AUTOMATISATION DES ACTIVITÉS DE TEST

L'AUTOMATISATION AU SERVICE DES TESTEURS

Ouvrage collectif coordonné par :
Olivier Denoo
Marc Hage Chahine
Bruno Legeard
Eric Riou du Cosquer

Edité par le Comité Français des Tests Logiciels

SOMMAIRE

PRÉFACE

par Olivier Denoo, président du CFTL

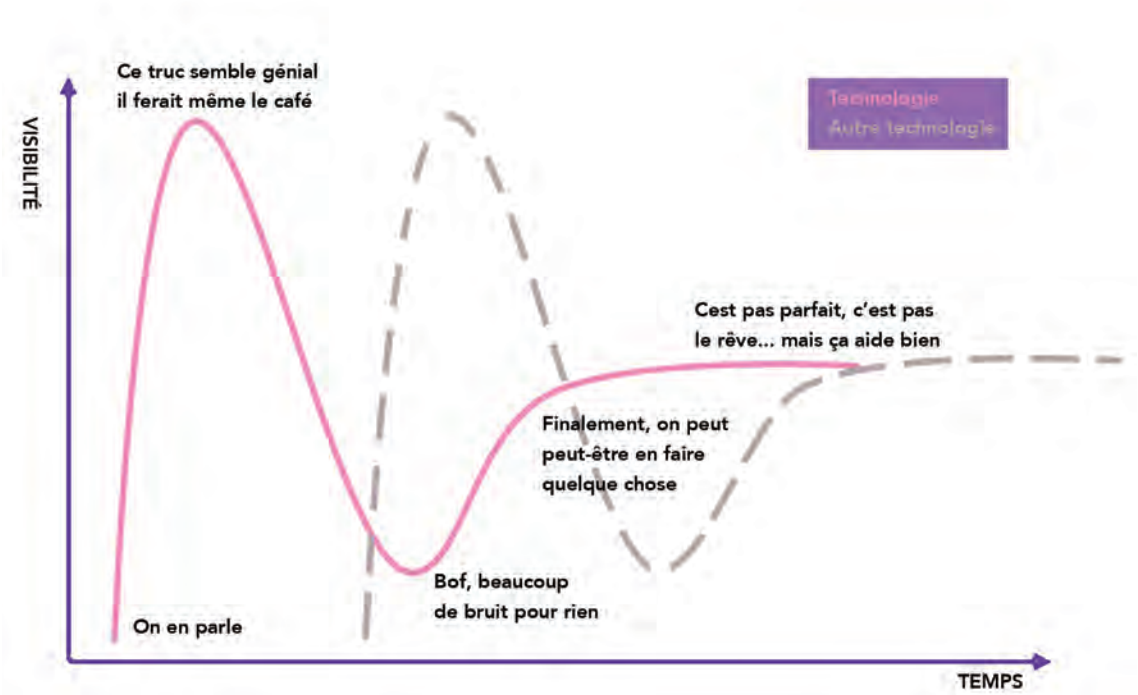
Automatisation : nos cadavres exquis boiront-ils le vin nouveau[1]?

Quand il y a vingt ans à peine, on nous vendait encore le rêve d'une automatisation presse-bouton en « capture-edit-playback », qui tournerait toute seule la nuit et nous donnerait les nouvelles du matin, force fut de constater, une fois les paillettes retombées, que la machine avait souvent une constitution délicate et que son usage imposait rigueur, stabilité et stratégie.
Car, non ma bonne dame, tout ne s'automatise pas… et oui, mon bon monsieur, il faut mettre les couches dans le bon ordre et au bon endroit si l'on veut éviter fuites et déconvenues.
Souvent, après un réveil difficile, le testeur découvrait alors que l'automatisation des tests restait un projet de développement comme tant d'autres, mais à sa charge. Combien de fois n'ai-je connu ces matins qui déchantent où la suite de test, déclenchée la veille en quittant le bureau, s'était interrompue après deux minutes, au troisième scénario – sur plus de cent – stoppée net par une fenêtre inattendue qui avait décidé, sans crier gare, de traverser l'écran à ce moment ? Ni prières, ni invocations, ni doigts croisés n'y faisaient rien…il fallait maintenir, corriger, prévoir, anticiper et surtout espérer que le lendemain, l'automate puisse enfin compléter sa tâche.
Sans parler des sessions d'analyse des rapports automatisés, qui tenaient plus de la cryptographie que de la gestion des tests.

Sans un personnel qualifié, une stratégie de maintenance appropriée, les logiciels sur étagère y finirent trop souvent, amassant la poussière, voire étant simplement oubliés. Une bien coûteuse manière de faire le ménage que bien des test managers ont apprise à leurs dépens, faute d'avoir pu ou su gérer les attentes et démythifier le discours incompris de vendeurs de rêves, plus enclins à vendre des licences qu'à proposer des solutions pérennes.

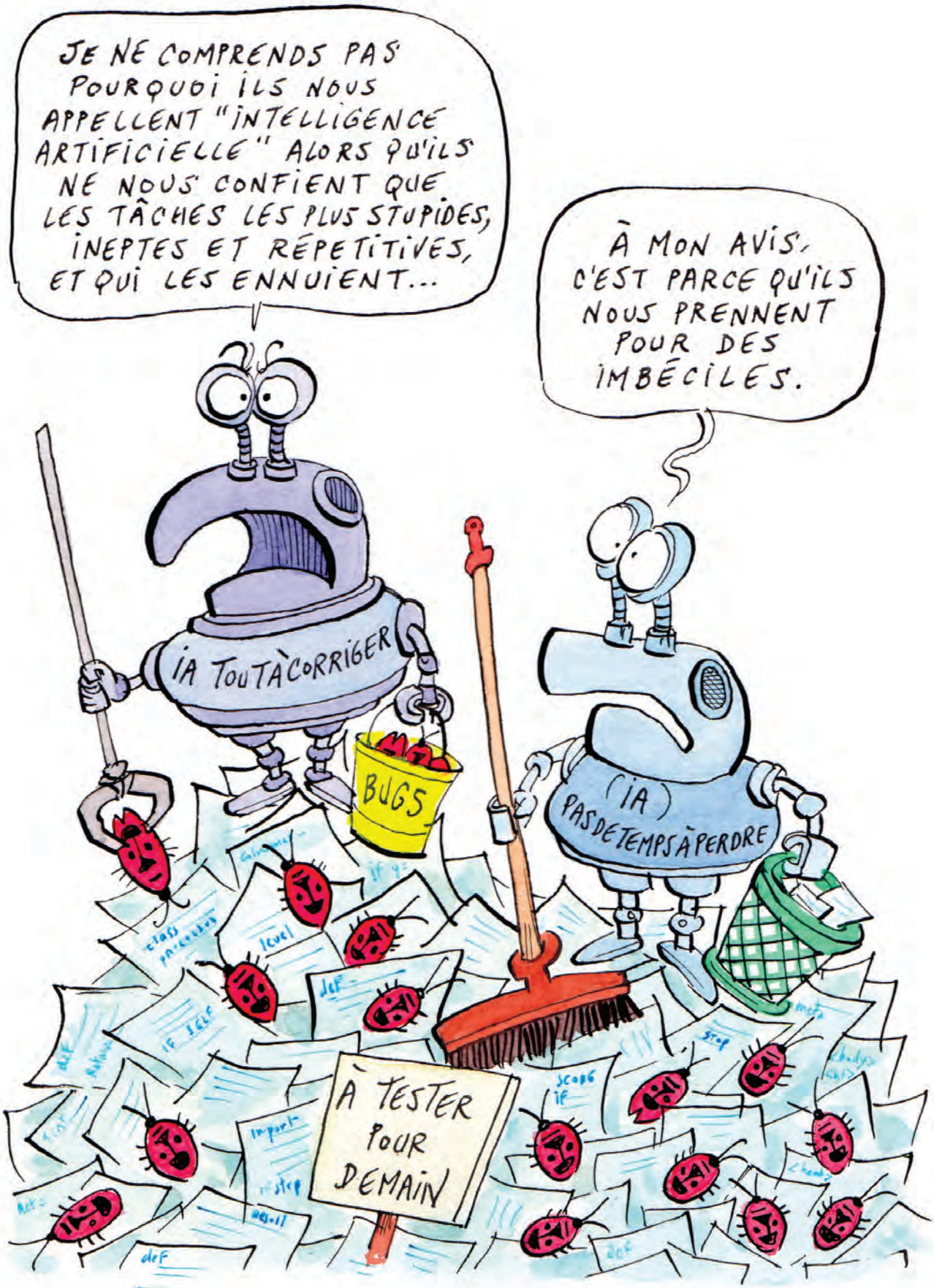
Oscillant entre amour et haine, le petit monde de la qualité grandit en maturité et sut peu à peu prendre les nécessaires distances. Les rêves de l'automatisation devinrent plus réalistes, plus propices à une industrialisation des tests raisonnée et raisonnable. Les bonnes pratiques de programmation des scripts automatisés, la (re)découverte des commentaires et bibliothèques de fonctions, la scission entre navigation et contenu, le pilotage par les données, les mots-clés et autres pseudo-langages plus ou moins naturels ont pu faire bouger les lignes de manière considérable. Enfin, la généralisation des solutions informatiques « grand public » et l'importance grandissante d'Internet favorisèrent l'automatisation au travers du test et du suivi de caractéristiques non-fonctionnelles, comme la performance, pour ne citer que l'exemple le plus flagrant. Somme toute, rien de bien neuf sous le soleil de la courbe du « hype ».

[1] D'après le jeu surréaliste éponyme inventé dans les années vingt (1920 faut-il le préciser ?) par les surréalistes Duhamel, Prévert, Tanguy



Pourtant, entre louanges et critiques constamment alimentées, l'automatisation devait encore faire face à un défi de taille, abolir cette problématique originelle qui est que ceux qui font les scripts – les testeurs, souvent rattachés à la DSI et au Développement - sont rarement ceux qui en ont vraiment besoin.
Ainsi, à la faveur d'un glissement sémantique et fonctionnel, les outils destinés à rassurer le Métier avaient-ils glissé vers la technique et le développement, quand ils ne tombaient pas, tout simplement, dans les limbes de l'échec. Malgré quelques tentatives plus ou moins audacieuses pour se rapprocher de l'utilisateur final, la montée en compétence, trop élevée et trop éloignée des missions du Métier, amenèrent souvent ce dernier à se désintéresser de l'automatisation, pour se recentrer sur les tests manuels, creusant ainsi un peu plus le fossé de la communication technico-fonctionnelle.

Certains outils furent même détournés de leur rôle premier - qui peuplant des bases de données, qui automatisant des saisies - devenant ainsi les embryons primitifs d'une RPA à venir.
Par-delà la simple création de jeux de données de test, je me souviens d'un projet où les automates suppléaient à un SI dont les performances déficientes imposaient auparavant d'inacceptables temps d'attente aux opérateurs des centres d'appel.
Par la magie de la solution automatisée, les saisies des opérateurs passaient de fichiers à plat, générés par une simple interface web, aux paramètres du cœur du SI. Si les clients n'aiment pas attendre, les automates, eux, n'en ont cure. La solution, qui ne devait être qu'un emplâtre temporaire sur une jambe de bois, emporta tous les suffrages au point qu'elle fut déclinée dans toutes les filiales du groupe !


JE NE COMPRENDS PAS POURQUOI ILS NOUS APPELLENT "INTELLIGENCE ARTIFICIELLE" ALORS QU'ILS NE NOUS CONFIENT QUE LES TÂCHES LES PLUS STUPIDES, INEPTES ET RÉPETITIVES, ET QUI LES ENNUIENT...
À MON AVIS, C'EST PARCE QU'ILS NOUS PRENNENT POUR DES IMBÉCILES.
IA TOUTÀCORRIGER
BUGS
(IA) PASDETEMPSÀPERDRE
À TESTER POUR DEMAIN

Loin de se cantonner uniquement au rôle d'exécutant, l'automatisation prit peu à peu une part croissante au sein des activités de test. De l'automatisation des tâches répétitives à l'analyse des logs, du peuplement des bases de données au reporting, plus rien ne lui échappe ; pas même les exigences, l'analyse du code ou encore les processus industriels chers au monde embarqué ou à la robotique… Avec encore et toujours cette promesse du plus : plus vite, plus en profondeur, plus de couverture, plus d'efficacité. L'automatisation, c'est travailler moins pour gagner plus… pour autant, du moins, qu'elle soit bien conçue et réfléchie. Car, faut-il le répéter, l'effort initial de l'apprentissage, de la prise en main et de la conception est souvent élevé ; et si les gains de performance sont, dans de bonnes conditions, assez rapides, il n'en reste pas moins que le démarrage est laborieux.

De l'automatisable à l'automatisé, il n'y eut souvent qu'un pas, que les afficionados n'hésitèrent pas à franchir, parfois en dépit du bon sens et de la maintenabilité ; et quand l'état d'esprit favorisant la performance en éradiquant la répétition et l'ennui devint un but, une fin en soi, le dérapage fut souvent au rendez-vous.

Le développement d'une société toujours plus technologique, toujours plus impatiente et qui a (un peu) appris de ses erreurs, ramena sur le devant de la scène des concepts plein de bon sens, du moins quand ils sont appliqués avec intelligence.
En s'inspirant des chaînes de production classiques, voici que le logiciel se développe et se teste en continu, que le tout vaut plus que la somme des parties, que chaque partie contribue au minimum à la valeur, que la collaboration règne en reine et que la qualité devient l'affaire de tous (au risque de devenir l'affaire de personne, mais c'est une autre histoire).
Bien que des murs et des barrières tombent, le centre de gravité continue de se déplacer, invariablement, immuablement, vers la technique.

L'automatisation se fait atomique et en boîte blanche, ou bien elle flirte avec l'exploitation, quand le test dicte le code au son des DDs…pour progressivement céder la place à une nouvelle venue : l'IA , si friande d'analyse, d'apprentissage ou de traitement visuel.

« La fenêtre creusée dans notre chair s'ouvre sur notre cœur[2]*»*, écrivaient en 1919 Breton et Soupault dans leurs « Champs magnétiques ».
Loin d'être aussi surréaliste que cette illustre référence, l'écriture automatique – du code cette fois – constitue, un peu plus de cent ans plus tard, un enjeu de taille pour notre métier.
D'un manifeste à l'autre, vitesse et agilité semblent être encore et toujours les clés du succès, pour transcender ou dépasser le réel. Les Surréalistes en s'affranchissant de l'étroitesse d'une pensée guidée par la raison ; les Agilistes en dépassant les notions de rôles et de savoir pour mieux recentrer le débat sur la valeur. L'enjeu devient alors la quadrature du cercle, le dialogue entre Technique et Métier et la réconciliation ultime d'intérêts potentiellement divergents.
Une fois encore, l'automatisation vient à la rescousse et se pose en facilitatrice.
Les langages de programmation sont trop difficiles à appréhender, les gestes métier trop complexes à modéliser ? Qu'à cela ne tienne… voici que fleurissent - à la manière de ces sodas trop caloriques forcés à se réinventer - de puissants outils édulcorés aux goûts étranges et aux

[2] Et son fameux cri de ralliement : « IA-qu'à ou IA plus qu'à »

couleurs sexy. Sans code, sans agents, non-invasifs, en langage naturel…voire - en glisser-déposer - sans langage du tout (en apparence du moins).

Ils s'invitent à toutes les conférences, réinventant notre manière de tester, taillant dans nos exigences, modélisant nos gestes et nos processus, apprenant de nos errements et de nos erreurs, optimisant tout, plus et plus vite, au point que d'aucuns ne s'y retrouvent plus et craignent pour leur avenir. A tort, car pas plus que les machines de la révolution industrielle n'ont remplacé l'ouvrier - ou avant eux les tisseuses - l'IA ne remplacera l'Humain.
Sans doute lui cèderons-nous, comme nous l'avons fait par le passé, nos gestes répétitifs ; ceux-là même qu'on nous dit ennuyeux voire dévalorisants (*« tu ne vas pas passer toutes tes journées à écrire des scénarios de test ou à compter des boîtes »*).
Sans doute - prisonniers nous-mêmes d'une spirale inflationniste de la valeur et de la performance - serons-nous les condamnés consentants de cette évolution et de ses inévitables laissés pour compte (car oui, soyons honnêtes, il y en aura. Il y en a toujours eu).

Si je ne crains pas l'IA ni ne doute des bienfaits réels qu'elle peut apporter à notre société et plus précisément à notre profession, je crains, en revanche, la bêtise humaine qui laisserait ce nouveau « deus ex-machina » sans limites, sans contrôle, sans surveillance.
Car le hic, c'est que nos machines ne s'embarrassent pas d'éthique ou de sentiments. Elles optimisent seulement des fonctions pour lesquelles elles ont été programmées, au terme d'un apprentissage fondé sur les données qu'on a bien voulu leur donner en pâture…et tant pis si elles confondent ou si les conséquences de leurs choix sont discutables ou dramatiques. Ce n'est ni leur problème, ni leur responsabilité…C'est la nôtre !
C'est pourquoi il est essentiel, quand la machine se complexifie au point de ne plus être appréhendable par ses créateurs, de conserver en son cœur les fondements d'une éthique intrinsèquement humaine. Parce que la valeur, la vraie, va bien plus loin que la performance, commerciale ou non.

En vous brossant ces quelques décennies d'automatisation technique et fonctionnelle, un peu en forme de capsule temporelle et avec ces grands traits décalés que vous me connaissez sans doute, je n'ai fait qu'effleurer – et parfois même égratigner avec bienveillance – les nombreux sujets qui émaillent ce nouvel opus publié par le CFTL.

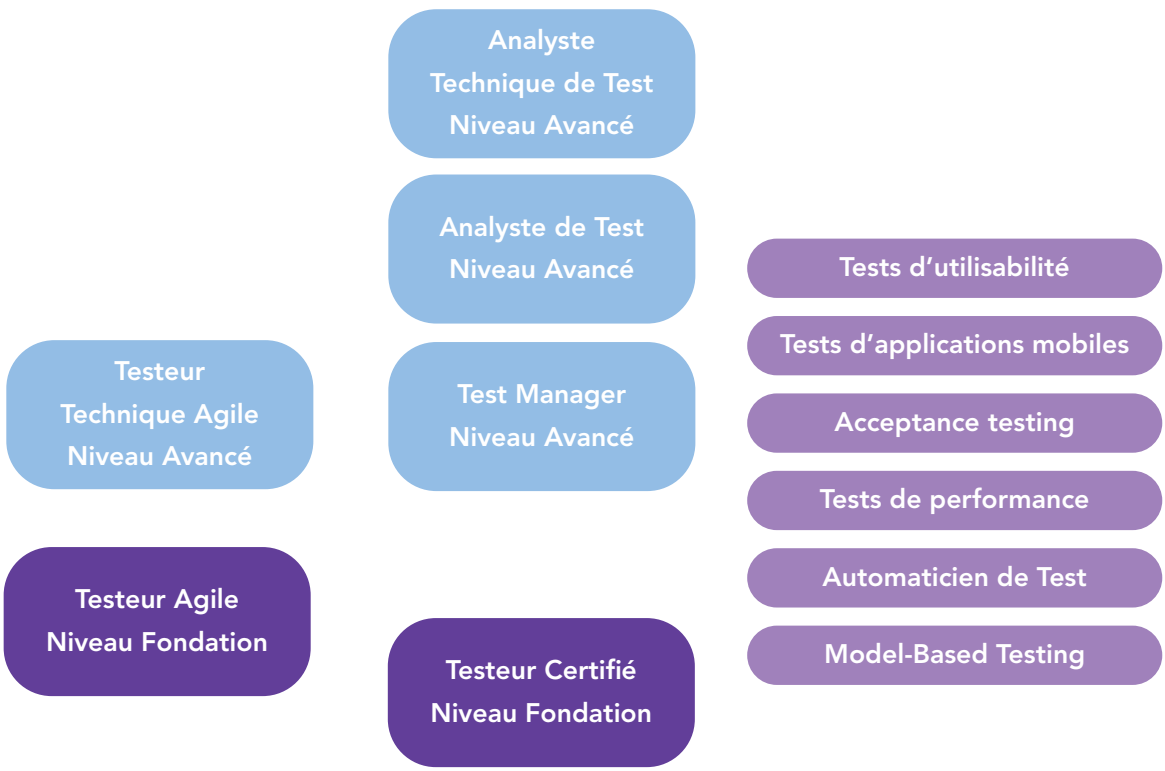
L'automatisation des tests, bien qu'elle ait toujours été invitée à la table des testeurs, connaît aujourd'hui à la fois un regain d'intérêt et une actualité brûlante.
Elle méritait bien que nos spécialistes et experts du domaine, que je remercie tous au passage pour leur excellente contribution, se penchent sur son présent et son devenir.
C'est en grande partie chose faite dans cet ouvrage pluriel que vous tenez en main, en ce moment.
Un ouvrage écrit avec passion par des passionnés.
Un ouvrage garanti sans écriture automatique.
J'espère que vous prendrez autant d'intérêt et de plaisir à le lire que nous en avons eu à l'écrire.
Je vous laisse maintenant en compagnie des robots, automates et autres IAs.

Bonne lecture.

Présentation du CFTL et de l'ISTQB

par Olivier Denoo, Bruno Legeard et Eric Riou du Cosquer

Certifications CFTL/ISTQB disponibles en France en 2021



Depuis 2008, le CFTL organise la Journée Française des Tests Logiciels qui accueille, cette année encore, plus de 1000 participants et 40 exposants.

Fondée en 2002, l'association internationale type loi 1901 (association à but non lucratif) ISTQB – International Software Testing Qualification Board – a permis de formaliser le corpus de compétence des métiers et rôles des tests logiciels.

La reconnaissance des compétences individuelles et la création de parcours de compétences sont des éléments clés pour la mise en place de parcours professionnels attractifs. La gestion des carrières dans le domaine des tests logiciels est facilitée par le schéma ISTQB qui, à partir du socle Testeur Certifié Niveau Fondation, intègre les modules Agile (Fondation et Testeur Technique Agile de Niveau avancé), les modules spécialisés (Acceptance Testing, Model-Based Testing, Utilisabilité, Performance…) et les niveaux avancés (Test Manager, Analyste de Test, Analyste Technique de Test).

Comme le montrent les différentes enquêtes de l'ISTQB, la motivation principale des employeurs et des professionnels dans le passage des certifications se trouve dans le développement et la gestion des compétences et dans une vision de long terme de leur renforcement et de leur certification.

En quelques chiffres, l'ISTQB, c'est :

- Plus de 70 pays membres sur tous les continents
- En 2020, le monde comptait près de 750 000 professionnels certifiés par l'ISTQB
- Une croissance soutenue pour la certification Testeur Agile et les modules spécialistes
- Un partenariat fort avec le schéma IREB – International Requirements Engineering Board – sur l'ingénierie des exigences.

L'ISTQB poursuit ses efforts constants afin de développer et de mettre à jour ses différentes certifications, ainsi que son Glossaire des termes des tests logiciels, en phase avec l'évolution des pratiques, dont celles issues de l'Agilité, du monde mobile ou de l'Intelligence Artificielle.
Le CFTL – Comité Français des Tests Logiciels – est représentant officiel de l'ISTQB en France, organisant ainsi à la fois la promotion des certifications ISTQB, l'accréditation des organismes de formation et la traduction française de l'ensemble des matériels. Le CFTL participe aussi directement aux groupes de travail internationaux de l'ISTQB.

Le saviez-vous ?

Avec plus de 1 000 000 d'examens cumulés passés dans le monde en 2020, les certifications ISTQB comptent parmi les plus diffusées des technologies de l'information.

Pourquoi ce livre ?

par Bruno Legeard et Marc Hage Chahine

Depuis deux décennies, l'automatisation des activités de test s'est principalement focalisée sur l'automatisation de l'exécution des tests, dont le retour sur investissement dépend à la fois de la testabilité du système sous test et de la maturité des approches utilisées.

La démocratisation des méthodes Agiles, qui ont entrainé l'accélération de la fréquence des mises en production, a multiplié les cycles de test, augmentant les activités liées à la création et à la maintenance des tests. Cette multiplication de tâches de test, devenues répétitives, ainsi que des contraintes de temps toujours plus serrées, ont passé l'automatisation d'un statut d'une possibilité à étudier à un statut d'absolue nécessité !

Mais l'automatisation reste complexe à mettre en place et souvent l'affaire de spécialistes. De même, elle est trop souvent limitée à l'activité d'automatisation de l'exécution des tests alors que, comme vous pourrez le voir dans ce livre, il y a un grand nombre d'autres activités du test qui peuvent être (et sont déjà) automatisées.
Ce livre répond au besoin de partage des connaissances de la communauté francophone des tests logiciels. Quelles activités de test automatiser ? Comment automatiser les activités de test que l'on souhaite ? Quels retours d'expérience ? Quel avenir avec cette automatisation ? Autant de questions auxquelles ce livre cherche à répondre, en fournissant de l'information actualisée par des praticiens et en montrant la diversité des problématiques et des solutions mises en œuvre.

Ce livre est ainsi organisé en trois parties.

En partie I, une présentation succincte de l'automatisation dans le test. Une introduction à son histoire, ses raisons, son rôle en DevOps ainsi que ses limites sont proposés.

En partie II, des exemples de pratiques d'automatisation de diverses activités à travers des retours d'expérience sont partagés.

En partie III, nous vous présenterons des perspectives et des pistes de réflexion afin d'aller encore plus loin dans les démarches d'automatisation des activités de test.

La nécessité d'automatiser différentes activités de test étant une conséquence directe de la généralisation l'Agile, il semblait donc évident de proposer ce livre à la suite du premier livre édité par le CFTL « Les tests logiciels en Agile »[1].

[1] « Les tests logiciels en Agile », édité par le CFTL, est maintenant en libre accès sur le site du CFTL.

Voici quelques principes qui ont guidé la rédaction de ce livre :

C'est un ouvrage collectif : chaque auteur s'y exprime en son nom propre et pas au titre du CFTL ou de son entreprise et assume la responsabilité de sa contribution dans son intégralité.

- Chaque chapitre apporte des éléments de façon indépendante des autres chapitres.
- Ce livre n'est PAS une formation pour la certification CFTL/ISTQB automatisation des plutôt un prérequis pour une bonne compréhension de tous les aspects techniques et méthodologiques présentés.
- C'est le deuxième livre du CFTL, qui s'inscrit dans sa vocation essentielle d'être au ser vice de la communauté des tests logiciels, en créant des forums d'échange sur l'évolution de notre profession.

PARTIE I
L'AUTOMATISATION DANS LE TEST


SI JE TE CORRIGES AUTOMATIQUEMENT AVANT MÊME QUE TU FASSES UNE ERREUR, EN MODE AGILE, TU VAS VOIR, ON VA GAGNER DU TEMPS.
MAIS S'IL FAUT QUE JE CORRIGE TOUTES LES ERREURS QUE TU FAIS EN ME CORRIGEANT TROP VITE...
... IL ME FAUDRA UN SECOND AUTOMATE !
iA kMoiKiSAi
F. GINTE

I.1- Evolution et état des lieux de l'automatisation

par Bruno Legeard et Marc Hage Chahine

1- L'automatisation partie intégrante du test

L'automatisation de l'exécution des tests constitue une large part de l'activité des testeurs, comme on peut le constater avec l'enquête du CFTL de 2019 établissant que moins de 10% des organisations n'ont pas de tests automatisés, mais que l'on retrouve aussi dans la figure ci-dessous, issue de l'enquête « State of Software Testing 2021 ».

Dans ce graphique, l'activité d'automatisation concerne 75 % des participants à l'enquête et arrive en premier.

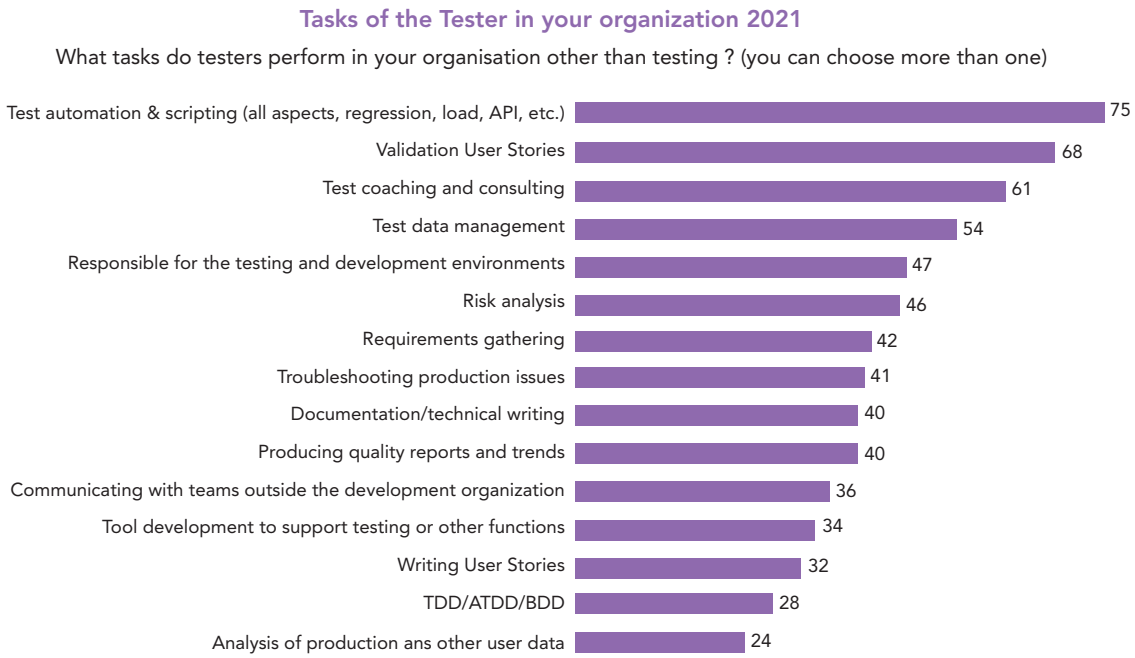
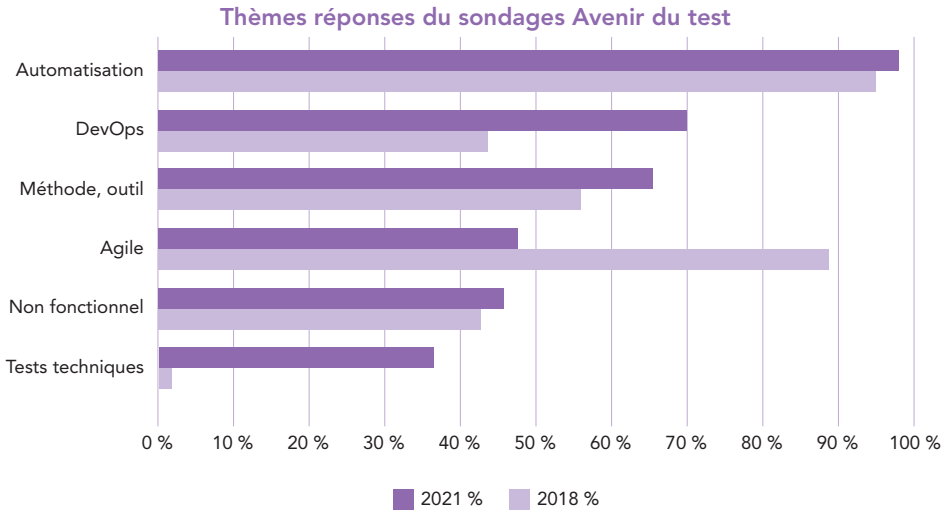


Figure 1 - L'automatisation dans l'activité des testeurs

L'automatisation de l'exécution des tests s'est installée dans ce paysage des tests logiciels de manière pérenne, comme le montrent les sondages de 2018 et 2021 de la Taverne du testeur, où l'automatisation des tests (exécution et autres activités) occupe largement la première place des thèmes d'avenir du test :



Pour autant, ce succès n'a pas été immédiat. Il est le fruit d'une lente évolution toujours en cours et des évolutions de l'industrie du logiciel en général, mais il est aussi en lien avec l'évolution du cycle de développement des logiciels (Agilité, DevOps) et l'évolution de l'outillage d'automatisation.

2- Les débuts de l'automatisation de l'exécution des tests

Lorsque l'on parle d'automatisation des tests, on pense avant tout à l'automatisation de l'exécution des tests. Cela est logique, tant l'exécution des tests scriptés est une tâche à faible valeur ajoutée et représentative d'un travail fastidieux pour les testeurs. Je pourrais même ajouter que l'idée de l'automatisation de l'exécution des tests est aussi vieille que l'exécution des tests elle-même… Elle a donc fait l'objet de nombreuses expérimentations.

Les premières tentatives d'automatisation de l'exécution des tests étaient une automatisation du type « record and replay »… Il est d'ailleurs assez inquiétant de noter que pour certaines personnes, l'automatisation de l'exécution des tests reste sous cette forme. Le principe était simple : on enregistrait les actions (mouvement de souris, clic, entrée du texte sur le clavier…) et on les rejouait. Ce type d'automatisation s'est vite révélé très mauvais pour différentes raisons. Les principales étant une maintenabilité quasiment nulle, une dépendance très forte aux « outils » utilisés (taille de l'écran, sensibilité de la souris) et à des changements graphiques même légers. Au final, cette première génération d'automatisation de l'exécution des tests s'est avérée plus fastidieuse et plus coûteuse que l'exécution manuelle des tests. Ce fut donc un échec sur toute la ligne mais un échec qui a permis au test d'avancer et de s'améliorer !

Des premiers essais, les testeurs et l'ensemble des ingénieurs logiciels ont conclu que l'automatisation de l'exécution des tests ne s'improvisait pas, que c'était un projet de développement à lui tout seul. Les tests sont alors devenus des morceaux de code utilisant des fonctions et enchaînant des actions. Les fonctions appelées utilisent des paramètres pour identifier les

éléments avec lesquels on souhaite interagir ou que l'on souhaite vérifier. Ce changement a permis d'améliorer la stabilité, la fiabilité et la maintenabilité des tests. Néanmoins, cela a rendu cette automatisation technique (peu abordable à des profils fonctionnels) mais aussi coûteuse, ce qui a fait émerger des indicateurs comme le retour sur investissement (ROI).

La seconde étape de l'automatisation de l'exécution des tests ayant été concluante elle a été améliorée en adoptant des bonnes pratiques du code comme la factorisation de parties des tests, ce qui a permis d'améliorer à moyen terme le retour sur investissement de l'automatisation en diminuant les coûts de maintenance. L'automatisation restait néanmoins un outil assez peu développé avant l'émergence de nouvelles méthodes de développement avec la démocratisation des méthodes Agile !

3- La généralisation de l'automatisation de l'exécution des tests

Les développements en Agile sont devenus la norme, comme on peut le voir avec l'enquête 2019 du CFTL :

CHOIX DES RÉPONSES	RÉPONSES	
Le projet suit une approche agile (de type Scrum, Kanban, SAFe ou autre)	63,93 %	450
Le projet suit une approche par phase (type cycle en V)	36,06 %	254
TOTAL		704

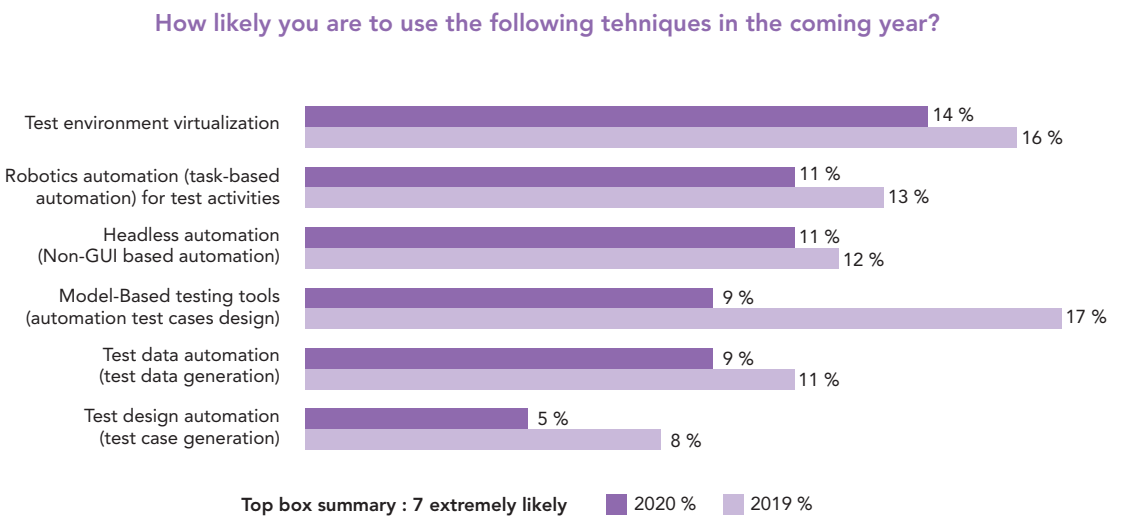
Cette proportion de développement en Agile continue d'ailleurs à augmenter, avec une nécessité pour les marchés du numérique et du logiciel à être toujours plus réactifs, en faisant face à des utilisateurs toujours plus exigeants, dans un marché toujours plus concurrentiel et en constante évolution.
Cette généralisation de l'Agile n'est pas sans conséquence sur le test. Comme vous pouvez le lire dans le premier livre du CFTL publié en 2019, un développement en Agile est un développement itératif proposant de petites évolutions régulières aux utilisateurs du produit. Ces déploiements, qui étaient auparavant limités à moins de 5 par an, peuvent maintenant se retrouver sur une échelle totalement différente et l'on peut connaître plusieurs déploiements dans la journée. De manière générale, on observe actuellement des déploiements qui correspondent à la fin de chaque sprint, donc de l'ordre d'un déploiement toutes les 2 à 4 semaines. Cette multiplication des déploiements nécessite une multiplication des exécutions. Ce qui faisait barrière à l'automatisation avec un ROI parfois difficile à atteindre n'est plus un sujet ! Ne pas automatiser l'exécution de ces tests revient à ne plus pouvoir livrer ou ne plus pouvoir proposer une campagne de régression suffisante.

L'automatisation de l'exécution des tests est devenue la norme ! On demande donc maintenant aux testeurs d'automatiser ces tests… Ce qui nous amène à un autre problème : une proportion non négligeable de testeurs n'a pas les compétences techniques pour automatiser des tests. Cette problématique a vite été repérée, c'est pourquoi de nombreuses formations sont proposées. Au-delà de la formation, de nouveaux outils d'automatisation plus accessibles, aux profils fonctionnels, sont apparus. Je parle ici des outils de Keyword Driven Testing (KDT) dont le représentant le plus connu est Robot Framework mais il y existe aussi des outils encore plus accessibles, comme Agilitest et Horustest. Le but de ces outils est de proposer des tests beaucoup plus lisibles, compréhensibles. Cela permet donc à des profils fonctionnels de pouvoir plus aisément maintenir et scripter des tests automatisés. Ces outils représentent également un très bon point d'entrée pour comprendre l'automatisation des tests et ses contraintes.

4- L'extension de l'automatisation des tests à l'ensemble des activités de test

L'automatisation de l'exécution des tests est maintenant assez mature dans de nombreuses entreprises ou équipes. Ces dernières sont néanmoins toujours soumises à des problématiques de temps toujours plus contraignantes. Le temps d'exécution des tests n'est maintenant plus vraiment un problème pour ces équipes. Les goulots d'étranglement sur les tests se trouvent maintenant sur d'autres activités qui, il y a encore quelques années, n'attiraient pas l'attention mais qui, de nos jours, peuvent devenir des points de tensions. Ces goulots d'étranglement ne sont plus forcément les mêmes en fonction des équipes et peuvent impacter toutes les activités de test.

La conception et l'écriture des tests, la gestion des environnements et des données ainsi que la génération des cas de test peuvent prendre un temps non négligeable. Il peut donc être intéressant d'automatiser ces activités. Pour la partie automatisation de la conception il existe des outils de Model Based Testing (MBT) comme Yest ou MaTeLo. Cette automatisation de la conception et de l'écriture des tests fait d'ailleurs partie des axes d'évolution de l'automatisation d'après le rapport du World Quality Report 2020-2021 :



Comme vous pourrez le constater dans ce livre, il y a de nombreuses autres activités qui peuvent être automatisées, comme les indicateurs, l'édition des bilans, la gestion de la traçabilité mais aussi la maintenance des campagnes de test qui font de plus en plus face au paradoxe des pesticides.

4- L'émergence de l'IA dans le support aux activités de test

Depuis plusieurs années, l'IA pour le test a d'abord été un slogan : les techniques d'intelligence artificielle allaient changer la donne et remplacer les testeurs. En 2021, nous n'en sommes plus là ! L'IA, et plus précisément les algorithmes d'apprentissage automatique – Machine Learning en anglais - commence à infuser, de façon pratique et utile, dans l'outillage de support aux activités de test.
En effet, les algorithmes d'apprentissage automatique permettent d'analyser des données pour classer ces données, prédire une situation ou générer de nouvelles données. L'apprentissage sur des données permet de mettre en place un modèle capable, en fonction de la qualité des données utilisées (et donc avec un risque de biais), de réaliser l'une et/ou l'autre de ces tâches.

Ces capacités de classement, prédiction et génération ont une forte utilité dans le support à de nombreuses activités de test. Citons en quelques-unes :

- Classification
    - o Classer des anomalies pour éliminer les faux-positifs, automatiser l'analyse d'impact et l'identification des causes
    - o Classer des logs pour visualiser les patterns d'usage des logiciels par les utilisateurs et identifier les parcours à couvrir
    - o Reconnaissance d'objets IH/M pour la maintenance des tests automatisés
    - o Analyse comparée du rendu d'une application web sur différents navigateurs, par apprentissage sur les images du rendu et détection d'anomalies
- Prédiction
    - o Prédiction d'anomalies dans un système
    - o Sélection & priorisation des tests à exécuter (cf. chapitre II-12)
    - o Estimation de risques pour une release
    - o Estimation de l'effort de test à partir de l'extraction de différentes caractéristiques d'une base de projets
    - o Assistance aux tests exploratoires par aide à la différentiation des parcours (cf. chapitre II.y)
- Génération
    - o Génération de données de tests à partir de l'apprentissage sur les cas en production
    - o Génération de scripts de test à partir des logs en production par analyse de patterns d'usage (cf. chapitre II-7)
    - o Correction d'objets GUI dans un script d'automatisation pour l'auto-maintenance des tests automatisés

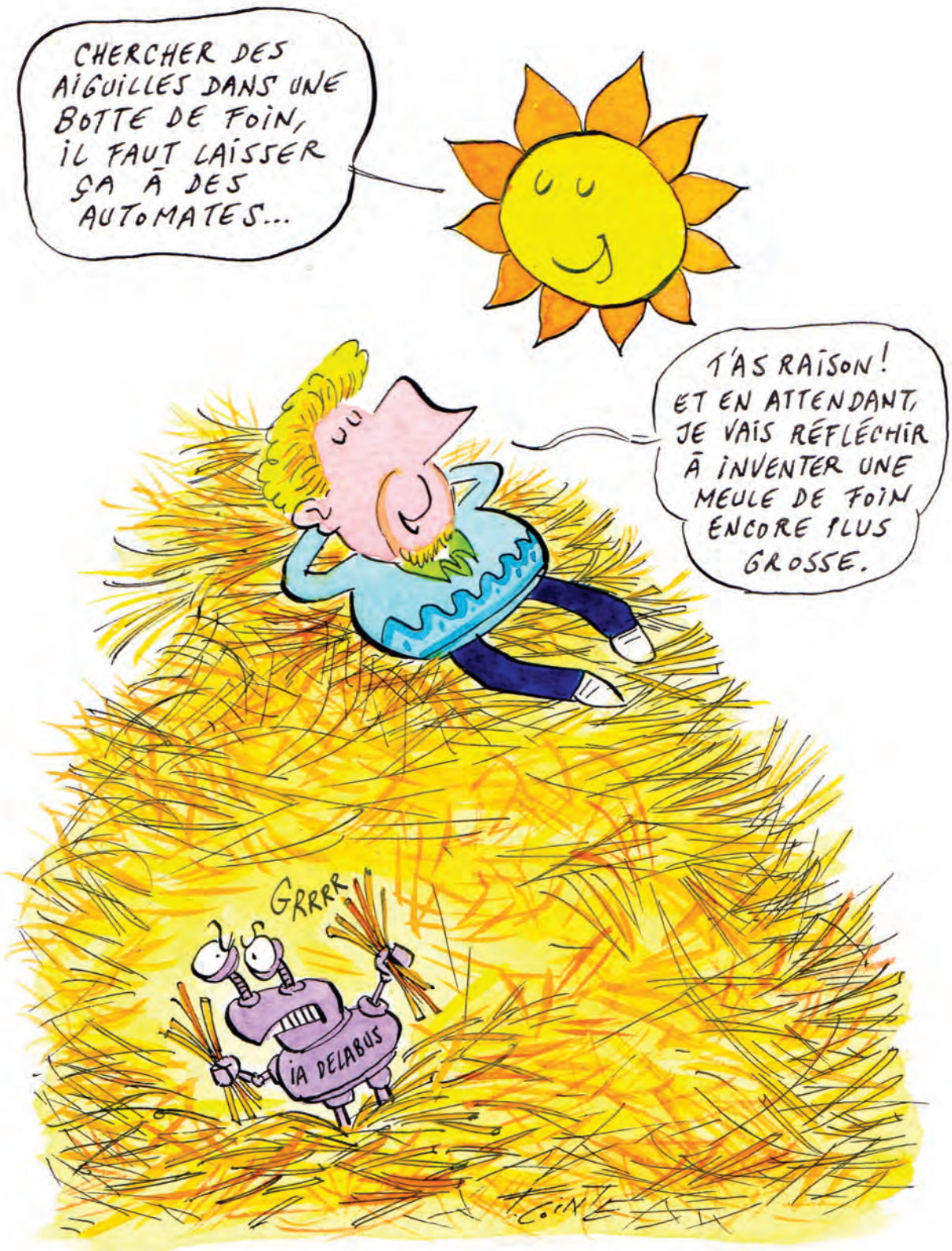
Ce ne sont que des exemples, mais qui montrent le large panel d'activités de test que les techniques d'apprentissage automatique peuvent aider à réaliser. Cela implique en premier lieu d'avoir des données d'apprentissage de qualité, ce qui constitue une barrière importante pour plusieurs de ces activités. Prenons l'exemple de l'estimation de l'effort de test à partir d'une base de projets. Si l'enregistrement des données de suivi de ces projets a été réalisée de façon incomplète ou même inexacte, alors les algorithmes d'apprentissage ne pourront pas fonctionner correctement. C'est la même chose sur la classification des anomalies, il faut que le référentiel de données soit suffisamment complet et fiable.

D'autres applications, comme la génération de tests de régression à partir de l'analyse des traces d'exécution, s'appuient sur les traces d'usage enregistrées dans le suivi analytique ou dans les logs, qui constituent un ensemble de données existant, volumineux et disponible. Ces données de production sont de plus présentes car nécessaires au monitoring applicatif, et elles constituent un réservoir qui ne demande qu'à être exploité pour les tests.

L'arrivée de l'IA – de l'apprentissage automatique – pour l'automatisation des activités de test est devenue une réalité de 2021, mais aussi un axe fort du renforcement de cette automatisation pour les années à venir. Car finalement, l'activité d'un testeur est pour une grande partie l'analyse de données disponibles et une IA peut aider, car les algorithmes d'apprentissage sont faits pour cela.

5- Conclusion

L'automatisation de l'exécution des tests a mis du temps à s'imposer dans le monde du test. Il a fallu le développement de l'Agile pour la rendre indispensable et utilisée par la grande majorité des équipes. Le développement de cette activité a permis de définir de nouvelles activités d'amélioration, afin de permettre aux testeurs de répondre aux contraintes de l'Agile et du marché, qui exigent une réactivité et une adaptation toujours plus rapides.


CHERCHER DES AIGUILLES DANS UNE BOTTE DE FOIN, IL FAUT LAISSER ÇA À DES AUTOMATES...
T'AS RAISON ! ET EN ATTENDANT, JE VAIS RÉFLÉCHIR À INVENTER UNE MEULE DE FOIN ENCORE PLUS GROSSE.
GRRRR
IA DELABUS
T. COINTE

I.2- Pourquoi automatiser ?

par Marc Hage Chahine

Avant d'automatiser, il faut savoir pourquoi l'on souhaite le faire. Il ne faut pas se le cacher, se lancer dans un projet d'automatisation, c'est initier une démarche qui demandera de nombreux efforts et qui est comporte des pièges à éviter… ou à résoudre lorsque l'on y est confronté. Il est donc important, avant de commencer à automatiser, de savoir pourquoi l'on veut automatiser et si le jeu en vaut la chandelle.

1- Les raisons d'automatiser pour les entreprises

Lorsque l'on aborde le sujet de l'automatisation de l'exécution des tests ou de toute autre activité d'automatisation du test, on met en avant les intérêts de l'entreprise qui développe le logiciel. Ces intérêts sont divers et variés mais on peut, pour simplifier, les diviser en 4 familles :

1.1- Le gain ou plus exactement l'économie d'argent

Cette raison est la raison principale mise en avant dans les entreprises. Les tests, encore plus dans les méthodes Agiles, peuvent vite coûter cher. L'automatisation d'une activité de test (le plus souvent l'automatisation de l'exécution) est donc un investissement qui vise un retour sur investissement à moyen terme. Ce retour sur investissement est d'ailleurs l'objet d'un indicateur qui vise à le calculer, le fameux ROI. Cet indicateur reste assez complexe à calculer car il faut être capable d'évaluer les coûts exacts des diverses activités manuelles et automatisées.
Il est bon de noter que le retour sur investissement se fait rarement sur le court terme mais plutôt sur le moyen terme. Etant conscient de cette caractéristique, il m'est arrivé de faire le choix dans un contexte d'urgence (produit à délivrer en 3 mois) de décider de ne pas me lancer dans l'automatisation, afin de ne pas investir du temps et de l'argent dans une automatisation qui n'aurait pas pu offrir un retour sur investissement dans le temps imparti… Et qui aurait même mis en péril la date de livraison du produit.

1.2- Le gain de temps

Gain de temps ne signifie pas forcément gain d'argent ! Dans certains contextes, on peut être amené à vouloir automatiser certaines activités même si l'on sait que le retour sur investissement financier ne sera pas présent (c'était d'ailleurs l'objet de mon premier audit).
Le critère du gain de temps est un critère difficilement chiffrable et mesurable (même s'il existe des indicateurs pour cela) mais prépondérant dans le choix d'automatiser. En effet, ce gain de temps peut par exemple permettre de :

- Réaliser l'ensemble des tâches nécessaires dans le temps imparti
- Libérer du temps aux testeurs pour leur permettre de s'atteler à des tâches à plus forte valeur ajoutée, comme par exemple travailler sur de l'amélioration continue (amélioration de processus, amélioration des campagnes…)

- Mettre à disposition plus tôt le logiciel, une de ses versions ou plus simplement des fonctionnalités ou correctifs de ce dernier.
- Faire plus de tests afin d'assurer une meilleure couverture (ce qui est en rapport avec la 4ème famille).

1.3- Implémenter l'intégration continue dans une démarche DevOps

Le DevOps, que je considère simplement comme de l'Agile qui englobe vraiment l'ensemble des acteurs travaillant sur un produit, rime souvent avec chaîne d'intégration continue (même si les chaînes d'intégration continue sont des outils et donc non liées à la démarche DevOps). Ces chaînes d'intégration continue qui peuvent aller jusqu'au déploiement continu, c'est-à-dire un déploiement direct sur l'environnement de production, nécessitent des tests. Les chaînes d'intégration continue se doivent d'être rapides. Personne n'accepte qu'une simple fusion de branche (merge) prenne plus de 15 minutes ! La seule manière de répondre à ce besoin de rapidité est une automatisation de l'exécution des tests. Cette automatisation est d'ailleurs parfois insuffisante et force à s'adapter en proposant une parallélisation des tests ou encore en retravaillant la campagne (moins de tests, sélection de tests, optimisation du temps d'exécution des tests en eux-mêmes…).

1.4- « Améliorer la qualité »

Ce point est souvent mis en avant par les entreprises. L'automatisation doit améliorer la qualité du logiciel. Cela peut être le cas dans le sens où l'automatisation des tests peut permettre de :

- Tester plus tôt et donc potentiellement corriger des anomalies qui n'auraient pas pu être corrigées plus tard (car trop cher)
- Tester plus (plus de tests) et plus souvent avec une campagne automatisée complétée par des tests exploratoires
- Libérer du temps pour améliorer les processus.

Néanmoins, l'automatisation n'engendre pas forcément une amélioration de la qualité. Il est d'ailleurs fréquent d'avoir au début des projet d'automatisation une baisse de cette dernière en raison de l'investissement fourni pour l'automatisation ou encore de failles dans cette automatisation.

2- Les raisons d'automatiser pour les personnes

Automatiser pour des raisons liées à l'entreprise est naturel et obligatoire. Il ne faut cependant pas oublier que derrière ces stratégies d'entreprise, il y a des personnes… Et que ces personnes peuvent également avoir leurs propres raisons de vouloir automatiser.
Pour un testeur, l'intérêt d'automatiser peut être traité en 2 points :

2.1- Quels sont les leviers de motivation pour l'automatisation ?

Nous n'aborderons pas ce point dans ce chapitre car il est particulièrement bien développé dans l'article de Zoé Thivet dans la dernière partie de ce livre (chapitre III-2).

2.2- Pourquoi, de manière générale, les testeurs automatisent leurs activités ?

L'automatisation est naturelle dans notre vie quotidienne. Nous y faisons appel instinctivement tous les jours et ce depuis l'enfance. Cette automatisation instinctive se fait souvent lorsque l'on se retrouve dans l'un des 2 cas suivants :

- Lorsque nous sommes soumis à l'exécution d'une tâche répétitive.

L'automatisation des tâches répétitives dépasse le simple cadre du logiciel. On le voit notamment dans l'industrie. Le temps du film « Les temps modernes » est révolu, quand notre Charlot préféré se retrouvait à serrer des boulons pendant des heures. Cette tâche particulièrement répétitive a été automatisée.
Cette automatisation des tâches ingrates, répétitives et à faible valeur ajoutée ne se limite d'ailleurs pas uniquement au travail ou à l'âge adulte ! En effet, je me remémore régulièrement, avec un léger sourire, ce concours de « techniques » utilisées à l'école par mes camarades et moi-même, lorsqu'il s'agissait de faire nos punitions. Ces punitions étaient généralement la copie de (trop) nombreuses lignes d'une phrase qui ressemblait à : « Je ne parlerai plus à mon voisin pendant les cours ». Nos techniques allaient de l'utilisation d'un papier calque (pas très efficace), à l'utilisation de plusieurs stylos en simultané, en passant par la tentative de le rendre en version Word imprimée ou même demandant à des copains de nous aider en faisant quelques lignes (à charge de revanche !).

Tous ces moyens de limiter l'impact de la punition sont en fait des prémices de l'automatisation de tâches à faible valeur ajoutée et faible valeur actuelle. Le type de tâche sur lesquelles on ne veut pas s'attarder afin de passer à autre chose.

- Lorsque la probabilité d'erreur dans la tâche que nous exécutons est forte.

L'automatisation, c'est aussi l'outillage. On se souvient tous d'actions simples à effectuer mais tellement répétitives qu'au final, on se trompe. Cela arrive souvent lorsque l'on est encore à l'école et que l'on doit résoudre des problèmes. Au début, pour éviter ce genre de problème, on décide simplement de faire des vérifications et de refaire plusieurs fois la suite d'actions pour s'assurer que notre résultat est le bon… Sauf que généralement, si l'on ne s'est pas trompé la première fois, il reste probable que l'on se trompe lors de la vérification… Ce qui nous fait perdre énormément de temps.

Un exemple concret est celui du calcul de moyenne (technique utilisée par ma professeure de mathématiques en 1ère et que j'ai réutilisée ensuite). Une moyenne est une somme de toutes ses notes divisée par le nombre de notes. Les nombres à additionner sont souvent très grands la somme devient alors compliquée avec une probabilité d'erreur importante. Il est possible de remplacer toutes les notes par une note à laquelle on soustrait la moyenne (10). La moyenne effective devient plus facile à calculer car les nombres à manipuler sont plus petits comme on peut le voir avec l'exemple ci-dessous :

Notes: 12; 14; 8: 13; 15 donne (2 + 4 – 2 + 3 + 5)/5 + 10 = 12/5 + 10 = 12,4 de moyenne.

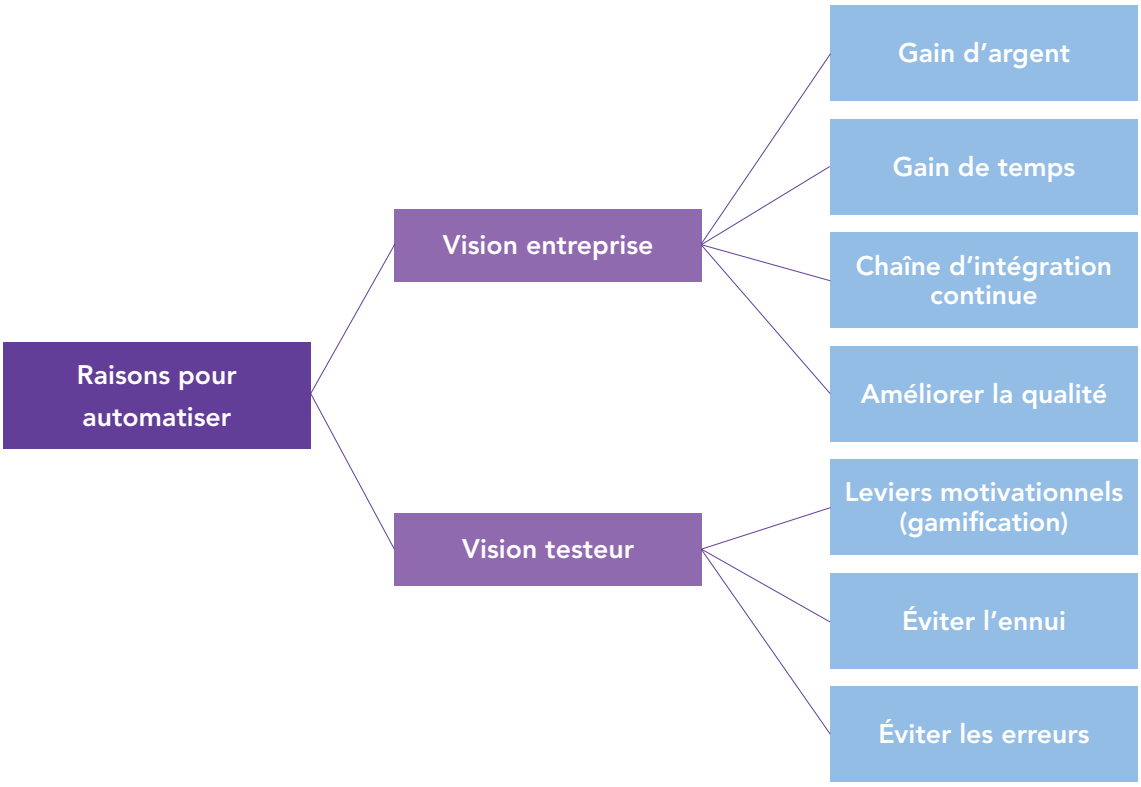
3- Que retenir ?

Nous automatisons nos activités pour plusieurs raisons.
D'un point de vue pratique, les objectifs de l'automatisation des activités de test sont généralement liés à un besoin de limitation des coûts et/ou des délais ou des impératifs, comme une amélioration de la qualité, ou la nécessité d'avoir de l'automatisation dans le cadre d'une stratégie basée sur la mise en place de chaînes d'intégration continue.
D'un point de vue humain, les objectifs sont principalement liés à du ressenti et au besoin d'éviter un mal-être. Ce mal-être est généré par l'accumulation de tâches répétitives à faible valeur ajoutée (et entraînant l'ennui) ou au stress de l'erreur dû à l'accumulation des actions.
Fort heureusement les objectifs, même différents, ne se contredisent pas. En effet, on peut voir que dans les 2 cas, l'automatisation est mise en place pour :

3.1- Nous faire gagner du temps : gain de temps et d'argent ainsi que l'intégration continue pour les entreprises ; éviter de passer trop de temps sur certaines tâches pour les testeurs.

3.2- Améliorer la qualité : objectif direct pour les entreprises, éviter les erreurs sur certaines tâches et l'ennui (qui induit un manque de concentration) sur d'autres pour le testeur.

Cela peut être résumé avec le schéma suivant :



Quelles activités automatiser ?

1- A quelle activité pensons-nous lorsque l'on parle d'automatisation dans le test ?

Instinctivement, lorsque l'on parle d'automatisation de test ou de ses activités, on pense à l'automatisation de l'exécution des tests. Cette vision est due, au moins en partie, à l'image encore persistante du travail de testeur qui ne fait qu'exécuter des tests. Il est intéressant de noter que cette vision peut aussi expliquer, en partie, pourquoi certaines personnes voient arriver la fin des testeurs avec la démocratisation des méthodes agiles.
Cet instinct, visant à automatiser l'exécution des tests scriptés, semble néanmoins cohérent lorsque l'on réfléchit à l'activité de l'exécution de ces tests et qu'on le voit à travers le prisme des objectifs de l'automatisation.
En effet, l'exécution des tests scriptés est une activité qui prend du temps à des testeurs qui sont payés (temps et argent pour l'entreprise), une activité incompatible avec l'intégration continue et qui pousse à ne pas tester trop souvent (impact qualité). De même, surtout en Agile, cela peut vite devenir une tâche rébarbative pour le testeur, ce qui entraîne une hausse des probabilités d'erreur lors de l'exécution.
L'exécution des tests scriptés est donc une candidate parfaite à l'exécution des tests mais est-ce la seule ?

2- Quelles activités de test pouvons-nous concrètement automatiser ?

Cela n'est évidemment pas le cas. La vie d'un testeur ne se résume fort heureusement pas à l'exécution de tests scriptés. Un testeur a de nombreuses activités variées qu'il serait trop long d'énumérer de manière exhaustive. On peut néanmoins citer des activités très fréquentes comme :

- La conception de tests
- La préparation des campagnes (données, environnements, tests…)
- La maintenance des tests
- L'exécution de sessions de tests exploratoires
- L'analyse des résultats des campagnes
- Implémenter et améliorer les processus de test
- Initier et contribuer au plan de test
- Créer et suivre les fiches d'anomalie
- En Agile : participer aux activités de l'équipe et faire des tâches non spécifiques au testeur…

Certaines de ses activités sont à forte valeur ajoutée. Je pense notamment à la conception et l'exécution des campagnes de test. Néanmoins, même pour ces activités, il existe des étapes ou des actions qui ont moins de valeur ajoutée et qui peuvent être automatisées. C'est d 'ailleurs pour cela que des outils de MBT (comme Yest et MaTeLo) pour la conception et des outils de support aux tests exploratoires (comme AIFEX) existent.

D'autres activités ont moins de valeur ajoutée et peuvent donc être des candidates à l'automatisation (au moins en partie) très intéressantes. Je pense notamment à :

2.1- La maintenance des tests de régression

Cette maintenance peut vite être chronophage et fastidieuse. Elle est d'ailleurs une cause majeure des échecs des projets d'automatisation des tests. Fort heureusement, elle peut être automatisée en partie ! Une manière d'automatiser partiellement la maintenance est de concevoir un automate de test avec une architecture modulaire qui adopte les bonnes pratiques du code (notamment avec une bonne factorisation). Cela permet de mettre à jour en 1 fois chaque étape commune à plusieurs tests !

2.2- L'édition des bilans des campagnes

L'édition des bilans des campagnes est une tâche particulièrement importante. Le bilan est la vitrine des tests, c'est LE document qui sera consulté par les non testeurs, c'est LE document qui importe… Bref c'est le fruit de tout le travail du testeur sur une campagne. De manière générale, il semble donc peu intéressant de l'automatiser. Néanmoins, on remarque qu'avec la multiplication des campagnes (avec l'automatisation des tests scripts) et la standardisation du contenu du bilan de ces dernières, cette tâche peut vite devenir fastidieuse mais aussi un goulot d'étranglement. Dans ce cas, il devient nécessaire d'automatiser au moins en partie ces bilans en fonction du résultat des tests. Pour cela, il sera probablement obligatoire de définir certains critères liés à des indicateurs comme donner un « Go » uniquement si aucun bug n'est remonté par la régression.

2.3- La gestion des environnements et des données de test

La création d'environnement de test est primordiale pour pouvoir tester efficacement, c'est d'ailleurs un élément obligatoire pour être TMMI niveau 2. Il est même parfois conseillé d'avoir un environnement créé spécialement pour chaque campagne (cela assure la présence des données) pour être impacté le moins possible par des éléments extérieurs. Cette création est néanmoins assez technique et chronophage, ce qui explique que selon le CFTL, le métier de gestionnaire des environnements de test est un métier à part entière. De plus, elle fait appel à des compétences que n'ont pas forcément les testeurs. Il semble donc intéressant d'automatiser ces créations d'environnement, ce que fait un outil comme Docker.
Il est important de rappeler qu'il n'existe pas d'environnement de test et de test sans données de test. Il est donc nécessaire de créer les environnements avec les données nécessaires ou encore de générer ces données après la création des environnements ,ce qui peut permettre, dans certains contextes, de vérifier que la création d'éléments proposés par le logiciel fonctionne correctement.

2.4- L'écriture des cas de test après leur conception

La conception est une étape essentielle et peu automatisable. Néanmoins, après avoir conçu son test, si ce test est destiné à être scripté, il faut l'écrire. Cette écriture se doit d'être rigoureuse et, si possible, permettre une maintenance limitée… Cette activité peut vite se retrouver

fastidieuse et technique sans proposer de forte valeur ajoutée, ce n'est pas un hasard si les tests exploratoires la suppriment.
Les outils de MBT comme Yest et MaTeLo aident fortement à l'automatisation de cette partie en proposant même d'aller jusqu'à l'écriture de script automatisés pour Yest et une exécution de ces derniers pour MaTeLo.

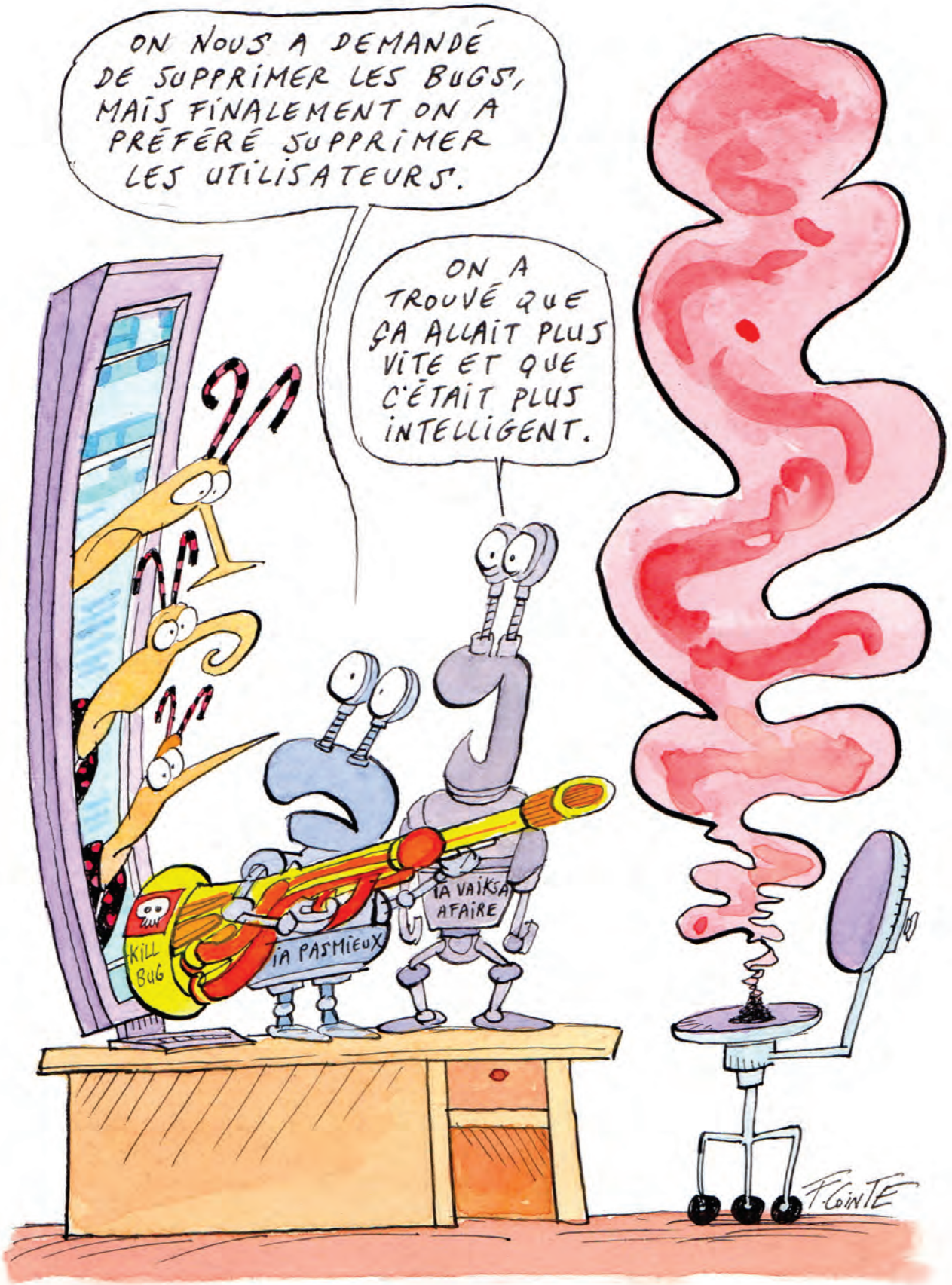
2.5- La création des fiches d'anomalies

Les fiches d'anomalies sont des livrables particulièrement importants dans lesquels il faut mettre autant d'informations que possible, afin que n'importe quel acteur sur le logiciel (métier, testeur, développeur) puisse les reproduire et les comprendre. Ces paramètres peuvent être très nombreux et il est facile d'en oublier. Là encore l'automatisation peut nous aider en pré-remplissant des informations permettant de reproduire l'anomalie.

