LES TESTS LOGICIELS EN AGILE

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

En guise d'introduction à cet ouvrage...

Depuis plus de dix ans déjà, le CFTL a accompagné sympathisants et membres sur le chemin parfois long et tortueux de la qualité informatique et, plus prosaïquement, des tests logiciels.

Que ce soit au travers de nos actions ou de nos conférences, en France ou à l'étranger, nous avons sans relâche porté le message, fidèles à notre mission, pour faire reconnaître notre profession (au sens large) et ses bonnes pratiques. Nous avons œuvré à soutenir les vocations et les initiatives destinées à promouvoir une informatique meilleure et plus sûre pour tous, que ce soit au travers de la reconnaissance de schémas de certifications, de l'accréditation de nombreux organismes de formation ou de notre action auprès des instances publiques et privées.

Aujourd'hui, alors que le paysage informatique – et plus largement, notre société tout entière – subit de profondes mutations, nous avons demandé à des experts reconnus de collaborer avec nous pour partager leur expérience de l'Agilité. Dans cet ouvrage, ils vous en parlent telle qu'ils la vivent sur le terrain ou telle qu'ils l'ont vécue, en tant que transformation, en tant que révolution culturelle.
Nous leur avons demandé de partager leurs réussites – pour que vous puissiez vous en inspirer si vous le jugez utile – mais aussi leurs échecs ou leurs galères – pour vous permettre de les anticiper, pour éviter de tomber dans les mêmes pièges.
Nous leur avons laissé le soin de vous narrer leurs histoires, leurs solutions, sans faux-semblant, sans fard ni censure. Loin des solutions dogmatiques et toutes faites, cet ouvrage se veut un recueil incomplet de l'état de l'art vu au travers du prisme de ces quelques professionnels qui, comme nous, œuvrent chaque jour à faire une informatique plus belle, plus qualitative, plus performante et plus sûre.
Consommez-le sans modération. Puissiez-vous y trouver quelques réponses ou même simplement matière à réflexion pour vous aider à construire votre propre cheminement, votre propre expérience vers cette transformationnumérique promise.
Ce livre est avant tout le leur…Il est maintenant le vôtre…Écoutons-les.

L'Agilité dans son contexte

Agilité : tendance forte ou mal nécessaire ?

L'Agilité est dans l'air du temps. Cette tendance, indéniable, est constamment confirmée, que ce soit par les enquêtes - comme le **World Quality Report** de Sogeti et Microfocus, l'enquête annuelle menée par **l'ISTQB®**, les enquêtes ponctuelles menées par le CFTL - ou plus simplement le travail de terrain.


C'EST LA MÉTHODE AGILE ! MAINTENANT, ILS ME FONT UN TEST À CHAQUE RELEASE D'UNE NOUVELLE LEÇON.
C'EST BEAUCOUP MIEUX QU'UN EXAMEN FINAL AU BOUT DE 10 ANS !
OUAIS...
C'EST JUSTE DOMMAGE QUE TU N'AIES QUE DES ZÉROS
PLAN DE TESTS ATDD
LEÇON 2.3.1 0/20
LEÇON 2.3.2 0/20
LEÇON 2.3.3 0/20
LEÇON 2.4.0 0/20
LEÇON 2.4.1 0/20
4.2
0/20
LEÇON 3.0
0/20
F. Cointe

Plus personne aujourd'hui n'oserait prétendre ignorer le phénomène, ne fût-ce que par pure convenance, car aujourd'hui **ne pas être Agile équivaut souvent à être un « has been »**, un laissé-pour-compte sur le chemin des TIC (Technologies de l'Information et de la Communication).

Pourtant, **l'Agilité n'est pas neuve** : son manifeste, écrit par quelques passionnés rebelles en marge d'un séjour aux sports d'hiver, inspiré de travaux antérieurs comme ceux de B. Boehm (RAD) ou de Beck, Cunningham et Jeffries (XP), date déjà de 2001 et son site, à peine encore maintenu a déjà pris plus de rides que nos vieux TRS80.

Cette véritable révolution de nos pratiques a donc mis des années avant de connaître l'engouement qu'elle rencontre aujourd'hui. Petit à petit, patiemment, elle a grignoté le terrain occupé par les méthodes « classiques », remplaçant peu à peu les modèles en cascades et en V, dictant ses nouveaux vocables et ses rituels pour finir par **s'imposer comme tendance forte** dans le paysage des TIC.

Il aura donc fallu plus de dix ans – quinze peut-être – pour que ces concepts, simples et pleins de bon sens au demeurant, commencent à s'imposer au plus grand nombre.

D'aucuns pourraient penser qu'il s'agit là du temps nécessaire pour acquérir l'indispensable maturité ; j'y verrais plutôt la combinaison d'un **changement sociétal profond,** accompagné d'un changement radical de cible.
Je m'explique…

À l'origine, l'Agilité s'érigeait en combattant des **pratiques quasi-dogmatiques,** irrationnelles et menant sans véritable discernement à un **inutile gaspillage des ressources vives** de nos entreprises (par la sur-documentation et le peu de transparence ou de visibilité sur les livrables notamment).
Le message, aussi pertinent fût-il, passait un peu à côté de sa cible, dans un monde qui sortait du passage à l'an 2000 (et plus près de nous à l'Euro) et qui se tenait prêt à relever de nouveaux défis – Internet pour tous, futurs hauts débits et services en mobilité notamment. L'innovation ne laissait plus vraiment de place à la méthode ou à la méthodologie. Il fallait avant tout tenir et assurer.

L'Agilité resta donc tapie dans l'ombre, anecdotique, réduite à la portion congrue dans les sociétés aux organisations encore fortement structurées et peu propices à des modes de fonctionnement moins lisibles (sinon plus flous) ou plus collaboratifs.

Avec l'évolution des « nouvelles technologies » qui se normalisèrent peu à peu, l'Agilité put enfin rencontrer son public : celui d'une **« société des écrans », hyper connectée, mobile, nomade,** plus collaborative et surtout **fondamentalement impatiente.**
Avec un lien au monde permanent et une disponibilité quasi illimitée de l'information, elle eut aussi le génie de monopoliser ce message marketing qui fait mouche et qu'on nous ressasse ad libitum :
« L'Agilité : c'est délivrer la valeur plus rapidement ! »

Sans doute est-il utile, à ce stade, de décortiquer quelque peu le propos :

La notion de valeur, aujourd'hui centrale dans nos économies pour le moins bousculées, se traduit dans le Manifeste Agile original selon 4 axes majeurs :

1. **Les individus et leurs interactions** plutôt que les processus et les outils.
2. **Des logiciels opérationnels** plutôt qu'une documentation exhaustive.
3. **La collaboration avec les clients** plutôt que la négociation contractuelle.
4. **L'adaptation au changement** plutôt que le suivi d'un plan.

Il est intéressant de constater que trois de ces axes concernent des **valeurs fondamentalement humaines et humanistes** (les interactions entre individus, la collaboration avec le client, le droit à l'erreur ou au changement), alors que l'acception généralement retenue dans notre société de consommation, éminemment pragmatique, voire quelquefois cynique, se limite le plus souvent au second axe (la **livraison opérationnelle** du produit ou du service).

La notion de **rapidité**, quant à elle, s'inscrit en filigrane des second et quatrième principes ; là où d'aucuns voudraient nous faire croire que **tout est possible, instantanément…et moins cher** de surcroît.

Tout le succès de l'approche réside dans ce credo : en délivrant vite, toujours plus vite, l'Agilité rencontre le besoin d'une société qui a muté, devenant de plus en plus impatiente, dans un contexte d'**hyper offre** ou d'**hyper choix**. Une société mondialisée qui vit dans l'instant, dans l'immédiateté, avec déjà un pied dans l'IA et le futur, sans cesse en quête de nouveauté et de changement. Une société prête à payer le prix de solutions moins durables – car évoluant sans cesse – moins abouties – car d'abord basées sur des produits minimums viables – mais aussi plus à l'écoute du Métier et du Client, plus proche des besoins et de leur constante évolution. L'Agilité, c'est un peu Darwin en accéléré, c'est l'évolution s'inscrivant dans l'air du temps !

L'Agilité, cette incomprise.

Pour ses principaux détracteurs, il n'y a aucun doute : c'était bien mieux avant !
L'Agilité n'a ni le monopole de la valeur, ni celui de la performance – au temps pour le discours marketing – au contraire, elle favoriserait l'indécision en donnant une fausse impression d'impunité au changement et l'absence de documentation qu'elle prônerait ferait courir un risque technologique sans précédent, a fortiori dans une informatique de plus en plus complexe, interopérable et interconnectée.

C'est aller un peu vite en besogne.

Certes, l'Agilité **incomprise** ou **mal implémentée** amène à de fâcheuses dérives. Certaines organisations confondent, en effet, la carte et le territoire, substituant de simples **rituels** mal compris à l'esprit de l'Agilité. Car il n'est pas nécessaire d'affubler des équipes « cross-fonctionnelles » de noms bizarres, comme « les créatures » ou « les gremlins » ; de calfeutrer les fenêtres pour éviter de voir s'envoler post-its et projets au gré d'un vent capricieux ; de se la raconter tous les matins, debout serrés les uns contre les autres ou encore de ne jamais dépasser les limites

temporelles dogmatiques du scrum idéal pour faire de l'Agilité. Ceux qui ne l'ont pas compris sont souvent les mêmes qui, forcés de faire bonne figure pour faire plaisir ou client, au management ou plus simplement pour s'inscrire dans la continuité de l'histoire, prennent le train en marche sans bien comprendre **les enjeux de la destination** ou encore **du voyage**.

Ce sont aussi ceux-là qui, jetant le bébé avec l'eau du bain, refusent de documenter, détricotent la qualité en laissant analystes métier et testeurs sur le bord de la route, tant « l'homo technologicus », stressé en permanence par **des échéances toujours plus serrées** et par ce besoin impérieux de la **performance à moindre coût**, cède à la facilité et dévoie la notion de **« plutôt que »** pour la traduire en un **« et non plus »** si tentant quand la montre se fait pressante.
Le risque, évident et bien réel, serait cette sorte d'autojustification de la machine **à remonter le temps**. Celle qui nous ramènerait, au prétexte de préceptes mal compris ou volontairement mal employés, à confondre **des temps révolus** où l'on tapait du code avant même de savoir à quoi il allait servir ; où les développeurs se passaient bien volontiers de leurs collègues de la qualité (comment peut-on être testeur ?) ; où les sirènes de l'automatisation totale et à tout crin menaient les projets au désastre. Ce sont bien là les pires ennemis de l'Agilité au point que nos collègues anglo-saxons ont donné un nom amusant à ces pratiques douteuses ou approximatives : **« Fragilité »** !

