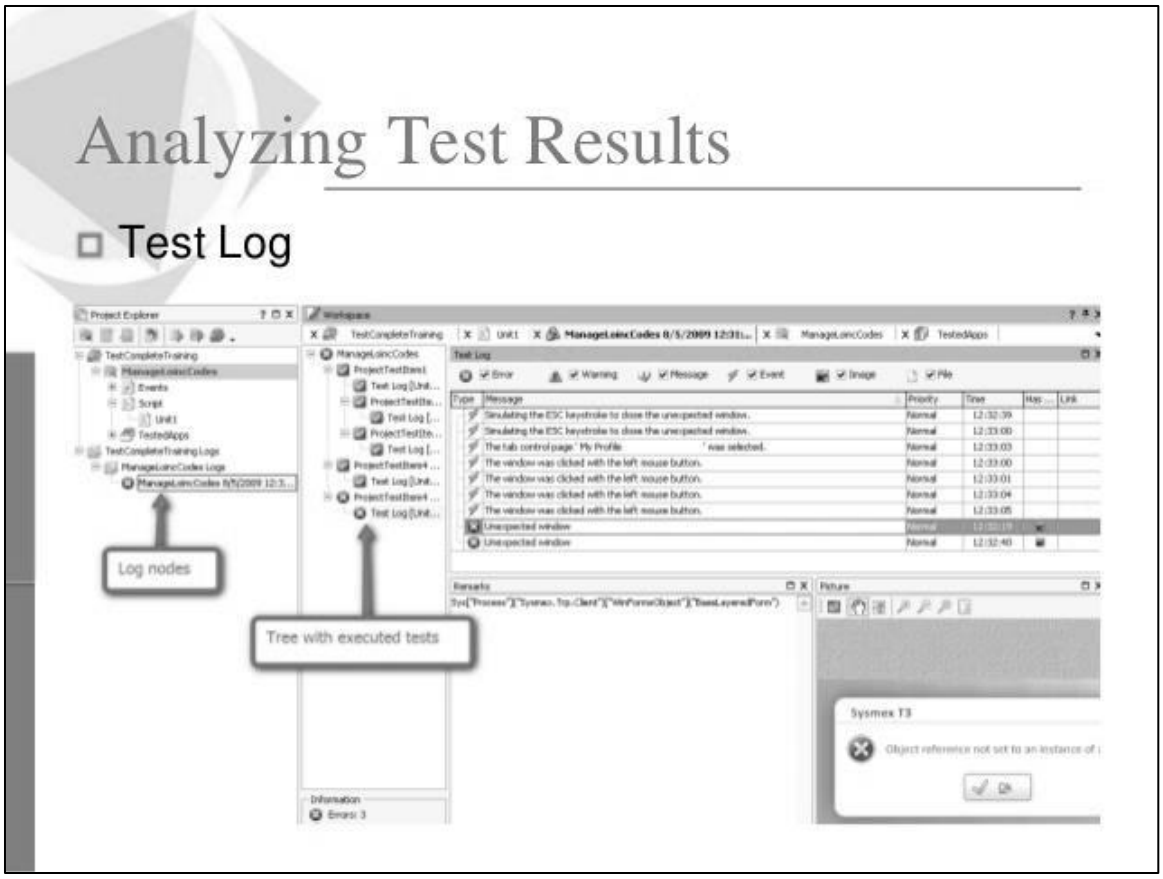


Figure -A II-5 : Résultats des tests après exécution [44]

ANNEXE III

Fonctionnalités du prototype robot de tests automatisé.