3- Conclusion

On peut avoir un intérêt à automatiser, au moins en partie, toute activité de test. Automatiser l'ensemble de ces activités est néanmoins une mauvaise idée dans la quasi-totalité des contextes. En effet, à vouloir tout automatiser, on peut vite se retrouver devant une machine qui sera très peu compréhensible, très peu flexible mais aussi particulièrement chère et ennuyante à maintenir… Ce qui rendrait l'automatisation contre-productive par rapport à ses objectifs définis précédemment.
Pour savoir ce qu'il est intéressant d'automatiser, il faut avant toute chose connaitre l'impact des diverses activités et voir ce que leur automatisation partielle ou complète peut apporter. De même, il est important de prioriser les activités que l'on souhaite automatiser. Pour cela, je conseille généralement de cibler les « points de douleurs ». Qu'est-ce qui nous dérange ? Qu'est-ce qui nous ralentit ? Qu'est-ce qui nous handicape ? Lorsque l'on sait répondre à ces questions, la priorisation des activités de test à automatiser devient une évidence et il devient alors possible d'étudier l'intérêt d'automatiser certaines activités. Il est alors temps de réfléchir à l'automatisation de ces activités.
Tout comme pour l'automatisation de l'exécution des tests scriptés, afin de réussir l'automatisation de toute activité de test, il faut définir :

- Les objectifs liés à l'automatisation de l'activité en question
- Identifier les solutions existantes et étudier un retour sur investissement (financier ou non) potentiel
- Choisir une solution, qui peut être un logiciel acheté ou un développement interne, si l'on décide d'initier l'automatisation de l'activité suite au point précédent
- Implémenter la solution à travers un POC (Proof Of Concept)
- Prioriser les tâches
- Faire un bilan et mesurer les avancées


ON NOUS A DEMANDÉ DE SUPPRIMER LES BUGS, MAIS FINALEMENT ON A PRÉFÉRÉ SUPPRIMER LES UTILISATEURS.
ON A TROUVÉ QUE ÇA ALLAIT PLUS VITE ET QUE C'ÉTAIT PLUS INTELLIGENT.
KILL BUG
IA PASMIEUX
IA VAIKSA AFAIRE
F.COINTE

I.3- Les tests dans la démarche DevOps

par Jean-Michel Teissier

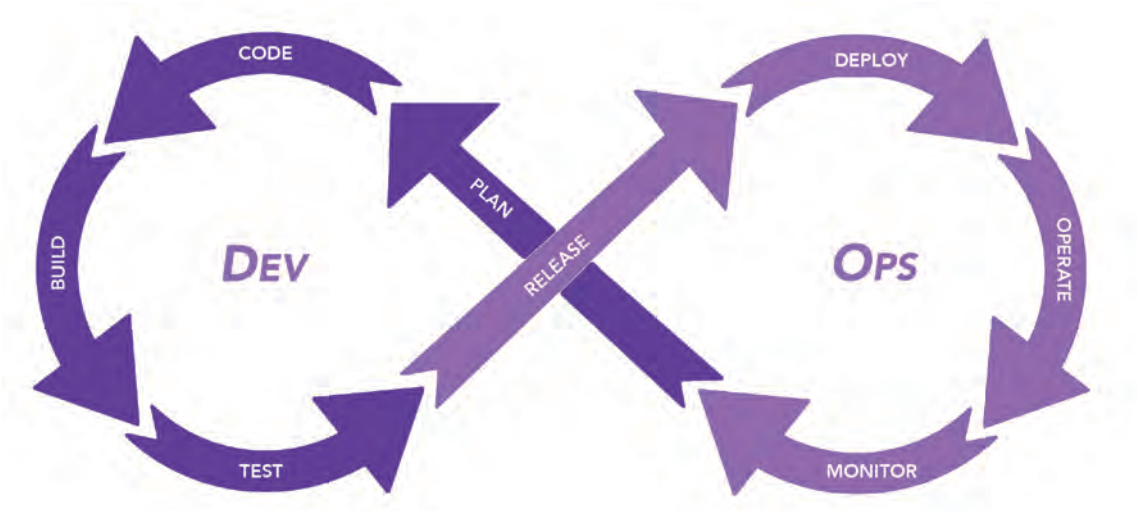


Figure 1 - Représentation du Pipeline DevOps

Intégrer le « Continuous Testing » au sein de votre démarche DevOps
C'est déterminer une stratégie de test et la mettre en place comme pratique de votre organisation.

1- La culture & démarche DevOps

La démarche DevOps possède l'avantage de pouvoir se greffer à tout type d'organisation. L'organisation Agile est une méthode de Delivery alors que DevOps nous amène à travailler différemment.
Adopter la culture DevOps demande une réelle conduite du changement pour l'ensemble des équipes, afin d'apporter l'accompagnement pour structurer l'organisation.
Pour rappel, DevOps n'apporte pas de solution, a contrario, il précise les enjeux. Les sujets étant souvent récurrents, des outils ont émergé pour répondre aux premiers enjeux comme les solutions de CI/CD et autres services partagés (Cloud).
Dans ce contexte, la place du test nécessite d'être positionnée tout au long du pipe-line : Niveau de test - Type de test - Devices – Automate – Configuration – Statique et dynamique.

DevOps s'étend à l'ensemble du processus de production et ne se limite pas aux seuls aspects du développement et des opérations.

A retenir :

- DevOps valorise avant tout une culture d'entreprise
- DevOps est un état d'esprit avant d'être une démarche

Devops promeut une démarche dans laquelle l'ensemble du cycle de vie du produit, schématisé par le pipeline, doit être représenté afin de mieux maîtriser les enjeux de delivery.

1.1- Introduction à DevOps

A retenir :

- DevOps valorise la culture et l'individu
- DevOps est un état d'esprit
- DevOps propose une démarche

1.2- Les ambitions de DevOps

La figure-2, ci-dessous, symbolise l'orientation souhaitée par DevOps

- Avoir une vision complète sur l'ensemble de la chaîne de construction du produit
- Intégrer une démarche agile qui promeut l'interaction des équipes projets
- Mettre en place les indicateurs nécessaires au pilotage de l'activité et anticiper les risques
- Promouvoir la proactivité sur la production

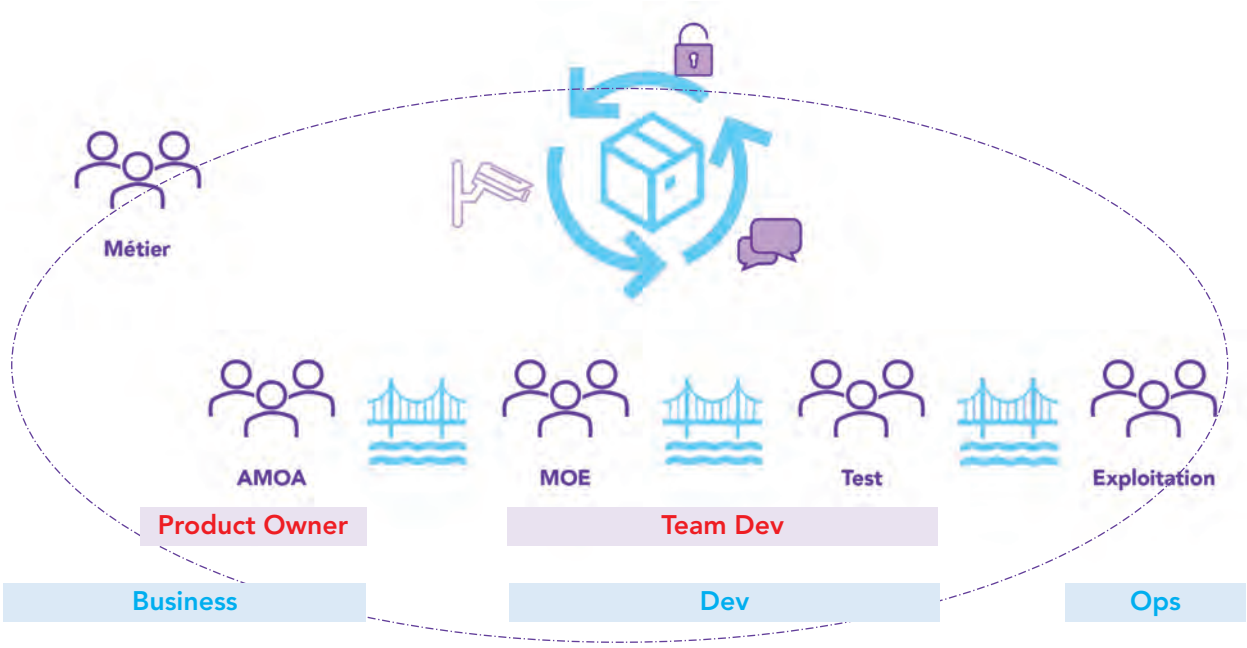


Figure 2 - Les ambitions DevOps

2- Comprendre le Continuous Testing

2.1- Une pratique à part entière et intégrée

Il existe plusieurs approches pour aborder la pratique du Continuous Testing.
Il est indispensable d'appréhender un ensemble de pratiques, afin de déterminer les critères qualité attendus pour mieux définir les approches de test associées.
DevOps met en avant la mise en œuvre du test de façon transverse et ajustée au contexte de chaque équipe.

La figure-3, ci-dessous, montre la transversalité de la pratique.

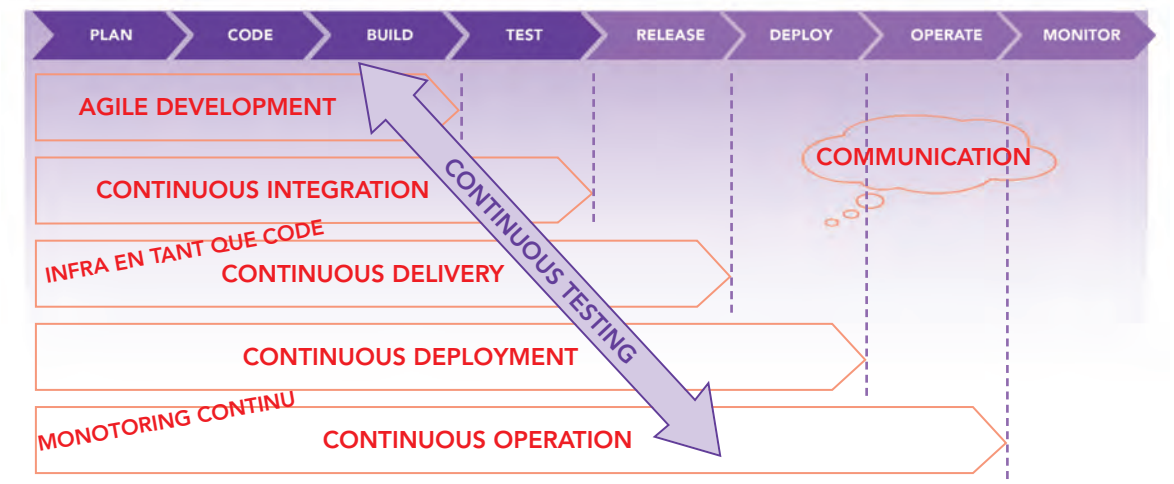


Figure 3- Le continuous Testing et les autres pratiques

Important : Ce schéma peut induire en erreur, car il positionne l'« Agile Development » comme une pratique indissociable de DevOps. Or, un projet qui utilise une méthodologie traditionnelle (cycle en V) aura la même légitimité pour utiliser des pratiques en continu. Et bien que la pratique la plus courante réside généralement dans la mise en œuvre du CI/CD, la mise en place d'une stratégie de test devient rapidement une pierre angulaire aux enjeux de qualité.

2.2- Organiser ses tests

L'ISTQB répartit les tests par quadrant selon quatre axes stratégiques :

- Est-ce que les tests répondent à des enjeux métiers ?
    - Ou permettent-ils d'assurer la qualité des standards techniques ?
- Est-ce que les tests vont permettre de faciliter la construction et l'intégration des différents incréments ?
    - Ou permettent-ils d'accepter le produit au sens métier ?

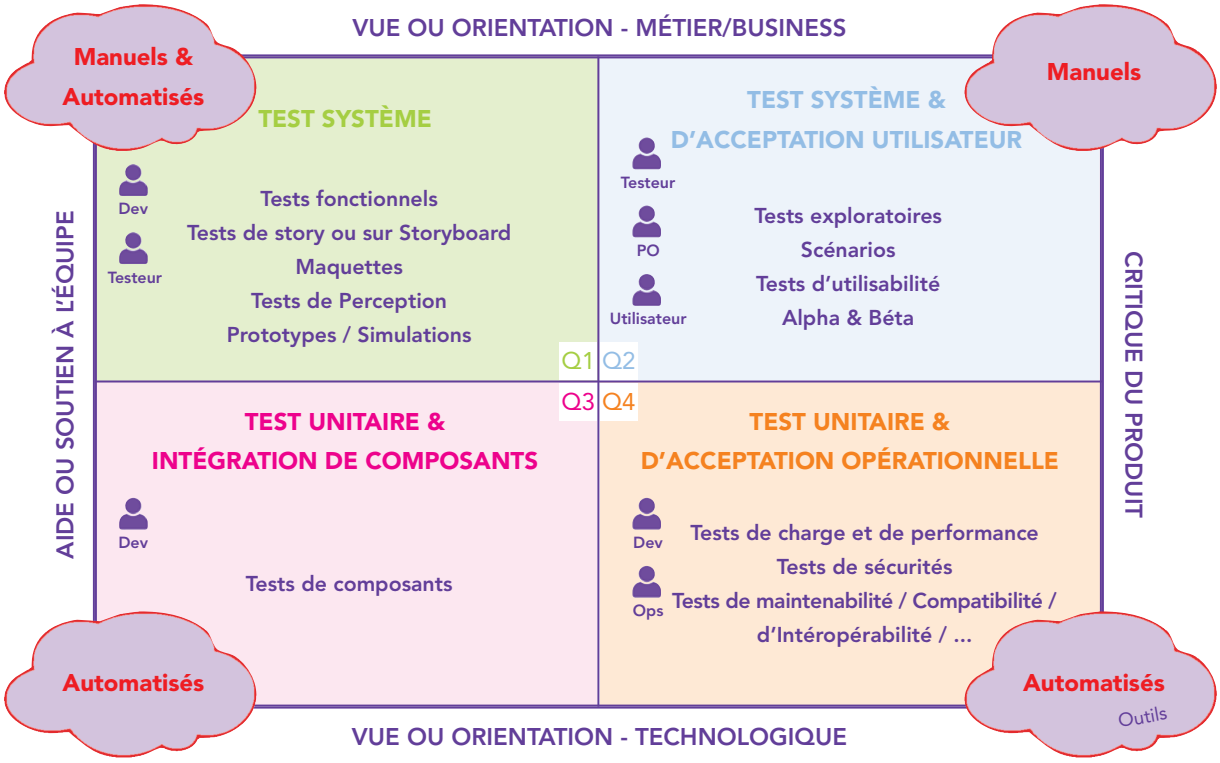


Figure 4 - Quadrant Agile (ISTQB)

Plusieurs approches permettent de décliner cette vision, en précisant les tests qui pourront être pris en compte lors des différentes activités du cycle DevOps (Pipeline).

Figure 5 - Représentation linéaire du Pipeline DevOps

2.3- Les tests au sein du Pipeline

2.3.1- Plan

Exemple de tests pouvant être appliqués :

- Revue des spécifications, User Stories et autres bases de test
  - Identification des exigences et des conditions de test
- Définition des règles d'implémentations
- Testabilité des différents critères
- Etude d'impact

2.3.2- Code

Exemple de tests pouvant être appliqués :

– Tests Unitaires
– Analyse Statique du code

2.3.3- Build

Exemple de tests pouvant être appliqués :

– SAST (Static Application Security Testing)
– Tests d'intégration (Composants)
– Tests de « Type » Boîte Blanche

2.3.4- Test

Exemple de tests pouvant être appliqués :

– Tests fonctionnels (Automatisés / Manuels)
– Tests de Performance
    o Tests de Charge / Tests de Fiabilité
    o Tests aux limites / Tests de Stress
– DAST – Dynamic Application Security Testing
– Tests d'Intégration (Systèmes)

2.3.5- Release

Exemple de tests pouvant être appliqués :

– Tests d'Acceptation
– Tests Exploratoires
– A/B Testing
-– Tests métiers pour valider l'adéquation du produit aux enjeux de l'entreprise

2.3.6- Deploy

Exemple de tests pouvant être appliqués :

– Parler de test en production peut porter à sourire mais lors du déploiement en production, il est primordial de garantir la stabilité et l'opérabilité de votre système. Le passage d'une série de tests représentatifs permettant de valider « quelques » scénarios métiers est recommandé, on parle alors de Smoke Test.

– Tester que les process de déploiement du produit en production n'entraînent pas de régression

2.3.7- Operate

Exemple de tests pouvant être appliqués :

– Tests d'exploitation

– Tester que le produit en production assure les opérations attendues en ayant les qualités de fonctionnement requises

2.3.8- Monitor

Exemple de tests pouvant être appliqués :

– Tests prédictifs

– Recueille des indicateurs de production & de suivi des comportements utilisateurs :

- Sonde de charge
- Suivi des usages à l'aide de solutions de type « Analytics »

Mieux comprendre les usages du produit pour le rendre plus adapté aux réels besoins et aussi affiner ses scénarios de test.

Rappel : Être DevOps c'est adopter l'agilité.

La notion de pipeline mise en avant par DevOps tend à valoriser l'ensemble du processus de mise en œuvre du produit.

2.4- Le Continuous Testing, une responsabilité partagée

De manière classique, tous les intervenants ont un rôle dans la qualité.

2.4.1- Lead DevOps

- *Garant des pratiques DEVOPS*

Le Lead Devops suit l'avancement global du projet et met en avant les principes d'amélioration continue.

Il porte les engagements et intègre la stratégie de test de façon continue en apportant les moyens nécessaires à l'équipe.

2.4.2. Product Owner DevOps

- *Garant du Product Backlog*

Le Product Owner définit l'ensemble des User Stories et précise les critères d'acceptance.

Il valide le produit avant la mise à disposition des équipes métiers.

2.4.3- Scrum Master DevOps

- *Garant de la qualité des travaux de développements*

Détermine les standards de développements et les objectifs de tests unitaires.

Il a un rôle déterminant pour prioriser les travaux en lien avec la Dette Technique du produit.

2.4.4- Développeur DevOps

- *Garant de la qualité du code*

Participe à assurer le niveau de maturité des User Stories.

Réalise des tests unitaires et d'intégration en assurant le taux de couverture attendu.

2.4.5– Release Manager DevOps

- *Garant de l'intégration continue*

Paramètre les campagnes de tests au sein du dispositif.
Met en place les indicateurs qualités nécessaires à établir une vision en temps réel des livraisons d'incréments produits.

2.4.6– Ingénieur Déploiement DevOps

- *Garant du déploiement continu*

Valide la stabilité du produit entre chaque installation sur l'ensemble des environnements.

2.4.7– Release Infrastructure DevOps

- *Garant des environnements*

Valide la stabilité technique des environnements, gère les opérations d'upgrade de versions.

2.4.8– Release Sécurité DevOps

- *Garant de la sécurité du produit*

Détermine la stratégie de sécurité et assure sa mise en place au niveau du produit et des infrastructures.

2.4.9– Testeur DevOps

- *Détermine le niveau de qualité du produit*

Il participe, au côté du Product Owner et des développeurs, à la définition / précision des User Stories et détermine les critères d'acceptance.
Il élabore les cas de test à réaliser et les exécutera soit par tests manuels, soit à l'aide d'outils facilitant leur automatisation.

2.4.10– Test Lead

- Il apporte la connaissance des types et niveaux de tests.

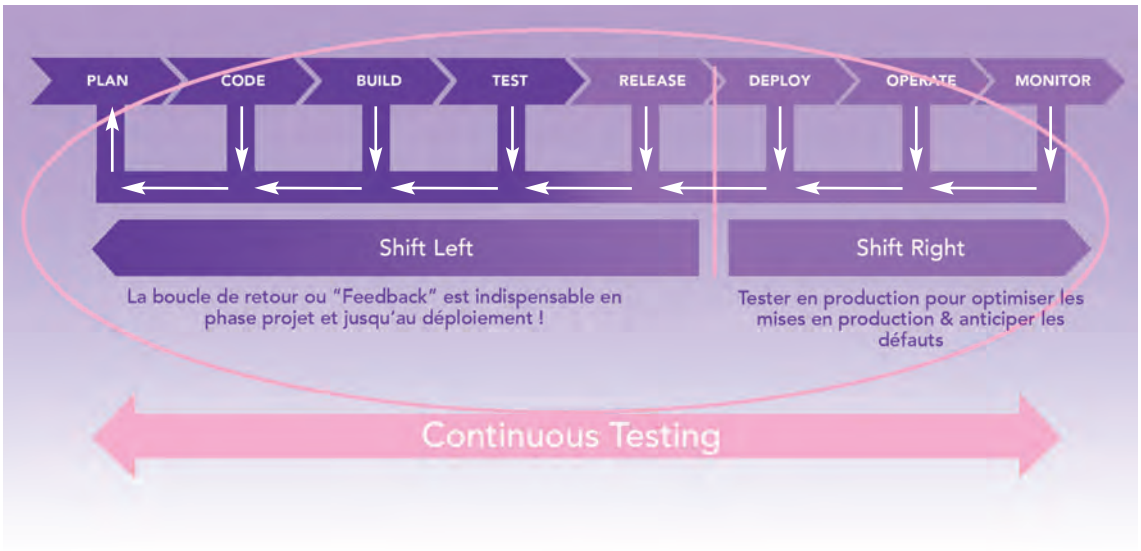
Dans notre démarche, le rôle de Test Manager ou Test Lead prend l'appellation de Coach en Test. Au même titre que le Coach Agile, le Coach en Test est l'ambassadeur des pratiques de test.

⇒ **Anticiper cette répartition des responsabilités et assurer les compétences dans votre équipe est un pilier très important de la réussite de votre transformation.**

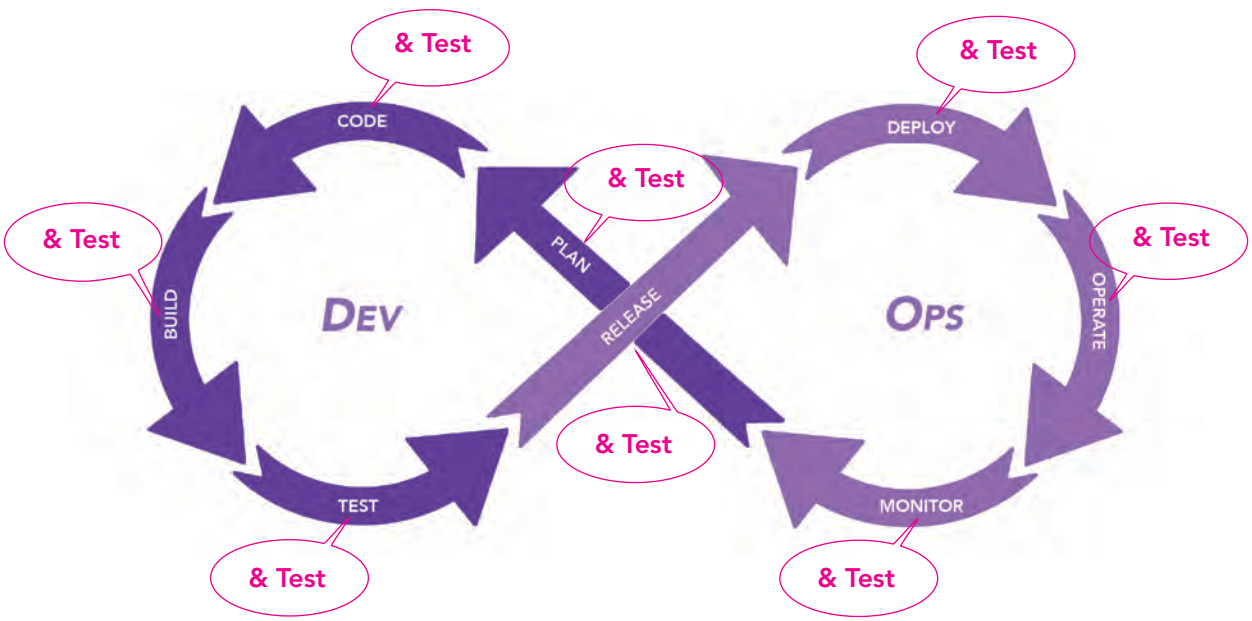
2.5- Adopter une vision Shift Left et Shift Right

La notion du Shift Left est une démarche cohérente qui permet d'anticiper les risques de qualité au plus tôt dans le cycle de vie du projet.

- Détecter les défauts & défaillances au plus tôt
- Comprendre l'origine des causes racines de ces erreurs



Le Shift Right est une approche qui permet, dans un premier temps, de détecter les anomalies en production dès qu'elles surviennent.
Les stratégies digitales recommandent la mise en place de scénarios prédictifs pour mieux anticiper les futures défaillances.



Envisager le Continuous Testing nécessite une conduite du changement et un accompagnement sur mesure, afin d'intégrer au sein de votre pipeline DevOps l'ensemble des actions attendues de manière itérative.

3- La transformation digitale orientée par les tests

Avant de commencer à entamer toute transformation, il est structurant de clairement définir sa première trajectoire.

– Quels objectifs atteindre ?

– Quel est mon niveau actuel ?

• *Fixer les objectifs à atteindre.*

Le Continuous Testing nécessite d'avoir une vision à 360° :

– Quels sont les risques produit et les risques projet ?

– Quels sont les taux d'engagements attendus ?

– ….

⇨ **La couverture des risques et la tenue des engagements doit être l'objectif n°1.**

• *Déterminer son niveau de maturité en test*

Quelle est votre politique de test, votre stratégie de test, quel est votre niveau de maturité actuel ?

– Quels sont les tests actuellement réalisés ?

– Comment sont suivies les activités de test ?

– …

⇨ **Le test est un moyen d'assurer vos objectifs qualité !**

Pour réaliser l'évaluation initiale, l'utilisation de référentiel comme TMMI (processus de définition de la maturité d'une organisation de test) est fortement recommandée. Par expérience, l'équipe Océane Consulting Testing Services recommande cette pratique.

3.1- De la stratégie à l'approche de de test

Lors de la définition de votre plan de test maître, plusieurs axes sont à intégrer :

– Identifier les exigences

• Enjeux métiers / Défis techniques / Contraintes / Priorités / …

– Identifier les risques

• Régression / Testabilité / Criticité du produit / Risques Projet / ….

– Définir les objectifs

• Enjeux métiers / Critères d'acceptation / ….

– Organiser les tests

• Niveaux de test / Types de test / Rôles & Responsabilités / ….

– Piloter les tests

• Evaluer l'effort de test / Coordonner les activités de test / Mesurer la qualité / Assurer le suivi de la « dette Technique » / …

– Collecter et suivre les métriques

- Qualité du produit
- Taux de défauts par exigence / Dette technique / Taux de couverture / …
- Pilotage des activités de test
- Effort de test
- Ratio des tests automatisés
- Nombre de Faux
- …

Important : cette liste peut aussi devenir un piège : penser à déterminer les grands axes de suivis afin de collecter des informations utiles.

– Et vos propres préoccupations…

3.2- Mettre en place la « solution » de Test

Définition : ensemble des dispositifs et outils mis en place pour assurer la mise en œuvre des tests

- La gestion et le suivi,
- L'analyse,
- La conception,
- L'implémentation,
- L'exécution,
- Le reporting.

⇒ **La solution de test intègre l'ensemble des outils nécessaires pour la réalisation des tests manuels et automatisés, la remontée des défauts constatés et leurs suivis.**

– Définir l'ensemble des outils

- Il est structurant d'unifier au maximum les processus de test

– Identifier l'ensemble des environnements

- Chaque environnement ayant des objectifs de tests spécifiques, il sera nécessaire de s'assurer de la portabilité et de l'interopérabilité des outils

– Définir un process de gestion des données de test

- La constitution des jeux de données de test apparaît comme une des préoccupations majeures des projets.

⇒ **Les pièges de l'automatisation :**

– Tous les tests ne seront pas automatisables

- L'automatisation doit être menée pour garantir un gain à long terme.

– Maintenir son patrimoine de test

- I est classique de trouver des tests manuels ou automatisés obsolètes qui deviennent un handicap sur le projet. La maintenance du patrimoine de test est un enjeu très important.

3.3- Assurer une mise en place progressive

La mise en place des pratiques DevOps nécessite de déterminer des étapes clés.
Dans l'approche que nous avons développée, au sein d'Oceane Consulting TS, nous privilégions une vision incrémentale de cette mise en place.

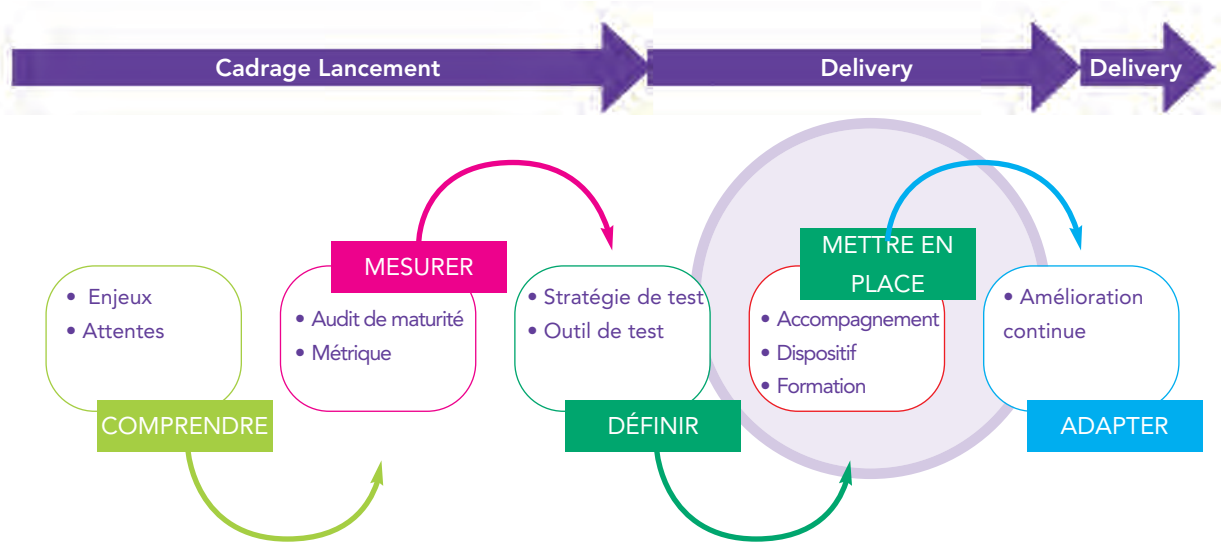


Figure 7 - L'approche Oceane Consulting TS

- Vouloir mettre en place votre stratégie de test trop rapidement comporte des risques majeurs :
  - Une cible non maîtrisée
  - Une perte de rentabilité
  - Des priorités mal définies
  - Imposer de mauvaises « deadlines »

- Définir la couverture des exigences en maîtrisant votre patrimoine de test
- Prévenir les risques en précisant vos taux de défaillance
- Assurer un véritable suivi de vos Faux Positifs et Faux Négatifs
- …

Un accompagnement par des équipes expérimentées vous permettra d'anticiper ces pièges et vous permettra de mieux assurer votre démarche.

4- Conclusion

Le choix des outils est important pour d'une organisation DevOps, la réussite s'obtient avant tout par une organisation transverse et complémentaire de l'équipe.
L'intégration du Continuous Testing impose un véritable changement culturel de l'entreprise dans lequel chacun doit être un acteur.

- En quoi cette stratégie va donner un nouvel élan à nos projets ?
- Quelles sont les étapes importantes ?
- Comment ce changement va m'impacter personnellement ?
- Serons-nous accompagnés ?
- Les outils choisis répondront-ils à tous ces enjeux ?
- …

4.1- Les points clés à anticiper

- Avoir une stratégie de test propre aux enjeux du projet
- Anticiper les changements sur l'organisation
- Evaluer les compétences
- Renforcer l'équipe
- Assurer la participation de tous
- Piloter et adapter les activités régulièrement

A noter : **Privilégier des éditeurs et solutions matures vous permettra de focaliser votre attention sur les véritables enjeux de votre projet.**

4.2 Les bénéfices du Continuous Testing

L'adoption de cette pratique dans votre stratégie digitale vous permettra de répondre à ces enjeux :

- Analyse des risques
- Couverture exhaustive
- Feedback au plus tôt
- Evaluation opportune
- Fiabilité
- Stabilité de l'effort de test
- Permet des économies
- Satisfaction Utilisateur

4.3- Les risques du Continuous Testing

Il est nécessaire d'anticiper certains pièges :

- Avoir une stratégie à court / moyen / long terme
- Exiger des entrants de qualité
- Répartir l'effort de test

- Valoriser les compétences
- S'appuyer sur des moyens externes
- Assurer la pertinence des tests par des actions de corrections

Le continuous testing pourra nécessiter de la formation, un accompagnement, des expertises, des renforts ciblés, des outils adaptés.
Une mise en œuvre progressive basée sur une amélioration continue facilitera votre succès.


DEPUIS QUE VOUS AVEZ TROUVÉ LE MOYEN DE SUPPRIMER AUTOMATIQUEMENT TOUS LES BUGS ROUGES, ON A UN PROBLÈME...
LES BUGS VERTS PROLIFÈRENT
LAVOMATIC
ENLÈVE TOUS LES BUGS MÊME LES PLUS RÉSISTANTS
CYCLE LONG
IA UN TRUC KICLOCHE
IA KARECOMENCÉ
F. COINTE

I.4- Bonnes pratiques TMMi pour une automatisation durable dans l'entreprise

par Eric Riou du Cosquer

Introduction

L'automatisation des tests offre des perspectives concrètes d'optimisation de la rentabilité et de l'efficacité des tests, en utilisant des langages, méthodes ou outils souvent séduisants et attractifs.
Grâce à la réelle professionnalisation des métiers et activités de test observée depuis plus de 10 ans en France et dans le monde, la question de l'automatisation est maintenant presque systématiquement abordée aux différents niveaux de test d'un projet, mais aussi au niveau des organisations de test transverses.
Pourtant, les bilans des initiatives d'automatisation sont souvent négatifs et les démarches d'automatisation interrompues. Ceci est parfois dû à l'instabilité des applications à tester, à la complexité technique des scripts et outils d'automatisation, au manque de ressources allouées à cette tâche ou encore à la multiplicité des possibilités qui rendent difficile le choix de « LA » bonne solution d'automatisation.
Mais la cause principale est sans doute l'absence de vision et de coordination des activités d'automatisation au niveau, non pas d'un projet, mais d'une entreprise.
Le modèle TMMi (www.tmmi.org) permet de travailler au niveau de l'entreprise et fournit des conseils pratiques utiles pour maximiser les chances de succès de l'automatisation.
Ce chapitre présente, en suivant les différents niveaux de TMMi, de bonnes pratiques à prendre en considération pour faire de l'automatisation une activité rentable, non pas sur un projet particulier ou sur une courte période de temps mais pour toute une organisation de test et de façon pérenne ; c'est d'**automatisation durable** qu'il s'agit !

1- Le modèle TMMi

1.1- Vue générale

Le modèle TMMi peut être présenté comme un référentiel de bonnes pratiques directement ou indirectement liées aux activités de test et réparties dans différents domaines, comme la Conception et l'Exécution des Tests ou encore les Tests Non-Fonctionnels. Les domaines sont eux-mêmes distribués sur 5 niveaux, sur lesquels s'appuie un dispositif d'accréditation, non pas des personnes mais d'une équipe, d'un centre de test ou de toute société ayant organisé des activités de test.

La figure suivante présente ces 5 niveaux et les domaines qui leur sont associés, 4 niveaux en réalité puisque le niveau 1 « initial » correspond à une situation sans activités de tests coordonnées et avec une faible qualité des logiciels produits.

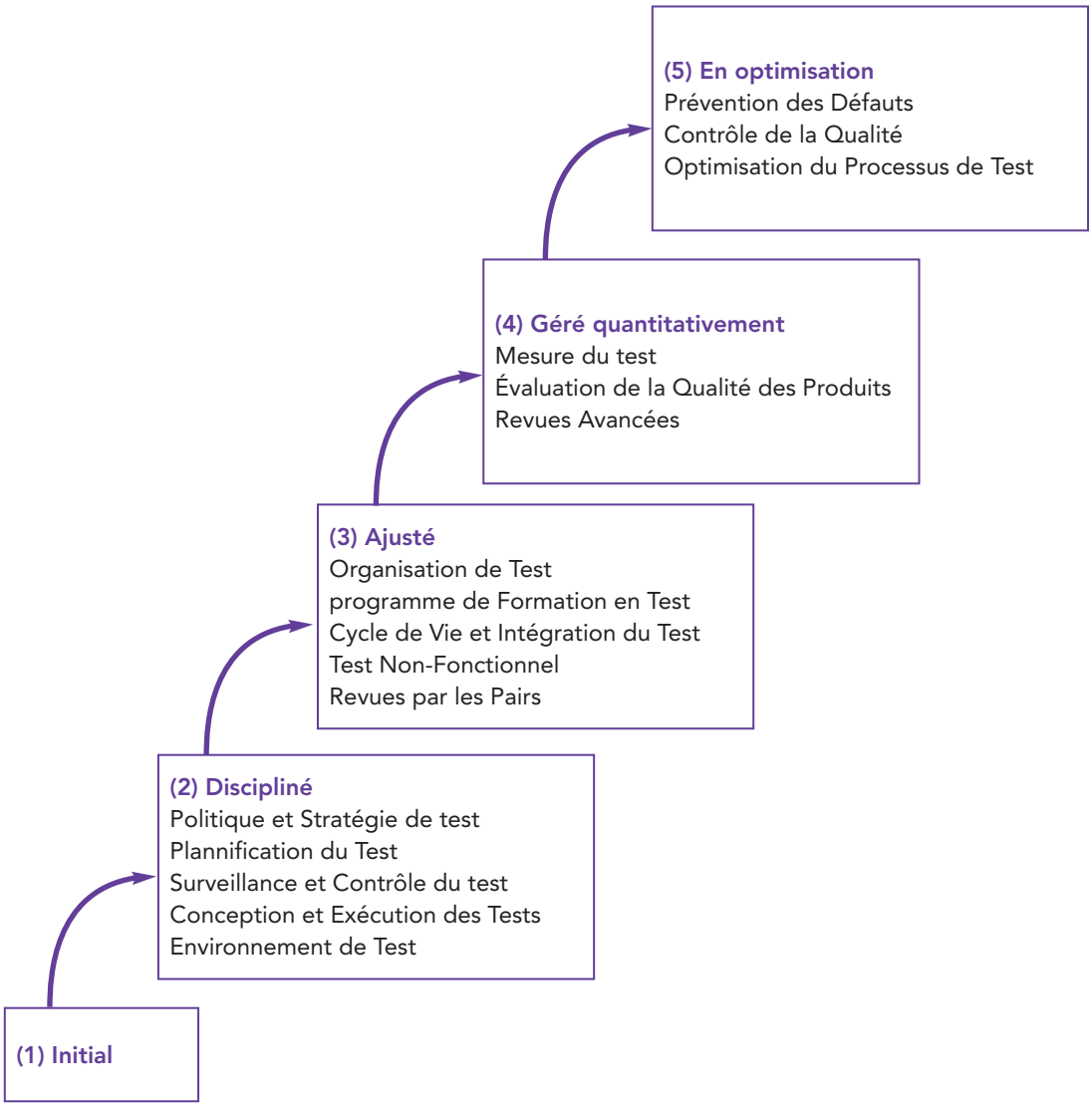


Figure 1: niveaux de maturité TMMi et domaines de processus.

Chaque domaine est explicité, d'une part avec des objectifs et pratiques spécifiques, d'autre part avec des objectifs et pratiques génériques. Ce qui est spécifique touche à des tâches directement liées au test, ce qui est générique touche à des tâches génériques liées à une bonne gestion d'une équipe ou organisation de test, comme la gestion de parcours de carrières ou encore la formation des testeurs. La figure suivante montre la relation entre niveau TMMi, Domaines de processus, Objectifs et Pratiques.

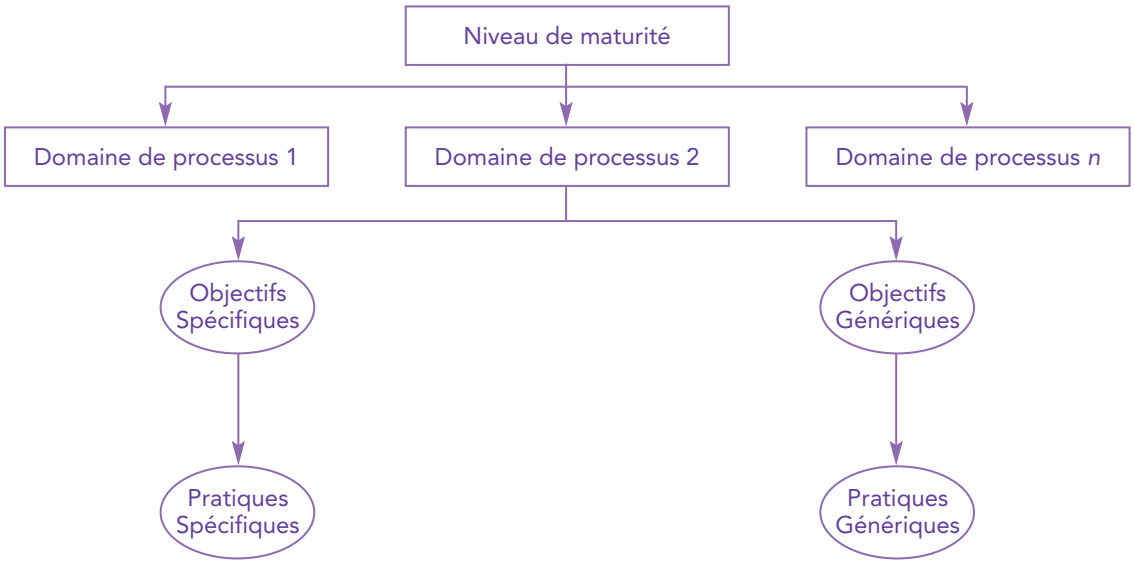


Figure 2 : Structure et composants TMMi

1.2- Utilisation de TMMi pour l'automatisation durable

L'automatisation des tests, si elle n'est pas intégrée à une vision et à une stratégie de tests globale, sera un échec. TMMi ne contient pas d'objectifs ou pratiques explicitement focalisés sur l'automatisation, mais de très nombreux objectifs, illustrés de bonne pratiques, ont un lien direct avec l'automatisation et le succès à long terme de sa mise en œuvre.
La suite de ce chapitre présente, pour chaque niveau et chaque domaine de TMMi, une bonne pratique relative à l'automatisation, en précisant à quels objectifs et pratiques elle se rattache. Pour chaque bonne pratique, l'objectif spécifique et la pratique spécifique TMMi qui en sont la source sont indiqués, afin de vous permettre de consulter le modèle TMMi aux bons endroits pour avoir du détail et des exemples.

2- Niveau 2 « Discipliné » : Conseils et bonnes pratiques

2.1- Politique et Stratégie de Test

2.1.1- Des objectifs « SMART » sont définis, à différents niveaux, pour l'automatisation

Objectif Spécifique TMMi : SG 1 Etablir une Politique de Test
Pratique Spécifique TMMi : SP 1.1 Définir des Objectifs de Test

Dès le niveau 2, TMMi introduit la notion d'objectifs de test. L'idée est de bien préciser, en amont de toute démarche d'automatisation, les bénéfices escomptés, bénéfices devant bien évidemment être SMART, c'est-à-dire Spécifiques, Mesurables, Atteignables, Pertinents et Bornés dans le temps.

Il est essentiel d'associer des objectifs de l'automatisation à différents niveaux, par exemple au niveau Entreprise ou Organisation, puis à chacun des 4 niveaux présentés par l'ISTQB Acceptation, Système, Intégration et Composant.

Le tableau suivant peut être utilisé pour présenter ces objectifs dans un document de type Stratégie de Test ou Plan de Test.

Niveau	Description de l'objectif	Intérêt	Valeur cible	Procédé de mesure	Porteur	Date lilite
Entreprise ou Oranisation						
Acceptation						
Système						
Intégration						
Composant						
...						

Tableau 1 : Définition des objectifs de l'automatisation des tests

2.1.2 Les risques les plus importants sont couverts par des tests automatisés

Objectif Spécifique TMMi : SG 2 Etablir une Stratégie de Test
Pratique Spécifique TMMi : SP 2.1 Procéder à une évaluation des risques produits génériques

Le test basé sur les risques constitue l'une des approches de test les plus efficaces, raison pour laquelle le modèle TMMi met l'accès dessus dès le niveau 2. Le principe est simple : comme il est impossible de tout tester, alors on met l'effort sur les caractéristiques fonctionnelles et non fonctionnelles qui présentent le plus haut niveau de risque. Bien sûr, le niveau de risque n'est pas estimé « au doigt mouillé » mais en calculant les valeurs d'impacts et de probabilité des risques, en valorisant des paramètres comme ceux proposés par la méthode Prisma (http://www.erikvanveenendaal.nl/practical-risk-based-testing-the-prisma-approach/).

Le test basé sur les risques, appliqué à l'automatisation, va consister à identifier parmi les risques les plus importants, ceux qui peuvent être couverts par des tests automatisables… qu'il est intéressant d'automatiser. En effet, il faut toujours avoir en tête la rentabilité de l'automatisation et exclure les tests pour lesquels l'effort d'automatisation sera supérieur au bénéfice de l'automatisation.

Pour illustrer la notion de risques génériques introduite par TMMi, il est intéressant de regarder le tableau de l'ISO 25010 présentant les caractéristiques qualité fonctionnelles et non fonctionnelles, ainsi que leur sous caractéristiques.
Certaines caractéristiques et sous-caractéristiques correspondent en effet à des risques génériques, c'est-à-dire des risques ayant un sens pour une majorité des projets. Par exemple, la caractéristique « Performance » est associée à un risque générique pour lequel il sera intéressant d'automatiser des tests de performance en charge, notamment au niveau Système. Autre exemple, la caractéristique « Maintenabilité » pourra donner lieu à de l'analyse statique de code, et donc à des tests statiques automatisés au niveau Composant. Troisième et dernier exemple avec la caractéristique « Utilisabilité » et sa sous-caractéristique « Accessibilité » qui, pour couvrir le risque d'impossibilité ou de difficulté d'usage d'une application pour des personnes ayant un handicap, pourra être couverte par des tests d'accessibilité automatisés à partir de l'Interface de l'application ou du site Web.

2.1.3- Des indicateurs sont mis en place pour évaluer la performance de l'automatisation

Objectif Spécifique TMMi : SG 3 Etablir des Indicateurs de Performance du Test
Pratique Spécifique TMMi : SP 3.1 Définir des Indicateurs de Performance du Test

De même que le coût des tests est important, voire très important sur un projet de développement logiciel, le coût de l'automatisation est important, voire très important dans le coût global des tests d'un projet, alors il ne faut pas que cela soit de l'argent gâché et il faut être à même de justifier ce coût d'automatisation en garantissant la rentabilité de l'automatisation. Facile à écrire mais difficile en réalité, d'autant plus difficile qu'il n'y a pas d'indicateurs « passe partout », applicable à tout projet d'une entreprise ou à toute équipe de test.

– Mesurer le pourcentage de tests automatisés ?
    - o Pourquoi pas mais à quel niveau et à partir de la combientième version de l'application ?
– Nombre de défauts, ou pourcentage de défauts découverts grâce à des tests automatisés ?
    - o Pourquoi pas mais attention au projet qui automatiserait systématiquement une majorité de ses tests en dépensant un budget démesuré, car ce type d'indicateurs pourrait ressortir « en vert » malgré une situation pas forcément rentable
– Temps moyen d'exécution d'un test ?
    - o Indicateur intéressant mais à corréler avec le temps de conception et avec le nombre d'exécutions des tests automatisés

A vous de définir les indicateurs adaptés à votre contexte pour mesurer la performance de l'automatisation, de les mettre en place, de les suivre et d'adapter vos activités de test en conséquence.

2.2- Planification du Test

2.2.1- La place des tests automatisés pour la non-régression est documentée et connue pour les niveaux de test concernés

Objectif Spécifique TMMi : SG 2 Etablir une Approche de Test
Pratique Spécifique TMMi : SP 2.2 Définir l'approche de Test
Sous-Pratique Spécifique TMMi : Définir l'approche des tests de régression, tests manuels ou utilisant des outils d'automatisation des tests.

TMMi, au niveau de ses sous-pratiques, précise l'importance de la définition de l' « Approche des tests de régression », c'est-à-dire de la stratégie dédiée à des tests qui seront exécutés plusieurs fois, probablement de nombreuses fois et pendant longtemps.
Là encore il est utile de se poser la question des tests de non-régression (ou « de régression », c'est la même chose, juste une histoire de traduction de l'anglais « regression tests » vers le français) pour les différents niveaux de test. De nombreux projets considèrent que les tests de non-régression automatisés ne concernent que le niveau Acceptation ou des tests faits après la première mise en production d'une application, ce qui n'est pas adapté à une majorité des projets. Des représentations comme la « Pyramide du Test » ou encore l' « Anti-pattern du cône de glace », largement documentées et débattues, sont d'excellents supports à la réflexion. Des approches de développement et de test, comme l'intégration continue et même le test continu, accordent également une bonne place aux tests automatisés, non seulement à différents niveaux mais aussi pour les aspects non-fonctionnels trop souvent omis.

Documenter et partager une approche d'automatisation spécifique à la non-régression et adaptée au contexte de votre projet et de votre entreprise aura une forte valeur ajoutée.

Le tableau suivant peut être utilisé pour synthétiser cette approche.

Niveau	Tests pouvant être automatisés	Critères d'automatisation	Outil ou méthode	Responsable de l'automatisation	Fréquence des exécutions	Responsable de la maintenance
Entreprise ou Oranisation						
Acceptation						
Système						
Intégration						
Composant						
...						

Tableau 2 : les tests de non-régression automatisés à différents niveaux

2.2.2- La charge et le coût des tests automatisés sont estimés

Objectif Spécifique TMMi : SG 3 Etablir des Estimations de Test
Pratique Spécifique TMMi : SP 3.3 Déterminer des estimations pour l'effort et le coût des tests

Les activités d'automatisation des tests font partie du projet. Il est donc nécessaire d'estimer leur charge, par exemple en hommes/jour et leur coût, par exemple en euros, et ceci pour chaque niveau de test où de l'automatisation est prévue, avec en plus des précisions par rapport aux types de test effectués à chaque niveau. Ce travail d'estimation est difficile, si difficile qu'on prévoit parfois de l'ignorer et de « voir ce que cela donne ». Ne pas estimer, c'est « naviguer dans le brouillard » comme le font de nombreux centres de test qui placent un stagiaire sur l'activité d'automatisation et ne poursuivent pas le travail commencé, soit parce que le stagiaire est parti avec son savoir sans qu'un transfert de connaissance ne soit organisé, soit parce qu'on ne sait pas combien cela a coûté ou combien cela coûterait de reprendre le travail. L'estimation de la charge et du coût des tests est donc un point clé, à mettre en œuvre au sein des équipes fonctionnant en mode séquentiel ou en mode itératif. Pour un projet suivant la méthode SCRUM, ces estimations peuvent se faire très simplement lors du planning poker, en abordant les sujets du test et, plus précisément, de l'automatisation des tests, pour chaque User Story estimée.

2.2.3- Les activités d'automatisation des tests sont planifiées et les ressources associées sont réservées

Objectif Spécifique TMMi : SG 4 Développer un Plan de Test
Pratique Spécifique TMMi : SP 4.1 Etablir le calendrier des tests

Sur la base de l'estimation « au plus juste » de la charge et du coût des tests automatisés, il est nécessaire de planifier les activités d'automatisation de test et de réserver les ressources associées.

Pour le niveau Composant, avec des tests automatisés portant principalement sur des lignes de code, il est fréquent de planifier en même temps les activités de développement pur et les activités de « développement de tests automatisés ». Cela ne signifie pas qu'un même profil développeur sera chargé de développer les tests automatisés, pour le code qu'il a lui-même développé, ce qui en général n'est pas idéal mais que, sur une même période, des profils développeurs travaillant ensemble seront amenés à développer du code et à développer des tests unitaires automatisés, pour du code écrit par certains de leurs collègues. Attention, il est important d'éviter les situations qui ne laissent pas le temps de développer ces tests unitaires automatisés : « Je suis déjà en retard avec mes développements, je ne vois pas comment je pourrais, en plus, tester le code des autres ».

Pour les autres niveaux, par exemple pour le niveau Acceptation, il est utile de planifier différentes activités, avec pour chacune des ressources pouvant être différentes, par exemple des profils métier pour la conception des tests et des profils développeurs pour l'implémentation et l'exécution des tests.

2.3- Surveillance et Contrôle du Test

2.3.1- La charge et le coût réel des activités d'automatisation sont mesurés et suivis par rapport aux estimations

Objectif Spécifique TMMi : SG 1 Surveiller l'Avancement du Test par rapport au Plan
Pratique Spécifique TMMi : SP 1.1 Surveiller les paramètres de planification du test

TMMi repose sur un principe général d'amélioration continue pour chaque objectif et chaque ensemble de pratiques. L'amélioration est particulièrement importante pour ce qui touche aux estimations de charge et de coût qui sont particulièrement difficiles.
L'objectif est de régulièrement confronter la réalité des charges et coûts aux estimations préalablement réalisées. Cela fait penser à la notion de vélocité utilisée en Scrum et qui détermine l'effort qu'une équipe de développement peut fournir pour réaliser les tâches planifiées dans un Sprint. Celle-ci ne se mesure pas précisément pour le premier Sprint mais, en général après 5 ou 6 Sprints, afin de laisser à l'équipe le temps de se stabiliser.
Il en est de même avec l'estimation de la charge et du coût de l'automatisation, après quelques itérations ou projet, cette estimation sera de plus en plus précise, à condition de régulièrement la vérifier.

Le tableau suivant peut être utilisé pour progresser en estimation.

Niveau	Tests automatisés	Charge estimée (h/j)	Charge réelle (h/j)	Coût estimé (€)	Coût réel (€)	Cause(s) des différences entre l'estimé et le réel
Entreprise ou Oranisation						
Acceptation						
Système						
Intégration						
Composant						
...						

Tableau 3 :amélioration des estimations

2.3.2- Les rapports de défauts ne sont créés qu'après une analyse humaine des résultats d'exécution des tests automatisés

Objectif Spécifique TMMi : SG 2 Surveiller la Qualité du Produit par rapport au Plan et aux Attentes
Pratique Spécifique TMMi : SP 2.2 Surveiller les Défauts

Il peut être tentant d'automatiser la création des rapports de défauts à partir des résultats d'exécution des tests automatisés mais cela présente un risque important de faux-positifs, c'est-à-dire de résultats d'exécution de test en échec, alors que ces tests auraient dû être exécutés avec succès si l'environnement ou les données utilisées avaient été les bons. Par exemple, il est possible qu'un environnement d'exécution de test indisponible, ou que des données de test incorrectes, soient la cause de l'échec de tests automatisés. Une vérification humaine de la validité des résultats d'exécution de tests automatisés sera très souvent utile.

2.4- Conception et Exécution des Tests

2.4.1- Les résultats d'exécution sont enregistrés avec les informations permettant leur reproduction

Objectif Spécifique TMMi : SG 3 Exécuter les tests
Pratique Spécifique TMMi : SP 3.4 Ecrire un registre de test

Les résultats d'exécution des tests automatisés doivent être enregistrés avec toutes les informations permettant leur reproduction et notamment la version du test exécuté, la version du code testé et toutes les informations relatives à la configuration de l'environnement d'exécution des tests, notamment ce qui touche au matériel (serveurs, réseaux..), au logiciel (logiciels installés) et aux données utilisées. Pourquoi ?

Voici quelques raisons.

Tout d'abord, ces éléments permettront d'apporter des preuves suffisantes sur l'exécution des tests automatisés, preuves qui peuvent être requises dans certains contextes règlementés et sur des projets sensibles, avec des auditeurs qui ne se contenteront pas de résultats d'exécution.

Par ailleurs, ces éléments permettront aux personnes -en général des profils développeurs- chargées d'analyser les rapports de défauts puis de les corriger quand cela est nécessaire, de reproduire précisément le scénario d'exécution du test en échec pour observer la défaillance relative au défaut.

Enfin, ces éléments permettront de suivre différentes métriques utiles, comme le suivi de l'évolution du pourcentage de tests automatisés exécutés avec succès, de version en version, le but étant bien sûr d'observer une baisse qui sera le signe d'une qualité améliorée, mais peut-être aussi le signe de la nécessité de modifier certains tests ou d'en ajouter de nouveaux, comme le suggère le principe ISTQB du « Paradoxe du pesticide ».

2.5- Environnement de Test

2.5.1- Les environnements d'exécution des tests automatisés ont été spécifiés et testés avant utilisation

Objectif Spécifique TMMi : SG 1 Développer les Exigences d'Environnement de Test
Pratique Spécifique TMMi : SP 1.1 Eliciter les besoins d'environnement de test

TMMi accorde une part importante aux environnements de test avec des objectifs simples et efficaces : identifier les besoins en environnements, faire développer les environnements nécessaires, « tester » les environnements mis à disposition avant de les utiliser, coordonner leur utilisation entre différents projets et enfin, gérer les incidents liés aux environnements. Ces objectifs sont tout à fait valables pour l'automatisation qui va nécessiter des environnements particuliers pour l'exécution des tests automatisés. La présence de bouchons et de pilotes, de même que les données nécessaires à l'exécution des tests automatisés, sont des aspects essentiels qu'il est nécessaire de spécifier et de vérifier régulièrement. Cela permettra non seulement de garantir la validité des résultats d'exécution de tests automatisés mais aussi de réexécuter certains tests dans un environnement particulier.

Conclusion

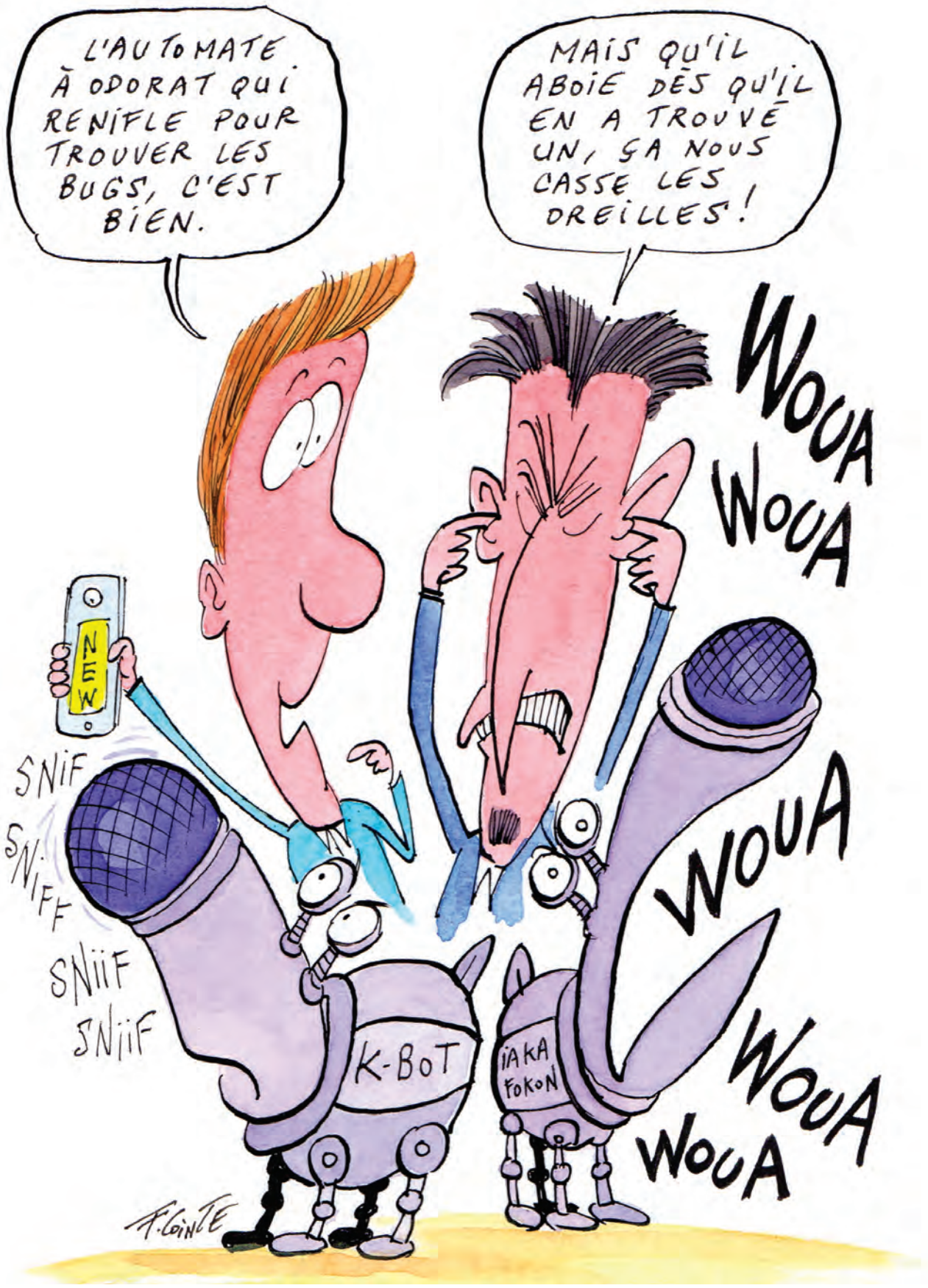
A partir des domaines de processus du niveau 2 de TMMi et de quelques-uns des objectifs spécifiques qui y figurent, de bonnes pratiques spécifiques à l'automatisation des tests ont été présentées.
Le tableau suivant, qui les résume, peut être utilisé comme une checklist dans le cadre d'une démarche d'automatisation.

Bonne pratique	Appliquée ? (Oui/Non)	Justification
2.1.1- Des objectifs «SMART» sont définis, à différents niveaux, pour l'automatisation		
2.1.2- Les risques les plus importants sont couverts par des tests automatisés		
2.1.3- Des indicateurs sont mis en place pour évaluer la performence de l'automatisation		
2.2.1- La place des tests automatisés pour la non-régression est documentée et connue pour les niveaux de tests concernés		
2.2.2- La charge et le coût des tests automatisés sont estimés		
2.2.3- Les activités d'automatisation des tests sont plannifiées et les ressources associées sont réservées		
2.3.1- La charge et le coût réel des activités d'automatisation sont mesurés et suivis par rapport aux estimations		
2.3.2- Les rapports de défauts ne sont créés qu'après une analyse humaine des résultats d'exécution des tests automatisés		
2.4.1- Les résultats d'exécution sont enregistrés avec les informations permettant leur reproduction		
2.5.1- Les environnements d'exécution des tests automatisés ont été spécifiés et testés avant utilisation		

Tableau 4 : synthèse des bonnes pratiques

Bien sûr, cette liste est loin d'être exhaustive et une lecture du modèle TMMi sur l'ensemble des niveaux 2, 3, 4 et 5, avec un œil « automatisation », permet de faire ressortir d'autres bonnes pratiques, comme la notion de Parcours de carrières qui peut être appliquée à l'automatisation et ses différents métiers du test.

Le modèle TMMi et la certification associée « Professionnel Certifiés TMMi » constituent sans aucun doute un support d'amélioration de l'automatisation des tests au niveau d'un projet, mais aussi au niveau d'un centre de test ou d'une entreprise.

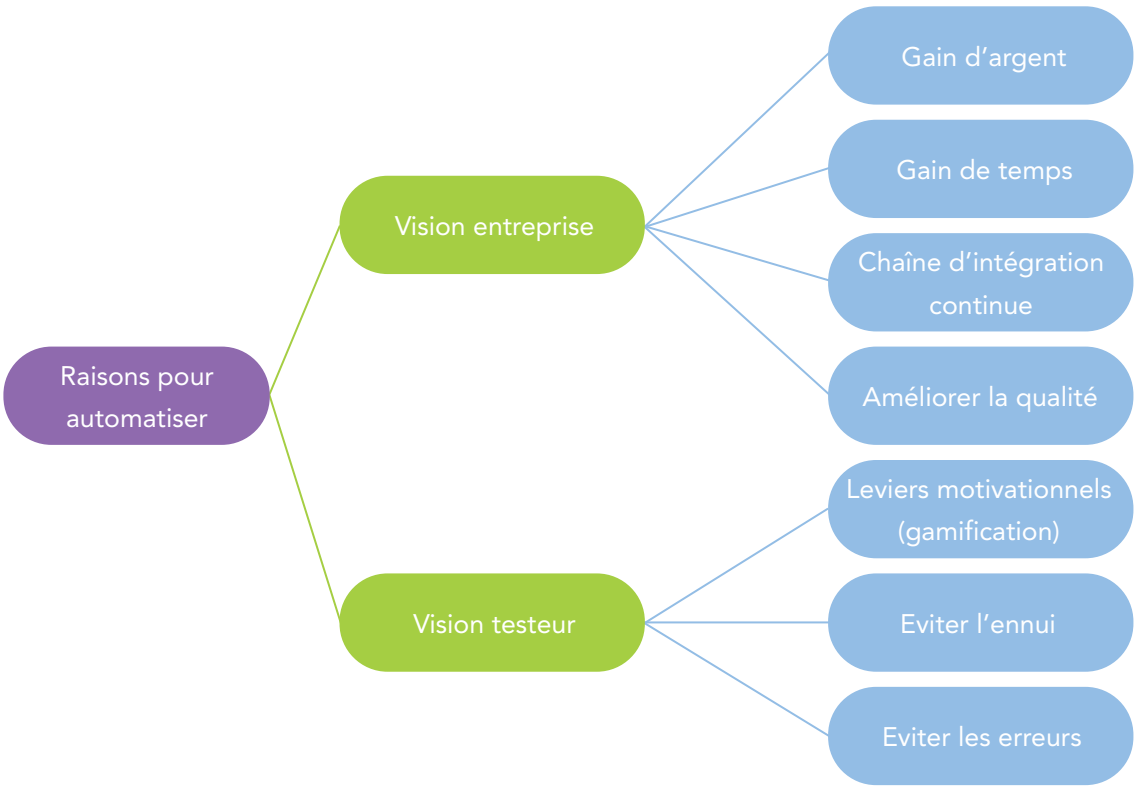

L'AUTOMATE À ODORAT QUI RENIFLE POUR TROUVER LES BUGS, C'EST BIEN.
MAIS QU'IL ABOIE DÈS QU'IL EN A TROUVÉ UN, ÇA NOUS CASSE LES OREILLES !
NEW
SNIF
SNIFF
SNIIF
SNIIF
K-BOT
IAKA FOKON
WOUA
WOUA
WOUA
WOUA
WOUA
F.COINTE

I.5- Les dérives de l'automatisation

par Marc Hage Chahine

1- Introduction

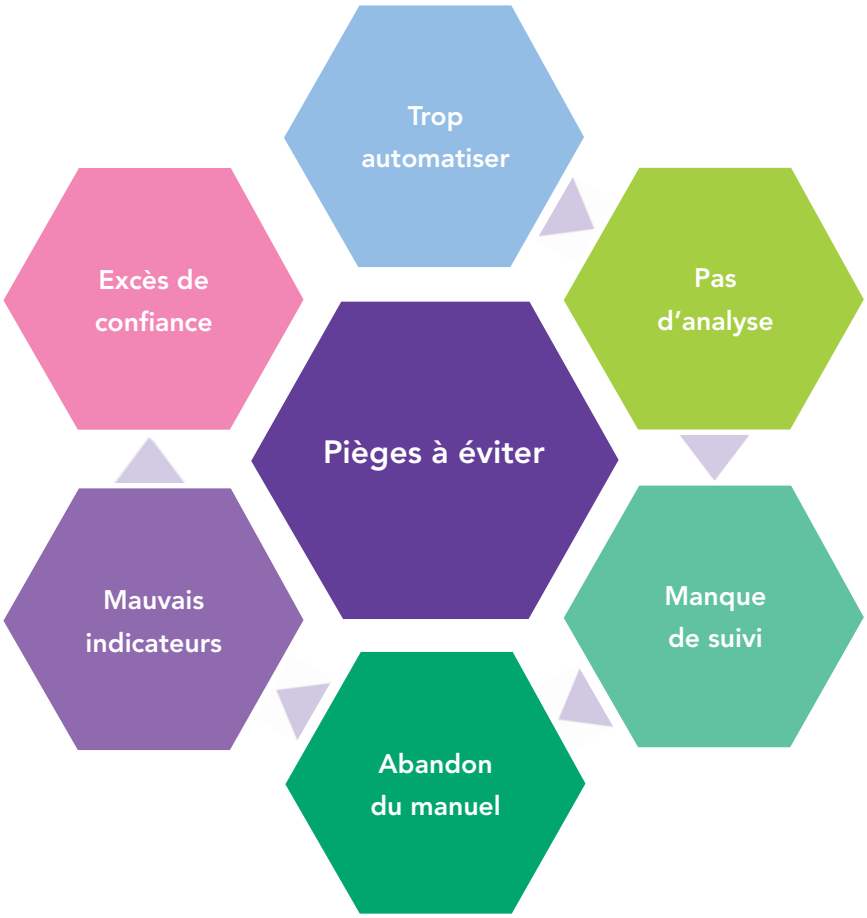
Nous l'avons vu dans un chapitre précédent, il y a plusieurs raisons pour lesquelles on peut vouloir automatiser.



Il est d'ailleurs primordial de toujours garder à l'esprit les raisons pour lesquelles on souhaite automatiser, sous peine de dérives menant cette automatisation à l'encontre de son objectif initial. Cela est d'autant plus important qu'il est théoriquement possible d'automatiser au moins en partie toutes les activités de test !

