

EPSI  
73, rue de Marseille  
33000 Bordeaux

SII  
11, avenue Neil Armstrong  
33700 Mérignac

## L'AUTOMATISATION DE TESTS

Martin Lezer

Adrien Sifre

2013

# **REMERCIEMENTS**

Je remercie l'entreprise SII de son offre de stage.

Je remercie Étienne David de m'avoir accepté dans son agence.

Je remercie Adrien Sifre de m'avoir guidé le long du stage.

Je remercie Mélanie Mozas de m'avoir fait confiance.

Je remercie Thomas Chauvet pour sa collaboration sur le projet.

# SOMMAIRE

<b>INTRODUCTION .....</b>	<b>1</b>
<b>I. Types de tests .....</b>	<b>8</b>
1. Les méthodes .....	8
2. Les tests fonctionnels.....	12
3. Les tests non fonctionnels .....	19
4. L'organisation.....	24
<b>II. L'AUTOMATISATION DE TESTS .....</b>	<b>29</b>
1. L'amélioration de la qualité.....	29
2. La réduction des coûts.....	31
3. Le calcul du gain.....	34
<b>III. MISE EN OEUVRE.....</b>	<b>42</b>
1. Choisir quoi automatiser .....	42
2. Choisir les outils .....	45
3. Le processus.....	50
<b>CONCLUSION .....</b>	<b>56</b>
<b>GLOSSAIRE.....</b>	<b>64</b>
<b>Table des matières .....</b>	<b>66</b>
<b>BIBLIOGRAPHIE.....</b>	<b>68</b>
<b>ANNEXE.....</b>	<b>69</b>
<b>SYNTHESE ANGLAIS .....</b>	<b>I</b>

# INTRODUCTION

Le coût mondial des dysfonctionnements des logiciels est évalué à 312 milliards de dollars chaque année.

Une étude menée par le National Institute of Standards and Technology (NIST) sur l'économie du secteur des logiciels aux États-Unis a démontré que les logiciels défectueux coûtent environ soixante milliards de dollars aux entreprises américaines chaque année, soit un tiers du marché des ventes de logiciels dans le pays.

L'une des plus importantes défaillances logicielles survenues aux États-Unis a impliqué l'entreprise Knight Capital Group le premier août deux mille douze. Par le biais de cette maison de courtage située à New York, onze pour cent des actions cotées en bourse américaines ont été impactées. Le logiciel d'échanges boursiers a causé de violentes fluctuations dans les prix en multipliant les échanges aléatoires sur le marché. La société a perdu quatre cent quarante millions de dollars en l'espace de trente minutes, ce qui correspond à trois fois le total de ses profits annuels. L'entreprise Knight Capital Group s'est retrouvée à la limite de la banqueroute et de nombreuses entreprises américaines, dépendantes des échanges de cette société, se sont vues menacées à leur tour. Il a fallu que les investisseurs renflouent la société avec quatre cents millions de dollars pour éviter les pertes en cascade.

Comme il l'a été expliqué précédemment, la Judge Business School de Cambridge, en Angleterre, avec la collaboration de l'entreprise Undo Software ont estimé que le coût total des bugs dans le monde s'élève à 312 millions de dollars par an. Depuis 2008, soit depuis cinq ans, l'Union européenne a dépensé cinq cent quatre-vingt-quatorze milliards de dollars pour sauver l'économie de la Grèce, l'Irlande, le Portugal et l'Espagne. Or, si nous multiplions le coût annuel des logiciels défaillants en Europe par cinq, soit cinq ans, nous obtenons plus du double de ces dépenses.

L'entreprise Undo Software conclut son étude en exprimant sa crainte. D'après elle, si aucune solution n'est trouvée pour inverser cette tendance, le coût grimpera en flèche chaque année.

Pour obtenir ces résultats, les deux études se sont basées sur les conséquences économiques des logiciels défectueux sur les sociétés qui les utilisent. En effet, de nos jours, chaque entreprise dépend de logiciels pour chaque étape de la production. Que ce soit pour le développement, la distribution ou le suivi des produits et services.

Nous pouvons citer par exemple les chiffres calculés par le NIST pour leur étude du marché des logiciels aux États-Unis. D'après celle-ci, les ventes de logiciels correspondaient à cent quatre-vingts milliards de dollars l'année 2000 sur le sol américain. Il existait six cent quatre-vingt-dix-sept mille ingénieurs en informatique et cinq cent quatre-vingt-cinq mille programmeurs aux États-Unis. Ce qui implique des millions de lignes de code créées chaque année.

D'autre part, les bugs impactent aussi les entreprises qui créent ces logiciels. En effet, d'après les deux études citées précédemment, un développeur passe plus de la moitié de son temps de travail à corriger les erreurs du programme qu'il est en train de développer. Or, un célèbre adage affirme: " Le temps, c'est de l'argent ". En conséquence, l'entreprise qui l'emploie perd de l'argent lorsque le temps de production dure plus longtemps.

Cependant, il n'est pas possible de s'arrêter aux conséquences économiques des défaillances des logiciels quand des vies sont en jeu.

En effet, le domaine militaire n'échappe pas aux besoins informatiques des entreprises.

Il est possible de citer par exemple l'incident survenu en 1991 en Arabie Saoudite. Un logiciel installé par les soldats américains leur permettait de se protéger des missiles tirés sur eux. Le logiciel rencontra un problème. Le temps pris pour le résoudre a engendré la mort de vingt-huit soldats.

La place que prend l'informatique dans notre société est de plus en plus importante. Que ce soit pour raisons économiques ou humaines, cette évolution pousse les créateurs de logiciels à programmer en se focalisant sur la qualité du produit.

Cependant, la qualité d'un produit n'est pas facile à définir car elle est un concept complexe et polymorphe. Et plus particulièrement dans le domaine de l'informatique.

Au départ, la qualité d'un programme informatique se prouvait par l'absence de bugs. Elle pouvait donc se mesurer facilement. Il suffisait de compter le nombre de bugs par lignes de code. On estimait que deux bugs toutes les mille lignes de code étaient un bon ratio.

Certains chercheurs en informatique ont commencé à réfléchir sur la définition de qualité logicielle.

Philip B. Crosby fut l'un des premiers à s'intéresser à ces recherches. Il a pu établir quatre principes fondamentaux:

- La qualité est définie par la conformité aux spécifications.
- Le système mis en place pour produire de la qualité réside dans la prévention et non pas dans l'évaluation. Il faut donc créer de la qualité au premier essai. Nous sommes responsables de nos erreurs, il ne faut pas que quelqu'un d'autre doive inspecter notre travail.
- L'objectif est de zéro défaut.
- Pour mesurer la qualité, nous devons mesurer le prix de la qualité. Si les imperfections sont corrigées rapidement, il y a un effet positif immédiat sur la performance et sur les relations avec le client. Une entreprise se doit d'investir dans la formation de ses employés pour éliminer toute erreur et ainsi récupérer le coût par l'absence de problèmes.

Une autre figure des réflexions sur la qualité logicielle fut Deming qui a défini un concept plus large que Crosby. D'après lui, la qualité doit être définie par la satisfaction du client.

Les entreprises IBM et Hewlett Packard se sont basées sur ces réflexions pour mettre en place un système d'évaluation de la qualité. Pour ces deux sociétés, la qualité d'un logiciel

était étroitement liée avec la satisfaction de l'utilisateur. L'entreprise IBM a véritablement créé une science de l'enquête de satisfaction avec huit dimensions:

- fonctionnalité
- facilité d'utilisation
- performance
- fiabilité
- instabilité
- maintenabilité
- documentation
- disponibilité

En 1985, le Total Quality Management (TQM) a été fondé par la Naval Air Systems Command pour décrire son approche de la qualité. Cette théorie s'appuie sur les travaux de Crosby, Deming, mais aussi sur ceux de Juran, Ishikawa et Feigenbaum. Elle reprend les premières idées selon lesquelles le succès à long terme en termes de qualité peut être atteint par la focalisation sur la satisfaction du client. Mais elle ajoute qu'une culture de la qualité doit être mise en place dans l'entreprise pour améliorer les différents processus internes, les produits et les services.

D'après cette théorie, une entreprise doit suivre quatre caractéristiques essentielles:

- la focalisation sur le client: une satisfaction totale doit être atteinte par des études de besoins et de satisfaction
- l'amélioration des processus: il faut réduire les variations de processus
- la culture de la qualité: il faut mettre en place une culture de la qualité dans l'entreprise à toute échelle
- les mesures et analyses: un système de mesure de la qualité doit être mis en place orientée sur le but à atteindre

En ce qui concerne la satisfaction client, il faut s'intéresser au modèle Kano élaboré en 1984 par Noriaki Kano.

D'après lui, la satisfaction et l'insatisfaction d'un client ne sont pas diamétralement opposées. En effet, une fonctionnalité d'un produit peut apporter une certaine satisfaction à un client sans pour autant que son absence cause son insatisfaction.

Noriaki Kano définit trois types d'attentes clients:

- les attentes de base: ces fonctions ne sont pas forcément exprimées par le client car elles sont pour lui évidentes. Elles ne sont donc pas un levier de satisfaction. En revanche, leur absence provoque de la frustration chez le client et donc de l'insatisfaction. Elles sont donc obligatoires.
- Les attentes proportionnelles: ces fonctions sont exprimées par le client. Sa satisfaction est proportionnelle à leurs performances.
- Les attentes attractives: ces fonctions ne sont pas exprimées. Cependant, elles peuvent répondre à un besoin inconscient du client. Elles créent donc une surprise, suivie par un certain enthousiasme. Il éprouve donc une grande satisfaction.

Des normes telles que celles de la famille ISO 9000 et le CMMI furent créées en s'appuyant sur la théorie TQM. Les certifications obtenues par l'application de ces normes permettent aux entreprises de démontrer qu'elles sont aptes à fournir un service ou un produit conforme aux exigences des clients et aux exigences réglementaires.

Il a été évoqué précédemment qu'il existe une étroite relation entre les travaux sur la définition de la qualité et l'absence de défaillances dans les logiciels. Mais aussi entre la qualité et la conformité aux exigences du client. Le processus de tests a été mis en place dans la programmation pour vérifier le bon fonctionnement d'un logiciel. Il est devenu une partie importante du cycle de production du logiciel.

Un logiciel n'est pas comme les autres processus physiques dans lesquels on reçoit une entrée puis une sortie est produite. Il diffère dans la manière d'échouer. En effet, il est impossible de tout détecter face aux nombreuses valeurs possibles contenues, contrairement aux systèmes physiques. Le processus de tests doit donc s'adapter à cette différence.

Il correspond à la pratique d'exécuter un programme avec l'intention de trouver les erreurs.

Il a quatre objectifs principaux:

- la démonstration: Il prouve que le logiciel peut être intégré, vendu, avec un risque acceptable. Les fonctionnalités, sous certaines conditions, marchent et peuvent donc être intégrées.

- La détection: il découvre les erreurs et défaillances et les trace.
- La prévention: il fournit des informations permettant de réduire le nombre d'erreurs. Mais aussi des moyens d'éviter les problèmes dans le futur.
- Améliorer la qualité: les fonctions citées précédemment permettent d'améliorer la qualité du logiciel.

De nombreux types de tests logiciels sont présents dans le processus de développement d'une application. Chacun de ces types s'attache à vérifier une partie différente du logiciel ou un domaine particulier du logiciel.

L'organisation des tests logiciels au sein du cycle de développement dépend de la méthode de gestion du projet. Les méthodes de gestion de projet les plus populaires sont la méthode du cycle en V et les méthodes agiles. Dans chacune d'elles, les tests logiciels ne sont pas effectués de la même manière ou au même moment.

Il existe deux manières de tester les logiciels:

- le test manuel: Il est effectué par un humain qui vérifie le code et note les bugs.
- Le test automatique: Il est géré par un logiciel.

Il faut s'intéresser à l'importance des tests automatiques. L'automatisation de tests présente certains avantages par rapport aux tests manuels:

- plus sûre: elle est capable de répéter des opérations précises sans intervention humaine et donc sans erreurs humaines
- programmables: les tests sont plus sophistiqués
- compréhensible: une suite de tests est créée pour chaque fonctionnalité
- réutilisable: il est possible de réutiliser les mêmes tests pour chaque version de l'application
- rapidité: les tests sont effectués plus rapidement qu'avec implication humaine
- plus faible coût: le nombre de ressources nécessaires est diminué

Cependant, l'automatisation des tests ne peut pas être appliquée à tous les types de tests.

D'autre part, la mise en place d'un tel système a un certain coût et est difficile à mettre en place. Il faut donc effectuer une étude préalable afin de définir les zones du logiciel sur lesquelles l'automatisation de tests est nécessaire.

Avant d'installer un tel système, il est important d'étudier les inconvénients liés à ce processus mais aussi ce que cette mise en place peut apporter à l'entreprise.

Une fois la décision prise, différents choix de technologies devront être établis avant de passer à l'installation du système.

# I. TYPES DE TESTS

Il existe deux approches différentes en matière de tests logiciels. La méthode de la boîte noire et la méthode de la boîte blanche. Chaque type de test suit une de ces méthodes.

## 1. Les méthodes

### a. La méthode de la boîte blanche

Les tests de la boîte blanche sont des tests de la structure interne d'une application. On se concentre sur le code et l'infrastructure du programme. Le testeur est capable d'interagir avec les composants internes de l'application.

Le terme " boîte blanche " vient de la possibilité de regarder à travers la boîte pour observer le comportement interne de l'application.

La technique de la boîte blanche est la plus importante des techniques de test. En effet, parmi les six tests les plus basiques du cycle en V, trois utilisent cette technique: le test unitaire, le test d'intégration, que nous verrons par la suite, et le test de régression que nous avons déjà expliqué.

La technique de la boîte blanche permet de tester différentes caractéristiques de l'application: la sécurité de l'application, le chemin traversé par chaque entrée dans le code, si la sortie correspond à celle que l'on attend, le comportement des boucles dans le code et enfin chaque objet et fonctions dans le code

Ces tests sont effectués par les développeurs. En effet, il est important de comprendre le code de l'application afin de pouvoir écrire des tests pour chaque processus interne.

Il existe différentes techniques de boîte blanche.

Tout d'abord, le test de validité des boucles.

D'autre part, le test des branches. Lorsqu'il existe plusieurs options dans une condition, chaque décision doit être testée.

Il existe aussi le test de flux de données qui contrôle comment les données parcourent le programme.

Mais aussi la couverture des instructions qui correspond au calcul du pourcentage d'instructions qui sont exécutées par les cas de test. Si le pourcentage est inférieur à 100 %, cela signifie que des lignes de code ne sont pas testées

De plus, la technique de la couverture des branches. C'est le calcul du pourcentage de décisions dans le programme testées.

Il y a la couverture des conditions, qui est le calcul du pourcentage de décisions dans le programme dont chaque état est testé dans les cas de test.

Enfin, il faut évoquer le test du chemin de base. C'est l'utilisation d'un chemin logique pour créer des cas de test basiques exécutant au moins une fois chaque déclaration.

Cette méthode comporte certains avantages.

Elle permet d'ajouter de l'introspection. En effet, les testeurs peuvent utiliser les objets de l'application par la programmation. Ainsi, même si l'interface graphique de l'application change, les cas de test sont toujours à jour.

Mais aussi de la stabilité. La technique de la boîte blanche offre une plus grande stabilité et réutilisation des cas de test si les objets internes de l'application ne changent pas.

Enfin, elle présente de la rigueur. Effectivement, les tests de boîte blanche sont les seuls pouvant examiner chaque procédure interne à l'application.

Cependant, nous pouvons relever certains inconvénients à cette méthode.