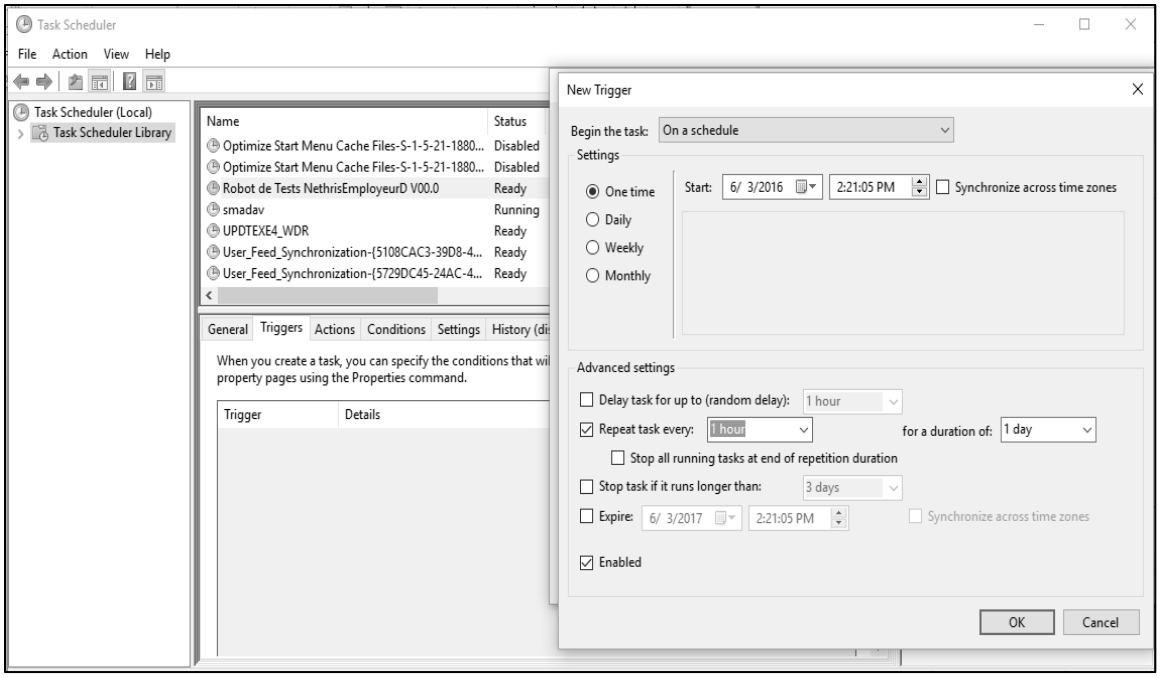


Figure -A III-1 : Exécution automatique et périodique avec le gestionnaire des tâches