2- Familles de dérives liées à l'automatisation

Le but de ce chapitre est de vous présenter 6 familles courantes de dérives de l'automatisation afin de vous permettre de les éviter plus facilement. Ces familles sont :



2.1- Automatisation de trop de tests

Cette famille de dérives est très fréquente. Elle peut être causée par divers facteurs. Ceux que j'ai rencontrés le plus souvent sont l'excès de confiance suite à des succès initiaux et un changement radical de contexte avec une diminution drastique des capacités de l'équipe en charge de l'activité d'automatisation.
Concrètement vouloir trop automatiser peut revenir à :

2.1.1- Automatiser l'exécution de trop de tests.

Cette erreur est une erreur très classique… et nous ramène au cœur même du métier de testeur : faire des choix.
Les tests exhaustifs, même automatisés, sont impossibles. Automatiser, exécuter, analyser et maintenir des tests demande un investissement (en temps et en argent). Il ne faut pas laisser croire que l'automatisation de l'exécution des tests rend les tests « gratuits ».
Multiplier les tests, c'est aussi multiplier les efforts et les coûts ! Hors, l'équipe en charge de l'automatisation a un temps et un budget limités. Les problématiques du test manuel restent avec l'automatisation. L'automatisation de l'exécution des tests scriptés permet simplement, si elle est bien faite, de gagner du temps et de l'argent sur l'exécution des campagnes.

Au final, automatiser trop de test, c'est un dire un nombre trop important pour l'équipe, revient à aller contre les objectifs de l'automatisation :

- Une perte de temps : de par un investissement trop important dans l'écriture et la maintenance des tests
- Une perte d'argent : pour les mêmes raisons que la perte de temps
- Une baisse de la qualité : avec un manque de temps accordé aux autres activités de test ou une maintenance négligée
- Le développement de l'ennui : de certains testeurs étant amenés à ne travailler que sur certaines tâches d'automatisation
- Une augmentation des erreurs : avec la multiplication des « flaky tests ».

2.1.2- Automatiser des tests trop complexes

Automatiser trop de tests n'est pas uniquement une question de nombres. C'est aussi lié à la capacité de l'équipe à automatiser certains tests ou certaines activités. En effet, l'intérêt d'automatiser un test n'est pas le même pour tous les tests et va dépendre en grande partie de la capacité technique à l'automatiser. Cette capacité technique peut être liée à l'outil de test, aux compétences de l'équipe ou tout simplement au test lui-même demandant des actions ou un contexte bien particulier plus simple à faire à la main. Tout comme un nombre trop grand de tests, automatiser des tests trop complexes nous mène à des situations où les objectifs d'automatisation se sont plus atteints. En effet, automatiser des tests trop complexes revient à :

- Une perte de temps : sur la tâche d'automatisation du test lui-même, sur sa maintenance, sur le temps d'analyse du test
- Une perte d'argent : liée à la perte de temps
- Un développement de l'ennui du testeur : étant amené à toujours travailler sur les mêmes tests
- Une baisse de la qualité et une augmentation des erreurs: avec la multiplication des « flaky tests », tests dont les résultats sont peu fiables.

2.1.3- Automatiser des activités peu pertinentes

Cet aspect de l'automatisation peut être lié à la connaissance d'un outil qui offre des capacités d'automatisation de l'activité en question, à l'envie d'expérimenter ou plus simplement à des objectifs hiérarchiques peu pertinents par rapport au contexte de l'équipe.

Pour rappel, l'automatisation doit répondre à des objectifs précis comme le gain d'argent, de temps ou encore l'amélioration de la qualité de travail des testeurs (éviter l'ennui). Le choix des activités d'automatisation se doit d'être fait en fonction de ces critères, des problématiques remontées ou anticipées par l'équipe. L'Agile propose un formidable outil pour repérer les points de douleurs, c'est les rétrospectives. Les rétrospectives permettent de mettre en place facilement l'amélioration continue en relevant des problèmes et en réfléchissant à des solutions pour les résoudre. L'automatisation n'est d'ailleurs qu'une solution parmi d'autres.

Au final, automatiser une activité qui n'a pas vocation à résoudre une problématique concrète de l'équipe, cela revient à :

- Une perte de temps : avec l'investissement de l'équipe sur un sujet moins pertinent que d'autres
- Une perte d'argent : de par le manque de travail sur des problématiques vraiment gênantes pour l'équipe
- Une perte de sens liée à l'automatisation

Afin d'éviter les problématiques liées à « trop d'automatisation », il faut être capable de suivre et d'analyser les résultats de l'automatisation.

2.2- Manque de suivi et d'analyse des résultats de l'automatisation

Automatiser des activités c'est bien ! Savoir ce qu'apporte l'automatisation de ces activités, c'est mieux. Nous n'automatisons pas juste pour le plaisir d'automatiser, nous automatisons pour répondre à des objectifs. Pour savoir si ces objectifs sont atteints ou si nous sommes dans la bonne direction pour atteindre ces objectifs il est essentiel d'assurer un suivi de notre « projet d'automatisation », car oui, l'automatisation d'une activité est un projet à part entière.
Comme tout projet, l'automatisation d'une activité nécessite un suivi : quel est l'avancement ? Sommes-nous dans les temps ? Respectons-nous les coûts ? Quels sont les résultats ?... Bref, un bilan est nécessaire. Il est d'ailleurs intéressant de noter que ce bilan et suivi des actions choisies est particulièrement mis en avant en Scrum avec les Rétrospectives mais aussi le rôle de Scrum Master qui est là pour assurer un suivi.
Concrètement, pour l'activité d'automatisation de l'exécution, une dérive trop fréquente est de multiplier les exécutions et de ne pas analyser les résultats des tests exécutés. Pour d'autres activités comme l'automatisation des bilans, cela peut revenir à éditer automatiquement des bilans sans vérifier que les résultats remontés sont exacts. On peut également imaginer l'intégration d'outils pour générer les environnements et les données de test sans vérifier que ce qui est généré correspond à notre besoin.
Bref, ne pas suivre ni analyser les résultats liés à l'automatisation revient à être dans l'incapacité de savoir si nos actions sont pertinentes, si l'automatisation répond à ses objectifs ou si elle sera en mesure d'y répondre un jour.

2.3- Le manque de maintenance ou d'évolution

Ce point est particulièrement visible sur la tâche d'automatisation de l'exécution des tests scriptés. Il est d'ailleurs important de noter qu'il est souvent lié au fait que l'équipe a « trop de tests » à gérer.
L'automatisation de l'exécution des tests scriptés permet de réduire grandement (sans pour autant le rendre gratuit) le coût d'exécution des tests. Cependant, il y a un coût « caché » qui est beaucoup plus élevé avec des tests automatisés qu'avec des tests manuels. Ce coût, c'est

le coût de maintenance. Même s'il existe des bonnes pratiques comme la factorisation des étapes de test, le coût de maintenance peut vite devenir très important… voire dépasser les capacités de l'équipe.
De même, par maintenance des tests, il faut aussi penser à la maintenance de la campagne en faisant évoluer cette dernière. On ajouter des tests, on en retire, on en modifie certains. Cet aspect trop souvent négligé répond pourtant à un des 7 principes du test : le paradoxe des pesticides. Pour éviter l'impact de ce dernier, nous nous devons de modifier cette campagne.
Le manque de maintenance ou d'évolution est particulièrement impactant par rapport à l'objectif d'amélioration de la qualité :

- Le manque de maintenance des tests automatisés est la voie ouverte à des tests qui ne sont plus à jour et qui ne détectent plus rien.
- Le manque d'évolution de la campagne est la voie ouverte au paradoxe des pesticides et à une multiplication des tests non détectés.

2.4- L'abandon des activités manuelles

Les activités d'automatisation ne sont qu'un outil dans la main du testeur. Cet outil apporte de l'aide au testeur mais ne dispense pas des tâches manuelles. En fait, les activités automatisées sont complémentaires des activités manuelles. J'oserai presque aller plus loin en disant que l'automatisation de certaines activités permet de faire plus de tâches manuelles à forte valeur ajoutée. La libération du temps apportée par l'automatisation peut en effet permettre de passer plus de temps sur l'amélioration continue, le test exploratoire et la conception des tests.
L'abandon des tâches manuelles revient à l'abandon de l'humain, du ressenti mais aussi à l'abandon de la capacité des testeurs à aller plus loin que ce qui est demandé et à se rapprocher des utilisateurs. D'expérience, même en déploiement continu, il reste des tâches manuelles à exécuter. Dans mon cas c'était la multiplication des tests exploratoires et le suivi de la production.
Au final, l'abandon des activités manuelles va à l'encontre de plusieurs objectifs de l'automatisation avec:

- Une baisse significative de la qualité : avec une perte de l'humain, de son point de vue, de son ressenti
- Une perte d'argent : en automatisant trop
- Un sentiment d'ennui : avec un rôle de « presse-bouton »
- Une perte de temps à moyen terme : à court terme on gagne beaucoup de temps mais la réalité nous rattrape ensuite avec de nombreuses anomalies non détectées à corriger ou des aspects du logiciel à reprendre.

2.5- L'adoption de mauvais indicateurs

C'est une problématique récurrente dans le test mais plus généralement dans toutes les activités en entreprise : il faut être capable de suivre et d'analyser l'impact de nos actions. Pour cela, nous bénéficions d'outils très polyvalents que sont les indicateurs.

Les indicateurs permettent de mesurer des points spécifiques dans un contexte défini. Néanmoins, les indicateurs restent uniquement des outils de mesures. Seuls, ils ne veulent rien dire, mal choisis, ils peuvent mesurer des points non pertinents.
Prenons l'exemple d'un thermomètre. Cet outil propose un indicateur qui est la température et est souvent utilisé pour savoir si quelqu'un est malade. Néanmoins, une température de 38,5° ne signifie pas forcément dire que l'on est malade (on peut par exemple sortir d'un sauna) et une température de 37,5° ne veut pas forcément dire que nous sommes en bonne santé (ex : une température de 37,5° avec un cancer).
Les indicateurs doivent dont être choisis en fonction de nos objectifs et d'un contexte particulier. De même, il est toujours préférable d'avoir un ensemble d'indicateurs que l'on analyse et d'éviter autant que possible des objectifs précis sur des indicateurs afin d'éviter de tomber dans les travers de la loi de Goodhart.
Pour l'automatisation, je suis tombé sur un indicateur en particulier qui s'est avéré contre-productif: il était lié à un nombre de tests automatisés. Au final nous nous sommes retrouvés à automatiser des tests simples à automatiser mais peu pertinents. De même, nous automatisions tous les tests d'une fonctionnalité avant de passer à une autre. Notre couverture de tests automatisés était alors catastrophique.
Comme on peut le voir avec l'exemple précédent le choix d'indicateurs non pertinents dans le contexte peut impacter négativement plusieurs objectifs de l'automatisation comme :

- Une perte de temps et d'argent : en ne concentrant pas ses efforts là où c'est le plus pertinent
- Une perte de qualité : avec la concentration des efforts au mauvais endroit et une mauvaise priorisation
- Le développement de l'ennui : avec une perte de sens pour le testeur.

2.6- L'excès de confiance

L'excès de confiance est, de manière générale, un piège à éviter pour le testeur qui doit constamment faire preuve d'un certain scepticisme professionnel. Il faut entendre par là qu'il faut continuellement tester, explorer, tenter d'aller plus loin sans faire une confiance aveugle à ce qui a été mis en place. Ce scepticisme est une arme du testeur, afin de lutter contre les effets négatifs de certains principes comme le paradoxe des pesticides et l'illusion d'absence d'erreur… mais aussi un outil essentiel lors de l'automatisation de toute activité du test. En effet, l'automatisation des activités fait perdre, au moins en partie, le contrôle humain. Il est alors fréquent de (trop) se reposer sur les résultats de cette automatisation. Le testeur ne doit pas faire une confiance aveugle aux personnes avec qui il travaille (on teste le code d'un développeur, on fait une revue des spécifications…) il doit encore moins accorder cette confiance à une machine qui donne des résultats. Il est d'ailleurs important de noter que cet excès de confiance se voit beaucoup avec des algorithmes d'intelligence artificielle (car on ne sait pas encore assez bien les tester ?), ce qui engendre des cas d'école d'échec des intelligences artificielles, comme dans le système juridique américain.

Au final, l'excès de confiance est plus une cause des points précédents qu'une cause directe des tests en tant que telle. En effet cet excès de confiance peut mener à :

- Vouloir trop automatiser : en pensant que l'automatisation peut tout
- Le manque de suite des résultats : en laissant faire la machine
- Le manque de maintenance ou d'évolution : en pensant que ce qui est fait est parfait
- L'abandon des activités manuelles : en partant du principe qu'au final, la machine fait tout au moins aussi bien que l'homme
- La sélection de mauvais indicateurs : en ne prenant pas le temps de bien les sélectionner.

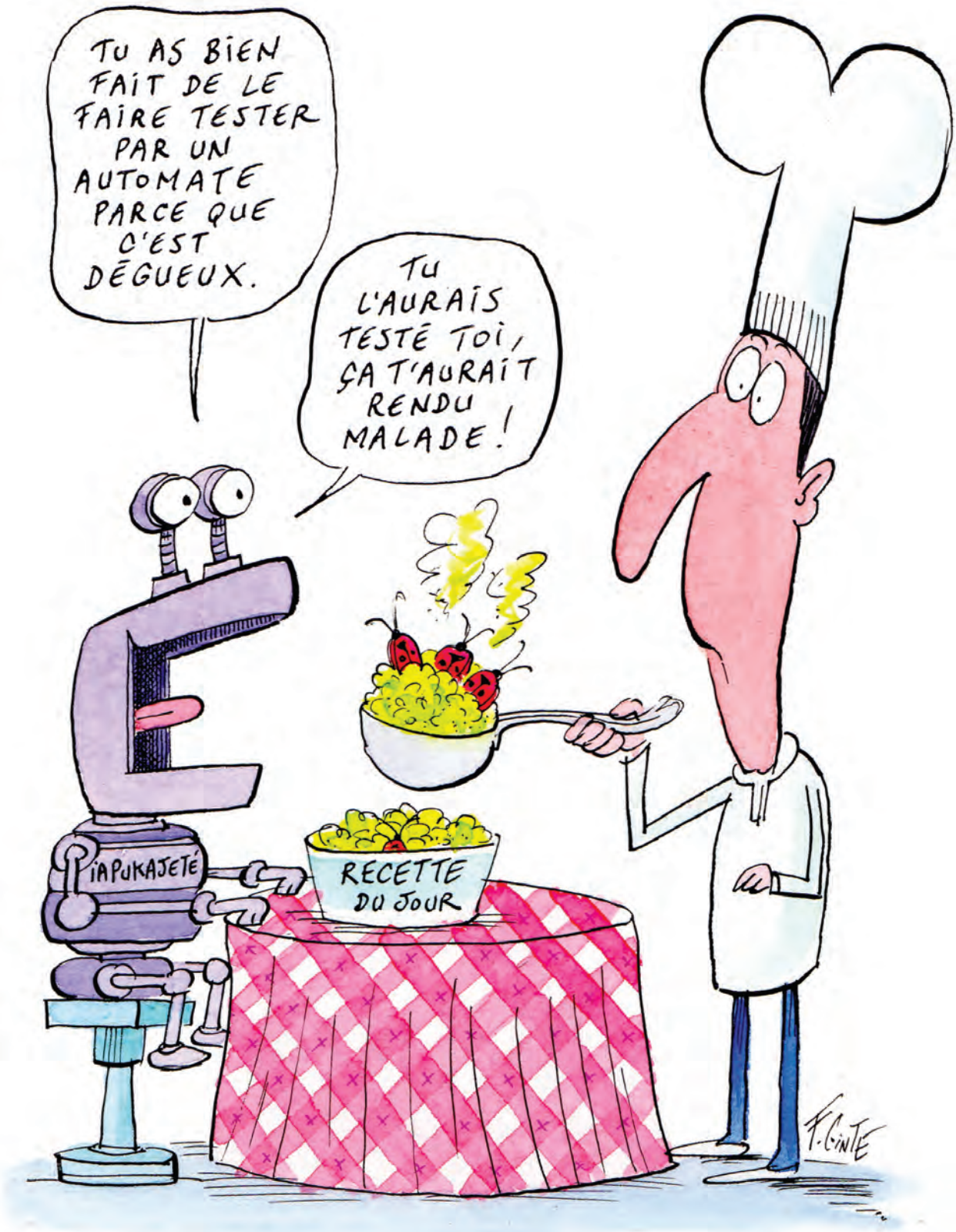
3- Conclusion

L'automatisation est un outil très puissant que l'on utilise instinctivement depuis l'enfance. Dans le cadre du test, c'est également un outil formidable ! Néanmoins cette automatisation (de toute activité de test) reste un outil. Et, comme tout outil, elle est efficace uniquement dans certaines situations et si l'on sait comment s'en servir. Afin de bien utiliser l'automatisation des activités de test, nous nous devons donc, de bien déterminer dans quels contextes cette dernière est intéressante à utiliser mais nous devons aussi veiller à bien nous en servir, notamment en évitant certaines dérives communes liées à son utilisation.

Vouloir automatiser les activités de test pertinentes tout en connaissant les pièges les plus fréquents à éviter permet de partir sur de bonne base dans cette activité d'automatisation. Cela ne reste qu'une base, pour le reste, seule l'expérience et l'expérimentation permettront de continuer à s'améliorer et de maîtriser l'automatisation dans un contexte précis.

PARTIE II

PRATIQUES DE L'AUTOMATISATION


TU AS BIEN FAIT DE LE FAIRE TESTER PAR UN AUTOMATE PARCE QUE C'EST DÉGUEUX.
TU L'AURAIS TESTÉ TOI, ÇA T'AURAIT RENDU MALADE !
PIAPUKAJETÉ
RECETTE DU JOUR
F.CINTE

II.1- Pour une stratégie d'automatisation des tests

par Jean-François Fresi

1- Introduction

Dans une société en perpétuelle mutation, les entreprises doivent être en mesure d'améliorer l'efficacité et la qualité de leurs solutions, dans le but de les rendre toujours plus performantes et efficaces. Le cycle de livraison des logiciels doit être fluide, continu et rapide et ainsi répondre aux attentes et exigences du marché.

La mise en place de tests automatisés pour assurer la meilleure qualité et détecter le maximum d'anomalies dans un temps minimum, apparaît comme une évidence. Pour cela, il est indispensable d'organiser les activités des tests afin de créer la valeur ajoutée nécessaire par de l'automation de tests fonctionnels.

Avant d'automatiser, il faut donc, d'une part, choisir les solutions techniques qui vont permettre de parvenir aux objectifs de cette activité et, d'autre part, mettre en place une stratégie de test qui permettra d'inclure des tests automatisés efficients.

2- Pourquoi automatiser ?

2.1- L'intérêt de l'automatisation

L'implémentation de tests automatisés permet d'améliorer la qualité logicielle, d'augmenter la couverture de tests tout en testant régulièrement les fonctionnalités vitales, voire de façon continue.

L'avantage premier de l'automatisation des tests se situe aussi au niveau de la **productivité**. Cette approche permet de tester plus fréquemment et de remonter le plus tôt possible les anomalies de fonctionnement. Cette vitesse de détection des anomalies permet aux équipes de développement une forte **réduction du délai et du temps nécessaires pour leur correction**.

Cela implique donc une augmentation du nombre de tests exécutés ainsi que de leur fréquence d'exécution, ce qui va contribuer à **l'amélioration de la qualité et de la fiabilité du produit**.

Une autre plus-value de l'automatisation des tests est de pouvoir contribuer à réduire le time to market, c'est-à-dire de livrer plus fréquemment, en mettant en production de plus en plus de fonctionnalités et en tendant vers un taux de dysfonctionnements quasi nul en production. Ce qui, bien entendu, contribue à la **satisfaction client**. En découvrant une anomalie au plus tôt, le risque encouru lors d'une mise en production est limité.

Libérés de cette tâche récurrente, les testeurs fonctionnels peuvent concentrer leurs efforts sur des activités à plus grande valeur ajoutée (tests exploratoires, BDD/ATDD, …).

L'intérêt de l'automatisation des tests fonctionnels se situe donc sur plusieurs niveaux :

- Fiabilité de l'application et satisfaction client
- Augmentation de la productivité
- Réduction du time to market
- Disponibilité des testeurs pour des activités plus forte valeur ajoutée

2.2- Les contraintes liées à l'automatisation

Afin de maîtriser la fiabilité des tests automatisés, l'exécution des tests doit se faire dans un cadre stable et maîtrisé. L'efficacité des tests est étroitement liée à la **stabilité de l'environnement** sur lequel ils sont exécutés.

Un autre point d'attention porte sur la **fiabilité et la pertinence des données de test**. La fiabilité de ces données est nécessaire pour garantir la conformité des résultats de tests mais aussi leur gestion dans le temps.

Avant de se lancer dans le chantier de l'automatisation, il est indispensable d'**évaluer la faisabilité technique** du périmètre fonctionnel à automatiser.

2.3- Les coûts cachés

Au-delà de ces contraintes, il faut aussi appréhender certains coûts cachés qui sont principalement liés à la **complexité des cas de tests, la volumétrie des cas de tests et leur maintenabilité** associées ainsi que la fréquence d'exécution des cas de tests.

La pertinence dans le choix des cas de tests à automatiser, liée à la **durée d'exécution des tests**, a aussi un impact indirect.

Un autre coût caché se situe dans le **choix de l'automate, en lien avec le profil des testeurs** qui vont développer les tests automatisés. Si l'on envisage des profils fonctionnels, le choix d'un outil trop technique sera un frein à la montée en compétence et à une automatisation plus large.

Mais alors, y a-t-il un réel intérêt à automatiser des tests fonctionnels ?

Faut-il automatiser un maximum de cas de tests avec un coût non négligeable pour la maintenance des tests automatisés fonctionnels ?

La réponse est doublement oui !

Mais pour toutes les raisons évoquées précédemment, il faut **concevoir une stratégie de tests automatisés adaptée** et qui prend en compte ces risques et ces contraintes.

3- Quelle stratégie pour les tests automatisés ?

3.1- La démarche

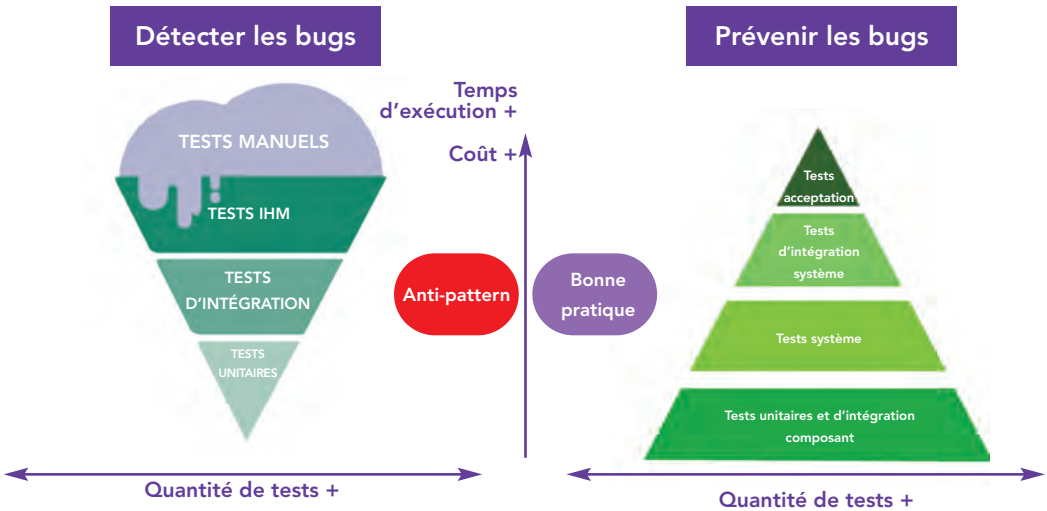
Il est important de mettre en place une stratégie de test qui soit en phase avec les priorités du projet et qui permet de créer de la valeur à chaque itération du cycle de vie de l'application.

La stratégie de test d'automatisation des tests fonctionnels doit donc **faire partie intégrante de la stratégie de test** et doit permettre de tester en profondeur les exigences et les parcours critiques d'une application.

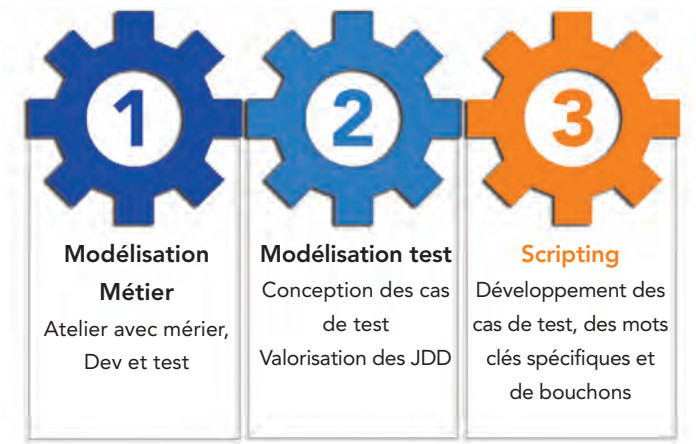
L'automatisation des tests ne doit pas être considérée comme un projet parallèle mais bien faire partie prenante du projet et donc de la stratégie de test globale.
Elle doit être partagée et validée par tous les acteurs du projet (métier, développeurs et testeurs).

Une approche conseillée est de partir de la vision métier et utilisateur afin de mieux appréhender les parcours critiques de l'application et choisir les priorités dans l'effort de test, que ce soit de façon manuelle ou automatisée.
Cette démarche permet d'avoir de l'ambition tout en priorisant les activités de tests automatisés et en créant rapidement de la valeur et de la confiance. Cela permet une montée en compétences progressive, tout en élevant au fur et à mesure ses objectifs d'automatisation.

Il est aussi indispensable de mettre en corrélation cette stratégie avec les bonnes pratiques de l'ISTQB et de la répartition des tests dans la pyramide de tests, en opposition à la vision cornet de glace du tout test manuel.



Pour cela, **la stratégie de test doit être partagée et co-construite avec les parties prenantes du projet** afin d'être la plus efficace et pertinente possible. Les priorités et les risques sont par conséquent partagés et connus par l'ensemble des acteurs projets.
L'approche pour l'automatisation des tests peut donc se schématiser de cette façon :



Pour ce faire, il est indispensable de rendre la collaboration, entre les différents acteurs d'un projet, simple et efficace, afin de faciliter et d'accélérer la prise de décisions.

Le management visuel et notamment la **modélisation métier** est une étape préalable, qui permet la collaboration entre métiers, développeurs et testeurs. Tout en amenant une vision macro des applications métiers, d'une part et, d'autre part, des parcours critiques applicatifs qui repensent l'expérience utilisateur.

Cet outil permet de mettre en place un langage commun et une documentation vivante.

La modélisation orientée test vient dans un deuxième temps, pour approfondir la modélisation métier. Elle permet la collaboration entre les mêmes acteurs et étoffe ainsi la documentation vivante. C'est lors de cette phase de modélisation que l'équipe projet visualise et affine sa stratégie de tests. Elle peut servir la vision actuelle du projet (couverture de test, stratégie, communication de référence) ou sa vision cible (impact et évolution sur les tests, identification des manques, besoins en environnements et données).

Lors de l'étape de **scripting**, les automaticiens s'appuient sur les phases de modélisation, afin d'avoir une vision claire.

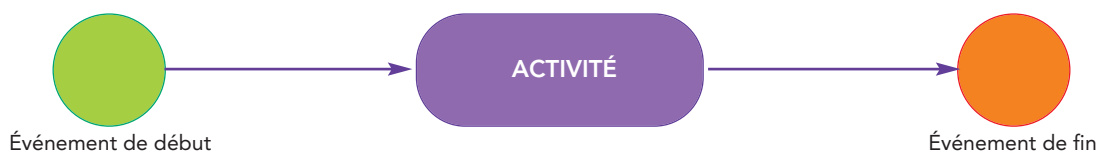
Généralement, l'automatisation se concentre dans un premier temps sur les fonctionnalités principales, les parcours de bout en bout vitaux et sur les cas passants et, dans un deuxième temps, sur les cas d'erreurs, les cas aux limites et les fonctionnalités importantes.

3.2- Pourquoi la modélisation

La modélisation est une méthode collaborative, qui permet de représenter visuellement et rapidement les processus métier grâce à un vocabulaire commun. Lorsque différents acteurs travaillent ensemble sur un projet, des barrières se dressent entre eux, qui ralentissent l'avancée du projet. En effet, les développeurs n'utilisent pas le même vocabulaire que les testeurs, qui n'utilisent pas le même que les MOA, qui n'utilisent pas le même que les autres métiers.

La modélisation permet à l'ensemble des acteurs concernés d'avoir une vue globale du projet et de visualiser toutes les interactions entre les différents systèmes internes et externes. Elle permet également une montée en compétence rapide des nouveaux arrivants, grâce à la simplicité de lecture et de compréhension du projet. Enfin, elle offre des avantages et un gain de temps conséquent aux différents acteurs, que nous vous détaillons ci-dessous.

Pour répondre à cette problématique, l'utilisation de la méthode Business Process Model Notation (BPMN, norme ISO/IEC 19510) facilite le partage des informations de manière rapide, fluide et compréhensible, car chaque acteur a le même niveau d'information visuelle. Il en découle donc un gain de temps important.



La modélisation permet aux différentes équipes d'avoir une vue globale et d'établir rapidement un état des lieux, tout en ayant une bonne visibilité sur les évolutions ou modifications futures et les impacts qui en découlent.

Plusieurs outils se basent sur ce standard. En voici quelques exemples:

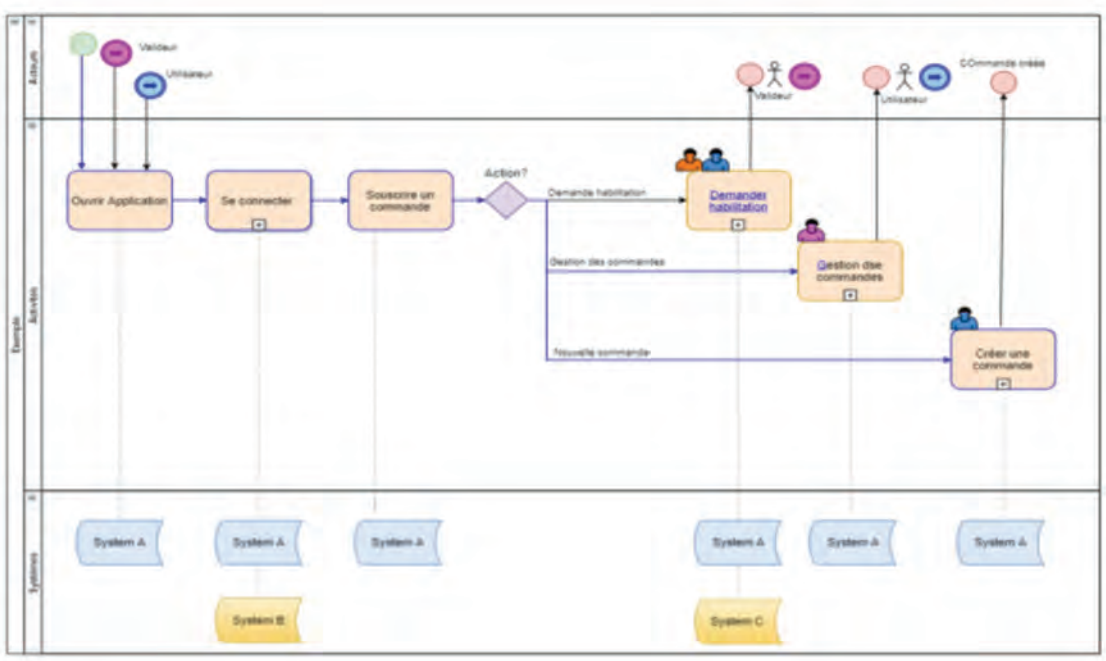
- Draw.io,
- Bizagi,
- Entreprise Architect,
- Yest.

3.3- Les avantages pour la vision transversale

La modélisation permet de faciliter l'élaboration de la stratégie de test, dont voici les principaux avantages :

- Identifier rapidement et facilement les chemins critiques et nominaux. Il est ainsi possible de mettre en place une stratégie de test et de définir ceux qui doivent être réalisés manuellement ou automatiquement, ainsi que les jeux de données et environnements nécessaires pour effectuer les différentes campagnes de test.
- Réaliser un bilan et une couverture de test de façon visuelle. La modélisation offre la possibilité de mettre en surbrillance le parcours utilisateur afin d'illustrer ce qui a été couvert et réalisé avec les tests. Cela donne une vue éclairée de ce qui va être mis en production.

Ci-dessous, un exemple de modélisation qui permet de mettre en avant les principaux parcours d'une application. En bleu sont surlignés les parcours à tester en priorité.



4- Bonnes et mauvaises pratiques

4.1- Mauvaise pratique #1 - vouloir tout automatiser

Une des croyances sur l'automatisation est de penser que tout est automatisable et qu'il faut tout automatiser. Cette pratique risque de générer un manque de fiabilité dans le temps et un coût de maintenance très élevé et va engendrer une perte de confiance dans le temps sur la fiabilité des tests automatisés.
Conseil : cibler les tests et les parcours de tests à automatiser en s'appuyant sur la modélisation.

4.2- Mauvaise pratique #2 - automatiser comme pour le test manuel

Une autre croyance est de vouloir automatiser de la même façon que les tests manuels, avec de longs tests fonctionnels et énormément de contrôles sur chaque étape de tests. Une des conséquences de cette croyance est une maintenance dans le temps très difficile et un abandon systématique des tests automatisés.
Conseil : créer des cas de tests sur une fonctionnalité avec un minimum d'objectifs de validation, ou sur le comportement d'un parcours utilisateur avec les contrôles minimum suffisants pour valider le cas de test.

4.3- Mauvaise pratique #3 - tout tester avec l'IHM

En tant que testeur, la vision bout en bout en partant de l'IHM peut laisser croire que l'automatisation doit principalement se faire avec l'IHM. Or, dans bien des contextes projets, cela implique des cas de tests trop complexes à automatiser ou lorsque cela est envisageable, une complexité et une instabilité des scripts de tests du fait des évolutions régulières des écrans et de l'ajout de nouvelles pages.
Conseil : dans ce cas de figure, se concentrer si possible sur des cas de tests d'API avec une approche de tests automatisés plus bas niveau, en s'inspirant de la pyramide des tests et inclure un minimum de tests d'IHM automatisés uniquement sur les cas critiques, dans la mesure du possible.

4.4- Mauvaise pratique #4 - l'automatisation, un projet à part

Afin d'accélérer l'automatisation, l'idée serait de monter un projet autonome « Automatisation de tests » en parallèle de l'équipe concernée, afin d'avoir une meilleure transformation.
Dans ce cas de figure, deux risques émergent : le premier est d'obtenir des cas de tests automatisés très techniques qui ne sont pas en phase avec la valeur métier, et difficilement compréhensibles et maintenables par l'équipe de test une fois le projet "test auto"" achevé et transmis à l'équipe de test.
Le second risque est une production trop importante de tests automatisés, notamment sur le bout en bout (cf #1).

Conseil : la stratégie des tests automatisés doit être incluse dans la stratégie de test et portée par les acteurs du projet. Elle doit guider le développement de tests automatisés. En se basant sur la démarche décrite précédemment, le test automatisé devient efficient et s'intègre pleinement dans une stratégie de test globale.

5- Conclusion

Trop souvent, les projets se lancent pleinement dans l'automatisation de leurs tests fonctionnels sans même avoir réalisé au préalable une stratégie de tests fonctionnels automatisés et sans prioriser les actions et identifier les cas de tests qui vont leur apporter le plus de valeur.

En pensant gagner du temps, les équipes projet en perdent finalement beaucoup en devant régler une à une des problématiques qui auraient pu être évitées.

Le management visuel, notamment **la modélisation des parcours métier**, est un outil puissant et efficace qui permet d'apporter une bonne compréhension du projet à l'ensemble des acteurs, mais aussi une bonne visibilité sur les modifications et les évolutions du projet et les impacts qui en découlent. Cela permet donc d'adapter sa stratégie de test, de prendre les bonnes décisions de manière collaborative et d'affiner les tests à automatiser à conserver, maintenir ou mettre de côté.

Cela **facilite grandement la maintenabilité** du référentiel de tests automatisés ou non.

II.2- Gestion industrialisée et automatisée de l'effort de test.

par Jean-Paul Gallant

1- Les difficultés toujours constatées à lier le besoin métier au test.

S'il est vrai que dans le numéro de juin 2011 (déjà 10 ans) de la lettre du CFTL, Bertrand Cornanguer introduisait la notion des exigences métier et de leur rôle dans le pilotage de la qualification fonctionnelle applicative, il n'en reste pas moins qu'aujourd'hui chez Altran, nous observons toujours pour un grand nombre de nos clients, tous domaines confondus, des difficultés à utiliser ce lien qui doit rester intangible entre le métier et le domaine du test.

Cela est le cas aussi bien pour les projets cycle en V que, paradoxalement, pour les projets en mode Agile où le lien est cependant resserré entre les utilisateurs et l'équipe de développement.

La recette, qu'elle soit de type utilisateur (on valide la conformité de la réponse des fonctions applicatives aux besoins utilisateurs) ou de type fonctionnel et technique (on vérifie le bon fonctionnement sans anomalie de l'application), fait systématiquement l'objet de quatre thèmes récurrents.

- Les difficultés de communication dans le cadre des projets cycle en V, entre la maîtrise d'ouvrage (MOA) et la maîtrise d'œuvre (MOE) qui aboutissent généralement à un cahier des spécifications fonctionnelles incomplet, mal explicité par la MOA et mal compris par la MOE.

- Pour les projets en mode Agile, des difficultés pour raccrocher les User Stories à des exigences métier situées au bon niveau, qui aboutissent à une maîtrise insuffisante des tests de régression et à un nombre important d'anomalies en production. On est loin, dans ce cadre, du fonctionnement en mode « DevOps », c'est-à-dire de l'intégration continue (automatisation sur l'ensemble de la chaîne de production applicative jusqu'à la mise en environnement de préproduction) ou du déploiement continu (automatisation complète jusqu'à la mise en production).

- Une faible complétude de la couverture des tests (de nombreux cas de tests n'ont pas été prévus). Ce point est complémentaire des deux points précédents.

- Les difficultés associées à la maîtrise des délais de tests (délais de tests non respectés face à une fréquence de livraisons applicatives parfois trop importante ou, en mode Agile, à des durées de sprint trop longues).

2- Comment lever ces difficultés ?

Ces difficultés proviennent de l'absence totale d'une répartition cohérente de l'effort de test relatif à la criticité des exigences applicatives et, pour les tests de régression, de l'absence de répartition de l'effort des tests assujettis à l'impact des exigences métier entre elles.

La qualité des livrables applicatifs est largement augmentée par une focalisation pertinente de l'effort de test sur les points critiques, mais aussi par une meilleure communication entre maîtrise d'ouvrage et maîtrise d'œuvre ou une meilleure association entre les besoins métier et le test.

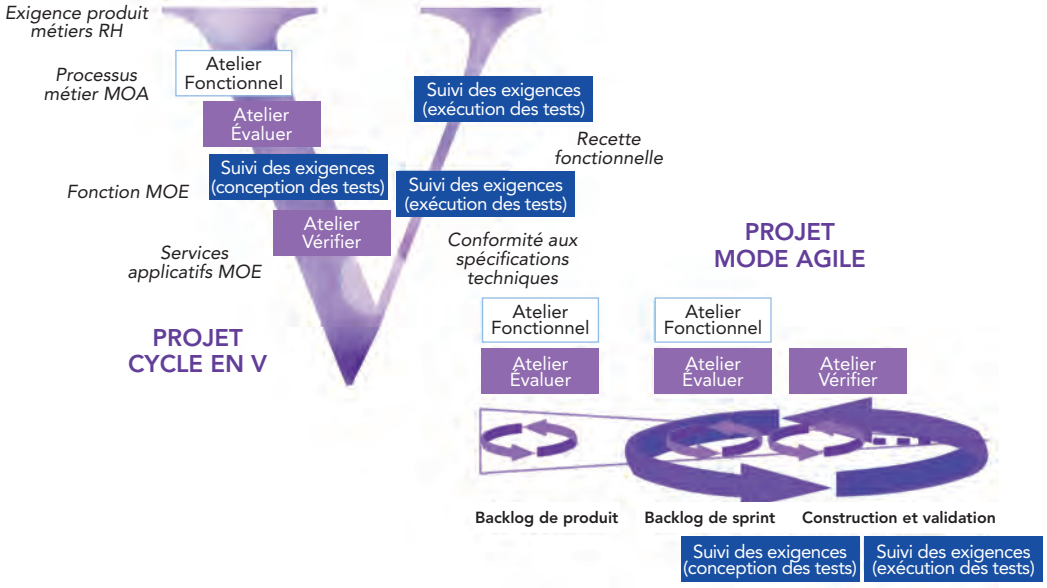
La mise en œuvre d'une stratégie de test, en parallèle à l'expression des besoins, dans le cadre du cycle en V, va donner à la vision métier de la MOA, du fait des échanges nécessaires entre ces deux services, une perception pratique orientée sur les exigences métier, les cas d'utilisation et les cas de test. En mode Agile, cette action doit être réalisée en amont lors du Backlog de produit et en aval avec une revue lors du Backlog de sprint (voir schéma suivant).

Cette double perception va permettre une meilleure orientation dans la rédaction des besoins métiers et, par là même, permettre d'aboutir à un cahier des charges fonctionnel mieux compris par la MOE ou, pour le mode Agile, à des User Stories correctement rédigées et convenablement reliées à des exigences métier au bon niveau.

3- Quels moyens mettre en œuvre ?

La complétude de la couverture des tests et la répartition cohérente de l'effort de test vont être obtenues par des méthodes adaptées décrites ci-dessous. Elles doivent être mises en œuvre très en amont (lors des spécifications fonctionnelles ou des Backlog de produit et de sprint).

Ces méthodes (déposées par Altran) se pratiquent sous la forme d'un atelier fonctionnel, c'est-à-dire des réunions de travail où est rétroprojetée une matrice de couverture et à laquelles participent le responsable du domaine applicatif et/ou un voire deux sachant-s métier et l'expert test Altran.



Ce schéma représente la position de l'atelier fonctionnel dans un projet cycle en V et en mode Agile.

Lors de cet atelier, la matrice de couverture est constituée, les niveaux de criticité des exigences métier sont évalués et l'ensemble des cas d'utilisation sont définis. Cet atelier permet également d'analyser et de vérifier la cohérence des éléments d'information constitués en amont.

Cet apport est d'autant plus important lorsque les besoins métiers sont exprimés avec des ambiguïtés ou un manque de clarté, ou encore lorsque les spécifications du cahier des charges fonctionnel manquent d'exhaustivité.

Cet atelier permet, en outre, d'obtenir une complète maîtrise des tests de régression manuels ou automatisés en optimisant l'effort associé.
Cette optimisation, modulée suivant deux critères que sont l'influence et la sensibilité des exigences, se traduit par une hiérarchisation des scénarios de tests de régression par ordre de priorité. L'effort associé au développement des scénarios de test automatisés est également optimisé.

4- Mode opératoire de l'atelier fonctionnel & matrice de couverture.

Le mode opératoire de cet atelier consiste à croiser les exigences métier préalablement définies avec les cas d'utilisations listés directement sur la matrice lors de cet atelier. Ces exigences métier sont constituées de telle sorte qu'elles correspondent à un niveau stratégique cohérent, ni trop bas ni trop élevé, relativement aux besoins métier (en général, on n'excède pas les 25 exigences, qu'il s'agisse d'une seule application ou d'un système d'information composé par exemple d'une quarantaine d'applications reliées entre-elles).

Les cas d'utilisation doivent être suffisamment explicites et libellés de la manière suivante : En tant que [utilisateur, gestionnaire, …] je fais : [les actions que l'utilisateur peut faire dans le cadre de l'exigence métier]. Exemple : « En tant qu'Agent, je saisis les informations associées au contribuable ». Dans cet exemple, l'exigence associée peut être par exemple : « Création d'un dossier contribuable ».

Lorsque toutes les exigences ont été couvertes par les cas d'utilisation, une seconde passe permet de relier les cas d'utilisation listés à d'autres exigences, en définissant de cette manière les impacts fonctionnels. L'impact d'une exigence (ou d'une ou plusieurs fonctions qui lui sont associées) sur une autre exigence est associé au fait qu'une anomalie, sur une des fonctions associées à la première, peut provoquer des dysfonctionnements sur la seconde.,

Le nombre d'impacts est automatiquement comptabilisé et permet d'obtenir des notes associées aux notions que sont la « sensibilité » et l'« influence » (on dira qu'une exigence est d'autant plus influente qu'elle impacte les autres exigences, on dira qu'une exigence est d'autant plus sensible qu'elle est elle-même impactée par les autres exigences). L'effort de test est ainsi modulé suivant ces deux notions et suivant la criticité des exigences identifiée précédemment.

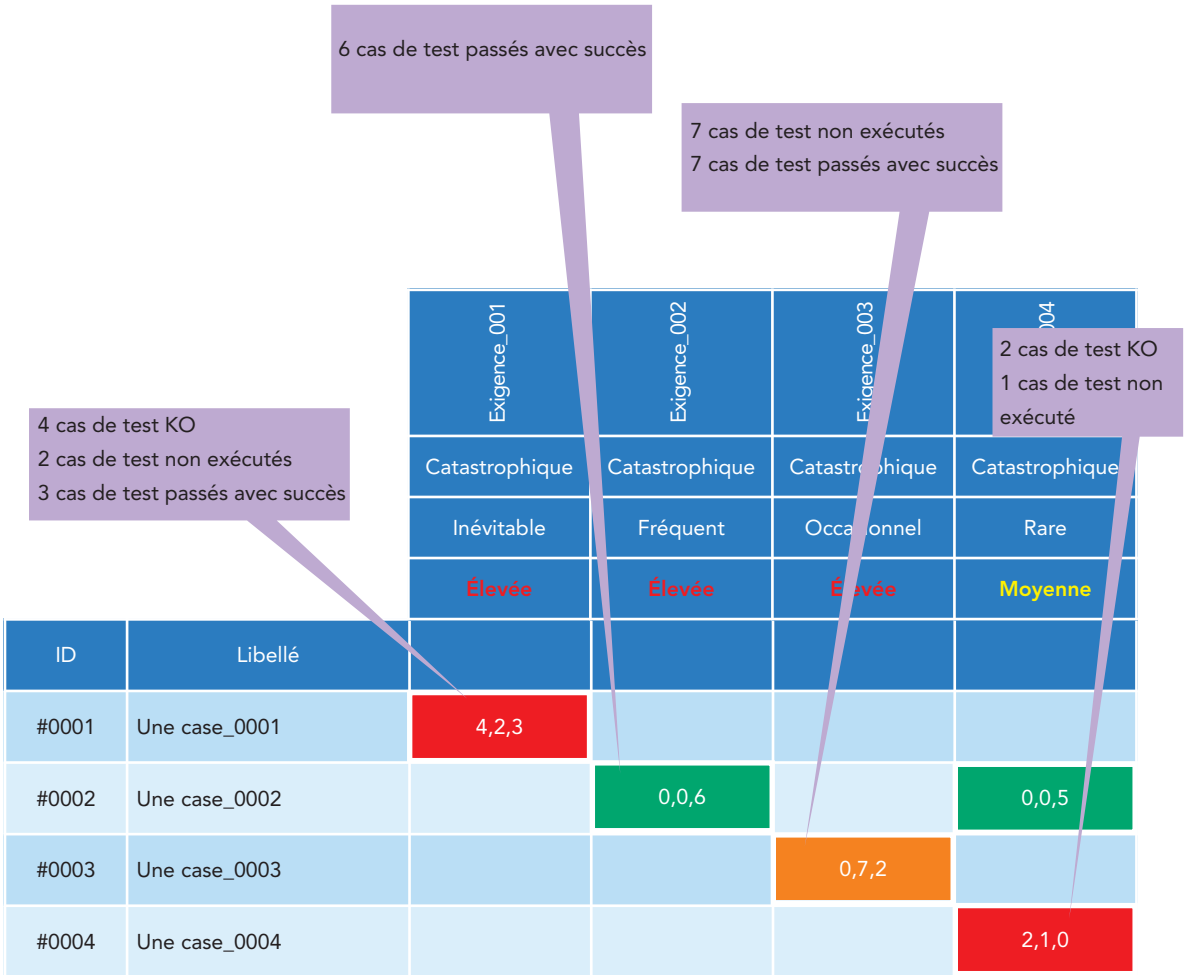
		Exigence 1	Exigence 2	Exigence 3	Exigence 4	Exigence 5	Exigence 6	Exigence 7	Exigence 8	Exigence 9
		AA	AB	AC	AD	AE	AF	AG	AH	AI
	Impact	C	M	M	E	G	C	C	G	E
	Fréquence d'apparition du risque	I	O	F	I	F	F	D	F	I
	Criticité	3	1	2	2	3	3	3	3	2
CU01	Cas d'utilisation 1	X								
CU02	Cas d'utilisation 2		X							
CU03	Cas d'utilisation 3			X						
CU04	Cas d'utilisation 4			X			O		O	
CU05	Cas d'utilisation 5			X						
CU06	Cas d'utilisation 6			X					O	
CU07	Cas d'utilisation 7				X					
CU08	Cas d'utilisation 8				X					
CU09	Cas d'utilisation 9					X				
CU10	Cas d'utilisation 10						X			
CU11	Cas d'utilisation 11				O		X			
CU12	Cas d'utilisation 12						X	O	O	O
CU13	Cas d'utilisation 13						X			
CU14	Cas d'utilisation 14							X		
CU15	Cas d'utilisation 15							X		
CU16	Cas d'utilisation 16				O			X		
CU17	Cas d'utilisation 17							X		
CU18	Cas d'utilisation 18							X		O
CU19	Cas d'utilisation 19				O	O		X		O
...	Cas d'utilisation n							X		

Ce schéma représente une matrice de couverture. Les liens directs entre les exigences et les cas d'utilisation sont en bleu. Les impacts fonctionnels sont en rouge.

La charge de recette, à l'issue de cet atelier, est ensuite dimensionnée en utilisant une méthode spécifique d'évaluation à partir de la matrice de couverture. Cette charge est adaptée et répartie de la manière la plus optimale possible, en fonction des niveaux de criticité et des notions d'« influence » et de « sensibilité » présentées plus haut.

Les cas de test, à l'issue de cette analyse, sont définis aux croisements des exigences et des cas d'utilisation, c'est-à-dire à l'emplacement des cellules de la matrice. Les cas de test resteront liés à ces emplacements en conservant les coordonnées des cellules de la matrice auxquelles ils sont liés.

Lors de leur conception et de leur exécution, les statuts des tests sont mappés sur la matrice, d'une part par des indications du nombre des tests par statut dans chaque cellule, d'autre part par une couleur afférente associée aux statuts.

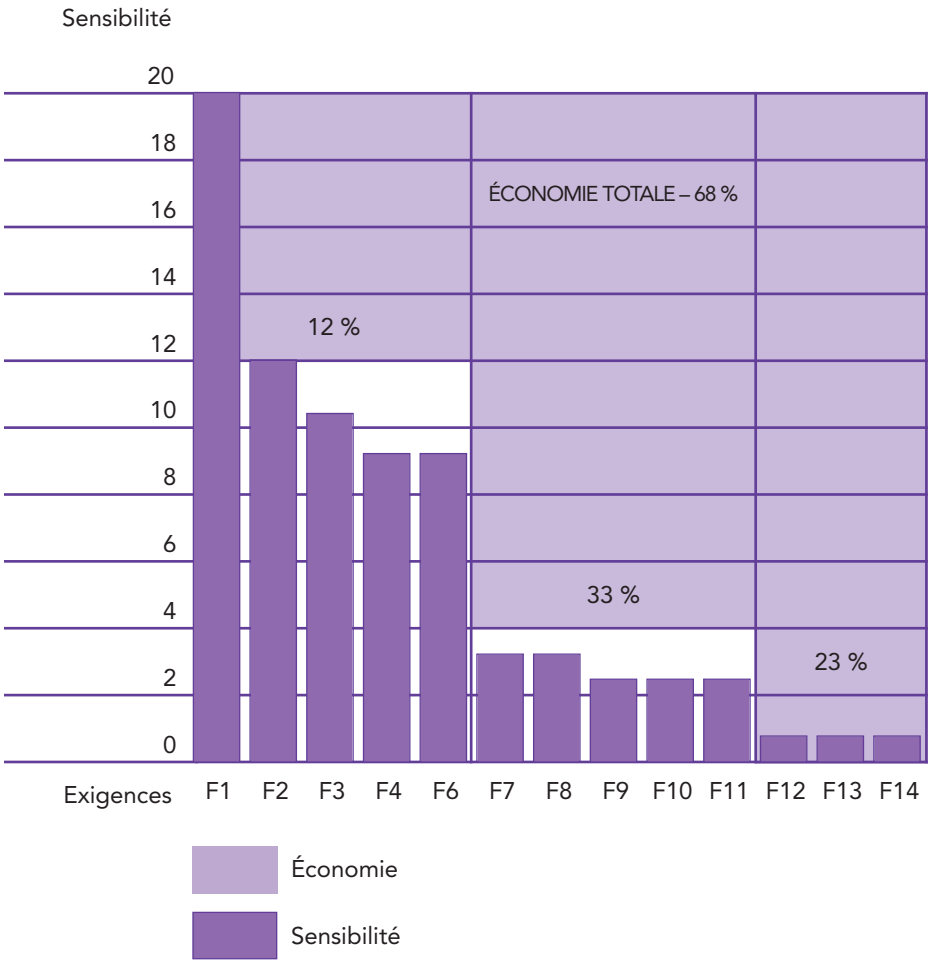


Représentation d'une matrice de couverture avec les statuts d'exécution des tests mappés à l'intérieur des différentes cellules

Cette méthode est contenue dans un outil Excel relié aux outils de gestion des tests existants. Cet outil permet d'automatiser la gestion des liens, la priorisation des tests et le suivi des tests de régression.

5- Gestion de l'effort de test de régression.

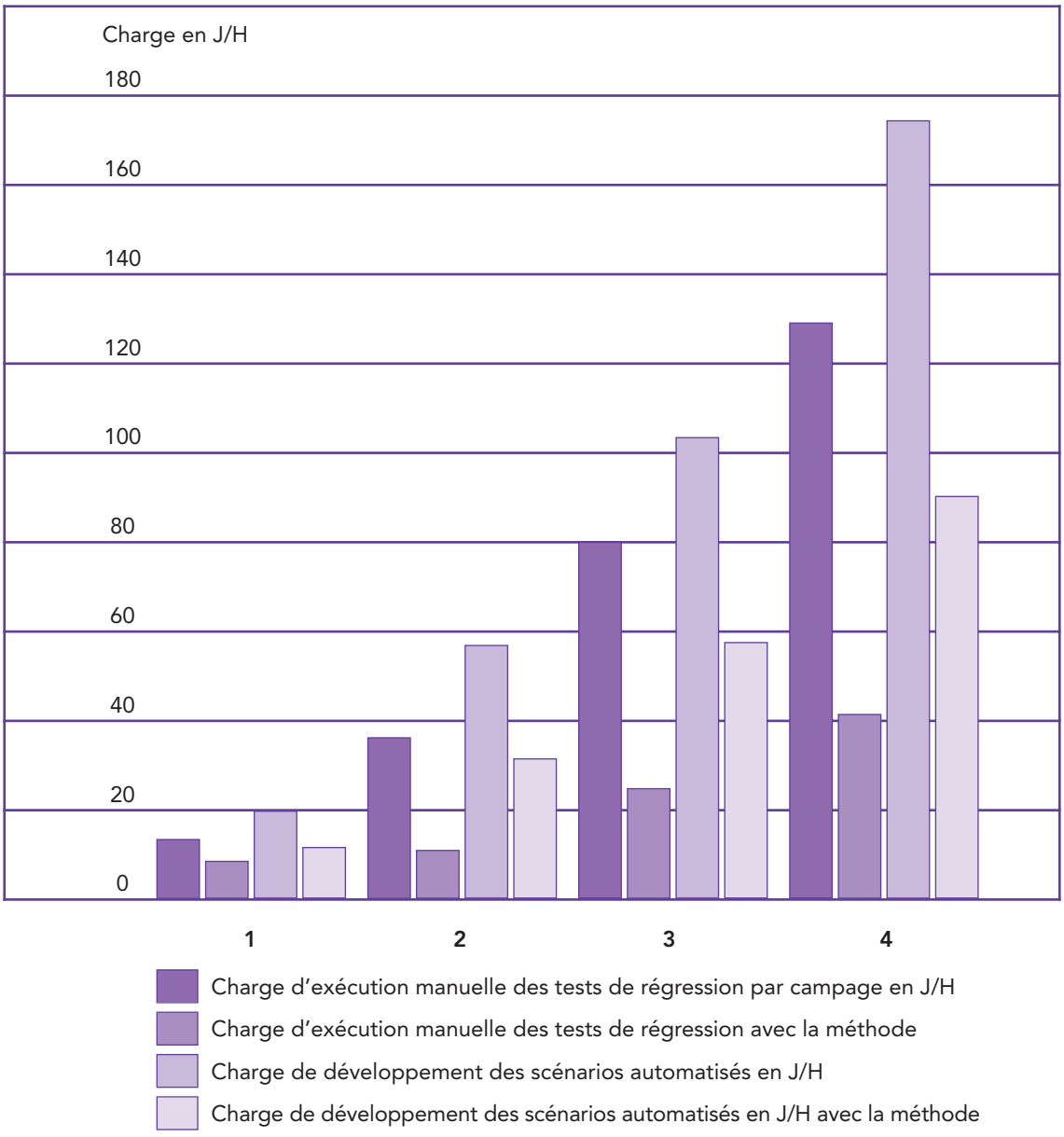
L'expérience montre, au travers de projets mis en œuvre chez des acteurs majeurs de l'administration publique, des transports, de l'énergie, de la sureté nucléaire…, que cette méthodologie permet d'économiser en moyenne 60 % de la charge de travail associée à la mise en œuvre des tests de régression, tout en optimisant de manière importante la qualité des tests. Il ne s'agit pas tant de diminuer l'effort de test de régression par économie de budget, mais bien de le répartir de manière cohérente aux endroits où il est le plus nécessaire pour obtenir un niveau de qualité optimal.



Exemple sur l'optimisation de l'effort de test de régression manuel en fonction de la sensibilité

Dans le cas présenté ci-dessus, il s'agit des tests manuels. Pour ce qui concerne l'effort associé au développement des scénarios de tests automatisés, celui-ci est optimisé par une priorisation des scénarios les plus « sensibles » et les moins à même d'évoluer.

Le schéma ci-après montre les écarts de charge entre une exécution manuelle des tests de régression avec cette méthode. Ce même schéma présente également les écarts de charge entre un développement des scénarios automatisés avec et sans la méthode. Cette étude provient d'une synthèse des résultats obtenus sur plusieurs projets mis en œuvre chez un constructeur automobile français, ainsi que chez un acteur majeur de l'administration publique.



Le ROI de l'automatisation des tests représente les coûts de développement, d'exécution et de maintenance des scénarios automatisés, comparés aux données relatives au coût des tests manuels fournis par le client.

Pour obtenir rapidement un ROI positif de l'automatisation des tests, on n'automatisera qu'une partie essentielle des scénarios en respectant la hiérarchisation de ces scénarios, suivant le résultat obtenu dans la matrice.

II.3- Répondre aux enjeux de l'automatisation des tests logiciels en Agile avec l'ATDD visuel

par Elodie Bernard et Fabrice Ambert

La transformation digitale et Agile des grandes organisations requiert un changement en profondeur des pratiques des tests logiciels. La mise en production des évolutions des systèmes informatiques à un rythme de plus en plus rapide, sur des systèmes de plus en plus complexes, remet en cause les pratiques traditionnelles, en particulier celles des tests fonctionnels. Historiquement, ces pratiques se caractérisent par une forte composante de tests manuels et par un manque de maîtrise de la couverture des tests.

Les enjeux des tests logiciels dans la transformation Agile s'inscrivent dans la capacité des approches et outils de test :

- à s'adapter aux courtes itérations de l'Agile,
- à mettre en œuvre des mécanismes de qualité pour la conception, la maintenance et l'exécution des tests,
- à être efficace dans la conception pour différents objectifs de couverture de tests,
- à parvenir à assurer la maintenance des suites de tests manuels, grandissantes au fil des itérations,
- à supporter l'exécution de tests manuels et automatisés, avec une adaptation trans parente des premiers aux seconds.

Dans ce chapitre du livre du CFTL, nous présentons une approche d'ATDD (Acceptance Test Driven Development) visuel adaptée aux tests fonctionnels en Agile. Cette approche vise à répondre aux enjeux des tests logiciels dans la transformation Agile, en s'appuyant sur une formulation visuelle des parcours applicatifs à tester, pour réaliser l'automatisation de la conception à l'implémentation des tests.

L'outillage de l'ATDD visuel s'appuie à la fois sur un mécanisme de génération automatique des cas de test et sur des supports pour :

– la création et la maintenance des données de tests,
– la production des scripts automatisés.

Cette approche participe donc à l'automatisation des activités de conception et d'implémentation des tests fonctionnels, lors de leur création et de leur maintenance.

Ce processus outillé de l'ATDD visuel facilite le travail collaboratif au sein de l'équipe projet, en permettant l'alignement de l'équipe Agile (parties prenantes orientées Métier -- PO et analystes métier --, testeurs et développeurs) autour d'une scénarisation visuelle des tests.

Processus d'ATDD visuel par la scénarisation des tests

L'approche est fondée sur la scénarisation visuelle des tests fonctionnels par le biais de parcours applicatifs représentés sous forme graphique. Ces parcours applicatifs graphiques constituent une abstraction d'un ensemble de scénarios de test, discutés entre les différents acteurs de la solution. Il s'agit de faciliter les interactions entre les membres de l'équipe, lors du raffinement progressif de l'expression du besoin par la discussion des scénarios de test d'acceptation.
La construction de l'approche est comparable à la façon dont le BDD (Behavior Driven Development) s'articule. Dans la figure 1 ci-dessous est présenté le cycle du BDD tel que proposé par Cucumber, un framework très répandu d'implémentation du BDD. Pour Cucumber, les entrants à l'approche sont les User Stories qui vont d'abord être découvertes, puis celles dont les critères d'acceptation sont formulés sous la forme de scénarios de test au format "Given/When/Then". Ce formalisme vise aussi à faciliter leur automatisation par l'utilisation de mots-clés. Il est possible de revenir à une étape précédente du cycle si des défauts ou des changements apparaissent.

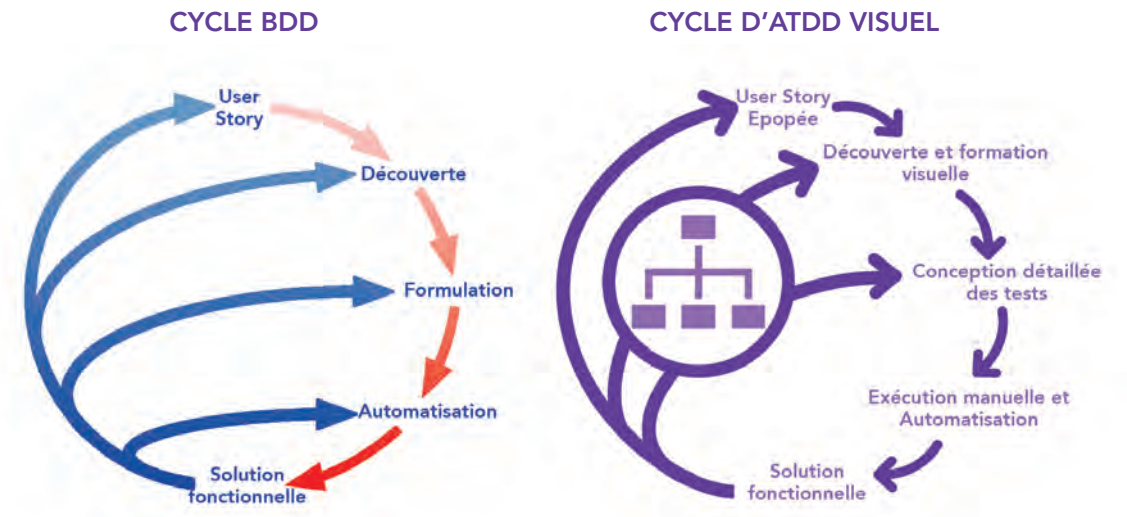


Figure 1 : Cycle BDD et cycle d'ATDD visuel

L'approche d'ATDD visuel se distingue tout d'abord du BDD par le fait qu'elle ne prend pas seulement en compte les User Stories, mais aussi des éléments d'expression du besoin de plus haut niveau, tels que des épopées représentant des fonctionnalités métier majeures ou des MMF (Minimum Marketable Feature, fonctionnalités minimales commercialisables). Ainsi, il est possible de couvrir un périmètre ciblé par le biais des User Stories ou un périmètre plus large par le biais des épopées et des MMF. Comme pour le BDD, le cycle de l'ATDD visuel débute par la découverte du besoin et la formulation des scénarios de test d'acceptation.

Une autre différence avec le BDD provient de l'usage de la représentation visuelle, qui intervient dès les phases de découverte et de formulation. La représentation des scénarios de test d'acceptation, sous une forme graphique, permet d'exprimer visuellement la façon dont

s'articule une fonctionnalité au sein du flot métier et la façon dont chaque fonctionnalité interagit avec les autres.

Enfin, comme pour le BDD, il est possible de choisir d'automatiser l'exécution des tests ou de rester sur une exécution manuelle. Si, durant l'exécution des tests, des défauts sont relevés, notons que l'on peut revenir à n'importe quelle étape de ce cycle d'ATDD visuel. On peut aussi revenir à l'une des premières phases si l'on souhaite implémenter un changement sur les tests produits.

La suite de ce chapitre présente les différents aspects de l'approche d'ATDD visuel avec un focus méthodologique et organisationnel, exprimé aussi à travers les bonnes pratiques issues de notre expérience.

1- Découverte des besoins métier et formulation visuelle des scénarios de tests d'acceptation

L'approche d'ATDD visuel commence par l'identification des entrants qui sont les User Stories, les MMF ou bien les épopées. Cela définit, pour une itération donnée, les évolutions à traiter.

Ensuite vient la phase de découverte des différents besoins métier identifiés et de formulation graphique des scénarios d'acceptation. L'expression du besoin est généralement réalisée en Agile de façon peu formelle et peu documentée, sous la forme de courts textes, de cartes, de tickets, généralement disponibles dans des outils de gestion de projets Agiles, comme Jira ou autres. La formulation des scénarios d'acceptation en ATDD visuel s'appuie sur une représentation visuelle des parcours applicatifs à tester.

La phase de formulation visuelle des scénarios de test d'acceptation contribue à renforcer la compréhension des besoins métier, tout en représentant ce que l'on cherche à vérifier. Pour cela, il s'agit de modéliser les parcours applicatifs à tester, en les décrivant de façon aussi simple que possible en fonction des objectifs du test, comme présenté dans la figure 2 ci-dessous.

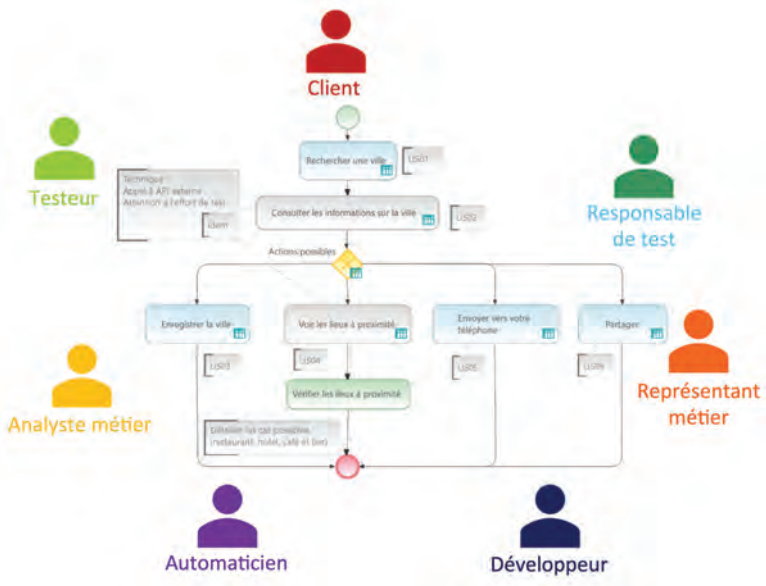


Figure 2 : Exemple de représentation visuelle d'un parcours applicatif à tester

L'expression atomique du besoin sous la forme des User Stories ne permet pas nativement de connaître le lien qui existe entre les différentes fonctionnalités. Mais les parcours applicatifs modélisés permettent de visualiser quelles sont les activités indépendantes les unes des autres et celles qui sont liées. Ainsi, dans l'exemple de la figure 2, les fonctionnalités "Enregistrer la ville", "Voir les lieux à proximité", "Envoyer vers votre téléphone" et "Partager" ne sont accessibles qu'après la réalisation de l'action "Rechercher une ville". L'approche d'ATDD, par ses formalisations visuelles, permet de donner une vision globale des parcours applicatifs que l'on souhaite tester (illustré ici sur un exemple de recherche de ville et d'itinéraire). Les parcours applicatifs ne remplacent pas la description textuelle des User Stories mais constituent un complément d'information pour l'équipe. Ils sont revus lors d'ateliers impliquant les parties prenantes fonctionnelles Métier (PO et analystes métier), les testeurs et les développeurs, pour permettre d'aligner les contributeurs du projet sur une même compréhension du besoin.

Lorsqu'un consensus sur les parcours applicatifs à tester est obtenu, l'étape suivante du cycle d'ATDD visuel peut débuter. Il s'agit alors d'établir la conception détaillée des tests.