Si l'Agilité a parfaitement su placer son message pour le rendre attractif et contemporain, force est de constater qu'avec ses allures protéiformes qui se réinventent constamment, ses approches multiples (scrum, kanban...), ses produits dérivés (les fameux xDD, CI et autres « continuous everything », DevOps et j'en passe...), son appétence forte pour les solutions outillées (parfois éphémères, peu abouties voire carrément gadget) et ses rôles multi-casquettes (les limites entre Métier – Développeur – Analyste – Scrum Master ou Testeur, sans oublier évangélistes et gourous prosélytes autoproclamés en tout genre)… **l'Agilité manque parfois de clarté ou de lisibilité organisationnelle.**

C'est sans doute le prix à payer pour cette véritable révolution – le terme n'est pas trop fort – qui bouleverse nos pratiques durablement et de fond en comble.

Plus qu'un changement : une contre-culture qui – à son tour – devient mainstream

Parfois, on se lève un matin et on se rend compte que dehors, tout a changé.
Voilà, c'est fait, maintenant nous sommes Agiles ! découvre-t-on un jour en allant au bureau.

Fruit de notre profonde mutation sociétale, l'Agilité partage avec l'innovation ces caractéristiques qui appellent à un **total changement de paradigme** ; une **transition culturelle pour accompagner notre transformation numérique**.

L'Agilité ne s'improvise pas, elle ne se décrète pas, et si elle s'organise ce n'est que pour mieux s'adapter et changer à nouveau. L'Agilité se vit, au cœur de nos métiers et de nos DSI…ou elle n'est pas.

C'est bien là son principal challenge. Bien plus qu'une conduite du changement « classique »,

elle amène à une totale **refonte culturelle** qui embrasse le **changement en continu** et favorise la **collaboration** – non plus une collaboration structurée, hiérarchisée, mais bien une collaboration favorisant les facteurs humains, l'innovation et la créativité à tous les niveaux, l'empathie et les approches centrées sur le client ou l'expérience utilisateur.

L'Agilité repose sur une **communication ouverte et bienveillante, une gestion efficace des forces et des faiblesses** de chacun, un management de la complémentarité et des compétences humaines.

Elle remet au goût du jour la problématique des **exigences non fonctionnelles**, sans cesse passée à la trappe des méthodes dites « traditionnelles », au prétexte du manque de temps ou de moyens.

Elle revisite aussi au passage les notions de cycle de vie, de performance ou même d'**automatisation**.

Elle repense complètement le **modèle organisationnel** et les **compétences** professionnelles tantôt en mixant les rôles et les casquettes, tantôt en rapprochant métier et développeurs, tantôt en rapprochant développeurs et exploitation.

Elle réajuste enfin les idées reçues, osant les **livraisons en continu**, les **shifts à gauche** ou même **à droite**, des **tests pilotés par le modèle**, quand ce n'est pas l'**IA** qui s'en mêle…

Pas étonnant dès lors que certains finissent par y perdre leur Cobol.

« Le Testeur doit se réveiller »

Il ne fait plus du **test** – qu'on se le dise - mais de la **qualité**…et la qualité, c'est l'affaire de tous (que les Analystes Métier et autres Managers me jettent la première pierre).
Il se fait plus technique, automatise à tout crin et ne rechignerait même pas à lire du code dans le texte, il se fait aussi plus business que le Métier lui-même. Il y en a même pour dire que la MEP ou ses environnements n'ont plus de secrets pour lui…et ah oui, de temps à autre, il testerait encore…incognito !

Et puis surtout ma bonne dame, aujourd'hui il communique, il collabore !

D'un coup d'un seul, ces **rôles** qui ont mis des années à entrer dans les mœurs **ont évolué** et se sont modifiés au gré des tendances et parfois d'une certaine forme de Novlangue. Que l'on se rapproche du Métier ou de l'Utilisateur, du Développement ou de l'Exploitation, les slogans et les **« teams »** se noient dans des constellations d'**outils** qui s'affichent tous plus simples, plus performants ou plus **collaboratifs** les uns que les autres.

C'est bien là, passez-moi le jeu de mots, le talon d'Agile.

Pas plus qu'on enseignait le test ou la qualité aux informaticiens d'avant la transformation, on n'apprend pas aujourd'hui la collaboration ou la **communication** sur la plupart des bancs des écoles d'informatique. Et pourtant, tous sont censés **partager, co-créer** en équipe, sauter d'un profil à l'autre en toute **complémentarité multidisciplinaire**…un comble !

Comme si nos anciennes tours de Babel – au temps où les testeurs et les développeurs étaient encore clairement identifiés - ne nous avaient rien appris. Comme si « l'homo-informaticus » avait un sens inné de la collaboration ouverte et de la communication naturelle ! Comme si de telles choses pouvaient même exister, tout simplement...

Certes, il y a de la **richesse** à **repousser les limites**, à **dé-cliver les rôles** et à bousculer les barrières. Encore faut-il nous en donner les vrais moyens.
Si je n'ai que peu de craintes quant au futur de l'**automatisation** et à la capacité des qualiticiens à **l'intégrer dans leur approche** (après tout cette tendance, parfois plus proche du développement que du test lui-même est plus ou moins connue depuis des années), il n'en va pas de même dans d'autres domaines plus vitaux encore pour le bon fonctionnement de l'Agilité. Dans une approche mondialisée et sur-informatisée où tout un chacun devient un client réel et non plus potentiel, l'informatique par les informaticiens et pour les informaticiens a vécu et les teams agiles doivent impérativement se doter des outils nécessaires pour se comprendre et se faire comprendre : j'ai nommé les « **soft skills** ».

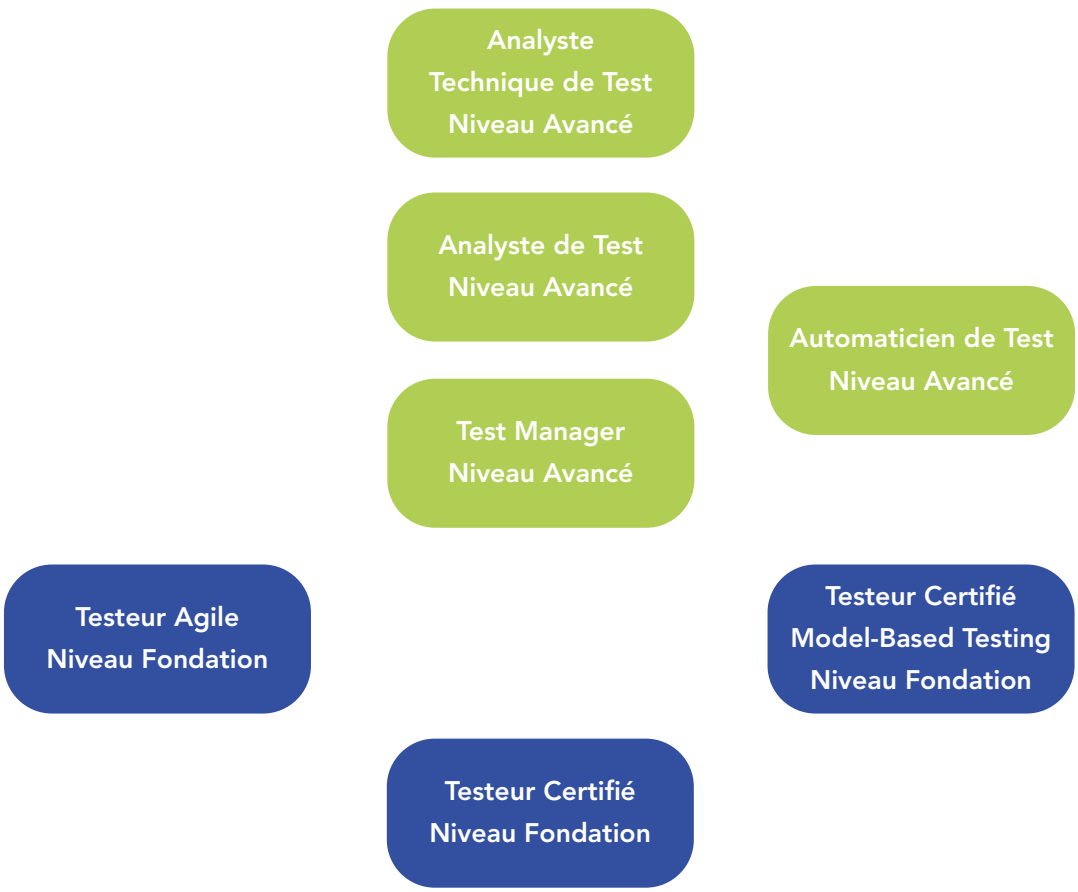
Et voilà donc que les ingénieurs et autres programmeurs doivent radicalement changer d'attitude, se muer en **communicants**, apprendre les codes de l'**empathie**, dialoguer avec l'**utilisateur**, partager avec lui son **expérience (Ux)**.
Voilà qu'ils doivent se remettre en question, en permanence, lors des phases d'**exploration** ou de **conception** des **produits et services**.
Voilà qu'ils se doivent de **convaincre**, de porter la bonne **parole,** pour mieux vendre le résultat de leurs cogitations, en interne et dans l'équipe tout d'abord, puis à l'extérieur et aux clients ensuite.
Voilà, enfin, qu'ils doivent intégrer dans des cycles courts, des compétences technico-fonctionnelles mais aussi humaines. Certains sont doués pour définir la **vision** ou les **concepts**, d'autres pour **mener** les **collaborateurs** et le projet à bien, d'autres **challengent, polissent** et **complètent**, d'autres encore **mettent de l'huile dans les rouages**…et tous sont nécessaires pour arriver à bon port.