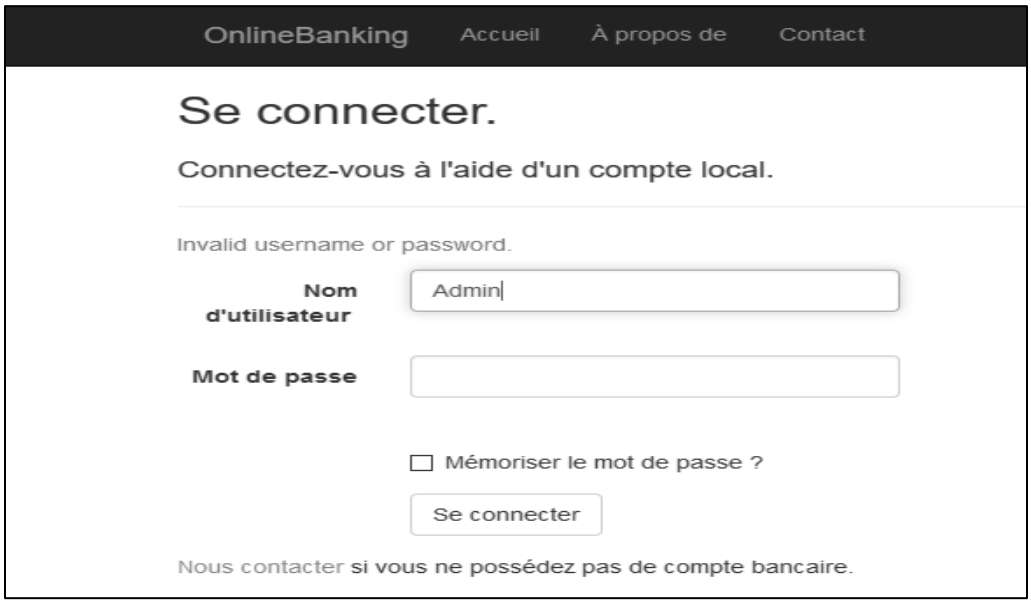


Figure -A III-2 : Navigation, vérification et validation

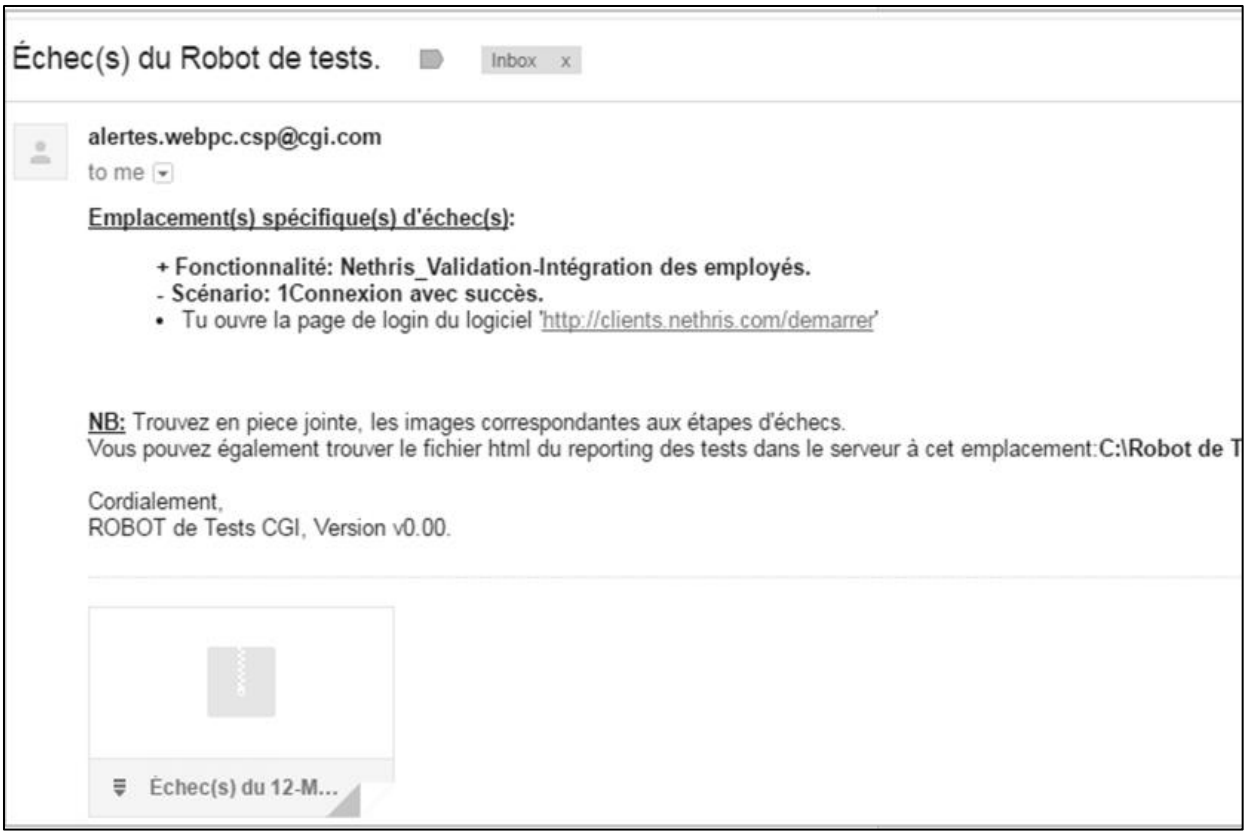
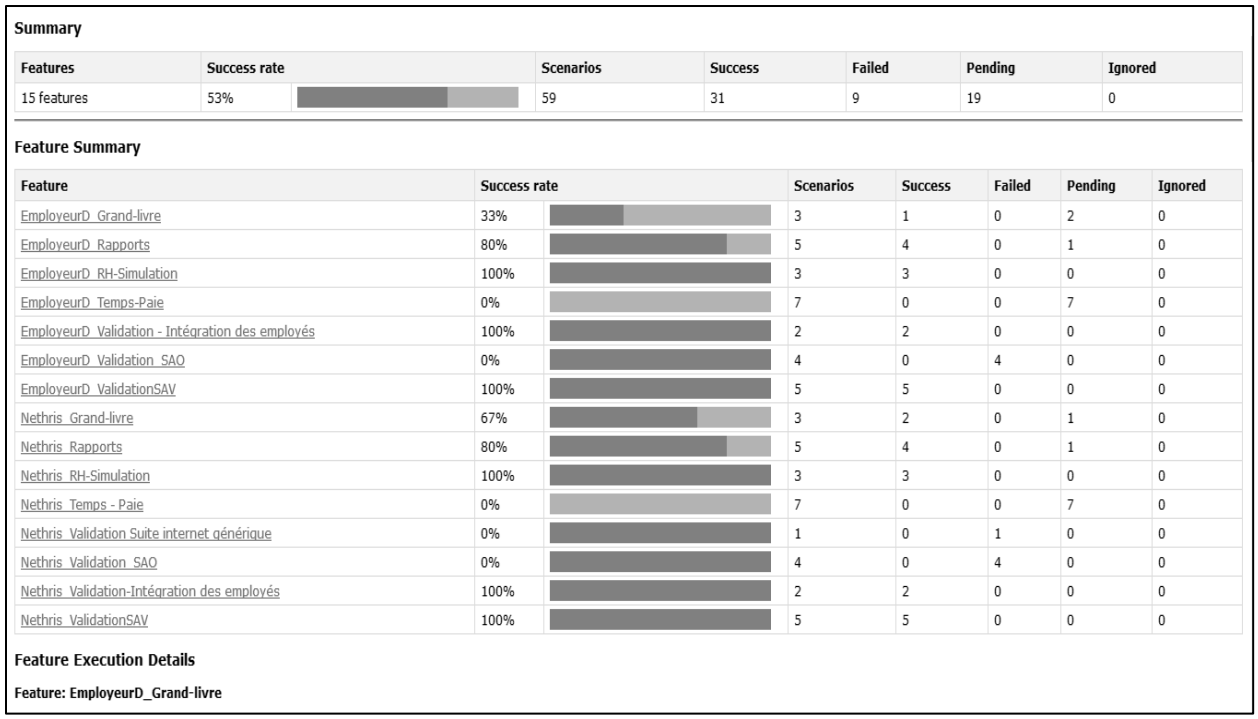


Figure -A III-3 : Courriel d'échec de test

Figure -A III-4 : Courriel d'avertissement de tests



Summary

Features	Success rate	Scenarios	Success	Failed	Pending	Ignored
15 features	53%	59	31	9	19	0

Feature Summary

Feature	Success rate	Scenarios	Success	Failed	Pending	Ignored
EmployeurD_Grand-livre	33%	3	1	0	2	0
EmployeurD_Rapports	80%	5	4	0	1	0
EmployeurD_RH-Simulation	100%	3	3	0	0	0
EmployeurD_Temps-Paie	0%	7	0	0	7	0
EmployeurD_Validation - Intégration des employés	100%	2	2	0	0	0
EmployeurD_Validation_SAQ	0%	4	0	4	0	0
EmployeurD_ValidationSAV	100%	5	5	0	0	0
Nethris_Grand-livre	67%	3	2	0	1	0
Nethris_Rapports	80%	5	4	0	1	0
Nethris_RH-Simulation	100%	3	3	0	0	0
Nethris_Temps - Paie	0%	7	0	0	7	0
Nethris_Validation Suite internet générique	0%	1	0	1	0	0
Nethris_Validation_SAQ	0%	4	0	4	0	0
Nethris_Validation-Intégration des employés	100%	2	2	0	0	0
Nethris_ValidationSAV	100%	5	5	0	0	0