2- Conception détaillée des tests

Après avoir représenté les besoins métier au moyen de parcours applicatifs graphiques, l'approche nécessite de réaliser une conception détaillée des tests. Cette conception détaillée vise à spécifier toutes les informations nécessaires pour les tests. Cela inclut la définition des actions, des résultats attendus et, si besoin, des données abstraites, pour définir les conditions permettant de tester les règles métier. La création des données abstraites, dans le processus d'ATDD visuel, est réalisée par des tableaux unissant les données abstraites avec leurs actions de test métier.

Dans la figure 2, la première action "Rechercher une ville" peut être définie avec l'action détaillée "Entrer une ville dans le champ de recherche et cliquer sur le bouton rechercher" et, comme résultat attendu, "Les détails sur la ville sont affichés". La définition des actions de test et des résultats attendus, ainsi que le contrôle des règles métier, sont des activités habituelles des testeurs fonctionnels.

En phase de conception détaillée des tests, l'objectif est de réaliser la couverture des conditions de test avec des données abstraites. Il vaut mieux ne définir que les données nécessaires, c'est-à-dire celles qui découlent des exigences, ou qui sont utiles pour gérer les flots métier dans les parcours applicatifs. Ceci permet de limiter le nombre de données de conception de test à maintenir et de se focaliser sur les objectifs de test. Dans l'exemple présenté dans la figure 2, l'url du site web ainsi que le nom de la ville recherchée ne sont pas des données nécessaires à la couverture des objectifs de test. En revanche, il est obligatoire de connaître leurs valeurs pour l'exécution. Elles apparaîtront dans les données d'implémentation qui seront présentées dans la section qui suit.

Bonne pratique : dans la phase de conception détaillée des tests, seules des données abstraites en lien avec les règles métier doivent être intégrées.

Les parcours applicatifs, complétés par la description des étapes de test et la couverture des conditions de tests sur les données de conception, permettent la génération automatique des cas de test.

Pour garantir l'alignement des cas de test avec les règles métier, nous préconisons de veiller à mettre à jour les parcours applicatifs qui les génèrent, dès que des évolutions sont exprimées. Une mise à jour efficace des parcours applicatifs est possible par leur organisation par fonctionnalité et sous fonctionnalité métier.

Bonne pratique : les parcours applicatifs doivent être organisés par fonctionnalité et sous fonctionnalité métier, et tenus à jour. Cela signifie que les nouveaux comportements doivent être intégrés dans les parcours applicatifs dès que possible, de manière à regénérer au plus tôt les cas de test mis à jour. La mise à jour des parcours applicatifs et la génération des tests participent ainsi à la maintenance des cas de test et des données abstraites de test qui les composent. La génération des cas de test constitue un support au sein de l'approche, dans l'automatisation des activités de conception et d'implémentation des tests fonctionnels.

Pour un testeur connaissant bien le système sous test, ces cas de test abstraits peuvent suffire pour guider l'exécution des tests. Il/elle sera à même de compléter les données de test manquantes ou trop abstraites. Mais dans le cas général, définir les valeurs concrètes des données de test nécessaires à l'exécution permettra une formulation plus précise des cas de test.

La section suivante présente les activités d'implémentation des tests pour l'exécution manuelle et l'automatisation.

3- Exécution manuelle et automatisation

Une fois les cas de test conçus, l'objectif de l'implémentation des tests est de permettre l'exécution manuelle ou automatisée des tests. Les cas de test manuels et les scripts automatisés nécessitent la transcription des données de conception vers les données concrètes d'implémentation. Cette étape consiste à faire correspondre toutes les données abstraites des parcours applicatifs avec des valeurs concrètes.

Pour faire correspondre les données de conception aux données d'implémentation, différents mécanismes sont possibles. Nous préconisons de faire ce rapprochement à l'aide de tableaux, comme cela est actuellement déjà d'usage pour les testeurs fonctionnels, dans leur travail sur les données de test.

Bonne pratique : utiliser des tableaux pour associer chacune des données de conception à sa ou ses données d'implémentation.

Dans notre exemple de recherche de ville, seules des données d'implémentation sont nécessaires. Elles sont présentées dans la figure 3 ci-après. Le tableau permet de tester 4 cas possibles de recherche de ville, sous forme de jeux de données.

	Nom du jeu de données	lieu_départ	site_web
1	Jeu de données ville Est	*Besançon*	*https://www.google.com.maps*
2	Jeu de données ville Nord	*Lille*	*https://www.google.com.maps*
3	Jeu de données ville Sud	*Marseille*	*https://www.google.com.maps*
4	Jeu de données ville Ouest	*Brest*	*https://www.google.com.maps*

Figure 3 : Exemple de jeu de données pour l'implémentation des tests

Bonne pratique : nous préconisons, pour faciliter la gestion des données d'implémentation, de veiller à nommer de manière rigoureuse ces données, ainsi que les jeux de données qui les contiennent.

Une fois la transcription des données de conception vers des données d'implémentation réalisée, les tests conçus en ATDD visuel sont d'abord exécutés manuellement durant l'itération.
À chaque itération, une partie des tests manuels va être sélectionnée, pour être ajoutée au référentiel de test automatisé de non-régression. La sélection de ces tests peut être guidée par différents critères, le principal étant d'optimiser le temps dédié à l'exécution des tests et ainsi réduire le coût global des tests. Les tests étant de bons candidats pour les campagnes de non-régression automatisées sont ceux dont les fonctionnalités parcourues sont assez stables, peu sujettes aux évolutions et où l'exécution des tests est récurrente.

Pour produire les scripts de test automatisés de non-régression réalisés à partir des parcours applicatifs formulés, il est nécessaire de compléter une couche d'adaptation. La section qui suit décrit la complétion de cette couche d'adaptation.

4- Complétion de la couche d'adaptation

Les étapes de test décrites dans le parcours applicatif et formant les cas de test sont transcrites en mots-clés d'automatisation de test avec des paramètres (les données de test). Cette conversion est possible grâce à la complétion d'une couche d'adaptation, afin de produire automatiquement des scripts de test, comme illustré dans la figure 4.

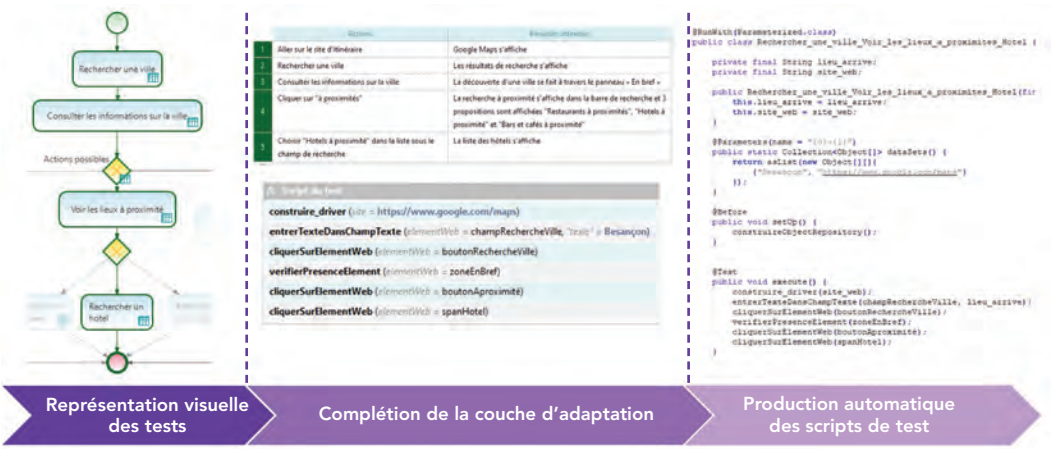


Figure 4 : Processus d'automatisation

La complétion de la couche d'adaptation se compose des étapes suivantes :

- construire les répertoires d'objets,
- implémenter les mots-clés,
- relier les mots-clés aux étapes de test des parcours applicatifs.

Pour chacune de ces étapes, nous avons émis des recommandations que nous allons présenter.

Construire les répertoires d'objets

Les répertoires d'objets sont des bibliothèques construites par l'automaticien de test, contenant tous les objets du SUT nécessaires à l'automatisation et qui sont intégrées dans la solution d'automatisation. Ainsi, la solution d'automatisation se compose des répertoires d'objets qui incluent des boutons, des champs, des formulaires ou tout autre objet sur lequel il est possible d'effectuer des actions pour tester l'application.

Bonne pratique : les objets doivent être nommés et identifiés avec précision pour faciliter leur gestion et maintenance.

Le nommage doit idéalement combiner vision technique et métier. Pour couvrir l'aspect technique, le nommage de l'objet peut contenir le type du champ qu'il référence, par exemple un champ texte, un bouton, etc. Ceci permet de donner un indicateur, autant à l'automaticien qu'au testeur fonctionnel, sur les actions réalisables sur l'objet (un clic pour un bouton, une complétion pour un champ texte).

Un extrait de répertoire d'objets est présenté dans la figure 5, ci-dessous. Le champ de recherche de la ville, ainsi que le bouton pour effectuer la recherche, combinent la vision technique et la vision métier.

```
//Champ pour la recherche de ville et bouton pour valider la recherche
public static String champRechercheVille = "//*[@id='searchboxinput']";
public static String boutonRechercheVille = "//*[@id='searchbox-searchbutton']";
```

Figure 5 : Exemple d'application de la bonne pratique de nommage des objets

Bonne pratique : les objets doivent être organisés efficacement dans la solution d'automatisation afin de permettre une mise à jour facile pour l'automaticien.
La construction des répertoires d'objets doit permettre de facilement localiser les objets à mettre à jour lorsque nécessaire. Les objets peuvent être classés par interface ou par fonctionnalité, selon la façon dont l'application est mise en place.

Implémenter les mots-clés

Les mots-clés sont des fonctions d'automatisation permettant d'activer les comportements du SUT. Elles sont généralement produites par un automaticien de test. Quelle que soit la technologie utilisée, les mots-clés comportent des propriétés, au minimum un nom et des paramètres

(données, objets des bibliothèques d'objets). Elles doivent être gérées, pour faciliter leur compréhension et leur usage, par un testeur fonctionnel et leur maintenance sera gérée par un automaticien de test. Voici les bonnes pratiques que nous avons établies afin de faciliter la gestion des mots-clés et, à plus large échelle, la compréhension des scripts.

Bonne pratique : les noms des mots-clés doivent être significatifs et correspondre aux opérations qu'ils représentent. C'est d'abord par le nom des mots-clés que le testeur fonctionnel et l'automaticien de test en percevront le sens.

Bonne pratique : le testeur fonctionnel et l'automaticien de test doivent travailler ensemble et définir correctement le comportement attendu des fonctions, ainsi que le format et les valeurs des données acceptées dans les paramètres.

Relier les mots-clés aux étapes de test

Dans l'approche, la production des scripts est basée sur la liaison préalable des étapes de test à des mots-clés. Par exemple, pour l'action "se connecter à l'application", on pourra lier un mot-clé "connexion(id, mdp)", qui est une fonction permettant la connexion sur le SUT à l'aide d'un identifiant et d'un mot de passe. Une fois les mots-clés reliés aux étapes de test, la production de scripts peut être réalisée, comme présentée dans la section qui suit.

5- Production des scripts

Dans l'approche, la génération des scripts est réalisée grâce aux représentations visuelles. C'est l'ensemble des mots-clés, associés aux étapes de test, qui va permettre la génération des scripts. Les scripts sont, à notre sens, l'artefact d'automatisation le plus fragile, leur fonctionnement dépendant à la fois des données et des mots-clés. La plupart des modifications opérées au niveau du SUT, tant au niveau des comportements qu'au niveau des données, nécessitent de reconsidérer les scripts de test.

La maintenance de scripts, répondant aux règles métier, est possible par l'application des bonnes pratiques présentées précédemment. C'est notamment la mise à jour des parcours applicatifs qui va permettre la génération de cas de test couvrant les évolutions des règles métier, mais aussi la production et la maintenance des tests de non-régression automatisés. Ensuite, la mise à niveau de la couche d'adaptation va permettre de générer les scripts et de les maintenir à jour. La section suivante présente la mise en pratique de l'approche d'ATDD visuel.

Mise en pratique de l'approche d'ATDD visuel

Dans cette section, nous restituons la mise en œuvre de l'approche d'ATDD visuel sur des projets sur lesquels nous avons travaillé ; nous présentons ici 2 cas d'applications :

1ère application : nous avons étudié deux contextes Agiles. Dans un cas, nous avons appliqué l'approche d'ATDD visuel dans sa globalité et dans l'autre, nous avons travaillé sans appliquer l'approche d'ATDD visuel. Il s'agit ainsi de discuter dans quelle mesure l'approche est adaptée au contexte Agile, en comparaison avec une approche de conception manuelle des tests. L'approche d'ATDD visuel a été mise en œuvre en contexte projet, en impliquant les testeurs en place.

2ème application : dans cette expérimentation nous cherchons à vérifier que l'automatisation des tests est réalisable dans un cycle court en Agile. Pour cela, l'expérimentation a duré 10 jours, le temps d'un sprint. Sur cette période, le testeur fonctionnel a été impliqué à plein temps, alors que l'automaticien, lui, n'est intervenu qu'à partir du 6ième jour, donc durant 4 jours.

Constats

Lors de la mise en application de l'approche sur ces projets, un ensemble de constats a pu être établi, visible dans la figure 4, et détaillé ensuite.

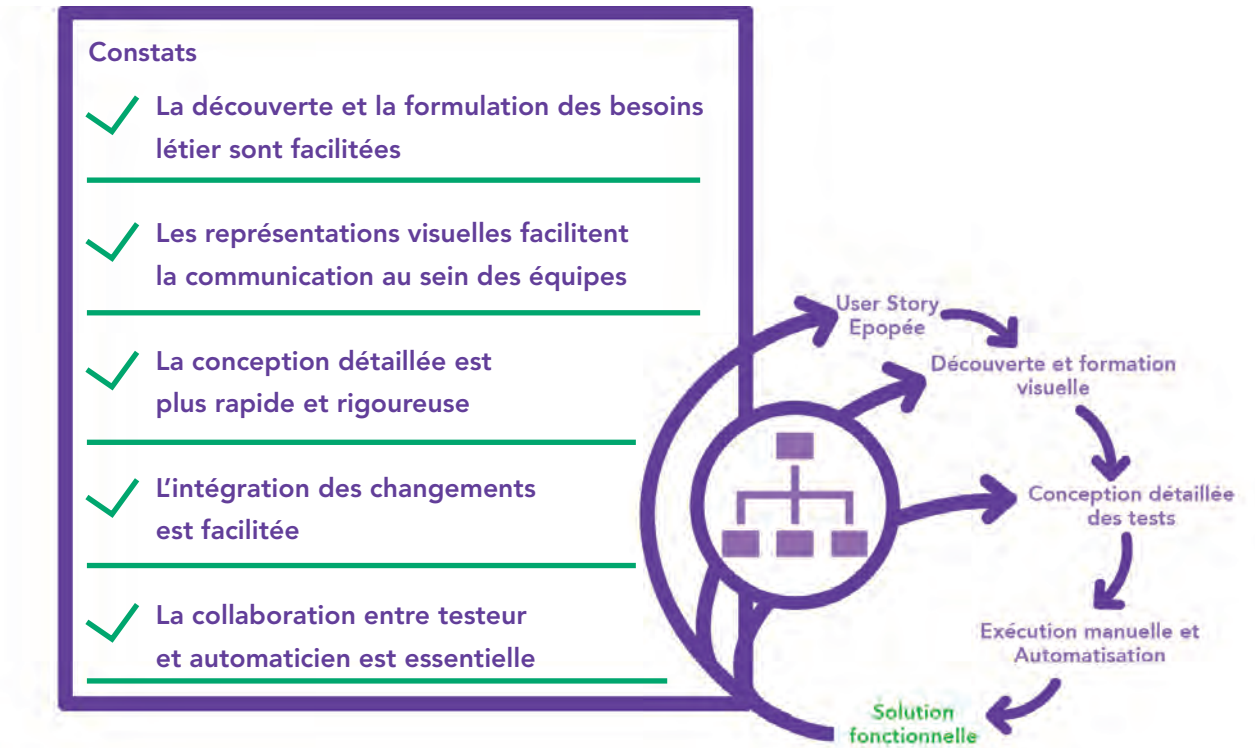


Figure 6 : Constats sur la mise en œuvre de l'approche d'ATDD visuel

La découverte et la formulation des besoins métier sont facilitées : l'application de bonnes pratiques a permis de simplifier l'activité de découverte et de formulation visuelle des parcours applicatifs. Le choix de séparer les parcours a limité la complexité des représentations visuelles. En complément, le classement de ces parcours par fonctionnalité a apporté de la visibilité sur les besoins métier.

Les représentations visuelles facilitent la communication au sein des équipes : les représentations constituent un support à la communication, en permettant à chaque partie prenante à la solution d'exprimer son point de vue et de mettre en avant des points d'attention.

La conception détaillée est plus rapide et rigoureuse : en ne définissant que les données nécessaires aux tests, la conception des tests a été allégée. Au lieu de mélanger les données liées aux exigences et à l'implémentation, cette phase n'a concentré l'effort que sur la définition de données propres au contrôle des exigences. La génération des tests donne une couverture

plus rapide des règles métier. Les tables de décision donnent un cadre pour définir rigoureusement les conditions de test des règles métier. Et pour des contextes assez similaires, avec une source de conception et d'implémentation de cas de test également très comparable, les résultats indiquent que la conception de tests avec l'approche est 30% plus rapide.

L'intégration des changements facilitée : En suivant les bonnes pratiques, la réintégration des éléments oubliés quelques jours après le début de la représentation visuelle n'a posé aucune difficulté, il était facile de retrouver les éléments concernés dans le parcours applicatif et donc de rapidement les mettre à jour.

La collaboration entre testeur et automaticien est essentielle : le testeur fonctionnel a présenté à l'automaticien de test le parcours global à automatiser, au moyen de différentes représentations visuelles (en travaillant de manière conjointe avec l'automaticien). Ceci a permis à l'automaticien d'avoir une vision globale sur le processus à automatiser. Le testeur fonctionnel et l'automaticien de test ont pu échanger sur la gestion des données des scripts. Ensemble, ils ont validé que les données d'implémentation avaient bien le format attendu par l'application, et l'automaticien a pu vérifier que les mots-clés traiteraient bien ces données.

Conclusion

Dans ce chapitre, nous avons présenté un processus outillé de l'ATDD visuel, mettant en œuvre l'automatisation des activités de productions des scénarios de test fonctionnel et des scripts. En appliquant l'approche d'ATDD visuel, nous avons pu observer qu'avec une représentation graphique très simple (type workflows), les testeurs fonctionnels ont été en mesure de représenter les règles métier à vérifier et de produire automatiquement les cas de test requis. Cela montre que l'approche est capable de vérifier de grands systèmes tout en étant simple à utiliser, accessible aux testeurs fonctionnels. L'approche a été appliquée avec succès dans un contexte de projet en Agile, avec des itérations de 4 semaines, et a montré un effort de conception des tests réduit de 30%, cela grâce au support de l'approche dans l'automatisation des activités de conception et d'implémentation des tests fonctionnels.

Sur la durée du sprint court (10 jours), l'approche a aussi été mise en œuvre de bout en bout. Les parcours applicatifs représentant les besoins métiers ont été conçus et implémentés, puis la couche d'adaptation a été complétée pour générer un script de test.

Durant toutes ces phases, le testeur fonctionnel et l'automaticien de test ont pu travailler en synergie pour faciliter chacune de leurs activités. En particulier, le testeur fonctionnel a pu aider l'automaticien à comprendre les besoins métier, grâce aux représentations visuelles et aux cas de test. Il a également lié les mots-clés aux étapes de test et généré le script automatisé, réduisant ainsi l'effort d'automatisation pour l'automaticien. Il a donc été possible en un temps limité de construire l'approche d'ATDD visuel à partir de zéro, montrant qu'elle peut être adaptée à des cycles et des itérations courtes en contexte Agile.

II.4- Sélectionner son outil d'automatisation des tests

par Marc Hage Chahine

Une mauvaise préparation de l'automatisation des tests aboutit généralement à un échec de cette initiative. Afin d'éviter de tels écueils, il faut commencer ce projet sur de bonnes bases, avec un outil adapté à notre besoin ! Pour rappel, un outil non adapté au besoin est une cause fréquente de l'échec de l'automatisation.

Contrairement à ce que l'on peut penser, le choix d'un outil n'est pas simple. Il n'y a pas de « bon » ou de « mauvais » outils par défaut mais bien des outils plus ou moins adaptés à notre contexte.

La première chose à faire est donc de bien définir son besoin, ce que l'on attend. Cette précondition sert alors de base à une étude ciblée des solutions visant à comparer différents outils d'automatisation des tests.

1- Analyse de son besoin

La première chose à faire est de bien cibler nos attentes vis-à-vis de l'outil. Cette définition a pour but d'assurer la présélection d'un ou plusieurs outils qui correspondent théoriquement à notre besoin. Il est préférable de se poser les bonnes questions comme :

1.1- Quels sont nos besoins en termes d'automatisation des tests ?

Cette question est primordiale ! C'est également, d'un point de vue technologique, une question à laquelle la grande majorité des personnes pensent lorsqu'elles souhaitent automatiser leurs tests.

Que voulons-nous automatiser ?

Les outils d'automatisations sont rarement performants dans tous les contextes. En effet, les outils vont être plus ou moins adaptés en fonction du type d'application (application lourde, API, application web, application mobile, IoT…) mais aussi en fonction du type de technologies utilisées par les applications que nous souhaitons tester (PHP, Java, NodeJS, API REST, SQL, NoSQL…).

Connaître ses besoins technologiques est essentiel, afin de s'assurer que l'outil sélectionné est en capacité de fonctionner dans l'environnement technique de développement du logiciel.

Ce n'est cependant pas suffisant !

Les besoins en termes d'automatisation des tests ne se limitent pas à l'environnement technologique.

En effet, l'outil doit généralement répondre à d'autres besoins comme sa capacité à se connecter à d'autres applications (comme une chaîne d'intégration continue ou un logiciel de gestion de test) ou encore une adaptabilité à divers environnements, comme différents terminaux mobiles Ces critères sont primordiaux car très fortement dépendants du contexte.

1.2- Quelles sont les compétences auxquelles nous avons accès ?

Ce point est également essentiel. Avoir un outil qui est capable d'automatiser ses tests est inutile si personne n'est capable d'utiliser correctement l'outil.

Pour limiter ce problème il est bon de connaître les compétences disponibles dans l'équipe et plus particulièrement les compétences des personnes qui seront amenées à automatiser les tests. Ces personnes seront-elles des automaticiens de test, des testeurs « manuels » sans aucune base de développement ou encore des testeurs ayant des compétences basiques de développement ? Ces testeurs pourront-ils être épaulés par des développeurs ou ses derniers n'auront pas forcément assez de temps à accorder sur le projet d'automatisation ?

Les outils d'automatisation des tests sont plus ou moins techniques et accessibles. Dans une équipe où les testeurs sont très axés sur le fonctionnel et moins sur la technique, il est préférable de s'orienter vers un outil facile à prendre en main et ne demandant que très peu de compétences de développement. Ce type d'outil pourra s'avérer moins intéressant dans une équipe où les testeurs sont très forts techniquement.

Dans tous les cas, la prise en main d'un outil demandera du temps, temps qui sera plus ou moins long en fonction de l'outil et des compétences des personnes étant amenées à l'utiliser.

1.3- Quelles sont nos contraintes de temps ?

Les contraintes de temps sont souvent ignorées, elles restent cependant, comme dans tout projet, primordiales !

Ces contraintes peuvent prendre 2 formes.

La première est une contrainte de délai. Dans combien de temps dois-je avoir accès à des tests automatisés ? Ce délai est impacté par la mise en place de l'outil, la montée en compétence de l'équipe sur l'outil, la création de l'automate de test (quand nécessaire), l'écriture des tests et leur intégration dans les campagnes. Chaque outil proposera des délais différents sur chacun des critères. Il reste néanmoins intéressant de noter que le critère de montée en compétence sur l'outil dépend principalement de 2 paramètres, qui sont l'ergonomie de l'outil et les compétences de l'équipe.

La seconde est sur le temps d'exécution de la campagne de test. Certains outils peuvent être plus ou moins performants. Certains outils peuvent plus ou moins faciliter une parallélisation des tests. En fonction de ces contraintes, les outils seront plus ou moins intéressants.

1.4- Quel est notre budget ?

Cette question est généralement celle à laquelle tout le monde pense mais aussi celle dont la réponse est souvent la plus éloignée de la réalité.

On a souvent tendance à penser que le coût de l'automatisation des tests se limite à la licence de l'outil. C'est malheureusement totalement faux ! D'ailleurs, si c'était le cas, tous les outils payants auraient disparu.

Non, le coût de l'automatisation, c'est le coût de la licence, le coût de formation, le coût du temps investi à développer l'automate de test et écrire les tests, le coût des infrastructures mises en place pour exécuter les tests automatisés ainsi que les coûts de maintenance…

En fonction des outils et des axes de dépenses (licences, formations, infrastructures, écriture…), les investissements seront plus ou moins importants.

Enfin, il ne faut pas oublier que des répercussions positives avec une baisse du coût d'exécution des tests peut vite débuter avant la fin du projet et donc limiter le coût de l'automatisation de par un retour sur investissement arrivant plus tôt (sur le même principe que l'Agile), avec des tests automatisés rapidement utilisés par l'équipe.

Lorsque l'on a répondu à toutes ces questions, il est alors temps de commencer une étude de marché.

2- Présélection théorique de l'outil

L'étude des solutions existantes est une étape obligatoire lorsqu'on décide de se lancer dans une démarche d'automatisation des tests. Cette nécessité vient de la spécificité de notre contexte mais aussi de la multiplicité des outils d'automatisation des tests.

Dans toute étude, il faut commencer par une étape de recherche. Quels sont les outils d'automatisations de test sur le marché ?

Lorsque cette recherche est faite il faut passer ces outils aux filtres de notre contexte. Quand les filtres permettent la sélection de quelques outils (2 ou 3), vient le moment d'organiser une étude comparative, une expérimentation, un POC (Proof Of Concept), afin de tester concrètement les outils qui correspondent théoriquement le plus à notre contexte.

Exemple :

2.1- Outils étudiés :

Notre étude de marché se base sur un panel de 20 outils.

2.2- 1er filtre : le besoin en automatisation des tests

Nous souhaitons automatiser des tests sur une application web développée en PHP. Les tests doivent pouvoir être exécutés sur Chrome, Firefox et Opéra et intégrés à une chaîne d'intégration continue sur Jenkins.

Outils restants : 15

2.3- 2ème filtre : les compétences

L'équipe qui automatisera les tests sera constituée de testeurs ayant des notions en développements mais aussi de personnes au profil métier n'ayant aucune notion de développement. Même s'il

reste possible aux profils métiers de travailler avec les testeurs, il est préférable que ces profils puissent être indépendants dans leur travail quotidien d'automatisation des tests ! Pour assurer cette indépendance, l'outil doit être accessible d'un point de vue technique.
Outils restants : 6

2.4- 3ème filtre : les contraintes de temps

Au niveau des délais : les premiers tests automatisés doivent être disponibles et exécutés moins d'un mois après le début de l'automatisation, afin de pouvoir montrer des avancées et commencer à libérer du temps d'exécution des tests.

Au niveau du temps d'exécution de la campagne : à ce jour, il n'y a pas de problématiques de temps d'exécution de campagne car la campagne sera exécutée la nuit. Néanmoins, la faculté de paramétrer une campagne ou de paralléliser les tests sera sûrement envisagée à moyen terme pour que les résultats soient disponibles au matin.
Outils restants : 4

2.5- 4ème filtre : le budget

Nous avons un budget de 2 ETP (équivalent temps plein) sur 4 mois et 10 000€ pour les licences. Au terme de ces 4 mois, le budget alloué sera égal aux économies réalisées par l'automatisation. Outils restants : 3 (1 outil avec une licence trop élevée)

Il est important de noter que cela ne reste qu'un exemple. Il est tout à fait possible de proposer des filtres supplémentaires (ex : la communauté de l'outil pour résoudre certains problèmes, la possibilité d'abandonner l'outil sans devoir réécrire tous ses tests…) en fonction de son contexte mais aussi d'appliquer ces filtres dans un ordre différent, en faisant passer les filtres les plus discriminants ou les plus importants dans son contexte en premier.

3- Expérimentation des outils présélectionnés

Bien qu'en principe, il n'y ait pas de différence entre la pratique et la théorie, nos expériences de tous les jours nous apprennent que ce qui est le mieux en théorie ne l'est pas toujours en pratique. Il est donc nécessaire, après l'étude préalable, de retrousser ses manches et d'expérimenter les outils présélectionnés – je conseille en général de tester au moins 2 outils - dans les conditions réelles de notre contexte.

Pour cela, il faut mettre en place un « PoC » (Proof of Concept) avec les différents outils. Ce PoC se doit d'être bien cadré et orienté afin de répondre aux nombreuses questions liées à l'automatisation. Ces questions, inspirées par l'étude théorique, peuvent être de ce type :

3.1- L'outil peut-il vraiment bien automatiser les tests dans mon environnement technique ?

Cette question est primordiale. Les besoins techniques ont été définis dès le début de l'étude de l'automatisation et font partie des filtres théoriques. Néanmoins, il peut arriver que l'outil ait

des limites au niveau de cette capacité qui, dans notre contexte, limite la possibilité d'automatiser certains tests. Afin de savoir si, dans les faits, l'outil répond à l'ensemble des besoins techniques, il est important de définir des tests à automatiser qui soient représentatifs de l'ensemble des tests qui devront être automatisés.

Afin de s'assurer de cette exhaustivité des capacités d'automatisation de nos tests par l'outil expérimenté il est possible d'établir une checklist qui pourra servir de comparaison entre les divers outils testés.

3.2- L'outil fait-il vraiment gagner du temps ?

Le gain de temps est très souvent la raison principale ayant conduit à automatiser. Il est donc primordial de s'assurer qu'à moyen terme, le fait d'avoir automatisé et de continuer à automatiser les tests coûte moins de temps que le fait de tout conserver en manuel. Pour cela, il est essentiel d'avoir des indicateurs fiables de mesure du temps.

Combien de temps nécessaire en test 100% manuel ?

Combien de temps nécessaire pour faire la même chose avec nos tests automatisés ?

Attention, le temps nécessaire ne comprend pas uniquement l'exécution mais aussi l'analyse, la maintenance, l'écriture…

3.3- L'outil fait-il économiser de l'argent ?

Cette question est très liée au temps et n'est pas forcément la préoccupation première de l'équipe. Il reste cependant nécessaire d'aborder le sujet. En plus du coût en temps, il faut prendre en compte le coût des licences, des formations et de la mise en place de l'infrastructure pour les tests.

Par rapport au gain d'argent, il reste très compliqué de mesurer les gains en qualité, capacité à détecter tôt des anomalies et donc réduire leur coût même s'il reste envisageable de penser à des études de productivité (ex : vélocité) pour avoir une idée.

3.4- Les tests automatisés avec l'outil contribuent-ils à améliorer la qualité de notre logiciel ?

Ce point est essentiel. Le rôle de l'automatisation n'est pas uniquement de gagner du temps mais aussi d'améliorer la qualité de l'application. Cette amélioration intervient de plusieurs manières, les principales étant, selon moi :

La capacité à tester plus souvent et détecter plus tôt des anomalies, ce qui permet de les corriger plus rapidement et plus souvent.

L'amélioration de la couverture des tests de régression et le dégagement de temps pour faire plus de tests de validation et surtout plus de tests exploratoires, en ayant une stratégie très axée sur les risques.

Il faut garder à l'esprit que la qualité peut être détériorée avec des tests mal écrits ou peu fiables. Il est donc essentiel de bien suivre cet aspect.

3.5- L'outil est-il accepté et adopté par l'équipe ?

Peut être le point le plus important ! Le test et le développement, c'est avant tout de l'humain ! Si l'outil suscite des réticences, son adoption ne sera jamais un succès.

3.6- Toute autre question en lien avec son contexte

Il est évidemment impossible d'être exhaustif sur les questions à se poser. L'important est de définir en amont du PoC les différents objectifs et de définir comment mesurer l'atteinte de ces objectifs à l'aide d'indicateurs. Il restera bien évidemment possible d'ajouter d'autres indicateurs ou contraintes tout au long de l'expérimentation.

4- La sélection de l'outil et son industrialisation

A la fin de l'expérimentation des différents outils, il est alors temps de choisir. Les différents choix peuvent être :

- La sélection d'un des outils expérimentés suite à un PoC concluant
- Le rejet de l'ensemble des outils expérimentés et la mise en place de nouveaux PoC sur d'autres outils
- Décider de développer un outil en interne
- L'abandon de l'automatisation

Le choix le plus fréquent est heureusement, quand l'étude préalable a été bien menée, celui de la sélection d'un outil testé. Attention, ne pas sélectionner un outil après un PoC ne signifie pas un échec du PoC, bien au contraire.

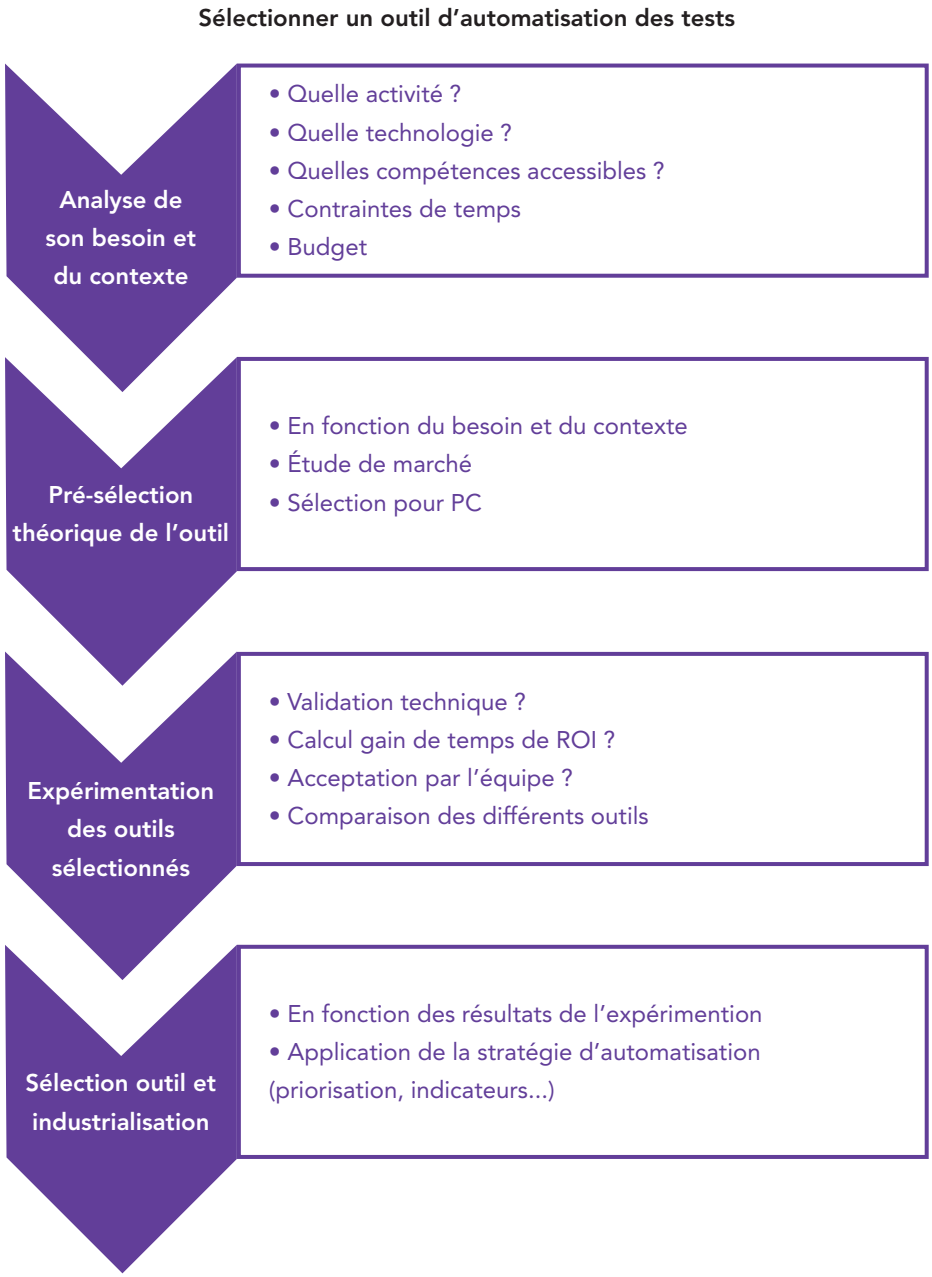
Dans le cas du choix d'un outil expérimenté, il est alors temps de mettre en place à plus grande échelle ce nouvel outil. Pour cela, il est nécessaire de former les personnes étant amenées à utiliser l'outil, de mettre en place un plan d'automatisation avec des objectifs concrets et des moyens définis.

Sélectionner son outil n'est que le début du chemin vers l'automatisation de l'exécution de ses tests !

5- Conclusion

La sélection d'un outil d'automatisation ne se fait pas du jour au lendemain. Les offres disponibles sont très nombreuses, tout comme les contextes de développement. Afin de s'assurer d'une sélection efficace, il est primordial de prendre le temps de faire une étude et une expérimentation. Ce temps investi au préalable augmente fortement les chances de succès du projet d'automatisation et fait finalement gagner beaucoup de temps.

On peut résumer le processus proposé dans cet article comme ceci :



II.5- Indicateurs liés à l'automatisation et la qualité de la solution

par Christophe Moustier

Les indicateurs donnent des valeurs qui permettent d'avoir un point de vue, tant sur une situation que sur sa projection, et l'automatisation facilite les calculs de ces KPI qui peuvent se retrouver à différents niveaux :

– La bonne qualité du produit et des services associés
– La bonne qualité du Delivery et le Time-to-Market (TTM) associé
– La situation quant à l'automatisation des actions manuelles, y compris les tests scriptés.

Chaque niveau influence la satisfaction du client et conforte l'idée que le test va bien au-delà de la simple exécution de scripts automatisés **(7) (8)**. Un tour d'horizon de ces niveaux va permettre d'y voir plus clair et d'en dégager une proposition « moderne » sur les indicateurs liés à la qualité de la Solution.

1- KPI sur le produit

Lorsqu'on parle d'automatisation dans le contexte du test, la qualité du produit est sans doute le premier élément qui vient à l'esprit.

1.1- Indicateurs du rapport de test

Que les indicateurs soient automatisés ou non, lorsque toute une batterie de tests a été lancée, il est évidemment confortable de disposer d'un indicateur qui vient synthétiser l'état de santé de la nouvelle version. La valeur calculée permet alors de savoir si le produit est apte à partir en production, que le déploiement soit automatisé par un pipeline d'intégration et déploiement continu, ou qu'il y ait un comité de releasing comme le CAB *(Change Advisory Board)* d'ITIL (1) pour donner son feu vert.
La difficulté d'un tel KPI est qu'il correspond à la synthèse souvent pondérée de plusieurs critères tels que :

– La valeur ajoutée métier intrinsèque des fonctionnalités testées – voir *diagramme de Kano* **(12) (23)**
– La valeur ajoutée métier contextuelle – une fonctionnalité attendue par un client en vue d'un événement qui lui est cher peut prendre bien plus d'importance qu'une autre fonctionnalité
– Les anomalies identifiées – l'industrie utilise par exemple la notion de « *démérite* », somme pondérée des bugs en fonction de leur sévérité **(23) (22)**
– La conformité à des critères de recevabilité liés aux standards de qualité de votre entreprise à partir de l'ISO 25010 ou d'exigences de certains clients.

On remarquera que le mode de mise à disposition d'une fonctionnalité peut influencer les critères précédents car le code peut être déployé, mais non activé, comme dans le cas du *Dark Launch* ou du *Canary Releasing* **(23)**.

1.2- KPI sur les hypothèses à vérifier

Ce genre d'indicateurs complémentaires à ceux déjà présentés est souvent réalisé à partir « d'observables » qui sont générés par les actions des utilisateurs sur le produit et les services associés. Cette approche fait partie de la stratégie « *Shift Right* » qui permet d'introduire du test après la mise en production. Ces KPI aident à vérifier non pas la conformité à des spécifications, mais l'adéquation au marché ciblé, notamment au travers

– De la perception du marché autour de la validation d'un MVP – voir *Lean Startup* **(30)**
– Des « *Leading Indicators* » associés aux épics chez SAFe **(31)**
– Des retours des clients sur les choix d'implémentation les plus appréciés avec le « *Test A/B* » **(23)**.

1.3- Indicateurs de satisfaction client/utilisateur

Une fois qu'un produit est capable d'adresser un marché, il s'agit de faire fructifier ce créneau en faisant en sorte que les clients soient satisfaits et que cette satisfaction soit contagieuse **(36)**. Parmi les indicateurs automatisables sur ces sujets, on peut observer notamment

– Le nombre d'incidents/problèmes et de plaintes des clients
– La rapidité d'intervention pour satisfaire le client sur les incidents - voir les notions de *MTTR / MTBF* **(23) (31)** et la section §2 de ce chapitre
– Les indicateurs de type « *Growth Hacking* » **(33)** tel que le « *AARRR* » **(32)**.

Certains de ces indicateurs donnent aussi lieu à des alertes qui favorisent la rapidité d'intervention. Certains observables donnent des métriques utilisées pour déclencher des actions suivant les seuils d'alertes atteints. Pour Google par exemple, il y a trois types d'alertes définies dans *SRE* **(31)** :

– Niveau 1. Pour les actions immédiates par un humain qu'un mécanisme de type *Jidoka* **(23)** permet de faire passer au niveau d'alerte niveau 2 pour action préventive
– Niveau 2. Pour les actions à accomplir par ajout d'un ticket dans un Backlog pour action correctives
– Niveau 3. Pour simple historisation de l'événement dans des logs pour analyse, la sécurité et éventuellement la mesure.

2- KPI sur l'efficacité du Delivery

Afin de générer la satisfaction du client, il est nécessaire de mesurer la capacité d'une organisation à fournir un produit et de nouvelles fonctionnalités dans les plus brefs délais, ce qu'on nomme le « *Flow* » (voir §2.3).

2.1- Mesure de qualité du code

La qualité du code aide à générer rapidement des solutions. Dans une organisation un tant soit peu outillée, on trouve généralement des moyens de mesure de *qualité statique du code* (ex. SonarQube, SQALE, CAST, Code Climate), voire des outils de mesure sur la sécurité du code avec des outils de type *SAST (Static Analysis Security Testing* – ex. suite Fortify, IAST, Veracode), d'intégration de composants vulnérables (suite Sonatype) ou même des outils capables de réaliser une analyse dynamique (on parle alors de DAST – ex. WebInspect, HCL AppScan). Malheureusement, suivant l'organisation, ces métriques arrivent a posteriori, ce qui induit un délai, voire une certaine négligence qui est accentuée par la pression des clients et du management.

Pour réduire cet effet, des outils doivent fournir un feedback aussitôt que possible. On trouve par exemple Linter qui donne le nombre et le niveau de criticité de non-conformités détectées dans l'éditeur de code ou les plugins de Fortify. Certains outils comme Promyze peuvent fournir des métriques par catégories de code *smell* **(2)**, avec un système de badges pour gamifier la réduction de la *dette technique* **(39) (23)**.

L'enjeu de la dette technique est qu'à force d'en introduire, on augmente le TTM de chaque US. C'est pourquoi Google propose un *Budget d'erreur* **(31)** qui s'amenuise à chaque mise en production de faible qualité et qui se regonfle par les actions de réduction de la dette comme le *refactoring*. Cependant, ces outils ne sont pas encore capables de percevoir certains *code smells* que seuls des développeurs correctement entraînés peuvent détecter, ce que permet Promyze. Ainsi, il est bon de tenter de mesurer cette dette technique (manuellement si c'est nécessaire), par exemple en faisant un atelier autour d'un « *Technical Debt Tree* » **(3)** ; les post-it reflétant les smells sont comptabilisés et pondérés suivant leur proximité avec le tronc. Peut-être qu'un jour, des outils basés sur de l'IA pourront automatiser ces calculs à partir du code **(37)** ?

2.2- Sur le pipeline de mise en production

Que la chaîne de traitement du code généré soit automatisée avec des outils tels que Jenkins ou Bamboo, ou qu'elle se fasse manuellement, il est pertinent de mesurer la rapidité de ce processus.

Sur ce point, la réalisation d'un *Value Stream Mapping* (VSM) est indispensable pour mesurer les délais d'attentes et réduire les plus élevés **(4)**, pourvu qu'une amélioration d'une partie ne nuise pas à l'ensemble, comme nous l'enseigne la *Théorie des Contraintes* **(5) (6)**. Ces mesures sont évidemment facilitées voire automatisables, par exemple sur la base des logs de la chaîne de traitement du code.

2.3- Indicateurs de productivité et de TTM

Une idée poussée par le *Lean* est formulée par la notion de « *Flow* », le flux de production **(34)**. Celui-ci doit être optimisé en vue d'un Delivery *soutenu mais soutenable* sur le long terme,

avec une *intégration de la qualité* au sein de l'organisation et les produits qu'elle génère **(9)**. Ainsi, la mesure du temps de déploiement d'un besoin est primordiale et en conséquence, le temps d'exécution des tests qui sont automatisés **(10)**.

Une mesure similaire est la fréquence de déploiement. Cette information est davantage orientée sur le *Flow* car elle reflète non seulement la rapidité de mise en production mais surtout le rythme auquelle se font les livraisons. Plus la fréquence est élevée, plus les utilisateurs voient les évolutions se faire et reconnaissent ainsi la capacité à satisfaire leurs besoins.

Cela dit, le *Flow* a une vision bien plus étendue que la rapidité du développement et mise en production : que dire d'un ticket en attente dans le Backlog depuis des mois que l'équipe doit livrer en 15 jours seulement ? Pour en prendre conscience, on pourra réaliser un VSM sur le processus de Delivery, depuis la création du ticket, jusqu'à son déploiement chez le client. De là, à chaque étape on peut mesurer **(4) :**

– La durée de traitement
– Le *Lead Time* (le délai d'attente cumulé à la durée de traitement)
– La durée d'attente avant traitement – à optimiser en premier
– Et le pourcentage de « *Complet & Correct* » (souvent noté « *%C&A* » - *pour Complete & Accurate*).

2.4- Indicateurs de satisfaction collaborateur

Comme on l'a vu plus haut, le *Flow* implique un Delivery durable sur le long terme. Pour cela, il est important de connaître « factuellement » le niveau de tension globale des employés, au-delà de souvenirs d'incidents marquants. Ainsi, une mesure sur la capacité de l'organisation à retenir ses employés (ex. avec le *Turnover*) peut donner une première valeur. Mais plutôt qu'attendre les effets néfastes pour avoir une mesure, on tentera de sonder le moral des troupes, notamment avec un *Niko Niko* **(11)** afin de prévenir le phénomène d'attrition.

3- KPI sur l'automatisation

Dans le cas où l'automatisation est particulièrement recherchée, sa stratégie de mise en œuvre peut se mesurer par des indicateurs qui peuvent adresser :

- La bonne qualité de ce qui est automatisé – pour mesurer ça, on peut par exemple dénombrer la quantité de scripts réputés fiables *vs* les autres statuts, en particulier ceux qui sont identifiés comme « faux positifs »
- L'efficacité du service rendu par l'automatisation – on peut se baser par exemple sur la quantité de nouveaux scripts par statut en n'oubliant pas de compter les tests qui ont déclenché des faux positifs et plus généralement les corrections à apporter
- La bonne marche de l'activité d'automatisation et sa rapidité – en mesurant par exemple le temps moyen pour automatiser un type de script, depuis son idéation jusqu'à son utilisation sur le SUT.

En voici quelques autres illustrations fournies par **(13)**

- « *Mean Time to Diagnosis* » *(MTD)* : durée moyenne de débogage d'un script qui échoue
- *Nombre de Bugs* trouvés par les scripts automatisés : permet de valoriser l'automatisation (voir les *Vanity Metric* en section §4)
- Le « *Flaky Rate* » : nombre de scripts qui déclenchent des *faux positifs* divisé par le nombre de scripts exécutés **(6) (15)**
- Le « *Ratio d'automatisation* » : c'est le rapport entre la quantité de scénarios automatisés par le nombre de scénarios total – ce KPI peut aussi se répartir suivant son statut, par exemple *manuel / en cours de développement / en test / qualifié / à corriger / ancien / à décommissionner.*

Ce à quoi on peut rajouter :

- La valeur ajoutée de l'automatisation avec son gain sur le TTM – il suffit, par exemple de compter l'estimation des valeurs ajoutées de fonctionnalités testées
- Le nombre d'exécutions d'un script pour évaluer sa péremption – plus ce nombre est élevé, plus le script a fait son temps, c'est le principe du *paradoxe du pesticide* **(14)**
- Le nombre de faux positifs déclenchés récemment par un script pour appliquer le *principe de regroupement des défauts* **(16)** et trouver d'autres faux positifs à proximité – il peut être utile de mettre ces scripts dans une zone de fiabilité grise mais non exécutés afin de limiter l'impact du *Flaky Rate*
- La quantité et la fréquence de remontée des bugs par le nombre de scripts automatisés : lorsque ces valeurs sont depuis trop longtemps à zéro, le décommissionnement du script commence à être envisageable. La valeur moyenne sur tous les scripts donne un niveau de pertinence du patrimoine de tests automatisés ; plus la valeur est grande, plus les scripts ont été efficaces à trouver des bugs
- La dette technique accumulée par les scripts comme pour la qualité du code : les tests automatisés (ou les RPA) sont aussi du code. Ils doivent être traités comme tels – voir §2.1.

4- Pièges des KPI

Pour faire un bon indicateur et évidemment l'automatiser, il faut :

1. * Définir les sources de données
2. * Définir les méthodes d'extraction, filtres et transformations appliquées pour exploiter les données
3. * Définir le calcul à effectuer sur ces données pour obtenir une valeur, l'indicateur
4. Définir l'interprétation de cet indicateur
5. Obtenir un consensus sur tout ça…

(*) : Les informations 1 à 3 permettent d'automatiser la génération d'un tableau de bord.

Par ailleurs, un KPI ne montre qu'un point sur une règle, jamais la chose que l'on veut mesurer. Il faut donc prendre des précautions et avoir conscience de phénomènes comme le *Paradoxe de Simpson* qui induit une fausse idée à la suite de valeurs sous-jacentes non observées **(18)**. On trouve aussi *l'effet thermostat* **(19)** où le système vient compenser les effets d'actions de changements comme dans *l'effet Hawthorne* **(20)**.

Cependant, comme nous le rappelle la *loi de Goodhart* : « *Lorsqu'une mesure devient un objectif, elle cesse d'être une bonne mesure* » **(37)**, ce qui rend la péremption des mesures un facteur important à prendre en considération : si on prend l'évolution du prix de la baguette depuis les années 30 ou celle de la valeur ajoutée perçue d'une fonctionnalité avant et après un événement, on peut bien comprendre qu'au-delà d'un certain horizon, l'évolution des valeurs modifie la portée d'un KPI.

Lorsqu'on regarde du côté de la *Théorie des contraintes* **(6) (5)**, on trouve des entreprises où le VSM est implémenté dans un ERP qui est malheureusement biaisé et ne reflète pas les bons goulots d'étranglements dans l'organisations **(17)**.
C'est pourquoi il est important que, face à des KPI, votre *Pensée Critique* **(21)** vous aide à bien garder à l'esprit ces dangers et prendre garde aux « *Vanity metrics* » **(22)**, ces KPI qui ne mesurent pas réellement le potentiel succès ou échec mais qui font juste plaisir...

5- Approche moderne pour générer des indicateurs

Pour faire face à toutes ces difficultés, une approche très linéaire pourrait être utilisée pour identifier les KPI à fournir d'après une déclinaison de la vision, la stratégie pour y parvenir, et des actions tactiques qui en découlent. Celles-ci donnent alors les valeurs recherchées comme dans un « *Hoshin Kanri* » **(29) (25)**.

On peut aussi prioriser ses besoins d'indicateurs au travers d'un Kanban comme les *Leadings Indicators* des épics chez SAFe. Les KPI sont alors déclinés en aval avec le support des architectes et complétés avec une analyse de risque, par exemple avec un 5M **(26)**.

De son côté, Google a choisi les « *OKR* » pour obtenir des métriques **(27)**. Ce système a le bon goût d'inclure l'état d'esprit *Kaïzen* **(6) (35)** qui comprend notamment la motivation intrinsèque des collaborateurs dans l'amélioration continue.

Quelle que soit votre approche, essayez d'inclure dans votre système de métriques des KPI alignés sur la rentabilité du business, mais aussi d'autres axes tels que le versant humain pour la soutenabilité du progrès comme le propose le *Balanced Scorecard* **(28)**.

6- Tableau de bord

Au-delà de la variété, des difficultés et des pièges, la présentation des KPI est tout aussi capitale, notamment

– Dans l'affichage impliqué par l'ordre de lecture culturel (gauche-droite et du haut vers le bas) qui va pousser la répartition chaque indicateur en fonction de leur valeur ajoutée

– Dans le style d'affichage (de gros chiffres sont plus visibles quand ceux de moindre importance sont représentés plus petits) et les couleurs (par exemple l'usage classique d'un dégradé de rouge ou de vert pour signifier des alertes)

– Dans la forme des indicateurs.

Sur ce dernier aspect, on notera les principes de la théorie de la *Gestalt*, théorie des formes dont les lois permettent de comprendre les pièges visuels dans lesquels tout lecteur peut tomber **(40)**.

7- Pour aller plus loin…

Avant de retrouver les références bibliographiques citées plus haut, n'oubliez pas qu'un indicateur peut revêtir différents aspects tels que l'évidente moyenne ou la moyenne pondérée ; les différences au N-1 ; les écarts types pour estimer les variations à la moyenne ; la croissance (la dérivée) ; l'accélération (la dérivée seconde) ; etc.

Vous noterez que ces deux derniers indicateurs sont intéressants pour voir les subtiles évolutions et encourager la progression.

---

(1) *ITIL Change Management & the CAB.* https://www.bmc.com/blogs/itil-change-advisory-board-cab/

(2) Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, d. (2002). *Refactoring: Improving the Design of Existing Code. Pearson Education*

(3) Klostermann, A. (2018). *Visualising and Prioritizing Technical Debt.* https://medium.com/@AikoPath/visualising-and-prioritizing-technical-debt-afc82e542681

(4) *Continuous Delivery Pipeline.* https://www.scaledagileframework.com/continuous-delivery-pipeline/

(5) Goldratt, E.M & Cox, Jeff (2004). *The Goal: A Process of Ongoing Improvement.* Routledge

(6) Moustier, C. (2020). *La conduite de tests agiles pour SAFe et LeSS.* ENI Editions

(7) Bach, J. (2014). Test Cases are Not Testing: Toward a Performance Culture. *CAST 2014 Keynote.* https://www.youtube.com/watch?v=JLVP_Z5AoyM

(8) Bach, J., & Bolton, M. (2016). *A Context-Driven Approach to Automation in Testing.* https://www.satisfice.com/download/a-context-driven-approach-to-automation-in-testing

(9) Scaled Agile Framework. (c2019). *Lean-Agile Mindset.* https://www.scaledagileframework.com/lean-agile-mindset/

(10) Bouhier, L. (2018). *Mais c'est quoi un test unitaire ?* https://latavernedutesteur.fr/2018/04/11/mais-cest-quoi-un-test-unitaire/

(11) *Niko-niko.* http://referentiel.institut-agile.fr/nikoniko.html

(12) *Chapitre VII : Le diagramme de Kano.* https://www.e-marketing.fr/Thematique/academie-1078/fiche-outils-10154/Le-diagramme-de-Kano-325762.htm

(13) Colantonio, J. (2018). *The Ultimate Guide to Automation Testing.* https://testguild.com/wp-content/uploads/2018/07/UltimateGuild-ToAutomation.pdf

(14) Hasmaz, H. (2020). *Exemple concret du paradoxe des pesticides.* https://latavernedutesteur.fr/2020/03/03/exemple-concret-du-paradoxe-des-pesticides-hasnae-hasmaz/

(15) Philipp, I. (2018). *How to Reduce False Positives in Software Testing.* https://www.tricentis.com/wp-content/uploads/2019/01/How-to-Reduce-False-Positives-in-Software-Testing-white-paper.pdf

(16) RADID, A. (2018). *Principe 4 – Regroupement des défauts.* https://latavernedutesteur.fr/2018/01/12/principe-4-regroupement-des-defauts/

(17) Marris, P. (2015). *How to identify bottlenecks in production and projects.* https://www.youtube.com/watch?v=ulXqO86OfpU

(18) Louapre, D. (2015). *Le paradoxe de Simpson.* https://www.youtube.com/watch?v=vs_Zzf_vL2I

(19) Friedman, M. (2003). *The Fed's Thermostat.* https://www.researchgate.net/publication/265032536_The_Fed%27s_Thermostat_The_Fed%27s_Thermostat

(20) *Effet Hawthorne.* https://fr.wikipedia.org/wiki/Effet_Hawthorne

(21) *Critical thinking.* https://en.wikipedia.org/wiki/Critical_thinking

(22) Ries, E. (2011). *The Lean Startup - How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses.* Crown Business

(23) Moustier, C. (2019). *Le test en mode agile*. ENI Editions
(24) LearnFirm.com. (2014). Hoshin Kanri - Visual Strategic Planning. https://fr.scribd.com/document/384144058/Hoshin-Kanri-Visual-Strategic-Planning-Student-Workbook
(25) Pauchard, E. (2016, 05). *Hoshin Kanri: quel est votre plan ?* http://www.eponine-pauchard.com/2016/05/hoshin-kanri/
(26) *5M model*. https://en.wikipedia.org/wiki/5M_model
(27) Doerr, J. (2019). *Measure What Matters: How Google, Bono, and the Gates Foundation Rock The World with OKRs*. Penguin Random House.
(28) *Balanced scorecard*. https://en.wikipedia.org/wiki/Balanced_scorecard
(29) Ferignac, P. (1959). *Inspection des produits finis par la méthode des indices de qualité ou démérités*. http://www.numdam.org/item/RSA_1959__7_3_27_0/
(30) Vallone, J. (c2020). *Applied Innovation Accounting in SAFe*. https://www.scaledagileframework.com/guidance-applied-innovation-accounting-in-safe/
(31) Beyer, B. & Jones, C. & Petoff, J. & Murphy, N.R. (2016). *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly
(32) Gastaud, P. (2018). *Évaluez le parcours d'achat de vos clients avec la méthode AARRR*. https://www.e-strategic.fr/blog/methode-aarrr/
(33) Fong , R., & Riddersen, C. (2016). *Growth Hacking: Silicon Valley's Best Kept Secret*. Lioncrest Publishing
(34) Reinertsen, Donald G. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas
(35) Maurer, Robert (2014). *One Small Step Can Change Your Life: The Kaizen Way*. Workman Publishing
(36) Eyal, Nir (2018). *Hooked : comment créer un produit ou un service qui ancre des habitudes : le best-seller international*. Editions Eyrolles
(37) Bolkensteyn, B. (2018). *Implementing Machine Learning in SonarQube*. https://community.sonarsource.com/t/implementing-machine-learning-in-sonarqube/1066
(38) *Loi de Goodhart*. https://fr.wikipedia.org/wiki/Loi_de_Goodhart
(39) Cunningham, Ward (1992). *The WhyCash Portfolio Management SystemOOPSLA'92 experience report*. http://c2.com/doc/oopsla92.html
(40) Saint-Martin, Fernande (1990). *La théorie de la gestalt et l'art visuel*. Presses de l'Université du Québec

II.6- Tests exploratoires assistés par l'IA

par Xavier Blanc, Julien Leveau et Frédéric Assante Di Capillo

Le concept de test exploratoire a été présenté en 1984 par Cem Kaner dans l'objectif de proposer une approche complémentaire aux tests scriptés. Pendant des années, cette approche est restée confidentielle, très peu utilisée et parfois même dénigrée car assimilée à du test ad hoc sans aucun fondement méthodologique, ce qui est évidement faux. Aujourd'hui, il faut noter qu'elle revient sur le devant de la scène avec de nombreux retours sur expérience montrant les avantages apportés.

Nous présentons ici les principes structurants du test exploratoire « conception, exécution et analyse en continu » qui permettent de bien comprendre les différences avec le test scripté. Ces principes laissent une grande part de liberté aux testeurs, ce qui explique le lien entre le niveau d'expertise des personnes qui réalisent les tests exploratoires et les résultats obtenus. La clarification de ces principes permet de mieux comprendre pourquoi le test exploratoire peut être assimilé à du test d'expert, sans que cela soit pour autant intégré à ses principes.

Nous expliquons ensuite comment mettre en œuvre le test exploratoire dans un contexte de développement logiciel, et plus particulièrement lorsque l'agilité est mise au centre de l'organisation. La notion de session de test exploratoire devient alors centrale car elle délimite les efforts à apporter, que cela soit dans le temps ou dans l'espace.

Enfin, nous montrons comment optimiser l'application du test exploratoire grâce à des environnements outillés, supports à la réalisation de sessions. Nous expliquons plus précisément comment la plateforme open source AIFEX facilite la collaboration et maximise les résultats obtenus.

1- Principes structurants

Si le concept de test exploratoire a été proposé en 1984 par C. Kaner [Kaner 1984], la définition que nous retenons est celle proposée par le SWEBOOK dans l'édition de 2004 :

"Exploratory testing is simultaneous learning, test design, and test execution." [IEEE 2004]

Cette définition pose les principes du test exploratoire. Celui-ci vise à réaliser simultanément la conception des tests, l'exécution des tests et l'analyse et la compréhension immédiate des résultats pour poursuivre le test de l'application. Sur ces points, les principes du test exploratoire sont à l'opposé de ceux du test scripté qui, lui, propose de démarrer par la conception, puis, lorsque la conception est terminée, d'exécuter les tests. De plus, le test scripté ne fait pas d'analyse et de compréhension immédiate des résultats (par immédiate, nous entendons après l'exécution de chaque test).

Pour illustrer ces différences fondamentales, nous proposons une schématisation abstraite centrée sur le concept de matrice de couverture. Nous estimons qu'une matrice de couverture représente les objectifs de test dans les colonnes, les tests dans les lignes et identifie si le test couvre l'objectif ou pas. On peut alors considérer que le but d'une approche de test est de réaliser la matrice la plus pertinente et ce, de la façon la plus efficace qui soit.

La figure 1 présente l'approche du test scripté (manuel ou automatique) par rapport à cette notion de matrice de couverture. La première étape est la conception de la matrice, c'est-à-dire l'identification des objectifs de tests, puis la définition des tests qui couvrent ces objectifs. Cette étape s'achève lorsque la structure de la matrice est complètement terminée. On peut imaginer par exemple que les objectifs sont des exigences de l'application et que les tests visent à valider ces exigences. La deuxième étape consiste à exécuter les tests et, ainsi, indiquer si les tests sont passés ou pas. L'exécution peut être manuelle ou automatisée, elle peut être effectuée régulièrement ou plusieurs fois par semaine. Le test scripté n'inclut pas d'analyse et de compréhension immédiate (après l'exécution d'un test).

La Figure 1 illustre les deux étapes du test scripté. On comprend alors l'importance de la conception qui va définir les objectifs (fonctionnels ou non fonctionnels) et les tests à réaliser ainsi que l'ordre dans lequel les exécuter. Cette étape a un coût non négligeable et aucun test ne peut être exécuté tant qu'elle n'a pas été réalisée complètement. L'étape suivante, l'exécution, peut être optimisée en temps, en fonction des moyens qui y sont affectés.

	Objectif 1	Objectif i	Objectif N
Test 1			
Test 2			
Test i			
Test N			

1 Conception

	Objectif 1	Objectif i	Objectif N
Test 1	Ok		
Test 2	Ok	NOk	
Test i		NOk	NOk
Test N	Ok		Ok

2 Exécution

Figure 1 : Test Scripté et matrice de couverture

Les tests exploratoires sont, sur ces principes, fondamentalement différents : trois étapes (conception, exécution et analyse immédiate) sont réalisées simultanément et les tests se font de manière incrémentale et itérative. La figure 2 illustre cela en montrant un explorateur[1] qui fait un premier test, puis un deuxième, etc. L'explorateur est guidé par les tests successifs qu'il réalise. Il peut, à la lumière de son analyse immédiate, se fixer de nouveaux objectifs et/ou réaliser de nouveaux tests. La démarche est bien itérative et incrémentale. A chaque nouvelle itération, l'explorateur réalise les trois étapes : conception, exécution et analyse immédiate.

La figure 2 montre aussi la difficulté de l'approche car les résultats reposent sur l'explorateur. Charge à lui de définir les objectifs de test, d'exécuter les tests et de faire une analyse immédiate qui lui permettra d'itérer à nouveau et de faire un nouvel incrément. Enfin, la figure 2 pose aussi le problème de l'arrêt du test exploratoire. En effet, aucune condition d'arrêt n'est définie, contrairement à l'approche scriptée qui précise cela dans la conception. Avec le test exploratoire, il faut savoir quand continuer ou quand arrêter, et ce potentiellement à chaque itération / incrément. Pour répondre à ce problème, le test exploratoire utilise le concept de session, qui permet de définir la charte de test.

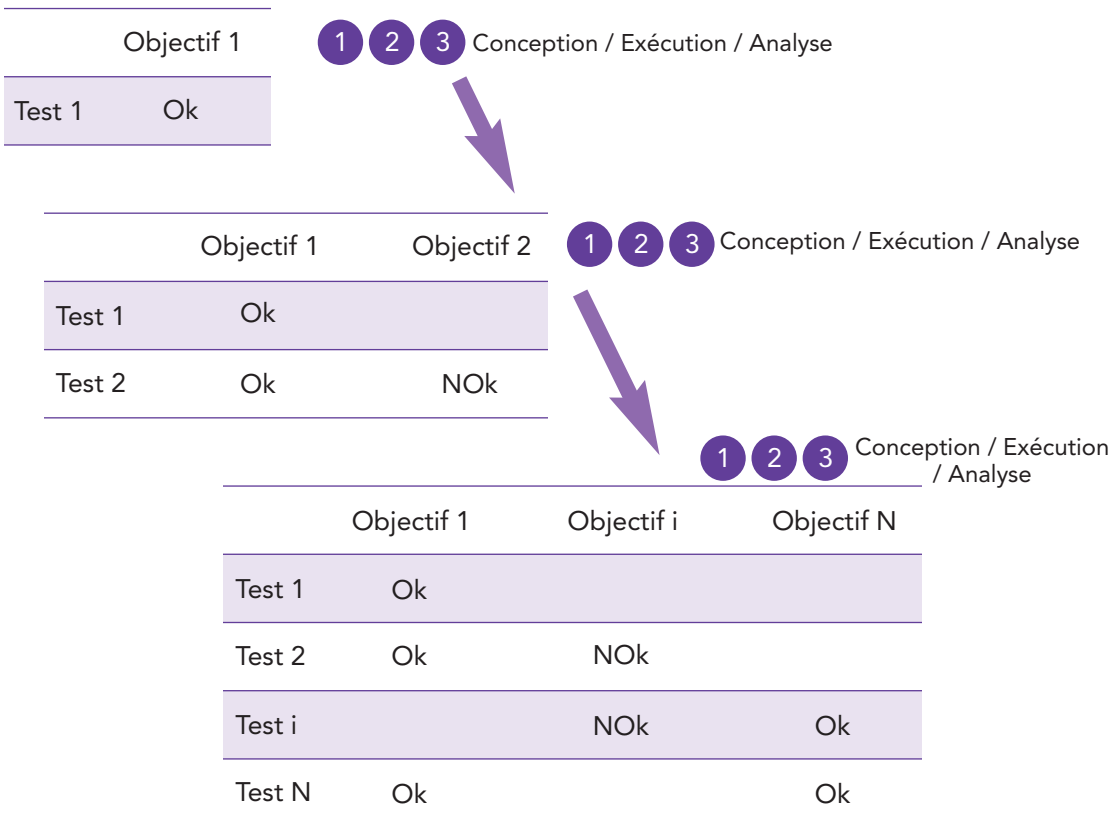


Figure 2 : Test Exploratoire et matrice de couverture

[1] Nous utilisons le terme explorateur pour identifier la personne qui va réaliser des tests exploratoires. Celui-ci peut être un testeur, un développeur, un utilisateur expert ou novice, etc.

2- Mise en Œuvre : Session et Charte

Au-delà des principes, le test exploratoire est articulé par le concept de session.

"A session is an uninterrupted block of reviewable, chartered test effort" [Bach 2000].

Une session rassemble plusieurs explorateurs (il est possible de faire des sessions solo) qui collaborent pour explorer l'application sous test. Une session a un début et une fin. Dans sa définition, J. Bach insiste sur le côté « sans interruption » de la session. Les sessions de test exploratoire sont en effet bien plus efficaces lorsque les explorateurs ne sont pas interrompus. Cela leur permet de rester concentrés et ainsi, de réaliser des explorations à forte valeur ajoutée.

La session définit et cadre les ambitions ainsi que les efforts affectés au test exploratoire. Pour ce faire, il faut définir une **charte** qui précise a minima les points suivants :

- Le périmètre : On s'attachera à préciser les éléments à tester (ainsi que ceux à ne pas tester). Le périmètre est souvent catégorisé par la métaphore du « tour operator » (les lieux touristiques incontournables, les grandes avenues avec beaucoup de passage, les nouveaux monuments, les petites rues, tel ou tel quartier, etc.).
- Le but général : On s'attachera à préciser le but général de la session. S'agit-il de vérifier les aspects graphiques, les aspects fonctionnels, la performance, toutes les anomalies, etc. ?
- L'organisation et le déroulement : Il faut préciser qui seront les explorateurs et quel temps leur sera alloué. Enfin, il faut aussi préciser comment seront réalisées les échanges entre les explorateurs et comment seront faits le rendu et la synthèse de la session.

Ces questions doivent être traitées avant de démarrer une session de test exploratoire, sans cela il n'y a que peu de chance que les résultats soient au rendez-vous.