S'il est un fait que **le testeur a – et aura longtemps encore – un rôle à jouer dans le paysage de l'informatique agile**, que ce soit en se rapprochant du code, du geste métier ou de l'exploitation, il me paraît indispensable que celui-ci s'accompagne d'un réel **soutien scolaire et professionnel** en matière d'**Ux** ou de **soft skills**.
Sans cela, l'approche tout entière, privée de ses indispensables fondements collaboratifs et humains, deviendrait caduque, perdant d'un seul coup, à mes yeux du moins, tout son intérêt, tout son bénéfice. Autant laisser aux machines le soin de s'autoprogrammer…mais cela est une autre histoire qu'il nous reste à écrire…ou non !

Présentation du CFTL et de l'ISTQB

par Olivier Denoo, Bruno Legeard et Eric Riou du Cosquer

Certifications CFTL/ISTQB disponibles en France en avril 2019



Depuis 2008, le CFTL organise la Journée Française des Tests Logiciels qui accueille chaque année 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 est un élément clé 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 spécialisés (Agile, Model-Based Testing) et les niveaux avancés (Test Manager, Analyste de Test, Analyste Technique de Test, Automaticien 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
- Mi-2018, le monde comptait plus de 605 000 professionnels certifiés par l'ISTQB
- Une croissance de 14% pour la certification Testeur Agile
- Un partenariat fort avec le schéma IREB – International Requirements Engineering Board – sur l'ingénierie des exigences.

En 2018, l'ISTQB a publié une mise à jour de la certification Niveau Fondation et du glossaire des termes des tests logiciels en phase avec l'évolution des pratiques, y compris celles issues de l'Agilité.

Le CFTL – Comité Français des Tests Logiciels – est le représentant officiel de l'ISTQB en France, organisant 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 850 000 examens cumulés passés dans le monde en 2018, les certifications ISTQB comptent parmi les plus diffusées des technologies de l'information. Le Million d'examens passés est attendu pour 2020 !

Pourquoi ce livre ?

par Olivier Denoo, Marc Hage Chahine, Bruno Legeard et Eric Riou du Cosquer

Le passage en Agile transforme profondément nos pratiques des tests logiciels sans que la recette d'une transformation réussie soit connue !

En effet, les principes du développement logiciel agile, tels que XP – eXtreme Programming – par exemple, ont été d'abord déployés dans le contexte de petites équipes, souvent chez des éditeurs et réunissant certaines bonnes propriétés telles la colocalisation de tous les membres de l'équipe, une implication forte du Product Owner sur site et un engagement sans faille des développeurs dans la qualité du code (avec mise en pratique des revues de code et une forte couverture des tests unitaires).

Or, depuis quelques années, le passage à l'Agilité est réalisé dans des contextes très variés, en particulier au sein des grandes organisations et des grands projets où les situations organisationnelles sont complexes, les patrimoines applicatifs importants et l'imbrication des applications et sous-systèmes très forts. Ce n'est plus l'Agilité des « early adopters » mais celle du « mainstream » !

Ce livre répond au besoin de partage des connaissances de la communauté francophone des tests logiciels. Comment l'Agilité s'intègre et modifie nos pratiques ? Quelles sont les nouvelles approches de tests, les techniques et l'organisation des activités les plus pertinentes ? Quels sont les retours d'expérience ? 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 – Aspects spécifiques des tests en Agile – nous introduisons les aspects spécifiques des tests dans l'Agilité en termes d'état d'esprit, de rôle des testeurs et d'organisation des équipes.

En Partie II – Pratiques des tests en Agile – nous offrons un catalogue de pratiques, couvrant à la fois les techniques de test (ATDD/BDD, Automatisation, Tests exploratoires, …), la façon d'organiser les tests (Agilité à l'échelle – SAFe, métriques, tests basés sur les risques, …) mais aussi les contextes spécifiques tels l'IoT ou le test en production.

En Partie III – Retours d'Expérience – nous présentons des analyses pratiques des réussites et des échecs de terrain.

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 Testeur Agile Niveau Fondation. Au contraire ! Avoir les connaissances de base d'un Testeur Agile et être certifié CFTL/ISTQB Niveau Fondation serait plutôt un prérequis pour une bonne compréhension de tous les aspects techniques et méthodologiques présentés.
• C'est le premier livre du CFTL, qui s'inscrit dans sa vocation essentielle d'être au service de la communauté des tests logiciels, en créant des forums d'échange sur l'évolution de notre profession.

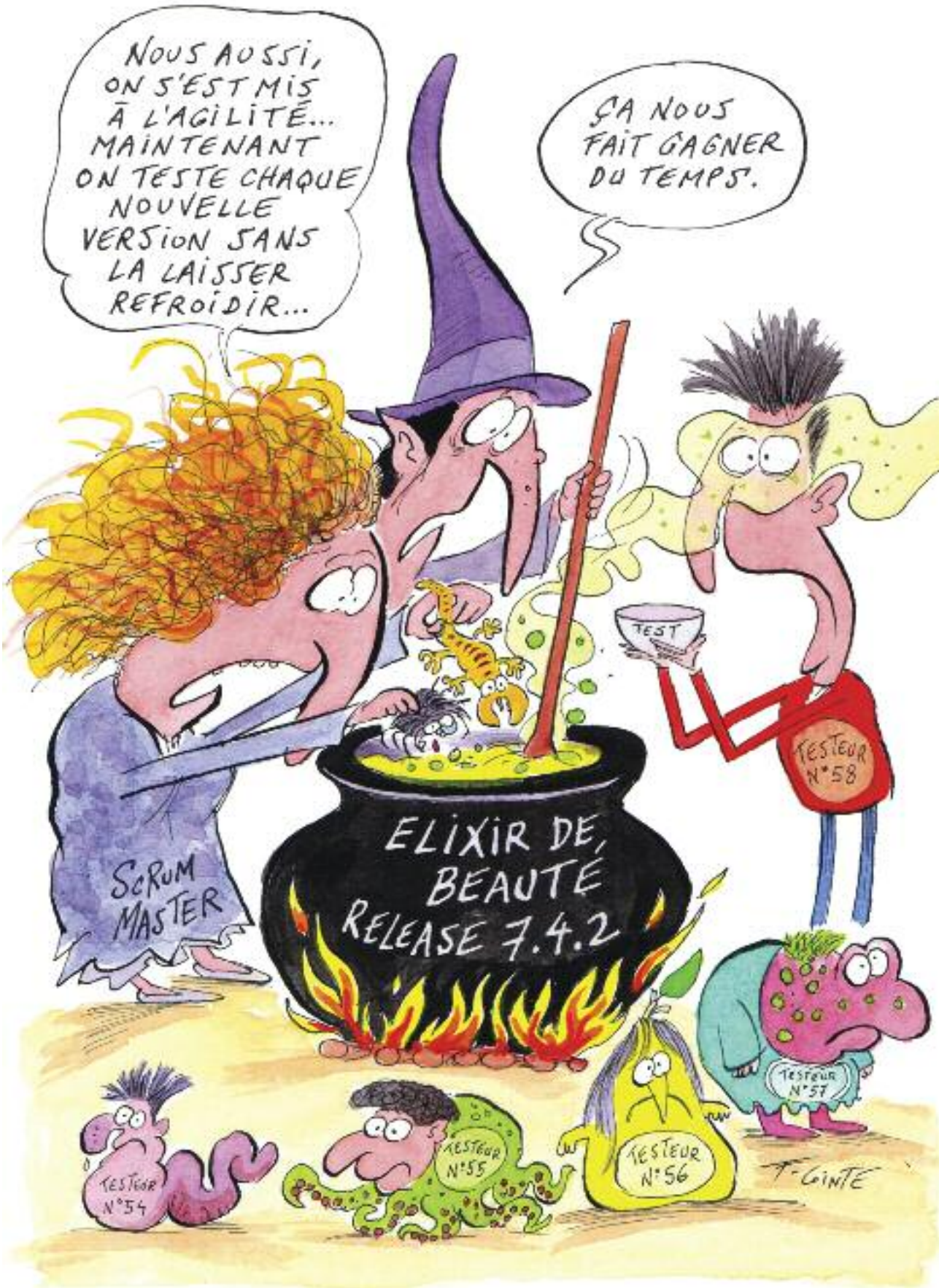
Le saviez-vous ?

Dans l'enquête CFTL 2019, la moitié des personnes se positionnant sur un profil « Testeur » ont indiqué « Testeur Agile ».

Le comité éditorial «Les Tests en Agile» vous souhaite une bonne lecture.

PARTIE I

ASPECTS SPÉCIFIQUES DES TESTS EN AGILE


NOUS AUSSI, ON S'EST MIS À L'AGILITÉ... MAINTENANT ON TESTE CHAQUE NOUVELLE VERSION SANS LA LAISSER REFROIDIR...
ÇA NOUS FAIT GAGNER DU TEMPS.
SCRUM MASTER
ELIXIR DE BEAUTÉ RELEASE 7.4.2
TEST
TESTEUR N°58
TESTEUR N°54
TESTEUR N°55
TESTEUR N°56
TESTEUR N°57
F-COINTE

I.1- Comment l'Agilité fait évoluer les pratiques des tests logiciels - Données chiffrées