D'une part, la complexité. En effet, pour pouvoir examiner chaque partie de l'application, il est nécessaire que le testeur connaisse parfaitement son fonctionnement et soit très expérimenté

De plus, cette méthode contient une certaine fragilité. Si des changements sont apportés au code, que les objets changent de nom ou qu'un nouveau chemin est ajouté, les scripts des cas de test deviennent obsolètes. Ils nécessitent donc un d'être constamment maintenus.

Enfin, l'intégration. Ceci car des outils de tests doivent être installés sur le système. Ceci peut engendrer des problèmes de performance sur la machine. De plus, ces outils doivent

être compatibles avec tous les systèmes d'exploitation si le programme testé l'est aussi. Ce qui n'est pas toujours le cas.

Cette méthode s'oppose à l'approche de la boîte noire.

## b. La méthode la boîte noire

Contrairement à la technique de la boîte blanche, les tests de la boîte noire sont effectués sans regarder la structure interne du code. Ces tests sont entièrement basés sur les spécifications du logiciel. On se concentre sur les entrées et sorties du système. On vérifie que les sorties correspondent à celles attendues sans regarder comment l'application fonctionne. Ainsi, puisqu'il n'y pas besoin de comprendre le comportement du programme, le testeur et le développeur peuvent être deux personnes différentes.

Le terme « boîte noire » vient du fait qu'on ne peut pas voir à travers la boîte noire. Autrement dit, on ne peut pas voir le code à l'intérieur d'un programme.

Trois des six plus importants tests utilisent la technique de la boîte noire : le test système, le test d'acceptation et le bêta test.

Les tests de la boîte noire peuvent être effectués en différentes étapes.

Tout d'abord, on analyse les besoins et les spécifications de l'application. Ensuite, le testeur choisit des entrées valides et invalides. Après, il détermine les sorties qu'il doit obtenir avec ces entrées. Par la suite, on crée les cas de test avec ces entrées suivie de leur exécution. Le testeur compare alors les sorties avec les sorties attendues. Enfin, on reporte les bugs et une fois qu'ils sont corrigés, on teste de nouveau.

Il existe différentes méthodes de boîte noire.

La première est la méthode basée sur un graphe : un graphe d'objets est créé en rapport avec l'application. À partir de celui-ci, chaque relation entre objets est accompagnée d'un cas de test.

Une autre méthode consiste à deviner l'emplacement des erreurs. Cette technique est laissée à l'appréciation du testeur. Il doit chercher et deviner où pourraient se situer les erreurs et créer les cas de test en fonction.

D'autre part, il y a l'analyse des valeurs à chaque limite: on suppose que si une application fonctionne avec la valeur maximale et minimale, il y a de fortes chances qu'elle marche avec une valeur intermédiaire. Le testeur utilise donc le maximum et le minimum pour chaque valeur.

Enfin, le partitionnement en classe d'équivalence où les valeurs d'entrées sont divisées en différentes classes de données à partir desquelles le testeur écrit ses cas de test

La technique de la boîte noire procure différents avantages.

Tout d'abord, la facilité d'utilisation. En effet, les testeurs n'ont pas besoin de connaître le comportement interne de l'application. Il est donc plus facile de créer les cas de test, comme le ferait un utilisateur final.

D'autre part, le développement de cas de test plus rapide. Le testeur se concentre essentiellement sur les différentes possibilités qui lui sont données par l'interface graphique et n'a pas besoin de chercher dans la structure interne chaque chemin que l'application pourrait emprunter et qui aurait besoin d'être testé.

Enfin, la simplicité : la technique de la boîte noire se focalise seulement sur les sorties attendues par le testeur en fonction d'entrées. La complexité de l'application n'a pas de conséquences sur la méthode de test.

Cependant, cette technique comporte aussi des inconvénients.

Le premier est la maintenance. Effectivement, cette méthode de test étant essentiellement basée sur les entrées de l'application, si celles-ci ont été changées, les scripts devront être modifiés en fonction.

Mais aussi le manque d'introspection. En effet, si le testeur n'entre pas complètement dans une application, il est difficile d'imaginer qu'il a pu être capable de tester en profondeur le système.

Les types de test peuvent être divisés en deux types distincts. Les premiers sont fonctionnels et en opposition les non fonctionnels.

## **2. Les tests fonctionnels**

Les tests fonctionnels vérifient la concordance entre les fonctionnalités de l'application et les spécificités métier. Cette famille de types de test utilise la méthode la " boîte noire ". Elle ne concerne donc pas le code source de l'application.

### **a. Les tests unitaires**

Les tests unitaires sont effectués durant le processus de développement d'un logiciel. Chaque composant individuel du programme est testé. Ce composant correspond à la plus petite partie d'une application. En programmation orientée objet, cette partie est la classe. Tandis qu'en programmation procédurale elle peut être une fonction, une procédure ou un programme individuel. Généralement, un test unitaire comporte quelques entrées et une seule sortie. Ils sont conçus et exécutés par le développeur.

Un test unitaire est réalisé en trois étapes: la création du plan de test, l'écriture des cas de tests et enfin l'exécution des tests.

Il est important de consacrer du temps à créer les tests unitaires car ils permettent de réduire les coûts. En effet, détecter et corriger les erreurs durant la phase de développement engendre une économie de temps par rapport à une correction tardive dans le cycle de conception d'un logiciel.

D'autre part, les tests unitaires fournissent d'autres avantages.

Après modification du code, le développeur peut vérifier que l'application fonctionne toujours.

De plus, il est possible de tester une partie du projet sans avoir besoin d'attendre que les autres soient terminées

Enfin, les tests unitaires permettent à un développeur de comprendre les fonctionnalités fournies par une application sans l'avoir programmée lui-même. En effet, en regardant les tests unitaires, il peut examiner le fonctionnement du logiciel.

Cependant, il existe certaines limites.

En effet, il est compliqué de tester chaque cas possible. Autrement dit, le déroulement d'un programme peut emprunter différents chemins en fonctions d'une multitude d'entrées. Le développeur ne peut pas concevoir de test unitaire pour chacun de ces cas.

D'autre part, le comportement en tant que groupe de chacune unité n'est pas vérifié. C'est pour cela que l'on doit procéder à un test d'intégration.

## b. Les tests d'intégration

Un logiciel est constitué de différents modules pouvant être développés par différents programmeurs. Les tests d'intégration consistent à vérifier la communication des données entre ces différents modules après les avoir regroupés en une seule entité.

Il existe différentes raisons démontrant l'importance de ces tests.

En effet, les modules d'un programme étant développés par différentes personnes, il peut y avoir une logique et un point de vue qui varient. Les tests d'intégration permettent de vérifier que ces modules peuvent travailler de manière unie.

D'autre part, les besoins et spécifications client peuvent être modifiés pendant la phase de développement. Il est donc important de tester le programme après intégration pour vérifier que les modifications ont été prises en compte si chaque module n'a pas été testé de manière unitaire.

De plus, des erreurs peuvent subvenir entre les interfaces des modules et la base de données. De même entre le matériel et les modules.

Enfin, il peut y avoir des problèmes de gestion des erreurs après intégration

Il existe trois méthodes différentes pour effectuer les tests d'intégration.

La première est intitulée le Big Bang. Dans celle-ci, on intègre tous les modules en une seule fois, puis on teste le système. Cette approche peut être efficace si le système est d'une importance relative. Cependant, cette méthode peut créer des oubli de tests car une quantité importante est engendrée d'un coup. D'autre part, les testeurs doivent attendre que chaque module soit terminé avant de pouvoir commencer leur travail. Ce qui peut leur laisser moins de temps s'il y a un retard de livraison. Enfin, certains modules peuvent contenir un risque plus important que d'autres. Or, avec cette approche, ils ne seront pas testés en priorité.

Il existe aussi la méthode de bas en haut dans laquelle on teste tout d'abord les modules de bas niveau avec les modules situés au niveau au-dessus. Le problème d'attente de tous les modules avant le test est supprimé. Cependant, les modules les plus importants, qui contiennent donc un niveau plus critique sont testés en dernier. Ce qui laisse moins de temps pour corriger des défauts majeurs.

Enfin, il y a la méthode de haut en bas. On teste les modules du plus haut niveau jusqu'au niveau le plus bas. Ainsi, les modules avec un haut niveau critique sont testés en priorité. En revanche, les testeurs doivent simuler le comportement des modules de plus bas niveau pour pouvoir les tester, ce qui ajoute une charge de travail supplémentaire.

La démarche de tests d'intégration s'effectue en plusieurs étapes. Tout d'abord, préparer le plan de test. Ensuite, concevoir les scénarios, les cas et les scripts de test. Après, exécuter les cas de tests. Puis, rapporter les défauts. Ensuite, tester de nouveau les anciens défauts après correction. Enfin, il faut répéter les trois étapes précédentes jusqu'à ce que le test soit un succès.

### **c. Les tests de régression**

Le but des tests de régression est de vérifier que toute modification apportée à l'application n'a pas affecté d'autres parties du programme.

Pour cela, un test de régression consiste à exécuter à nouveau des cas de test sur toute l'application ou sur une sélection des parties d'un programme pour s'assurer que cela fonctionne toujours.

Une partie d'un programme peut contenir de nombreuses dépendances au sein de l'application. Lorsque cette partie est modifiée pour corriger un bug ou que du code a été rajouté, il se peut que certaines parties du programme, en relation avec les modifications, rencontrent des problèmes.

Il est donc important en termes de qualité de vérifier à chaque fois que de nouveaux problèmes ne sont pas apparus dans le code.

Il existe différentes techniques de test de régression.

La première est de tout tester à nouveau. Chaque test existant est donc exécuté à nouveau. Il subsiste cependant un problème de coût avec cette méthode qui consomme beaucoup de temps et de ressources.

Une autre technique est la sélection des cas de test à exécuter qui est moins. Celle-ci est moins coûteuse.

Enfin, il existe la priorisation des cas de test dans laquelle on sélectionne les cas de test en fonction de l'impact commercial, des fonctionnalités critiques et des fonctionnalités les plus utiles à l'application. Cette technique est la plus efficace car elle permet de réduire le coût des tests de régression.

#### **d. Les tests d'acceptation**

Les tests d'acceptation sont exécutés après les tests unitaires, d'intégration et système. Ils font partie de la phase finale du cycle en V, avant la mise en production du programme.

Ils permettent donc de valider la fin de relation commerciale établie avec le client.

Le programme est testé afin de vérifier s'il répond aux exigences fixées par le client au début du cycle en V et donc de la production.

Le test d'acceptation est un test de « Boîte noire ». Nous décrirons par la suite la signification de cette classification.

Il existe différents types de tests d'acceptation.

D'une part, les tests d'acceptation internes qui sont exécutés par des membres de l'équipe, mais qui ne sont pas concernés par le développement de l'application.

De l'autre, les tests d'acceptation externes qui sont réalisés par des personnes extérieures à l'équipe travaillant sur l'application. Il en existe aussi deux types.

Tout d'abord, les tests d'acceptation client qui sont effectués par le client lui-même.

Mais aussi, les tests d'acceptation utilisateur (ou test Beta) dans lesquels les personnes concernées par l'utilisation du logiciel, que ce soit le client ou les clients du client, sont chargées de tester le logiciel.

Les tests d'acceptation sont importants dans le cycle en V car les développeurs se basent sur des documents comprenant les exigences clients sans pour autant être mis en relation avec le client. Ce manque de communication peut engendrer une incompréhension du besoin client. Il est donc nécessaire que le client puisse tester l'application et la valider avant la mise en production.

D'autre part, il peut exister un manque de communication durant la phase de développement concernant des changements que le client souhaite apporter à l'application. Il est donc important que le client puisse vérifier que l'application est conforme à ses besoins.

Afin de pouvoir exécuter les tests d'acceptation, différentes tâches doivent être terminées: l'écriture des besoins client, le code de l'application, les tests unitaires, d'intégration et système, aucun défaut n'a été trouvé dans la phase d'intégration, seules les erreurs de design sont acceptables, pas de défaut majeur survenu dans les tests de régression, l'environnement de test d'acceptation et enfin l'accord de l'équipe des tests système pour commencer les tests d'acceptation.

Le déroulement des tests d'acceptation s'effectue en différentes étapes.

Il faut tout d'abord effectuer l'analyse des besoins client.

Ensuite, il faut passer à la création du plan de test. C'est la stratégie à appliquer pour vérifier que l'application correspond aux besoins spécifiés par le client.

Par la suite, identifier les scénarios et cas de test. Ils doivent respecter les besoins et les cas de test permettent de couvrir ces scénarios.

Puis, il faut préparer des données de test.

Ensuite, lancer les tests et enregistrer les résultats. Si des bugs sont trouvés, il faut relancer les tests après correction.

Enfin, il faut valider la phase de test. Une fois les tests d'acceptation terminés, les plans de tests, scénarios, cas et résultats sont envoyés pour valider le commencement de la mise en production.

## e. Les Bêta tests

Ces tests sont effectués par des utilisateurs réels sur un environnement réel. Ils sont considérés comme des tests d'acceptation externes. On définit un nombre d'utilisateurs afin

d'obtenir leurs avis sur la qualité du produit qu'ils ont utilisé. Ces testeurs ne doivent pas faire partie de l'entreprise qui développe le produit. D'autre part, ils utilisent l'application sur leurs propres machines.

Ainsi, ils peuvent découvrir des erreurs dans le programme qui n'avaient pas été trouvées pendant le cycle de développement. D'autre part, ils permettent d'obtenir un avis avant de lancer le produit sur le marché. Ceci afin de modifier le produit pour qu'il réponde aux besoins avant d'entrer dans sa version finale. Enfin, ils permettent de montrer qu'un produit est prêt à être commercialisé.

Il existe différents types de Bêta tests. Dans le type traditionnel, le produit est distribué sur le marché visé. En ce qui concerne le public, tout le monde peut tester le produit à partir d'un lien sur internet. Enfin, la technique dans lequel le produit est distribué à l'intérieur d'une entreprise et est utilisé par les employés.

Ces tests comportent certains avantages.

Tout d'abord, le produit étant testé par de véritables utilisateurs, si les retours sont positifs, on diminue le risque d'échec.

Ensuite, l'infrastructure réelle a pu être testée.

D'autre part, la qualité du produit est améliorée grâce aux retours des utilisateurs.

Enfin, le client peut voir que l'entreprise fait preuve de bonne volonté en essayant d'améliorer le produit. De ce fait, la satisfaction du client augmente.

Cependant, des problèmes peuvent être rencontrés. En effet, en comparaison avec les autres tests qui sont effectués au sein de l'entreprise qui créé le logiciel, il y a une perte de contrôle lors de Bêta tests car le produit est lancé dans le monde réel. Il devient difficile de récolter les résultats des tests. De plus, il est difficile de choisir qu'elles sont les personnes les plus à même de tester le logiciel et de maintenir leur participation sur le projet.

## f. Les tests de détection de fumée

Le test de détection de fumée correspond à un test de vérification de la compilation d'un programme.

Le terme « détection de fumée » vient des tests de matériels informatiques. Lorsqu'un élément passait le test on disait qu'il n'avait pas pris feu lors de la première utilisation.

Ce test sert donc à vérifier que les fonctionnalités les plus importantes du programme marchent. Si le test passe, l'application est testée plus en profondeur.

Il est un sous-ensemble du test de régression que nous avons évoqué précédemment.

Nous pouvons relever plusieurs avantages à l'utilisation de ce test.

D'une part, minimiser les risques d'intégration. En effet, le test de détection de fumée permet d'avoir une première vérification qu'une application fonctionne toujours après chaque modification. Ainsi, une erreur peut être trouvée plus rapidement. Elle est donc plus facilement corrigible que si elle était survenue au moment de l'intégration du produit.