Feature Execution Details

Feature: EmployeurD_Grand-livre

Figure -A III-5 : Rapport sommaire de toutes les fonctionnalités du testé

Feature: EmployeurD_Grand-livre

Scenario	Status	Time(s)
2Demandes d'arrêts de paiement [show]	pending	3.586
3Traitements des données de paie [show]	pending	4.017
1Connexion avec succes - Nethris [show]	success	37.616

Feature: EmployeurD_Rapports

Scenario	Status	Time(s)
1Connexion avec succes - Nethris [show]	success	21.797
5Suppression du plus vieux rappoort après 50 produits [show]	pending	2.933
3Générateur de rapport standard [show]	success	44.556
4Générateur de rapport gain déduction [show]	success	43.588
2Générateur de rapport à disponibilité publique [show]	success	39.217

Feature: EmployeurD_RH-Simulation

Scenario	Status	Time(s)
1Connexion avec succes - Nethris [show]	success	19.875
3Simuler une paie [show]	success	41.54
2Modification au dossier employé [show]	success	23.388

Feature: EmployeurD_Temps-Paie

Scenario	Status	Time(s)
3Feuille de temps - Soumission	pending	NaN
4Générer les transactions de paie	pending	NaN
7Validation du retour de paie dans Nethris	pending	NaN
1Connexion avec succes - Nethris	pending	NaN
5Demander un prétraitement	pending	NaN

Figure -A III-6 : Rapport détaillé des scénarios d'une fonctionnalité spécifique

ANNEXE IV

Installation de l'outil SpecFlow.

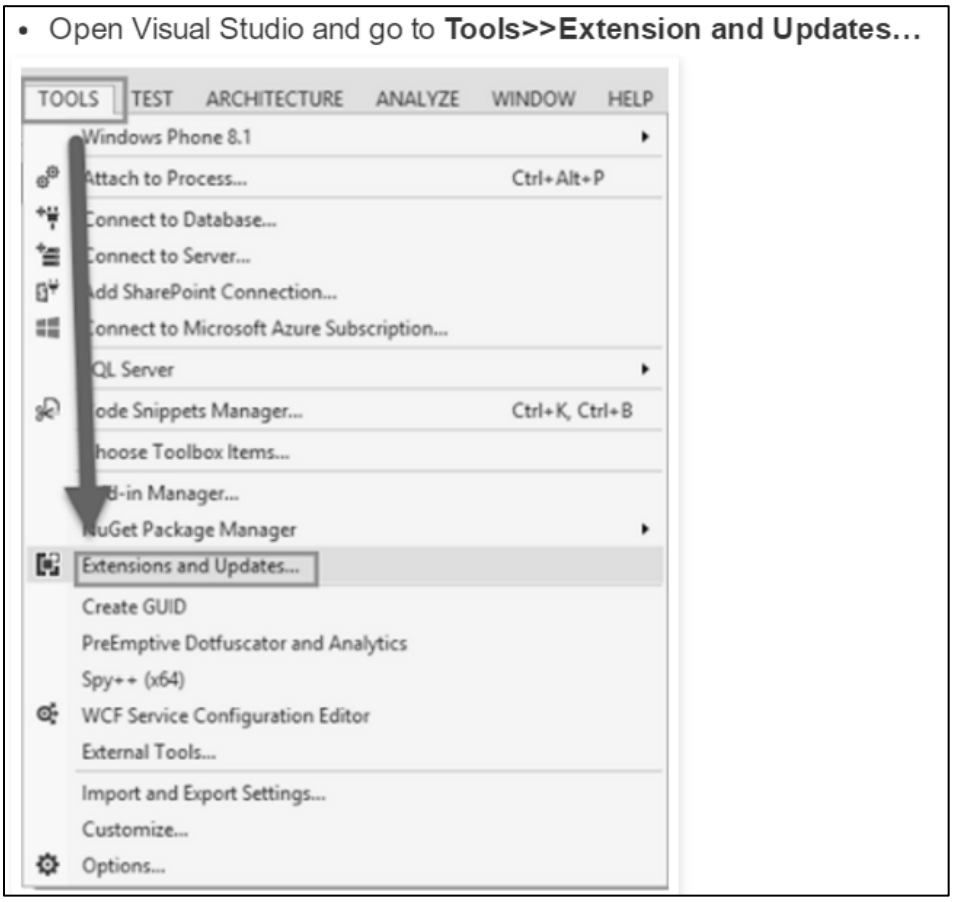


Figure -A IV-1 : Installation de SpecFlow, étape 1 [45]