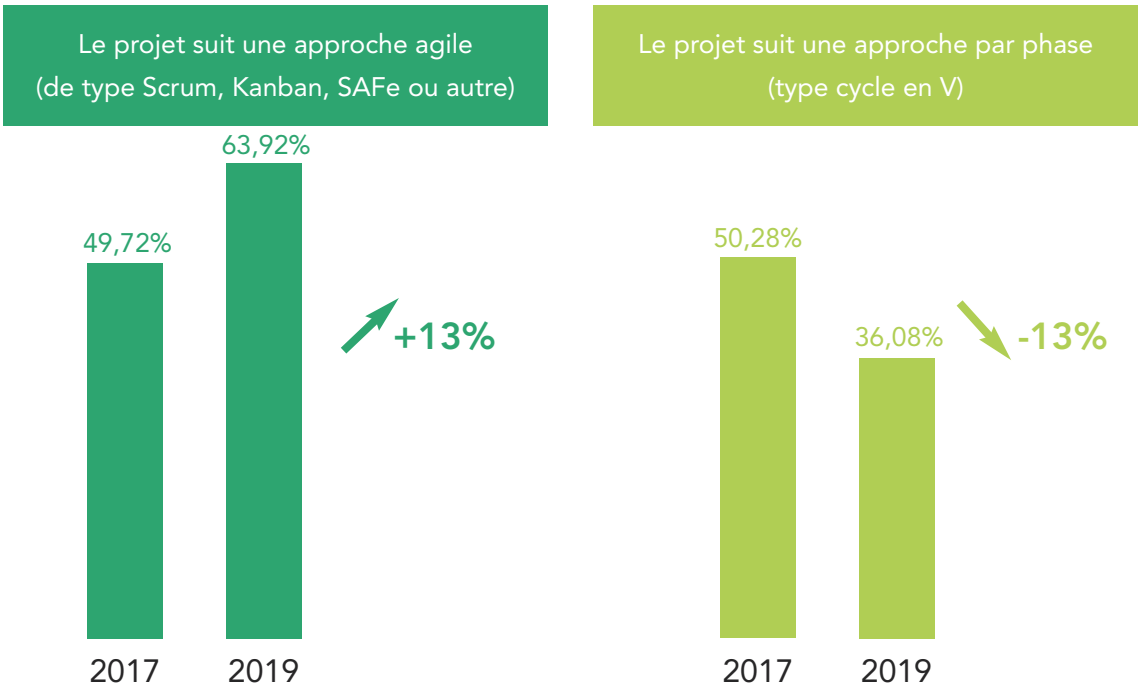
par Bruno Legeard et Alexis Todoskoff

Du 1er décembre 2018 au 20 janvier 2019, le CFTL a réalisé la 3ème enquête de son observatoire des pratiques des tests logiciels, après celles de 2017 et 2013. Au travers d'un questionnaire très complet, la communauté française du test logiciel fournit des données précises sur l'état des pratiques et, lorsque l'on compare les résultats 2019 avec ceux de 2017 et 2013, on obtient une vision claire des tendances en cours. A chacune de ces enquêtes, ce sont plus de 700 personnes de l'ensemble des domaines économiques et des différents métiers du test logiciel qui ont répondu. Les résultats complets de ces enquêtes sont disponibles sur le site du CFTL.

Dans ce chapitre, nous focalisons notre analyse sur la transformation des pratiques des tests logiciels avec l'arrivée de l'Agilité au travers des réponses à l'enquête CFTL 2019 et des résultats 2017 et 2013, lorsque cela est pertinent.

1- Progression de l'Agilité.

Les résultats de l'enquête CFTL 2019 montrent une adoption de l'Agilité dans 64% des réponses, indiquant une forte progression par rapport à 2017 qui avait donné 49% de contexte agile. Lors de l'enquête CFTL de 2013, nous n'avions pas posé cette question.

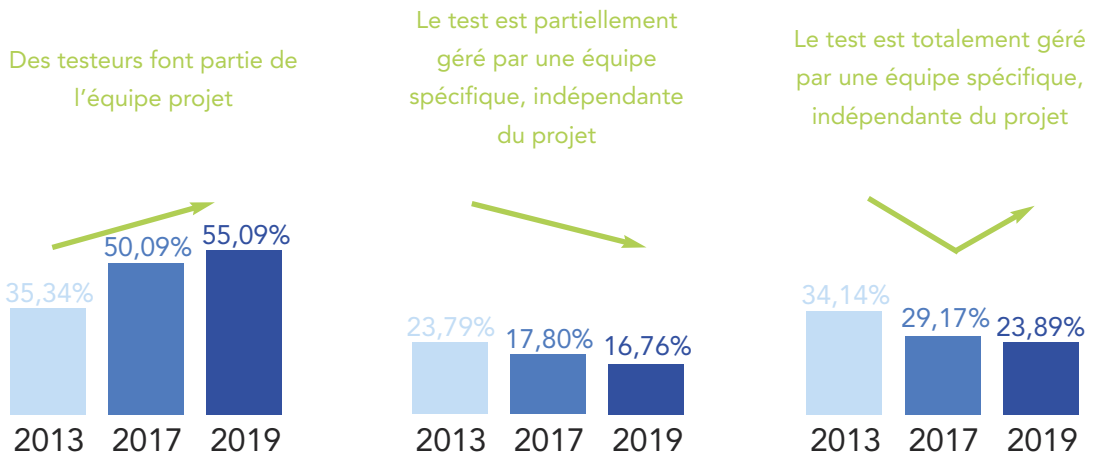


Ce résultat montre que l'Agilité est devenue « mainstream » : c'est maintenant la pratique la plus courante en développement logiciel.

2- Organisation des tests : des testeurs plus fréquemment intégrés dans l'équipe projet.

Les résultats de l'**enquête CFTL 2019** et la progression depuis 2013 montrent que les testeurs sont de plus en plus intégrés dans les équipes Projet, à 54,30% en 2019 contre 35,34% en 2013.

Question : Quelle situation correspond le mieux à l'organisation des activités de test sur les projets sur lesquels vous intervenez ?



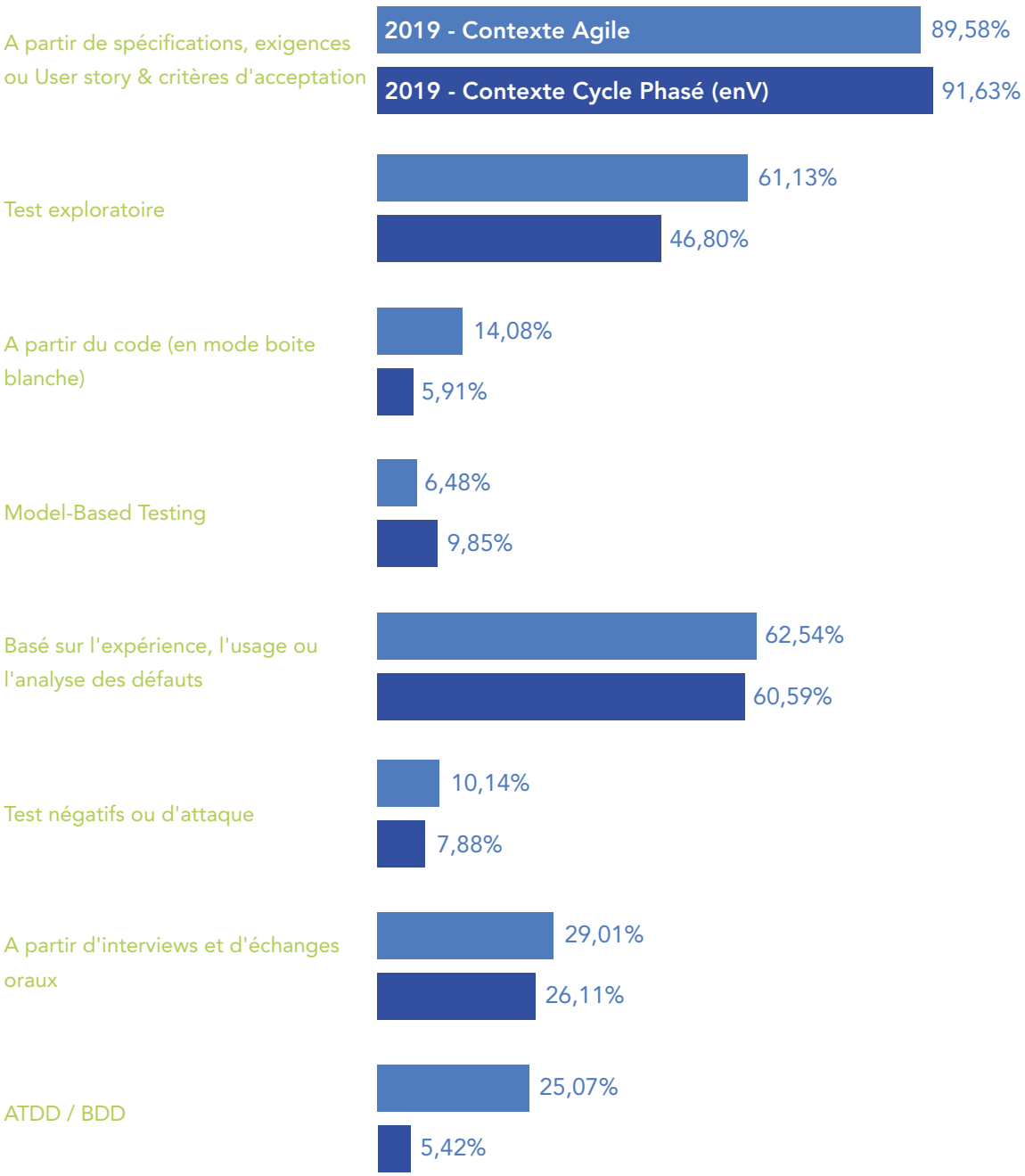
Lorsque l'on filtre les réponses de l'enquête CFTL 2019 à cette même question, en fonction du type de processus de développement mis en place, on constate que l'Agilité conduit à intégrer plus fréquemment les testeurs dans l'équipe projet par rapport au contexte de développement type cycle en V. Ce n'est pas une surprise car un des piliers des pratiques agiles (cf. chapitre I.3) est l'équipe intégrée, composée du représentant du client, des développeurs et des testeurs.

Question : Quelle situation correspond le mieux à l'organisation des activités de test sur les projets sur lesquels vous intervenez ?

3- Conception des tests : l'empreinte des pratiques issues de l'Agilité.

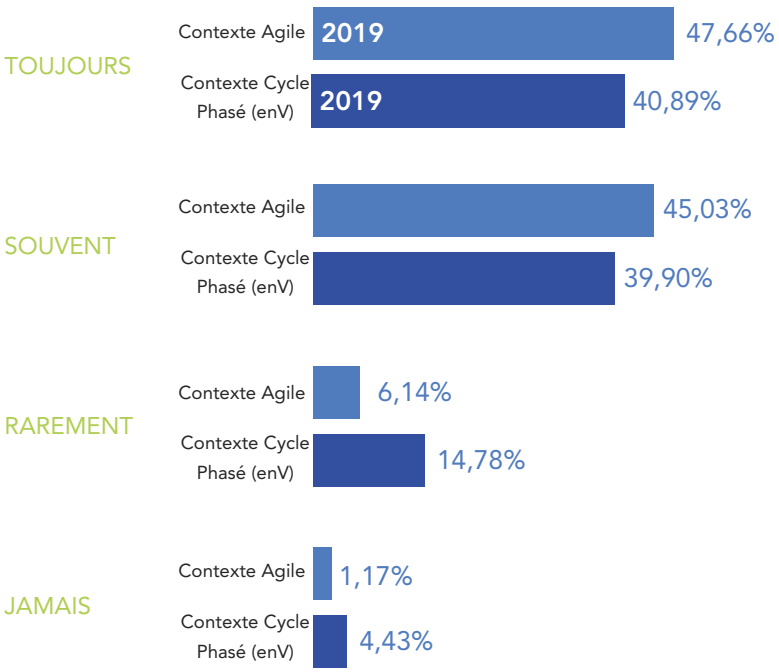
Le contexte agile influence directement les techniques de conception de tests mises en œuvre, comme le montre le tableau suivant, en particulier pour les tests exploratoires et l'utilisation des pratiques ATDD/BDD, plus spécifiques de l'Agilité. En d'autres termes, l'Agilité conduit à une évolution des techniques de test, à la fois en faisant apparaître de nouvelles pratiques (telles que l'ATDD/BDD par exemple) et en renforçant d'autres pratiques préexistantes (telles que les tests exploratoires par exemple).