De plus, cela réduit les risques de faible qualité. En effectuant le test de détection de fumée tous les jours, on empêche les problèmes de prendre le contrôle du programme. Le système est constamment dans un bon état.

D'autre part, il est possible de découvrir les problèmes majeurs. Effectivement, le but premier du test est de se rendre compte que l'application rencontre un problème important.

Enfin, il survient une réduction des coûts: plus un problème est trouvé tôt dans le cycle de développement, moins il coûte cher. C'est pour cela qu'effectuer un test de détection de fumée tous les jours permet de savoir si le logiciel rencontre un problème grave.

## **g. Les tests de santé**

Le test de santé consiste à vérifier que les changements apportés à un programme ont permis de corriger les bugs existants sans ajouter d'erreurs. On confirme que chaque fonctionnalité de l'application marche correctement.

Si une erreur est trouvée, la nouvelle version du programme est rejetée sans avoir besoin de la tester en profondeur. Cela permet donc, comme pour le test de détection de fumée, d'économiser du temps et donc de l'argent.

Le test de santé est sous-ensemble du test d'acceptation que nous avons évoqué précédemment.

Il n'est pas documenté et est effectué par une équipe de testeurs.

À l'inverse des tests fonctionnels, les tests non fonctionnels s'appuient sur l'expérience client et les performances de l'application.

### **3. Les tests non fonctionnels**

Les tests non fonctionnels se basent sur les besoins et les scénarios spécifiés par le client.

#### **a. Le test de charge**

Le test de charge consiste à analyser le comportement d'un logiciel dans des conditions réelles. On simule de multiples utilisateurs qui vont manipuler le logiciel.

Ce test permet de déterminer la capacité maximale du logiciel en termes d'utilisateurs et si l'infrastructure actuelle du logiciel est suffisante pour gérer la charge atteignable après sa mise en production.

Ces tests sont importants car la performance d'un logiciel peut avoir d'importantes retombées économiques pour les entreprises. Nous pouvons prendre comme exemple les sites internet. On considère qu'un utilisateur attend aux maximums huit secondes avant de quitter un site qui serait encore en train de se charger.

Les tests de charge permettent de visualiser différentes données au sein de l'application: les temps de réponse pour chaque opération, les performances de chaque opération sous des conditions de charge importante, les performances de la base de données sous différentes charges, les délais de transmission des données entre le client et le serveur, les problèmes d'architecture du logiciel, les problèmes d'infrastructure matérielle et enfin les problèmes de configuration du serveur.

Il existe différents types de tests de charge.

D'une part, le test de performance. Dans ce test, on enregistre les temps de réponse d'un système sous une charge définie.

De plus, il existe le test de stress qui détermine quelle est la charge maximale qu'un système peut supporter. On stress l'application jusqu'à ce que sa limite soit dépassée. C'est-à-dire que le système se rompt.

D'autre part, le test d'adaptabilité. Il permet de savoir si le système actuel peut supporter un accroissement de charge durant son fonctionnement.

Mais aussi le test de stabilité dans lequel on vérifie que l'application ne connaît pas de variation de performance pendant un long temps d'utilisation.

Enfin, il existe le test de volume. Il consiste à observer le comportement de l'application pendant de larges transferts de données, de calculs et de processus avec ces données.

La mise en place de tests de charge doit suivre différentes étapes. Tout d'abord, il faut créer un environnement de test. Ensuite, concevoir les scénarios de test. Puis, pour chaque processus de test: préparer les données, déterminer le nombre d'utilisateurs, fixer les vitesses de connexion de ceux-ci, définir le matériel utilisé (navigateurs internet, systèmes d'exploitation) et configurer les serveurs. Par la suite, il faut tester les scénarios et collecter les résultats. Ceci est suivi par l'analyse de ces résultats. Après il y a le réglage du système pour enfin tester à nouveau.

## b. Le test de résistance

Le test de résistance consiste à déterminer la capacité de l'application à conserver un degré d'efficacité sous des conditions défavorables et sa gestion des erreurs. Ces conditions extrêmes peuvent être une surcharge du réseau, le chargement de multiples processus et de requêtes demandant de fortes ressources. Il permet de vérifier que l'application ne rencontre pas d'échec. Cependant, si c'est le cas, il est ainsi possible de connaître la limite de l'application en examinant les résultats du test.

Il est important de réaliser des tests de résistance afin de modifier l'application pour qu'elle puisse prendre en charge des pics d'activité. En effet, si le logiciel rencontre un certain

succès et que le nombre d'utilisateurs augmente, il est indispensable que le logiciel ne rencontre pas de problèmes pour qu'il n'y ait pas de pertes pour l'entreprise.

D'autre part, le test de résistance permet de vérifier que les erreurs sont gérées de manière efficace. En effet, si le logiciel rencontre un problème, un message d'erreur doit être prévu.

Le principe est le même que celui du test de charge. La différence est que ce dernier vérifie que l'application fonctionne sous des conditions normales, celles qui sont censées être rencontrées à la mise en production de l'application.

Il existe différents types de tests de résistance.

Le premier est le test de résistance distribué. Pour effectuer ce type de test, une architecture client-serveur est réalisée. Le rôle du serveur est d'envoyer des tests de résistance aux clients puis de vérifier leur statut. Tout d'abord, les clients contactent le serveur. Celui-ci enregistre leurs coordonnées puis commence à envoyer des données. Pendant cette étape, les clients envoient des signaux de vie au serveur. Si le serveur n'en reçoit plus, le test échoue. Il faudra donc vérifier les résultats. Idéalement, ce test doit s'effectuer la nuit avec des serveurs de capacité importante pour une meilleure efficacité.

Un autre type de tests de résistance est le test de résistance d'application. Celui-ci consiste à trouver les défauts d'une application tels que le blocage des données, les problèmes de réseau et le manque de performance.

Il existe aussi le test de résistance système. Il est utilisé lorsque de multiples systèmes sont installés sur le même serveur. Il permet de déterminer si une application bloque les données d'une autre application.

Il faut aussi évoquer le test de résistance exploratoire. Ce type est focalisé sur l'analyse du comportement du logiciel face à des scénarios improbables.

### **c. Le test d'utilisation**

Les tests d'utilisation sont effectués par un panel de futurs utilisateurs. Pour cela, on étudie de potentiels futurs utilisateurs pendant leur manipulation de l'application.

Ils permettent de déterminer si l'application est utile. C'est-à-dire si elle fournit des fonctionnalités nécessaires à l'utilisateur. D'autre part, si elle est utilisable. Autrement dit, ces fonctionnalités sont faciles et plaisantes à utiliser. Enfin, ils déterminent si elle est désirable. Si l'utilisateur souhaite utiliser une application contenant ces fonctionnalités.

Les tests d'utilisation permettent de surveiller cinq caractéristiques de l'application.

La première est l'efficacité. Est-il facile pour l'utilisateur d'accomplir certaines tâches lors de la première navigation sur l'application ?

La seconde caractéristique est la facilité d'apprentissage : une fois que l'utilisateur a appris à utiliser l'application, à quelle vitesse peut-il accomplir les tâches ?

D'autre part, la mémorisation : après un certain temps entre la première et la seconde utilisation, l'utilisateur arrive-t-il à reproduire les tâches accomplies ?

Mais aussi les erreurs : combien l'utilisateur fait-il d'erreurs ? Quelle est l'importance de ces erreurs ? Enfin, arrive-t-il à surpasser ses erreurs ?

Enfin la satisfaction : quel est le plaisir éprouvé par l'utilisateur pendant le test ?

Le processus de test d'utilisation s'effectue en cinq étapes.

Tout d'abord, il faut planifier. On commence par définir le but de ce test d'utilisation. Il faut préciser quelles sont les fonctionnalités principales et critiques à tester. En fonction de celles-ci, il faudra définir différentes tâches que l'utilisateur devra suivre pour les tester.

Ensuite, il est nécessaire de recruter. C'est-à-dire trouver le nombre de testeurs en fonction du plan de test et des profils désirés.

Vient ensuite la phase de test de l'application qui est suivie par l'analyse des données. Il faut analyser les résultats des tests afin de donner des recommandations pour l'amélioration du produit.

Et enfin transmettre les résultats. Autrement dit, envoyer l'analyse des données à l'équipe de développement afin qu'il puisse travailler sur l'application.

## d. Le test de sécurité

Le test de sécurité s'assure que le système ne comporte pas de faiblesses qui pourraient engendrer une fuite d'informations.

Le but de ce type de test est d'identifier les menaces possibles que le système pourrait rencontrer et trouver les failles de sécurité qui permettraient à ces menaces de pénétrer le système. Grâce à cela, les développeurs peuvent corriger le code vulnérable.

Il existe différents types de test de sécurité.

Le premier est l'examen de vulnérabilités. Un logiciel examine automatiquement le système pour trouver des vulnérabilités déjà identifiées et connues.

Un autre type de test de sécurité est l'examen de sécurité. Il consiste à identifier les faiblesses du système et du réseau. Par la suite, il fournit des solutions pour réduire ces risques.

D'autre part, il existe le test de pénétration. Celui-ci sera décrit par la suite. Il correspond à une simulation d'attaque sur le logiciel. Il permet d'analyser le comportement d'une partie du système face à un piratage informatique.

Enfin, il faut mentionner le test d'évaluation des risques. Il permet de classifier par niveau les risques encourus par l'application. Ceci afin de connaître le niveau d'attention qu'il faut porter aux tests sur l'application.

Les tests de sécurité doivent être intégrés tout au long du cycle de développement du projet. Chaque étape du cycle doit inclure une vérification de la sécurité.

## e. Le test de compatibilité

Ce type de test consiste à vérifier que l'application testée est capable de fonctionner sur des matériels différents, de multiples systèmes d'exploitation si les spécifications le précisent et des environnements réseau distincts.

Il existe deux types de vérification de compatibilité.

Le premier est le test de compatibilité en arrière. Il vérifie que le logiciel fonctionne avec les versions antérieures du matériel ou des applications.

L'autre type est le test de compatibilité en avant. En opposition au précédent, ce sont les dernières versions du matériel et des logiciels qui sont utilisées.

Le processus de test se décompose de la manière suivante.

Tout d'abord, il faut définir l'environnement sur lequel l'application devra être testée.

Une fois cette étape terminée, les testeurs peuvent procéder à l'installation de l'environnement, et au lancement de l'application.

Enfin, ils peuvent analyser les défauts, les fixer, puis tester à nouveau.

## **f. Le test d'installation**

Le test d'installation est très important puisque l'installation d'une application est le premier contact qu'un utilisateur a avec un logiciel. Il faut donc s'assurer qu'il ne rencontre pas de problème avec cette étape.

De plus, le logiciel connaît un processus de dématérialisation. Par conséquent, les applications sont de plus en plus vendues sur internet. Les formats des logiciels tendent donc à se diversifier par rapport au format CD. Ce qui complique la tâche des développeurs.

D'autre part, l'installation d'un logiciel est affectée par un autre paramètre, le système d'exploitation de l'utilisateur. Le processus d'installation peut être différent en fonction des systèmes.

Le test d'installation permet donc de vérifier que le logiciel peut être installé, désinstaller et réinstaller sur le ou les systèmes d'exploitation visés. Mais aussi partiellement ou entièrement. Par la suite, il consiste à établir une première vérification de l'état du logiciel après installation. Tout d'abord, vérifier les fichiers de configuration de l'application, puis l'état des données et enfin un test basique des fonctions de l'application.

Les projets de développement de logiciel sont généralement gérés d'après deux organisations différentes. Les types de test peuvent donc être utilisés différemment dans le cycle de développement.

# **4. L'organisation**

La méthode la plus utilisée depuis les années 1980 est la méthode du cycle en V.

## **a. Le cycle en V**

Il est une extension de la méthode en cascade. Elle démontre la relation entre la phase de développement du logiciel, phase utilisée dans la méthode en cascade, et la phase de test

du logiciel. Au lieu de descendre d'étape en étape comme dans la méthode en cascade, le processus remonte en associant chaque étape de développement à une étape de test.

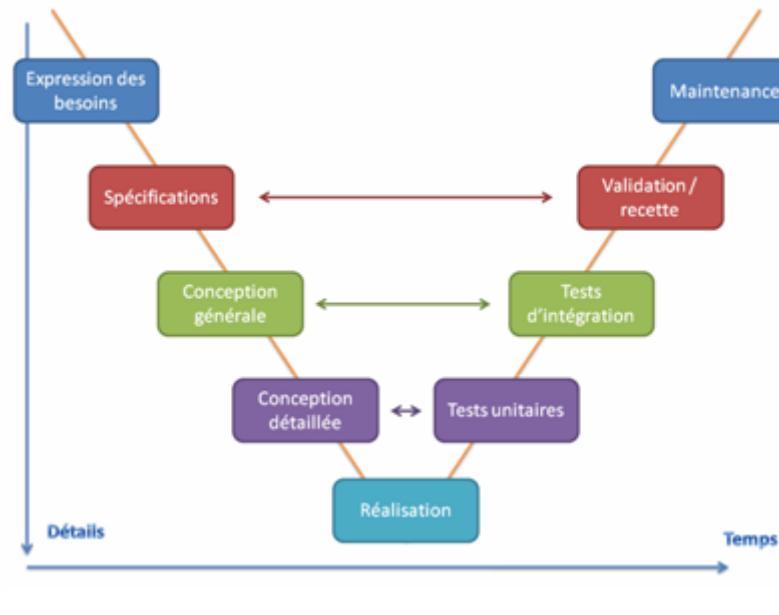


Schéma Cycle en V

Comme le montre le schéma précédent, le cycle débute par une analyse des besoins, suivie par une étape de spécifications. Ensuite, l'équipe conçoit l'architecture de l'application, suivie par une conception plus en détail. Enfin, le cycle de développement se termine par la programmation de l'application.

En parallèle à la conception en détail et après la programmation, les tests unitaires sont exécutés. Ensuite, en parallèle à la conception de l'architecture, l'équipe passe aux tests d'intégration. Les tests systèmes suivants correspondent à l'étape d'écriture des spécifications du cycle de développement. Enfin, l'équipe termine avec les tests d'acceptation et les Bêta tests.

Cependant, le modèle du cycle en V rencontre certains problèmes. En effet, en utilisant cette méthode, le cycle de développement ne peut pas gérer les changements apportés aux spécifications, changements presque inévitables dans une relation commerciale.

D'autre part, des études démontrent que l'on ne peut pas envisager de séparer les phases de tests unitaires et d'intégration de l'étape de programmation. En effet, les phases de tests requièrent plus de temps et donc plus d'argent.

C'est pour cela que de nouvelles méthodes se popularisent.

## b. L'agilité

Les méthodes agiles se sont propagées grâce à l'alliance agile et son manifeste agile écrit en 2001. Les membres de l'alliance ont défini les principes fondamentaux de l'agilité.

Le premier est l'attachement aux individus. C'est-à-dire les interactions avant les processus et les outils.

Il faut que le logiciel opérationnel. La documentation est moins importante.

D'autre part, la collaboration avec un client passe avant une négociation de contrat.

Et enfin, il faut s'adapter au changement plutôt que de suivre un plan.

En matière agile, chaque méthode organise de manière différente le cycle de développement du logiciel et donc le processus de test.

En général, le déroulement d'un projet dans les méthodes agiles s'effectue en plusieurs itérations. Une itération s'étend d'une à trois semaines. Chaque itération inclut le cycle de développement complet d'un programme dans la méthode de cycle en V. Elle se termine par une livraison du programme. Autrement dit, une itération se compose de l'analyse, de la conception, de la programmation, les tests et les tests d'acceptation. Les méthodes agiles utilisent en effet un processus d'intégration continue. À chaque modification du code source, on vérifie qu'aucune régression n'a été introduite.