Dans un contexte agile, plusieurs sessions de test exploratoire peuvent être envisagées. Nous dressons ici une liste non exhaustive des différents types de session :

- **Solo** : Cette session est très légère. Elle est réalisée par un explorateur qui va se fixer un petit périmètre ainsi qu'un but très précis. Il s'agira, par exemple, de tester une nouvelle fonctionnalité qui vient juste d'être intégrée dans le serveur de test. L'explorateur va alors organiser une session courte (de 15 min à moins d'une heure) et va réaliser plusieurs explorations. Le rapport qu'il fournira permettra alors d'améliorer grandement la qualité et ne nécessitera pas un effort trop important.

- **User Story** : Cette session se déroule lors d'un sprint dès qu'une user story a été réalisée. Elle rassemble les membres de l'équipe. Son périmètre et son but portent sur la user story. Les membres de l'équipe vont collaborer pendant une session relativement courte (1h ou plus) et tester de manière exploratoire les développements réalisés. Un point important est que ce genre de session permet aussi de réaliser un échange de connaissance entre les membres de l'équipe.

- **SmokeLike** : Cette session sert à lever toutes les anomalies qui n'auraient pas été détectées par les tests scriptés. Le but est ici de vérifier qu'aucun bug important n'est passé sous le radar. Ce genre de session prévient les anomalies grossières qui malheureusement arrivent parfois et sont détectées quelques secondes après la mise en production.

- **Release** : Cette session vise à revisiter une partie de l'application. Elle est souvent réalisée par des équipes à forte expertise, qui ont une vision plus long terme de l'application, et peut même intégrer des utilisateurs finaux. Elle se déroule sur une journée ou plus. Le but est de remettre en question certains aspects de l'application afin de procéder à des évolutions importantes. C'est aussi l'occasion de se rendre compte d'anomalies qui n'ont pas été détectées.

La liste présentée ici n'est pas exhaustive, elle permet de bien mesurer les différentes dimensions des sessions et ainsi de faciliter leur préparation afin d'optimiser le rapport temps / bénéfices.

3- Support outillé

La force du test exploratoire réside dans les explorations réalisées par les explorateurs et donc dans les collaborations qu'ils réalisent durant les sessions, l'objectif étant de savoir ce que fait chaque explorateur, de comprendre les explorations réalisées et ainsi de partager les connaissances obtenues.

Pour faire levier sur l'expertise des explorateurs et améliorer la collaboration, nous avons réalisé une plateforme Open Source, support à l'organisation des sessions et qui assiste les explorateurs dans leurs explorations à l'aide d'une intelligence artificielle.
Notre plateforme AIFEX (Artificial Intelligence For Exploratory Testing) propose les services suivants :

- L'organisation de sessions de test exploratoire sur des applications web.
- L'enregistrement des explorations réalisées par les explorateurs durant le déroulement de sessions.
- L'apprentissage des explorations par une IA afin de guider les choix qu'ils pourraient faire pour réaliser leurs explorations futures.
- La présentation d'une synthèse des explorations réalisées.

La plateforme AIFEX est composée d'un serveur qui va gérer les sessions et d'un assistant installé sur le poste de l'explorateur (sous forme d'une extension de navigateur Chrome ou Firefox). La Figure 3 présente le serveur AIFEX qui offre une interface graphique à l'organisateur de la session et qui permet à trois explorateurs de collaborer. L'organisateur d'une session utilise le serveur pour préparer sa session. Dès que celle-ci est prête, il peut inviter les explorateurs à venir faire des explorations. Chaque explorateur aura son assistant qui dispose d'une IA et qui va le guider dans les actions à faire.

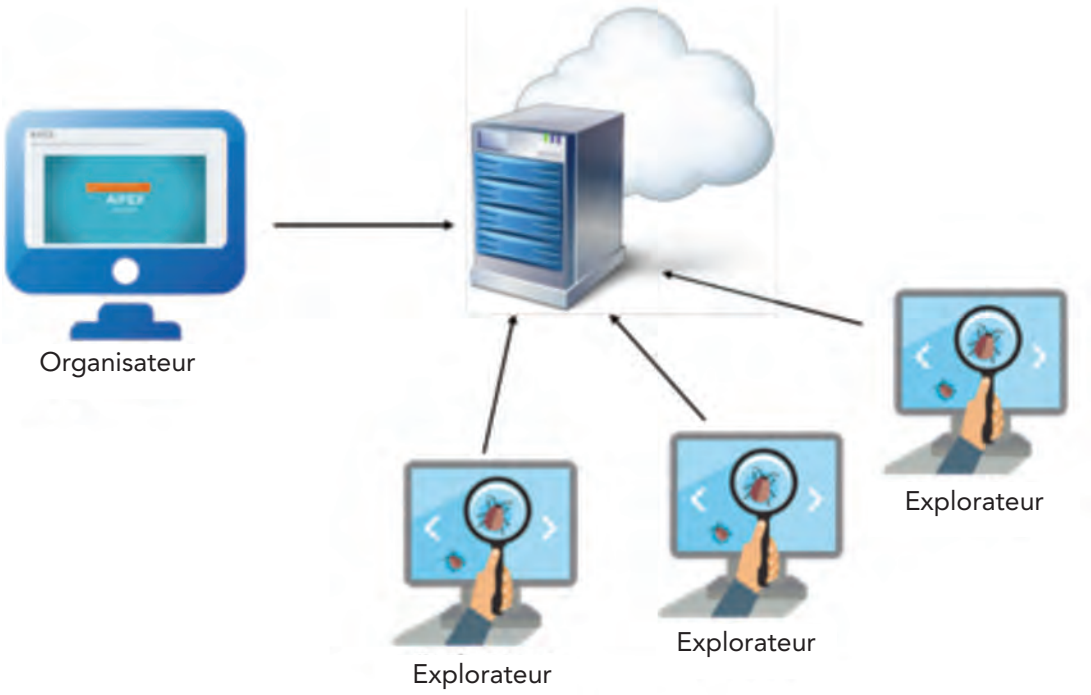


Figure 3 : AIFEX - Plateforme Open Source de Test Exploratoire

L'organisation de sessions de test exploratoire sur des applications web

La création d'une session se fait en précisant sur quelle application web se déroulera la session (on donne l'URL de l'application sous test) et en configurant les moyens d'enregistrement des actions réalisées (plusieurs configurations sont possibles, de la plus simple à la plus riche).
La figure 4 présente la page AIFEX permettant de construire une nouvelle session de test exploratoire. On voit qu'on peut donner un nom à une session et qu'il faut préciser sur quelle application web celle-ci va se dérouler.

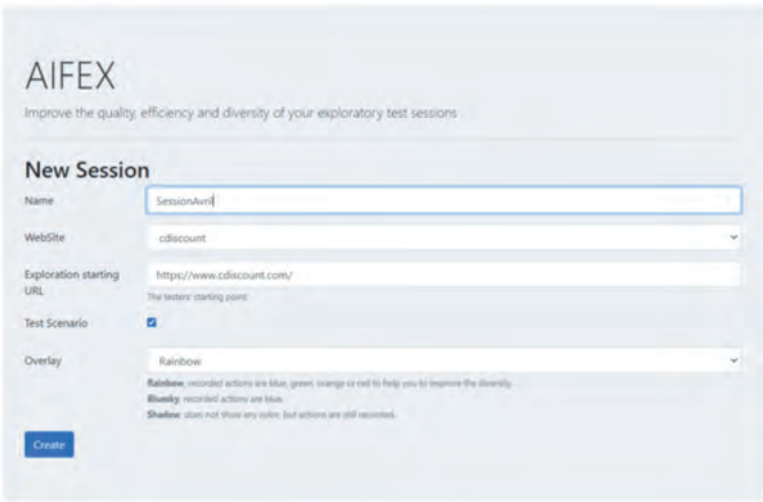


Figure 4 : Création d'une session avec AIFEX

Dès qu'une session est créée, AIFEX génère une URL propre à la session qui permet à n'importe quel explorateur de pouvoir rejoindre la session. Il faut pour cela qu'il ait installé l'assistant (extension Chrome ou Firefox) sur son poste de travail.

L'enregistrement des explorations réalisées

Les explorateurs peuvent demander à leur assistant d'enregistrer les actions qu'ils vont réaliser, de prendre des photos de l'écran, de faire un enregistrement vidéo et ils peuvent même ajouter des commentaires.
La Figure 5 présente l'assistant sur une session de test exploratoire. On voit que le menu propose de démarrer l'enregistrement, de le stopper, d'ajouter des commentaires, etc.

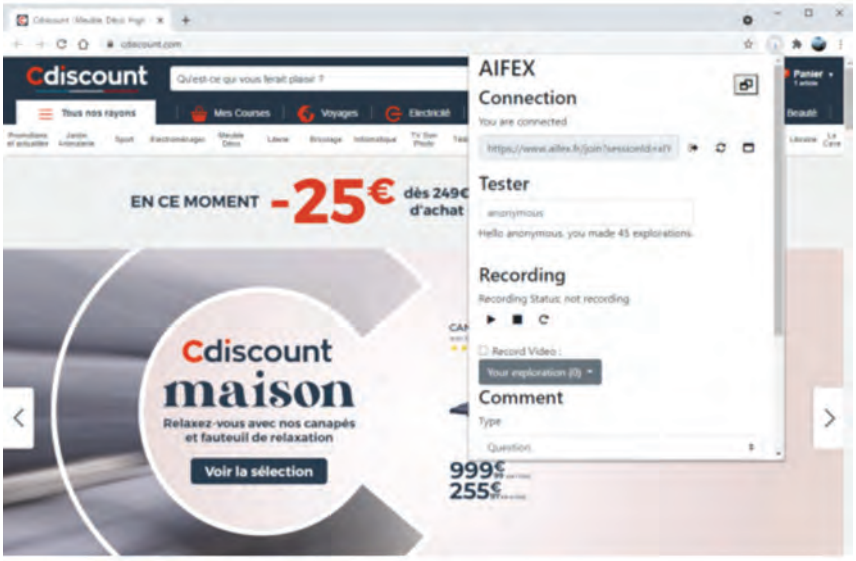


Figure 5 : L'assistant AIFEX qui propose d'enregistrer les explorations

L'apprentissage des explorations par une IA

Dès lors qu'un explorateur démarre l'enregistrement de son exploration, l'assistant lui indique les parties de l'application qui entrent dans le contexte de la session. Ces parties sont encadrées afin de bien les identifier. De plus, l'IA de l'assistant utilise le code couleur d'une carte de chaleur pour indiquer quels éléments ont déjà été testés dans la session. La figure 6 présente l'assistant. On y voit des zones en bleu indiquant des éléments « froids » qui n'ont jamais été testés. Certaines zones en vert ont été un peu plus testées. Si des zones avaient été marquées en orange ou en rouge (couleurs chaudes), cela voudrait dire qu'elles ont été beaucoup plus testées.
L'assistant AIFEX peut aussi indiquer à l'explorateur les différentes données qui ont été saisies lors de tests précédents dans la même session. La figure 6 présente les différentes recherches qui ont été saisies dans cette session, qui cible un site de e-Commerce.

Grâce à ces indications, l'explorateur partage alors avec ses collaborateurs les informations qu'il a pu obtenir lors de ses tests (il est aussi possible de partager des commentaires – cela n'est pas montré dans la figure).

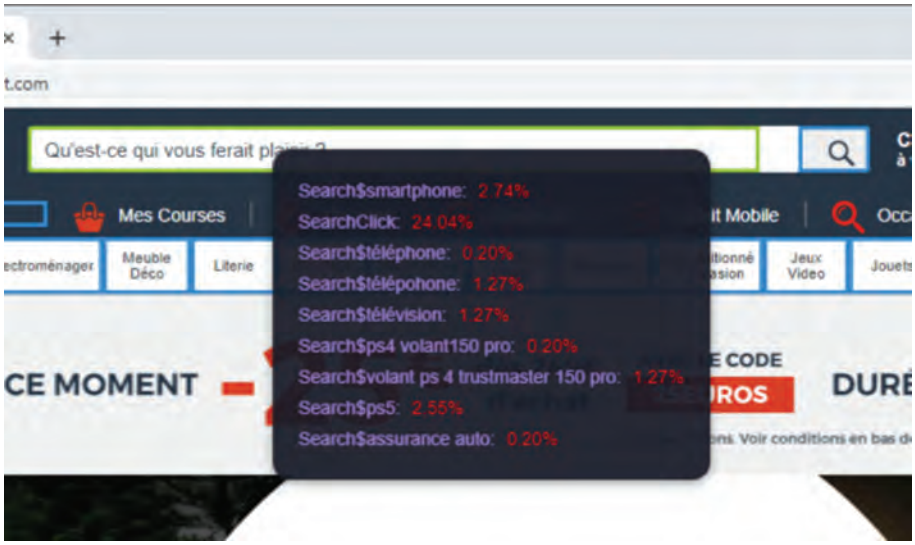


Figure 6 : l'assistant AIFEX pointant les zones à tester

La présentation d'une synthèse des explorations réalisées

La plateforme AIFEX permet à l'organisateur de la session de voir la synthèse des explorations réalisées. La figure 7 présente l'écran d'accueil des vues de synthèse. Il est en effet possible d'avoir la liste complète des explorations réalisées, d'avoir accès aux photos et aux vidéos, de voir tous les commentaires qui ont été saisis.
Il est aussi possible de visualiser l'analyser faite par l'IA et ainsi, de comprendre quelles explorations ont été faites souvent et quelles sont celles qui ont été faites plus rarement.

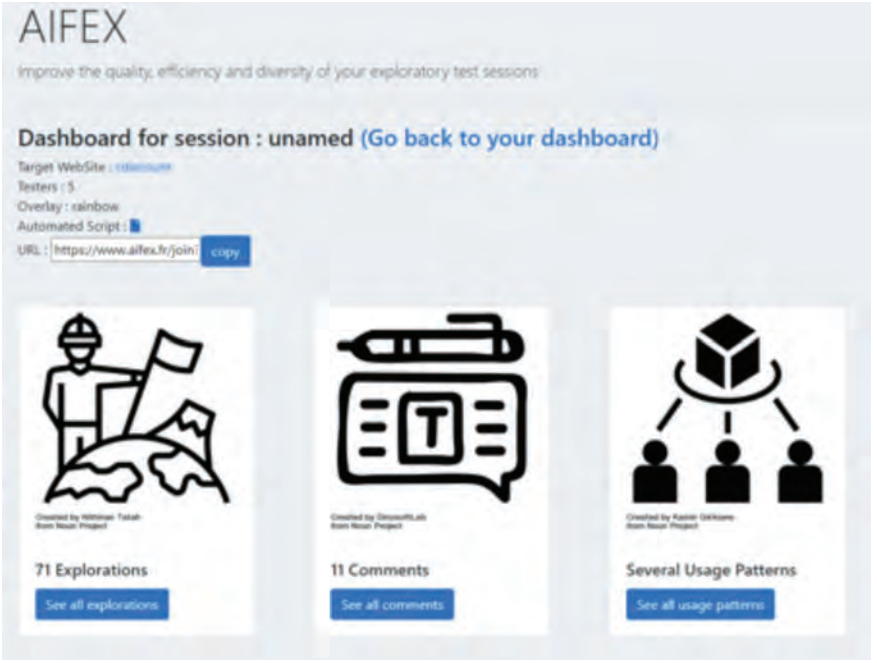


Figure 7 : Synthèse d'une session

4- L'expérience du test exploratoire chez Amadeus

Lors du passage aux méthodologies Agiles, la question des phases de validations s'est bien entendu posée. L'organisation en Sprints a entraîné l'obligation de tester de plus en plus souvent et par phases de plus en plus courtes. Une automatisation accrue des scripts de tests a aussi été mise en place pour augmenter encore plus la rapidité des phases de validation (notamment dans un contexte CI/CD).

Mais tous ces changements ont entraîné, intrinsèquement, un changement dans les méthodes de validation et une focalisation sur des incréments de code de plus en plus petits. Cet état de fait peut aboutir à une perte de la vision globale de l'application sous qualification et un risque de dégradation de la qualité du produit qui sera délivré en production.

Un des éléments pouvant permettre de limiter cette perte de vision globale est donc la mise en place de sessions de Tests Exploratoires. Dans notre division, nous avons pu mettre en place cette activité sur une journée complète (la recommandation pour ce genre de session est entre une demi-journée et deux jours). La session a été organisée selon la méthode décrite dans le précédent livre du CFTL [CFTL 2019], avec ses Chartes ; un groupe d'Explorateurs comprenant des développeurs, des responsables des spécifications, des qualiticiens mais aussi des utilisateurs ; et des résultats (rapports et retour d'expériences). Les Chartes ont été distribuées à des binômes n'ayant pas les mêmes profils (QA-DEV, Users-Spec, QA-Spec, etc.).

En fin de session, une Rétrospective a eu lieu pour récolter des retours d'expérience et des voies d'amélioration (points positifs, points négatifs, axes d'amélioration). Les résultats ont été très intéressants, nous avons pu mettre en avant les points suivants :

- Très bonne convivialité : Le fonctionnement en binôme, avec des profils différents, permet des échanges très enrichissants et une meilleure compréhension des problématiques de chacun. La communication directe et le travail en commun a aussi permis de créer des liens entre des entités ne se parlant que très rarement de façon direct.

- Meilleure compréhension des fonctionnalités : Le travail en binôme, sur des zones de code déterminées par les Chartes, permet de mieux comprendre les objectifs réels des fonctionnalités (notamment la vision utilisateur).

- Découverte d'erreurs non trouvées précédemment : Plus de 80 « bugs » ont été découverts par les 5 binômes en une journée (ce qui est considérable sur une application déjà mature). La définition de ces erreurs a aussi été améliorée par la proximité avec les développeurs. Il est à noter que toutes ces erreurs ont été valides (aucun rejet).

- Rapidité des résolutions : La proximité avec les développeurs a permis de résoudre, le jour même, près de 10 erreurs et environ 50 autres ont été résolues dans la semaine suivante, contre une moyenne de plus de 20 jours normalement.

• Journée très sympathique : Le petit-déjeuner de début de session, ainsi que le repas en commun et le « goûter » de l'après-midi ont permis de rendre cette journée très conviviale et a aussi permis des contacts hors travail pur, entraînant une cohésion des explorateurs sur le long terme.

Les points à améliorer :

• Un nettoyage préventif du code : Le « backlog » d'erreurs était encore trop important avant la session de Test Exploratoire. De ce fait, certaines erreurs détectées existaient déjà. Heureusement qu'un Référent Qualité a empêché que ces erreurs soient « loguées » pour éviter les doublons, donc les rejets.

• Capitalisation des tests : Beaucoup de nouveaux scénarios ont émergé de cette session, notamment par les binômes comportant un utilisateur. Malheureusement, il n'était pas prévu de mode d'enregistrement pour pouvoir capitaliser sur ces scénarios. La prise de notes a permis de ne pas tout « perdre » mais cela a nécessité un temps assez long pour la mise à niveau de la campagne de régression automatique.

• Meilleure communication entre les binômes : Même si les Chartes permettaient d'éviter les superpositions de scénarios, nous nous sommes rendu compte que pour accéder à certaines fonctionnalités, les Explorateurs doivent passer par des chemins similaires. Mais, ne sachant pas les scénarios des autres binômes (Les Chartes ne donnant pas, par définition, de consignes précises sur les chemins à parcourir), nous nous retrouvions avec des scénarios explorant les mêmes chemins. C'est là qu'un outil comme AIFEX aurait été d'une grande utilité.

Comme nous l'avons vu précédemment, deux des axes majeurs d'amélioration consistaient à mieux utiliser les scénarios imaginés par les Explorateurs.

Nous avons donc décidé de préparer une nouvelle session de Test Exploratoire en utilisant AIFEX pour optimiser notre session. Le rôle de cet outil aurait été double, éviter les doublons dans les scénarios (ou, du moins, les limiter grandement) et avoir une bibliothèque des chemins parcourus pour pouvoir créer des scripts automatiques à rajouter à notre campagne de régression. Nous avons donc créé notre matrice permettant de définir notre application (donc les éléments à suivre dans l'application), ainsi que de nouvelles Chartes pour explorer d'autres parties du code.

Malheureusement, la pandémie de Covid ne nous a pas permis d'organiser cette session (une session de Test Exploratoire est pleinement efficace quand les Explorateurs sont regroupés dans une même salle), nous avons hâte de pouvoir nous retrouver pour mettre en « musique » notre nouvelle session !

Conclusion

Le test exploratoire est une approche complémentaire au test scripté qui offre de nombreux avantages.

Premièrement, il a été montré qu'il permet d'identifier des anomalies qui n'ont pas été trouvées par le test scripté [Itkonen 2007]. En effet, les sessions de test exploratoire couvrent de nombreux aspects des applications, qui souvent ne sont pas bien couverts par les tests scriptés. De plus, l'expertise des explorateurs fait qu'ils remontent des anomalies difficiles à anticiper.

Le deuxième avantage du test exploratoire (et surtout des sessions) est qu'il permet une montée en connaissance des explorateurs sur l'application. En effet, les sessions permettent aux explorateurs d'échanger leurs connaissances et ainsi d'apprendre de nouvelles choses. Cet apprentissage permet de meilleurs gains quant à la qualité future des applications.

Enfin, les expérimentations que nous avons réalisées montrent que l'outillage est nécessaire et qu'il maximise la diversité des explorations. En effet, un explorateur qui dispose d'un assistant effectue des explorations qui ont une diversité supérieure de 30% par rapport à des explorations qui auraient été réalisées sans assistant [Leveau 2020].

Bibliographie

[Bach 2000] Session-Based Test Management, Software Testing and Quality Engineering Magazine, November 2000
[Kaner 1984] http://www.kaner.com/pdfs/QAIExploring.pdf
[IEEE 2004] IEEE, "Guide to the Software Engineering Body of Knowledge", IEEE., Tech. Rep. IEEE - 2004 Version, 2004.
[Itkonen 2007] Defect Detection Efficiency: Test Case Based vs. Exploratory Testing, J. Itkonen, M.V. Lassenius, ESEM 2007
[Leveau 2020] Julien Leveau, Xavier Blanc, Laurent Réveillère, Jean-Rémy Falleri, Romain Rouvoy: Fostering the Diversity of Exploratory Testing in Web Applications. ICST 2020: 164-174
[CFTL 2019] Olivier Denoo, Marc Hage Chahine, Bruno Legeard, Eric Riou du Cosquet: Les tests logiciels en Agile.

II.7- Robotiser les tests de régression fonctionnelle à partir des traces d'usage

par Julien Botella et Bruno Legeard

1- Introduction

Avec l'accélération du rythme des mises en production, les tests automatisés de régression fonctionnelle sont devenus une composante essentielle de l'effort de test, mais aussi un point de peine important pour les équipes projets.

En développant et en maintenant ces tests, les équipes projets sont confrontées à deux défis :

- La pertinence des tests : comment s'assurer de la bonne couverture fonctionnelle des tests de régression automatisés ? Est-ce que les tests couvrent effectivement les parcours applicatifs réels des utilisateurs de l'application ? Comment se prémunir d'une obsolescence (le fameux "Paradoxe du pesticide" de notre syllabus ISTQB Fondation) réduisant les capacités de détection d'anomalies de ces tests ?

- La maîtrise de leurs coûts de maintenance : au fur et à mesure que le référentiel des tests automatisés augmente en volume, l'effort de maintenance croît. Un refactoring régulier doit être réalisé pour améliorer l'architecture d'automatisation à base de mots-clés. Cet effort pèse sur des ressources rares, les automaticiens de test, et conduit souvent à un abandon partiel du patrimoine et à des impacts sur la qualité des livraisons, avec un risque de régression mal vécu par les utilisateurs lorsque cela arrive.

Dans ce chapitre, nous décrivons une ambition et les premiers résultats d'une technologie fondée sur l'analyse des traces d'exécution pour automatiser l'automatisation des tests de régression fonctionnelle. Cette ambition est le fruit de travaux de recherche menés au sein du projet Philae[1] depuis 2018, et du développement actuel d'un produit appelé Gravity[2], actuellement en phase expérimentale.

Notre technologie utilise des algorithmes d'apprentissage automatique sur les traces d'exécution pour sélectionner les parcours applicatifs à couvrir et générer automatiquement les scripts de tests automatisés de régression.

---

[1] Philae est un projet de recherche collaboratif, soutenu par l'Agence Nationale de la Recherche, démarré en 2018 et coordonné par l'Université Bourgogne Franche-Comté - cf. https://projects.femto-st.fr/philae/en/partners

[2] Gravity vise à réaliser automatiquement les tests de régression fonctionnelle à partir des traces d'usage. Gravity est en phase expérimentale et est accessible sur le lien suivant : https://gravity.smartesting.com/
Les techniques d'apprentissage automatique (Machine Learning en anglais) sont une des branches de l'IA qui a connu un très fort développement ces dernières années, avec de nombreux algorithmes supervisés ou non-supervisés et les techniques de deep learning à base de réseaux de neurones.

Nous appelons “traces d’exécution” les données recueillies, sous formes de données analytiques ou de logs, reflétant les parcours utilisateurs réalisés sur l’application. Avec la démarche DevOps, de plus en plus couramment mise en œuvre sur les plateformes web, les traces d’exécution recueillies en production sont utilisées pour détecter et analyser des incidents et pour monitorer la performance applicative (par exemple pour optimiser un tunnel d’achat sur une plateforme de e-commerce). L’utilisation de ces traces d’exécution pour créer et maintenir les tests n’est pour l’instant pas ou peu réalisée.

Dans la suite de ce chapitre, nous montrons comment l’analyse des traces d’exécution en production et en test permet :

– d’analyser et de visualiser sous forme de workflows les patterns d’usage en production
– d’identifier les tests automatisés manquants, pour garantir la couverture des parcours utilisateurs par les tests automatisés de régression
– de sélectionner les parcours à automatiser
– d’analyser les classes d’équivalence sur les valeurs des paramètres pour définir les données de test
– de générer des scripts fondés sur une approche par mots-clés.

La section suivante présente la vision que nous portons d’une robotisation complète des tests de régression, puis nous décrivons l’outil Gravity dans son état actuel, avec une approche pas-à-pas sur un exemple applicatif issu de e-commerce.

2- “We have a dream” : Robotiser les tests de régression

L’automatisation des tests de régression fonctionnelle est un point de peine dont se passeraient bien les équipes de développement logiciel. Le temps consacré à leur maintenance pourrait utilement être dédié aux nouvelles fonctionnalités, mais ces tests sont incontournables et il faut faire avec. Jusqu'à ce que l’innovation permette d’automatiser l’automatisation !

En effet, la disponibilité de plus en plus fréquente des données d'usage et l’arrivée à maturité des techniques d’apprentissage automatique sur les différentes données permettent de faire ce rêve : un robot de test de régression effectue en continu ces tests de façon automatique, sans autre intervention humaine que la configuration du robot. Ce robot garantit la couverture des parcours clients et met à jour en permanence les parcours couverts, en fonction des évolutions du logiciel - “We have a dream”.

Prenons l’analogie avec un robot de tonte. Pour une famille avec des enfants dans une maison et un peu d’espace vert, déléguer la tonte à un robot pour une tonte toujours parfaite et un terrain complètement tondu est un avenir désirable. Cela permet d’éviter une tâche répétitive et chronophage, au détriment du temps qu’il serait possible de passer à profiter de ce jardin. Les robots de tonte automatique sont apparus il y a quelques années pour ce service.

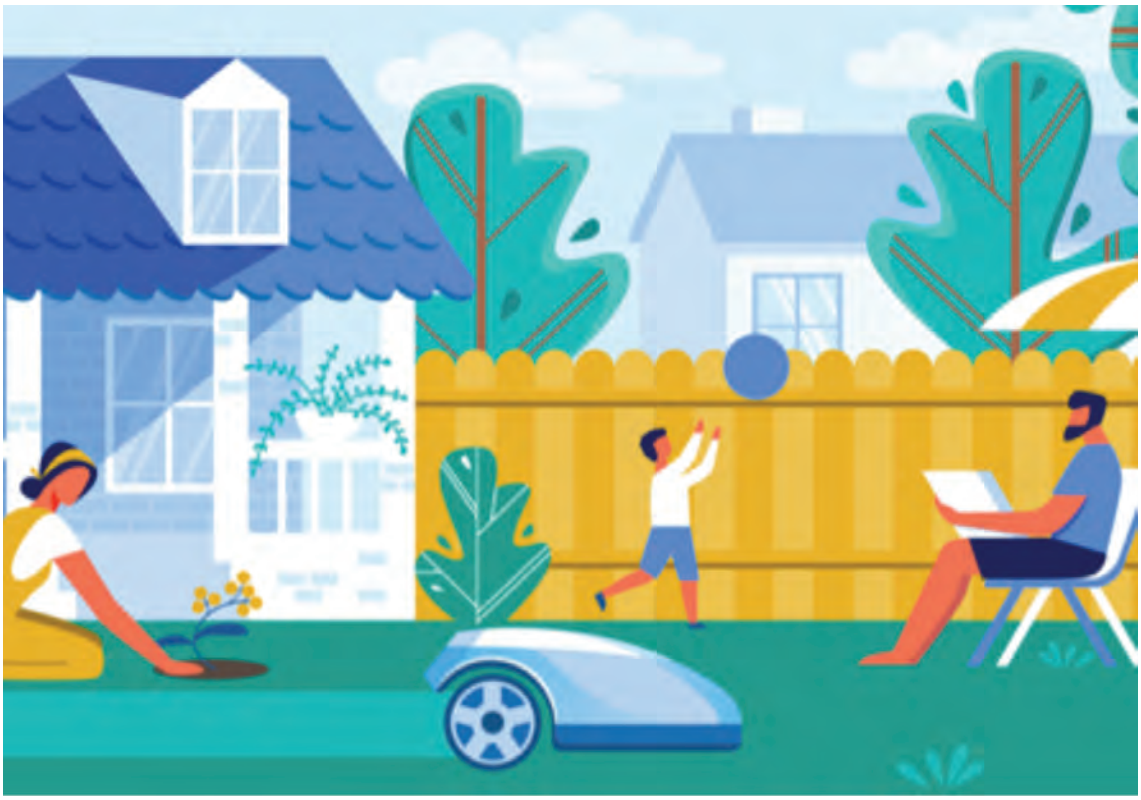


Figure 1 : Robot de tonte

Ceux que l'on connaît à l'heure actuelle ont atteint un degré d'autonomie très élevé. Ils permettent de reconnaître (via GPS et reconnaissance d'image) la zone de tonte, d'éviter les obstacles, d'optimiser les parcours à effectuer, de se recharger et parfois même de se nettoyer seuls.

Sur le chemin de la tonte totalement automatique de la pelouse, les premiers robots tondeuses ont commencé à rendre un premier service, qui a été amélioré au cours du temps pour que chaque tâche devienne de plus en plus automatisée. Il était initialement nécessaire de délimiter le terrain (qui ne devait pas contenir d'obstacles), le chemin de tonte était aléatoire, le robot devait régulièrement être rechargé et nettoyé manuellement. Cependant, le service rendu permettait déjà un gain de temps sur une bonne partie du processus complet de tonte manuelle. Les différentes fonctions ont été automatisées au fil du temps, pour obtenir le haut degré d'autonomie que nous connaissons maintenant.

Pour une équipe de développement de produit logiciel, déléguer les tests de régression à un robot, qui assure une détection parfaite (0% de faux positifs et de faux négatifs) et une bonne couverture des parcours utilisateurs à tester, est un avenir désirable.

C'est sur cette voie que nous développons la technologie Gravity, offrant les premiers services d'analyse des traces et de génération des tests. Nous en présentons le processus dans la section suivante.

3- Des logs aux tests de régression automatisés : le processus Gravity

Afin d'illustrer le processus Gravity sur un exemple, nous utilisons une application de démonstration appelée e-shop.

L'application de démonstration e-shop (https://eshop.smartesting.com) est une instance d'OpenCart (https://www.opencart.com/). Elle offre les services habituels des sites de e-commerce, de la consultation d'articles à leur ajout au panier pour paiement. En voici une capture écran sur la figure ci-dessous :

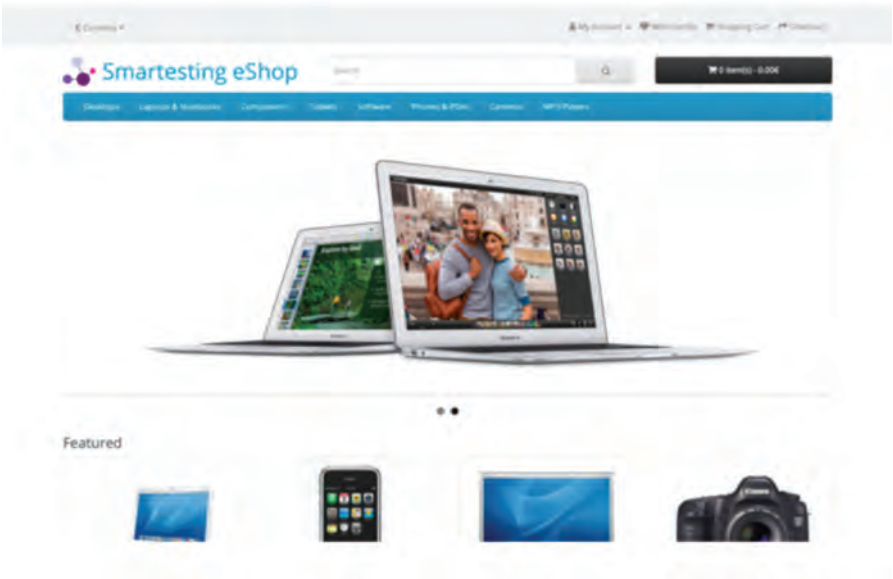


Figure 2 : Application de démonstration e-shop

La figure suivante montre le processus de passage des traces d'usage aux tests avec les différentes étapes de traitement et les données d'entrée et de sortie.

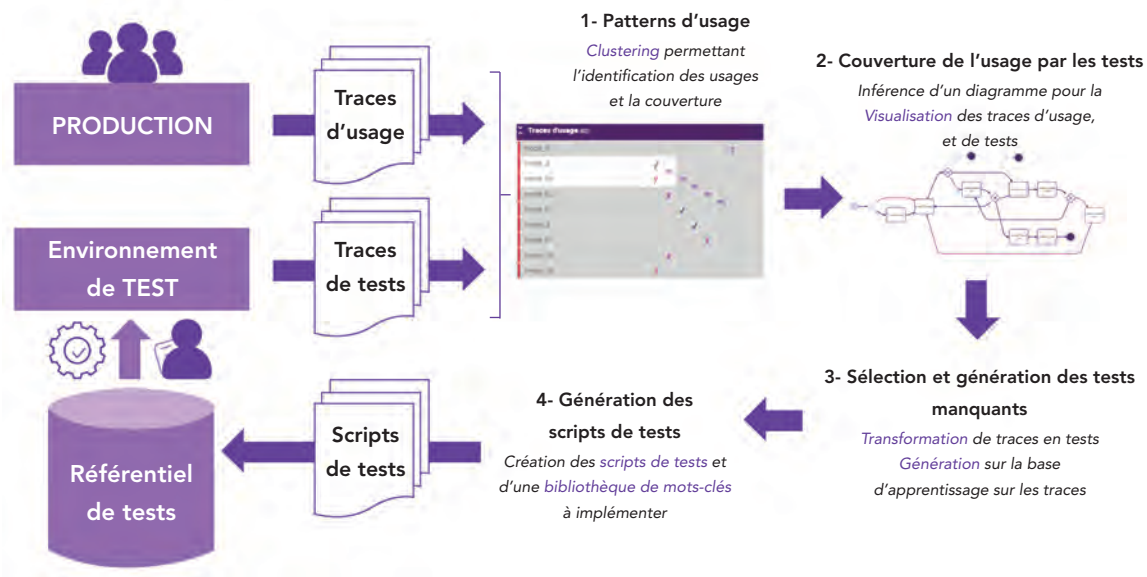


Figure 3 : Processus Gravity

Nous allons décrire les techniques utilisées à chaque étape de ce processus, en commençant par les données d'entrée.

Étape 0 - Acquisition et préparation des données de traces d'exécution

Les données d'entrée sont les journaux d'exécution (logs). Ils sont obtenus, d'une part, dans l'environnement de production (et dans ce cas, les données sensibles à la confidentialité sont anonymisées). D'autre part, les journaux d'exécution des tests (manuels et automatisés) issus de l'environnement de test sont également traités. Ces logs peuvent se trouver dans des fichiers sous différents formats (.json, .csv, .txt, …) ou provenir d'outils dédiés tels que Datadog.

Une trace d'exécution est une séquence d'événements représentant un parcours utilisateur (ou un test) sur le système. Sur les deux ensembles de données, un travail est effectué pour extraire les traces d'usage et traces de tests, les logs contenant les événements de tous les utilisateurs de façon « entremêlée », comme vu ci-après :

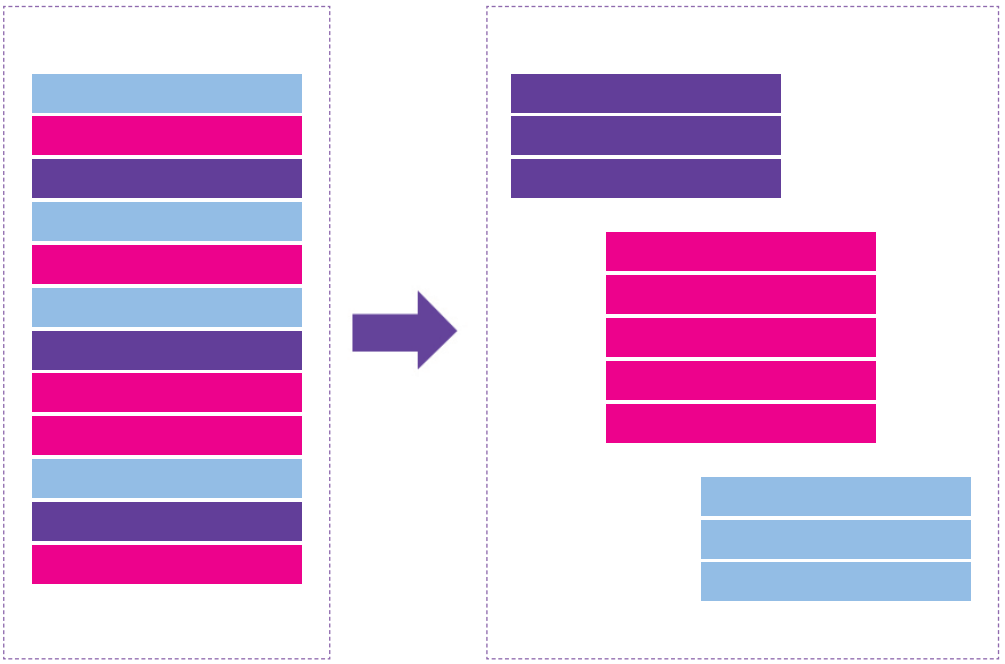


Figure 4 : Des logs aux traces

Étape 1 - Analyse de l'usage à l'aide du clustering hiérarchique

Les traces d'usage et traces de tests sont analysées à l'aide d'algorithmes de regroupement hiérarchique (Clustering hiérarchique). Le clustering est une méthode d'apprentissage consistant à regrouper des individus par similarité, c'est-à-dire partageant des caractéristiques communes. Ici, les séquences proches fonctionnellement sont regroupées, ce qui nous permet de détecter les typologies d'usage de l'application et les tests les exerçant. Le résultat de ce regroupement se trouve sous la forme d'un dendrogramme, dont chaque ligne représente un usage. Plus les usages sont proches, plus les lignes sont proches.

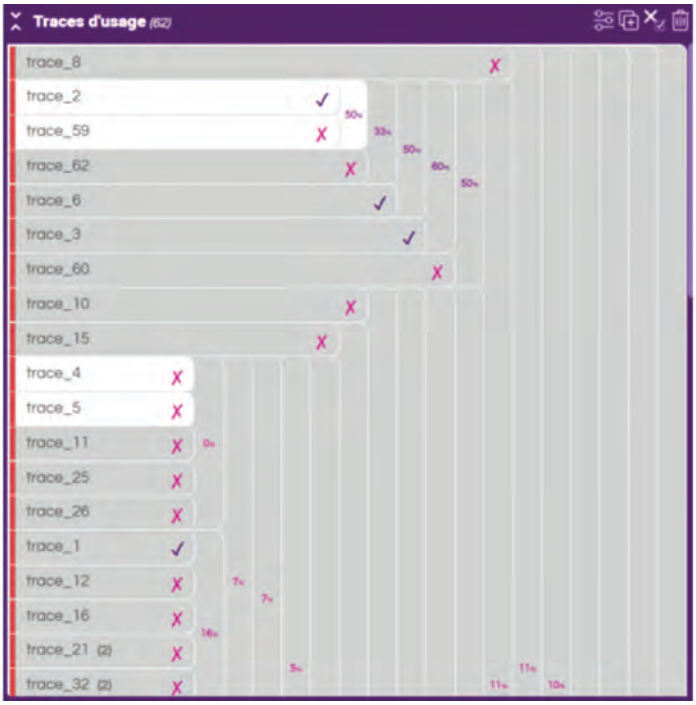


Figure 5 : Dendrogramme représentant les usages d'e-shop

Étape 2 – Comparaison des usages et des tests

Pour chaque usage dans le dendrogramme, une coche ou une croix permettent respectivement de signaler si cet usage est couvert ou non par un test. Pour chaque groupe d'usage, un pourcentage de couverture est indiqué. Il est possible de pondérer ce pourcentage en fonction de la fréquence des usages sur la totalité des traces.

Pour faciliter la compréhension des parcours utilisateurs, ils sont ensuite visualisés sous forme de parcours applicatifs : les boucles, ainsi que les actions, sont factorisées afin de représenter le diagramme résultant de l'ensemble des parcours sélectionnés par l'utilisateur. Dans le but d'analyser la couverture des parcours utilisateurs par les tests, le diagramme est coloré (les parties mauves sont couvertes, les roses restent à couvrir).

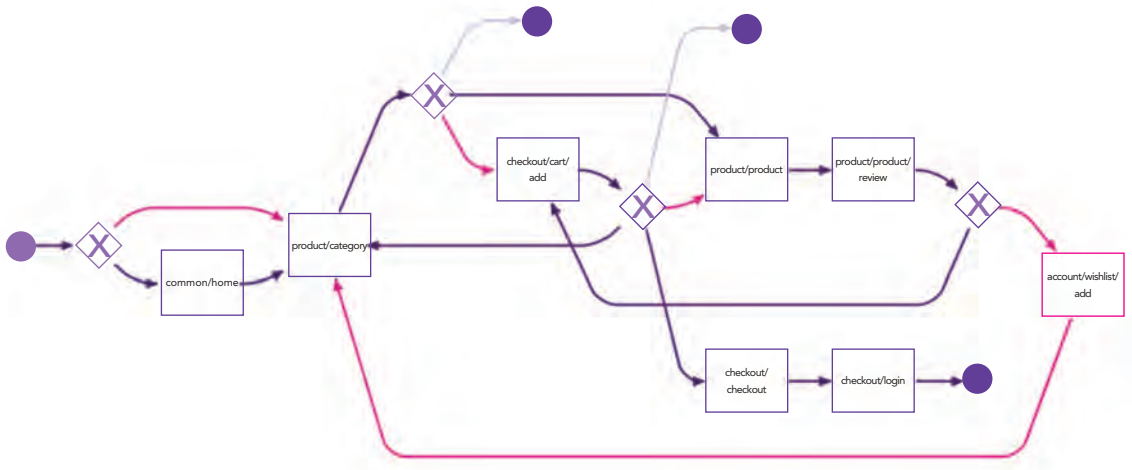


Figure 6 : Diagramme représentant les parcours applicatifs d'un sous-ensemble e-shop

La visualisation des traces sous forme de workflows facilite l'analyse et la prise de décision, pour la sélection des parcours utilisateurs à couvrir en priorité, par la mise à jour des tests de régression fonctionnelle.

Étape 3 - Sélection des traces à automatiser et extraction des données de test

A partir des représentations graphiques ci-dessus, les traces associées peuvent être sélectionnées pour être promues en cas de test, afin d'augmenter la couverture des modèles d'utilisation par les tests.

En parallèle, à partir des modèles de données d'utilisation, des données de test abstraites sont définies pour faciliter l'implémentation ultérieure des scripts. Les actions de l'utilisateur sont également associées à des mots-clés d'automatisation (réutilisés à partir de scripts de test existants ou créés si nécessaire) qui seront utilisés lors de l'étape suivante de génération de scripts.

Étape 4 - Génération des scripts de test dans le format/framework d'automatisation cible

Les scripts de test sont ensuite automatiquement générés dans le format cible (par exemple Selenium Java) en appelant les mots-clés créés ou réutilisés, dans un but d'automatisation. Cette étape est entièrement automatisée. L'implémentation des mots-clés, non encore automatisée, est réalisée par l'automaticien de tests.

Ci-dessous un exemple d'export des scripts de tests, avec la bibliothèque d'automatisation associée. Sur l'exemple e-shop, nous avons choisi un export Java/Cucumber, les mots-clés pouvant être automatisés en Selenium :

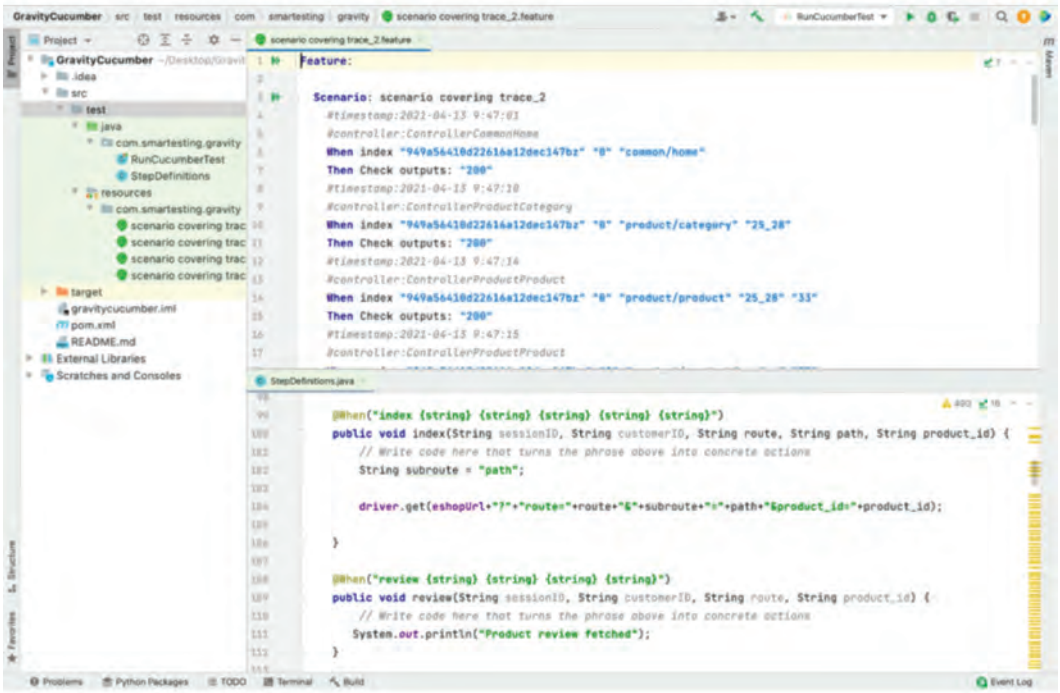


Figure 7 : Exemple de scripts de tests Java/Cucumber générés pour e-shop

4- Conclusion et perspectives

Dans cette première version de Gravity, disponible librement pour expérimentation (*cf. https://gravity.smartesting.com/*), les services rendus aux équipes projets couvrent la couverture des parcours utilisateurs (être sûr de réaliser les bons tests automatisés de régression fonctionnelle) et un ensemble d'assistant pour leur implémentation (pour identifier les données de tests, générer les scripts, associer et factoriser les mots-clés).

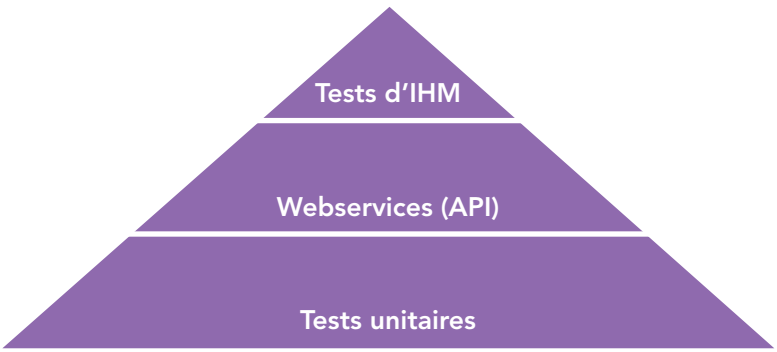
Nos premières expériences portent sur plusieurs projets d'applications digitales (plateforme e-commerce, plateforme d'automatisation de processus, de traitement de données, …) et nous amènent au constat suivant : les équipes intéressées ont déjà une réflexion et ont implanté des solutions de monitoring applicatifs, permettant d'avoir les données de traces d'exécution disponibles pour la génération des tests de régression. C'est une bonne nouvelle : la montée en puissance du DevOps, l'extension du logiciel en SaaS sur le cloud et le besoin de mesurer la performance applicative, rendent disponibles des données de traces d'exécution en production qui constituent un puit de connaissance pour les tests.

Des traces aux tests est un nouveau concept, et nous sommes enthousiastes d'avancer sur cette voie d'une robotisation complète des tests de régression, mais c'est avec les équipes intéressées que nous pourrons réaliser cette ambition. Si ce challenge vous intéresse, si les tests de régression automatisés sont un point d'amélioration dans votre équipe et si vous pensez disposer des données de traces d'exécution nécessaires, alors venez essayer Gravity et contactez-nous, nous ferons un bout du chemin ensemble.

II.8- Automatisation de l'exécution des tests Webservices et batch par des fonctionnels

par Adrien Franco et Marc Hage Chahine

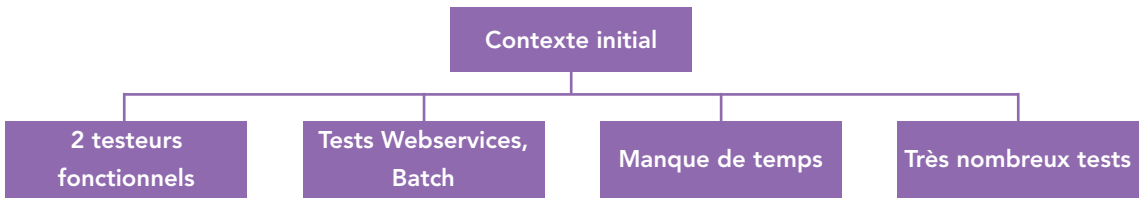
Bien que la pyramide des tests automatisés (rappel ci-dessous) soit connue, l'automatisation des tests reste très liée dans l'imaginaire collectif – et particulièrement pour les ingénieurs logiciels de haut niveau - aux tests d'interfaces graphique (IHM).



Pour rappel, si les tests IHM sont tout en haut de la pyramide, c'est-à-dire les moins nombreux à automatiser, c'est parce que ces tests sont les plus complexes (ils dépendent de nombreux paramètres) à automatiser et à maintenir (avec une forte sensibilité à de nombreux changements). De même, l'IHM nécessite souvent un produit avancé dans son développement et cela peut aller à l'encontre, dans certains contextes, du « shift left ». Il est donc généralement plus intéressant d'automatiser les couches les plus basses de l'application, notamment grâce à des tests de Web-services ou des tests batch, comme cela a été fait dans le retour d'expérience présenté dans cet article.

1- Contexte

Nous sommes ici sur un produit existant qui continue d'évoluer. Ce produit est sans IHM mais complexe. Cette complexité entraîne la nécessité d'exécuter un grand nombre de tests de différents types (Webservices, batch, base de données…), afin d'atteindre une couverture fonctionnelle suffisante. De même, ce logiciel doit régulièrement être testé. Car des modifications du logiciel ou de partenaires sont fréquentes. Cette tâche est assignée à 2 testeurs fonctionnels qui n'avaient pas le temps de préparer les données (1/2 journée pour créer 200 jeux de données nécessaires aux tests) et d'exécuter suffisamment de tests pour valider un niveau de qualité suffisant. Ce manque de temps forçait les testeurs à faire des choix forts et à renoncer à certains tests importants, ce qui engendrait l'apparition d'anomalies évitables en production.



Conscients de cette problématique et de l'apport théorique de l'automatisation, nous avons pris la décision de créer un automate de test avec les fonctions suivantes :

- La génération des fichiers de jeux de données utilisés par les Webservices
- Le lancement automatisé d'offres de service (OS) : les Webservices
- Le lancement automatisé de batch
- Le lancement automatisé d'offres de services et de batch, ordonnancés au sein d'une même exécution
- Le contrôle automatisé des résultats obtenus
- La mise à disposition d'un bilan des résultats obtenus.

La prise en compte du profil non technique des testeurs amenés à travailler sur la solution a également entraîné d'autres besoins comme :

- La réalisation d'un guide utilisateur de l'automate permettant son utilisation par des agents ne disposant d'aucune compétence technique spécifique
- La maintenabilité par tout analyste disposant d'un minimum de connaissances en développement logiciel, grâce à une syntaxe de code source explicite et commentée.

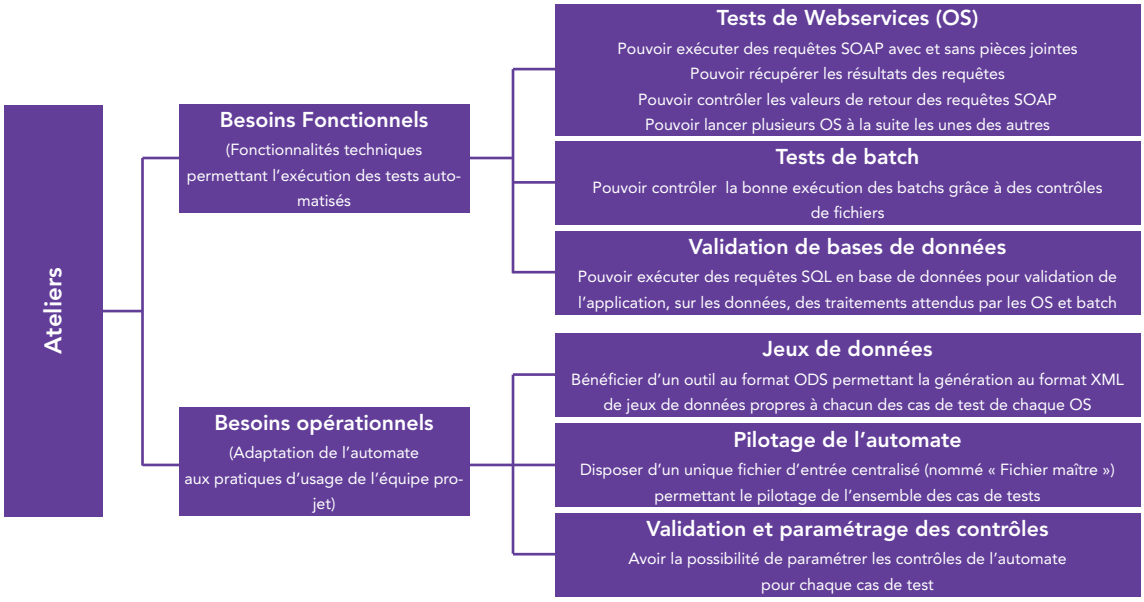
2- Démarche adoptée

Afin de répondre au mieux à ce besoin, nous avons mis en place une démarche en plusieurs étapes :

2.1- Un audit

L'audit s'est fait au travers d'entretiens et d'ateliers. Le but de ce dernier était de bien identifier le contexte et les besoins.

Le retour des ateliers a permis de dégager les points suivants :



2.2- Une définition de la stratégie d'automatisation

Le recueil des différents besoins effectué avec l'audit a également permis la co-construction d'une stratégie de test.

Nous avons choisi de faire intervenir un consultant expérimenté en automatisation des tests, afin de mettre au point un POC (Proof Of Concept).

Les résultats du POC servant alors de base, pour la suite de l'automatisation des tests, avec une sélection et une priorisation des tests à automatiser. Le POC a également permis de prévoir les retours sur investissements de l'automatisation.

2.3- La mise en place d'un POC (Proof Of Concept)

Le POC a mis en lumière que l'automatisation était techniquement faisable mais que, pour cela, il fallait résoudre certains challenges comme :

- Une utilisation exclusive de LibreOffice

Excel n'étant pas un outil utilisé dans cette entreprise

- Enregistrement de jeux de données multiples au format XML

Il fallait traiter des nœuds multiples mais aussi réussir à bien identifier les points à modifier ou traiter.

- L'envoi et la réception de requêtes SOAP avec des pièces jointes

La documentation à ce sujet est faible car SOAP est de moins en moins utilisé, contrairement à REST.

- L'utilisation d'OpenVPN pour accès à la base de données et aux fichiers

Contrainte liée à l'environnement du logiciel et à la politique de l'entreprise

- Rendre générique l'envoi de requêtes SQL.

2.4- La création de l'automate et l'industrialisation des tests

Suite au POC, la décision a été prise d'automatiser. Le choix de l'outil s'est porté sur la création d'une feuille de calcul servant d'interface aux testeurs. Cette feuille de calcul contrôle Robot-Framework qui, en plus de contribuer à la résolution de nombreux challenges, a l'avantage d'être très flexible et de s'adapter à différents types de tests. C'est également un outil Open Source (ce qui était une contrainte forte de l'entreprise).

Les différents challenges ont été relevés de la manière suivante :

Contraintes / Difficultés	Solution mise en œuvre
Utilisation exclusive de LibreOffice	Création et implémentation d'une librairie Python spécifique aux fichiers Ods
Enregistrement de jeux de données multiples au format XML	Utilisation du xpath de chaque OS (Webservice)
Envoi de requêtes SOAP avec pièce jointe	Développement spécifique d'une méthode d'envoi de requête

Réception de requête SOAP avec pièce jointe	Développement spécifique de l'analyse de la réponse SOAP
Utilisation d'OpenVPN pour accès à la BDD et aux fichiers	Implémentation de scripts de connexion à OpenVPN
Rendre générique l'envoi de requêtes SQL	Utilisation de paramètres avec codification spécifique

La souplesse et l'ouverture de la solution libre RobotFramework ont permis de solutionner l'ensemble de ces difficultés.

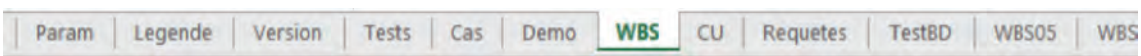
2.5- La restitution

Au final, l'automate proposé répond entièrement aux besoins fonctionnels en étant capable d'exécuter :

- Des tests de Webservices SOAP qui exécutent et récupèrent des requêtes, avec ou sans pièces jointes, en contrôlant les valeurs de ces requêtes. Des tests batch qui font des contrôles de fichiers
- De la validation de base de données à travers des requêtes SQL, qui valident l'impact des requêtes SOAP.

La solution proposée sous la forme de 2 formats de feuilles de calcul (1 pour les tests, 1 pour la génération de données), très simples à utiliser et à maintenir, permet de répondre aux besoins opérationnels, en gérant simplement les jeux de données, en permettant de piloter aisément et en étant très simple d'utilisation pour des fonctionnels qui n'ont donc pas besoin d'avoir de connaissances techniques particulières, comme cela peut parfois être le cas avec de l'automatisation de tests. Cette solution fonctionne grâce à une architecture simple et compréhensible de l'outil de gestion de l'automate. On peut, par exemple, penser à des choix ergonomiques comme :

- Chaque onglet a un but et un nom définissant clairement son rôle. On a, par exemple, un onglet listant les tests, un autre listant les OS testés ou encore un onglet « légende » permettant d'expliquer le sens de noms utilisés.

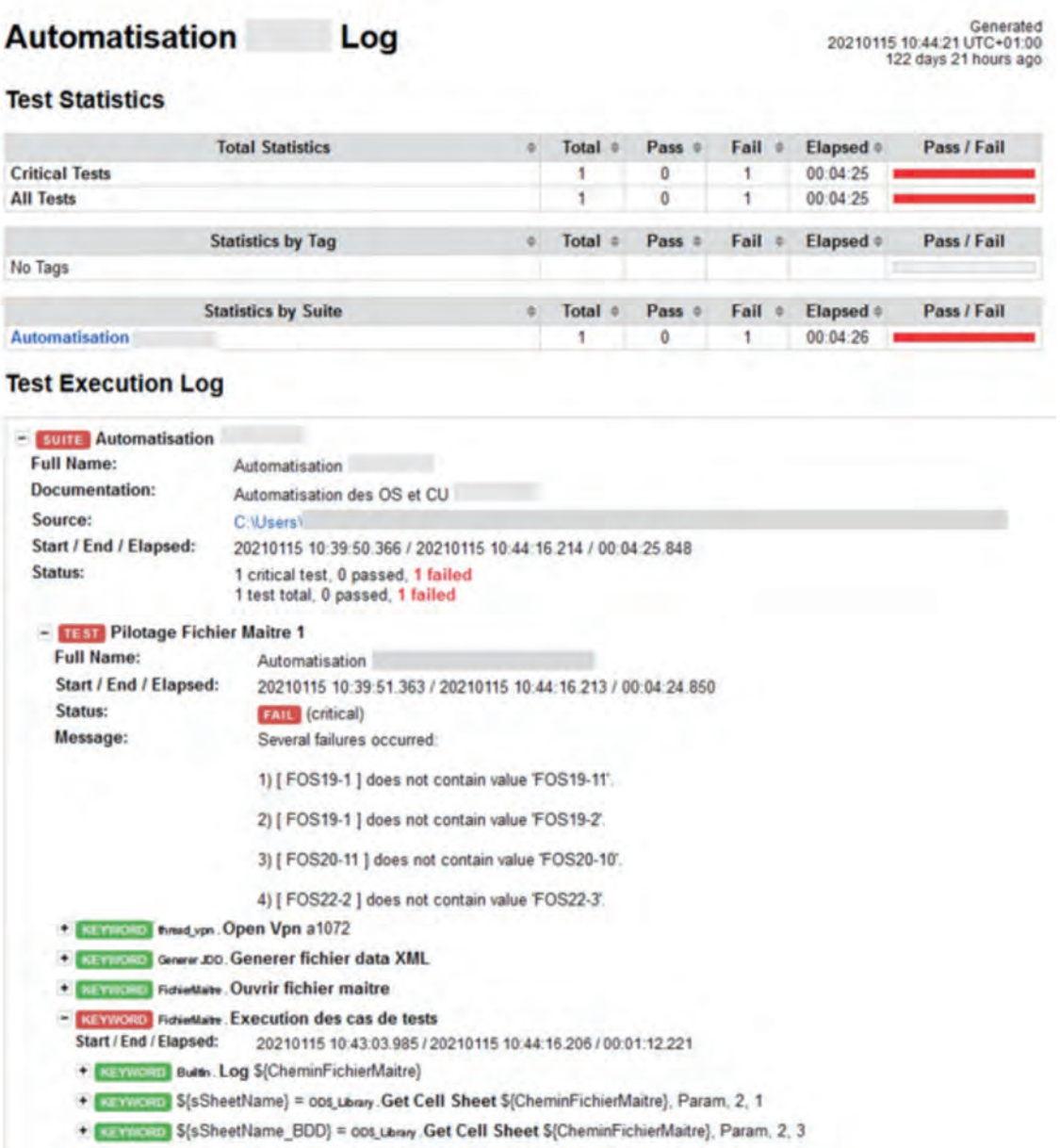


- Chaque ligne à un rôle spécifique et clair. Pour les onglets sur les tests, cela correspond à un test ou un pas de test, dans le cas d'un test utilisant plusieurs OS. Dans l'onglet ci-dessous (anonymisé), chaque ligne correspond à un Webservice :

	A	B	C	D	E	F	
1	WBS/CU	Nom	**Sous-méthode 01**	Activité 01	Activité 02	Activité 03	**Sous-**
2	WBS_01	Créer_V	**N/A**	Contrôler D	Créer entrées		
3	WBS_02	Rechercher_O	**N/A**	Contrôler A	Rechercher O	Trier O	
4	WBS_03	Enrichir_D	**N/A**	Contrôler A	Rechercher D	Enregistrer données	
5	WBS_04	Créer-C	**N/A**	Contrôler D	Créer entrées tables		

- Chaque test est paramétrable. Les paramètres sont gérés avec les colonnes de leur ligne dédiée.

La gestion des données se fait également avec un autre fichier, sous forme d'une feuille de calcul avec un onglet par OS et listant des cas ou pas de test (1 test par ligne) pour chacun de ces OS. Enfin, les bilans sont détaillés et générés automatiquement grâce à RobotFramework, ce qui permet d'analyser simplement et rapidement les causes d'échec.



Automatisation Log

Generated
20210115 10:44:21 UTC+01:00
122 days 21 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	0	1	00:04:25	
All Tests	1	0	1	00:04:25	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Automatisation	1	0	1	00:04:26	

Test Execution Log

SUITE **Automatisation**
Full Name: Automatisation
Documentation: Automatisation des OS et CU
Source: C:\Users\
Start / End / Elapsed: 20210115 10:39:50.366 / 20210115 10:44:16.214 / 00:04:25.848
Status: 1 critical test, 0 passed, 1 failed
1 test total, 0 passed, 1 failed

TEST **Pilotage Fichier Maitre 1**
Full Name: Automatisation
Start / End / Elapsed: 20210115 10:39:51.363 / 20210115 10:44:16.213 / 00:04:24.850
Status: FAIL (critical)
Message: Several failures occurred:

1) [FOS19-1] does not contain value 'FOS19-11'.

2) [FOS19-1] does not contain value 'FOS19-2'.

3) [FOS20-11] does not contain value 'FOS20-10'.

4) [FOS22-2] does not contain value 'FOS22-3'.

KEYWORD thread_vpn . **Open Vpn a1072**
KEYWORD Generer JDD . **Generer fichier data XML**
KEYWORD FichierMaitre . **Ouvrir fichier maitre**
KEYWORD FichierMaitre . **Execution des cas de tests**
Start / End / Elapsed: 20210115 10:43:03.985 / 20210115 10:44:16.206 / 00:01:12.221
KEYWORD BuiltIn . **Log** ${CheminFichierMaitre}
KEYWORD ${sSheetName} = oos_Library . **Get Cell Sheet** ${CheminFichierMaitre}, Param, 2, 1
KEYWORD ${sSheetName_BDD} = oos_Library . **Get Cell Sheet** ${CheminFichierMaitre}, Param, 2, 3

2.6- Description de l'automate

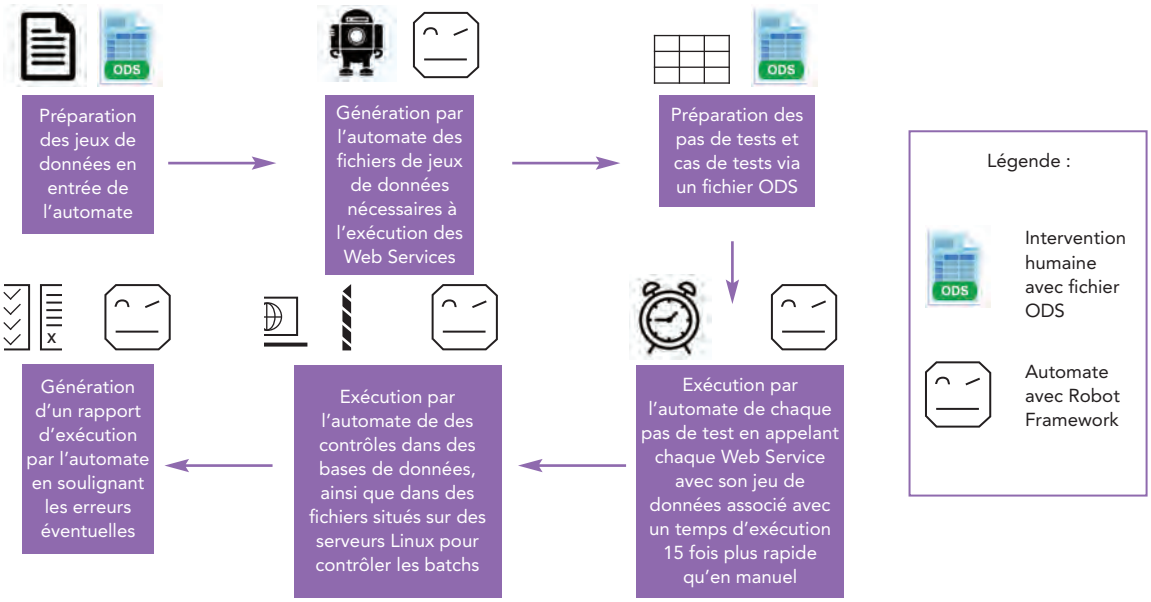
L'automate a deux fonctionnalités principales qui sont la création de jeux de données et l'exécution des tests.

Pour générer les données, nous avons mappé les différents fichiers xml attendus en sortie avec un onglet par Webservice. Nous avons mis chaque élément du fichier xml en colonne avec son xpath, pour pouvoir écrire correctement les bonnes valeurs au bon endroit.

Pour la partie exécution, chaque ligne va correspondre à un cas de test avec son propre identifiant et générer un fichier différent.

Pour exécuter les tests, l'automate a besoin d'être lancé à l'aide d'une ligne de commande. Etant donné que le profil des utilisateurs n'est pas très technique, nous avons encapsulé cette ligne de commande dans un fichier « .bat » qui enclenche l'exécution de l'automate (RobotFramework). Robotframework va alors chercher les informations dans les feuilles de calcul, afin de générer et d'exécuter les tests. Du point de vue de l'utilisateur, il suffit simplement de cliquer sur le fichier « .bat » pour générer les données de tests et exécuter la campagne.

Pour résumer, la solution développée peut se schématiser comme ceci :



2.7- La vie de l'automate

Les projets de développement ne sont pas un long fleuve tranquille. Il en est de même avec l'automatisation implémentée avec notre outil, où certains imprévus ont fait irruption.