Question : Quel(s) type(s) de techniques de conception de tests utilisez-vous ? (plusieurs réponses possibles)



De façon un peu paradoxale, l'utilisation d'une expression des besoins formalisée pour concevoir les tests est plus forte dans le contexte agile que dans le contexte par phases. Cela peut être dû au fait que, dans le contexte par phases, les documents de spécifications n'ont pas été maintenus et ne peuvent donc pas être utilisés comme bases des tests. Peut-être aussi que la maturité globale moyenne est plus forte parmi les répondants à l'enquête CFTL 2019 en contexte agile.

Question : Sur les projets sur lesquels vous intervenez, est-ce que des exigences, des User Story ou des spécifications formalisées sont utilisées pour concevoir les tests ?



Dans le **contexte agile**, dans plus de **92%** des cas, à la question **« Disposez-vous d'une solution outillée de gestion des exigences ou du Backlog du produit ? »** la réponse est **OUI**, alors que ce n'est que de **68% dans les projets par phases** (type cycle en V).

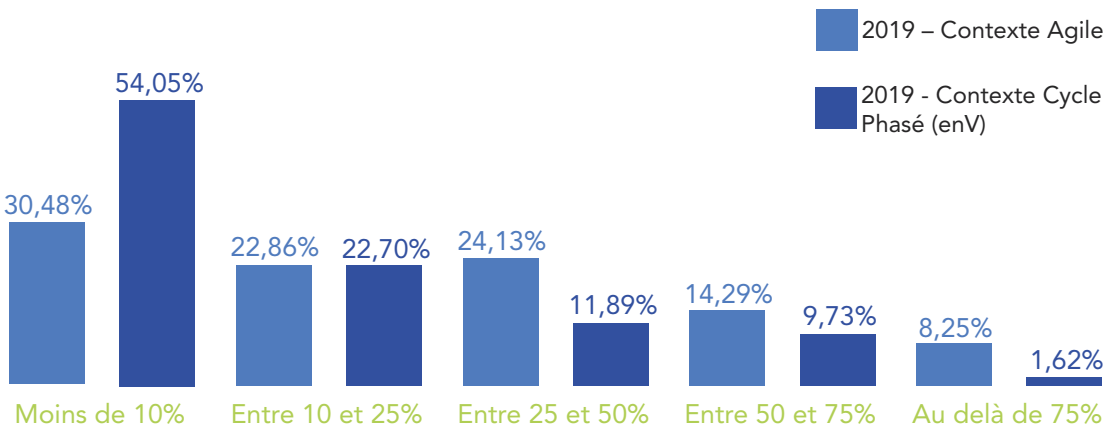
Cette différence est liée à l'utilisation des User Story comme outils de pilotage des développements itératifs. La gestion des itérations avec un tableau de type Kanban est une pratique très courante des projets en Agile et les outils de gestion du projet et du backlog du Produit en Agile sont très fréquemment déployés (voir par l'exemple l'enquête « Stage of Agile » dont nous présentons les résultats dans le chapitre suivant).

Ainsi, l'Agilité conduit à une plus grande rigueur dans le suivi de l'expression du besoin (les User Story en Agile) au travers des itérations, facilitant le suivi de la couverture des User Story par les tests.

4- Une automatisation plus forte des tests en Agile et liée à l'intégration continue.

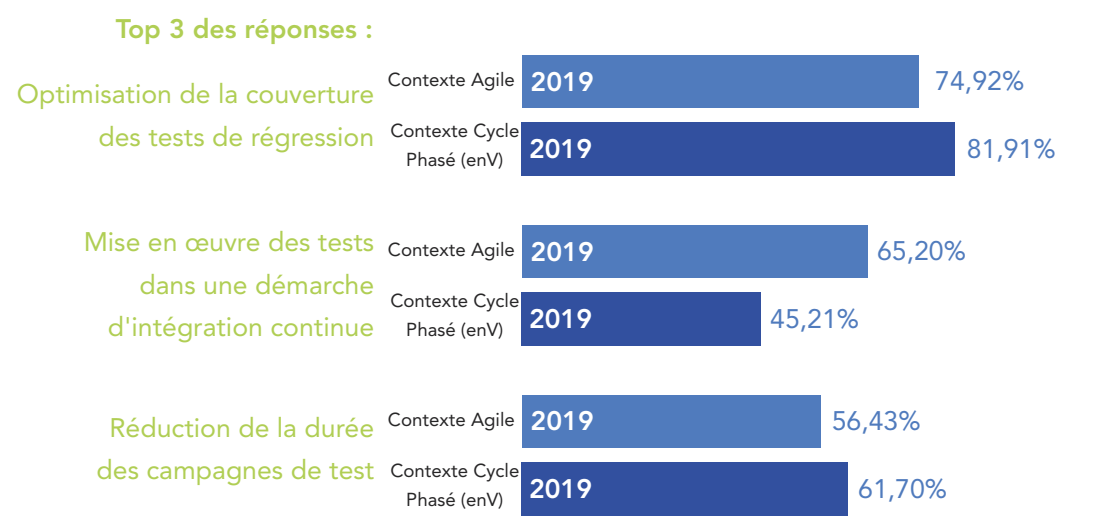
De façon attendue et clairement corroborée par les résultats de l'enquête CFTL 2019, le contexte agile induit une plus forte automatisation des tests. En effet, dans 46% des cas en Agile, on constate plus de 25% d'automatisation du patrimoine de tests, alors que ceci concerne seulement 23% des cas dans les contextes de développement par phases (type cycle en V).

Question : Quelle estimation faites-vous du pourcentage d'automatisation du patrimoine de test ?



Les motivations indiquées de cette automatisation sont aussi légèrement différentes suivant le contexte.

Question : Quelles sont vos trois principales motivations pour entamer une démarche d'automatisation de l'exécution des tests ?



On voit ainsi qu'en contexte agile, la mise en œuvre des tests en intégration continue constitue une motivation forte de leur automatisation.

De plus, à la question « L'automatisation des tests est-elle mise en œuvre dans un contexte d'intégration continue ? », plus de 75% des répondants en contexte agile indiquent que c'est le cas au moins partiellement, alors que le pourcentage est de 52% dans le contexte par phases. La mise en œuvre de l'intégration continue et l'automatisation des tests dans ce cadre constituent un marqueur clair de l'évolution des pratiques vers l'Agilité : ce qui n'est pas une surprise mais est corroboré par l'enquête CFTL 2019.

5- Les facteurs de succès des activités de test agile : la communication et l'équipe intégrée.

A la question **« Quels sont pour vous les facteurs de succès des activités de test ? »**, les répondants de l'enquête CFTL 2019 ayant indiqué un contexte de projet en Agile répondent à près de **49% : la communication et l'équipe intégrée**. Les facteurs de succès : « Intégration des testeurs dans l'équipe projet », «Inclure les testeurs dans les phases amont », « Proximité des testeurs avec les développeurs, les concepteurs et le Product Owner » sont mentionnés de façon fréquente.
Ce résultat semble logique car la collaboration étroite entre le Product Owner, les concepteurs, les développeurs et les testeurs constitue un des fondements de l'Agilité (cf. les valeurs « Les individus et les interactions » et « La coopération avec le client » du Manifeste Agile). Sans communication forte et permanente, favorisée par l'équipe intégrée, l'Agilité ne peut exister.
Les questions de compétences et de formation, à la fois au niveau des testeurs et au niveau de l'équipe, remonte comme un facteur de succès dans 46% des cas.

D'autres facteurs de succès des activités de test en Agile sont également évoqués de façon récurrente :
– l'outillage (« Plateforme technique adaptée et évolutive », « Robustesse des outils »),
– la rigueur,
– la méthodologie (« Méthode et industrialisation », « Process de test »),
– l'automatisation des tests,
– l'adhésion au bénéfice du test (« Toutes les équipes doivent être conscientes des enjeux et bénéfices du test », « Avoir une vision de l'importance des tests et de la qualité partagée par les équipes produit »),
– la reconnaissance du métier (« Soutien de la hiérarchie », « Implication de la direction avec sponsoring », « Soutien du management pour les activités de test »),
– l'organisation (« Un sens de l'organisation strict », « Une organisation cohérente avec les besoin des tests »),
– la formation (« Formation des testeurs », « Formation continue », « Compétence technique », « Expérience du testeur »).

On retrouve dans ces thématiques, de façon logique, les principaux ingrédients de l'Agilité que l'on pourrait résumer en : Outils – Processus – Individus – Soutien managérial. A cela, il faut ajouter l'Automatisation des tests, qui est incontournable en contexte agile, pour permettre de respecter la démarche itérative courte. Il faut noter aussi que « la reconnaissance du métier » constitue toujours un facteur important de succès. Ce « combat » ne semble pas encore avoir été totalement gagné. Tous ces facteurs de succès décrivent ainsi les principales difficultés rencontrées dans les pratiques des tests en Agile. En effet, sans soutien managérial et technique mais aussi sans réelle organisation, méthodologie, rigueur et compétence des équipes, il est très difficile de fonctionner efficacement en mode agile.