Les développeurs travaillant en méthode agile utilisent donc constamment des tests unitaires et des tests d'intégration. De plus, ils conçoivent des tests d'acceptation en collaboration avec le client.

Le Test Driven Development (TDD), ou développement dirigé par les tests, est une pratique utilisée dans la méthode agile eXtreme Programming. Elle a été codifiée par le libre de Kent Benck intitulé "Test Driven Development : By Example". Comme son nom l'indique, elle consiste à écrire des tests avant d'écrire le code. Cette méthode de programmation est de plus en plus populaire au sein du secteur informatique.

Elle se base sur une répétition de cycles courts. Ces cycles couplent la programmation, l'écriture des tests unitaires et l'activité de remaniement. Elle suit les règles suivantes:

- créer un test unitaire décrivant un aspect du programme
- exécuter le test et vérifier qu'il échoue
- écrire le minimum de code pour que le test passe
- remanier le code pour qu'il soit le plus simple possible

Cette méthode de programmation présente certains avantages.

Tout d'abord, elle permet une meilleure couverture de tests. En écrivant les tests avant le code, on s'assure que la totalité ou presque de ce code est couvert par des tests.

Elle évite aussi la présence de documentation. En effet, il a été remarqué que les développeurs n'aiment pas lire de la documentation. Ils préfèrent lire le code. Or, par cette méthode, la documentation de l'application est remplacée par les tests. De plus, le développeur ne perd plus de temps à écrire une documentation. Il y a donc une réduction de coût.

D'autre part, elle engendre une meilleure architecture de logiciel. Effectivement, TDD inclut dans son cycle l'étape de remaniement de code. Ceci engendre un code plus propre. D'autre part, il existe une réelle réflexion avant chaque ajout de fonctionnalité pour trouver qu'elle est la meilleure façon de procéder d'après le code existant.

De plus, il a été constaté une augmentation de la productivité. Les études de déroulement des projets en utilisant cette méthode ont permis de mettre en avant une amélioration de la productivité de l'équipe. En effet, les développeurs préfèrent travailler sur des cycles courts. Ceci leur permet de visualiser rapidement leur avancée plus tôt que lors de longues phases.

Enfin, la conséquence de la mise en place d'une telle méthode est une réduction des coûts. Précédemment, les recherches sur le coût de correction de défauts dans un programme ont été évoquées. Ainsi, plus une erreur est identifiée tôt, moins le développeur passe de temps à la corriger et donc moins cette erreur coûte. Or, dans cette méthode, les problèmes sont trouvés pendant le cycle de développement.

Le grand nombre de types de test complexifie le cycle de test. De plus, ce dernier est très important dans le processus de développement. Or, l'automatisation de test facilite le travail des testeurs.

## **II. L'AUTOMATISATION DE TESTS**

La mise en place d'une automatisation des tests au sein de l'entreprise est une étape déterminante au sein d'une entreprise. Cette décision doit donc être mûrement réfléchie. Il est important de peser le pour et le contre avant de prendre cette résolution.

### **1. L'amélioration de la qualité**

Comme il l'a été évoqué précédemment, la qualité des logiciels informatiques nécessite d'être améliorée. Ceci car les défaillances logicielles ont un impact de taille sur l'économie mondiale et peuvent même en avoir sur les vies humaines. Certains apports de l'automatisation par rapport aux tests manuels tendent à répondre aux besoins de qualité logicielle.

#### **a. Moins d'erreurs humaines**

Le processus de test est considéré comme l'étape la plus difficile dans le cycle de développement d'un logiciel. En effet, cette tâche prend souvent plus longtemps à être réalisée que le développement du logiciel en lui-même.

La première raison pour laquelle cette étape est si longue est la complexité du code développé. Plus le projet est compliqué, plus le processus de test est complexe.

D'autre part, un testeur peut rencontrer des problèmes relatifs au manque de documentation du code. Or, lors d'un cycle de développement, il existe une certaine rotation des effectifs. Donc si ce manque de documentation a été réalisé par un développeur absent de l'entreprise, il y a impossibilité de comprendre le code.

De plus, si les personnes ayant programmé l'application doivent aussi la tester, ils ont donc une compréhension avancée du fonctionnement du logiciel. Or, le test d'un logiciel doit correspondre à une première expérience utilisateur. Les testeurs ne sont donc pas à même d'effectuer les mêmes erreurs que l'utilisateur lors du premier lancement du logiciel.

Enfin, le processus de test peut s'avérer répétitif. Une personne chargée d'effectuer les tests manuellement peut donc commencer à se lasser de son travail. Par conséquent, sa motivation s'en ressent et il ne fait donc plus attention aux détails. Ce comportement est dangereux pour le cycle de développement du logiciel puisque son manque d'attention

entraîne une perte de qualité au niveau de la zone testée. Or, les parties d'un logiciel sont liées entre elles. La perte de qualité s'étend donc au logiciel complet.

La diminution de l'implication humaine dans le processus de test permet aussi d'augmenter la profondeur de test.

### **b. Augmentation de la profondeur de test**

La profondeur de test est la mesure correspondante aux parties d'un code qui sont couvertes par un test. Couvrir complètement le code d'une application, c'est donc s'assurer qu'aucune défaillance n'est passée outre notre surveillance. L'absence de dysfonctionnement améliore donc la qualité d'un logiciel.

Il est prouvé qu'un logiciel gagne en complexité tout au long de son cycle de vie. De ce fait, le nombre de tests nécessaires pour couvrir l'application augmente avec le temps. De plus, si des changements sont apportés au code d'un logiciel, il est indispensable de le tester entièrement à chaque fois.

L'automatisation de test permet donc d'obtenir une meilleure profondeur de test comparée aux tests manuels. En effet, les tests longs et fastidieux sont souvent évités par les testeurs, comme il l'a été évoqué précédemment. Non seulement l'automatisation de test se charge d'exécuter ces tests, mais elle peut aussi le faire sur de multiples ordinateurs et différentes configurations. De plus, l'automatisation de test est capable de visualiser le comportement interne d'une application tel que les statistiques de mémoire, les données, le contenu des fichiers. Ceci afin de vérifier qu'ils se comportent comme ils le devraient. L'automatisation de test permet donc d'exécuter des milliers de cas de test tout en prodiguant une couverture impossible à égaler manuellement.

D'autre part, les testeurs qui ne perdent plus de temps à travailler sur des tests fastidieux peuvent se concentrer sur des tests plus complexes qui nécessitent une réflexion humaine.

L'automatisation de test permet aussi de réduire les coûts de développement d'un logiciel.

## 2. La réduction des coûts

Comme il l'a été évoqué précédemment, il existe un lien entre qualité et automatisation de test. Or, il est aussi possible de joindre qualité et bénéfices.

### a. Le lien entre qualité et bénéfices

Comme la définition de la qualité de Joseph M. Juran l'implique, non seulement le client achetant un produit de haute qualité y gagne, mais l'entreprise aussi.

Tout d'abord la qualité améliore la réactivité et l'innovation de l'entreprise.

L'innovation est un des facteurs principaux de la réussite d'une entreprise à s'adapter au changement et à se démarquer de ses concurrents. En ce qui concerne la création d'applications, un logiciel démontrant une grande flexibilité, et donc qualité, permettent aux entreprises d'apporter un changement rapide à son comportement en réponse à de nouveaux besoins exprimés. De plus, si une équipe de développeur est souvent focalisée sur la correction de problèmes rencontrés par le logiciel, ils n'ont donc plus de temps pour se concentrer sur l'innovation.

D'autre part, la qualité peut devenir un signe de différence d'une entreprise par rapport aux autres. Lorsqu'une entreprise produit des logiciels de meilleure qualité que ses concurrents, cela permet de la différencier aux yeux du client. Il est possible de citer par exemple les entreprises automobiles allemandes. La réputation de ces entreprises est basée sur les produits de qualité qu'elles conçoivent. Aujourd'hui, les voitures appartenant à ces marques sont considérées comme des produits de luxe dans le domaine automobile.

En 1987, dans son livre intitulé " Poor Quality Costs ", James Harrington définit les coûts qui disparaîtraient si le produit conçu était parfait.

Il divise les coûts en deux catégories distinctes.

La première contient les coûts directs de la pauvre qualité. Elle aussi est divisée en deux sous-catégories.

Tout d'abord, les coûts contrôlables.

Ceux-ci contiennent tout d'abord la prévention à la pauvre qualité. C'est-à-dire tous les coûts impliqués dans l'aide apportée aux employés pour réaliser un travail de qualité. Il est possible de citer par exemple les formations payées à ces employés.

Ensuite, parmi les coûts contrôlables, il ajoute les coûts d'évaluation. Autrement dit, les coûts relatifs à la vérification du travail fourni. Par exemple, inspecter le travail réalisé.

Une autre sous-catégorie contient les coûts résultants d'une faible qualité.

Celle-ci comporte tout d'abord les coûts internes des erreurs. C'est-à-dire les erreurs trouvées avant la vente du produit. Par exemple, devoir tester à nouveau un logiciel.

D'autre part, cette sous-catégorie contient les coûts externes des erreurs comme la gestion du support utilisateur. Autrement dit, les coûts qui arrivent après la vente du produit car il a été livré au client comportant des erreurs.

La deuxième catégorie concerne les coûts indirects de la pauvre qualité.

Elle est divisée en trois sous-catégories.

Tout d'abord, les coûts relatifs au client. C'est-à-dire les inconvénients pour le client survenus à cause du manque de qualité du produit qu'il a acheté. Il est possible de citer par exemple son manque de productivité en utilisant le logiciel pour travailler.

Il existe aussi la sous-catégorie relative aux coûts d'insatisfaction du client. En effet, un client peut par exemple exprimer son mécontentement. Ce qui peut freiner les ventes d'un produit.

Enfin, les coûts de la perte de réputation aux yeux des clients. L'insatisfaction des clients, comme il l'a été évoqué précédemment, peut engendrer une perte de réputation en fonction de son importance. Or la réputation d'une entreprise est un facteur important sur les ventes de produit.

D'autre part, les corrections de défaillances logicielles ont un coût pour l'entreprise qui le conçoit. Plus un dysfonctionnement est détecté tard, plus il coûte cher à l'entreprise. En effet, une recherche de Caper Jones en 2000 intitulée " Software Assessments, Benchmarks, and Best Practices " a démontré qu'un bug décelé dans la phase de spécification coûterait 139 dollars, lors de la conception 456 dollars, 977 dollars pendant la programmation, 7 136 dollars dans la phase de test et enfin 14 103 dollars après livraison du logiciel. Il y a donc

effectivement un accroissement exponentiel du prix de correction pendant le cycle de développement.

Il y a donc véritablement un lien entre l'absence de défaillances, donc qualité et réduction des coûts pour l'entreprise.

Il est connu dans le monde de l'entreprise que " le temps, c'est de l'argent ". Or, l'automatisation permet de réduire le temps passé par les testeurs à travailler sur une application.

### **b. Une économie de temps**

L'automatisation de test réduit le temps passé à effectuer certaines étapes du processus de test.

Tout d'abord, l'automatisation permet de diminuer le temps pris pour générer les données de test.

Avant de procéder à la phase de test de l'application, un testeur doit préparer des données adéquates. Cette étape requiert un travail conscient. La réussite des phases suivantes dépend de ces données. Ces données doivent être en corrélation avec les spécifications métier. La préparation des données inclut des données brutes ou dans des fichiers, des vérifications de la cohérence de ces données par rapport aux spécifications mais aussi la liaison entre ces données et les cas de test. Ce travail peut donc s'avérer fastidieux et long s'il est effectué manuellement. Tandis que l'automatisation peut effectuer cette tâche rapidement.

Par la suite, les testeurs passent à l'étape d'exécution des tests. Cette étape est aussi fastidieuse que la phase précédente si elle est effectuée manuellement. Or, les frameworks d'automatisation de test permettent de lancer les scripts de test à chaque exécution avec une très faible interférence manuelle. En effet, les outils d'automatisation de tests permettent la sélection et l'exécution de test avec seulement un clic sur une icône. Cela peut prendre seulement quelques secondes.

Ensuite vient la procédure d'analyse des résultats des tests. Les outils d'automatisation fournissent des fonctionnalités de rapport et de log des informations. Certains affichent les différents résultats avec des couleurs rouges ou vertes en fonction du succès du test. Ces

fonctionnalités améliorent l'analyse des résultats. De plus, il est possible de comparer les résultats obtenus avec les résultats espérés.

Enfin, certains outils d'automatisation de test permettent de documenter les erreurs qu'ils ont pu découvrir avec une très faible intervention humaine. La documentation inclut l'identification du script de test défectueux, le cycle de test qui était exécuté, une description de l'erreur et la date à laquelle il l'a trouvée. Cette fonctionnalité est importante dans le cycle de test car elle permet aux testeurs de comprendre pourquoi ces défauts existent.

Cependant, la mise en place d'un système d'automatisation de test au sein d'une entreprise peut s'avérer coûteuse. Avant de procéder à une telle dépense, il est donc nécessaire de calculer le retour sur investissement qui sera obtenu après installation du système.

### **3. Le calcul du gain**

Différentes méthodes de permettront à l'entreprise de calculer l'apport en matière économique de la mise en place de l'automatisation de test.

#### **a. Le retour sur investissement**

Le retour sur investissement (ROI: Return On Investment) est calculé par la division entre les bénéfices acquis et l'investissement réalisé par un objet. Le but est de voir le retour sur investissement à long terme. L'analyse du ROI de la mise en place de l'automatisation de test permet de déterminer une approximation de son coût. Ceci afin de justifier un tel effort.

La formule générale du calcul d'un ROI est la suivante:

$$ROI = \frac{Gains - Investissements}{Investissements}$$

Lorsque la mise en place concerne un nouveau projet, il faut diviser la valeur qu'apportent les tests automatisés par le coût de l'essai de l'installation pour calculer le ROI.

Parfois, l'automatisation est mise en place après les tests manuels. Dans ce cas-là, qui arrive le plus souvent, il faut comparer l'automatisation au système manuel actuel.

Pour calculer le ROI de l'automatisation de test, il faut comprendre en compte différents facteurs. Certains sont fixes tandis que d'autres sont variables.

En ce qui concerne les coûts fixes, il faut tout d'abord calculer le coût matériel, que ce soit le nouveau matériel nécessaire ou la mise à jour de l'existant. D'autre part, il faut analyser le prix des licences des différents outils nécessaires à l'automatisation ainsi que celui du support utilisateur après l'achat de ces logiciels. De plus, il faut estimer le coût de la conception de l'environnement d'automatisation de test et celui de l'installation de cet environnement. Ceci ajouté au coût de maintenance de cet environnement. Enfin, le temps pris à former les utilisateurs sur les différents outils doit être pris en compte.

Les coûts variables quant à eux englobent le temps pris à créer les cas de tests. Que ce soit l'étape de conception, implémentation, l'exécution, la maintenance et l'analyse des résultats de ces tests. De plus, le coût de mise en place de l'oracle peut être aussi variable. L'oracle est le système permettant de vérifier si le programme a passé les tests.

D'autre part, il est nécessaire de fixer une variable de temps. Cette variable est liée à la durée après laquelle l'entreprise espère un retour sur investissement. Généralement, cette variable doit correspondre à la prochaine date à laquelle le logiciel doit être livré. En effet, cette date correspond au moment où l'entreprise commence à tirer profit du lancement du logiciel.

L'avantage principal de l'automatisation de test est la diminution de coût d'exécution des tests. Deux variables doivent donc être aussi prises en compte dans le calcul du ROI. La première est le nombre d'exécutions de test avec un système d'automatisation de test mis en place, mais aussi le nombre d'exécutions sans ce système. Il paraît évident que la première variable est presque toujours supérieure à la deuxième.