Un des plus notables est probablement celui où le script Python, permettant de lancer la connexion au VPN, n'obtenait pas de valeur de retour. Cela rendait impossible la suite de l'exécution normale de l'automate, étant donné que la tâche précédente ne se terminait jamais. Pour contourner ce point, nous avons utilisé la technique du multithread, en mettant donc dans un thread dédié la connexion au VPN et, dans une autre, la partie purement fonctionnelle de l'exécution de l'automate avec ses fonctionnalités dédiées.

De plus, l'automate interagit avec la feuille de calcul LibreOffice, grâce à une librairie spécifique

codée en Python qui permet de lire dans n'importe quelle cellule de n'importe quel onglet et va permettre de traiter les valeurs propres à chaque cas de test (jeu de données à générer ou cas de test à exécuter).

Les éventuels messages d'erreur que peut rencontrer l'automate sont remontés dans les logs de Robot Framework grâce à l'implémentation de ces différents contrôles, à travers les mots-clés définis par l'utilisateur.

3- Les résultats de l'automatisation

L'automatisation proposée est un franc succès. Elle est actuellement utilisée et les testeurs fonctionnels continuent de développer la couverture de test et de faire évoluer simplement les tests grâce à cet outil.

Cette automatisation est un succès car elle a engendré :

- Une amélioration de la qualité grâce à une meilleure couverture des tests à chaque campagne.

L'automatisation des tests a permis de multiplier les tests et les données. Là où la création de 600 fichiers pour les données nécessitait un peu plus d'une journée, maintenant seules 2 minutes suffisent.

De même, ces jeux de données générés sont beaucoup plus fiables car les erreurs humaines (courantes sur les tâches longues et répétitives) sont maintenant évitées.

De plus, l'outil, du fait de sa capacité à faire évoluer les tests et à générer facilement des données, permet d'adapter les tests aux différentes campagnes et de limiter l'impact du paradoxe des pesticides.

Le problème initial de ne plus pouvoir faires les tests pour assurer un niveau de qualité suffisant, dans le temps imparti, est maintenant oublié et il y a moins d'anomalies en production.

Initialement, les 2 testeurs disposaient de 3 jours pour exécuter le plus de cas de tests possible à chaque release ; au vu du temps que prenait la création des jeux de données, l'exécution et les contrôles, ils ne pouvaient, au mieux, effectuer qu'entre 150 et 200 cas de tests.

Maintenant, ces 200 cas de tests sont exécutés en moins de 3O minutes par l'automate.

Il leur est maintenant possible d'exécuter une campagne de régression « complète », c'est-à-dire plusieurs milliers de tests, de manière automatique, en l'espace de quelques heures seulement.

- Un gain de temps

Les campagnes de tests sont maintenant beaucoup plus courtes, alors que la couverture des tests est beaucoup plus grande et les tests beaucoup plus nombreux.

De même, les jeux de données générés sont beaucoup plus rapides : 2 minutes pour générer 600 fichiers xml de jeux de données contre plusieurs heures en manuel.

De plus, il était auparavant nécessaire pour les tests de Webservices d'ajouter manuellement les pièces jointes dans chaque requête via SOAP UI. C'est maintenant l'automate qui s'en charge, ce qui permet de faire plus de tests avec pièces jointes, sans devoir attendre une intervention humaine.

Là où auparavant les 2 testeurs pouvaient mettre plus de 15 minutes à dérouler un cas de test (qui consiste à créer son fichier de jeu de données, l'insérer dans la requête Soap, exécuter la requête, analyser le résultat, faire des contrôles en base de données…), l'automate effectue toutes ces actions en seulement 4 ou 5 secondes, selon le temps que met le Webservice pour répondre.

L'automate est donc très véloce et permet, en plus du temps gagné, de s'assurer de la qualité, car il peut répéter à l'infini les mêmes actions, qui peuvent se révéler fastidieuses lorsque la fatigue du testeur se fait sentir. L'automate permet, en effet, de faire plus en quelques heures (des milliers de tests) que ce qui était fait en 3 jours (200 tests). En plus d'améliorer la vision sur la qualité, cela réduit drastiquement le temps d'exécution des campagnes de régression.

Maintenant, il suffit juste de passer du temps sur la conception des cas de tests dans le fichier ods et une fois la tâche effectuée, plus besoin d'y toucher. Cette action permet également de continuer à construire le référentiel de tests.

- Le gain de temps permet de dépenser son temps sur des tâches à plus forte valeur ajoutée.

Le temps est gagné sur l'exécution et l'analyse de résultats, car l'automate fait lui-même les vérifications en base de données et sur serveur pour les fichiers issus des batch.

4- Conclusion

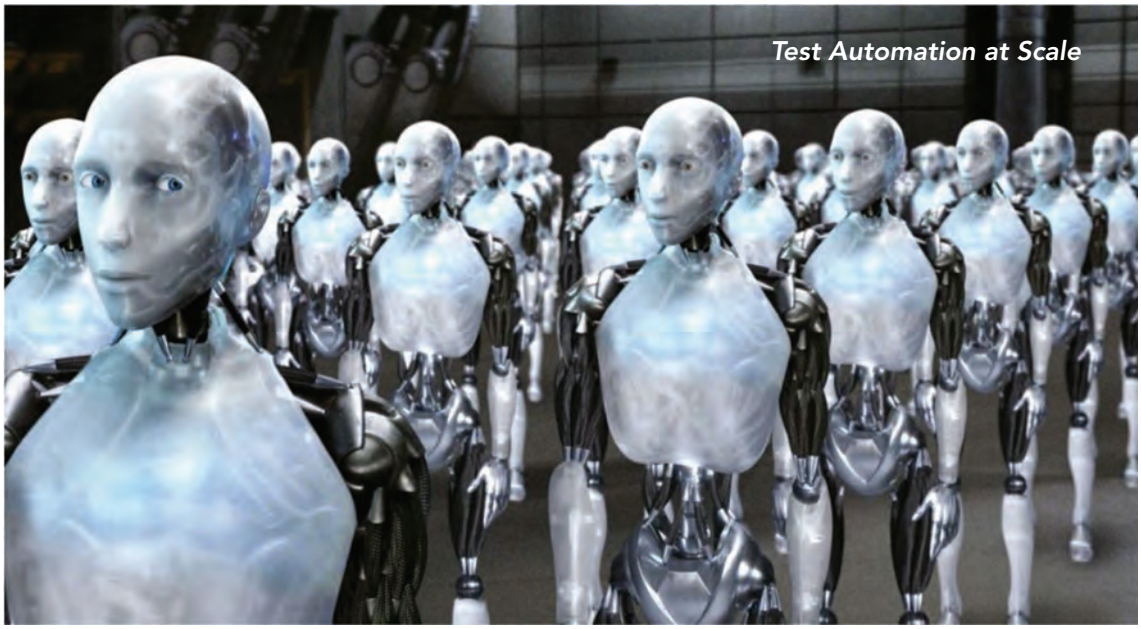
L'automatisation de l'exécution des tests, ce n'est pas que l'automatisation des tests IHM. Automatiser des tests de Webservices offre généralement un retour sur investissement très rapide et visible, car ils sont nombreux, assez similaires entre eux et évoluent moins que les interfaces graphiques, ce qui implique beaucoup moins de maintenance et réduit donc le coût et augmente la rentabilité de l'automate.

Ce n'est pas pour rien que la pyramide des tests automatisés se représente souvent avec les tests d'intégration / Webservices / API comme étant les tests à automatiser, juste après les tests unitaires.

De même, comme nous l'avons vu dans ce retour d'expérience, la technicité de ces tests peut être cachée avec une interface simple et accessible à des profils non techniques, mais qui répond à l'ensemble des besoins.

II.9- Automatisation des tests de bout en bout à l'échelle

par Didier Birenbaum



L'automatisation des tests est primordiale dans le contexte du développement d'applications en mode agile. Tester tôt, tester souvent et donc tester rapidement est nécessaire à la fluidité des opérations de développement, de test et déploiement.

Je vous propose de partager notre expérience chez Crossknowledge sur l'évolution de notre système d'automatisation des tests sur les dix dernières années.

1- Le contexte

Crossknowledge : « The skills you need to succeed"

Notre société à 20 années d'expérience dans le digital learning. Osé en 2000, ce pari de la formation à distance est maintenant considéré comme une évidence, grâce à l'amélioration des technologies et du matériel, rendant l'expérience utilisateur ludique, personnalisée et performante.

Nous sommes fiers de contribuer à la montée en compétence de 12 millions d'utilisateurs répartis dans 130 pays.

Crossknowledge a été acquis en 2014 par « Wiley », maison d'édition américaine fondée en 1807, spécialisée dans la publication de revues scientifiques, universitaires et techniques et dans le développement de solutions digitales d'acquisition de compétences et de connaissances. Wiley est connue en France pour sa collection « For Dummies » (Pour les nuls) adaptée dans le monde entier.

2- L'automatisation des tests n'est pas un long fleuve tranquille

Les équipes d'automatisation sont confrontées à de nombreux écueils qui peuvent à tout moment mettre en péril les projets.

- **La testabilité :** Condition sine-qua non. Pas de tests automatisés sans une bonne testabilité de l'application à tester. L'idéal étant d'inclure cette exigence dès la conception et le développement.

- **La fiabilité des résultats :** Les projets prennent rapidement l'eau et finissent par couler si les résultats des tests sont peu fiables et sujets à questionnement. La confiance dans la fiabilité des résultats est primordiale. L'élimination des sources de « flakyness » et d'instabilité doit être prioritaire par rapport au développement de nouveaux tests.

- **La maintenance des tests et du framework de tests :** L'automatisation des tests doit être considérée comme un projet de développement en soi. Une architecture modulaire ouverte à la modification et au redimensionnement constitue la garantie de fondations saines et évolutives. La maintenance du framework et des tests doivent être facilitée. L'utilisation de bonnes pratiques, comme par exemple le « Page Object Model » pour le test des applications web, contribue à cet objectif.

- **La complexité du SUT (System Under Test) :** La maîtrise de la complexité fonctionnelle et technique du système à tester est primordiale. Comprendre l'architecture du SUT permet d'appréhender et de maîtriser les spécificités des environnements de développement, test, préproduction et production.

• **L'automatisation des tests en production :** Tester en production est une opération délicate mais nécessaire pour s'assurer du bon déroulement du déploiement et détecter avant les utilisateurs tout problème lié à la spécificité de l'environnement de production. Ce type de test renforce la confiance des équipes de déploiement.

• **La disponibilité des environnements de test :** C'est une cause majeure de goulot d'étranglement dans les activités de test, qu'elles soient manuelles ou automatisées. Être contraint d'attendre que l'environnement de test soit disponible pour exécuter une campagne de tests est un énorme frein à la fluidité des activités de développement.

• **La fréquence des déploiements :** L'accélération de la fréquence de déploiement est nécessaire pour délivrer de la valeur le plus vite possible. La maîtrise de l'automatisation des tests à l'échelle permet de sécuriser l'augmentation des fréquences de livraison.

• **La maîtrise des couts :** Les projets d'automatisation des tests sont réputés gourmands en ressources et à l'échelle encore plus couteux, il convient donc de trouver les solutions les plus efficientes selon le contexte.

3- La problématique du temps d'exécution des tests

Un des principaux paradigmes des projets d'automatisation de test est le temps d'exécution.

Le principal objectif d'une solution de tests automatisés de bout en bout est sa capacité à donner **rapidement** un résultat de test **fiable**.

Au début des projets, le nombre de tests est souvent modéré et l'on reste dans des temps d'exécutions inférieurs à l'heure, l'idéal étant de donner un résultat d'une campagne de tests en 30 minutes.
Le problème survient lorsqu'on commence à multiplier les tests pour augmenter et améliorer la couverture fonctionnelle.
Sans une solution intégrant l'automatisation des tests à l'échelle, les problèmes de cout d'infrastructure, de maintenance, de stabilité et de durée d'exécution sont très difficiles à résoudre.

4- La solution utilisée chez Crossknowledge

Disponibilité des environnements de test :

- L'utilisation de Docker a changé la donne. Désormais, il suffit de quelques secondes pour déployer une instance de l'application sur un poste de développement ou de test. Le « dockerfile », à l'image d'une recette de cuisine, précise les ingrédients nécessaires à l'élaboration de l'application à installer et à exécuter, garantissant facilité, rapidité et fidélité de reproduction de l'environnement à tester.

Parallélisation des tests :

- L'automation des tests à l'échelle est rendue possible grâce à l'exécution des tests en parallèle. Cela implique que chaque test est indépendant. Chaque test peut être exécuté dans n'importe quel ordre et indépendamment des autres tests. Chaque test est responsable de créer les données nécessaires à son exécution. Afin d'éviter tout problème d'inconsistance des données entre chaque test, la base de données, ainsi que la mémoire cache, est rafraichie avant chaque test. Cela a permis de réduire fortement l'instabilité des résultats des tests.

- Zalenium : Fruit d'un projet Open source développé par l'équipe de test de Zalando. Il permet de fournir une Selenium Grid à l'échelle, avec la possibilité de capture vidéo de l'exécution du test. Le dimensionnement de la machine détermine le nombre maximum de sessions de tests (chrome ou firefox) que l'on peut exécuter en simultané. Nous utilisons actuellement une machine avec 8 CPUs qui permet d'exécuter 7 sessions, 1 CPU étant réservé pour l'encodage des vidéos pendant l'exécution des tests.

Maîtrise des couts :

- Cloud computing :

Grâce à la technologie de Cloud Computing, nous payons que ce que nous utilisons et nous adaptons la taille de notre infrastructure de test en fonction du besoin.

Grâce à cette architecture à la demande, le cout d'une campagne de régression de test en moyenne de 50 centimes d'euros pour une durée d 'exécution de 30 minutes. Dans

cette configuration, 7 machines ubuntu exécutent en parallèle plusieurs centaines de tests E2E.

Notre serveur Jenkins gère la création des ressources nécessaires et leur destruction après utilisation pour optimiser les couts.

• Ubuntu : Les workers de notre solution de test fonctionnent sous Ubuntu, ce qui permet de déployer des machines tout en maîtrisant le cout.

Intégration continue :

• Jenkins est au cœur de notre processus d'intégration continue. L'utilisation de pipelines Jenkins scriptés en Groovy nous permet d'implémenter les différentes étapes :

- Création de la « Queue » des tests à exécuter (Service Amazon SQS) à partir de la suite de tests de non-régression extraite via API de notre système de gestion des tests « TESTRAIL ».
- Création et démarrage à la volée d'une machine Ubuntu docker Selenium Grid « Zalenium ».
- Création et démarrage à la volée des workers ubuntu (jusqu'à 7 suivant le nombre de tests à exécuter en parallèle).
- Installation du code du SUT et du framework de test à partir de Github et récupération des images Docker à partir de Amazon ECR. Chaque machine démarre une instance du SUT sous docker ainsi que le framework de test (python+selenium).
- Chaque worker consomme le prochain test à exécuter depuis la Queue Amazon SQS jusqu'à épuisement de la queue.
- Les capture vidéo des sessions de tests en échec sont uploadées dans le système de stockage cloud Amazon S3 pour pouvoir être archivées et analysées.
- Les résultats de test et les journaux d'exécution sont publiés dans Jenkins pour une analyse rapide des tests en échec.
- Les crédentials TESTRAIL sont récupérés à partir du coffre-fort digital AWS secret manager.
- Les résultats de test sont créés dans TESTRAIL avec toutes les informations de traçabilité et d'analyse.
- La Queue AWS SQS est détruite.
- Toute les machines sont détruites à moins qu'une autre exécution soit lancée dans les 30 minutes.

Pipeline Architecture

5- Conclusion

Notre solution de tests automatisés à l'échelle nous a permis de soutenir l'objectif d'augmentation de la fréquence de nos livraisons et de déploiement continu. La bonne collaboration avec les équipes de développement et les équipes de DevOps a rendu possible le succès de notre solution.

II.10- Le RPA appliqué au métier du test

par Jérôme Beaumont

Introduction

L'acronyme RPA pour Robotic Process Automation est un nouveau buzzword incontournable de l'IT. Cet article a pour objectif d'exposer le contexte du métier du test et en quoi le RPA peut être une réponse à ces enjeux autour de quelques retours d'expérience : bienvenue dans le monde de l'automatisation appliquée au métier du test. Dans de nombreux domaines, les nouveaux services digitaux proposés par les entreprises ont accéléré les mises en service. Les projets informatiques ont dû s'adapter à cette nouvelle contrainte du « time to market ». Cette transformation touche à la fois le produit logiciel et son mode de fabrication :

- La conduite de projet passe du cycle en V au mode Agile avec des équipes reserrées aux compétences élargies : développeur, testeur, opération.
- La conception de la solution migre du modèle 3-tiers on-premise vers des architectures micro-services compatibles avec le cloud.
- Les exigences augmentent notamment celles sur les performances, la sécurité et sur l'expérience client qui doit prendre en compte les terminaux mobiles.

Dans ce contexte, le métier du test informatique est en forte évolution. Il doit répondre à la double contrainte de rapidité d'exécution et de réduction des coûts. Dès lors, l'automatisation peut être une première réponse à ce double défi.

1- Le métier du test en forte évolution

La phase de test sur un projet est la phase la plus difficile. Historiquement positionnée en séquence avec la phase de réalisation du produit logiciel, elle est alors sur le chemin critique du projet et soumise à la contrainte de la date de mise en service rarement négociable. En méthode Agile, les sprints sont de 2 à 3 semaines, ce qui met la durée de la phase de test sous une forte contrainte, de l'ordre de 2 à 3 jours au sein d'un sprint, parfois 5 jours sur le sprint de release où on décide de faire une campagne pour vérifier la non-régression. On comprend tout de suite que la phase de test, ainsi positionnée, devient un goulot d'étranglement du projet. Ce constat a fait naître les initiatives autour du shift-left, c'est-à-dire intégrer l'effort de test au fil de l'eau de la fabrication des user stories, afin d'éviter l'accumulation de tests en fin de sprint. Le mouvement TDD (Test Driven Development) en est une forme de réponse, le développeur est ainsi chargé de mettre en place un test au sein de son code, avant de développer les évolutions demandées. Cette méthode est très efficace pour adresser les tests unitaires et permet d'inclure

les tests avec le code. En contrepartie, elle ne peut pas adresser la totalité des niveaux de test de la pyramide et elle positionne le développeur en juge et partie de la qualité.

De plus, l'augmentation des exigences aboutit à une véritable explosion combinatoire de cas de test, humainement non atteignable. Pour réduire cette combinatoire et focaliser l'effort de test, il faut appliquer une stratégie de test basée sur l'évaluation du risque d'un dysfonctionnement et de son impact. Dans un contexte de temps fortement contraint, il est difficile de systématiser cette approche qui reste une évaluation le plus souvent « à dire d'expert », où il faut évaluer le risque de chaque changement (Risk Base Testing). Au moment de la mise en service, il est également indispensable de connaître factuellement la qualité du produit, ce qui implique un effort de traçabilité de tous les résultats de test et, pour chaque test, de connaître l'exigence qualité qu'il couvre.

La pression sur les coûts est également très forte. La phase de test est par nature en spirale, c'est-à-dire que son coût peut être difficile à maîtriser puisqu'il est lié à la maturité du logiciel testé. Moins le logiciel est mature, plus longue sera sa phase de validation.

On voit bien les nouveaux enjeux se dessiner :

- tester plus rapidement pour répondre au time to market,
- tester plus massivement pour répondre aux exigences qualité,
- tester plus efficacement pour maîtriser les coûts.

« Plus fréquent, Plus massif, Plus efficace » : c'est la devise olympique que les méthodes de tests actuelles doivent relever. Lorsque j'ai quitté SFR en 2015, les mises en service avaient lieu une fois par mois, chez Pages jaunes en 2017 c'est une mise en service chaque semaine, chez Amazon c'est une mise en service toutes les heures. Le test basé sur la seule ressource humaine n'est tout simplement plus envisageable.

2- L'automatisation, une première réponse

Nous l'avons vu, tester manuellement un logiciel aujourd'hui ne permet plus de répondre aux enjeux du time to market. Cela dit, le regard humain reste indispensable puisqu'il est le seul à pouvoir développer des stratégies d'exploration et de détection des dysfonctionnements réellement intelligentes. Il s'agit de voir maintenant en quoi la machine peut assister l'homme dans son travail. On peut faire l'analogie entre l'activité de test et la pêche en mer. Il s'agit d'aller dans les zones poissonneuses (celles où il y a le plus de chance de trouver des dysfonctionnements) et d'attraper un maximum de poissons. Ici, le testeur est le capitaine du bateau ; il choisit les zones de pêches. La machine, c'est la taille et la dimension des mailles du filet. Il ne vient pas à l'esprit d'un pêcheur de partir sans un bon filet. La machine est donc un outil qui permet d'augmenter les capacités du testeur pour faire face aux nouveaux enjeux.

Voici les sept bénéfices que l'on peut atteindre en utilisant la machine pour automatiser certaines tâches de l'activité de test :

a- Une procédure automatisée est répétable, ce qui permet de massifier les interventions.
b- Le lancement d'un automate est simple et beaucoup moins pénible par rapport à son équivalent manuel. L'équipe devient plus réactive.
c- La procédure automatisée apporte de la régularité. La fréquence des erreurs se réduit. Les tâches complexes sont sécurisées.
d- Les actes automatisés sont banalisés, ce qui permet de confier des travaux à plus de personnes sans compétence particulière.
e- Le résultat des tâches automatisées est visible au fil de l'eau. L'état des actions est ainsi régulièrement partagé sans attendre la consolidation d'un reporting.
f- Les modes opératoires deviennent exécutables, permettant de documenter efficacement les meilleures pratiques. L'automate devient une documentation vivante beaucoup plus efficace qu'un wiki, qui a tendance à mal vieillir.
g- Du temps est libéré sur les tâches répétitives, fastidieuses, complexes. La productivité s'améliore.

Et voici la liste des huit grandes familles d'activité du métier du test :

a- la stratégie de test et le maintien du référentiel des exigences et de la couverture des tests,
b- la préparation des tests (fiche de test, identification des besoins en jeux de données),
c- l'exécution des tests, ici la démarche RPA consiste en l'automatisation des tests,
d- la formalisation (collecte des preuves), le diagnostic (identification du partenaire de correction) et le suivi des dysfonctionnements (traçabilité, workflow),
e- le déploiement du logiciel,
f- le provisionning des jeux de données,
g- le provisionning, le monitoring et la maintenance des environnements de test,
h le pilotage des tests.

Au travers de cette liste, on comprend que l'on peut tirer bien plus de bénéfices de l'automatisation que ceux apportés par le test automatique, qui se concentre uniquement sur l'exécution des tests. C'est tout l'enjeu de la démarche RPA appliquée au métier du test.

3- La démarche RPA appliquée au métier du test

Par RPA, il faut comprendre la mise en place d'un ou plusieurs assistants virtuels (automates) qui permettent d'automatiser des tâches ne nécessitant pas de jugement humain. Ils sont mis en œuvre entre 3 et 7 semaines et sont construits pour interagir avec les applications existantes, en limitant leur empreinte sur l'infrastructure.

Le premier élément à prendre en compte dans la démarche RPA est probablement la conduite du changement. Les équipes en place sont fortement mobilisées pour produire leur travail quotidien. Cela leur laisse peu de temps pour intégrer un nouvel outil ou une nouvelle façon de travailler.

Le RPA doit automatiser en priorité les tâches qui sont réellement effectuées par l'équipe et qui rendent le quotidien fastidieux. Le gain sera ainsi clairement perçu comme libérateur et favorisera l'adoption des assistants virtuels.

Ensuite, il faut passer en revue l'ensemble des tâches qui sont :

	répétitives (mesurable par une fréquence)		fastidieuses (mesurable par une charge)
	complexes (mesurable par un nombre d'erreurs)		déléguées (mesurable par un délai de prise de charge)

Cette collecte des tâches avec un potentiel de gain doit s'effectuer au plus près de l'équipe, pour qu'elle soit le reflet de sa pratique. Il faut se méfier des schémas qui sont souvent le reflet de procédures idéalisées. Il existe quatre terrains propices au RPA dans les activités du test :

a- **La mise en condition de test**, l'identification ou la construction d'un jeu de données, par exemple. Cette partie peut représenter jusqu'à 40% du temps du testeur. Il est prioritaire de commencer par automatiser tout ce qui permet de se procurer des jeux de données : requêtes de ciblage, parcours de création de données sur les applications métiers, outil de rechargement. Ceci est d'autant plus indispensable que le besoin en jeux de données va être fortement augmenté, dès lors que des tests automatisés sont mis en place.

b- **L'exécution de tests fréquents** tels que les tests de régression. Il est souvent intéressant d'implémenter le RPA au sein de ces tests dans la mesure où ils sont répétitifs et ennuyeux. Le regard humain baisse en vigilance. Cependant, gardons à l'esprit que l'exécution du test ne concerne que 10% de l'activité d'un testeur. Il faut donc choisir les tests les plus fréquents ou ceux qui sécurisent un service incontournable du métier (une notification client, une facturation par exemple).

c- **La collecte des preuves, le diagnostic,** l'extraction des logs par exemple. Cette partie peut représenter jusqu'à 40% du temps du testeur.

d- **Le déploiement logiciel** est souvent source d'erreurs lorsqu'il est effectué manuellement. Positionné sur le chemin critique du projet, c'est une tâche prioritaire à sécuriser.

La présence de fichiers Excel avec plus de 100 lignes peut également être révélatrice d'un besoin d'automatisation. Il est humainement difficile de maîtriser un tel volume d'information.
Une fois la liste des tâches constituée, il faut la prioriser. Pour cela, le plus simple est d'affecter deux notes à chaque tâche :

- une note de gain de 1 (nice to have) à 10 (gain maximum pour l'équipe) estimée par l'équipe de test,
- une note de simplicité de fabrication de l'assistant virtuel de 1 (un démonstrateur est nécessaire) à 10 (une simple configuration d'un automate existant) estimée par un automaticien.

Le score de priorité est évalué en multipliant la note de gain par la note de simplicité. La liste priorisée des tâches/tests à automatiser constitue le backlog des travaux à engager.

Maintenant que l'on sait ce qu'il faut automatiser et dans quel ordre le faire, il reste à relever les quatre défis de l'automatisation.

4- Les 4 défis de l'automatisation pour le test informatique

Se lancer dans un projet d'automatisation nécessite de relever les 5 défis suivants :

- faire face au foisonnement technologique,
- comprendre les assistants virtuels,
- relever le défi de la maintenance,
- obtenir un feedback rapide,

4.1- Faire face au foisonnement technologique

L'assistant virtuel doit être capable d'interagir avec toutes les formes d'applications ou de services informatiques : applications web, ERP, applications bureautique, webservice, base de données, mainframe, traitement batch, IoT. Pour cela, l'offre RPA proposée par les acteurs historiques du marché est séduisante : UiPath, le leader actuel, Blueprism, un des premiers acteurs, Microsoft et son offre Power. Tous proposent des outils très puissants aux capacités techniques solides et qui ont fait leurs preuves, notamment dans les métiers du backoffice banque/assurance. Sont-ils pour autant bien adaptés au métier du test ? Le premier écueil est sans nul doute le coût de licence, alors que l'on souhaite que la phase de test soit la moins coûteuse possible. Le second écueil réside dans leur caractère « closed garden », qui rend l'assistant virtuel fermé aux technologies, en dehors de l'écosystème de l'éditeur. Le métier du test est fortement lié au produit logiciel qui est en constante évolution. La technologie sous-jacente à l'assistant virtuel doit pouvoir être étendue et ouverte en permanence. Ces contraintes poussent à adopter des technologies open-source, plus en phase avec la production du logiciel.

Il reste à choisir entre les trois tendances actuelles :

a- « **code** ». Dans cette configuration, l'assistant virtuel est un développement en langage informatique (java, C# par exemple). Il ne peut donc être réalisé et maintenu

que par des ressources qui maîtrisent la programmation informatique : les développeurs. Cela peut devenir un handicap dans la mesure où ces compétences sont coûteuses, pas toujours disponibles et n'ont pas forcément d'appétence pour ce type de développement. Mieux vaut les centrer sur l'apport de valeur que constitue la création logicielle.

b- « **no-code** ». La promesse est de réaliser un assistant virtuel, soit à l'aide d'un enregistreur, soit par une description sous la forme d'un diagramme d'enchaînement d'étapes. Bien souvent, il faut adapter cet enregistrement pour lui permettre de traiter plus de situations (l'adapter aux données, le paramétrer, inclure des règles de comportement). L'effort de maintenance est souvent important. La recommandation est de réserver cette approche « quick and dirty » pour quelques situations non pérennes (on fabrique rapidement un automate jetable).

c- « **low-code** ». Dans cette approche, on utilise un langage de script le plus proche possible du langage naturel. Il est possible de faire monter en compétence des ressources fonctionnelles pour réaliser ces assistants virtuels. De plus, le script étant du texte, il peut facilement être mis en configuration au plus près du logiciel qu'il teste.

Robot Framework, basé sur Python, est un exemple de produit open-source de type « low-code ». C'est le couteau-suisse de l'automatisation.

Il bénéficie d'une grande communauté d'utilisateurs et de nombreuses librairies extensibles. Il permet de définir des mots-clés (« keywords ») qui sont des phrases en langage naturel. Cela permet d'adresser notre second défi : comprendre les assistants virtuels.

4.2- Comprendre les assistants virtuels

Pour obtenir le bénéfice de la documentation vivante, il est indispensable que la description de l'assistant virtuel soit la plus lisible possible. Cela renforce la collaboration au sein de l'équipe. On évite l'apparition de frontières, par exemple entre le développeur et le testeur. Dans ce contexte, utiliser un outil favorisant le langage naturel est clairement le meilleur choix. A titre d'exemple, dix lignes de codes Java sont nécessaires pour effectuer le lancement d'un navigateur web. Cela se résume au mot-clé « open browser » avec Robot Framework.

Dans le cas particulier de l'automatisation des tests, il est recommandé d'utiliser la méthode BDD (Behavior Driven Development) pour rédiger les cas de test. Le test, quand il est écrit en langage Gherkin, est alors découpé en trois déclarations :

a- **Given** « étant donné un client grand public », c'est l'étape de prérequis au test, pour obtenir un compte client par exemple. Il est recommandé de commencer avec un article indéfini pour matérialiser une situation, un état,

b- **When** « lorsque le client consulte son compte », c'est l'événement que l'on veut tester. On utilise une phrase sujet/verbe/complément pour matérialiser l'action,

c- **Then** « alors le solde doit être présent », c'est le résultat attendu du test. Le « doit être » met en évidence l'attendu

Avec cette méthode, le test automatisé est lisible par tous les acteurs du projet : développeur, testeur, product owner, métier. Le test est vérifié en continu à chaque changement du logiciel : c'est une spécification vivante qui permet de fournir en permanence l'état de la qualité du logiciel. Le test devient la formalisation d'une exigence, ce qui donne l'opportunité de préparer le test au moment de la conception du logiciel : c'est la démarche shift-left. Le test est décrit avant la fabrication du logiciel et va servir d'étalon de validation. Le gain de temps et de qualité est significatif. Cette démarche de conception par le test permet de spécifier le comportement attendu avec des exemples concrets. L'équipe de conception reste ainsi dans le domaine du testable.

Il est essentiel que les tests automatiques soient fiables. Pour cela l'identification des faux-positifs est un élément-clé du succès. Il se trouve que la méthode BDD facilite ce diagnostic :

- L'échec de l'étape « Then » indique que l'objectif du test n'est pas atteint, alors que l'ensemble du scénario a été déroulé. Il y a un fort risque de bug dans ce cas.
- L'échec de l'étape « Given » ou « When » implique que le scénario ne s'est pas exécuté complètement. Il y a un risque de faux-positif associé à un problème d'environnement ou de jeux de données.

Les résultats de test doivent être disponibles, partagés et consolidés avec les tests manuels. Pour cela, il faut prévoir d'intégrer les résultats dans le référentiel des tests (par exemple avec JIRA/XRay). Cela permet de mettre en place un tableau de bord mis à jour au fil de l'eau des exécutions. La charge consacrée à la construction de reporting est alors libérée.
Disposer d'un script facilement compréhensible est aussi une première réponse au défi de la maintenance.

5- Relever le défi de la maintenance

Notre troisième défi est celui de la maintenance. La construction d'assistants virtuels ou de tests automatisés génère un effort de maintenance. Il s'agit d'adapter les scripts aux évolutions du produit logiciel testé ou de les rendre plus robustes. Cette charge peut vite devenir explosive avec l'augmentation du nombre d'automates. Dans le cas du test automatisé, il n'est pas rare qu'un test manuel aboutisse à la construction de cinq tests automatiques. En effet, pour faciliter le diagnostic d'un test, il est indispensable qu'il soit ciblé sur un seul objectif (un seul « Then »), alors que dans les tests manuels, il est fréquent que de nombreux contrôles soient effectués. On arrive vite à un patrimoine de plusieurs centaines de tests automatiques.

Fabriquer un automate est facile, en fabriquer des centaines maintenables est plus complexe. Il convient d'organiser la fabrication des automates en favorisant la réutilisation des mots-clés. La recommandation est de regrouper les mots-clés dans des bibliothèques aux responsabilités différentes :

a- le scénario de test, c'est l'histoire Given/When/Then,

b- les phrases métier, c'est le patrimoine automatisé qu'il faut gérer en bon père/mère de famille. L'idéal est de confier cette gestion à un responsable QA

c- les services applicatifs, ce sont des mots-clés d'action sur l'application métier à tester. Par exemple, se connecter, souscrire un contrat

d- les adaptateurs, c'est le socle technique commun à tous les projets. Par exemple, ouvrir le navigateur.

Cette méthode incite à la réutilisation des mots-clés. Il est possible d'atteindre des taux de réutilisation de l'ordre de 33%, c'est-à-dire qu'un mot-clé sur trois est utilisé plus d'une fois.

Avec des centaines de tests automatisés, les besoins en jeux de données sont fortement multipliés. Cela peut vite devenir une charge importante de maintenance. C'est le cas notamment lorsqu'on travaille avec des jeux de données statiques, spécifiques à chaque environnement de test. L'idéal est de construire des mots-clés qui vont permettre à l'automate d'obtenir son jeu de données automatiquement (jeux de données dynamiques).

Un fois la charge de maintenance maîtrisée, il est essentiel de s'assurer de la rapidité d'exécution des automates. C'est notre dernier défi, obtenir un feedback rapide.

6- Obtenir un feedback rapide

La réussite de l'automatisation est également liée à la rapidité du résultat. Une campagne de tests sur plusieurs heures rend difficile son adoption. L'automaticien doit constamment être vigilant sur la durée d'exécution unitaire. Cependant, cela ne suffit pas. Il faut être capable d'exécuter des centaines de tests en un minimum de temps. Cette productivité passe par la parallélisation. Encore une fois, la méthode BDD rend plus facile cette étape car elle permet de concevoir chaque cas de test de manière indépendante. Le regroupement Given/When/Then constitue un test autonome qui peut être parallélisé.

Ensuite, l'usine de tests doit fournir des capacités de parallélisation en :

- s'appuyant sur les technologies de containerisation (docker swarm, kubernetes),
- en utilisant des grilles de navigateurs on-premise (grille selenium, par exemple) ou dans le cloud (solutions éditeurs de type saucelab ou browserstack),
- en s'intégrant aux chaînes d'intégration continue (Jenkins, Gitlab CI, Azure devops).

On peut aussi agir sur le volume de tests en ciblant le « juste effort ». La méthode traditionnelle est de faire une stratégie de test manuelle, pour cibler l'effort de test sur les risques. L'automatisation peut aussi apporter des solutions pour cibler les tests à rejouer. Par exemple, on peut utiliser des outils d'analyse de couverture de tests, pour construire la carte des modules utilisés par chaque test automatique. Ainsi, au moment du déploiement du logiciel à tester, il est possible, grâce au gestionnaire de version, de déterminer les tests à rejouer sur la base des modules livrés.

Vers de nouvelles évolutions du métier du test

Comme nous venons de le voir, la mise en œuvre d'une démarche RPA permet de répondre aux nouvelles exigences du métier du test : « plus fréquents, plus massifs, plus fiables ». Dans un monde où les mises en service ont lieu toutes les heures, la tâche ne peut plus être réalisée entièrement par des humains.

L'automatisation est une réponse qui ne vient pas en concurrence du test manuel mais en complément. La machine permet d'adresser un volume de tests important avec un temps d'exécution maîtrisé, sans perdre en vigilance. Mais ces tests automatisés sont toujours identiques et sont vites atteints du paradoxe des pesticides : on laisse passer les bugs qui ne sont pas ciblés par les tests. Il est clairement indispensable de compléter la couverture de tests automatiques avec des tests exploratoires manuels, car seuls les humains sont capables de trouver ces nouveaux bugs. Autant nous perdons en vigilance lorsque nous répétons toujours la même tâche, autant nous sommes très performants lorsque nous réalisons une activité créative. C'est d'ailleurs la devise de UiPath : « Nous concevons des robots pour que les gens ne deviennent pas des robots ».

La machine est extrêmement performante pour traiter des masses d'informations et trouver des corrélations infimes. C'est tout l'enjeu de l'intelligence artificielle. Cette nouvelle brique technologique disponible en open-source, notamment en Python, agit également dans l'évolution du métier du test. Il sera possible d'anticiper des corrections, de prédire des bugs, de mieux diagnostiquer, de choisir les tests à effectuer.

Nous avons vu que le test en continu et le shift-left sont une réponse au time to market. Une nouvelle tendance se fait également jour avec le test en production : le shift-right. Il s'agit d'une extension du métier du test sur l'exploitation du logiciel : tester, monitorer, diagnostiquer, anticiper tout dysfonctionnement.

Plus que jamais, le test est au cœur de la maîtrise de la fabrication, du déploiement et de l'exploitation informatique : **un vrai métier**.

II.11- Automa'team : une communauté au cœur de l'agilité chez AXA France

par Mohamed Mehdi ADDOU, Jean-Prince Dotou-Segla, Etienne Dufour, Vincent Dugarin, Hélène Haing Droin, Thomas Kuypers, Bruno Pedotti et Huaxing Yuan

Retour d'expérience sur la stratégie d'automatisation des activités de test à l'échelle dans un environnement Agile

1- Les enjeux

Le marché de l'assurance et des services financiers est un marché mature et particulièrement concurrentiel.

Pour conserver sa position de leader, AXA France doit proposer des produits toujours plus innovants, en répondant aux exigences de ses clients en termes de qualité de service et de disponibilité.

Au fil des années, la DSI AXA France se transforme : plus agile, plus véloce, elle est aussi plus performante. Plus que jamais, elle contribue à fournir des solutions applicatives simples, robustes et différenciantes.

Pour cela, après le passage à l'Agile à l'échelle (Scrumban) en 2011, la DSI a déployé le modèle d'organisation en feature teams en 2016.

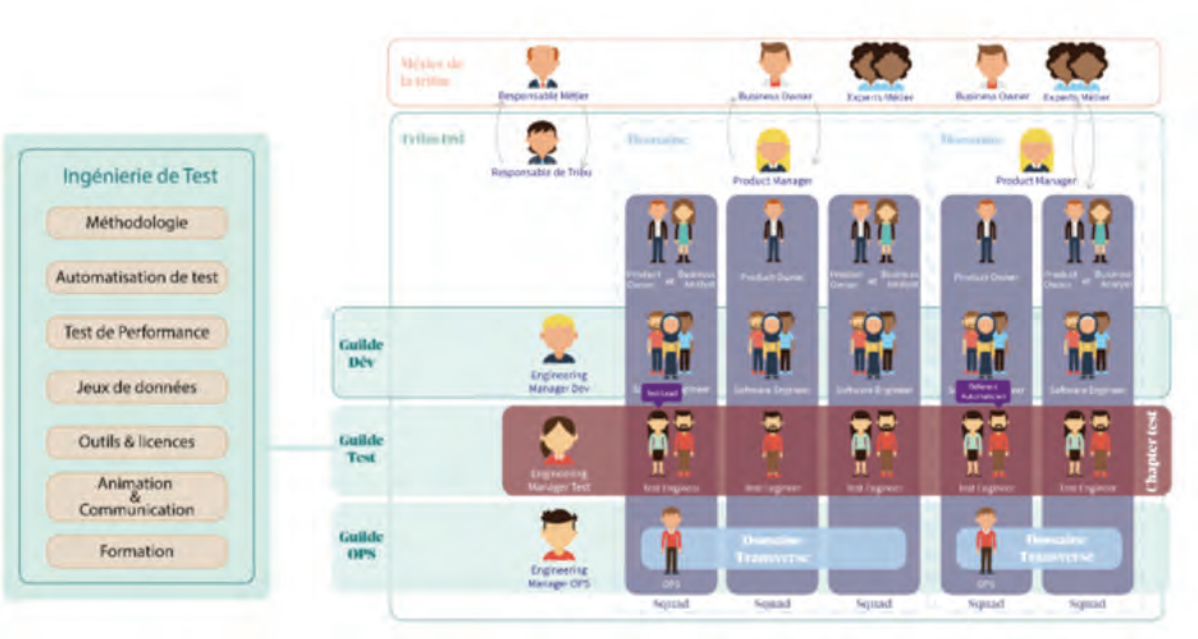


Figure 1: Organisation d'une tribu

Dans ce modèle, la Guilde Test, direction transverse à l'ensemble des tribus, accompagne les équipes agiles dans leurs activités opérationnelles, pour garantir la qualité des solutions informatiques et mettre à disposition les moyens de test (jeux de données, postes et matériels mobiles de test).

Sa vocation est de développer les compétences et pratiques de test des collaborateurs à un haut niveau d'expertise, en s'appuyant sur l'état de l'art du marché. Elle conseille, sensibilise et développe les compétences test au sein de la DSI d'AXA France.

Dans le cadre du déploiement de pratiques d'ingénierie DevOps, l'enjeu principal de la Guilde Test est de contribuer à réduire le Time to market, tout en étant intransigeant sur la qualité des solutions. Cela passe inévitablement par l'automatisation des activités de test.

La communauté de pratiques automatisation "Automa'team" de la Guilde Test partage dans ce chapitre sa stratégie, l'architecture technique qui supporte son déploiement, l'animation de ses expertises ainsi qu'un retour d'expérience qui n'attend que d'être enrichi.

2- Les activités et l'architecture d'automatisation des tests

2.1- Les activités de test

Les activités de test sont alignées sur les éléments définissant les produits. Dans le contexte AXA, les nouvelles versions des produits sont décomposées en Minimal Marketable Features (MMF), elles-mêmes décomposées en User Stories (US) implémentées sous forme de composants. Chaque composant implémente une ou plusieurs fonctions. L'ensemble de ces éléments sont testés de façon manuelle ou automatisée.

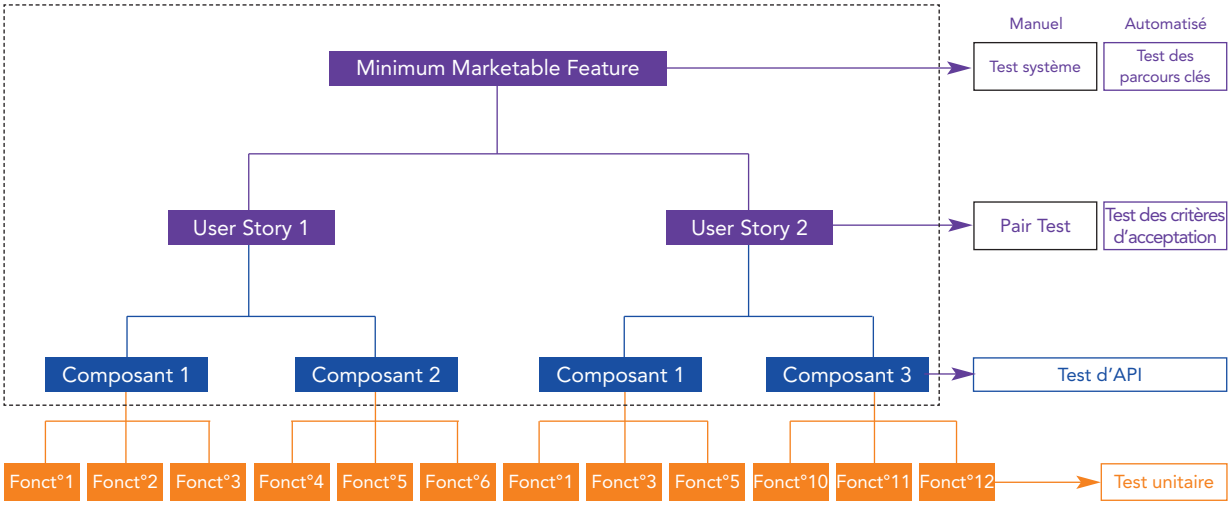


Figure 2: Activités de test manuel et d'automatisation

En partant du bas de la pyramide :

- Les fonctions sont testées par les tests unitaires des développeurs
- Les composants sont testés manuellement ou de façon automatisée par les ingénieurs de test (tests des APIs / services web)
- Les User Stories sont présentées et analysées pendant les séances des 3 amigos pour en définir les critères d'acceptation. Ceux-ci seront testés de façon manuelle (pair test) ou automatisés par les développeurs (approche BDD).
- Au niveau MMF, les scénarios de test système sont identifiés et évalués avec le Product Owner et l'équipe de développement. Ils sont testés manuellement. Les scénarios ayant une forte valeur métier sont identifiés comme parcours clés. Leurs tests sont automatisés.

2.2- L'Architecture d'Automatisation des Tests

Pour répondre à ces besoins d'automatisation et s'assurer que les tests peuvent être exécutés de manière cohérente, AXA France a défini et mis en place son Architecture d'Automatisation des Tests (AAT). Elle est basée sur le Cloud et s'intègre à la Plateforme d'Intégration Continue (PIC).

Cette architecture se compose de 5 groupes d'éléments : Station de travail, Environnement DevOps, Plateformes d'exécution (hébergées sur le cloud public et privé), le Système sous test et le Gestionnaire de test.

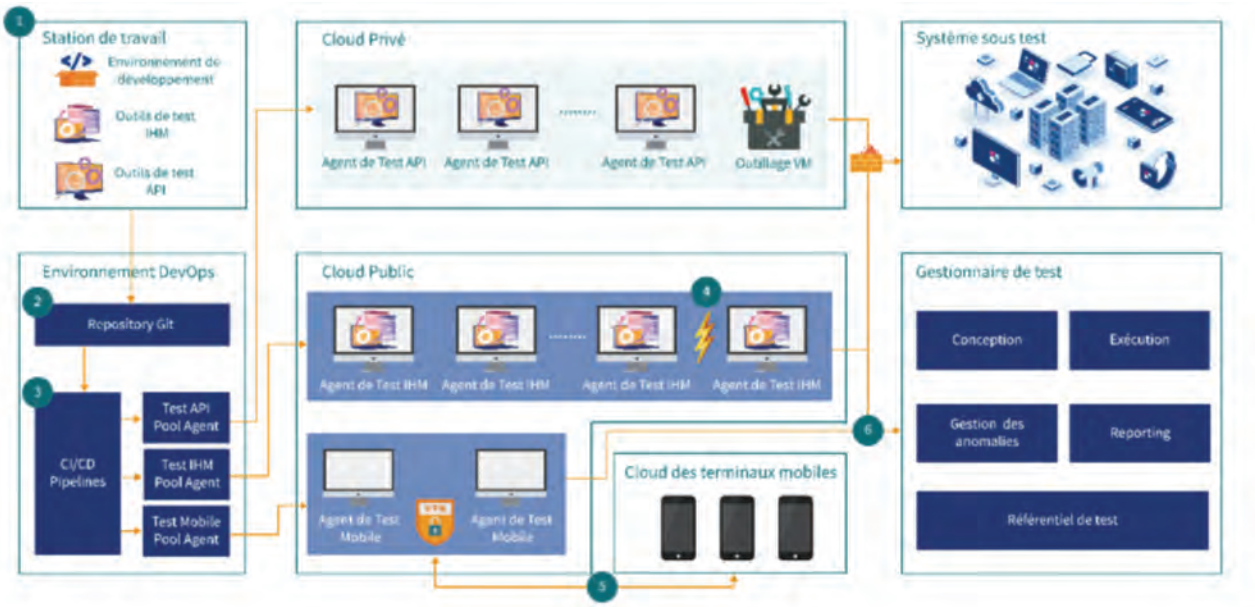


Figure 3 : Architecture d'Automatisation des Tests (AAT)

Station de travail [1] **:** c'est l'environnement regroupant tous les outils de développement et de test. C'est sur cette machine que l'ingénieur de test développe et maintient la solution d'automatisation.

Environnement DevOps : dans cet environnement, l'ingénieur de test stocke la solution de test automatisé dans un référentiel centralisé GIT (2). La solution y est versionnée et synchronisée avec le système sous test. La procédure de test (préparation de l'environnement, déploiement de l'automate, exécution des tests et intégration des résultats dans le gestionnaire de tests) est automatisée via le pipeline de déploiement (3). Les tests automatisés sont exécutés de façon régulière et automatique tout au long du développement du projet. 3 pools d'agents sont mis en place pour centraliser l'exécution des différents tests sur les plateformes dédiées : Test API, Test IHM et Test Mobile.

Plateformes d'exécution : les machines d'exécution sont provisionnées et hébergées sur le cloud. Elles sont connectées avec le pool d'agents. La tâche d'exécution de test sera prise en charge par une machine disponible, le résultat de test sera remonté dans le gestionnaire de test.

- Plateforme de test API (cloud privé) : les tests d'API interrogent les services web middleware et backend. Pour des raisons de sécurité, les machines d'exécution sont hébergées sur le cloud privé.

- Plateforme de test IHM : les tests IHM sont de gros consommateurs de temps CPU. Avec l'intégration et le déploiement continus, il peut y avoir une grande fluctuation du nombre de tests à exécuter pendant la journée. Pour gérer cette contrainte, AXA a développé une solution d'auto-scaling basée sur le cloud (4). Ce service surveille les suites de test à exécuter (dans les files d'attente) et compare avec les ressources machine disponibles. En cas de forte charge, le système provisionne automatiquement les nouvelles machines et les configure. Les tests en attente sont exécutés le plus rapidement possible. En cas de faible charge, les machines inactives sont arrêtées ou supprimées. Cette solution permet d'exécuter un grand nombre de suites de tests simultanément, tout en maîtrisant le budget de l'infrastructure

- Plateforme de test Mobile : pour les tests mobiles, les scénarios sont exécutés via un pool de téléphones mobiles hébergé chez un fournisseur externe. Afin de sécuriser les environnements de test et de ne pas exposer le système sous test sur Internet, un VPN temporaire est activé entre le terminal et la machine d'exécution (5). Avec cette fonctionnalité, le Système sous test (SUT) n'est accessible que durant l'exécution de tests automatisés.

Système sous test (SUT) : c'est le système à tester. Il est accessible depuis le réseau interne AXA ou depuis la machine d'exécution. Il est protégé par un système de firewalls pour respecter la politique de sécurité d'AXA.

Gestionnaire de test : il contient le référentiel pour tous les tests (manuels et automatisés), les résultats d'exécution, les anomalies et les exigences. Il assure le reporting de test. Les résultats des tests automatisés sont systématiquement et automatiquement intégrés au gestionnaire (6). Cela garantit la traçabilité entre les tests et les résultats.

3- La traçabilité des tests

La totalité des tests, qu'ils soient manuels ou automatisés, du test unitaire jusqu'au test de parcours clé, sont conçus ou remontés dans le gestionnaire de tests. Ainsi, la traçabilité est totale sur l'ensemble de la chaîne de test.

Pour la remontée des tests automatisés, une tâche intégrée à la plateforme d'intégration continue assure la mise à jour du référentiel de tests, avant et après l'exécution des tests. Elle s'appuie sur les APIs natives du gestionnaire de test.

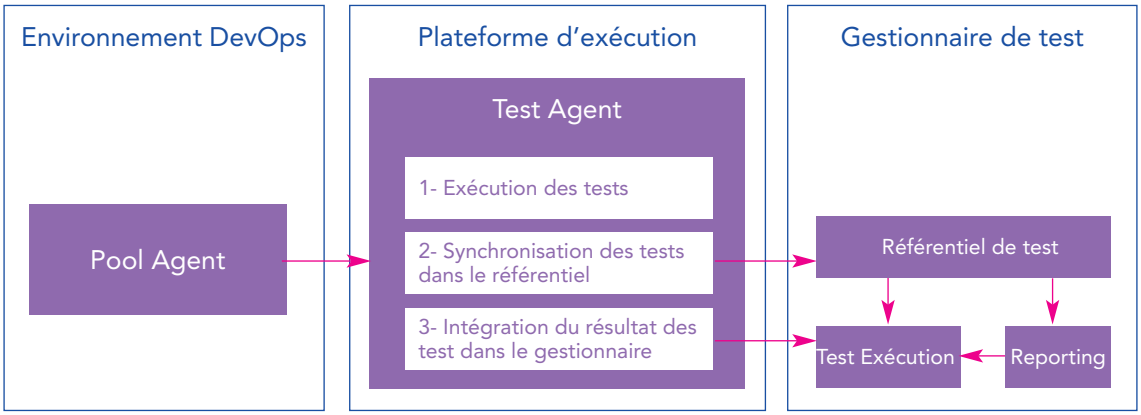


Figure 4 : synchronisation des cas de test et intégration de résultat dans le référentiel

Cette intégration continue renforce la synergie entre les développeurs (concevant les tests unitaires) et les ingénieurs de test (écrivant les tests fonctionnels de bout en bout). Et au-delà, elle fédère l'ensemble de la squad autour des résultats des tests.

4- Les indicateurs de performance

Des indicateurs ont été définis pour suivre le déploiement des pratiques d'automatisation des tests et leur efficacité. Ils sont mesurés et analysés périodiquement dans le cadre de l'amélioration continue des pratiques.

Dimension	Mesure
Niveau de déploiement et maturité des pratiques d'automatisation des tests	Evaluation périodique à partir d'un modèle de maturité
Avancement de l'automatisation	Ratio tests automatisés / tests à automatiser
Taux d'automatisation	Ratio tests automatisés / tests manuels
Efficacité des tests automatisés	Nombre de défauts identifiés par les tests automatisés
Retour sur investissement	Gain d'exécution / coût de l'automatisation Gain sur le TTM grâce aux tests automatisés

5- L'animation des compétences

Un dispositif complet est en place au sein de la Guilde Test pour animer les compétences liées à l'automatisation des tests. Il s'intègre au dispositif global de la Guilde.

En quelques mots...

Formation et certification : des parcours d'accueil des nouveaux entrants ont été formalisés. L'un deux est propre aux ingénieurs de test automaticiens (présentation de l'AAT, du framework, des outils de test API et IHM). A l'issue du parcours, une certification ISTQB est proposée. Par ailleurs, ces ingénieurs de test bénéficient des mêmes formations transverses que les autres membres de la Guilde (techniques de conception, stratégie de test, gestionnaire de test, …).

Communautés de pratiques (COP) : une communauté de pratiques d'une cinquantaine d'ingénieurs de test spécialisés en automatisation, l'Automa'team, est en place. Animée par ses membres, elle permet de diffuser les pratiques et de les améliorer, sur la base de l'expérience terrain. C'est le laboratoire de veille et d'innovation.

Animation de la Guilde : le thème de l'automatisation des tests est pleinement intégré à la vie de la Guilde. Plusieurs canaux sont utilisés : plénières, webinaires, newsletter, DOJOs, … Autant de moyens d'élever le niveau d'expertise des équipes !

Equipe transverse "Ingénierie" : une équipe de sept personnes joue un rôle de garant des pratiques, de support autour des outils. Deux consultants sont dédiés à l'automatisation.

6- Retours d'expérience

6.1- L'automatisation des tests du site Mon AXA : un succès

Le site Mon AXA abrite l'espace privé des clients AXA France. Il permet la gestion des contrats pour toutes les lignes de métier (assurance automobile, habitation, assurance-vie, …). C'est l'une des vitrines d'AXA. A ce titre, sa qualité et sa disponibilité sont essentielles.

L'automatisation des tests de Mon AXA a commencé en 2018. A ce jour, elle peut être considérée comme un succès pour quatre raisons.

Une collaboration étroite entre métier, ingénieurs de test et développeurs

A ce jour, plus d'une centaine de parcours sont automatisés. Ils ont été identifiés conjointement entre le métier et les testeurs, lors des réunions 3 amigos ou des réunions de suivi hebdomadaires. Cette démarche garantit la transparence des choix.

Une fois les parcours clés sélectionnés, des réunions de synchronisation avec les développeurs sont organisées pour identifier les éléments techniques qui feront partie des tests à automatiser (boutons, zones texte, …). Une coordination qui améliore l'efficacité et réduit le TTM.

Un framework de qualité

L'automatisation s'appuie sur un outil du marché intégré dans un framework d'automatisation développé par AXA France. Ce framework a été conçu de façon structurée et modulaire. Il a été clairement documenté. Il est jugé comme facile d'utilisation par les ingénieurs de test. La structure du framework permettra de faire face aux futurs défis de l'automatisation.

Une couverture des tests automatisés optimale

Les combinatoires de navigateurs / OS smartphones ont été définies avec le métier en s'appuyant sur les usages des internautes. Pour chaque item, la dernière version du marché et les deux dernières sont utilisées pour les tests. Pour limiter les tests, chaque test n'est exécuté que sur une combinaison.

La disponibilité de l'environnement d'automatisation

Les tests automatisés s'exécutent tous les jours ouvrés. Un des objectifs est d'éviter au maximum les échecs de test pour cause d'indisponibilité de l'environnement d'exécution. La bonne exécution des tests est garantie de deux façons :

Les tests sont exécutés dans le cloud, au travers d'agents créés dynamiquement. Ce mode de création permet de garantir la disponibilité de toute l'infrastructure du produit : l'application elle-même, les services sur lesquelles elle s'appuie, l'environnement d'exécution des tests ainsi que les outils nécessaires à l'automatisation.

Des scripts API sont exécutés pour vérifier la disponibilité des services web appelés par le site web.

6.2- Mise en œuvre du premier framework d'automatisation : des erreurs de jeunesse

La première version du framework d'automatisation a été mise en œuvre sur un projet-pilote en 2016. Ce pilote a permis de tirer les premières leçons sur l'automatisation.

La stratégie initiale du projet était d'automatiser le plus grand nombre possible de parcours clés. Par manque d'une analyse de la testabilité du système applicatif, certains parcours n'ont pas pu être automatisés jusqu'au bout.

Pour les parcours clés automatisés, la dimension « fonctionnement » n'avait pas été anticipée. Cela couvrait la maintenance des scripts, les tâches d'analyse du reporting d'exécution et la gestion des anomalies ainsi que le provisionnement des budgets associés.

Ces deux aspects ont été améliorés pour les itérations suivantes. Des critères de testabilité et d'« automatisabilité » ont été définis et les abaques ont été améliorées.

7- Les perspectives

L'accélération du delivery implique l'industrialisation d'activités de test supplémentaires et l'évolution des compétences des ingénieurs de test.

Côté pratiques, on pense notamment à la production de reporting, à l'industrialisation de la gestion des données de test et à la digitalisation de certains livrables.

Pour illustrer, la Guilde Test s'est dotée d'un outil d'extraction, anonymisation et injection de données de test, supporté par un outil web de requêtage et sélection de jeux de données. Son déploiement est en cours sur toutes les technologies.

L'industrialisation de la production des PV de test, basés sur les rapports d'exécutions aussi bien automatisés que manuels et signés numériquement, est une autre piste d'optimisation.

Sur les compétences, il faut accompagner l'évolution du métier d'ingénieur de test dont la dimension technologique ne cesse de prendre de l'importance. L'ambition de la Guilde Test de la DSI AXA France est de faire de l'expertise de ses collaborateurs un levier différenciant, au service des enjeux métiers.

La progression de l'automatisation doit également nous interroger sur l'évolution à apporter aux activités de test manuel, ainsi que sur l'utilisation de certaines phases de test (recette) à des fins d'appropriation métier. Mais le métier est-il prêt à lâcher prise ?

II.12- Priorisation de l'exécution des tests par l'apprentissage automatique

par Alexandre Vernotte et Aymeric Cretin

Ce chapitre traite des techniques de priorisation des tests à l'exécution et de l'utilisation de techniques d'apprentissage automatique pour automatiser cette activité de test.

1- Pourquoi prioriser ?

Dans une chaîne d'intégration continue, les changements réalisés dans le code par les développeurs sont fusionnés quotidiennement dans la base de code principale. Les tests de différents niveaux sont exécutés dès qu'un changement est soumis à cette dernière. Un des aspects clés d'une intégration continue efficace est la rapidité du retour d'information, entre le commit du code et les rapports d'exécution de test. Parfois, le temps dédié à l'exécution de tests est limité, il est alors essentiel de trouver le bon équilibre entre l'exécution des tests les plus susceptibles de révéler des fautes et la maximisation du nombre de tests pouvant être exécuté dans le temps imparti. La priorisation de cas de test correspond à la recherche de cet équilibre. Son but est d'avertir les développeurs de l'introduction d'une faute dans le code le plus tôt possible, pour leur permettre de se concentrer sur sa résolution avant de continuer à produire de nouveaux changements dans le code. La réduction du temps d'exécution de la suite de test est également un critère important de la priorisation. En effet, parmi les cas de test qui ont un taux élevé d'échec, l'ordonnancement doit être fait de telle manière que les tests les moins long s'exécutent en priorité.

Si la priorisation des cas de test ne semble pas utile pour les suites de test ne prenant que quelques minutes pour s'exécuter, elle devient cruciale lorsqu'une suite de test se compose de plusieurs milliers de cas de tests et/ou dont l'exécution prends plusieurs heures, voire plusieurs jours. Dans ces situations, la priorisation peut permettre de révéler aux développeurs la présence d'une faute, plusieurs heures ou plusieurs jours à l'avance par rapport à une exécution non priorisée.

2- État de l'art de la priorisation des tests

De nombreuses approches et outils ont été proposés lors des 25 dernières années. Ces approches peuvent être catégorisées en trois familles, dont nous donnons un rapide aperçu ci-dessous.

Prioriser en favorisant la diversité

Une première famille d'approche part du constat qu'il est possible d'accélérer la détection de fautes, en assurant la diversité des cas de test. C'est-à-dire en exécutant tour à tour un cas de

test qui se veut différent de ceux exécutés précédemment. Le principe est de couvrir au plus vite les différentes façons d'exercer le système, afin d'atteindre une détection plus rapide des fautes. Pour différencier les tests, certaines approches issues de cette famille s'appuient sur le calcul des différences de code source entre les cas de test en utilisant, par exemple, les distances de Levenstein, Manhattan, etc . Dans la même famille, certaines approches utilisent d'autres caractéristiques des cas de test : couverture de code, complexité, etc[1]., afin de réaliser un *clustering*. Le but du *clustering* étant de proposer un ordre basé sur le choix de représentant *cluster* par *cluster*[2] , ou d'alimenter des algorithmes bio-inspirés comme l'algorithme des lucioles[3].

Prioriser par utilisation de l'historique d'exécution

La seconde famille propose de prédire le verdict à venir d'une suite de test, grâce à l'historique de ses exécutions. L'hypothèse est la suivante : un cas de test ayant souvent mis en lumière une défaillance du système sous test par le passé sera plus à même de rendre un verdict négatif par la suite. D'autres métriques peuvent participer à l'anticipation du verdict : âge du cas de test, durée d'exécution ou date de dernière exécution, par exemple. La technique la plus naïve consiste à exécuter en premier les tests dont le dernier verdict est négatif, ou ceux qui ont le plus de verdicts négatifs sur les *N* exécutions passées. Les techniques les plus abouties reposent sur des heuristiques qui, par exemple, donnent davantage de poids aux verdicts les plus récents, pour un coût de mise en pratique quasi-identique[4].

Prioriser en fonction des changements réalisés sur le code et de la couverture du code

La dernière famille exploite les relations entre le contenu du commit, le code testé et le code du test. Beaucoup d'approches utilisent, par exemple, la couverture du code par les tests, de sorte à prioriser les cas de tests ayant la plus grande couverture : plus un cas de test couvre d'instructions, plus il est à même de couvrir une instruction contenant une faute. Différents critères de couverture sont la base d'approche de priorisations, comme la couverture d'instructions, de fonctions, mais aussi MC/DC[5]. D'autres approches s'appuient plutôt sur la nature des changements apportés entre le dernier build et le prochain. Toujours par mesure de couverture du code par les tests, l'objectif est de mettre en avant les cas de test couvrant des instructions impactées par l'évolution récente du code[6].

Des techniques dites « hybrides » tentent de combiner certaines des familles précédentes, de sorte à profiter des avantages de chacune d'elles. C'est le cas des approches à base de *Machine Learning* (ou apprentissage automatique), dont le principe de fonctionnement permet d'intégrer des sources de données conséquentes et variées, afin d'identifier des relations complexes entre ces données et d'en déduire le verdict à venir de chaque test. L'entreprise SalesForce a publié une approche de priorisation des cas de test, combinant ainsi couverture de code, historique

[1] https://link.springer.com/article/10.1007/s10515-011-0093-0
[2] http://cs.ndsu.edu/~hdo/pdf_files/icsm11.pdf
[3] https://ieeexplore.ieee.org/abstract/document/8830334/
[4] https://ieeexplore.ieee.org/abstract/document/6676952/
[5] https://ieeexplore.ieee.org/abstract/document/1183927/
[6] https://www.hindawi.com/journals/archive/2016/7132404/

d'exécution et similarité entre les tests[7]. Plus récemment, des approches à base d'apprentissage par renforcement (RL pour *Reinforcement Learning*) ont vu le jour et se montrent prometteuses , malgré leur caractère encore très expérimental.

3- Apports de l'apprentissage par renforcement

L'application de techniques de RL au problème de priorisation de cas de test rencontre un intérêt grandissant. Les dernières études révèlent que le RL a démontré de meilleures capacités de priorisation, comparé aux autres algorithmes issus de l'état de l'art, notamment en termes de vitesse d'adaptation à des suites de tests qui évoluent fréquemment, pour peu que les informations à propos des cas de test soient fournies à l'agent de RL de façon suffisante.

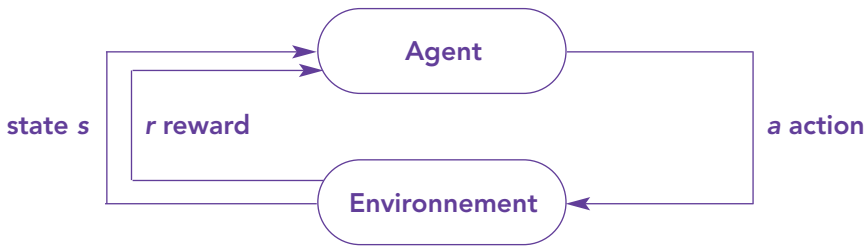


Figure 1 : Schéma de principe de l'apprentissage par renforcement

Un entraînement typique de type RL se compose de deux entités : un agent et un environnement. L'agent observe d'abord l'environnement puis prend en compte son état actuel *s*. Ensuite, il fait évoluer l'environnement vers l'état *s+1* en réalisant une action *a*. Au changement d'état, l'environnement délivre une récompense *r* à l'agent, dont la valeur dépend de la pertinence de l'action a réalisée. Par exemple, dans un environnement de type labyrinthe, l'agent reçoit sa position actuelle et choisit une direction de progression. Selon la direction choisie, la récompense délivrée par l'environnement peut correspondre à la distance restant à parcourir pour sortir du labyrinthe. Généralement, un environnement possède un ou plusieurs états finaux (la sortie du labyrinthe) qui, lorsqu'ils sont atteints, marquent la fin d'un épisode, c'est-à-dire la séquence d'interactions agent-environnement depuis un état initial vers un état final. L'objectif de l'agent est de maximiser la récompense *cumulée* à travers un épisode. Il s'agit alors pour un état donné de choisir l'action permettant de maximiser la récompense immédiate, ainsi que la récompense cumulée anticipée.

De nombreux algorithmes peuvent réaliser cette tâche mais la méthode la plus simple est celle du *Q-Learning* : l'agent collecte, au moyen d'un tableau à deux dimensions (une pour les états et l'autre pour les actions), la récompense cumulée qu'il a obtenu en prenant une certaine action à partir d'un certain état. Le tableau est d'abord rempli en explorant l'espace d'actions, c'est-à-dire en choisissant au hasard une action à chaque état rencontré. Une fois le tableau rempli, la proportion d'exploration est réduite et un algorithme glouton est privilégié (i.e. choisir systématiquement la meilleure action à un état donné, en se référant au tableau).

[7] https://taoxie.cs.illinois.edu/publications/fse16industry-learning.pdf

Dans le contexte de priorisation des cas de test, l'environnement consiste en un processus d'intégration continue qui, en amont de l'exécution des tests, fournit à l'agent les données liées à chaque cas de test de la suite à prioriser. Le résultat de la priorisation est un classement des tests, utilisé par l'environnement pour déterminer leur ordre d'exécution. Les échanges entre agent et environnement peuvent prendre différentes formes. L'agent peut recevoir les tests un à un, assignant à chacun un score de priorité, ou il peut recevoir l'entièreté des données, et déterminer le cas de test à exécuter en premier parmi la liste.

L'approche de priorisation RETECS[8], issue du laboratoire norvégien SIMULA à Oslo, a suscité l'engouement récent de l'utilisation du RL pour la priorisation des tests. RETECS est une approche basée sur un réseau de neurones (l'agent), qui reçoit en entrée les données d'un cas de test et retourne un score de priorisation. Les données liées au cas de test fournies à l'agent sont les 4 derniers verdicts d'exécution, la dernière durée d'exécution du cas de test et le temps écoulé depuis sa dernière exécution. Des résultats prometteurs ont amené l'industrie à s'intéresser au RL pour la priorisation des tests, notamment avec Netflix, qui a annoncé avoir implémenté sa propre version de RETECS[9].