Nous terminerons ce chapitre par des réponses à la question **« Quels sont les axes d'amélioration des pratiques du test importants pour l'avenir ? »** extraites de l'enquête CFTL 2019 :

- « Communication avec toutes les équipes et intégration au plus tôt des équipes de test. »
- « La pratique des activités de test doit tenir compte de plus en plus des tests des systèmes interconnectés avec des architectures hybrides et les meilleurs choix de l'environnement de test pour les systèmes intégrant les big data et l'IoT. »
- « Accélérer le changement de culture déjà amorcé, pour la professionnalisation d'un plus grand nombre de personnes aux métiers du test, notamment grâce à un rapprochement avec les équipes de développement et de meilleures compétences en automatisation. »
- « Mise en place plus générale de l'ATDD » / « Automatisation des tests avec mise en œuvre d'une démarche BDD »

Ces réponses montrent des défis de la transformation en cours des pratiques des tests logiciels dans le passage à l'Agilité, à la fois sur le plan de l'organisation des tests, de l'intégration des testeurs au sein de l'équipe, des approches de test utilisées et des questions liées aux compétences.


LES PLANÈTES IL Y A DEUX JOURS. LES ANIMAUX AVANT-HIER. L'HOMME HIER. ALORS? QU'EST CE QU'ON VA CRÉER AUJOURD'HUI?
LA FEMME!
PLANIFICATION DES SPRINTS
A. COINTE

I.2- Motivations, bénéfices et pratiques de l'Agilité

par Bruno Legeard

Dans le chapitre précédent, nous avons vu de quelle façon, à partir des résultats des enquêtes CFTL, l'Agilité transformait les pratiques des tests logiciels. Mais, avant de poursuivre sur ces pratiques des tests en Agile, nous allons dresser un tableau de l'Agilité telle que pratiquée en 2018, des motivations pour passer à l'Agilité, des bénéfices obtenus et des impacts concrets sur les équipes projets.

Sauf mention contraire, l'ensemble des chiffres de ce chapitre sont issus de la 12ème enquête « State of Agile™ » réalisée en 2018 par la société VersionOne et accessible en ligne.

1- Motivations du passage en Agile.

La première motivation du passage en Agile est **l'accélération des mises en production** qui apparaît dans 75% des réponses, assez loin devant les autres motivations (cf. Figure 1). Les organisations passent à l'Agile d'abord pour répondre à des enjeux de « time-to-market », c'est-à-dire pour permettre de mettre en production de nouvelles fonctionnalités ou des améliorations applicatives plus fréquemment. Ainsi, dans le contexte des grands Systèmes d'Information, cela veut dire passer de 4 releases par an à 10 mises en production et dans le contexte d'applications digitales (plateforme web, applications mobiles, IoT, …), cela peut aller jusqu'à des pratiques de déploiement en continu des mises à jour logicielles.

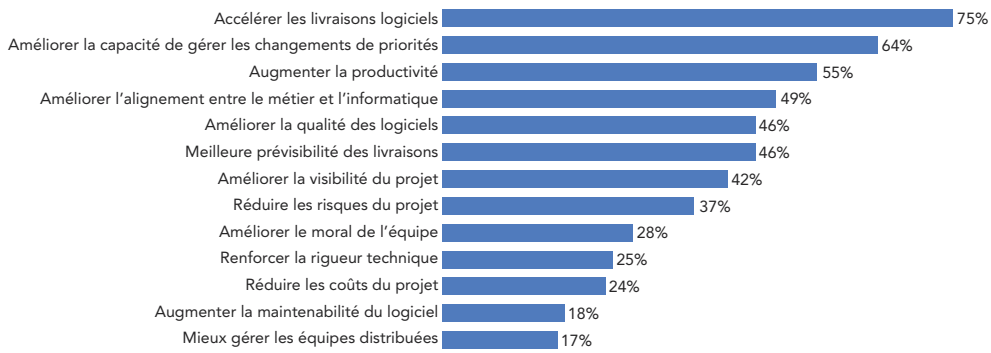


Figure 1 - Motivation du passage en Agile – Source « State of Agile – 12ème édition »

Cette accélération des mises en production a une incidence directe sur les tests : des incréments fonctionnels plus petits mais une fréquence des tests s'accélérant, ce qui mécaniquement va impliquer des besoins d'automatisation croissant des tests de régression (car tester en régression 4 fois par an n'est pas la même chose que tester 10 fois !).

La 2ème motivation exprimée du passage à l'Agilité est la **capacité à mieux gérer des changements dans les priorités** : répondre aux enjeux de marché pour les organisations passe par une capacité d'adaptation qui doit s'inscrire dans les systèmes logiciels. Avec des incréments plus petits et des mises en production plus fréquentes, le développement logiciel s'adapte aux évolutions des besoins du Métier. Les nouvelles pratiques des tests, telles que l'ATDD – Acceptance Test Driven

Developement – joue alors un rôle clé pour clarifier l'expression des besoins dans un triptyque : User Story, Critères d'acceptation et Scénarios d'acceptation.

La 3ème motivation est **d'augmenter la productivité** – hé oui ! – et cela touche les activités de test. Augmenter la productivité des activités de test, cela veut dire passer d'approches très manuelles à des techniques outillées, visant l'élimination des tâches inutiles et focalisées sur le résultat : mettre en production du logiciel de qualité satisfaisant les Métiers ! D'où la mise en œuvre de chaînes de production de tests automatisées, mais aussi des approches qui font intervenir le test le plus en amont possible (comme l'ATDD déjà cité) et qui rejoignent les deux motivations suivantes :
Améliorer l'alignement entre le métier et l'informatique et **Améliorer la qualité des logiciels**. Ces deux motivations sont liées : il faut renforcer la satisfaction des utilisateurs et des clients car le système informatique est de plus en plus au cœur du Métier des organisations. Les tests logiciels, incluant de façon affirmée les tests fonctionnels tels l'utilisabilité, les performances et la sécurité, jouent un rôle central dans cette prise en compte de la qualité des mises en production.

2- Est-ce que le passage en Agile apporte les bénéfices souhaités ?

La figure suivante montre les réponses en termes de bénéfices obtenus, tels qu'ils sont déclarés par les personnes ayant répondu à l'enquête « State of Agile ». Le passage à l'Agilité permet ainsi de répondre aux besoins de **l'accélération des mises en production** (réponse classée n°4) à celle de la **gestion de changements de priorités** (réponse classée n°1). Ce n'est pas surprenant : avec une approche itérative et incrémentale, la capacité de réponse aux besoins d'adaptation est plus forte et les fréquences de mises en production plus rapides : c'est la « raison d'être » de l'Agilité ! Mais l'Agilité favorise aussi un **meilleur alignement entre le Métier et l'informatique** (3ème bénéfice constaté), certainement du fait de la mise en place d'interactions fortes et régulières sur la gestion du Backlog du produit, et du fait de la relation entre le Product Owner et l'équipe Projet.

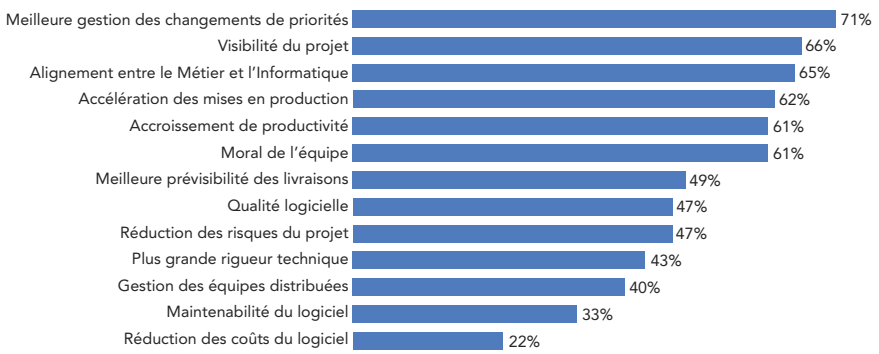


Figure 2 - Succès du passage à l'Agilité - Source « State of Agile - 12ème édition »

Deux autres bénéfices de l'Agilité sont restitués en tête de liste dans cette enquête, et qui vont dans le même sens : une équipe avec un meilleur moral et plus productive. Le passage à l'Agilité renforce la motivation de l'équipe de développement et une meilleure productivité, car mettre

en production régulièrement du logiciel qui répond aux besoins est effectivement une source de satisfaction professionnelle.

Est-ce à dire que nous serions au pays des Bisounours ? Non, les réponses précédentes sont des bénéfices constatés, mais rien n'est dit des efforts et de la gestion du changement nécessaires pour obtenir ces résultats. Ainsi, dans cette même enquête « State of Agile », 84% des répondants indiquent que leur organisation se situe dans une phase de maturation des pratiques de l'Agilité. Ce qui correspond à l'esprit de l'Agilité : adopter des techniques d'améliorations continues fondées sur une analyse régulière et rétrospective de ce qui fonctionne, ce qui doit être amélioré, et mettre en place des actions d'amélioration.

3- Les pratiques de l'Agilité les plus populaires.

Une fois comprises les motivations (et les résultats) du passage à l'Agilité, il est intéressant de percevoir les pratiques les plus mises en place dans le catalogue des pratiques de l'Agilité.

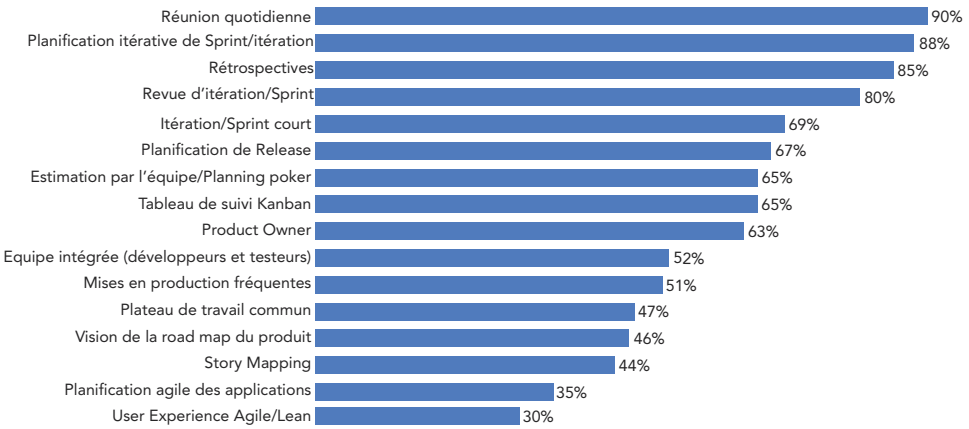


Figure 3 - Les pratiques les plus populaires - Source Source « State of Agile - 12ème édition »

Le Top 5 est sans vraie surprise (cf. Figure 3) :

• Les **réunions quotidiennes** de l'équipe (appelé Standup ou Scrum meeting[1]) debout et sur un temps court – chaque participant exprimant ce qu'il a fait, ce qu'il est en train de faire et d'éventuels points bloquant et ralentissant. La réunion quotidienne est le socle de l'Agilité au quotidien.