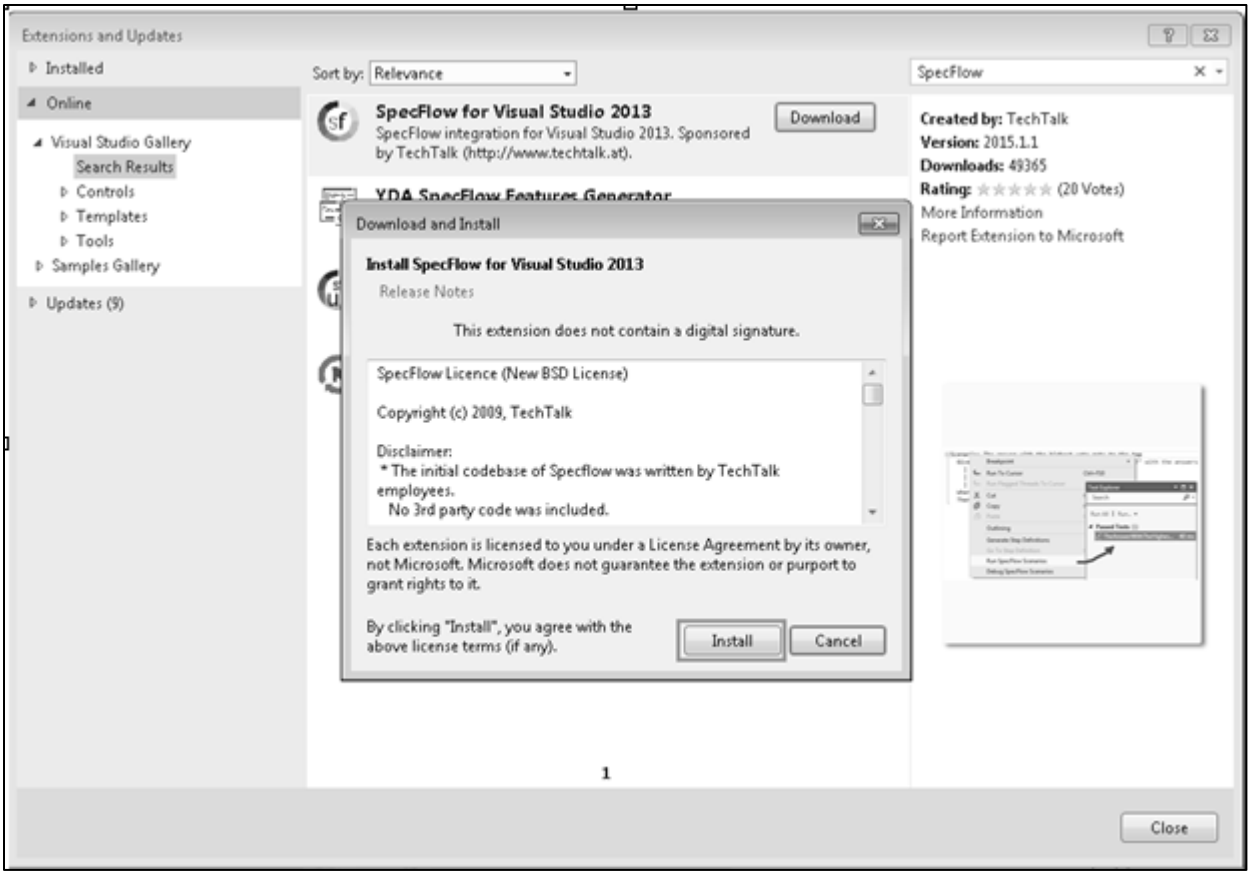


Figure -A IV-2 : Installation de SpecFlow, étape 2 [45]

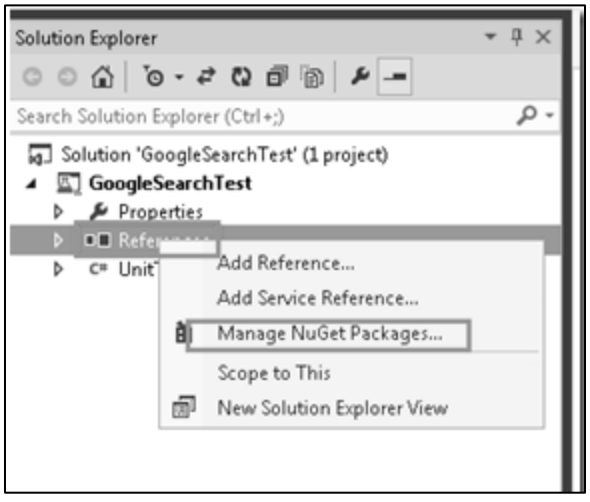


Figure -A IV-3 : Installation des packages NuGet sous VS 2013 [45]