Enfin, il est important de prendre en compte le coût du personnel travaillant sur le cycle de test. Cependant, cette donnée est difficilement quantifiable.

Il existe différentes formules de calcul d'un ROI.

La première est la méthode " Simple ROI ". Elle se base sur l'économie tirée de l'exécution des tests. Elle est adéquate lorsque l'on souhaite diminuer les coûts de l'étape de test.

Le coût total de l'investissement est calculé d'après la formule suivante:

*Coût de l'investissement*

- = *Coût du matériel + Coût de la formation du personnel*
- + *Coût du temps de développement des tests*
- + *Coût du temps de l'analyse des tests*
- + *Coût du temps de maintenance des tests*

où:

*Coût du temps de développement des tests*

- = *Moyenne de temps passé à développer un test par heure*
- \* *Nombre de tests à développer \* Prix du personnel par heure*

*Coût du temps de l'analyse des tests*

- = *Temps en heures d'analyse des tests par semaine*
- \* *Variable de temps définie par semaine*
- \* *Coût du personnel par heure*

*Coût du temps de maintenance des tests*

- = *Temps en heures de maintenance des tests par semaine*
- \* *Variable de temps définie par semaine*
- \* *Coût du personnel par heure*

Le gain correspond au coût des tests manuels car après la mise en place de l'automatisation, ils n'existeront plus. Il est défini par la formule suivante:

$$\text{Gain} = \text{Temps en heure pris pour exécuter un test manuellement}$$

- \* *Nombre de tests à exécuter*
- \* *Variable de temps sélectionnée en semaines*
- \* *Prix du personnel par heure*

L'avantage de cette méthode de calcul est que le type de résultat obtenu est exprimé en termes monétaires. Il permet donc de justifier facilement auprès de la hiérarchie le retour sur investissement obtenu après mise en place du système d'automatisation.

Cependant, elle présente certains inconvénients. En effet, la donnée correspondante au prix du personnel nécessaire par heure est difficilement quantifiable. Elle peut donc être erronée. D'autre part, cette formule prend en compte que tous les cas de tests manuels seront remplacés par l'automatisation, ce qui n'est pas toujours le cas. Enfin, cette formule ne contient pas l'augmentation de profondeur de test, apport de l'automatisation qui a été cité précédemment.

Une autre méthode appelée " Rendement ROI " est basée sur le " Simple ROI " mais se focalise sur l'investissement et le gain en termes de temps et non en termes monétaires. Ceci pour exprimer le rendement du retour sur investissement. Ce calcul est utilisé lorsque les projets utilisent déjà l'automatisation depuis un certain temps et qu'il n'y a plus besoin de justifier un tel investissement auprès de la hiérarchie. D'autre part, les facteurs utilisés dans la formule sont plus accessibles et donc moins erronés.

Contrairement à la méthode précédente, les données sont exprimées en temps, il faut donc les exprimer en jours et non pas en heures. Ceci car les tests manuels ne sont travaillés que 8 heures par jour, puisque c'est la durée de travail des ingénieurs par jour. Tandis que les tests automatiques peuvent être exécutés pendant 18 heures par jour s'ils sont arrêtés durant la nuit ou 24 heures à l'idéal sans cet arrêt.

La formule de calcul de l'investissement est donc la même que dans la méthode précédente à laquelle on ajoute le temps d'exécution des tests automatiquement et le temps d'exécution des tests manuellement. En effet, ces données n'étaient pas incluses dans le " Simple ROI " car l'exécution des tests s'effectue indépendamment sur une machine d'automatisation. Cependant, le calcul des facteurs est modifié pour correspondre à une donnée en temps.

$$= \frac{\text{Coût du temps de développement des tests}}{\text{Moyenne de temps passé à développer un test par heure} * \text{Nombre de tests à développer}}$$

$$= \frac{\text{Coût du temps de l'analyse des tests}}{\text{Temps d'analyse des tests par semaine} * \text{Variable de temps définie en semaine}}$$

*Coût du temps d'exécution des tests*

$$= \frac{\text{Temps d'exécution d'un test en heures} * \text{Nombre de tests à exécuter par semaine}}{18}$$

*\* Variable de temps définie en semaine*

*Coût du temps de maintenance des tests automatiquement*

$$= \frac{\text{Temps de maintenance des tests par semaine} * \text{Variable de temps définie en semaine}}{8}$$

*Coût du temps d'exécution des tests manuellement*

$$= \frac{\text{Temps d'exécution d'un test en heures manuellement}}{8}$$

*\* Nombre de tests restant à exécuter manuellement par semaine*  
*\* Variable de temps définie en semaine*

En ce qui concerne le gain, comme dans la méthode précédente, il faut calculer le coût des tests manuels sans automatisation:

$$\text{Gain} = \frac{\text{Temps d'exécution d'un test en heures manuellement}}{8}$$

*\* Nombre total de tests à exécuter*  
*\* Variable de temps définie par semaine*

L'avantage de cette méthode est que la variable d'estimation du prix du personnel par heure est enlevée. Ainsi, elle peut s'avérer plus fiable. Cependant, cette approche contient toujours l'inconvénient de la méthode de " Simple ROI ", le postulat selon lequel tous les tests manuels seront remplacés par l'automatisation.

Une troisième méthode de calcul a été définie pour pallier aux facteurs absents des approches précédentes. Contrairement à celles-ci, la méthode de " Réduction du risque " cherche à calculer le ROI en se basant seulement sur les bénéfices de l'automatisation indépendamment des tests manuels.

Comme il l'a été évoqué précédemment, l'automatisation de test permet d'améliorer la qualité d'un produit, et donc réduire le nombre de défauts. Le gain doit donc être calculé en estimant qu'une erreur a été trouvée sur le logiciel après sa mise en production. Il doit donc

prendre en compte le coût de correction de ce défaut mais aussi l'impact que cela aurait sur la relation client comme l'abandon de l'utilisation de l'application. Ce facteur doit donc être estimé par des personnes ayant un niveau hiérarchique élevé.

En ce qui concerne l'investissement, il est calculé de la même manière que dans la méthode " Simple ROI ".

L'avantage de cette méthode de calcul est qu'elle n'essaie plus de comparer les tests manuels et l'automatisation de test et ne cherche qu'à montrer les avantages de la mise en place du système d'automatisation.

Cependant, l'inconvénient majeur est que le gain est une valeur estimée et subjective. Il est impossible de connaître la perte exacte qu'un défaut provoquerait. Le calcul nécessite donc un travail minutieux et long. D'autre part, s'il n'existe plus la comparaison avec les tests manuels, il devient compliquer de justifier la mise en place de l'automatisation auprès de la hiérarchie qui se demanderait si ce système est la meilleure solution à appliquer. Elle pourrait par exemple estimer qu'il serait plus judicieux d'allouer plus de ressources aux tests manuels.

Pour conclure, le choix d'un type de calcul doit se faire en fonction de son adéquation à l'environnement actuel. Aucune des trois méthodes de calcul ne se démarque des autres et elles ont chacunes des avantages différents à apporter. Le choix doit donc suivre une réflexion liée à ce qui s'applique le mieux au sein de l'entreprise.

D'autre part, ces calculs démontrent que le ROI de l'automatisation de tests est étroitement lié au nombre de cas de tests qui doivent être exécutés. Autrement dit, une quantité importante de cas de tests justifie la mise en place d'un tel système.

Le calcul du ROI sert à justifier l'investissement dans une procédure d'automatisation de tests. Il existe cependant d'autres méthodes de calcul qui permettront de vérifier que l'automatisation de test apporte de la valeur après son installation.

## b. Les métriques

Il est compliqué de quantifier l'apport de l'automatisation de test. Les métriques d'automatisation de test permettent de mesurer la performance du système d'automatisation mis en place dans l'entreprise.

Il existe différents types de métriques.

Le premier correspond au pourcentage de couverture de tests. Ce calcul permet d'associer un chiffre au nombre de fonctionnalités de l'application qui sont couvertes par les tests.

Il se calcule de la manière suivante:

$$\text{Pourcentage couverture} = \frac{\text{Couverture de l'automatisation}}{\text{Couverture totale}}$$

Dans ce calcul, la couverture totale peut être égale au nombre de lignes de code de l'application. Cette approche est la plus commune. Mais elle peut aussi être calculée avec une autre méthode. Celle-ci s'appelle " les points de fonctions ". Elle consiste à quantifier les fonctionnalités incluses dans le logiciel d'après les spécifications du client.

D'autre part, il existe le calcul du ratio de défauts. Il correspond au pourcentage de nombre de défauts connus par rapport à la taille du système.

$$\text{Ratio de défauts} = \frac{\text{Nombre de défauts}}{\text{Taille du système}}$$

Cette fois encore, la taille du système est liée au nombre de lignes de code ou aux points de fonctions décris précédemment.

Cette métrique peut être aussi utilisée pour valider une livraison de l'application comme indicateur de la qualité produite.

Si le résultat est mauvais, les testeurs devront fournir des indications aux développeurs afin que les défaillances du système soient corrigées.

Une autre métrique importante est le calcul de l'efficacité de correction des défauts. Il sert à calculer le pourcentage des défauts qui ont été trouvés durant la phase de test et qui ont été corrigés.

Il se calcule de la manière suivante:

$$Efficacité correction = \frac{Nombre de défauts trouvés pendant les tests}{Nombre de défauts trouvés pendant les tests + Nombre de défauts trouvés après livraison}$$

Idéalement, il doit être proche de 100% pour valider la qualité du travail des testeurs. En effet, cela signifierait qu'aucune défaillance du système n'a été repérée après sa livraison.

Enfin, d'autres métriques permettent de valider l'effort de mise en place de l'automatisation de tests. Celles-ci correspondent à la mesure de la productivité des testeurs. En effet, il existe différentes métriques telles que le nombre de défauts trouvés pour 100 heures de test, le nombre de cas de test exécutés pour 100 heures de test ou encore le nombre de cas de test développés pour 100 heures de test.

Avant et après l'installation du système d'automatisation, il est important d'effectuer ces différents calculs pour comparer les résultats obtenus et ainsi vérifier que l'effort fourni apporte une productivité satisfaisante.

L'automatisation de test est un système complexe. Une fois que l'entreprise a décidé d'investir dans ce système, elle doit effectuer différentes étapes préalables à son installation.

# **III. MISE EN OEUVRE**

## **1. Choisir quoi automatiser**

### **a. Choix des cas de tests**

Il est impossible d'automatiser tous les cas de test. Il est important de déterminer quels sont les cas de test qui doivent être automatisés en priorité afin de réduire l'effort manuel. Le coût est donc diminué et la qualité améliorée. On peut ainsi obtenir un meilleur ROI.

L'avantage de l'automatisation est en partie lié au nombre de fois qu'un cas de test doit être exécuté. Chaque cas de test devant être exécuté de manière répétitive doit donc être automatisé.

D'autre part, il faut sélectionner les cas de test à haut risque ou critiques. C'est-à-dire étant liés aux fonctionnalités primaires de l'application. En effet, l'automatisation diminue le nombre d'erreurs humaines. Automatiser les fonctionnalités critiques permet donc de s'assurer que celles-ci fonctionnent correctement.

Il est important que l'automatisation de test permette de faciliter le travail des testeurs et développeurs. C'est pour cela que les cas de test fastidieux et difficile à effectuer manuellement doivent être automatisés. Ainsi, on diminue le risque d'erreurs humaines et la perte de temps.

Si les tests compliqués sont automatisés, il convient donc d'automatiser les tests d'un degré de complexité plus élevé et donc impossibles à gérer manuellement.

Lors de la phase de test d'un produit, il peut arriver, en fonction des besoins, que les tests soient exécutés sur différents environnements ou différentes configurations. L'automatisation est nécessaire sur ces cas de test car le testeur peut perdre du temps à modifier ces cas de test pour chacune des configurations.

Certains cas de tests peuvent utiliser des données de test complexes. En effet, pour tester chaque cas possible, un nombre important de jeux de données est nécessaire. Dans ce cas-là, le travail manuel est fastidieux. Il est donc préférable d'installer une automatisation pour ces tests.

Enfin, l'automatisation permet de minimiser la consommation de ressources. Ainsi, les cas de test nécessitant plusieurs ressources doivent être automatisés pour tirer avantage de cette automatisation.

Afin d'obtenir le meilleur retour sur investissement, il faut donc automatiser en priorité les cas de test qui consomment le plus de temps et de ressources.

En revanche, certains cas de test n'ont pas besoin d'automatisation. La difficulté de mise en place comporterait plus d'inconvénients que d'avantages apportés par la suite.

Il est important d'exécuter au moins une fois un cas de test manuellement avant de l'automatiser. Ceci afin de pouvoir identifier les besoins en terme d'automatisation.

D'autre part, il est compliqué d'automatiser des cas de test dont les besoins changent fréquemment et donc qui sont modifiés à chaque fois.

L'automatisation nécessite une certaine organisation. Il n'est donc pas possible d'automatiser des cas de test effectués de manière ad hoc, c'est-à-dire sans planification ou documentation, et aléatoires.

Comme il l'a été évoqué précédemment, l'avantage de l'automatisation est lié à la répétition. Il paraît donc inutile d'automatiser les cas de tests sont voués à n'être exécutés qu'une seule fois.

Enfin, il paraît logique que les cas de test ne pouvant être effectués que manuellement ne puissent pas être automatisés.

## b. Choix de la portée

L'automatisation de test ne permet pas d'automatiser tous les types de test. En effet, il faut sélectionner les types dont l'automatisation apporterait une valeur ajoutée au processus de test au sein de l'entreprise.

Tout d'abord, les types de test non fonctionnels tels que les tests de stress et de performance nécessitent un grand nombre de cas de tests. En effet, le but de ces tests est de vérifier que l'application fonctionne correctement lorsqu'elle subit des augmentations de charge. Pour cela, il est important d'exécuter de multiples cas de tests. Ceux-ci doivent être exécutés pendant une longue durée comme 24 ou 48 heures sur différentes machines. Or, les premiers avantages de l'automatisation sont la rapidité d'exécution et la facilité d'exécution sur différentes configurations. D'autre part, elle peut, contrairement aux tests manuels qui sont effectués pendant les heures de travail des ingénieurs, exécuter des cas de tests sur de longues durées. Il devient donc nécessaire de choisir ces types de tests comme premiers candidats à l'automatisation.

De plus, les tests de régression sont répétitifs par nature. En effet, ils consistent, comme il l'a été évoqué dans la première partie, à vérifier qu'un programme fonctionne toujours après chaque modification du code. Pour cela, une sélection de cas de tests doit être exécutée après chacune des modifications. Comme il l'a été évoqué précédemment, la répétition entraîne un manque de motivation lorsqu'elle est subie par une personne, et peut donc engendrer des erreurs. Or, un des avantages de l'automatisation est sa capacité à exécuter des cas de tests de manière répétitive. En effet, elle ne nécessite aucune configuration supplémentaire pour exécuter des tests à chaque modification du code. Il est donc important d'automatiser les tests de régression en priorité.

Enfin, il convient de sélectionner les tests fonctionnels en général. En effet, ces types de tests sont complexes à exécuter. Ils nécessitent donc une compétence d'expertise de la part des ingénieurs, compétence difficile à acquérir. Lors du cycle de développement d'un logiciel, il peut survenir une rotation des effectifs. L'entreprise ne peut donc pas se permettre l'absence de ressources nécessaires à la création de ces tests fonctionnels. Or, l'automatisation de tests requiert des compétences minimales en ce qui concerne l'exécution de ces tests. Ceci une fois

qu'ils ont été conçus préalablement par des ressources ayant les compétences requises. Il est donc nécessaire d'automatiser les tests fonctionnels.