• **La planification itérative des sprints**, qui impliquent l'ensemble de l'équipe, testeur compris, et bien sûr le Product Owner. Comprendre le besoin, évaluer son coût de développement et de test, discuter des critères d'acceptation sont les piliers de l'organisation du travail en Agile sur des itérations courtes.

• **Les rétrospectives** permettent à l'équipe de discuter et d'améliorer les pratiques. Il s'agit d'être acteur de l'amélioration continue des pratiques tant en développement, analyse du besoin que pour les tests.

[1]Les pratiques Agiles sont définies dans un jargon issu des différents cadres (XP, SCRUM, Kanban, Dev/ops…) et techniques (Intégration continue, Déploiement continu, TDD, ATDD/BDD, …) qu'il faut maîtriser et qui est présenté en synthèse dans le chapitre suivant

• Les **revues d'itération / de sprints** permettent de démontrer la valeur qui a été produite durant l'itération et d'obtenir du feedback. Tout est là : dans le feedback rapide obtenu par l'équipe pour permettre d'obtenir l'alignement Métier / Informatique.
• Les **itérations courtes** (de 2 à 4 semaines en général) permettent de cadencer la production (et les tests !).

Ces pratiques s'inscrivent dans un cadre méthodologique qui est aujourd'hui dominé par le cadre Scrum, mais parfois combiné avec d'autres inspirations telles que Kanban et XP. Le cadre Scrum a imposé une terminologie commune :

• Les rôles de Product Owner, Scrum Master, Développeur et Testeur,
• Les rituels sur les Scrum meeting (la réunion quotidienne), les revues de Sprints (le nom donné à l'itération en Scrum), les rétrospectives et les planifications de Sprints et de Releases
• Le Backlog du Produit (qui permet de gérer l'expression du besoin) et le Backlog du Sprint (qui permet de gérer la liste des tâches pour un Sprint donné)

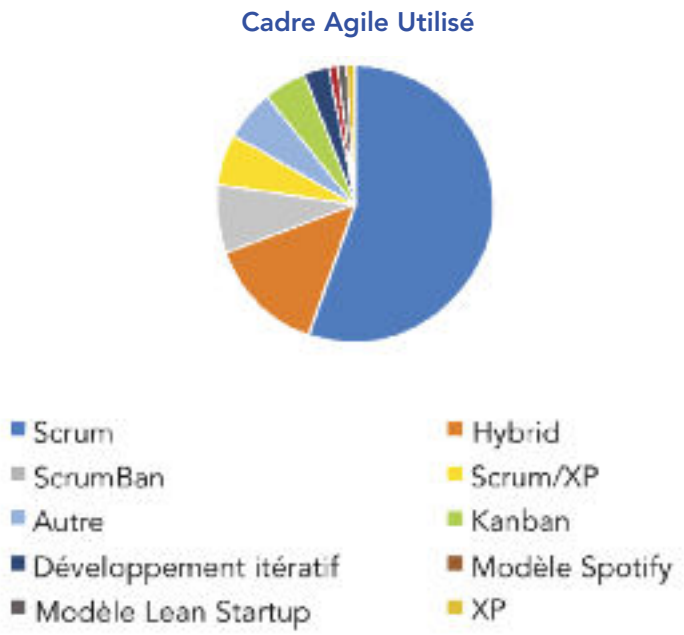


Figure 4 - Les cadres agiles les plus utilisés - Source « State of Agile - 12ème édition »

En Agile, le cadre est souple car la priorité est donnée à l'adaptation au contexte et à l'amélioration continue. L'esprit de l'Agilité prime sur le formalisme du cadre méthodologique ! Et de fait, ce caractère malléable de l'Agilité est aussi ce qui en fait le succès. Par exemple, 45% des répondants à l'enquête « State of Agile » indiquent utiliser l'Agilité dans un contexte de projet sous-traité à une organisation tierce. Quelle que soit la nature des développements (digital, systèmes d'information, et mêmes applications embarquées) et leur organisation (en interne ou sous-traitée), l'Agilité s'est répandue et continue à se répandre dans toutes les organisations, prenant des formes variées, mais avec les pratiques communes très diffusées vues précédemment.

Les pratiques d'ingénierie du logiciel constituent un autre aspect de l'impact de la diffusion des pratiques agiles. La figure 5 présente les techniques les plus mises en œuvre par les répondants à l'enquête « State of Agile ».

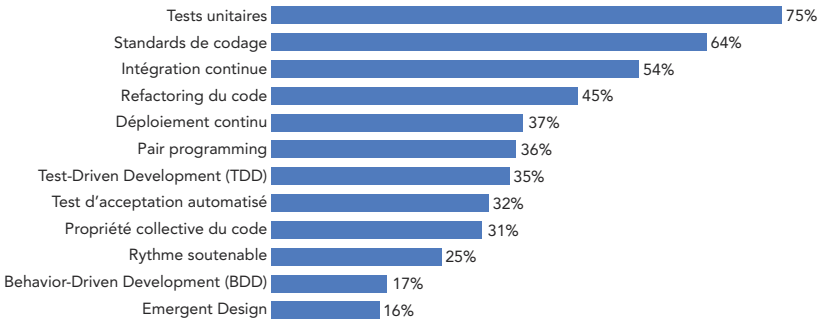


Figure 5 - Les pratiques d'ingénierie les plus populaires - Source « State of Agile - 12ème édition »

Le top 4 – Tests unitaires, Standards de codage, Intégration continue et Refactoring du code – constitue le socle des bonnes pratiques techniques en Agilité pour permettre de maîtriser l'accélération souhaitée des mises en production.

4- Et maintenant, ou va-t-on ?

Pour 26% des répondants à l'enquête « State of Agile », l'expérience dans l'Agilité est de 1 à 2 ans, et pour 35% de 3 à 5 ans. L'Agilité est encore jeune dans les organisations, les pratiques sont en maturation dans la plupart des cas et la période est à la consolidation pour en obtenir réellement les bénéfices.

L'évolution des pratiques des tests logiciels se situe au cœur de cette transformation car l'accélération des mises en production et l'augmentation des attentes en termes de qualité nécessitent de tester plus et plus souvent. C'est ce défi collectif auquel répondent les équipes, développeurs / Testeurs et Product Owner, dans les contextes variés qui fait la réalité des projets.

Les valeurs du Manifeste Agile (2001)

- Les Individus et leurs interactions plutôt que les processus et les outils.
- Des logiciels opérationnels plutôt qu'une documentation exhaustive.
- La collaboration avec les clients plutôt que la négociation contractuelle.
- L'adaptation au changement plutôt que le suivi d'un plan.

I.3- Agilité et tests en Agile - Manuel de survie dans la terminologie et les pratiques

par Bruno Legeard

L'idée de l'Agilité a émergé au début des années 2000, par opposition à des méthodes de développement du logiciel structurées par phases (spécification – conception – codage – test), fortement documentées à chaque phase et considérées comme lourdes et peu efficaces par les promoteurs de l'Agilité. Le Manifeste Agile définissant des valeurs de l'Agilité et des principes généraux associés a lancé cette approche avec le succès que l'on connaît aujourd'hui.

La sortie en 1999 du livre de Kent Beck « Extreme Programming Explained » avait déjà formalisé des pratiques techniques fondées sur des itérations courtes de développement logiciel, qui participent du socle de l'Agilité. Les méthodes de développement RAD – Rapid Application Development – et RUP – Rational Unified Process – dans les années 1990 avaient aussi proposé des démarches itératives et incrémentales. De plus, les pratiques industrielles du Lean Manufacturing, popularisées par Toyota dès les années 70-80 et visant une recherche d'efficience dans la production, constituent aussi une source d'influence historique importante dans le développement des approches Agile du développement logiciel.

Tout cela donne lieu aujourd'hui à différents cadres méthodologiques, tant dans la conduite et l'organisation des activités et des rôles des projets en Agile, que dans les pratiques d'ingénierie du développement et des tests associés. Ces approches utilisent souvent un vocabulaire très spécifique (un jargon en fait), dans lequel il est important de se repérer lorsque l'on est testeur en contexte agile. Ce chapitre vise à fournir des repères, de façon synthétique et sans remplacer les formations et certifications utiles dans le domaine (et en particulier la certification Testeur agile CFTL/ISTQB). Si vous êtes un Agiliste chevronné, vous pouvez facilement vous passer de la lecture de ce chapitre, sinon, vous y trouverez quelques repères utiles pour la suite du livre.

1- Cadres méthodologiques de l'Agilité.

Comme le montre la Figure 4 du chapitre précédent sur les taux d'utilisation des différentes approches agiles aujourd'hui, il y a une forte domination de Scrum, mais aussi un ensemble d'autres approches utilisées, y compris avec des hybridations entre approches. Comme nous l'indiquions précédemment, l'Agilité fournit un cadre au sein duquel l'esprit (les valeurs et principes de l'Agilité) doivent prévaloir sur le formalisme. Ce qui permet une bonne adaptation des approches agiles au contexte spécifique des projets et des organisations.

On peut distinguer les approches agiles plus orientées sur les pratiques de l'ingénierie du logiciel telles que XP et Software craftsmanship, celles plus orientées sur l'organisation des développements en mode itératif et incrémental (c'est-à-dire plus focalisées sur la conduite du projet agile) telles que Scrum et Kanban et plus récemment les approches visant à organiser les pratiques agiles à grande échelle – c'est-à-dire pour de grandes entreprises et/ou projets, telles que SAFe ou le modèle Spotify.

Nous listons dans le tableau suivant les principaux cadres méthodologiques de l'Agilité aujourd'hui utilisés (à partir des enquêtes disponibles telle que « State of Agile », déjà citée).