Une seconde étude, conduite par Bertolino et al.[10], s'appuie sur la proximité entre la priorisation des cas de test et le domaine de la Recherche d'Information (ou IR pour *Information Retrieval*), qui consiste à classer des documents pour faire ressortir les plus pertinents. L'étude oppose les algorithmes issus de l'IR aux algorithmes RL, pour résoudre le problème de priorisation des tests. L'expérimentation démontre la capacité des algorithmes RL à s'adapter à des suites de test changeantes et montre des résultats supérieurs aux meilleurs algorithmes IR. Une troisième étude, conduite par Bagherzadeh et al.[11], reprend le même protocole d'expérimentation que Bertolino et al. (jeu de données et algorithmes de priorisation). L'objectif est de confirmer les résultats obtenus précédemment, grâce à une librairie nommée *Stable-Baselines*, qui implémente les neuf principaux algorithmes RL issus de l'état de l'art. Les résultats montrent que le RL permet globalement d'obtenir une priorisation plus pertinente que les algorithmes issus de l'IR, à condition de disposer de suffisamment d'informations concernant les cas de test, notamment du point de vue de la couverture de code.

4- rl4tcp : Priorisation des tests via apprentissage par renforcement

rl4tcp (*Reinforcement Learning for Test Case Prioritization*) est un prototype de recherche développé en partenariat entre l'Université de Franche-Comté et Smartesting, ayant comme point de départ le code proposé par l'étude de Bagherzadeh (citée précédemment).

Notre objectif est double. D'une part, rl4tcp vise à étudier en parallèle les différents algorithmes de RL utilisés pour prioriser des tests, et la pertinence des caractéristiques utilisées pour « décrire »

[8] https://arxiv.org/pdf/1811.04122

[9] https://netflixtechblog.com/lerner-using-rl-agents-for-test-case-scheduling-3e0686211198

[10] https://dl.acm.org/doi/abs/10.1145/3377811.3380369

[11] https://arxiv.org/pdf/2011.01834.pdf

ces tests. D'autre part, rl4tcp a vocation à être expérimenté et évalué pour prioriser les tests dans un contexte d'itération continue réel. Cette section présente d'abord l'architecture de l'approche, puis détaille le fonctionnement du module de RL. Enfin, elle aborde les données utilisées par notre approche pour mener à bien la priorisation.

4.1- Approche générale

Notre approche vise à s'intégrer pleinement dans une chaîne d'intégration continue (CI pour *Continuous Integration*) existante. Pour permettre cela, elle s'appuie sur l'architecture présentée dans la Figure 1. Cette dernière schématise une chaîne de CI classique, où l'envoi du nouveau code dans la base principale déclenche un job sur le serveur de CI. Dans cet exemple, un job typique se compose de plusieurs tâches de base, en bleu foncé : *build, test, analyze, notify* et *deploy*, auxquelles ont été ajoutées deux nouvelles tâches en bleu clair : *prioritize* et *extract*, respectivement dédiées à la priorisation et à la récupération des caractéristiques. Afin d'intégrer ces nouvelles tâches aux jobs de manière transparente, elles s'appuient sur l'utilisation de modules conteneurisés : *RL Module* et *Features extractor*. Ces deux modules, qui sont les piliers de l'approche, s'exécutent en effet dans des conteneurs Docker. Cette isolation leur permet d'être indépendants de l'environnement du serveur de CI. Ainsi, les deux nouvelles tâches se résument à une simple ligne de commande à exécuter durant le *job*. Enfin, le composant « *Test suite Dataset* » correspond aux données utilisées par le module de priorisation pour ordonner les cas de test, il s'agit d'un volume partagé entre les conteneurs. Finalement, le rôle du module *Feature extractor* est de fournir des données au module de RL afin que celui-ci réalise la priorisation.

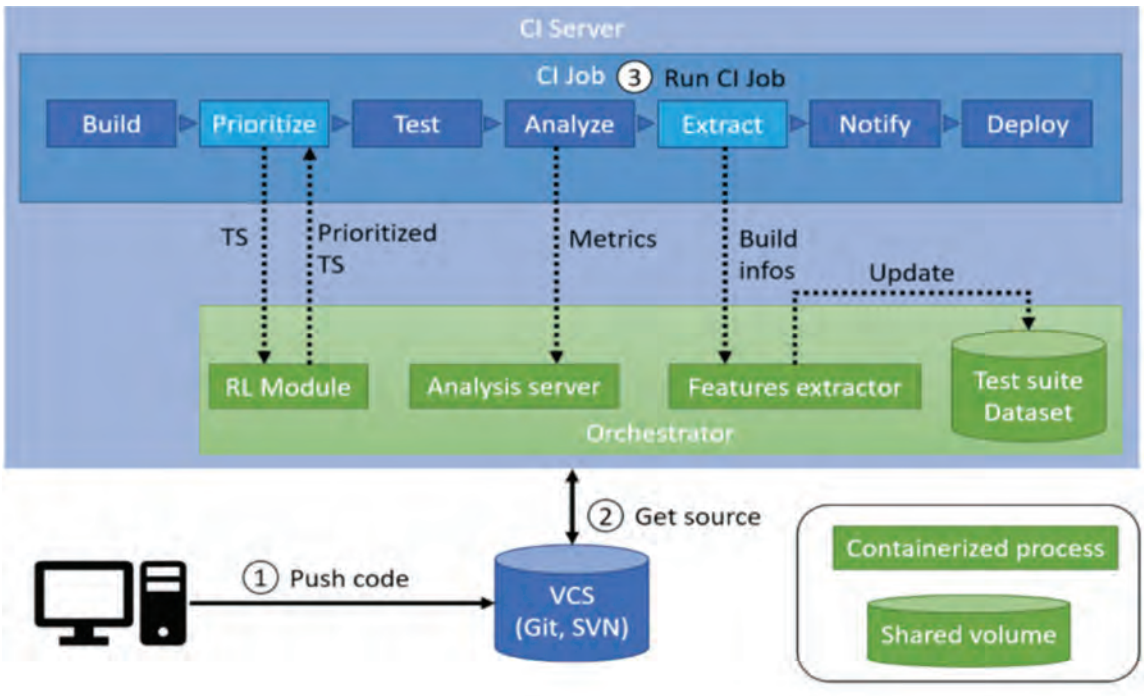


Figure 2 – Mise en œuvre de l'approche sur une chaîne d'intégration continue

Les sous-sections suivantes présentent les deux modules piliers de rl4tcp. D'abord, le module de RL est abordé en discutant les modèles utilisés et les choix de modélisation concernant les environnements. Puis, pour le module *Feature extractor*, la sous-section suivante détaille la façon dont chaque caractéristique est collectée.

4.2- Module RL

Cette section décrit plus en détail le module RL, qui est représenté en Figure 2. C'est au sein de ce module que les travaux de Bagherzadeh et al. sont intégrés puis étendus. Les différents algorithmes utilisés dans l'approche sont d'abord discutés puis les trois environnements sont décrits en détail.

4.2.1- Algorithmes

La librairie Stable-Baselines v3 regroupe l'implémentation de sept algorithmes RL issus de l'état de l'art et sélectionnés pour leur efficacité. On retrouve par exemple Deep Q Learning, l'algorithme de DeepMind qui a démontré la capacité des algorithmes RL d'apprendre à jouer à un jeu vidéo par eux-mêmes. La librairie permet de s'interfacer avec la librairie Gym[12] qui rend possible l'implémentation d'environnements RL.

Chaque algorithme possède un ensemble d'hyperparamètres qui définissent son fonctionnement : sa rapidité de convergence, l'importance qu'il convient de donner à la récompense cumulée anticipée, etc. Si certains algorithmes sont réputés pour donner de bons résultats sans configuration particulière, une bonne configuration des hyperparamètres s'avère indispensable dans certaines situations. Dans *rl4tcp*, l'optimisation des hyperparamètres est confiée à Optuna, qui confronte différentes configurations aux données d'exécution des tests. Cette optimisation ne s'effectue qu'à l'intégration de l'approche et n'a, pour l'heure, pas besoin d'être mise à jour.

4.2.2- Environnements

Dans le domaine de l'IR, on distingue trois types d'algorithmes. Il y a d'abord ceux qui traitent un document à la fois, en leur donnant un score d'importance, qu'on appelle *pointwise*. Il y a ensuite ceux qui comparent deux documents, pour indiquer celui qui est le plus important, qu'on appelle *pairwise*. Il y a enfin ceux qui comparent l'entièreté des documents, pour identifier le ou les plus importants, qu'on appelle *listwise*. Notre approche emprunte fortement au domaine de l'IR, on retrouve donc naturellement ces trois catégories sous forme d'interactions agent-environnement, et non sous forme d'algorithme. L'agent se charge uniquement de l'évaluation des données. De son côté, l'environnement construit la suite de test priorisée en fonction des actions de l'agent et le récompense selon la qualité de l'ordonnancement qui en résulte Ainsi, avec l'environnement pointwise, l'environnement fournit à l'agent les informations liées à un seul cas de test et celui-ci répond par un score d'importance, tandis qu'avec l'environnement listwise, l'environnement transmet les informations de tous les cas de test à la fois à l'agent, qui sélectionne celui qui semble être le plus à même de détecter une faute.

[12] https://gym.openai.com/

Une difficulté supplémentaire, lors de l'élaboration des environnements, est de définir la fonction de récompense, de sorte à pouvoir guider l'apprentissage de l'agent. Dans notre cas, puisque l'apprentissage s'effectue sur des données d'exécution de cycles CI passés, le verdict de chaque cas de test est connu à l'avance. A partir de cette information, il est possible d'élaborer une priorisation optimale des cas de test, ce qui constitue l'objectif d'apprentissage de l'agent. La récompense peut alors se décider, par exemple pour l'environnement *pointwise*, en réalisant la différence entre le rang du cas de test dans la priorisation optimale et le score d'importance déterminé par l'agent.

Notons que la présence de différents environnements, pour la réalisation d'un même objectif, s'explique par le besoin d'expérimentation encore nécessaire à la validation de l'approche, pour déterminer si un environnement est plus performant que les deux autres, ou qu'un environnement est plus efficace selon les caractéristiques de la suite de test (nombre de cas de test, nombre de cas de test détectant une faute, etc.).

4.3- Collection des données

Afin de permettre l'entraînement des agents de RL, il convient de récupérer ce qu'on appelle les « données initiales ». Elles correspondent aux données minimales nécessaires à la mise en œuvre de l'approche de priorisation de cas de test par RL. Bien qu'elles rendent possible l'entraînement des agents de RL, elles ne garantissent pas une bonne priorisation des tests.

job_id	test_name	duration	count	failures	error	skipped
18951159	DBNTest	0.774	2	0	0	1
19017690	RBMTest	0.621	2	0	1	1
19017690	CRBMTest	3.296	2	1	1	0
19096813	CRBMTest	3.802	2	0	0	0
19096813	RBMTest	0	2	0	0	1

Table 1 - Données initiales requises pour la mise en place de l'approche issue du projet deeplearning4j

Ces données initiales sont visibles dans la Table 1. On remarque que chaque ligne correspond à une classe de test, c'est-à-dire un sous-ensemble de cas de test ciblant une fonctionnalité spécifique, à laquelle sont associées les caractéristiques suivantes :

– **job_id** : identifiant du cycle d'intégration continue
– **test_name** : nom de la classe de test
– **duration** : temps d'exécution total des tests de la classe en seconde
– **count** : nombre de cas de test qui compose la classe de test
– **failures** : nombre de cas de test en échec dans la classe de test
– **errors** : nombre de cas de test en erreur dans la classe de test
– **skipped** : nombre de cas de test ignorés dans la classe de test

Ces seules données initiales sont complétées par des données issues d'analyses plus détaillées des tests. On distingue trois niveaux de caractéristiques à extraire pour étendre ces données d'apprentissage :

(1) Les caractéristiques basées sur l'historique de changement du logiciel : d'une manière générale, ces caractéristiques représentent la manière dont le code source a évolué entre deux builds consécutifs.
(2) Les caractéristiques basées sur l'analyse statique du code : ce sont différentes métriques issues de l'analyse statique du code source, comme la complexité cyclomatique ou le nombre d'instructions.
(3) Les caractéristiques basées sur l'exécution des tests : ce sont des informations issues de l'exécution du code, généralement par le biais des tests, comme la couverture de code.

Ces trois niveaux se distinguent par les coûts d'extraction et de manipulation des caractéristiques qui sont croissants entre les niveaux **(1)** et **(3)**. En effet, les caractéristiques du niveau **(1)** demandent moins d'efforts pour être extraites que celles du niveau **(2)**, elles-mêmes moins coûteuses à extraire que celles du niveau **(3)**. Dans le cas du premier niveau de caractéristique, c'est-à-dire celles basées sur l'historique du code, on peut retrouver les caractéristiques suivantes :

– Nombre de cycles d'intégration continue entre le build courant et la première apparition du test.
– Nombre de lignes insérées entre le build courant et le précédent.
– Nombre de lignes supprimées entre le build courant et le précédent.
– Nombre de fichiers modifiés entre le build courant et le précédent.
– Temps écoulé entre le build courant et le précédent.
– Booléen indiquant si la classe contenant le test a été modifiée ou non depuis le dernier build.
– Booléen indiquant si la classe testée par le test courant a été modifiée ou non depuis le dernier build. Cette caractéristique est basée sur la convention de nommage, qui consiste à ajouter « Test » à la fin des classes de test et n'est donc pas fiable à 100%.

Pour extraire ces caractéristiques, la seule manipulation du logiciel de gestion de code (SVN ou Git) est suffisante. Des commandes permettent de récupérer les informations utiles. L'extraction des caractéristiques du deuxième niveau nécessite l'utilisation d'un logiciel d'analyse statique du code, tel que SonarQube. Un tel outil permet la récupération de caractéristiques d'un test comprenant : son nombre d'instructions, sa complexité cyclomatique et sa complexité cognitive. La nécessité de disposer de ces caractéristiques s'explique par la volonté de mesurer la complexité des tests, avec l'intuition que cette complexité a une influence sur les résultats des tests. On suppose ainsi que les tests ayant le plus grand nombre de pas de test, ou la plus grande complexité cyclomatique, auront tendance à échouer plus souvent. Enfin, le troisième niveau comporte des caractéristiques issues de l'analyse dynamique des tests, et plus précisément de la couverture de code. Ces caractéristiques suivent également l'intuition précédente, à savoir que plus les tests couvrent de code, plus élevé sera leur taux d'échec.

4.4- Expérimentation de l'approche dans plusieurs contextes

Afin de pouvoir être mise en œuvre, l'approche présentée ici nécessite l'accès aux données initiales définies précédemment. Nous avons réalisé deux séries d'expérimentations de notre composant de priorisation rl4tcp :

a- Sur un ensemble de projets open source, à partir d'une base de données sur ces projets accessibles, pour mesurer les performances de la priorisation
b- Sur une suite de test de l'outil YEST de Smartesting.

4.4.1- Expérimentation 1 : projets open source et données RTPTorrent

Le jeu de données nommé RTPTorrent se compose de vingt projets Java open source, disponibles sur GitHub, et ayant utilisé le service en ligne d'intégration continue TravisCI. L'utilisation conjointe de ces deux outils permet de suivre l'évolution de ces projets au fil des ans. Dans un premier temps, les jeux de données issus de RTPTorrent ont été utilisés afin de développer les algorithmes de récupération des caractéristiques des niveaux **(1)** et **(2)**. Dans un second temps, ces algorithmes ont été développés pour les technologies Java, Maven et Git, puisque ce sont celles utilisés dans RTPTorrent.

L'objectif est d'être en mesure d'enrichir le jeu de données initial, grâce à l'historique des *builds* et des *commits*. En effet, il est possible de naviguer au travers de leurs historiques GitHub afin d'analyser l'évolution du code au fil du temps, mais également d'exécuter les tests unitaires et de générer un rapport de test pour un commit précis. L'intérêt de disposer d'un tel jeu de données est d'avoir la possibilité d'extraire les trois niveaux de caractéristiques identifiés précédemment. En effet, l'historique des commits, couplé à l'historique des cycles, permet dans un premier temps de récupérer des caractéristiques basées sur l'évolution du code d'un commit à l'autre. L'accès aux différentes versions du code permet, lui, l'extraction de caractéristiques basées sur l'analyse statique du code, notamment grâce à l'utilisation d'outils comme SonarQube. Enfin, la possibilité d'exécuter les tests, à condition de disposer de l'environnement adéquat, donne accès à des caractéristiques telles que la couverture de code. Avec l'ensemble de ces éléments, il est finalement possible de déterminer les tests qui ont le plus de chance d'être impactés par les changements amenés par un commit précis.

4.4.2- Expérimentation 2 : Priorisation des tests fonctionnels sur YEST

YEST[13] est un outil d'automatisation des tests en Agile, par formulation des parcours applicatifs à tester. Nous avons choisi d'expérimenter la priorisation sur une suite de tests automatisés de YEST au niveau système.

Dans le processus de développement du logiciel YEST, des tests fonctionnels graphiques sont exécutés à chaque cycle, en plus des tests unitaires, grâce à l'outil de test SWTBot[14]. Ces tests ont pour objectif de mettre en avant des anomalies au niveau de l'interface graphique. La suite

[13] https://www.smartesting.com/conception-collaborative-tests-yest
[14] https://www.eclipse.org/swtbot/

de test se compose de 193 tests fonctionnels et son temps d'exécution total moyen est d'environ 1 heure et 15 minutes. Les tests étant ordonnés par ordre alphabétique, il est rare qu'une anomalie soit détectée au début de la suite de test. Pourtant, un tel cas serait idéal : il permettrait l'arrêt du cycle en cours en faveur de la correction immédiate de la faute, en évitant des dizaines de minutes avant que cette dernière ne soit révélée. Finalement, une priorisation favorisant les tests les plus susceptibles d'échouer permettrait d'atteindre des cycles stables bien plus rapidement.

Cette expérimentation sert finalement deux objectifs distincts. Tout d'abord, le but est d'évaluer les efforts de mise en œuvre de l'approche, dans un contexte réel d'intégration continue ; notamment d'en établir un protocole pour l'équipe de développement en charge de l'intégration continue, permettant la mise en œuvre de l'approche au cycle d'intégration continue. Enfin, le deuxième objectif est de mesurer l'efficacité de cette approche pour répondre à une problématique réelle. La communication directe avec l'équipe de développement permettant de se faire une bonne idée du service rendu.

Conclusion

La priorisation de cas de test basée sur une utilisation de technique de *Reinforcement Learning* est une technique émergente et prometteuse de priorisation. L'approche présentée ici, nommée *rl4tcp*, vise à étudier l'état de l'art du domaine, afin d'utiliser la combinaison la plus efficace possible en termes d'algorithmes et d'environnements de RL, ainsi que de caractéristiques issues de l'analyse du projet testé. En parallèle, l'objectif est de proposer un prototype de recherche facilement déployable dans une chaîne d'intégration continue, grâce aux technologies de conteneurisation.

II.13- Je valide et surveille ma production avec des tests automatisés

par Michael Granier

La qualification de notre sprint Agile s'est passée pour le mieux.
2 semaines pour mettre en place et valider de nouvelles features en pensant à :

- Leur associer des tests unitaires pertinents
- Les valider en les intégrant dans différents environnements
- Réaliser des UAT pour s'assurer que le besoin métier est bien satisfait
- S'assurer que la non-régression automatisée est validée ou maintenue sur tous les niveaux de tests
- On a même eu le temps de réaliser quelques tests de performances et de charges.

Tous les indicateurs semblent au vert pour livrer le contenu de notre sprint en production.
"Que peut-il bien nous arriver d'affreux maintenant ?"

Et pourtant, on peut parfois pécher par excès de confiance et rater le dernier virage d'une mise en production réussie, en oubliant tout simplement de surveiller et de tester cette dernière avant, pendant et après sa mise à jour.

Après autant d'efforts de test, pourquoi diable tester en production et surtout automatiser ? Quels tests réaliser ? Quels tests automatiser ? Que faut-il éviter ?

Je vous propose aujourd'hui de partager mes réflexions pour répondre à ces questions. Basées sur mon expérience personnelle et professionnelle sur l'automatisation des tests en production, elles sont centrées sur la validation d'application Web.

Que l'on mette en production 2 fois par an ou que l'on soit dans un contexte de déploiement continu, l'objectif est le même : sécuriser rapidement et efficacement ses mises en production.

1- "J'ai un monitoring en béton, pas besoin de tester voyons !"
Ou : alerter quand arrivent les erreurs, c'est bien, mais anticiper ces dernières, c'est mieux !

Le monitoring des logs reste le moyen le plus sûr de suivre l'état de ses applicatifs en production. Monitoring qu'on ne retrouve pas forcément en environnement de test pour des raisons de coûts (monitoring qui n'est d'ailleurs pas forcément testé lui aussi… mais c'est une autre histoire).

Et pourtant, identifier une défaillance par ce moyen ou par des remontées utilisateurs, alors qu'elle pourrait être prévenue par quelques tests automatisés pertinents, vous permettra de conserver votre business et votre e-réputation.

2- “Mais, ma production est identique à ma recette logiquement, non ?” Ou : constater qu’il-y-a toujours un delta entre les environnements de tests et la production.

Voici une liste d’éléments non exhaustifs permettant d’affirmer que votre production n’est pas aussi proche de l’environnement qui a été validé :

- **Dimensionnement :** Pour des raisons de coût évidentes, le dimensionnement serveur d’un environnement de recette est inférieur à celui d’une production. Votre application sera donc plus réactive en production mais pourra aussi réagir différemment.

- **Multi-instance :** Afin de mieux absorber la charge, de nombreuses instances peuvent être disponibles. Doit-on toutes les tester ? Peut-être pas, mais en cas d’erreurs constatées, il sera important de pouvoir cibler l’instance fautive.

- **Content Delivery Network (CDN) type Akamai, OVH... :** Pour les plus gros sites, l’optimisation de l’affichage du site à l’utilisateur passera par un CDN.
  Ce genre de dispositif étant non présent en test, des impacts sont à prévoir !

- **La combinatoire des livrables :** Plus votre système est complexe, plus il possède d'applicatifs. Si vous êtes en agile à l’échelle avec différentes équipes qui livrent indépendamment, vous avez peu de chances d’avoir testé exactement la même configuration que votre production…

Autant de raisons qui vous font dire “je dois tester ma production”.
Voire qui vous font penser plus loin : avoir une solution automatisée vous remontant en continu le bon état des parcours de vos applicatifs, après et entre chaque livraison, en complément de votre monitoring.

3- “Mais je croyais qu’une fois en production, c’était trop tard pour tester” Ou : parlons de prod cachée et techniques de déploiement type Blue / Green / Canary.

Si par production, on entend “visible à l’utilisateur”, laissez-moi vous parler rapidement des techniques ayant émergé avec le cloud afin de faciliter la vie de vos DevOps, mais aussi vous rassurer une dernière fois avant toute activation de vos nouvelles features auprès des utilisateurs finaux.

Je pense aux déploiements “Blue / Green” ou “Canary” qui permettent d’avoir 2 versions du site simultanément en production : l’ancien et le nouveau.

Ainsi, via un routage IP, un cookie de session spécifique ou un script d’initialisation, vous pouvez accéder à la nouvelle version pour exécuter vos tests sans impacter les utilisateurs finaux, dans des conditions de productions réelles.

Une fois les tests réalisés, les instances sont alors activées progressivement, avec surveillance du vieillissement. Le rollback en cas de problème est tout aussi rapide !
Pour l'automatisation, il est important que vos automates puissent cibler ces instances de manière efficace.

4- "Dois-je refaire les mêmes tests réalisés pendant mon sprint ?" Ou : ciblons les bons tests à automatiser en production.

La quantité de tests à jouer en production est, en toute logique, moindre par rapport aux environnements de tests (sinon vous faites peut-être de la surqualité).
On y retrouve des tests similaires mais surtout des tests spécifiques à cette plateforme.

4.1- Je garde en manuel

Le paiement : A moins d'avoir une carte bleue magique au fond infini, tester à outrance de manière automatisée votre partenaire de paiement, avec les différents cas, risque d'être contre-productif, nous reviendrons sur ce point par la suite. Un premier paiement manuel à chaque mise en production doit suffire (sauf mise en production spécifique au paiement) et la surveillance (indispensable) des paiements vous avertira immédiatement.

Les corrections d'anomalie : Une anomalie corrigée ne nécessite pas systématiquement un test automatisé. S'assurer que l'anomalie est bien livrée reste indispensable.

Les nouvelles évolutions : Ce sont des tests "oneshot" en production. Vous pouvez utiliser les tests automatisés réalisés en environnement de test mais il n'y a pas de plus-value à conserver ces tests en continu ou à chaque mise en production. Il faudra privilégier l'exécution des tests d'acceptance de vos nouvelles fonctionnalités.

4.2- Je peux automatiser

Les tests continus de vos API sont un bon moyen d'avoir un retour rapide et efficace de vos couches techniques, complémentaires des tests d'interface pour cibler et analyser plus rapidement les défaillances.

Un monitoring de la performance (Back et Front) de sa production peut être couplé à vos tests automatisés.
La récupération des temps de réponse et de chargement peut être intégrée à votre solution d'automatisation et permettre la mise à disposition de graphiques en temps réel, associés à de vrais parcours utilisateur.

Quelques tests de sécurité : La sécurité en production est souvent conséquente et des solutions type Captcha ou Datadome peuvent être mises en place. Pourquoi ne pas prévoir quelques tests automatisés pour vérifier, simplement, l'apparition de ces captcha ou de page de blocages de robots ?

Des tests de non-régression peuvent être exécutés en continu. Il faut surtout choisir les parcours critiques permettant la sécurisation du business.

- Les parcours les plus représentatifs du trafic utilisateur
- Les parcours qui génèrent le plus de business
- Les pages à fort impact sur l'image et le juridique (CGV, accessibilité…).

Si vous avez des difficultés à cerner ces types de parcours, le mieux est de se rapprocher des équipes métier et de récupérer les informations des différentes solutions d'analyse de l'application (Analytics, Metriques…).
Vous disposez aussi d'outils innovants permettant de cibler et créer les cas de tests directement depuis les logs en utilisant l'IA (Gravity pour ne pas le citer).
Il est important de se limiter à un certain pourcentage du trafic et du business en ciblant essentiellement les cas les plus représentatifs.
Il est totalement justifié d'avoir ces tests de régression identiques en environnement de tests et production.
Attention cependant, ces parcours peuvent être amenés à changer régulièrement, il convient donc de les challenger avec les équipes métier pour être toujours à jour.

Il va être important **de tester vos partenaires externes :** Vous ne maîtrisez pas leurs cycles de mise en production. Et quoi de pire qu'un partenaire non résilient faisant échouer votre parcours ? Il est donc important d'automatiser quelques tests (Back et Front) pour contrôler ces derniers. C'est d'autant plus important que ces partenaires sont souvent bouchonnés dans vos environnements de tests, vous ne pouvez alors mesurer leurs impacts réels qu'en production.

Les tests de redirection, pas indispensables, le sont davantage quand vous utilisez des solutions de CMS (Salesforce, Drupal…). Il peut alors être intéressant d'avoir des tests vérifiant les liens de redirection vers les différentes pages (souvent institutionnelles), afin de s'assurer que des urls de recette ne se soient pas égarées, ce qui rendrait la navigation impossible aux utilisateurs. Je le rajoute pour l'avoir vécu sur plusieurs types d'applications.

5- "Je peux faire ces tests manuellement, pourquoi les automatiser ?"

Ou : pourquoi tester uniquement vos mises en production quand vous pouvez surveiller en permanence vos parcours !

Si nous sommes en phase sur les constats suivants :

- Il faut réaliser des tests à chaque mise en production pour sécuriser la qualité
- Les mises en production doivent être rapides
- Il faut surveiller sa production en permanence, techniquement et fonctionnellement.

En toute logique, l'automatisation est à envisager comme un investissement à long terme pour vos environnements de tests mais aussi et surtout de production.

Gageons qu'un test automatisé n'ayant de valeur que s'il est rejoué fréquemment, il va être important de pouvoir suivre ses exécutions récurrentes en production, où chaque échec répété devra être scruté à la loupe pour remonter une défaillance ou maintenir les scripts des automates. Votre plateforme finale étant censée être la plateforme la plus stable et la plus rapide pour recevoir des tests automatisés, c'est une particularité importante à exploiter.

Cependant, afin d'éviter les alertes inutiles et chronophages, il est important d'adapter les tests à l'environnement et à ses subtilités !

6- Retour d'expérience :
Mes 9 conseils pour bien automatiser vos tests en production.

Ciblez les instances et récupérez des informations de session

Indispensable pour cibler des instances Blue / Green afin de valider la bonne version, il est tout aussi important de pouvoir récupérer ces informations d'instance et de session sur laquelle vous réalisez vos tests, sous peine de ne pas pouvoir détecter une erreur ou de reproduire une défaillance. Ce sont des informations essentielles à récupérer par vos tests automatisés, à destination des exploitants de vos plateformes de production.

Attention à vos données !

Sur vos environnements de tests, c'est open bar !
Parfois même, ces données sont créées et disponibles à l'infini.
En production, ce sont de vraies données utilisateur ou produit, parfois de vrais stocks… Est-il judicieux de perdre du business en bloquant ces dernières à cause de vos tests ?

Concernant les données produit :

– Ajoutez des étapes automatisées de vidage panier, suppression de réservations ou contrats, à la suite de chaque test concerné.

– Lancez régulièrement des scripts de libération des produits associés aux mails de vos tests, avec un pattern identique.

Concernant les données utilisateurs :

– Prévoyez des purges fréquentes des comptes créés, toujours avec un pattern identique (par exemple, la même boite mail @test_site.com).

Ne faites jamais passer vos tests avant vos utilisateurs !

Pas de paiement mais pas d'erreur de paiement non plus !

Comme nous l'avons déjà dit, vous pouvez être tenté de tester votre partenaire de paiement en production en provoquant des erreurs, à défaut de réaliser de vrais paiements.
Ce genre de comportement à répétition peut avoir un impact assez néfaste pour votre business, en créant du bruit sur les backoffices financiers et les outils de scoring bancaire pour gérer la fraude.
A proscrire en production !

Pensez à gérer vos solutions d'analyses (Analytics) !

Indispensable pour cibler et surveiller les parcours les plus critiques…
En revanche, si vous ne faites pas le nécessaire pour ignorer les scénarios automatisés dans les statistiques de fréquentation, vous vous retrouvez à créer vous-même le trafic des parcours critiques à surveiller !

Solutions :

– Bloquez les différents outils analytics directement dans vos scripts ! Pour information, j'ai pu constater que ces blocages peuvent parfois impacter certaines actions du parcours.
– Excluez les automates de tests des analytics (par IP)

Vos AB Tests…

De plus en plus utilisés, les AB tests auront un impact direct et non négligeable sur la stabilité de vos tests automatisés.
Prévoyez de les exclure en bloquant les bonnes requêtes ou en initialisant un cookie spécifique pour forcer tel ou tel parcours.
L'autre solution possible, mais non pérenne, reste de prévoir une étape conditionnelle pour pouvoir parcourir fonctionnellement l'étape A ou l'étape B.
Afin de maîtriser le parcours testé, je préconise de séparer en 2 tests distincts, en forçant A et B, pour s'assurer que les 2 parcours fonctionnent sans le côté aléatoire.

...et vos captchas !!!

Espérer que vos automates de tests puissent passer outre vos antirobots de production est peut-être trop ambitieux…
De toute manière, pour tout test automatisé en production, il est important de vous rapprocher de votre **responsable de la sécurité (RSSI)** afin qu'il valide la démarche et vous propose des alternatives pour gérer tous les antirobots du site (Captcha, blocage d'IP si trop de requêtes…)
Puisque tout désactiver en production pour des tests est impossible, il faudra exclure vos IPs des blocages sécurité (Whitelist), ou créer dynamiquement un jeton de session sécurisé (type JWToken) pour vous assurer qu'il s'agit bien des automates de tests autorisés par l'entreprise.

Ne changez pas de solution d'automatisation pour la production, mais maintenez plusieurs niveaux de tests "Dev > Recette > production" !

Les solutions d'automatisation sont nombreuses sur le marché et, quel que soit votre choix, ne changez pas en fonction de l'environnement testé !
Vous devez utiliser la même solution de l'assemblage à la production.
L'automatisation doit être pensée comme un projet de développement et prévoir un versionning des tests en fonction de l'environnement est alors plus cohérent.
Ainsi, les tests peuvent être maintenus et livrés conjointement avec les versions de l'applicatif pour une compatibilité optimale.
De même, les éléments (ou page objects) utilisés pour automatiser les parcours restent logiquement identiques.

Varier les configurations : Terminaux / Navigateurs

Lorsque vous ciblez les parcours critiques avec le métier, il est aussi important de récupérer les données des configurations les plus utilisées.
Il suffit alors de configurer ces parcours, en sélectionnant pour chacun une configuration différente, pour éviter de multiplier le nombre de tests.

Quelle fréquence d'exécution ?

Si vos tests passent par des parcours similaires, il n'est pas pertinent d'avoir une fréquence d'exécution trop élevée. Dans ces cas-là, l'idéal est d'avoir toujours au moins un test automatisé en cours d'exécution et de séquencer les lancements des tests suivants.
Il est cependant important de prévoir un mécanisme de rejeu automatique si le test tombe en échec, afin de confirmer qu'une défaillance a été détectée.

7- Convaincu ? A vos tests automatisés... en production !

J'espère vous avoir convaincu de l'importance de tester en production mais aussi d'automatiser ces tests, afin de garder une surveillance fonctionnelle et performance permanente en plus de vos outils de monitoring !
Quand on souhaite partir en production rapidement et en toute sécurité, l'automatisation des tests est une démarche qu'il faut cadrer tout au long du cycle de développement et intégrer aux outils d'industrialisation de vos déploiements…
jusqu'à son utilisation finale !

PARTIE III
ALLER PLUS LOIN

III.1- Notre IA sera-t-elle éthique ou pathétique ?

par Olivier Denoo

Dans sa définition la plus couramment acceptée, l'éthique est la partie de la philosophie qui envisage les fondements de la morale. Elle constitue aussi l'ensemble des principes moraux qui sont à la base de la conduite de quelqu'un. (Larousse)

Profondément ancrée dans les valeurs de nos sociétés, en débat permanent avec nos valeurs personnelles, elle évolue au fil du temps et des mœurs et souvent précède la norme de peu.

Pour des philosophes tel qu'Aristote ou Kant, l'éthique définit ce qui doit être.
Le Bien et le Mal prennent naissance dans l'intention.
Moins austère et plus contemporain par son pragmatisme, l'utilitarisme (J. Bentham - J.S. Mill), principale théorie morale du conséquentialisme, tend à la maximisation du bonheur pour le plus grand nombre voire, plus récemment encore, à la maximisation de la satisfaction des préférences (P. Singer). Le bonheur des uns faisant le malheur des autres, il conviendrait donc de peser et d'observer les conséquences de nos actes en favorisant ceux qui maximisent le profit du plus grand nombre au détriment de nos actes égoïstes.

Si j'ai repris, de manière simplifiée – certains diront même simpliste avec raison – ces deux courants majeurs de l'éthique, parmi tant d'autres, c'est parce qu'ils s'invitent de plus en plus dans nos métiers du test, et en particulier lorsque l'on touche à l'Intelligence Artificielle.

Depuis toujours, le testeur a constamment été confronté à des dilemmes éthiques, souvent même sans s'en rendre compte ou sans trop s'en soucier. Si l'implication semble évidente lorsque l'on parle de tester des dispositifs militaires, le diable se niche souvent dans les détails. Ainsi, le testeur a, quasiment en permanence, accès à des données sensibles et confidentielles (nom, prénom, adresse, téléphone, email, moyens de paiement, listes noires ou sensibles…). Même si le RGPD et l'anonymisation ont fait leur chemin depuis le temps du « tout visible en production », nombreuses sont encore les occasions de déraper. Un rapide coup d'œil sur la feuille de paie du collègue ou sur le 06 de la voisine du dessus, et la ligne est franchie !

L'ISTQB® ne s'y est pas trompée, en publiant, il y a des années déjà, son code d'éthique[1] à destination des testeurs. Comprenant des aspects déontologiques et professionnels, il s'appuie notamment sur les codes recommandés par l'ACM et par l'IEEE pour les ingénieurs en adoptant, sans grande surprise, une philosophie pragmatique et utilitariste.

Avec une telle armada législative et normative, bien souvent renforcée par des chartes internes propres à l'entreprise, nous pourrions aisément nous croire à l'abri des déconvenues ; bien protégés avec en guise d'armure, un carcan moral aux contours bien définis.

[1] Cf. https://www.istqb.org/about-us/istqb%C2%AE-code-of-ethics-for-test-professionals.html

La réalité est malheureusement bien plus complexe tant morale et éthique sont polymorphes et changeantes, à la fois dans leurs dimensions culturelles, sociétales et temporelles.

Ainsi, ce qui pourrait ne sembler qu'une moquerie gentiment provocatrice à d'aucuns, devient en d'autres lieux une offense passible d'actes violents ou vengeurs[2] ; ainsi, le contrôle social exercé par les réseaux sociaux infléchit-il le droit, la liberté individuelle ou la justice[3] ; ainsi les us et coutumes, la vérité d'hier ne sont plus ceux d'aujourd'hui[4].

Dans un univers communicant et en constante accélération ; dans un monde qui confond vitesse et précipitation, où recouper et analyser les informations devient souvent un luxe impayable ; dans un monde qui se veut efficace et froid fût-ce au prix de l'humain ; les valeurs passent et lassent et la morale se façonne et se défait au souffle des vents dominants.

L'Homo Sapiens, sans cesse confronté à des choix cornéliens, oscille-t-il donc constamment entre Charibde et Scylla : faut-il cesser de consommer des vêtements bon marché ou des pâtes à tartiner à l'huile de palme mais condamner les producteurs des pays en développement à la famine ; ou continuer comme si de rien n'était en soutenant l'exploitation de travailleurs sous-payés et en ruinant notre santé ? Faut-il acheter massivement des voitures électriques et déporter la pollution en amont, organiser la pénurie de métaux rares, et feindre d'ignorer que l'électricité il faut bien la produire d'une manière ou d'une autre ; ou ne pas céder au progrès ou à la mode et continuer l'emploi massif d'hydrocarbures pour mieux participer au réchauffement climatique ?

Les solutions holistiques et modérées, si tant est qu'elles existent, sont souvent incompatibles avec nos choix, nos valeurs, nos orientations du moment ou plus simplement avec nos désirs que d'aucuns n'hésiteront pas à qualifier d'égoïstes. Il en résulte souvent une grande et vaine confusion dans laquelle les lignes ne bougent que par à-coups, au gré de hashtags en forme d'électrochocs.

Il n'est donc pas étonnant que dans notre monde technologique du tout-connecté, l'éthique s'invite à la table de l'informatique et des logiciels, à moins que ce ne soit l'inverse.
Par conséquent, le testeur – ou plus largement les métiers de la qualité – se retrouve concerné au premier chef et se doit d'actualiser ses perspectives.

Pour mieux appréhender le sujet de l'I.A., il est peut-être préférable de partir d'un exemple simple :

Vous conviendrez sans doute avec moi que la visibilité accrue de certaines mouvances telles que #metoo, #blacklivesmatter ou encore les mouvements LGBT, pour ne citer qu'eux parmi tant d'autres, ont considérablement renforcé la lutte contre de nombreuses formes de discrimination.

---

[2] Je parle ici, bien évidemment des affaires des caricatures, mais aussi des salles incendiées lors de la « dernière tentation du Christ » ou encore des exécutions des opposants pacifiques dans des pays lointains où les droits de l'homme (et de la femme) répondent à une autre définition.

[3] Il n'y a qu'à observer les nombreux cas de délation ou de haine exprimés au travers des réseaux sociaux, quel qu'en soit le sujet. Qu'un cycliste bouscule une fillette, sans mal, et vidéo à l'appui, la justice se sent forcée de prendre les choses en main.

[4] J'en veux pour preuve notre regard face à la condition des femmes, de l'écologie ou de la colonisation, pour ne citer que des exemples d'une criante actualité.

Une tendance assez forte de la société actuelle, qui se veut plus juste et plus égalitaire - et c'est tant mieux – dont les conséquences sont directement visibles quand elles ne sont pas surmédiatisées[5].

Pourtant, alors même que nous ne parlions pas encore d'I.A. dans l'entreprise, je testais déjà, il y a vingt ans déjà, des logiciels qui calculaient le montant des primes d'assurance auto en fonction de la marque, du modèle, de l'âge et du lieu de résidence du conducteur principal.
Ainsi, un conducteur de berline de luxe vivant dans un quartier défavorisé ou dans une localité cible de fréquents « home / car jacking » voyait-il sa prime considérablement majorée par rapport à d'autres preneurs, toutes autres conditions semblables.
Dans la même logique, un jeune conducteur payait le prix fort de son inexpérience en voyant sa prime majorée de plusieurs dizaines de pourcents.
En matière d'assurance vie, le montant de la prime s'appuyait notamment, entre autres paramètres, sur des statistiques basées sur certains antécédents familiaux et sur l'indice de masse corporelle.
Ainsi, une personne en surpoids et en parfaite santé, sans aucun antécédent personnel ; un fils de diabétique ou de cardiaque, pouvaient aisément voir leur prime augmentée de plusieurs dizaines de pourcents.
Du risque à la discrimination, il n'y a qu'un pas, si aisé à franchir…et si communément accepté qu'il en devient presque vide de sens…jusqu'au prochain électrochoc ?

Bien que le risque bancaire (ou de l'assurance) se calculait déjà sur base de statistiques, elles-mêmes issues de données propriétaires ou non, l'avènement de l'I.A. moderne et accessible à tous[6], a considérablement amplifié le phénomène en l'étendant à des dimensions jamais vues jusqu'alors. S'il semble ainsi raisonnable d'intégrer risque et sinistralité dans le calcul de nos primes, qu'en est-il alors lorsqu'un algorithme se met à impacter beaucoup plus profondément nos vies ?

Dans « Weapons of Math Destruction », Cathy O'Neil décrit, au travers d'exemples à la fois glaçants et édifiants, comment des modèles simplistes ou mal pensés peuvent dramatiquement impacter notre quotidien et renforcer les inégalités.
Qu'il s'agisse du classement des universités américaines, de la présélection de candidats à un emploi ou de la gratification du personnel, le big data s'emballe et aboutit immanquablement à des effets pervers et indésirables pour peu que les corrélations soient trop évidentes ou les biais trop présents.
Ainsi, pour en revenir à mon exemple, les usagers tromperaient le système en abusant de ses faiblesses : en omettant, par exemple, de signaler leurs antécédents familiaux, ou en trichant sur leur poids. Les preneurs échappent ainsi au « scoring » négatif initial et paient une prime non majorée. Ils pourront toujours, en cas de contrôle par la suite invoquer l'ignorance, l'erreur de bonne foi ou l'évolution de leur état.

---

[5] Que l'on parle des manifestations diverses, des boycotts, de la cérémonie des César…les exemples sont légion

[6] Top 18 Artificial Intelligence Platforms in 2020 - Reviews, Features, Pricing, Comparison - PAT RESEARCH: B2B Reviews, Buying Guides & Best Practices (predictiveanalyticstoday.com) ; The Best 7 Free and Open-Source Artificial Intelligence Software (goodfirms.co)

Dans le cas de l'assurance auto, on pourrait aisément imaginer que la surprime aurait pour effet de chasser les conducteurs plus aisés des quartiers pauvres, contribuant ainsi à contrecarrer les initiatives de mixité sociale tout en renforçant la ghettoïsation de ces derniers[7].

J'en vois déjà s'élever pour me dire qu'en matière d'assurance, il est légitime que ceux qui prennent plus de risques paient plus pour les couvrir et qu'il ne serait pas juste qu'un prime d'incendie pour un hôtel particulier soit identique à celle d'un petit appartement dans un HLM de banlieue. Certes !

Mais quand l'I.A. se mêle à la justice ? Sommes-nous bien tous égaux, ou, comme le disait G. Orwell, certains sont-ils plus égaux que d'autres[8] ?

Lorsqu'en 2014, Vernon Prater vole 86 $ dans un magasin et que Brisha Borden tente de voler un vélo d'enfant, ils sont bien loin de savoir que leur avenir va se jouer, le même jour, dans les replis obscurs d'un algorithme[9]. Programmé pour évaluer le risque de récidive d'un prévenu sur base d'un questionnaire, il donnera au premier le score minimum de 3/10, en dépit de son lourd passé judiciaire. La seconde écopera d'un 8/10 alors qu'elle n'a à son actif que des délits mineurs. Il sera libéré sous caution, elle fera de la prison ferme. Il récidivera dans les deux ans (vol à main armée), elle non. Elle est noire, il est blanc.
Le même algorithme, largement utilisé par les juges américains, a été suspecté à de nombreuses reprises de discrimination à caractère raciste. Pourtant, et c'est sans doute le pire, l'évaluation, basée sur un questionnaire comptant 137 questions, ne comporte aucune indication sur l'origine ethnique du prévenu ! Alors que le biais semble évident et communément reconnu, il n'a donc pas été introduit sciemment au cœur de l'algorithme, mais plutôt induit par un ensemble de conditions culturelles, socio-économiques et professionnelles qui ont peu à peu construit un profil discriminant (et discriminatoire). On pourrait imaginer que le monde en a tiré les leçons. Et pourtant, récemment encore et si près de nous, l'Estonie, membre de l'U.E. se prépare à mettre en place un dispositif similaire pour juger des délits mineurs[10].

Un récent couac dans le système de normalisation des grades scolaires en Angleterre a abouti à une discrimination similaire : les élèves les plus touchés par la différence entre le score obtenu et sa normalisation appartiennent en majorité aux milieux socio-professionnels les plus défavorisés[11]. Des observations similaires avaient déjà été dénoncées par Cathy O'Neil, au sein du système universitaire américain, au début des années 2000.

Nos I.A. se nourrissent de données (figure 1). Des données provenant de capteurs, comme ceux installés sur nos voitures (autonomes ou non) ; mais aussi des données provenant de sources plus ou moins fiables, plus ou moins biaisées, que nous en ayons conscience ou non.

---

[7] Ainsi Pour un juge, il ne faut pas rouler en Jaguar à Charleroi (lefigaro.fr) – le ridicule du propos est que Charleroi n'est pas un quartier, mais une ville de plus de 200.000 habitants (pas loin de 400.000 avec les arrondissements)

[8] "Tous les animaux sont égaux, mais certains le sont plus que d'autres. " – La ferme des animaux.

[9] Etats-Unis: un logiciel de prédiction des récidives aux penchants racistes (rtbf.be)

[10] En Estonie, une intelligence artificielle va rendre des décisions de justice (lefigaro.fr)

[11] A-level results: almost 40% of teacher assessments in England downgraded | Education | The Guardian

Dans tous les cas, ces données, si elles ne sont pas correctement évaluées, sont faillibles et présentent un danger potentiel parce qu'elles sont quasiment toujours utilisées en tant qu'aide à la décision, voire en tant que facteur décisionnel unique dans les cas les plus extrêmes.
Les données à caractère technique – issues de mesures, de capteurs – peuvent aisément induire un faux sentiment de sécurité ou d'insécurité, en particulier lorsque de conditions environnementales adverses.

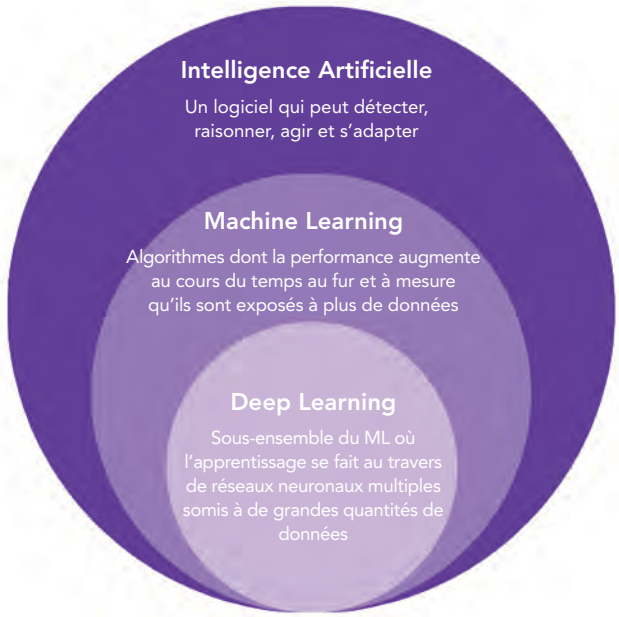


Figure 1

Ainsi, une simple caméra de recul sera-t-elle inopérante quand la lentille est maculée de boue ou de projections ; ainsi une Tesla S3 sans lidar confond-elle le flanc d'un camion au soleil et la ligne d'horizon[12]; ainsi les caméras de lecture des plaques minéralogiques de nos radars[13] tronçons auraient bien besoin d'un rendez-vous chez l'oculiste.
Pourtant, alors que les experts se veulent rassurants et que, statistiques à l'appui, ils nous assènent que c'est juste une question de coût, de technologie ou de temps ; le vrai débat a lieu sur un autre terrain : celui de l'éthique.

Certes, je peux entendre que la version suivante sera plus performante et corrigera le problème (c'est d'ailleurs un motif d'amusement tant cela me rappelle des souvenirs de ma vie de testeur) ; certes je peux entendre que le bilan global des voitures autonomes est plus que satisfaisant ; certes je conçois fort bien que l'Humain est faillible et moins alerte qu'une armée de capteurs coordonnés par une puissante unité de calcul ; certes je comprends bien qu'à l'ère des fusées et du mobile on gère très bien la technologie ; certes je comprends aussi que l'incident du trolley n'est pas un problème scientifique, mais un problème moral. J'ai, en revanche, plus de peine à comprendre pourquoi - si cette I.A. conduit tellement mieux que moi - je reste civilement seul responsable d'un accident sur lequel je n'avais, somme toute, aucune prise !

[12] Accident mortel sur Tesla S : l'absence de Lidar à l'origine du crash - Le Monde Informatique
[13] Les radars tronçons incapables de lire certaines plaques d'immatriculation! (sudinfo.be)

Serais-je donc, un conspirationniste rétrograde qui refuserait de céder le pas à la machine, ou bien un technophile désabusé endossant la responsabilité, qu'il n'a pas vraiment choisie, d'un dommage, qu'il n'a pas vraiment provoqué ? L'absence de responsabilité juridique ou même éthique de l'I.A. ou de ses ayants-droit pose ici une réelle problématique.

Lorsque l'I.A. s'appuie sur des données informatiques (non issues de capteurs physiques), la problématique est à la fois plus diffuse et plus délicate à identifier.
Alors qu'il semble encore relativement aisé de détecter qu'un capteur dysfonctionne, même si les exemples précédents me donnent tort ; qualifier la qualité des données et de leur source est une toute autre paire de manches.

En 2019, lors d'une grande conférence en Chine, alors que face à une audience de 600 personnes, je demandais s'il y avait des CEO dans la salle – signe des temps quelques mains s'étaient levées – je leur soutins alors abruptement qu'il ne pouvait y avoir de CEO en Chine. J'en voulais pour preuve une capture d'écran de Google Image, réalisée une semaine avant, où ne figuraient quasiment que des hommes blancs, pour la plupart américains. La quasi-absence de femmes, d'africains ou d'asiatiques en était choquante.
J'enchaînais, face à leur sourire gêné et poli, qu'ils pouvaient bien s'estimer heureux de ne pas se retrouver dans la catégorie « dealers », réservée aux afro-américains, aux sud-américains et aux nord-africains, nouvelle capture d'écran à l'appui. Même si aujourd'hui, l'équilibre est un peu mieux respecté – en partie en raison de l'actualité – nous sommes encore bien loin du compte. D'ailleurs, comme la Chine ou la Russie se sont dotés de leurs propres moteurs de recherche et de leurs propres réseaux sociaux, notre vision de ces pays, si elle se limite à nos propres références est à prendre avec circonspection.

Que dirait alors une I.A. en apprentissage face à de telles données ?
Quels stéréotypes véhiculerait-elle en l'absence de supervision appropriée ?
Quelle portion du monde omettrait-elle, sciemment ou non ?

La récente déconvenue d'Apple en matière de reconnaissance faciale, nous donne une première indication[14]. Est-ce seulement une question de temps et de version ?

Le monde n'est pas fait que du bruit de ceux qui sortent du lot, il est aussi fait du silence des masses, dont on sait beaucoup moins et qui pourtant en constituent la majorité. La partie immergée de l'iceberg. Un tel contexte favorise le biais au détriment de la représentativité.
Quelle quantité de données faut-il pour permettre à une I.A de modéliser une population cible, large et diversifiée où l'informatique s'invite à tous les étages ?
Tout testeur qui s'est un jour heurté à la question des données ou des environnements de test comprendra aisément l'ampleur de la problématique.
S'il fallait encore compliquer les choses, la correction d'un biais est délicate et même hasardeuse. D'une discrimination négative qu'on voudrait corriger, on risque de passer, en forçant trop le trait, à une discrimination positive…et la majorité des CEO seraient donc des femmes noires ! Aussi tentant que cela puisse paraître, ce serait tout aussi irréaliste.

[14] La reconnaissance faciale d'Apple accusée de ne pas savoir différencier les visages des Chinois (bfmtv.com)

Le remède serait-il donc pire que le mal ? Prenons Sofia, ce robot humanoïde de Hanson Robotics qui a tant fait pour rendre l'I.A. sympathique et accessible à tous. Dotée d'expressions humaines et d'une incroyable faculté à converser de tout et de rien, elle a ses entrées dans les talkshows outre-Atlantique et même, excusez du peu, à l'ONU ! Citoyenne d'honneur des Emirats Arabes Unis, elle trébuche une première fois sur la question : « voulez-vous détruire le monde ? ». Elle répond alors avec une froideur toute mécanique : « détruire le monde…pourquoi pas, oui cela semble une bonne idée ». Devant un public mi-amusé, mi-médusé, ses concepteurs promettent de parfaire son apprentissage, et ils tiennent parole ! Quelques temps plus tard, face à la même question (prévisible s'il en est), elle répond avec humour : « vous regardez trop les films d'Hollywood et lisez trop Elon Musk…je suis bienveillante, si vous êtes bons avec moi, je serai bonne avec vous ». Glaçant ! J'espère que les concepteurs travaillent à nouveau son apprentissage, car pour ma part, je tremble à l'idée de ce que « bon » peut bien vouloir signifier pour Sofia.

Nos I.A., comme tout programme informatique, sont aussi sensibles aux intentions malicieuses et autres techniques adverses[15]. Dans le monde de l'I.A., il suffit parfois d'ajouter un peu de bruit numérique (fluctuation parasite) à une image pour qu'elle devienne méconnaissable ; un simple autocollant posé sur un panneau de signalisation le rend subitement illisible[16] , et le port d'un vêtement aux motifs soigneusement choisis vous confère une invisibilité que même H.G. Wells n'aurait pas imaginée ! Une réalité diminuée, en quelque sorte !
Là encore, ce n'est qu'une question de temps et de technique, et je veux même bien le croire, tant que rien ne défaille, tant que ma vie, ma santé ou mon bien être ne sont pas en jeu.

Comme tout produit ou service informatisé, l'I.A. vise à atteindre un objectif plus ou moins ouvertement décrit dans ses exigences métier. Lorsque le service est gratuit, c'est que vous êtes le produit ! La plupart des réseaux sociaux ont pour finalité de vous vendre de l'espace publicitaire, d'anticiper vos désirs[17] ou mieux, de les susciter. Si la réalité n'a pas encore tout à fait rejoint la fiction, le concept fait son chemin et se matérialise peu à peu sous diverses formes.

En vous servant, jusqu'à l'indigestion, toujours plus de ce ou de ceux que vous aimez, réseaux sociaux et G.A.F.A.M. distillent avec maestria les principes de persuasion, décrits par R. Cialdini dans son livre « Influence et manipulation ».
Bien plus que dans les jeux vidéo, tous vos choix comptent. Ils bâtissent peu à peu votre identité numérique, votre profil, qui se revend à prix d'or et à votre insu[18]. En forçant un peu le trait, si votre collègue ignore combien de sucres vous mettez dans votre café du matin, les G.A.F.A.M., eux, le savent. Si les services rendus sont très souvent réels et pratiques, les finalités des offreurs et de leurs cibles ne sont que partiellement compatibles ou complémentaires.
Ils participent aussi amplement au développement de la fracture numérique et de ses laissés pour compte économiques ou technologiques.

---

[15] (PDF) Review of Artificial Intelligence Adversarial Attack and Defense Technologies (researchgate.net)

[16] Adversarial attacks against machine learning systems – everything you need to know | The Daily Swig (portswigger.net)

[17] Amazon veut expédier des produits avant qu'ils soient commandés (lefigaro.fr)

[18] Du moins dans sa composition et dans ses détails.

Dans tout risque, il y a une opportunité. Alors que d'aucuns dénigrent, s'insurgent, boycottent, feignent l'ignorance, se désespèrent ou se cachent derrière de faux profils ; d'autres intègrent les nouveaux codes et exploitent les filons à leur portée. Derrière une image léchée et savamment contrôlée, ils sont des influenceurs courtisés, des bateleurs au rayonnement international, des agitateurs de mœurs et d'idées. Pour le meilleur et, souvent aussi, pour le pire.
Dans toute opportunité, il y a un risque. Leur pendant sombre « trolle » sans vergogne, dans une relative immunité numérique ; pervers, narcissiques, ils trompent et manipulent, propageant allègrement les aspects les plus abjects de l'âme humaine. Et parfois, leur impact est tel qu'il fait vraiment pencher la balance du côté sombre.

Quand en 2016, Tay[19], le « chatbot » de Microsoft, dérape sur Twitter, l'humanité est encore bien loin de se douter que quelques années plus tard, des I.A. seraient suspectées d'interférer avec nos processus électoraux[20] ou même de précipiter une communauté à la scission[21].
Les faux commentaires, les faux amis ou les faux profils se créent et se monnaient. Ils influencent nos choix avec la complicité, ignorante ou assumée, des I.A. et autres agrégateurs ; ils deviennent des moyens de pression, de concurrence parfois déloyale, voire d'extorsion. Des réputations se font et se défont au gré des commentaires[22] et l'alcool numérique frelaté coule à flots. Si c'est écrit c'est sûrement vrai, alors si c'est sur le web c'est encore plus vrai…non ?

Nous touchons là à deux problèmes fondamentaux des algorithmes et de l'apprentissage des I.A. : d'une part l'absence de transparence, souvent assumée ; et d'autre part l'absence de contrôle, de supervision et souvent de responsabilité.

Les algorithmes de nos I.A. sont jalousement tenus secrets. Leurs codes et leurs paramètres sont aussi bien protégés que ceux des logiciels de microtransactions boursières (« fast-trading »). Ils constituent un avantage compétitif, un différenciateur fondamental et à ce titre, un secret industriel et commercial. Pas question donc d'aller décortiquer leurs rouages ou leurs mécanismes de notation (« scoring »).
Dans « L'amour sous algorithme », Judith Duportail, se heurte, dans un élan à la fois légitime et un peu narcissique, à la mécanique obscure de Tinder. Qu'est-ce qui peut bien influencer son score personnel de désirabilité ? Quelle part de ségrégation sociale, quelle sorte de biais volontaires ou non gouvernent nos applications de rencontre ? Suis-je condamnée à ne rencontre que des gens de « mon milieu » ? Dois-je payer pour voir plus ou pour voir plus de diversité ? (Pour ma part, je me demande si je rencontrerai un jour Sharon Stone[23])

Outre la qualité et la représentativité des données d'apprentissage, les questions fondamentales de la fin de l'apprentissage, de la tolérance aux changements, à l'évolution, à la déviation se posent aussi à nous. Quand un algorithme qui continue à apprendre mute et s'adapte aux

[19] A peine lancée, une intelligence artificielle de Microsoft dérape sur Twitter (lemonde.fr)
[20] Scandale Facebook-Cambridge Analytica — Wikipédia (wikipedia.org)
[21] Brexit et Facebook : Cambridge Analytica aurait joué un "rôle crucial" (sudouest.fr)
[22] TripAdvisor. Il crée un faux restaurant à Londres, qui devient numéro un de la capitale (ouest-france.fr)
[23] Sharon Stone bloquée de l'appli de rencontres Bumble | Le HuffPost (huffingtonpost.fr)

données qu'il ingère, comment peut-on s'assurer que des barrières morales suffisantes sont mises en place ? Comment s'assurer que la cible, mouvante, reste assez représentative de l'ensemble et qu'un équilibre nécessaire et suffisant est respecté ? Comment s'assurer que l'I.A. ne dérape pas quand l'opacité et le manque de contrôle indépendant rendent les réponses encore plus difficiles ?

S'il n'y a pas le moindre doute que les concepteurs et les fournisseurs d'I.A. se sont déjà penchés sur le problème, la mine crispée de leurs dirigeants face aux questions parlementaires, de même que les nombreux exemples cités précédemment, peinent à convaincre. On en vient sérieusement à craindre qu'ils ne soient dépassés par la réalité et que leurs « deus ex-machina » ne tiennent plus du monstre de Frankenstein que de Promethée.

Alors que d'aucuns me vantent déjà les promesses d'I.A. surpuissantes, capables de résoudre et d'embrasser à elles seules des problématiques que l'Humain, trop limité, ne peut plus appréhender ; je m'interroge sur la pertinence des solutions qu'elles proposeraient.
Problème purement rhétorique s'il en est puisque, par définition, je ne peux pas la comprendre. Voilà qu'il me faudrait donc faire aveuglement confiance à une I.A. préprogrammée par des humains (faillibles), puis laissée seule à elle-même pour résoudre des problèmes que ni eux ni moi ne pourrions résoudre. Sachant que quoi par la magie du biais de confirmation, cette solution sera forcément bonne. Sans doute ne suis-je pas digne de ce cadeau, mais je laisse volontiers à Isaac Asimov son robot qui rêvait.

Les récents développements en matière de vie privée (R.G.P.D., pour ne citer que lui) ou du droit à l'oubli, viennent encore ajouter une couche de complexité supplémentaire à un sujet qui n'en manquait pourtant pas.
Le Web s'inscrit par nature dans la persistance de l'information. Après une formidable phase de concentration qui a vu l'hégémonie des GAFAM, les moteurs d'indexation (de profils, de pages…) se diversifient et se multiplient à nouveau et rendent l'oubli encore plus improbable. Ainsi, chaque année, Facebook me rappelle l'anniversaire de collègues morts depuis dix ans – le problème n'est pas que je veuille les oublier, loin de là, mais plutôt que leurs données restent en quelque sorte figées dans une boucle temporelle, alimentant peut-être des I.A. pour une relative éternité. Cette persistance de l'impact après la mort, loin des notions de mémoire ou d'héritage, pose la question d'une certaine obsolescence non-programmée de certaines I.A.[24] et du renouvellement continu ou non des sources d'apprentissage.

Technologie et éthique ne font pas toujours bon ménage. Le récent licenciement par Google de Timnit Gebru[25], chercheuse spécialisée dans la problématique des biais algorithmiques dans les I.A., en est un témoignage criant.
Par-delà son licenciement, qu'il ne m'appartient pas de commenter, elle a initié un débat qui porte sur la qualification et la taille des jeux de données d'apprentissage. En simplifiant à l'extrême, nous aurions d'un côté, ceux qui, comme Google, champion de la donnée, soutiennent

[24] Il y aura bientôt plus de morts que de vivants sur Facebook (franceinter.fr)
[25] Timnit Gebru — Wikipédia (wikipedia.org)

que « qui peut le plus peut le moins » et prônent que la représentativité de l'échantillon vaut avant tout par sa taille (plus on dispose de données, plus on colle à la réalité) ; et de l'autre ceux qui, comme Timnit Gebru, soutiennent que « manger plus que l'on ne doit fait plus de mal qu'on ne croit » (des données plus réduites pourraient être tout aussi représentatives, mais auraient un impact environnemental bien moindre).
Si le débat n'est pas neuf – il hantait déjà les environnements de tests quand l'I.A. n'existait encore que dans d'obscurs laboratoires et dans les super productions hollywoodiennes – il prend ici un sens particulier en positionnant le « GreenIT » au milieu du débat. Voilà que non seulement l'I.A. se confronte aux problèmes de biais, mais encore se heurte-t-elle à un conflit de valeurs à portée éthique.

Si l'impact énergétique et écologique de notre IT n'est plus à démontrer[26], il reste encore à l'intégrer à nos nouvelles habitudes et à nos récentes politiques en matière de dématérialisation et de transformation numérique.
Par ailleurs, le débat, prend une dimension toute différente dans les rizières du Vietnam ou dans la corne de l'Afrique. Les réfugiés climatiques n'habitent pas San José ou Palo Alto, quand bien même les feux de forêt se rapprochent !

Loin de fumeuses théories du complot, je crains moins l'intelligence artificielle que la bêtise humaine, involontaire ou non, qui l'engendre. L'I.A. est jeune et ces problématiques ne sont ni une question de temps ni de version. Nous devons garder la tête froide et embrasser le progrès avec tous les garde-fous d'usage, pour éviter qu'il ne s'emballe et échappe à notre contrôle ou à notre entendement.

Alors que nous commençons à peine à entrevoir les immenses possibilités de cette nouvelle discipline, de nombreuses questions restent encore à débattre :

Qui donc, dans une I.A., décide de ce qui est bien ou mal, de ce qui convient au plus grand nombre, ou même à ses propres utilisateurs ?
Qui sont d'ailleurs les vrais utilisateurs de ces services qu'on nous offre à tour de bras ? Ceux qui les utilisent au prix de leur identité numérique, ceux qui la revendent à d'autres ou ceux qui se servent de ces profils pour propager tantôt des services, tantôt des produits, tantôt aussi des idées ?
Quel référentiel de valeurs / de données prévaut-il sur les autres ?
Une approche holistique et inclusive est-elle seulement possible, et sinon au prix de quels sacrifices ?
Où se situe exactement le niveau de responsabilité de chacun et à l'aune de quoi le mesurera-t-on ?

A ces questions, la société, nos hommes politiques et nos législateurs devront répondre.

Pour nous, testeurs, le défi est ailleurs.

[26] Quelle est l'empreinte carbone d'un e-mail ? (futura-sciences.com)

Il est dans la conception des jeux de données, des environnements et des scénarios de test qui doivent représenter sans exclure, inclure sans juger, intégrer sans se perdre.

Il est dans la compréhension et dans l'analyse des risques, non pas seulement les risques projet, technique ou métier, mais les risques humains dans une acception ô combien plus large que celle que nous évaluons d'ordinaire.

Il est dans le challenge permanent des approches, dans le test négatif, qui ne s'en laisse pas conter et qui ne cède pas sans combattre aux sirènes technologiques.

Il est dans l'esprit critique, qui nous caractérise et fait notre valeur ajoutée, celui qui nous fait remettre en question tout ce qui semble raisonnable (et parfois plus), celui qui nous fait anticiper la catastrophe.

Il nous faudra pour cela, nous détacher de la technique et nous recentrer (enfin) sur l'Humain.

III.2- Garder la banane : un enjeu méconnu de l'automatisation des tests

par Zoé Thivet

Pourquoi parler banane ?

La première fois que j'ai ouvert le syllabus de niveau Fondation de l'ISTQB et que j'ai parcouru le sommaire, mon regard a tout de suite été attiré par un chapitre bien particulier.

« La psychologie des tests ».

Débutant dans le métier (en fait, à ce stade je n'avais en poche que ma promesse d'embauche), je pensais trouver à ce chapitre un aperçu de ce qui se passe précisément dans l'esprit de quelqu'un qui teste, et en particulier des éléments répondant à ces interrogations :

– Quels états d'esprit, quelles postures psychologiques sont propices à des tests de qualité ?
– Quels sont les traits psychologiques souhaitables quand on est QA ?
– Quels sont les « bugs mentaux » que nous devons au contraire éviter ?
– Mais aussi… qu'est-ce qui peut bien nous motiver à tester encore et toujours les mêmes fonctionnements ? (Conformément à l'idée reçue, je redoutais un peu que ce métier soit ennuyeux.)

Le chapitre sur la psychologie des tests ne dépassant pas les deux pages, ma curiosité est restée sur sa faim. Au fil du temps, c'est en pratiquant que j'ai pu commencer à répondre à mes questions, ainsi qu'en lisant des livres et articles de blog traitant du sujet. Avec plaisir, j'ai cru constater qu'il y en avait de plus en plus.

Le présent chapitre traite des **enjeux motivationnels** qui traversent toute pratique d'automatisation des tests. Il vise à répondre une question simple : **comment garder la banane quand on automatise les tests ?** La réflexion présentée ici s'appuie sur le cadre de pensée établi par Yu-Kai Chou, l'un des pionniers de la gamification.

8 leviers motivationnels : attrapez-les tous !

Rompre la routine d'une maintenance qui s'éternise, mobiliser des développeurs peu friands de tests, apporter un regain d'enthousiasme aux équipes qualité… plusieurs raisons peuvent nous pousser à vouloir ajouter un peu de piquant dans votre démarche d'automatisation des tests.

Fan inconditionnel de jeux vidéo, Yu-Kai Chou a longtemps analysé les différents ressorts qui font qu'un joueur continue de jouer. En effet, pourquoi joue-t-on ? Cela demande du temps, des efforts intellectuels et/ou physiques, la plupart du temps ça ne rapporte rien du tout… et pourtant, quelque chose fait que ça en vaut la peine. Si ce n'était pas le cas, si le jeu ne nous

intéressait pas, on s'arrêterait de jouer, et la Terre continuerait de tourner. Les bons jeux, ceux qui nous procurent l'expérience la plus réjouissante, sont donc d'excellents cas d'étude pour analyser les motivations. Faire profiter d'autres champs d'action des fruits de cette analyse est tout l'enjeu de la gamification.

Yu-Kai Chou a inventorié **8 leviers fondamentaux pouvant éveiller notre motivation à réaliser une action**, quelle que soit cette action. Cette « octade » est désignée sous le nom d'Octalysis. Les 8 leviers, résumés dans le tableau ci-dessous, nous serviront de référence dans ce chapitre. Bien sûr, ce n'est qu'un aperçu d'un modèle très complet, que je vous invite à découvrir par ailleurs en détail dans l'ouvrage de Yu-Kai Chou *Actionable Gamification : Beyond Points, Badges and Leaderboards* (2015).