D'autre part, l'entreprise doit automatiser les zones qui ne sont pas destinées à être modifiées fréquemment. En effet, l'automatisation de tests est un système complexe et son premier but est d'économiser le temps des testeurs. Il est donc nécessaire d'éviter à ces testeurs de devoir modifier constamment les cas de tests à exécuter. Ceci afin d'obtenir le meilleur ROI possible après la mise en place du système. Il est possible d'évoquer par exemple l'interface graphique de l'application. Celle-ci est vouée à évoluer constamment en fonction de l'ajout de fonctionnalités sur le logiciel. Cette portion de l'application ne doit donc pas être automatisée en priorité.

## 2. Choisir les outils

Le succès de l'automatisation de test dépend du choix opportun des outils. Choisir les bons outils peut prendre du temps. En effet, de nombreux produits sont disponibles sur le marché. Mais cette étape est importante puisque du choix des outils dépend le futur profit engendré par l'automatisation à long terme.

### a. Choix de l'outil d'automatisation

Avant de sélectionner un outil d'automatisation, il faut tout d'abord réfléchir à certains critères relatifs à l'entreprise et aux projets.

La plus grande préoccupation au sein d'une entreprise est le coût du produit. Il est donc important d'identifier le budget alloué à cet achat. Ceci afin de choisir une solution qui sera comprise dedans.

Idéalement, il faut que l'outil propose une période d'évaluation. Cela permettra à l'entreprise de vérifier qu'il correspond aux besoins des projets et par la suite pouvoir continuer avec un produit satisfaisant ou changer si besoin est.

Il faut aussi vérifier que le produit soit accompagné d'un support utilisateur tel que l'aide en ligne ou un manuel. Une fois l'outil acheté, il est important de pouvoir poser des

questions à l'éditeur en cas de problème. Si les réponses sont rapides, ceci peut faire économiser du temps et donc de l'argent à l'entreprise.

L'entreprise doit se renseigner sur le temps d'apprentissage du fonctionnement de l'outil. En effet, un outil puissant peut s'avérer trop complexe et donc difficile d'utilisation. Les utilisateurs peuvent perdre du temps à trouver les fonctionnalités dont ils ont besoin.

Enfin, un des critères les plus importants du produit est sa capacité de répondre aux besoins du projet.

Il faut donc analyser les différentes fonctionnalités incluses à l'outil pour vérifier qu'il répond aux attentes.

La première fonctionnalité à étudier est sa capacité d'adaptation au projet à automatiser et à de futurs projets.

Tout d'abord, l'outil doit pouvoir être utilisé sur de multiples systèmes d'exploitation. Ceci afin de fonctionner sur le projet actuel de l'entreprise mais aussi sur de futurs programmes dont on ne connaît pas encore les spécificités. Cependant, il n'est pas nécessaire que l'outil puisse marcher sur des systèmes que l'entreprise n'utilise pas ou n'a pas vocation à utiliser par la suite. En effet, un logiciel peut perdre en performance s'il a été développé dans des langages destinés à être utilisables sur tous les systèmes. Il faut donc trouver une solution intermédiaire qui satisfairait les besoins systèmes et la performance.

Ensuite, il est important de vérifier que l'outil supporte les types de test, évoqués précédemment, qui ont été choisis pour l'automatisation. Il pourrait sembler évident de choisir un outil qui permettrait d'automatiser tous les types de test. Cependant, il faut penser au souci d'apprentissage du logiciel. S'il propose trop de fonctionnalités, le coût de formation des utilisateurs peut être trop élevé. Une fois de plus, il faut considérer les types de test nécessaires dans le présent mais aussi dans le futur avant de choisir un outil.

D'autre part, l'outil doit comporter différents types de données d'entrée. En effet, pour tester une application, il faut pouvoir simuler des entrées utilisateur. Le choix d'un outil doit donc s'orienter vers celui qui peut gérer un grand nombre de types d'entrée. Il doit tout d'abord contenir les types nécessaires aux projets actuels devant être automatisés. Mais aussi des types

supplémentaires qui pourront être utilisés dans des projets futurs. Il est difficile de prévoir quels seront les types indispensables pour les futurs projets. Il est donc important de choisir le logiciel qui en supporte le plus.

Le travail d'équipe est un facteur important de la réussite d'un projet. De ce fait, il doit faire partie intégrante des fonctionnalités disponibles sur un outil de travail. Le choix d'un logiciel d'automatisation doit donc aussi s'orienter vers cette possibilité. Il faut donc choisir un outil qui supporte le travail d'équipe avec la possibilité de travailler sur le même fichier, partager les scripts de test ou encore le contrôle de code source.

De plus, l'outil doit contenir la possibilité de comparer les résultats des tests exécutés. Ainsi que l'export de ces résultats sur des fichiers externes. En effet, il est important que l'utilisateur puisse visualiser rapidement les différences entre de multiples exécutions.

Enfin, il est indispensable que le logiciel d'automatisation soit capable d'interpréter les différents contrôles graphiques qui peuvent être contenus dans le projet à automatiser. Tels que les listes déroulantes, les cases à cocher et les champs de texte.

## b. Choix du framework de test

Un framework d'automatisation de test est un environnement d'exécution pour les tests. Il est le système dans lequel les tests sont automatisés. Le framework est responsable de la définition du format dans lequel exprimer les attentes, se raccorder à l'application testée, exécuter les tests et rapporter les résultats.

Cependant, un framework de test est indépendant d'une application et doit être facilement maintenable et extensible.

L'avantage principal de l'utilisation d'un framework de test est lié à son indépendance par rapport à une application. Ainsi, chaque projet est porté sur le même environnement. Il n'y pas de modifications à effectuer. Grâce à sa capacité d'extension, le framework peut être modifié pour répondre aux besoins spécifiques à chaque projet.

Il existe différentes caractéristiques indispensables à un framework d'automatisation de test.

Certains frameworks d'automatisation comportent des classes qui contiennent beaucoup de fonctionnalités. Ce qui complique la maintenance des tests sur la durée. En effet, le code de test devient très complexe lorsque le nombre de tests augmente et que chaque condition est gérée. Il est donc important de choisir un framework qui fournit une grande flexibilité aux développeurs. Ils doivent pouvoir choisir les classes et modules qu'ils souhaitent utiliser lors de la création des cas de test.

D'autre part, une des caractéristiques indispensables d'un framework de test est la séparation entre exécution et validation des cas de test. Le code de validation doit être indépendant du code d'exécution. Ceci permet de concevoir un code moins complexe et plus facilement maintenable.

Il existe quatre types de framework d'automatisation de tests. Le choix d'un de ces types dépend de la taille et la complexité du projet.

Tout d'abord il existe le type modulaire. Il est basé sur l'abstraction et donc sur la création de scripts indépendants les uns des autres qui représentent, comme son nom l'indique, les modules de l'application testée. Ces modules sont liés par la suite pour créer de larges cas de test.

L'avantage de ce type est la facilité de maintenance des scripts grâce à leur division.

Cependant, le problème avec ce type de framework est que le script de test embarque de larges données de test. Par conséquent, si ces données doivent être modifiées, il faut donc modifier le code du script. Et plus le script est long, plus il devient difficile de le modifier.

Ce type de framework est donc adapté à de larges et stables applications.

Le second type de framework est le framework focalisé sur les données. Dans celui-ci, les entrées et sorties des tests sont stockées dans des fichiers externes. Un script de conduite des données est créé pour lire les données de test.

Le premier avantage de ce type est qu'il réduit le nombre de scripts nécessaires à l'exécution de tous les cas de test.

D'autre part, moins de code est nécessaire pour la génération de tous les scripts.

De plus, il permet une grande flexibilité lorsqu'il s'agit de maintenir le code et corriger les bugs.

Enfin, il est possible de créer les données de test avant que le système de test soit prêt.

Mais il existe un inconvénient. En effet, pour créer de nouveaux types de test, il faut programmer un nouveau script de conduite des données puisque les scripts de conduite et les données sont liés.

Ce type de framework est donc adapté aux applications qui sont toujours en cours de développement.

Un autre type de framework est le framework orienté mots-clé. Il est indépendant d'une application et utilise des données organisées en tableaux dans des fichiers externes. Ce framework tient son nom de mots-clés stockés eux aussi dans des fichiers externes qui sont liés à des actions effectuées sur l'application testée. Ces mots-clés dirigent les scripts de test. Ce framework est une extension du framework précédemment évoqué.

Ce type framework présente certains avantages.

Tout d'abord, étant basé sur le framework précédent, il présente donc les mêmes avantages que celui-ci qui ont été évoqués.

De plus, il ne nécessite pas d'expertise en automatisation de test. L'entreprise peut donc alterner les ressources disponibles pour créer ou éditer les cas de test.

Les mots-clés sont réutilisables pour chaque cas de test. Il n'a donc pas de perte de temps et donc les coûts n'augmentent pas.

En revanche, il comporte aussi des inconvénients.

Ce framework ajoute des fonctionnalités au framework orienté données. Par conséquent, il devient plus compliqué que celui-ci.

D'autre part, il apporte une certaine flexibilité aux testeurs. De par ce fait, les cas de test deviennent plus longs et plus complexes.

Ce type est donc utilisé plus particulièrement pour les applications de moindre importance.

Pour pallier aux inconvénients des deux frameworks précédents, un framework de type hybride a été conçu. Il est donc une combinaison de ces deux frameworks et cherche plus particulièrement à tirer le meilleur de ceux-ci. Il utilise les scripts orientés données et leurs

permet de fonctionner avec les librairies qui accompagnent le framework orienté mots-clé. Par conséquent, les scripts de données sont moins complexes.

L'avantage d'un tel framework est qu'il englobe les avantages des deux frameworks précédents.

Cependant, il devient donc plus complexe que ces deux frameworks.

Grâce à ces arguments, il est le type le plus utilisé en automatisation de tests. Il peut s'appliquer à des applications de taille moyenne à large et à longue durée de vie.

Lorsque l'entreprise a choisi les deux outils précédemment évoqués les plus aptes à répondre à ses besoins, elle peut donc commencer le processus d'automatisation de tests.

## 3. Le processus

Le succès d'une automatisation de tests dépend tout d'abord de l'équipe de testeurs chargée de son fonctionnement.

### a. L'équipe

Il est important de former une équipe dont les connaissances concordent avec l'automatisation de test. Par exemple, il faut sélectionner des ressources étant capables de développer des cas de tests. De plus, chaque ressource doit être capable de gérer les responsabilités qui lui ont été données au sein de l'équipe.

Le groupe de testeurs doit être dirigé par un chef d'équipe. Il a pour responsabilité de gérer l'équipe et la coordonner par rapport aux autres équipes du cycle de développement en fonction du plan de test qu'il aura créé. Pour gérer le groupe, il a l'autorité nécessaire pour assigner chaque testeur à une tâche qu'il a choisie.

D'autre part, l'équipe est composée de développeurs de tests. Ils sont responsables du développement des cas de tests, de leur exécution, de l'analyse et du rapport des résultats. Ils doivent avoir connaissance des fonctionnalités incluses dans l'application.

Le groupe est aussi composé de développeurs de scripts. Ils sont responsables du développement et de la maintenance du framework de test. Ils doivent connaître les outils de test et avoir des connaissances en programmation.

L'équipe doit inclure un responsable de la librairie de test. Il est responsable de la configuration et du contrôle de version de la librairie de test.

De plus, il doit y avoir une liaison avec le client. Ceci pour s'assurer que les tests suivent les critères d'acceptation de l'application. Il représente donc l'utilisateur final de l'application et est chargé d'accepter ou non le plan de test. Il aide aussi les développeurs sur leur travail des cas de tests et plus particulièrement sur les données à utiliser.

Il est nécessaire d'ajouter une liaison de développement. Cette liaison représente les programmeurs de l'application qui sont chargés d'avertir le responsable de la librairie d'un changement au niveau du comportement du logiciel ou de son environnement. Ils doivent aussi fournir les tests unitaires qu'ils utilisent pendant le développement.

Enfin, l'équipe doit comporter une liaison système. Cette partie de l'équipe est composée des administrateurs du réseau et des bases de données. Ils doivent s'assurer que l'équipe de test a accès à la plateforme de test et aux données relatives aux tests. De plus, il existe une liaison entre cette partie et le responsable de la librairie. En effet, il est important que les administrateurs avertissent ce responsable de tout changement en ce qui concerne la plateforme de test.

Une fois l'équipe constituée, un plan d'automatisation de tests doit être réalisé.

## **b. Le plan d'automatisation de tests**

Le plan est composé des différentes étapes nécessaires pour mettre en place l'automatisation de tests. C'est-à-dire les composants de la librairie de test, les ressources nécessaires, l'organisation dans le temps et les critères d'entrée et sortie nécessaires au passage à l'étape suivante. Ce plan est mis à jour régulièrement durant la phase d'automatisation en fonction des progrès effectués.

Il comprend notamment un contrôle de version. Autrement dit, à chaque fois que le plan de test est modifié, une trace est gardée pour permettre à l'équipe de se repérer.

D'autre part, il contient une description de l'application testée et notamment la version actuelle.

Une définition de la portée de l'automatisation est aussi incluse. Elle correspond à la liste des éléments qui doivent être testés automatiquement.

Une liste des ressources de l'équipe est insérée au plan de test, ainsi que leurs rôles.

Enfin, le plan comprend un agenda des différentes tâches qui seront effectuées pendant l'automatisation. Chacun des éléments de l'agenda inclut la description de la tâche à réaliser, la date à laquelle elle doit être terminée, le critère d'entrée nécessaire au commencement de cette tâche, le critère permettant la fin de la tâche, et enfin la date réelle à laquelle la tâche s'est terminée. Cet agenda permet une meilleure organisation du processus d'automation. Ceci afin de respecter les délais convenus.

Une fois le plan créé, l'équipe peut commencer à planifier le cycle de test.

### **c. La conception de la suite de tests**

Ils commencent tout d'abord par concevoir la suite de tests.

Une suite de tests est un ensemble de tests liés les uns aux autres. Ils sont groupés par rapport à leur fonction ou la zone de l'application qu'ils impactent. De plus, ils peuvent être rassemblés par rapport à leur priorité par rapport aux spécifications utilisateur. Ainsi, les tests critiques peuvent être exécutés ensemble et les tests de moindre importance eux aussi. Ils sont exécutés ensemble mais surtout dans un ordre défini. L'exécution de la suite de tests est une fonctionnalité présente dans l'outil de test.

Chaque test présent dans la suite partage le même contexte d'entrée et de sortie. Cela permet de préparer les mêmes données pour une même suite avant de les exécuter.

Une fois les suites de tests préparées, l'équipe peut procéder à la conception du cycle de test.

## **d. La conception du cycle de test**

Le cycle d'exécution de la suite doit être organisé. Il faut préparer la configuration de l'environnement relative à chaque cycle.

Un cycle de test se déroule de la manière suivante.

Tout d'abord, il faut configurer l'environnement. Ceci inclut la préparation de toutes les variables relatives à l'exécution. La plateforme doit donc être correctement configurée. Ceci implique le matériel mais aussi la librairie de test et l'application testée.

Ensuite, il est important d'initialiser la base de données pour s'assurer que les données sont dans un état connu et vérifié.

De plus, une séquence de cycles de test doit être préparée.

Enfin, le cycle se termine par la remise en état de l'environnement de test. Les fichiers de travail sont supprimés ainsi que l'historique.

Une fois ce cycle terminé. L'équipe entreprend l'exécution des tests.