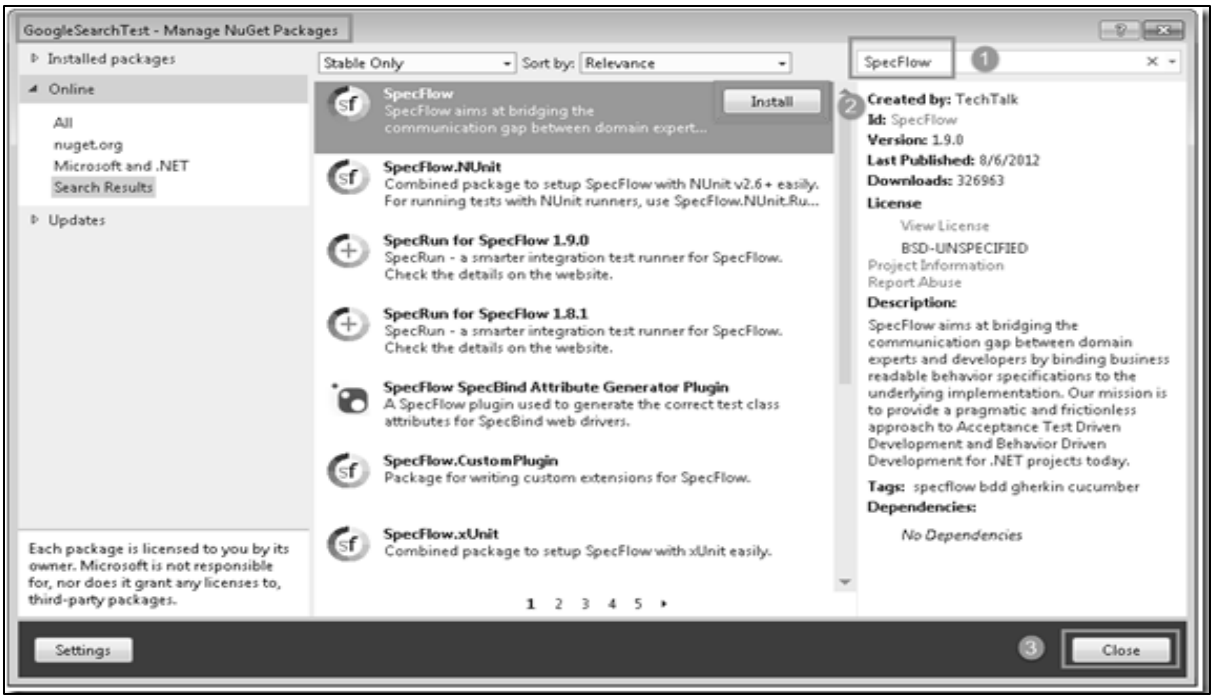


Figure -A IV-4 : Installation du cadriciel SPecFlow [45]

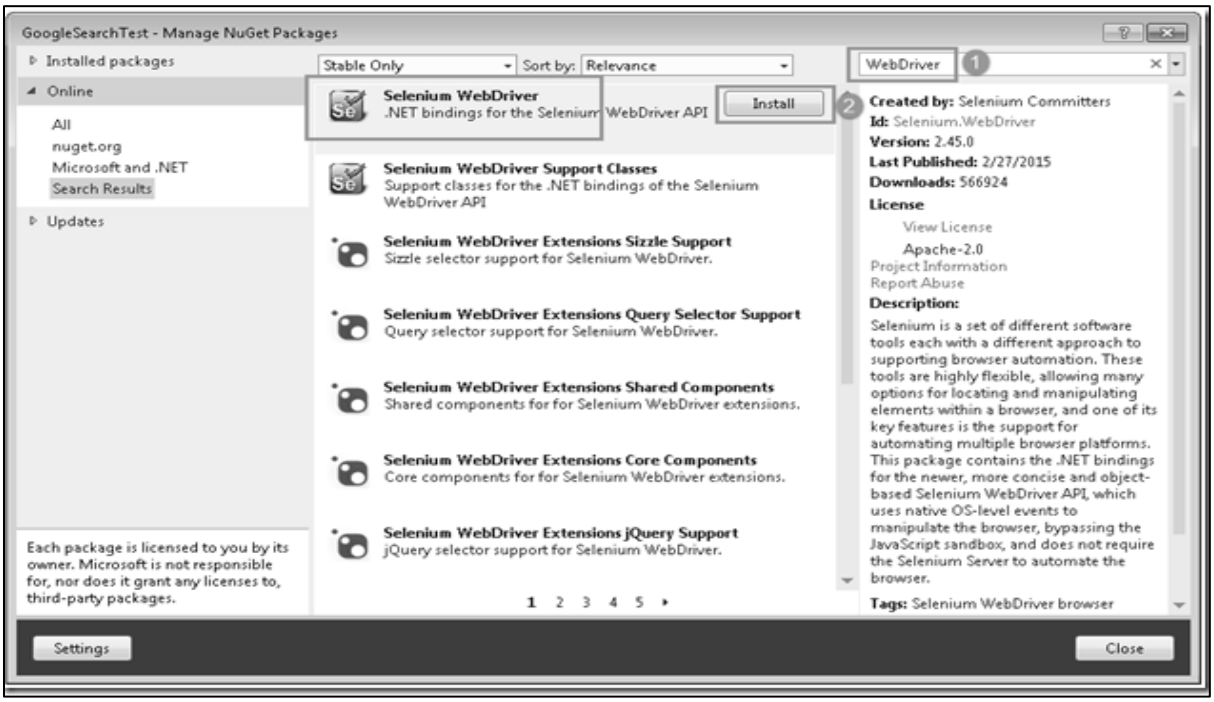


Figure -A IV-5 : Installation du cadriciel Selenium Webdriver [45]