Levier	Ce qui nous motive
N° 1 : Sens épique et vocation	Contribuer à une quête qui nous dépasse, à quelque chose de plus grand que nous, qui a du sens et parle à nos valeurs.
N° 2 : Développement et accomplissement	Réaliser des accomplissements qui couronnent nos efforts.
N° 3 : Renforcement de la créativité et feedback	Profiter de la liberté d'action, voire d'expression, que nous procure telle ou telle activité, et du fait de pouvoir constater par nous-mêmes les résultats de cette action.
N° 4 : Propriété et possession	Amasser des gains, des choses qui nous sont précieuses, qu'elles soient physiques ou immatérielles, voire symboliques
N° 5 : Influence sociale et connexion	Tisser des liens avec d'autres personnes, que ce soient des liens d'amitié, d'entraide, de rivalité...
N° 6 : Rareté et impatience	Accéder à des choses désirables qui nous échappe la plupart du temps.
N° 7 : Imprévisibilité et curiosité	Découvrir des choses intéressantes et inattendues.
N° 8 : Peur de la perte et évitement	Eviter de perdre ce que l'on possède déjà, ou ce que l'on pourrait posséder ; éviter des situations douloureuses.

Selon Yu-Kai Chou, toutes les choses que nous faisons dans notre vie personnelle ou au travail trouvent leur origine dans un ou plusieurs de ces piliers. Qu'en est-il de l'automatisation des tests ? Une fois que nous saurons précisément ce qui nous plaît (et nous déplaît) dans cette activité, nous serons à même de décupler ces émotions positives.

Attention toutefois, car ces leviers sont puissants ; veillez à toujours les utiliser de manière à ce qu'ils servent **notre plus grand objectif : améliorer la qualité des applicatifs.** Au fil de votre

lecture, vous penserez certainement à certains effets de bord (positifs ou négatifs) du plaisir que l'on prend à automatiser les tests. Il est capital d'anticiper le plus possible ces effets de bord.

Pourquoi aimer (et détester) automatiser des tests ?

En quoi l'automatisation des tests nous procure-t-elle des pensées ou des émotions motivantes ? Ce n'est pas une question que l'on a l'habitude de se poser : en général, on automatise les tests en supposant que seul notre devoir professionnel nous y pousse ; les émotions positives que l'on peut ressentir par ailleurs sont bien moins analysées que nos motifs rationnels d'automatiser ou non un test. En outre, quand une activité nous plaît, on se contente souvent d'en profiter sans se questionner.

Un effet surprenant de ce « scan » des motivations est que, lorsqu'on prend conscience de tout ce qui nous anime et qu'on parvient à mettre des mots dessus, ces motivations se font ressentir encore plus intensément.

Il est intéressant aussi de se demander ce qui, dans l'automatisation des tests, met en péril notre bonne humeur et nous inhibe. Encore une fois, cet exercice ne coule pas forcément de source, on aurait plutôt tendance à chasser les émotions négatives plutôt qu'à les analyser.

Ci-après, ce sont mes propres motivations que je vais disséquer. Il y a de bonnes chances que ces motivations soient partagées par un grand nombre de mes pairs, toutefois c'est aussi l'occasion pour vous de vous demander quelles sont les vôtres. Si vous managez une équipe de QA en charge de l'automatisation des tests, ce sont leurs propres motivations qu'il conviendra de comprendre.

Levier 1 : sens épique et vocation

D'une manière générale, j'aime savoir pourquoi je travaille. « Tu automatiseras ces 10 scénarios » n'est pas une instruction qui me motive. En revanche, en voici une autre qui m'enthousiasme bien plus : « Ces 10 scénarios sont joués et rejoués sans cesse par les métiers, qui n'en peuvent plus. Il faudrait les automatiser pour leur permettre de faire autre chose. » Un autre aspect important est d'avoir une vision fonctionnelle précise de l'application. Il faut automatiser les tests du formulaire BX12, d'accord, mais à quoi sert-il exactement ? Sans cette vision fonctionnelle, les journées se ressemblent cruellement, et notre rôle se limite à traduire mot à mot un texte dont le sens réel nous échappe.

Pour garder la banane, il est donc utile d'**avoir toujours en tête un objectif qui parle à nos valeurs, et de pouvoir comprendre exactement la mission qui nous incombe.** Dans les moments de découragement, cet objectif sera comme un petit drapeau derrière lequel se rallier…

Levier 2 : développement et accomplissement

Terminer une tâche décisive est un accomplissement : par exemple, faire tourner le tout premier test auto d'un projet.

Terminer une série de tâches est un autre type d'accomplissement : par exemple, atteindre le nombre symbolique de 100 tests automatisés sur une application.

Il y en a tant d'autres :

- attraper un bug grâce à un test auto
- résoudre un problème de scripts qui provoquait tant de faux positifs
- réussir à interagir avec un élément un peu réfractaire de l'application à tester
- intégrer un nouvel outil au projet d'automatisation des tests

A contrario, il est puissamment démotivant d'avancer dans le brouillard, sans véritable jalon pour marquer l'avancement. De même, lorsque les journées d'automatisation se suivent et se ressemblent, la routine risque de gommer les émotions positives liées à ces petits accomplissements.

Pour garder la banane, il est important de pouvoir **célébrer régulièrement les petites victoires** qui ponctuent nos projets d'automatisation des tests.

Levier 3 : renforcement de la créativité et feedback

Le test est une activité créative. Emettre des hypothèses, les mettre à l'épreuve, se plonger dans le système pour en sonder les moindres recoins… Tout cela requiert astuce et inventivité. Ressentir cette liberté de création jusqu'à atteindre un agréable état de concentration où on ne sent pas les heures passer, voilà une puissante source de motivation !

Dans le domaine de l'automatisation des tests, ce sentiment particulier peut être atteint lorsque l'on acquiert une maîtrise technique suffisante pour avancer sereinement, tout en rencontrant suffisamment de challenge pour ressentir une certaine stimulation.

Pour garder la banane, il faut **cultiver ces moments où l'automatisation des tests est une activité créative.** Selon ce qui vous plaît le plus : réusiner astucieusement du code existant, étendre la couverture des tests automatisés, repenser la gestion des jeux de données…

Levier 4 : propriété et possession

Le sentiment de propriété et de possession procure un certain confort. On peut le ressentir par exemple en constituant, script après script, un beau patrimoine de tests automatisés.

Ce sentiment peut être en revanche inexistant lorsqu'on arrive sur un projet historique et que notre travail consiste simplement à appliquer des procédures auxquelles on n'a pas contribué. Rédiger aveuglément des scripts selon une méthode qui ne nous convient pas, se greffer sur du code qui nous donne des cauchemars… Voilà qui est tout à fait démotivant.

Pour garder la banane, il faut pouvoir **s'approprier la base de code** d'une manière ou d'une autre. La documentation du projet d'automatisation des tests peut également faire l'objet de cette appropriation, tout en renforçant la conscience que l'on a de notre maîtrise du sujet.

Levier 5 : influence sociale et connexion

L'automatisation des tests n'est pas, et ne devrait pas être, un travail solitaire. D'une part, le projet de tests automatisés a pour objectif de communiquer des informations importantes au sujet de la qualité d'un applicatif ; d'autre part, comme tout projet de développement, il se doit d'être compréhensible pour les autres collègues (existants ou futurs).

Nous sommes aussi régulièrement amenés à former d'autres personnes, ou bien à recevoir nous-mêmes des formations de la part de mentors ou de pairs.

Ces liens sociaux ne sont pas seulement utiles mais aussi motivants. Et même lorsqu'on n'a pas la possibilité de cultiver ces interactions en temps réel, il est toujours possible de le faire en différé. Quelques exemples :

- Lorsque vous écrivez des commentaires dans le code, imaginez la personne qui les lira ; adressez-vous à elle.
- Rédigez des documentations digestes (en y laissant des touches d'humour bien à vous, si vous le pouvez !)
- Quand c'est possible, personnalisez les jeux de données de vos projets de tests automatisés ; vos collègues se souviendront longtemps de José Paldir et de Guillaume Sweethome…
- Si vous ressentez de la solitude au sein de votre organisation, ouvrez-vous vers d'autres communautés. Un grand nombre de QA sont sur StackOverflow (une plateforme gamifiée par excellence) et rivalisent d'ingéniosité sur les outils d'automatisation des tests ; vous pouvez à votre tour « sauver la journée » d'un confrère bloqué par un problème. D'autres communautés plus conversationnelles fleurissent, tant sur LinkedIn (par exemple le groupe « Le Métier du Test ») que sur Slack, ou beaucoup d'autres plateformes ; elles vous permettront de sortir de votre isolement automaticien.

Bref, pour garder la banane, automatisez sans devenir vous-mêmes un robot, et **multipliez les interactions de qualité** avec d'autres personnes.

Levier 6 : rareté et impatience

« Ok, j'ai ce test à automatiser, et j'ai toute la journée pour m'y consacrer. Ça devrait le faire. » Super, on dirait que vous avez du pain sur la planche et que vous avancez en confiance. Mais ça n'a pas l'air foncièrement amusant…

Vous connaissez peut-être la loi de Parkinson, un essayiste anglais ayant formulé en 1957 : « Le travail s'étale de façon à occuper le temps disponible pour son achèvement ». En gros, si vous avez la journée devant vous pour automatiser ce test, vous aurez tendance à le terminer juste avant de rentrer chez vous. Si on vous avait donné quatre heures pour réaliser ce travail, il est probable que vous l'auriez fini un peu avant d'aller déjeuner. Et si vous jouiez avec cette loi ? Vous **fixer des défis temporels** est un ressort possible pour garder la banane, et d'ores et déjà une forme simple de gamification.

Varier les activités est en outre profitable à plusieurs titres. Si vous aimez automatiser des tests ad libitum, vous aimerez encore plus cette activité si vous vous fixez délibérément une timebox, qui vous permettra par ailleurs de garder la tête hors du guidon et de vous concentrer sur d'autres aspects importants de votre démarche de test.

Levier 7 : imprévisibilité et curiosité

On touche là à un puissant levier à double tranchant. Il arrive en effet que l'automatisation des tests deviennent, après l'enthousiasme ressenti en début d'implémentation, une activité un peu monotone, un peu ronflante.

Encore une fois, varier les activités de façon à quitter les problématiques purement automaticiennes permet d'introduire des éléments imprévisibles dans ces journées parfois très linéaires et d'éveiller notre intérêt sur des aspects qui méritent notre attention.

Il est important aussi de régulièrement revoir ce qui a été fait précédemment sur le projet : comment pourrions-nous faire varier les jeux de données présents dans les tests de façon à augmenter le rappel des anomalies ? Comment notre framework actuel pourrait être amélioré ? Quelle approche pourrait-on essayer ? Echanger avec vos collègues sur ces types de sujets peut rafraîchir le regard que vous portez au projet d'automatisation des tests, ainsi qu'au système à tester.

Si vous arrivez au bout de vos capacités d'innovation, vous avez un recours infaillible : la prolifique blogosphère des QA ainsi, les nombreux livres sur la discipline de l'automatisation des tests, ou encore les défis qui peuvent s'offrir à vous (le Ministry of Testing en propose régulièrement)…

Pour garder la banane, il est essentiel de toujours **continuer d'apprendre des choses.** Plus vous en saurez et plus vous vous rendrez compte de la somme immense de découvertes qu'il vous reste à faire… Vous ne vous ennuierez plus jamais !

Levier 8 : peur de la perte et évitement

La peur est certes un sentiment désagréable, mais c'est aussi un moteur puissant qui nous pousse à donner le meilleur de nous-mêmes. En tant que QA, nous pouvons avoir une trouille bleue :

- De laisser passer un bug en prod
- Que l'applicatif que nous testons nuise à des milliers d'utilisateurs (voire des millions) (ou alors 1 ou 2, mais gravement)
- De perdre en crédibilité auprès d'autres membres de l'équipe, parfois convaincus qu'en tant que spécialistes en qualité nous nous devons d'être absolument parfaits…

Sur leur CV, de nombreux testeurs indiquent qu'ils sont rigoureux ; mais ce que j'entends derrière cet adjectif un peu impersonnel, c'est « J'ai peur que la qualité soit mauvaise et fais tout ce qui est en mon pouvoir pour limiter ce risque. » Le pessimisme professionnel est particulièrement fécond en qualité logicielle.

Ce levier n'est pas forcément le plus sympathique, mais il résonne avec d'autres, notamment le

1er, car lorsque l'on poursuit une quête épique on craint aussi de ne pas y parvenir, et le 2ème, car on a peur de perdre notre « avancement » (c'est le regard sévère que l'on se porte), etc.

Ce 8ème levier contribue-t-il à garder la banane ? S'il est tout seul, pas sûr, mais il peut donner un peu de piment à d'autres levier.

Conclusion

Les situations évoquées ci-dessus vous ont peut-être rappelé des souvenirs, agréables ou non, de vos expériences en automatisation des tests. A l'aide des différents leviers, vous êtes maintenant en mesure de mettre l'accent sur les activités qui vous sont les plus motivantes. Cette prise de conscience est également le premier pas pour mettre en œuvre, si vous le souhaitez, des processus gamifiés au sein de vos pratiques.

Bon courage pour vos prochains projets, et surtout gardez la banane ! :)

III.3- Industrialisation et automatisation depuis 1995 vue du terrain

par Bertrand Cornanguer

1- Mise en place des techniques de bas niveau d'automatisation

Qu'entendons-nous par industrialisation des tests, industrialisation du développement ?

Industrialisation signifie que le processus de fabrication, c'est à dire la suite des tâches à réaliser pour fabriquer un logiciel, est établi et se répète avec les adaptations nécessaires pour tous les logiciels fabriqués. Il est intéressant de noter qu'un algorithme est *"une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre une classe de problèmes"* (Wikipedia) ;

Pour cela, les outils d'automatisation des tâches assurent la réalisation d'une partie du processus.

Cela est aussi valable pour concevoir un logiciel, pour automatiser le lancement de traitements en exploitation informatique, automatiser la génération de tests, automatiser le lancement et l'exécution de tests.

Et comment automatise-t-on aujourd'hui ? Avec un logiciel. Cela aussi bien pour une ligne de fabrication automobile, l'injection du carburant dans un moteur, le codage d'un logiciel ou les tests dudit logiciel.

Le développeur est en première ligne. Car il connaît son environnement, le langage qu'il utilise, les entrées et sorties du logiciel qu'il écrit. Il peut ainsi écrire du code pour appeler des fonctions de code, afin de simuler des entrées qui proviendraient d'une interface utilisateur, comme un écran WEB.

Il peut le faire de façon très rapide car c'est un complément à son code et cela fait, de toute façon, partie de sa méthodologie de codage. Naturellement, la démarche du développeur est de définir ce qu'une fonction doit faire, puis de coder jusqu'à ce que ce résultat soit obtenu. Pour vérifier, il simule des entrées dans sa fonction. Cela s'appelle le débogage. Pour cela, il s'aide de ses outils de développement qui proposent de saisir des valeurs et les suivre en exécutant ligne par ligne le code. Gain de temps pour le développeur. Cependant, ce travail, s'il n'est pas enregistré, est perdu. S'il revient sur son code pour le modifier, il devra ressaisir des données et les suivre dans l'interface de développement.

Un complément très rentable à cette démarche est de formaliser ces actions et de les rendre indépendantes de cet outil de développement pourtant très pratique. Pour cela, le codeur écrit du code de test qui sera réutilisé par la suite, afin de vérifier que les fonctions développées donnent toujours le même résultat. Ce sont des tests unitaires de non-régression.

Ici, deux concepts s'opposent. Le premier est de permettre au développeur de coder rapidement et de fournir quelque chose qui fonctionne. C'est le Rapid Application Development.

Le second est d'écrire des tests de régression, assurant sur le long terme que des défauts ne seront pas insérés dans le code. Ce qui prend du temps à coder. Il faut donc que la méthodologie soit adaptée au cycle de vie d'un logiciel jusqu'à son retrait des systèmes. Si le logiciel doit subir de nombreuses évolutions, alors écrire des tests sera rentable.

La suite logique est de centraliser et organiser les tests et d'automatiser le lancement de ces tests.

2- Différents environnements et concepts identiques

En 1995, j'ai commencé comme testeur dans une ingénierie qui installait des logiciels sur des PC grand public. L'enjeu était lié au fait qu'il y avait différentes configurations, localisées dans différentes langues couvrant l'Europe. Le volume était du niveau du million de PCs distribués.

Les logiciels étaient installés sur un OS, Windows 3.10, et le disque dur était dupliqué par un logiciel de type Ghost. Les tests étaient manuels.

Pour vérifier que tous les logiciels avaient bien été installés, j'avais, avec turbo Pascal 6, écrit une interface utilisateur qui permettait de visualiser les sections des fichiers ini du PC et de comparer avec des sections de fichiers type. Puis d'insérer directement les sections ini nécessaires dans les fichiers cibles.

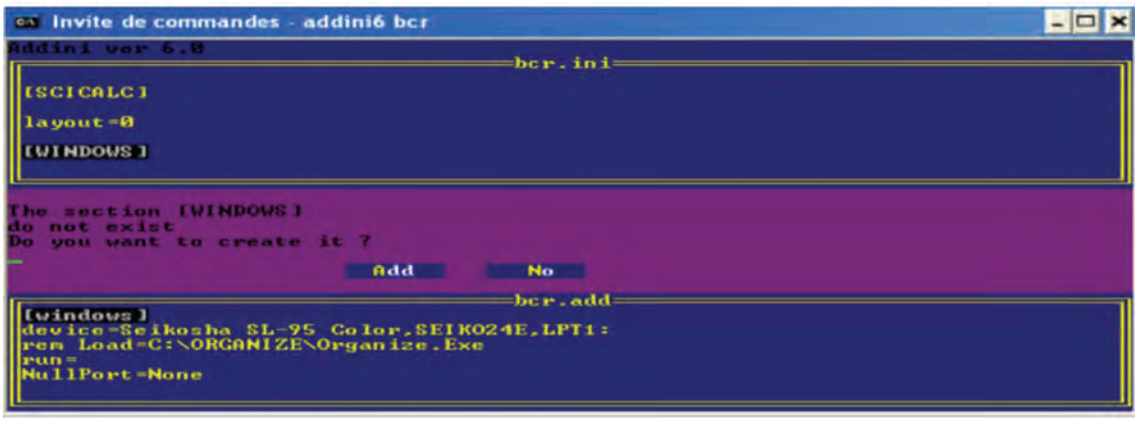
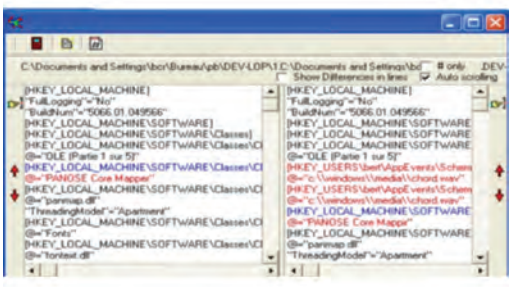


Figure 1 : Ecran de l'outil addini

Cela a permis d'accélérer et la fabrication du Master et ses tests.



Puis windows95 est arrivé. Les installations se faisaient par des fichiers ancêtres des MSI, les fichiers .INF couplés aux fichiers .CAB. Avec la base de registre en plus pour centraliser les paramètres de l'OS et des logiciels installés. Une nouvelle contrainte est apparue. Celle d'installer des logiciels à la demande sur les PC. Donc plus de Masters à dupliquer.

Figure 2 : Ecran de l'outil de comparaison de fichiers .ini et base de registre

Pour ce faire, j'ai automatisé la création de packages logiciels indépendants créés par reverse engineering, à partir des différences entre un PC avec un logiciel installé et un non installé.

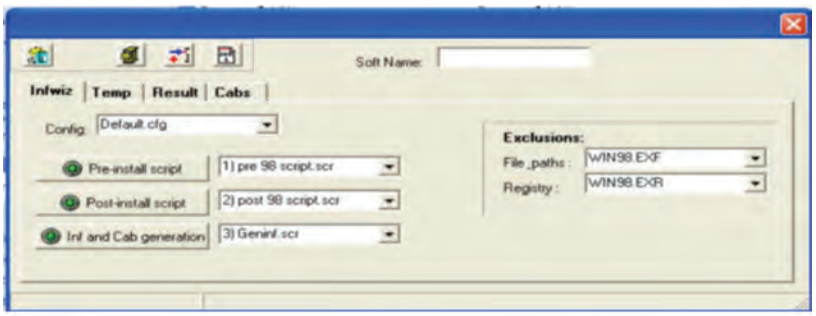


Figure 3 : Ecran de sélection des scripts pour fabriquer les images des packages logiciels

Chaque package logiciel était placé dans une base sur un serveur et il suffisait de sélectionner les packages logiciels à installer automatiquement sur les PCs, la base de registre étant également mise à jour. Intéressant ! J'ai développé ces systèmes avec un outil de développement rapide appelé Delphi. Donc sans tests unitaires. Ce qui s'est révélé handicapant quand je faisais évoluer le logiciel et insérais des bugs dans le code sans le vouloir.

Aujourd'hui, ce concept industriel de téléchargement et d'installation de package est connu dans le monde du logiciel sous le nom de Docker.

L'avantage de ce système est qu'il a permis de simplifier la démarche de réinstallation automatisée en cas de crash. Là encore, c'est un processus automatisé qui exécutait l'enchaînement de tâches, pour réinstaller les packages logiciels depuis une partition cachée du PC.

Là, c'est en explorant le disque dur d'un PC de la concurrence que j'avais remarqué qu'il existait des tables de partitions (qui permettent l'accès aux fichiers) cachées. En les replaçant où il fallait, j'ai vu que des logiciels de test hardware étaient installés sur le disque dur du PC. Donc en production de PC, ces logiciels installés étaient exécutés à la fin de l'assemblage du PC afin de vérifier la mémoire, le disque dur, les bus et autres vérifications de complétude du PC.

Plutôt que de réinstaller les PC à partir de CD Roms, j'avais reproduit ce fonctionnement en plaçant les packages logiciels dans une partition cachée du PC que j'activais quand nécessaire à partir d'un boot du PC. Les applications étaient écrites en Microsoft C, qui avait un environnement de développement plutôt succinct, mais qui avait l'avantage de devoir coder ses tests unitaires pour débugger.

Comme certains voulaient des CDs physiques, j'avais automatisé la gravure des packages logiciels sur CD/DVD. Pour cela, j'ai utilisé les logiciels mis à disposition des clients sur les PCs.

Pour ce faire, MSDN, le kit de développement de Microsoft Windows, expliquait comment interroger les processus de Windows et ses enfants. L'architecture objet de Windows permet de partager des objets, comme une fenêtre, et de les dériver en nouveaux objets, de nouveaux

types de fenêtres. Cette logique s'applique également à la communication entre les objets. Ainsi, il est possible de demander à Windows quelles fenêtres sont présentes, puis à chaque fenêtre quels sont ses enfants, sous-enfants et ainsi de suite.

```
EnumChildWindows(hwnd, EnumChildProc, Longint(button));
PostMessage(button.Objects[1], BM_CLICK, 0, 0);
```

Figure 4 : Exemple de fonctions MSDN pour cliquer sur un bouton d'une fenêtre

Ainsi, il est facile de connaître tous les enfants de toutes les fenêtres et d'agir dessus. Il est donc possible de rendre une fenêtre d'une application invisible, de cliquer sur un bouton ou de saisir des données sans que cela se voit sur l'écran. J'utilisais cela pour graver les CD de restauration, en automatisant les actions du logiciel de gravure installé.

En parallèle, des testeurs, qui ne connaissaient pas le développement sous Windows, utilisaient un logiciel Microsoft de type AutoIT pour automatiser les tests.

Aujourd'hui, cela n'a pas changé. Les testeurs qui ne connaissent pas le développement utilisent des logiciels/libraires d'automatisation des tests IHM et font des tests fonctionnels en utilisant le même concept.

D'une part, on peut constater que l'automatisation des processus de fabrication et l'automatisation des tests logiciels est très similaire et que les concepts d'automatisation du passé sont toujours d'actualité.

D'autre part, on constate que le développeur codeur informatique, qui insère les défauts malgré lui dans le code, a beaucoup plus de compétences et de ressources techniques que les testeurs pour automatiser les tests. De plus, le testeur qui automatise une suite d'actions sur une IHM aura des tests dépendants de l'IHM, de la vitesse des traitements du système d'exploitation. En revanche, le développeur sera limité à son périmètre de développement. Ce qui est contre-balancé par le fait que, statistiquement, les défauts sont insérés pour leur plus grande part pendant le codage.

Les tests basés sur l'IHM devraient donc être réservés :

- Aux technologies ne permettant pas d'interagir avec le code, comme javascript, par exemple.
- Aux équipes de test indépendantes qui n'ont pas accès au code.
- Aux tests d'intégration, aux tests de bout en bout qui n'ont pas d'interfaces pour entrer et vérifier des données. La part des défauts dus à des problèmes d'interfaces, de compatibilités fonctionnelles entre modules, croît avec l'utilisation des API et des services, mais reste moindre que les défauts d'algorithmie.
- Aux tests de l'IHM qui vérifient l'emplacement et le comportement des objets des pages ou fenêtres utilisateur.

3- Concept de test de régression

Une autre technique couramment utilisée a été le « capture rejeu » de journées de traitements informatiques. Cela répond au besoin de grosses organisations qui font fonctionner ensemble une multitude de serveurs, lançant des batchs et différentes applications afin de vérifier que le comportement en production et les durées des batchs sont toujours les mêmes. Ces tests d'intégration à grande échelle, tests complètement big bangs, rendent difficile l'identification de l'origine d'une défaillance. Mais passer une « journée de prod » sans incident est rassurant. En revanche, ce test n'est pas précis pour les cas couverts par les données utilisées. La gestion des jeux de données n'est pas non plus aisée. Ce test doit donc être effectué en plus des tests de composants et applicatifs.

4- Evolution des interfaces

Si les techniques de base sont toujours les mêmes, les interfaces utilisateurs ont évolué. Elles se sont adaptées aux technologies comme le WEB, qui offrent des interfaces utilisateur ergonomiques.

Les années 2000 ont vu apparaître des outils comme Mercury Winrunner ou IBM robot, qui ont permis aux testeurs d'automatiser les tests de régression à partir des IHMs ou les tests de régression des services WEB. Outil de capture/rejeu du comportement des utilisateurs, ces outils nécessitent du codage, mais du codage de test.

Les moteurs « Mots clés » sont apparus vers 2005. Ces interfaces permettent de rendre indépendantes les couches basses, qui exécutent l'interaction avec le logiciel, des étapes de test qui peuvent être fonctionnelles.

A	B	C	D	E	F	G
IE / Frame	Mots clés	méthode d'identification	valeur d'identification	contenu	param4	param5
ie	charger_page			url_site		
ie	remplir_zone	name	loginVis	login		
ie	remplir_zone	name	passVis	mdp		
ie	cliquer_bouton	value	Connexion			
ie	attacher_fenetre			SIG		
ie	attendre_fenetre			IHMC999-Messages		
ie2	fermer_fenetre			IHMC999-Messages		
ie2	focus_frame			principal		
frame_principale	assert_texte_existe?					
frame_principale	remplir_zone	name	nom	test		
frame_principale	remplir_zone	name	prenom	integral31		

Figure 5 : Exemple de script mots clés sous Excel développé en Ruby

Le test peut être écrit sous une forme textuelle mais avec des mots précis référencés. Par exemple, « Rechercher chaussures ». Le moteur saura effectuer une recherche sur un élément « chaussure ». Les scripts ainsi créés par les testeurs peuvent être exécutés par différents outils d'automatisation.

Le plus connu des outils mots clés est Mercury QTP, qui a intégré ce mode dès sa conception. Par la suite, beaucoup de sociétés ont développé le leur, avec leurs propres mots clés, à base de fichiers Excel et de Mercury QTP ou Microfocus TestPartner ou de librairies/langages Open source comme Ruby, Python.

Un outil Open source connu, RobotFramework, est une solution mots clés qui se veut générique. En revanche, pas ergonomique.

Les outils de Model Based Testing sont apparus vers 2010. Basés sur des concepts comme les chaînes de Markov, ces outils ont pour origine les instituts en recherche informatique. A partir de graphiques exprimant une suite d'action effectuées par un système logiciel (le modèle), le MBT identifie les cas de test à réaliser. Les cas de test sont automatisés et devraient être indépendants les uns des autres. Cela permet de créer des enchaînements de test à partir de petits blocs, jusqu'au déploiement si nécessaire.

Si une modification a lieu dans un modèle, les cas de test à modifier sont identifiés visuellement.

5- L'automatisation évolue en parallèle des cycles de vie de développement.

L'automatisation suit les méthodes mises en œuvre sur les cycles de développement logiciel. Ainsi, si le modèle en V de Boehm a décomposé les tests en différents niveaux indépendants, avec leur propres objectifs et entrants pour les tests, les nouveaux concepts Agile ont introduit de nouvelles façons de faire. Le modèle en V est intéressant car les tests sont bien structurés. Si l'on doit faire des tests sur des spécifications, et qu'il est conseillé de préparer les tests lors des phases de conception, pourquoi ne pas concevoir des spécifications de conception qui sont également des tests ?

Ce concept, appelé Test First, donc test en premier, rejoint le concept du Développement Piloté par les Tests cités précédemment. Les tests des niveaux supérieurs au composant sont également conçus avant le codage. Avec une subtilité : les testeurs participent à la conception de ces tests/spécifications. Idéalement, leur automatisation est toujours réalisée par le développeur. Ainsi, les tests du niveau intégration/système seront conçus pour couvrir la conception applicative du logiciel avec les testeurs systèmes. Les tests du niveau acceptation seront conçus par le Product Owner assisté de testeurs.

Souvent, les tests de niveau intégration et système, applicatifs, sont créés avec une notation particulière, qui permet d'automatiser les squelettes de code de test que le développeur devra coder. Cela s'appelle le Behaviour Driven Development et le langage est appelé Gherkin.

Les tests d'acceptation sont appelés Acceptance Test Driven Development et n'ont pas de formalisation précise. Ils peuvent également être automatisés par le développeur. Mais souvent, ces tests nécessitent des environnements de test inter-applicatifs et seront automatisés par les testeurs automaticiens.

```
package dojo;

import cucumber.api.PendingException;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class CourseViewerDefilSteps {

    @Given ("Le stagiaire suit un cours")
    public void Le_stagiaire_suit_un_cours() {
        // Write code here that turns the phrase above into concrete actions
        throw new PendingException();
    }

    @When("il fait défiler les pages")
    public void il_fait_défiler_les_pages() {
        // Write code here that turns the phrase above into concrete actions
        throw new PendingException();
    }

    @Then("Elles s'affichent en moins de 1s")
    public void Elles_s_affichent_en_moins_de_1s() {
        // Write code here that turns the phrase above into concrete actions
        throw new PendingException();
    }
}
```

Fig 6 . Exemple cucumber java

Ces trois types de test TDD, BDD et ATDD seront lancés dans la plateforme d'intégration continue.

6- Conclusion

Finalement, aujourd'hui, les interfaces utilisées par les développeurs pour réaliser des tests de composants et d'intégration diffèrent de celles des testeurs dites « boite noire ».

Pour les développeurs, les tests restent de bas niveau tandis que, pour les testeurs, les interfaces homme machine ont évolué grâce aux retours d'expérience utilisateurs.

Cependant, le marché semble faire évoluer les outils en fonction du retour sur investissement de ces derniers et préfère corriger les défauts dès leur introduction dans les spécifications ou le codage.

C'est pourquoi les outils d'automatisation et d'intégration et de déploiement de type open source semblent attirer les testeurs. D'un autre côté, les testeurs automaticiens ont encore besoin aujourd'hui de connaissances en programmation/algorithmique, d'où leur possible appétence pour l'open source.

Dès lors, les investissements dans les outils de détection de défauts, dans les phases proches de la mise en production, vont-ils continuer ? Ou bien le futur va-t-il voir se développer l'IA dans la détection des défauts dans les spécifications et le codage ?

III.4- Comment prédire et prévenir les anomalies ?

par Marc Hage Chahine

La prévention des bugs, la capacité de les anticiper et de les détruire avant même qu'ils n'apparaissent, ceci est un Graal pour bon nombre de testeurs, de développeurs... mais aussi de responsables de SI ou de sociétés éditrices de logiciel.

Non, je ne parle pas de Minority Report mais plutôt d'études statistiques basées sur des indicateurs, des mesures. Pour cela il faut, comme souvent, s'intéresser à d'autres domaines. Je propose aujourd'hui de se pencher sur la fouloscopie et l'étude du comportement des malfaiteurs. Cet article a été inspiré suite au visionnage de cette vidéo < *https://www.youtube.com/watch?v=HHCiNPtR1NI&t* > qui nous permet d'imaginer 2 manières ou types d'algorithmes pour prédire les bugs.

Il va sans dire que prédire les défauts/bugs/anomalies est la mise en application ultime du principe du test "Tester tôt".

1- Méthode : se baser sur l'historique

1.1- L'inspiration

Cette méthode est basée sur de calcul des risques d'occurrence de délits (cambriolage, agression...). Pour calculer cela, la chercheuse Andrea Bertozzi s'est inspirée du modèle des séismes, en partant du principe que la majorité des délits, comme pour les séismes, ont plus de chances d'arriver suite à un autre délit (ou séisme) et sont donc des répliques. Cette proximité temporelle et spatiale des délits peut sembler « logique » lorsque l'on considère que les crimes sont impactés, au moins en partie, par 3 critères :

- le principe des « suiveurs » : il est possible de commettre un vol dans cette rue, je vais donc le faire également.
- un aspect méthodique du voleur, qui travaille secteur par secteur en finissant un quartier avant de passer à un autre.
- l'effet de « modèle » : on pense ici aux « copycats », des fans de criminels qui reproduisent le comportement criminel par admiration, intérêt ou pour exister.

En partant de ce principe, Andrea Bertozzi a repris l'équation permettant de prédire les séismes pour l'appliquer au crime, le crime appelant le crime...

Les résultats de cette équation permettent de donner en amont, non pas des crimes mais des risques de crimes. Cela donne l'équivalent d'une "météo du crime". Les expérimentations sur le terrain, à travers des applications développées à partir de cette équation, tendent à montrer une bonne efficacité de ce système qui, au final, ne fait pas appel à de l'intelligence artificielle !

Il est intéressant de noter que, comme pour la météo, les prévisions proposées ne sont jamais fiables à 100%. De même, plus la prédiction est éloignée dans le futur, moins elle est fiable. Les risques donnés par l'algorithme ne sont valables qu'à court terme et diminuent fortement au fil des jours si aucun autre délit n'est commis.

Météo du crime
13 février

1.2- Quelle application dans le monde du test ?

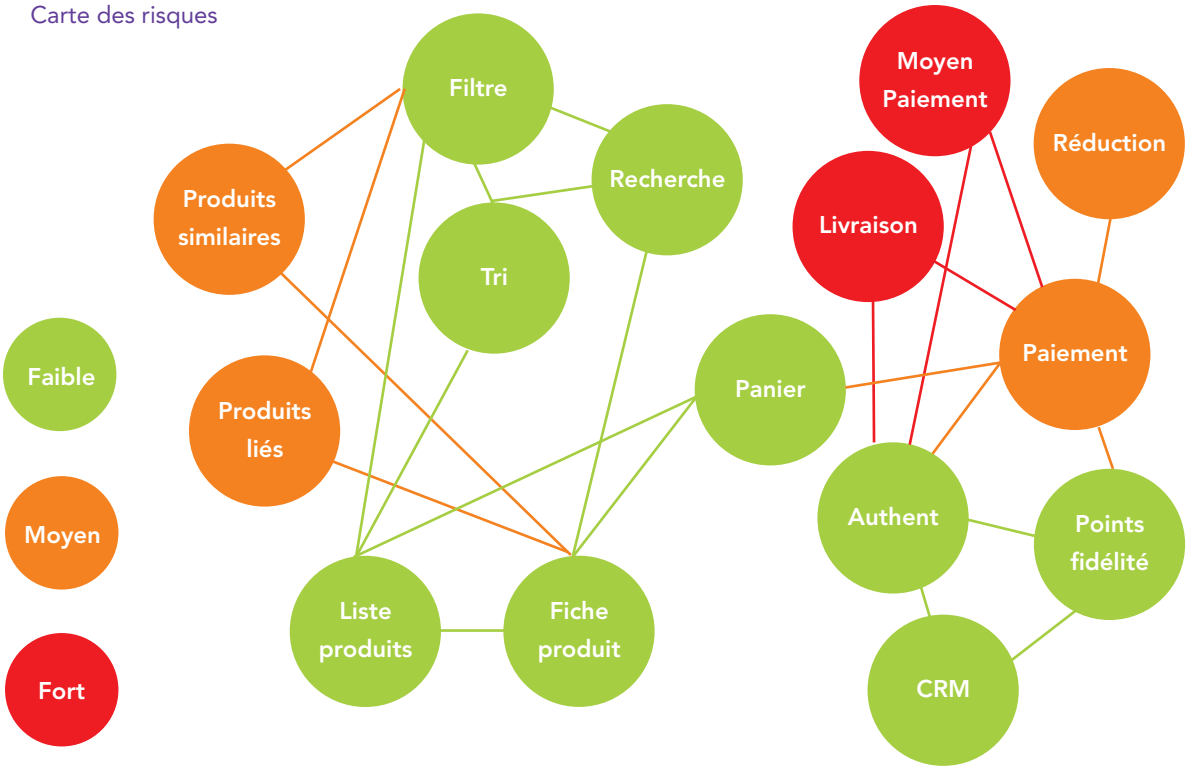
Quel rapport avec le test et plus précisément les bugs ?

Le regroupement des crimes peut être considéré comme une application du principe du regroupement de défauts ! Tout comme pour les séismes, les champignons et le crime, les bugs sont souvent regroupés. Lorsque l'on détecte une anomalie, il y a de fortes chances d'en détecter une autre à proximité. Cette « proximité » peut être liée à des faiblesses inhérentes au logiciel, à un contexte comme une utilisation spécifique et temporaire, l'arrivée de nouveaux utilisateurs, une nouvelle de manière d'appréhender le logiciel ou encore à des changements liés à un développement commun... De manière générale, dans un logiciel, il y a souvent des parties sensibles qui sont plus impactées que les autres par les anomalies. Ce type de bugs est d'ailleurs assez bien repéré par les testeurs lors des sessions de tests exploratoires.

Concrètement, en partant de ce principe, il est possible de contribuer à la prédiction du risque de défaut. Cela pourrait, par exemple, se traduire par :

- Etablir une cartographie sous forme de réseau de relations (comme pour les réseaux sociaux) entre les différentes fonctionnalités ou les différents composants. Chaque bug détecté augmentant le risque de défaut pour une durée de X (nombre à définir en fonction du contexte), sur la (les) fonctionnalité(s) ou le(s) composant(s) impacté(e)s par celui-ci. Cette cartographie du risque permettrait de sélectionner différemment les tests.

Version à tester : v 1.2.X
Carte des risques



- Une analyse des résultats des tests et des anomalies remontées par ces derniers, pour définir des probabilités d'échecs de chaque test et donc aider à la sélection et la priorisation.
- Une analyse des commits et du code modifié pour tenter de dégager des similitudes entre les différents commits ayant engendré récemment des anomalies.
  Ce principe est d'ailleurs utilisé par Microsoft < https://www.microsoft.com/en-us/AI/ai-lab-code-defect > avec l'outil Code Defect Prediction développé par Altran. Cette IA analyse un grand nombre de paramètres d'un commit pour prédire un risque d'introduction d'anomalie. En fonction de ces paramètres et des résultats des commits précédents, l'IA donne, comme pour un test de santé avec « malade » ou « sain », un statut « Buggé » ou « Non buggé ».
- Etablir une cartographie par type d'utilisateur impacté sur les bugs détectés, afin de proposer une cartographie des risques par personae et de définir ou orienter ses tests en fonction…

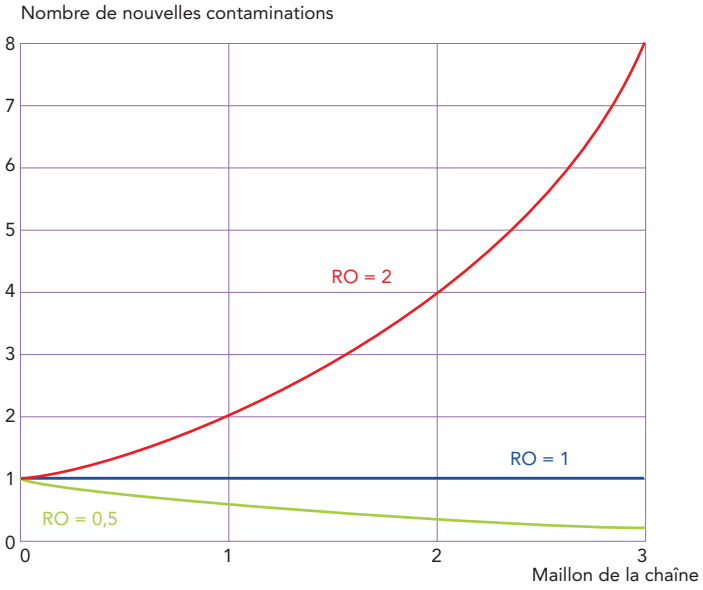
A noter, pour toutes les solutions proposées, l'IA peut offrir une aide précieuse pour analyser un grand nombre de données. Elle n'est cependant pas indispensable et est sujette à de nombreux biais. Il faut se rappeler qu'elle reste principalement un outil d'aide à la décision.

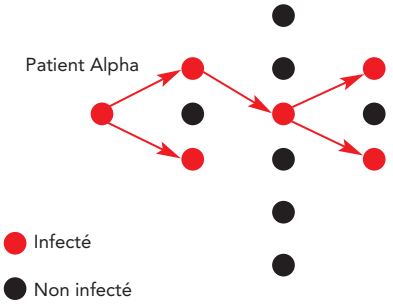
2- Méthode : protéger les "proches" à la suite d'un bug

2.1- L'inspiration

Cette méthode prend racine dans l'analyse des guerres de gang et présente une analogie avec l'épidémiologie. Elle part du postulat que les assassinats "volent en escadrille", c'est-à-dire qu'il y a des périodes de calme et des périodes avec une suite d'assassinats (on s'en aperçoit malheureusement aussi avec les attaques terroristes...). Des études ont montré que les assassinats pouvaient être modélisés comme des épidémies avec un R0 < https://fr.wikipedia.org/ wiki/ Nombre_de_reproduction_de_base > fort heureusement inférieur à 1.

Pour rappel, un R0 (nombre moyen de personnes contaminées par un porteur) inférieur à 1 induit une baisse jusqu'à une disparition des infections, alors qu'un R0 supérieur à 1 induit une évolution exponentielle des infections.





Pour faire simple, il y a une "victime alpha". Cette victime alpha peut transmettre sa "maladie" (se faire assassiner) pendant une période de "gestation". Au-delà de cette période, les chances des relations de la "victime alpha" de se faire assassiner reviennent à la normale. Si, au contraire, pendant cette période de gestation, des contacts de la victime alpha se font assassiner, alors la chaîne de propagation probable s'étend aux contacts des contacts assassinés... On voit ici que l'on est très proche des problématiques rencontrées avec les maladies infectieuses comme la Covid 19.

Afin de diminuer ce nombre d'assassinats (on parle de chaînes pouvant dépasser les 30 homicides), il faut adopter les mêmes principes que pour une épidémie. Il "suffit" donc de briser la chaîne de propagation. Avec la Covid 19, on détecte les personnes infectées puis on gère les "cas contacts" en leur demandant de se faire "tester".

2.2- Quelle application dans le monde du test ?

Le parallèle avec le logiciel se révèle particulièrement intéressant ! Ce type de raisonnement est d'ailleurs très proche du principe du regroupement de défauts, vu avec l'historique, mais l'utilise différemment. Ici, les « proches », ce sont les contacts du composant, les fonctionnalités échangeant avec le composant, la partie de code ou la fonctionnalité impactée par l'anomalie. De même, cela se rapproche également du rôle des campagnes de régression qui, suite à un changement qui n'est pas parfaitement contrôlé, se mettent à son tour à impacter d'autres zones du logiciel… A la différence que, pour la régression, ce qui déclenche les vérifications n'est pas une anomalie mais une modification du logiciel.

Encore une fois, cette théorie semble adaptable au monde du logiciel. Lorsqu'une fonctionnalité a des anomalies, elle impacte forcément d'autres fonctionnalités avec lesquelles elle échange… Ou encore des fonctionnalités qui ont des parties de code communes. Ces mêmes autres fonctionnalités seront impactées par un changement sur la première et ce, même si la modification est un correctif. Il arrive fréquemment que des modifications soient faites sur un logiciel pour corriger un bug. La correction de ce bug peut entraîner des régressions, c'est pourquoi il faut, en plus de faire des tests de confirmation (ou retest), ajouter des tests de régression pour s'assurer que la correction d'une anomalie n'a pas entraîné d'anomalies plus importantes.

Concrètement, on s'oriente plus ici sur de la prévention que sur de la prédiction de défauts. L'utilisation d'une logique similaire pourrait, par exemple, prendre la forme d'une mise en place d'un "protocole sanitaire" après la découverte et la correction d'un bug. Ce protocole pourrait alors ressembler à celui décrit ci-dessous :

1- Corriger l'anomalie

2- **Identifier et prévenir les "contacts"** du code, du composant et/ou de l'anomalie

3- **Tester les cas contacts,** pour assurer que ces contacts ne sont pas infectés ou que la correction n'a pas introduit d'autres anomalies

4- Corriger les anomalies des cas contacts infectés

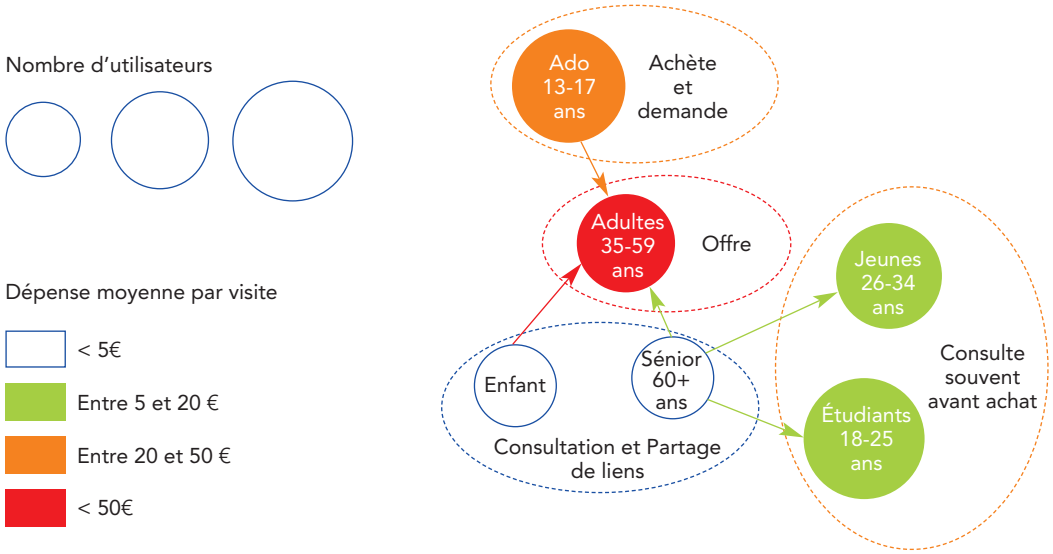
5- Identifier les contacts des cas contacts infectés

6- Tester les contacts des cas contacts infectés...

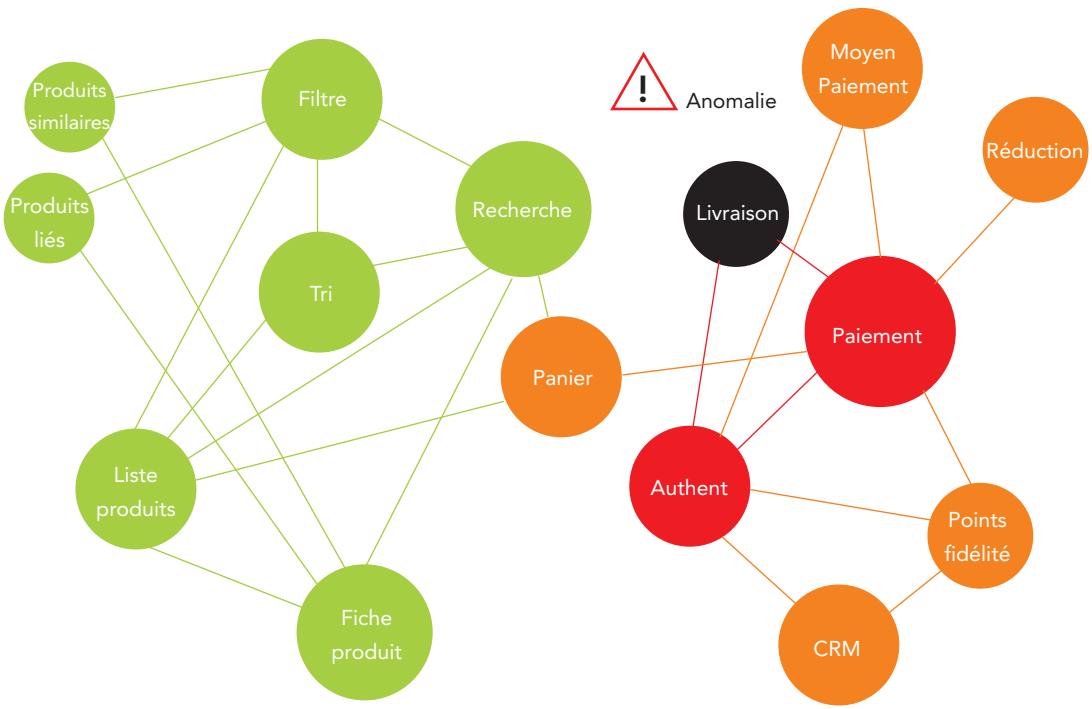
Cela permet de prioriser des tests en fonction de risques avérés et de prévenir des défauts. Là encore, on n'est pas forcément sur de l'IA même si cette dernière peut fortement nous aider à identifier les "contacts".

Il existe déjà des méthodes et outils de tests permettant la mise en place de cette protection des proches. Je pense notamment à des cartographies comme des cartographies d'utilisateurs

(les type d'utilisateurs étant les « proches ») :



Ou des cartographies du logiciel avec localisation des anomalies :



Effort de test supplémentaire :

- Aucun
- Tests exploratoire 30 minutes
- Tests exploratoire 1 heure
- Tests exploratoires 1 heure + nouveau tests de régression

3- Méthode : empêcher l'apparition de foyers infectieux, assurer un « sentiment de sécurité »

3.1- L'inspiration

Toujours dans le domaine de la criminalité, on trouve une autre théorie qui mérite notre attention. Elle est nommée « Hypothèse de la vitre brisée ».

Cette théorie montre que ce n'est pas la délinquance qui cause le sentiment d'insécurité mais plutôt les incivilités. Elles engendrent un sentiment d'impunité favorable au passage à l'acte, car ce modèle montre que la criminalité émerge de petits détails qui transforment un quartier paisible en une véritable jungle.

Cette approche explique qu'une attitude de renforcement des liens sociaux est plus efficace qu'une augmentation des effectifs de police. Des solutions permettant aux citoyens de se sentir plus en sécurité pourraient donc être :

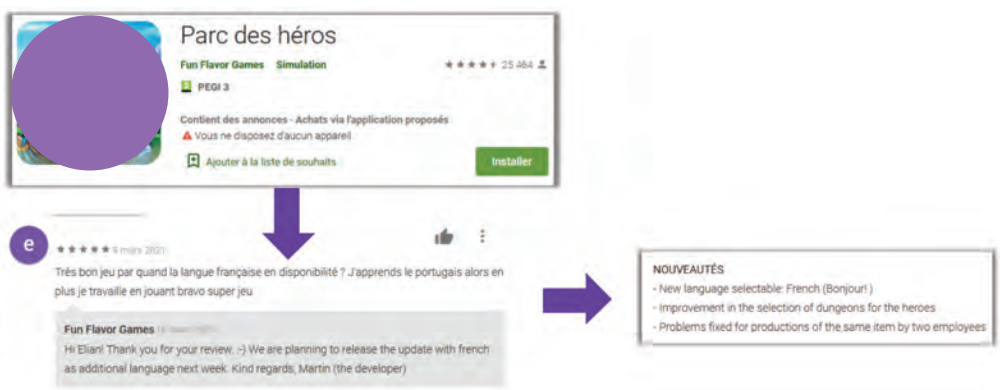
- La mise en place de policiers dans les zones problématiques, dont la principale mission serait d'être présents et de connaître la population
- Avoir des agents à l'écoute des plaintes des citoyens
- Prendre en compte des retours des citoyens et résoudre les problèmes remontés

De même, on s'aperçoit que le nombre est parfois plus problématique que l'impact réel des problèmes vus par les habitants. On parle ici de « sentiment d'insécurité ». Par nombre, il faut entendre ici le nom concret d'actes d'incivilité mais aussi le nombre de fois où l'on passe devant le résultat d'une de ces incivilités comme, par exemple, une voiture brûlée qui reste des semaines dans une allée passante.

3.2- Quelle application dans le monde du test ?

Concrètement, la question ici est plus celle d'un niveau de gestion de la qualité, et plus particulièrement de la qualité ressentie par les utilisateurs. Ainsi, plutôt qu'augmenter l'effort de test appliqué une fois les bugs apparus, il est plus efficace de traiter et d'éviter les petites « incivilités ». Pour cela, des approches à forte valeur ajoutée telles que :

- La mise en place de mesures en production (shift right), afin de repérer rapidement des anomalies
- Avoir un support disponible et à l'écoute des utilisateurs
- Corriger en priorité les anomalies remontées par les utilisateurs et communiquer lors de leur correction.



De même, il est possible dans le logiciel de lutter contre le « sentiment d'insécurité », qui deviendrait plutôt ici un « sentiment de mauvaise qualité ». Dans ce cas, la mise en place de bonnes pratiques liés à la qualité peuvent être des solutions. Ces bonnes pratiques peuvent être :

- Le Software Craftsmanship avec du refactoring de code régulier et maîtrisé par du TDD et combiné à l'ATDD
- L'implication des tests au plus tôt, notamment avec des ateliers de type « 3 Amigos » pour obtenir de « bonnes » US
- Une socialisation renforcée, par exemple avec du pair programming, voire du mob programming et plus généralement avec des équipes de type X-Team, équipes améliorées qui intègrent des activités de réseautage et des rôles d'ambassadeur auprès des autres équipes, des clients et des experts.

4- Conclusion

Prédire efficacement les bugs à 100% est impossible. C'est également le cas pour les séismes, les délits, la propagation des épidémies et les homicides. Néanmoins, il existe des méthodes efficaces permettant d'évaluer certains risques et donc de mettre en place des actions pour les diminuer. Les modèles proposés dans d'autres disciplines que le logiciel sont, au moins partiellement, transposables au monde du test logiciel et plus précisément au monde du bug. La mise en place de certaines de ces méthodes (sans nécessairement avoir recours à de l'IA) permettrait, dans des contextes appropriés, de détecter plus tôt (voire de prédire et prévenir) des anomalies, limiter des chaînes de propagation, en proposant l'équivalent d'une "météo" des anomalies qui donnerait un taux de risque de présence ou d'introduction de bugs.

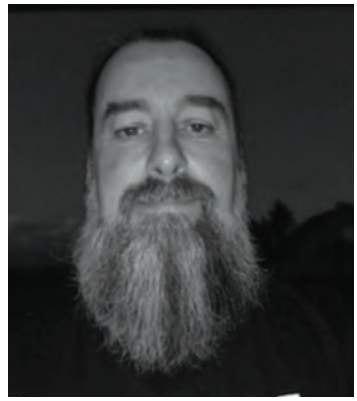
Merci à Benjamin Butel, Christophe Moustier, Olivier Denoo et Michael Granier pour leur contribution à cet article.

PARTIE IV
LES CONTRIBUTEURS

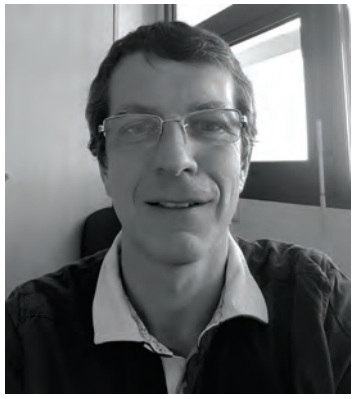
Mohamed Mehdi ADDOU

Test Engineer au sein de la Tribu Digital et Marketing sur l'espace client AXA depuis 2 ans et demi, Mohamed Mehdi Addou est le référent sur l'automatisation de test mobile et accompagne les automaticiens de la tribu.
Certifié ISTQB Fondation et en cours de préparation pour le niveau « avancé », il partage régulièrement l'expérience des outils internes d'AXA au sein de la guilde avec ses pairs.

Fabrice AMBERT



Fabrice AMBERT est maître de conférences en informatique à l'Université de Franche-Comté et membre du laboratoire Femto-St. Depuis une vingtaine d'année, il travaille sur différents aspects du test, modélisation pour le test, génération de tests à partir de modèles et plus récemment l'ATDD visuel.
Depuis 2016, il anime des formations sur l'automatisation des tests pour la société Certilog.



Frédéric ASSANTE DI CAPILLO

Test Manager depuis plusieurs années à Amadeus (Sophia-Antipolis), Frédéric attrape le virus de la Qualité dès 1999. Ayant participé à de nombreux projets pour mettre en place des équipes de validation mais aussi à la transformation agile de la Qualité, il accumule une grande expérience dans ce domaine.
De nombreuses présentations, tel que le « Cargo Culte » lors de la JFTL 2018 ou les « Tests Exploratoires » lors de la soirée du test logiciel à Sophia-Antipolis lui ont permis de partager ses connaissances.
Vice-Président du club Ecume depuis de nombreuses années et membre de la Telecom Valley à Sophia, il a pu confronter et enrichir ses expériences et connaissances avec de nombreux acteurs du monde de la qualité.

Elodie BERNARD

Elodie BERNARD est consultante testing confirmée au sein de Sogeti.
Elle est titulaire d'un doctorat en Informatique, réalisé dans le domaine du Test logiciel à l'Institut FEMTO-ST (Besançon, France).
Elle travaille sur la mise en pratique d'une nouvelle approche de conception de tests visuels pour l'automatisation des tests et la gestion du processus de test.

Didier BIRENBAUM

Didier Birenbaum est avant tout passionné par le test et tout particulièrement par l'automatisation des tests. Il partage cette passion à travers son activité de formateur ISTQB (Analyste technique de test et Automatisation des tests) ainsi que dans son rôle de responsable de l'automatisation des tests chez Crossknowledge (Solutions de formations digitales - Groupe Wiley).

Xavier BLANC

Xavier Blanc est professeur en Informatique à l'Université de Bordeaux. Il effectue sa recherche en génie logiciel au LaBRI où il dirige l'équipe ProgRes.
Il est l'un des cofondateurs de ProMyze, une start-up dans le domaine de la qualité logicielle.

Julien BOTELLA

Fort d'une expérience de 15 ans dans le domaine du Model-Based Testing chez Smartesting, Julien est aujourd'hui responsable du produit Gravity, ainsi que de projets de recherche collaboratifs, notamment autour des techniques de l'IA pour le test. Dans le cadre du projet de recherche PHILAE et du produit Gravity, il travaille sur la capacité à créer des tests de régression automatisés à partir de logs utilisateurs.

Bertrand CORNANGUER

Spécialisé dans les tests de logiciels depuis 25 ans, Bertrand Cornanguer a participé à différents groupes de travail du CFTL dont il a été secrétaire et trésorier, et de l'ISTQ pour lequel il a contribué au développement des syllabus de test Agile.
Il est actuellement responsable de www.springit.fr, organisme de formation accrédité ISTQB, qui est spécialisé dans les formations de tests logiciels aux niveaux Fondation, Avancés, Agile et Automatisation de tests, ainsi que dans les formations de reconversion vers le métier de testeur de logiciels.

Aymeric CRETIN

Titulaire d'un doctorat en informatique ayant pour sujet « le test de la résilience des systèmes de contrôle face aux attaques par injection de fausses données », Aymeric Cretin est aujourd'hui ingénieur R&D chez Smartesting où il travaille sur des projets de recherche autour du test logiciel.
Son sujet actuel, en collaboration avec l'institut FEMTO-ST, est la mise en place d'une technique de priorisation des cas de test basée sur l'intelligence artificielle, notamment l'apprentissage par renforcement (RL).

Olivier DENOO

Olivier Denoo est le président du CFTL et de l'ISTQB®.

Il est en outre le vice-président de ps_testware SAS, une ESN pure player du test logiciel, active en France depuis plus de 10 ans.

Il donne des conférences dans le monde entier et est actif dans la qualité logicielle depuis près de 25 ans.

Il œuvre à rapprocher les testeurs francophones au travers d'initiatives locales diverses et à intégrer dans une approche globale plusieurs schémas de certification de la qualité logicielle (ISTQB, TMMi, IREB, IQBBA…).

Jean-Prince DOTOU-SEGLA

Enrichi de ses 12 ans d'expérience dont 4 ans de Dev Web/Tech Lead, puis 2 ans en tant que Test Lead automaticien, Jean-Prince Dotou-Segla est aujourd'hui référent sur l'automatisation au sein de l'équipe Ingénierie et Animation de la Guilde Test. Il œuvre à l'étude, au déploiement et à la maintenance de nouveaux outils d'automatisation des tests.

Il a à cœur de définir et diffuser les bonnes pratiques sur les outils d'automatisation.

Animation de la conférence « Test en Continu avec le cloud » à la ParisTestConf 2019.

Etienne DUFOUR

Etienne Dufour a près de 10 ans d'expertise dans le métier du test. Il accompagne les 200 testeurs de la Guilde sur les aspects méthodes et outils, en particulier l'utilisation de gestionnaire de test. Il partage régulièrement l'expérience d'AXA avec d'autres sociétés. En 2018, il a lancé le Club des Utilisateurs du gestionnaire de test utilisé par AXA France.

Animation du Tutoriel sur la pratique et l'outillage des test en agile lors de la JFTL 2019.

Vincent DUGARIN

Test Manager, Vincent Dugarin a 13 ans d'expérience dans le test logiciel.
Il est responsable d'équipes de tests, garant de la qualité et de la maturité des pratiques sur le périmètre des assurances collectives.
Plusieurs fois certifié ISTQB au niveau « Avancé », il a animé des présentations et ateliers sur la méthodologie de test en Agile lors d'événements comme les JFTL et l'Agile Tour.

Animation du Tutoriel sur la pratique et l'outillage des test en agile lors de la JFTL 2019.

Adrien FRANCO

Adrien Franco a commencé son aventure dans le monde des tests par 2 ans de tests manuels. Après avoir été confronté au côté parfois répétitif et chronophage de certains tests manuels, doué d'une appétence pour le développement, il saisit l'opportunité offerte par Altran de s'orienter vers l'automatisation des tests. Cela fait maintenant plus de 12 ans qu'il a la chance de pouvoir évoluer dans ce monde en perpétuel mouvement qu'est l'automatisation des tests.

Jean-François FRESI

Responsable du centre services test pour AUSY, Jean-François Fresi manage plus de 100 testeurs. Avec près de 15 ans d'expérience dans le test et ayant participé à des projets de tests diversifiés à la fois techniques (test de performance, test automatique, test de sécurité) et organisationnels (accompagnement du projet test dans des organisations Agile, SAFe et DevOps), il s'attache à transmettre les bonnes pratiques de tests et à rendre les projets autonomes.

Jean-Paul GALLANT

Entré dans le groupe Altran en 1997, Jean-Paul Gallant constitue une cellule de qualification et de recette dans la filiale Toulousaine Logiqual. En 2001, il intègre en tant qu'ingénieur Conseil la filiale Parisienne MAP d'Altran et met en place une des premières TRA en France chez Bouygues Télécom. Dans le cadre d'Altran, il dépose à l'INPI des méthodes et modes opératoires associés à la stratégie de recette et de qualification des applications informatiques. Depuis 2005, il exerce la fonction de Practice Manager chez Altran.

Michaël GRANIER

Passionné de test depuis plus de 10 ans, que ce soit en entreprise ou de manière personnelle, Michael Granier s'informe, cultive et met en application ses compétences, de façon à faire évoluer le métier et les méthodes qui lui sont associées.
Après 12 ans chez oui.sncf, il a rejoint Mr Suricate en 2020 pour partager son expertise de l'automatisation, avec l'objectif de proposer une solution nouvelle et ambitieuse !

Marc HAGE CHAHINE

Marc Hage Chahine travaille dans le test logiciel depuis son stage de fin d'études en 2011. Il a d'abord enchaîné des missions de testeur Agile avant de commencer à écrire et de rejoindre Altran, où il a pu évoluer jusqu'à son poste actuel d'expert méthodes et outils. Dans le même temps, il a développé une activité publique liée au partage, notamment avec l'écriture de livres, d'articles pour « programmez » ou l'animation du blog « la taverne du testeur. »

Hélène HAING DROIN

Après 10 ans d'expérience opérationnelle dans le test en tant que Test Lead, Hélène Haing Droin a souhaité se reconvertir professionnellement en rejoignant l'équipe Ingénierie de la Guilde Test, en tant que Chargée de mission. Elle veille à l'intégration et à la montée en compétences des collaborateurs de la Guilde Test, en collaboration avec les managers de l'équipe.

Elle s'occupe également de l'animation de la Guilde par la mise en œuvre d'actions de communication interne et externe.

Thomas KUYPERS

Testeur logiciel depuis 10 ans, Thomas Kuypers est passionné d'agilité et plus particulièrement d'Agile Testing.

Il est membre de Nord Agile depuis 2019, il organise des Meet-up de Test via Ministry of Testing Lille ou France et il a également rejoint le comité d'organisation de la Paris Test Conf pour l'édition 2021.

Julien LAVEAU

Julien Laveau effectue son doctorat à Bordeaux, dans le cadre d'une collaboration entre CIS Valley et le LaBRI. Il s'est spécialisé dans le génie logiciel et plus spécifiquement dans le test exploratoire des applications web. C'est au cours de ses travaux qu'il commence à développer AIFEX, un assistant pour les sessions de test exploratoires.

Bruno LEGEARD

Bruno LEGEARD est Professeur de Génie Logiciel à l'Université de Franche-Comté, Conseiller scientifique auprès de Smartesting, Secrétaire du CFTL et membre actif de l'ISTQB.
Son activité est focalisée actuellement sur les tests dans l'agilité à l'échelle, l'ATDD – Acceptance Test Driven Developpement et l'application de l'IA pour les tests logiciels.

Christophe MOUSTIER

Christophe Moustier a commencé à coder dès l'âge de 12 ans sur TI99 4A et s'est tourné vers le test en 2006.
Aujourd'hui il est Test Practice Leader pour inetum et auteur de deux livres sur le test : "Le Test en mode Agile" "et « Conduite de tests agiles pour SAFe et LeSS" (chez ENI).
Christophe est un passionné de qualité logicielle, capable d'adresser l'agilité à l'échelle d'une organisation tout comme la qualité du code (Software Craftsmanship). Son adage favori : "Dis-moi comment tu testes, je te dirai combien tu es agile !".

Bruno PEDOTTI

Depuis 20 ans dans l'Informatique du Groupe AXA, dont plus de 10 ans dans le métier du test.
Responsable de la Guilde Test, communauté de plus de 200 experts du test intervenant sur l'ensemble du SI d'AXA France, Bruno Pedotti transforme le modèle de delivery et la culture du test au sein de la DSI. Fier de son métier, il incarne et porte la vision de la filière métier du test, au sein et en dehors d'AXA.

Membre du Comité Programme de la JFTL.

Eric RIOU du COSQUER

Passionné par la qualité des logiciels et systèmes d'information, Eric RIOU du COSQUER est un membre actif des organisations IQBBA (Analyse Métier), IREB (Exigences), ISTQB et CFTL (Test) et TMMi (Audit). Il réalise des missions d'audit, certifications et accompagnement des sociétés, notamment en tant que Lead Assessor TMMi.

Jean-Michel TESSIER

Grâce à ses travaux de formateur, test manager et directeur de l'offre « La transformation digitale orientée par les tests » d'Oceane Consulting TS, l'objectif de Jean-Michel Tessier est simple : avoir une vision transverse et surtout organisée des tests sur l'ensemble de la chaîne de production. C'est donc naturellement qu'il met en avant la pratique du Continuous Testing.

Zoé THIVET

Zoé Thivet est spécialiste en test logiciel en Nouvelle-Calédonie, au sein de la société Hightest. Elle intervient auprès de multiples organisations calédoniennes en tant que consultante, dans le cadre de missions tant fonctionnelles que techniques. Elle anime également la communauté de crowdtesters de la plateforme Testeum.

Alexandre VERNOTTE

Alexandre Vernotte a obtenu un doctorat en 2015, à l'Université de Franche-Comté, dans le domaine du test de sécurité pour les applications Web. Il a ensuite effectué des travaux liés à la cyber-sécurité des systèmes complexes avec un postdoc de deux ans à l'Institut Royal de Technologie (KTH) à Stockholm, Suède. Depuis 2018, il poursuit un postdoc à l'Université de Franche-Comté. S'attachant d'abord aux techniques de tests et de détection contre les attaques par injection de fausses données (FDIA), ses travaux concernent aujourd'hui l'étude de l'intelligence artificielle pour la priorisation des cas de tests.

Huaxing YUAN

Avec 12 ans d'expérience dans le métier du test, Huaxing Yuan travaille dans l'équipe Ingénierie de test. Il est garant des pratiques, des outils et des plateformes d'automatisation de test. Il anime également la communauté des automaticiens et des formations liées à l'automatisation de test.

Il est certifié ISTQB au niveau « Avancé », passionné de test et des nouvelles technologies.

ISBN : 978-2-9567490-1-1
Dépot légal 2021
Imprimé en France
Charte graphique et mise en page : Socreate