## **e. L'exécution des tests**

L'exécution des tests devient donc automatique. De par ce fait, il est nécessaire que les résultats de l'exécution soient entièrement documentés. En lisant les résultats, l'équipe est capable de déterminer les tests qui ont été exécutés avec succès ou ceux qui ne l'ont pas rencontré. De plus, il est possible d'analyser les performances de l'application pendant ces tests.

L'exécution des tests fournit donc un log de ces tests. Il comprend le résultat de chaque cas de test ainsi que le temps d'exécution pour analyser les performances.

Le résultat du cas de test consiste à marquer s'il est passé avec succès ou non. D'autre part, chaque cas de test est priorisé en fonction des spécifications du client. Il est donc possible de trouver les cas de test critiques qui n'ont pas fonctionné.

En ce qui concerne l'analyse des performances, les logs fournissent le temps de réponse de chaque cas de test si l'application est basée sur une architecture client-serveur. De plus, il inclut le temps d'exécution de chacune des fonctionnalités.

Les logs de tests contiennent aussi la configuration de l'environnement avec laquelle les cas de tests ont été exécutés. Ainsi, il est possible de connaître la configuration avec laquelle l'application fonctionne le mieux.

Enfin, le total des données fournies dans les logs est inscrit à la fin du fichier de log. Il est donc inscrit le total des cas de tests exécutés, ceux qui ont rencontré un succès, ceux qui ne sont pas passés et le temps total d'exécution.

D'autre part, l'automatisation de test écrit un fichier d'erreurs. Pour chaque cas de test n'ayant pas rencontré de succès, ce fichier contient les informations relatives au problème. Ceci permet aux testeurs d'analyser les problèmes rencontrés.

Les informations inscrites contiennent tout d'abord la description du cas de test ayant échoué.

De plus, le fichier d'erreur comporte les informations relatives à l'état de l'application au moment où l'erreur a été rencontrée comme la date et le temps, le résultat espéré et la tentative de réparation de l'erreur par l'application.

Le cycle de test se termine par l'analyse des résultats fournis par l'exécution des tests.

## **f. L'analyse des résultats**

Cette tâche est nécessaire car l'automatisation de test peut fournir des résultats difficiles à comprendre. De plus, ces résultats peuvent se montrer imprécis.

Les résultats imprécis correspondent à des informations erronées quant à l'état de l'application lorsque le test a échoué.

Ces problèmes peuvent être rencontrés dans plusieurs situations.

Tout d'abord, cela peut avoir un rapport avec l'environnement dans lequel l'application a été testée. Dans ce cas là, le cas de test échoue mais la raison réelle peut être en relation avec un problème de données si la base n'a pas été correctement initialisée, ou en relation avec un environnement qui a été mal configuré.

D'autre part, un test peut échouer si l'application a subi des changements et qu'ils n'ont pas été portés sur les cas de test.

Un problème peut être aussi rencontré lorsque l'erreur est présente dans le script de test.

De plus, il peut s'agir d'une erreur dupliquée. En effet, certains éléments peuvent être testés dans différentes situations. Le nombre d'erreurs inscrites peut donc être imprécis car il contient plusieurs fois la même erreur.

Enfin, un test peut rencontrer un faux succès. En effet, s'il existe une erreur dans le script de test et que le test ne parvient pas à atteindre ce qu'il doit tester, l'automatisation peut inscrire un faux succès. Pour trouver les faux succès, l'équipe de testeur peut analyser les temps de test. Effectivement, si un test s'exécute trop rapidement, il peut s'avérer qu'il rencontre un problème.

Après cette analyse, si un test a échoué et que cela est dû à une erreur de développement de l'application, le défaut est communiqué à l'équipe de développement. Une fois qu'il a été rapporté, l'équipe de testeur inscrit la description du cas de test qui a révélé ce bug, la date à laquelle il a été découvert et le développeur chargé de corriger le problème. Par la suite, les testeurs pourront inscrire si ce défaut a été résolu. Cela permet de suivre l'évolution des défaillances du logiciel pour être sûr qu'elles ont été corrigées.

Après avoir effectué toutes ces étapes, l'entreprise est donc parvenue avec succès à installer l'automatisation de tests au sein de son processus de développement de logiciels.

# CONCLUSION

L'automatisation de tests est un système coûteux et complexe à mettre en place dans une entreprise.

Pour procéder à l'installation de ce système au sein d'une entreprise dans les meilleures conditions, plusieurs étapes préalables d'analyse sont nécessaires.

Tout d'abord, il est important de sélectionner les cas de tests à automatiser et fixer leur priorité. En effet, l'automatisation ne peut pas s'appliquer à tous les cas de tests.

Les cas de tests avec lesquels l'automatisation améliore la productivité sont ceux qui nécessitent une exécution répétitive. En effet, l'un des avantages principaux de l'automatisation est la diminution de l'intervention humaine. Lorsqu'un testeur doit effectuer une tâche répétitive manuellement, il peut se lasser de son travail. Ceci a des conséquences sur sa motivation. Or, le manque de motivation laisse place au manque d'attention et donc à l'augmentation du nombre d'erreurs.

De plus, il convient d'automatiser les cas de tests à haut risque. C'est-à-dire ceux qui correspondent aux fonctionnalités principales de l'application. Comme il l'a été évoqué, l'intervention humaine laisse place à l'augmentation de la présence d'erreurs. Il est donc indispensable de s'assurer que les cas de tests liés aux priorités fixées par le client ne contiennent pas de défaut.

Enfin, la priorité est donnée aux cas de tests dont l'exécution contient un haut degré de complexité. Ceci pour la même raison que précédemment.

Ensuite, l'équipe chargée d'installer l'environnement doit choisir la portée de l'automatisation de test. Autrement dit, choisir les types de tests et les zones nécessitant d'être automatisés.

L'automatisation de tests est un domaine qui requiert des compétences pointues. Ceci est dû à sa complexité. De plus, l'avantage de ce système est de réduire la durée du cycle de test. Il est donc important de couvrir les zones qui ne nécessitent pas de modifications fréquentes. Afin que les testeurs ne soient pas obligés de constamment modifier les cas de tests.

En ce qui concerne les types de tests, l'automatisation ne permet pas de tous les gérer. Il faut sélectionner ceux dont l'automatisation apportera une réelle valeur ajoutée. Or, de nombreux types de test ont été créés pour améliorer la qualité du logiciel.

Ces types de tests se divisent en deux catégories.

La première correspond aux tests fonctionnels. Cette catégorie regroupe les tests qui permettent de vérifier que les fonctionnalités de l'application répondent aux besoins métier. Ces tests suivent la méthode appelée la " boîte noire ". C'est-à-dire qu'ils sont effectués sans examiner la structure interne du code source de l'application. Ces tests vérifient que les sorties correspondent à celles espérées après l'exécution avec certaines données en entrée.

Cette catégorie de test nécessite d'être automatisée. En effet, ces tests sont compliqués et nécessitent une certaine compétence. L'avantage de l'automatisation prend donc tout son sens.

Parmi ces différents tests, il faudra porter une attention supplémentaire aux tests de régression. Ce type de test sert à vérifier qu'une modification apportée au code source de l'application n'a pas eu de conséquence négative sur le fonctionnement de celle-ci. Ces tests peuvent s'avérer répétitifs car ils consistent à exécuter à nouveau les cas de tests ayant pu être atteints par la modification. Pour éviter l'erreur humaine, il convient donc de les automatiser en priorité.

D'autre part, il existe la catégorie de tests non fonctionnels. Celle-ci sert à vérifier que l'application suit les scénarios rédigés par le client tout en étant performante. Elle comprend des tests de sécurité, d'utilisation mais surtout de performance. Or, les tests de performance demandent énormément d'exécution de tests. En effet, si les testeurs doivent vérifier que le logiciel supporte des milliers de connexions simultanées, il paraît logique qu'ils n'essayent pas de lancer manuellement des milliers de tests. Dans ce cas, l'automatisation de test prend tout son sens car elle est capable d'exécuter ces milliers de connexions extrêmement rapidement.

Après avoir identifié les zones et types de tests devant être couverts par l'automatisation. L'entreprise peut commencer à réfléchir au choix de l'outil d'automatisation, parmi les nombreuses solutions existantes, qui répondra à ses besoins. Cette étape est importante car le succès de l'automatisation dépendra en grande partie de l'outil qui a été sélectionné.

Différents critères de base sont à prendre en compte dans la décision finale. Tout investissement au sein d'une entreprise dépend de son coût. Cependant, elle ne peut pas s'arrêter à de telles considérations et doit examiner les services qui sont fournis tel que le support utilisateur qui accompagne le produit. De plus, une période d'évaluation est un critère décisif dans le choix car elle permettrait à l'entreprise de vérifier que le produit correspond à ses besoins dans une situation réelle.

D'autre part, l'entreprise doit vérifier que les caractéristiques de l'outil d'automatisation s'appliquent à l'environnement du projet à automatiser et éventuellement des futurs projets. Il est possible de citer par exemple des critères tels que les systèmes d'exploitation devant être en accord avec ceux des projets, ou encore la possibilité de couvrir les zones qui ont été choisies précédemment, ou enfin la gestion des données d'entrées des tests qui doit être la plus vaste possible car il n'est pas possible de savoir qu'elles seront les données nécessaires aux futurs projets.

Enfin, certaines fonctionnalités doivent être présentes dans l'outil afin de permettre une meilleure productivité telle que la gestion du travail d'équipe avec le contrôle de version, la possibilité de comparer les résultats des tests à partir de l'outil pour faciliter leur visualisation et l'interprétation des contrôles graphiques présents sur les différents projets de l'entreprise.

Lorsque l'entreprise a pris sa décision concernant l'outil, elle doit choisir ensuite le framework d'automatisation de tests. L'avantage principal du framework est qu'il est indépendant d'une application. Le choix n'affectera donc pas les différents projets pouvant être automatisés. D'autre part, ce framework est modifiable et ainsi extensible. L'entreprise peut donc travailler dessus pour qu'il réponde à ses besoins exacts. Il correspond à l'environnement dans lequel les tests seront exécutés, au raccordement à l'application testée et à la collecte des résultats de l'exécution.

Il existe quatre types de framework d'automatisation.

Le premier, le type modulaire, s'attache à lier chaque script de test à une partie de l'application appelée " module ". Il permet une facilité de maintenance grâce aux divisions des scripts mais ceux-ci deviennent trop lourds car ils embarquent les données de test.

Ensuite, il existe le type concentré sur les données de test. En effet, contrairement au type précédent, les données sont stockées dans des fichiers externes et non pas sur les scripts

de test. L'inconvénient du type est principal est donc éliminé tout en gardant les avantages de maintenance. D'autre part, la différence entre données et scripts permet de séparer ces deux tâches dans leurs préparations. Cependant, les données sont gérées par un script de conduite. L'inconvénient est qu'à chaque changement des données, ce script doit être modifié.

De plus, l'entreprise peut choisir le type de framework mots-clés. Cela signifie que les scripts de tests dépendent de mots-clés stockés dans des tableaux sur des fichiers externes. Chaque mot-clé est lié à une action effectuée sur l'application testée. Donc, lorsqu'une action est effectuée, le framework se charge d'exécuter le script lui correspondant. Les avantages des précédents frameworks sont gardés tout en ajoutant une certaine facilité d'utilisation. L'inconvénient principal est que les scripts de test deviennent plus longs et donc plus compliqués à maintenir.

Comme il l'a été évoqué, chacun des types de framework précédents comporte des inconvénients. Le nom de ce framework est le type "hybride". Il réunit les avantages de tous les autres types de framework. Il utilise les scripts de données mais aussi ceux des mots-clés. Cependant, il devient donc plus complexe que les autres types. Malgré cela, il reste le type de framework le plus populaire en automatisation de test.

Si l'application à automatiser est de taille moyenne ou supérieure et que sa durée de vie est d'une certaine importance, il conviendra de choisir le type hybride de framework qui réunit le plus d'avantages à apporter à l'entreprise. Cependant, si le logiciel est de moindre importance, l'entreprise devra choisir parmi les trois premiers frameworks qui sont moins complexes et donc demanderont moins de travail. En effet, les bénéfices de l'automatisation ne pourront pas être ressentis si l'équipe de testeurs se focalise sur l'apprentissage du framework et la maintenance des scripts.

L'étude préalable à l'automatisation de test et étant terminée et les choix étant effectués, l'entreprise peut donc commencer le processus de test.

Tout d'abord, il est indispensable de former l'équipe chargée de s'occuper des tests automatisés. Cette équipe est dirigée par un chef d'équipe. Il est chargé de coordonner l'équipe de test avec l'équipe de développement. De plus, il a autorité sur toute l'équipe. D'autre part, l'équipe est composée en majorité de deux groupes. Le premier se charge de développer les tests tandis que l'autre doit s'occuper du framework. À côté, différentes personnes auront pour responsabilité de lier l'équipe à d'autres composantes extérieures telles que le client, le

système ou le développement. En ce qui concerne la liaison client, elle permet de vérifier que les tests sont en accord avec les critères qu'il a définis. Celle avec le système s'assure que l'environnement de test est toujours à jour. Enfin, il doit exister une relation entre les développeurs et les testeurs pour que ces derniers soient au courant des modifications apportées au logiciel.

Une fois que l'équipe de travail a été formée, elle peut procéder à l'élaboration du plan d'automatisation des tests. Chaque tâche à réaliser pour automatiser les tests est incluse dans ce plan. Il permet d'organiser les ressources disponibles ainsi que la description de chaque étape organisée dans le temps. De plus, il contient notamment une description de l'application testée et la définition de la couverture de tests que l'on souhaite automatiser.

Préalablement à l'exécution des tests, l'équipe doit tout d'abord effectuer une étape de conception.

Tout d'abord, la conception des suites de tests. Une suite de tests est un ensemble de tests. Ceux-ci sont groupés en fonction des zones de l'application qu'ils concernent. Chaque suite utilise les mêmes données de test.

Ensuite, il y a la conception du cycle de test. Pour chacune des suites de tests, l'équipe analyse l'environnement dans lequel elle doit être exécutée et les données dont elle a besoin.

Ces étapes d'analyse terminée, l'équipe peut procéder à l'exécution des tests. Celle-ci comporte différentes données relatives à son déroulement. Tout d'abord les résultats obtenus tels que le succès ou non du test. Mais aussi les performances du système pendant l'exécution. D'autre part, chaque erreur rencontrée et sa description durant cette phase sont inscrites dans un fichier.

Enfin, l'équipe termine par analyser les résultats obtenus pendant l'exécution des tests décrite précédemment. Cette étape est nécessaire car différentes conditions peuvent mener à générer de fausses erreurs. Une fois les vraies erreurs identifiées, les testeurs communiquent les problèmes ainsi que leur description à l'équipe de développeurs qui se chargent de les corriger.

L'installation de l'automatisation de tests au sein d'une entreprise est donc un processus long et complexe. Cependant, l'entreprise doit prendre l'amélioration de la qualité du logiciel produit. En effet, comme il l'a été évoqué, les tests manuels répétitifs engendrent des erreurs humaines. Mais l'automatisation permet aussi d'augmenter la profondeur de test, autrement

dit, la quantité de fonctionnalités vérifiées augmente. Ceci car elle comporte une capacité de visualisation en profondeur de l'application mais aussi car la performance et la précision d'exécution des tests sont décuplées par rapport aux compétences humaines.

D'autre part, le cycle d'automatisation des tests est coûteux. En effet, l'entreprise devra investir dans des licences de logiciels, du matériel mais aussi les ressources nécessaires à cette démarche. Mais l'automatisation engendre des économies à l'entreprise à long terme.