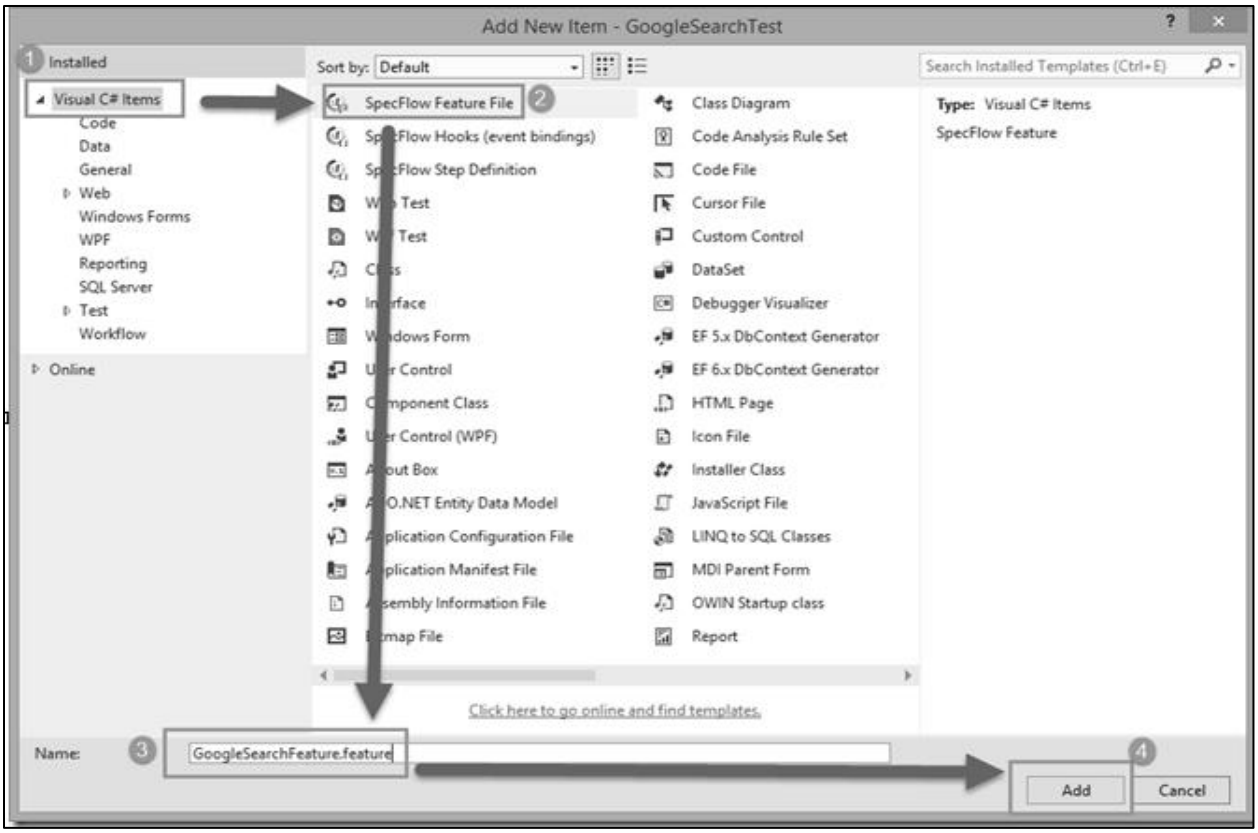


Figure -A IV-6 : Ajout d'un nouveau fichier de fonctionnalité SpecFlow [45]

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

[1]: Mayer, J. 1998

[2] Software Test Automation <http://en.wikipedia.org/wiki/Test automation> consulté le 15 mai 2016.

[3] R. M. Sharma, «Quantitative Analysis of Automation and Manual Testing » *International Journal of Engineering and Innovative Technology (IJEIT)* Volume 4, Issue 1, July 2014

[4]: Équipes de développement des applications Nethris et EmployeurD, <Centre de services de paie de CGI, Montréal-Canada> informations obtenues en juin 2015.

[5] : ISO24765, 2010

[6]: Wikipedia, test automation. <https://en.wikipedia.org/wiki/Test_automation > consulté le 16 mai 2016.

[7]: John Wiley et Sons, « The Art of Software Testing » ISBN 0-471-04328-1

[8]: wikiversity, Software testing/History of testing. <https://en.wikiversity.org/wiki/Software_testing/History_of_testing> consulté le 16 mai 2016.

[9]: Selenium official website, <http://www.seleniumhq.org/ > consulté le 16 mai 2016.

[10]: Coffee, 2003

[11]: Harpreet kaur et al Int., « Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete » *Journal of Engineering Research and Applications* Disponible en ligne www.ijera.com ISSN: 2248-9622, Vol. 3, Issue 5, Sep-Oct 2013, pp.1739-1743

[12]: Hildreth, 2004

[13]: Dustin, 2003. « Effective Software Testing: 50 Specific Ways to Improve your testing » p. 159

[14]: S. Berner, R. Weber, et R. K. Keller**,** « Observations and lessons learned from automated testing » dans *Proceedings of the 27th International Conference on Software Engineering ICSE 2005*, 2005, pp. 571–579

[15]: Mayer J., « The right way to view testing ». *Software Magazine, Supplement,* Vol 18, Issue 7, p. 3 – 7

[16]: Shea B., « Software Testing Gets New Respect ». *InformationWeek*, Issue 793, p97, 6p

[17]: M. F. Bashir et S. H. K. Banuri, « Automated model based software test data generation system » dans *Proceedings of the 4th International Conference on Emerging Technologies ICET 2008*, 2008, pp. 275–279

[18]: Cordell Vail, 12/02/02. « AUTOMATED TESTING TOOL EVALUATION » disponible en ligne http://www.vcaa.com/tools/wsipc-automatedtestingtoolevaluation.pdf

[19]: James Bach, 1999 « Test Automation Snake Oil ». *Windows Tech Journal (10/96) and the proceedings of the 14th International Conference and Exposition on Testing Computer Software*. V2.1 6/13/9

[20]: Carey Schwaber and Mike Gilpin, February 3, 2005. « Evaluating Automated Functional Testing Tools » disponible en ligne sur ftp://ftp.software.ibm.com/software/rational/web/reports/forrester_rft_eval_0205.pdf

[21]: Harpreet kaur et al. « Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete», *Int. Journal of Engineering Research and Applications* Disponible en ligne sur www.ijera.com ISSN : 2248-9622, Vol. 3, Issue 5, Sep-Oct 2013, pp.1739-1743

[22]: Aspire Systems, Inc., <http://www.aspiresys.com/WhitePapers/QTPvsSelenium.pdf >. Consulté le 28 mai 2016

[23]: SmartBear, comparaison des outils d'automatisation des tests <https://smartbear.com/learn/automated-testing/selecting-automated-testing-tools/> consultés le 28 mai 2016

[24]: Gouri Sohoni, on 4/28/2014 « Comparison of Automated Testing Tools: Coded UI Test, Selenium and QTP » disponible en ligne sur http://www.dotnetcurry.com/tools/1004/comparing-automated-testing-tools-codedui-qtp-selenium

[25]: Yogesh Kumar. « Comparative Study of Automated Testing Tools: Selenium, SoapUI, HP Unified Functional Testing and Test Complete » September 2015, Volume 2, Issue 9

JETIR (ISSN-2349-5162). Disponible en ligne sur http://www.jetir.org/papers/JETIR1509007.pdf

[26]: Vinita Malik, « Comparative Study of Automated Web Testing Tools », vol.6 issue 3 january 2016, ISSN 2278-621X, *international journal of latest Trends in Engeneering and Technology (IJLTET)*. Disponible en ligne sur http://www.ijltet.org/journal//55.pdf

[27]: The Organizing Committee, May 19-21 2016. « AQTR 2016 IEEE International Conference on Automation, Quality and Testing, Robotics. » Disponible sur youtube.com

Google , 10 nov. 2015 . « Google Test Automation Conference 2015. » disponible sur youtube.com

James Bach, 7 sept. 2011 « Open Lecture by James Bach on Software Testing. » Disponible sur youtube.com

[28]: Abid K P Abdulla, May 27, 2015. « Automation tools comparaison. » Disponible en ligne sur http://www.slideshare.net/KizhoreAbidPuthenpur/automation-tool-comparison

Aspire Systems, 5 décembre 2013. « Test Automation Tool comparison – HP UFT/QTP vs. Selenium. » Disponible en ligne sur http://fr.slideshare.net/AspireSystems/qt-pvs-selenium

Sauce Labs, 29 janvier 2015. « QTP/UFT vs. Selenium. » Disponible en ligne sur http://fr.slideshare.net/saucelabs/qtp-vs-selenium-what-to-consider

[29]: Microsoft, Supported Configurations and Platforms for Coded UI Tests and Action Recordings.< https://msdn.microsoft.com/en-us/library/dd380742.aspx> consulté le 16 mai 2016.

[30]: Naveen Varadaraju, 27 February 2015. « Getting Started with Coded UI Automation Tool ». Disponible en ligne sur http://www.evoketechnologies.com/blog/getting-started-coded-ui-automation-tool/

[31]: Rohit Rampal, 16th May 2014 Coded UI Introduction(MSDN Help). <http://www.codeduitutorial.com/> consulté le 20 juin 2016.

[32]: Naresh Chintalcheru, 25 juillet 2013. « Automation Testing using Selenium. » Disponible en ligne sur

[33]: Interface de Selenium IDE, <http://qaautomationqtp.blogspot.ca/2013/08/creating-first-test-in-selenium-ide.html> consulté le 20 mai 2016.

[34]: Tutoriel de selenium IDE, <http://www.softwaretestinghelp.com/selenium-ide-script-selenium-tutorial-3/ > consulté le 20 juin 2016.

[35]: SmartBear official website, <https://smartbear.com/product/testcomplete/overview/ > consulté le 20 mai 2016.

[36]: Interface TestComplete, <http://automated-360.com/testcomplete-tutorial/testcomplete-ide/> consulté, le 20 mai 2016

[37]: SmartBear official website, <https://smartbear.com/learn/automated-testing/selecting-automated-testing-tools/ > consulté le 20 mai 2016.

[38]: Interface UFT, <http://www.softwaretestingclass.com/type-of-testing-in-unified-functional-testing-uft-12-0-uftqtp-training-tutorial-6/ > consultée, le 20 mai 2016

[39]: Tutoriel de l'outil de test UFT, <http://www.softwaretestingclass.com/qtp-uft-hp-unified-functional-testing-training-tutorial-series/ > consulté le 20 juin 2016.

[40]:Figure expliquant l'intégration des outils de tests SpecFlow, Microsoft Test et Selenium webdriver ensemble, <http://www.infragistics.com/community/blogs/damyan_petev/archive/2013/09/05/infragistics-test-automation-part-2.aspx > consulté le 03 juin 2016.

[41]: Page web official de la technologie SpecFlow. < http://www.specflow.org/ > consulté le 20 juin 2016.

[42]: Objis, Tutoriel Selenium: installation Selenium ide <http://www.objis.com/formation-java/tutoriel-selenium-installation-selenium-ide.html > consulté le 20 juin 2016.

[43]: Shai Raiten, 7 mai 2013 « Advanced Coded UI Testing. » disponible en ligne sur http://fr.slideshare.net/ShaiRaiten/coded-ui-ws

[44]: RomSoft SRL, 17 juin 2013. « Test Complete » disponible en ligne sur http://fr.slideshare.net/romsoft/test-complete

[45]: Satish Kumar, Tuesday, 2 June 2015. «selenium Webdriver script with SpecFlow and Visual Studio. » disponible en ligne sur http://codeduiadda.blogspot.ca/