Tout d'abord grâce à l'amélioration de la qualité du logiciel. En effet, si la qualité du logiciel est améliorée, les ressources de l'entreprise peuvent consacrer moins de temps à corriger les défaillances logicielles. Il est donc possible d'employer ces ressources à se concentrer sur des domaines telle que l'innovation. Or, une entreprise qui innove se démarque de ses concurrents aux yeux des clients potentiels. D'autre part, la réputation de l'entreprise est améliorée car ses produits ne comportent plus de défauts. Une personne recherchant donc un logiciel correspondant à ses besoins va donc chercher à connaître le produit ayant le plus d'avis favorables par ses utilisateurs.

Mais aussi grâce à l'économie de temps qu'apporte l'automatisation des tests. Le cycle de test complet est réalisé plus rapidement. En effet, ce cycle commence par la préparation des données nécessaires aux tests. Or, ce travail est long et fastidieux manuellement mais peut être effectué automatiquement. De plus, comme il l'a été évoqué précédemment, l'exécution des tests elle aussi est gérée par l'automatisation. L'analyse des résultats et elle aussi facilitée par l'automatisation qui fournit des données précises et organisées.

Cependant, il peut paraître compliqué de justifier un tel investissement auprès de sa hiérarchie. En effet, celle-ci aura besoin de connaître les futures économies engendrées par l'installation de l'automatisation. Pour cela, il est possible de calculer le ROI (retour sur investissement) de l'automatisation de test. Un roi consiste à approximer le gain qui sera obtenu après la mise en place du système d'automatisation.

Différents facteurs doivent être pris en compte pour le calcul du ROI lié à l'automatisation de test. Tout d'abord, les prix du matériel et des licences qui peuvent s'avérer élevés. Mais aussi d'autres variables plus compliquées à estimer telle que le temps qui sera pris pour effectuer le cycle de test. D'autre part, l'entreprise doit fixer la date laquelle elle

espère un gain. Enfin, une étude doit être menée afin de déterminer le nombre d'exécutions de test que l'automatisation de test pourra couvrir sur le nombre de tests total.

Il existe différentes méthodes de calcul du ROI d'automatisation de test. Chacune de ces approches répond à un besoin spécifique. L'entreprise devra donc déterminer celui qui lui correspond.

Le premier, appelé " Simple ROI " consiste à calculer l'économie en terme monétaire qui sera apportée à l'entreprise si elle automatise son processus d'exécution des tests. L'avantage d'une telle méthode est que les dirigeants d'une entreprise peuvent s'avérer convaincus par les résultats lorsqu'ils sont financiers. Cependant, le calcul de ce ROI comporte certaines données estimées et donc peut manquer de précisions.

D'autre part, il existe le "Rendement ROI". Celui-ci cherche à diminuer l'imprécision du calcul précédent en retirant les variables estimées. Par conséquent, il ne calcule plus l'économie monétaire mais l'économie en terme de temps diminué par le processus d'automatisation. Cependant, elle aussi peut s'avérer imprécise car, pour effectuer ce calcul, on estime que tous les tests manuels seront automatisés. Or, comme il l'a été évoqué, cela n'est pas toujours possible.

Enfin, la dernière approche, la " Réduction des risques ", a été créée pour supprimer les inconvénients des deux méthodes précédentes. Elle ne prend pas en compte les tests manuels effectués précédemment et le gain est donc calculé en fonction du coût de correction d'un défaut après sa mise en production. Cependant, cette valeur de gain est donc estimée et n'est donc pas entièrement fiable.

Lorsque la décision d'automatiser les tests a été prise et justifier, et que le système a été installé, l'entreprise peut souhaiter valider les bienfaits jusqu'à lors supposé de cet investissement.

Pour calculer la valeur ajoutée de l'automatisation, il existe différentes métriques.

Il a été évoqué que l'automatisation améliore la qualité du logiciel en fournissant une meilleure couverture de l'application par les tests. Il est donc important de vérifier cela par une méthode calculant le pourcentage de couverture de test. Il sera alors possible de comparer le résultat à la couverture pendant les tests manuels.

D'autre part, il est possible de calculer le ratio du nombre de défauts contenus dans l'application par rapport à sa taille. En effet, l'automatisation doit engendrer une réduction des défaillances logicielles.

Enfin, la métrique la plus importante est celle qui correspond à l'efficacité de correction des bugs. C'est-à-dire le nombre de défauts trouvés après la livraison du produit. Idéalement, ce nombre doit être égal à 0. Si c'est le cas, cela validerait donc complètement le système mis en place puisque cela voudrait dire qu'il parvient à détecter tous les problèmes pendant le cycle de test.

L'automatisation de test est donc compliquée et coûteuse à installer au sein d'une entreprise. Cependant, si l'analyse préalable à sa mise en place et les différents choix d'outils ont été effectués de manière conscientieuse, les retombées économiques peuvent être importantes pour l'entreprise.

# GLOSSAIRE

- Automatisation de test: utilisation d'un logiciel pour contrôler l'exécution des tests
- Cas de test: ensemble de conditions et variables permettant au testeur de vérifier les fonctionnalités du logiciel
- Client: logiciel qui envoie des demandes à un serveur
- CMMI (Capability Maturity Model + Integration): modèle de référence, ensemble structuré de bonnes pratiques, destiné à appréhender, évaluer et améliorer les activités des entreprises d'ingénierie
- Compilation: programme informatique qui transforme un code source écrit dans un langage de programmation (le langage source) en un autre langage informatique (le langage cible)
- Environnement: pour une application, l'ensemble des matériels et des logiciels système, dont le système d'exploitation, sur lesquels sont exécutés les programmes de l'application
- Framework: ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel
- ISO 9000: ensemble de normes relatives à la gestion de la qualité publiées par l'Organisation internationale de normalisation (ISO)
- Log: enregistrement séquentiel dans un fichier ou une base de données de tous les événements affectant un processus particulier
- Métrique: compilation de mesures issues des propriétés techniques ou fonctionnelles d'un logiciel
- Objet: conteneur symbolique, qui possède sa propre existence et incorpore des informations et des mécanismes

- ROI: désigne un ratio financier qui mesure le montant d'argent gagné ou perdu par rapport à la somme initialement investie dans un investissement
- Script de test: ensemble d'instructions lancé sur l'application testée pour vérifier qu'elle fonctionne
- Serveur: dispositif informatique matériel ou logiciel qui offre des services, à différents clients
- Système d'exploitation: ensemble de programmes qui dirige l'utilisation des capacités d'un ordinateur par des logiciels applicatifs
- Test: désigne une procédure de vérification partielle d'un système

# TABLE DES MATIERES

<b>INTRODUCTION .....</b>	<b>1</b>
<b>I. Types de tests .....</b>	<b>8</b>
1. Les méthodes .....	8
a. La méthode de la boîte blanche .....	8
b. La méthode la boîte noire.....	10
2. Les tests fonctionnels.....	12
a. Les tests unitaires .....	12
b. Les tests d'intégration.....	13
c. Les tests de régression.....	14
d. Les tests d'acceptation.....	15
e. Les Bêta tests.....	16
f. Les tests de détection de fumée.....	17
g. Les tests de santé .....	18
3. Les tests non fonctionnels .....	19
a. Le test de charge.....	19
b. Le test de résistance .....	20
c. Le test d'utilisation .....	21
d. Le test de sécurité.....	22
e. Le test de compatibilité .....	23
f. Le test d'installation.....	24
4. L'organisation.....	24
a. Le cycle en V .....	24
b. L'agilité .....	26
<b>II. L'AUTOMATISATION DE TESTS .....</b>	<b>29</b>
1. L'amélioration de la qualité.....	29
a. Moins d'erreurs humaines .....	29
b. Augmentation de la profondeur de test .....	30
2. La réduction des coûts.....	31
a. Le lien entre qualité et bénéfices.....	31
b. Une économie de temps.....	33
3. Le calcul du gain.....	34

a.	Le retour sur investissement .....	34
b.	Les métriques .....	39
<b>III. MISE EN OEUVRE.....</b>		<b>42</b>
<b>1. Choisir quoi automatiser.....</b>		<b>42</b>
a.	Choix des cas de tests .....	42
b.	Choix de la portée .....	44
<b>2. Choisir les outils .....</b>		<b>45</b>
a.	Choix de l'outil d'automatisation .....	45
b.	Choix du framework de test.....	47
<b>3. Le processus.....</b>		<b>50</b>
a.	L'équipe .....	50
b.	Le plan d'automatisation de tests.....	51
c.	La conception de la suite de tests.....	52
d.	La conception du cycle de test.....	53
e.	L'exécution des tests.....	53
f.	L'analyse des résultats.....	54
<b>CONCLUSION.....</b>		<b>56</b>
<b>GLOSSAIRE.....</b>		<b>64</b>
<b>Table des matières .....</b>		<b>66</b>
<b>BIBLIOGRAPHIE.....</b>		<b>68</b>
<b>ANNEXE.....</b>		<b>69</b>
<b>SYNTHESE ANGLAIS .....</b>		<b>I</b>

# BIBLIOGRAPHIE

- KENT BECK, Test Driven Development: By Example, Pearson Education, 2003  
Description de la méthode Test Driven Development
- ROY OSHEROVE, The Art of Unit Testing: with Examples in .NET, Manning Publications, 2009  
Conseils sur la manière d'effectuer les tests unitaires
- ADAM GOUCHER et TIM RILEY, Beautiful Testing: Leading Professionals Reveal How They Improve Software, O'Reilly Media, 2009  
Importance des tests dans le cycle de développement et techniques
- RON PATTON, Software Testing, Sams Publishing, 2005  
Importance des tests sur la qualité logicielle et pratiques à suivre
- CEM KANER et JACK FALK et HUNG Q. NGUYEN, Testing Computer Software, Wiley, 1999  
Description des tests à effectuer sur un logiciel
- ELFRIED DUSTIN et THOM GARRETT et BERNIE GAUF, Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality, Addison-Wesley Professional, 2009  
Importance et mise en place de l'automatisation de tests
- JEZ HUMBLE et DAVID FARLEY, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Professional, 2010  
Techniques permettant d'améliorer le cycle de développement de logiciels

- PHILIP B. CROSBY, Quality Is Free: The Art of Making Quality Certain: How to Manage Quality - So That It Becomes A Source of Profit for Your Business, McGraw-Hill Companies, 1979  
Prévenir les problèmes de qualité dans le cycle de production
- DOROTHY GRAHAM et MARK FEWSTER, Experiences Of Test Automation, Addison-Wesley Professional, 2012  
Témoignages de mise en pratique de l'automatisation de tests

# **ANNEXE**

## **SYNTHESE ANGLAIS**

The global cost of software defects is estimated at 312 billions of dollars every year.

Not only this cost is important in the current economic climate, but also, a study conducted by the company Undo Software showed that without solution provided, it will continue to skyrocket in the coming years.

Software failures are not only economic significance. Indeed, nowadays, the computer is also used in hazardous areas such as aerospace or military. A software defect may therefore cause casualties.

It is for these reasons that the IT industry should attach importance to the quality of software.

However, the quality of software is not an easy concept to define. Various theories have been and still stand today. But these theories share certain characteristics related to quality: customer satisfaction, compliance with specifications and absence of defects.

Despite these differences of ideas, test step has taken an important place in the software development cycle in order to improve the quality of the developed software. Indeed, the test purpose is to find defects in an application previously launched.

It has four main objectives. The first is to prove that the software can be sold. Also, it finds bugs and traces them. It provides information to reduce the number of defects. Last but not least, it improves the software quality.

To achieve optimal quality, many types of software testing emerged. Each type is designed to test a particular area of the application and is intended to be run at different times of the development cycle.

It is possible to divide the types of software testing into two categories: functional testing and non-functional testing. The first category allows to verify that the software functionalities correspond to the business specificities. These tests do not check the internal behavior of the application. On the other hand, non-functional testing shall endeavor to verify the user experience and application performance.

The software development and thus integrated tests are usually organized according to two different and popular methods.

The first method is V-model. It is the most widely used since the 1980s. Each stage of the software development is associated with a step of testing. The cycle begins with the analysis of user needs. This step is associated with acceptance testing that come at the end of the V-model. Thereafter, the team proceed to write specifications that corresponds to the testing system. After that occurs the creation of the application architecture that is related to the integration testing. Finally, the development process ends with the application programming that is linked with unit testing.

The other approach to project management includes agile methods. This approach is increasingly popularized since 2001 and the writing of the fundamentals. Indeed, its creation is due to the limits met by the V-model. The main drawback is that it can not handle changes in customer requirements during the software development cycle. This is why the agile approach divides the software development cycle into several iterations. Each iteration corresponds to a complete development cycle of the V-model. The tests are conducted at the end of each iteration.

Both approaches attach great importance to the application test cycle. Cycle as it was mentioned above contains many types of tests. To improve the testing process, it is possible to implement the automation of these tests.

Manual testing is performed by a human sitting in front of a computer executing the test steps. Automation Testing means using an automation tool to execute the tests. The

automation software can also enter test data into the application, compare the expected and actual results and generate detailed test reports.

The test automation has certain advantages.

The most important is to improve software quality. The main feature that makes this possible is the reduction of human intervention. Indeed, the test step may be redundant and tedious for developers and testers. It therefore leaves room for human error. Automation avoid these mistakes. On the other hand, the test automation increases test coverage. Indeed, it is impossible for a human to test all possible conditions. Software is not tested as the others products because there are a lot of possible cases. The test automation can cover all these cases and thus ensure that the software works with all possibilities of entries.

In addition, it is known that "time is money". However, test automation reduces the time consumed in testing the software. Indeed, the time taken by each step of the testing process is reduced.

On the other hand, if the automation saves time, it allows the company to focus its efforts on other areas. For example, innovation. However, innovation is a key factor that allows the company to distinguish itself from its competitors. Which will benefit it later and will increase sales.

In addition, the test automation reduces cost of software development. Indeed, there is a link between product quality and cost reduction. It has been proven that the more a bug is fixed late, the more it costs. But automation identifies failures earlier in the development cycle of the software.

However, the implementation of test automation has a significant cost. Prior to the expenditure, it is important to calculate the return on investment (ROI). The ROI can calculate the long-term gain obtained if the tests are automated.

There are different ROI calculations.

The first is the "Simple ROI." It focuses on the economy in terms of money made by the execution time of the automated tests.

On the other hand, there is the "Efficiency ROI." It is based on the previous method, but by calculating the time saved.

The last is called the "Risk Reduction". It focuses on the benefits of test automation regardless of manual testing.

Each has advantages and limitations. We must choose the one that best suits the needs of the company.

On the other hand, there are automation testing metrics. It corresponds to calculations to measure the testers productivity before and after the introduction of automation. But also other measures such as the percentage of the application that is covered by the tests and the number of defects contained in the application.

These automation evaluation methods allow the company to make and justify the decision to begin the process of automation testing.

After the decision, the company can proceed with the implementation of the system.

The first step of this study is to choose what should be automated. To do this, we must study the test cases of the project. Priority should be given to test cases that require repeated execution. Test cases of high risk, i.e. that are related to the most important features of the application by the client, must be automated first too.

After, the company must choose the most suitable tool. This choice is primarily guided by the price of the tool, the user support offered during the use, and the features available.

Finally, prior to the installation of test automation, the company must choose the ideal test framework. There are different types of framework. Each of these fits a certain size of project. It is therefore important to choose analyzing the current but also future projects.

Then, the company can begin the automation process.

The first step is to create the team of testers. It must be composed of individuals with expertise in test automation. Each of them will have a different responsibility.

Next, the team should proceed with the conception of the test automation plan. This plan contains the tasks to be performed to complete the implementation of the automation system.

Finally, the team must develop automated testing.

The implementation of test automation in a company is not to be taken lightly. It is important to consider and study the needs of the business to ensure that automation is necessary and can bring benefits such as saving money and higher quality production.