Approche Agile	Description
XP – eXtreme Programming	XP tient un rôle historique très important dans l'émergence de l'Agilité. Ken Beck a publié en 1999 son livre « Extreme Programming explained » à partir d'expériences de développement d'une application Système d'Information chez Chrysler. XP met en avant des cycles courts de développement et des pratiques d'ingénierie telles que le Pair Programming, le refactoring régulier du code, l'intégration continue, le pilotage par les tests (Test-driven development – TDD), la conception simple et incrémentale, le feedback au plus tôt, …. La plupart des pratiques définies par XP ont infusé dans l'ensemble des approches de l'Agilité.
Software craftsmanship	L'idée du « software craftsmanship » est de constater que le développement logiciel peut être vu comme un travail d'artisan, qui peaufine son ouvrage avec d'autres compagnons ! Il s'agit de mettre en avant la nécessaire qualité du code, l'excellence attendue des développeurs et la fierté d'un bel ouvrage (c'est-à-dire d'un logiciel bien conçu, maintenable et sans bug). Les développeurs doivent rechercher cette qualité du code en progressant constamment et collectivement, pour délivrer de la valeur pour les clients.
Scrum	Il s'agit du cadre méthodologique le plus déployé aujourd'hui et fondé sur 3 piliers : les rituels (Scrum meeting, revue de sprint et démo, rétrospective, et planification incrémentale), les rôles (Product Owner, Scrum Master, développeurs et testeurs) et les artefacts gérés (Backlog du Produit et Backlog du Sprint). Les itérations (appelées Sprints) sont de 2 à 4 semaines généralement, de durée constante et avec l'implémentation du timeboxing (les mises en production se font à date fixe et la variable d'ajustement se fait sur le périmètre). L'expression des besoins est formalisée au travers de User Story, de critères d'acceptation et de scénarios d'acceptation.
Kanban	Il s'agit à l'origine (dès les années 70-80 chez Toyota) d'une méthode de production industrielle visant à organiser la production dans une démarche juste-à-temps, ou zéro-stock, c'est-à-dire intégrant les tâches de production pour les enchaîner de la façon la plus efficiente possible. En développement logiciel, cela se traduit en particulier par le tableau Kanban, qui visualise l'avancement des activités, typiquement entre A-faire, En-cours, En-test et Terminé. Ce tableau Kanban est repris dans le plupart des approches agiles, et en particulier en Scrum. Il y a des différences importantes entre Kanban et Scrum, en particulier le fait que Kanban organise un processus continu de production, adapté à la prise en compte des changements, sans mettre l'accent de façon stricte sur des itérations de durée fixe. Alors que l'itération de durée fixe (le Sprint) est le fondement de Scrum.

ScrumBan	ScrumBan vise à amender l'usage de Scrum sur plusieurs aspects qui peuvent créer des difficultés dans certains contextes : permettre des sous-équipes spécialisées au sein de l'équipe intégrée (par exemple sur une technologie particulière), intégrer un pilotage d'équipe et pas seulement d'auto-organisation promue par Scrum, gérer les tâches dans une logique de flux de production avec une vision globale de l'avancement de la User Story (plutôt que la vision en tâches de Scrum), réaliser la planification en fonction des besoins (lorsque la liste «A faire» courante s'amenuise).
SAFe	Scrum est typiquement adapté à des équipes de 6 à 9 personnes. Imaginez un Scrum meeting (la réunion quotidienne debout) avec 25 personnes autour de la table ! Cela ne peut pas fonctionner et pose la question d'une Agilité à l'échelle : comment passer à l'Agile sur un projet de grande taille ? SAFe – Scaled Agile Framework – vise à répondre à cette question en synchronisant plusieurs équipes agiles de petite taille autour d'un même système et objectif de production. La première version de SAFe a été proposée en 2007 et le schéma est en régulière évolution. La version actuelle (SAFe v4.6) propose de coordonner les équipes agiles par le concept de train de livraison (Agile Release Train) pour cadencer les mises en production. Le concept reprend l'idée du Timeboxing, pour synchroniser les mises en production régulières des développements réalisés par les différentes équipes et les intégrer dans un seul système.
Modèle Spotify	L'organisation proposée en 2012 par Spotify pour l'Agilité à l'échelle est le concept de Squad, c'est-à-dire d'équipes agiles (5 à 7 personnes typiquement), autonomes dans leur développement sur des fonctionnalités données. Les squads sont regroupés en tribus, regroupant et synchronisant les squads travaillant sur un même thème global, avec une articulation en chapitres pour synchroniser les développements liés entre eux. L'état d'esprit de chaque Squad est celui d'une start-up : délivrer de la valeur, rechercher le feedback, et s'adapter pour satisfaire l'utilisateur.
Lean Startup	Le concept de Lean Startup date du début des années 2000 avec les travaux de Steve Blanck (customer developement) puis de Eric Ries (qui a publié le livre Lean Startup en 2011). Les concepts clés sont ceux du MVP – Minimum Viable Product – c'est-à-dire la recherche de feedback au plus tôt pour qualifier et adapter la proposition de valeur au client. Cette adaptation (le terme pivot est utilisé), par itérations successives, en cherchant l'économie de moyen, est la clé du succès : échouer rapidement pour réussir plus vite ! Le lien avec l'Agilité est central dans l'approche Lean Startup : des itérations courtes et une recherche constante pour prioriser par la valeur délivrée et le feedback des utilisateurs.

2- Principes et pratiques courantes de l'Agilité.

Les pratiques de base de l'Agilité concernent à la fois l'organisation des développements, les pratiques de codage et l'organisation des équipes. Ces pratiques sont promues, parfois avec des variantes, dans la plupart des approches agiles listées précédemment. Voici quelques-unes des principales pratiques couramment utilisées dans l'Agilité.

Développement itératif et incrémental : c'est la pratique de base de l'Agilité consistant à organiser les livraisons en petits incréments au travers d'itérations courtes (typiquement 2 à 4 semaines en Scrum). Chaque itération vise la production d'un logiciel exploitable et apporte un supplément de valeur par rapport à l'incrément précédent.

12 principes généraux de l'Agilité

1- Satisfaire le client en priorité
2- Accueillir favorablement les demandes de changement
3- Livrer le plus souvent possible des versions opérationnelles de l'application
4- Assurer une coopération permanente entre le client et l'équipe projet
5- Construire des projets autour d'individus motivés
6- Privilégier la conversation en face à face
7- Mesurer l'avancement du projet en termes de fonctionnalités de l'application
8- Faire avancer le projet à un rythme soutenable et constant
9- Porter une attention continue à l'excellence technique et à la conception
10- Faire simple
11- Responsabiliser les équipes
12- Ajuster à intervalles réguliers son comportement et ses processus pour être plus efficace

Equipe intégrée : là aussi, un concept de base – la réussite des développements, passe par une communication permanente au sein d'une équipe intégrant le (la) représentant(e) du client (le rôle de Product Owner en Scrum), les développeurs et les testeurs (jusqu'au opérationnels en DevOps). L'idéal est d'avoir cette équipe intégrée, collectivement responsable de la qualité de la production, sur un même site pour favoriser la communication permanente entre les membres de l'équipe.

Stand-up meeting : on évite les réunions lourdes et faisant perdre du temps en Agile, mais l'équipe fait un point quotidien, debout, permettant une synchronisation rapide des activités en cours et des éventuelles difficultés rencontrées. En Scrum, chacun répond aux questions « Qu'est-ce que j'ai fait, que suis-je en train de faire, est-ce que je rencontre des difficultés ou aspects qui me ralentissent ? ».

Revue d'itération : chaque fin d'itération produit un logiciel fonctionnel qui est démontré lors de la revue d'itération à l'ensemble des parties prenantes du projet. L'objectif est d'obtenir du feedback sur les développements réalisés pour réajuster si nécessaire, mais aussi de démontrer l'avancement et donner de la visibilité au projet.

Rétrospective : Une autre pratique de base consiste, pour l'équipe agile, à analyser les pratiques pour en constater les réussites et les améliorations nécessaires. Idéalement, chaque fin d'itération donne lieu à rétrospective.

Planification incrémentale / planning poker : Le planning poker est une méthode d'estimation de l'effort de développement fondée sur l'expertise des membres de l'équipe ; les User Story ou les tâches de développement sont estimées dans leur complexité relative au travers d'un système de points suivant la suite de Fibonacci que l'équipe agile s'approprie. La planification

est incrémentale car plus l'horizon s'éloigne et moins l'équipe a besoin d'être précise dans son évaluation et inversement : les activités du prochain Sprint doivent être précisément estimées pour être planifiées.

Vélocité : c'est la mesure, en points, de la capacité à faire d'une équipe agile. La mesure de la vélocité permet à l'équipe de réaliser un engagement réaliste pour chaque Sprint. Chaque « point » étant différent selon les équipes, la vélocité ne permet pas de comparer la productivité de différentes équipes.

Timeboxing : Cela consiste à planifier la production sur la base de dates fixées de release et donc d'adapter le contenu comme variable d'ajustement aux aléas de production.

Intégration continue : Automatiser la production de l'application, à chaque changement de la base de code et lancer les différents types de de test, pour donner un feedback immédiat à l'équipe, constitue l'objectif de l'intégration continue. C'est une des pratiques essentielles de l'Agilité car elle apporte une vision partagée de l'avancement du développement et de la qualité du code. Plus d'informations dans le chapitre dédié à ce sujet.

Un petit exercice

Retrouver le sens de ces termes courants de l'Agilité : Backlog du Produit, Backlog du Sprint, Définition du terminé (Definition of Done), Définition du prêt (Definition of Ready), Epic et User Story, Refactoring et Pair Programming.

3- Pratiques d'ingénierie du développement et des tests en Agile

Le premier groupe de pratiques importantes pour les tests en Agile concerne les 'xDD' : TDD, ATDD et BDD. Il s'agit, dans les 3 cas, de mettre en œuvre le principe d'un pilotage par les tests mais à des niveaux différents ou avec des nuances d'application.

TDD – Test Driven Developement – est une pratique des tests unitaires qui consiste à définir les tests d'une méthode ou d'une fonction avant de coder ladite méthode ou fonction. En d'autres termes, le développeur doit penser et formaliser les cas de tests du code à implémenter comme précurseur du code lui-même (et non pas l'inverse qui consiste à développer les tests du code une fois le code écrit). L'enquête « State of Agile » montre que 75% des répondants réalisent des tests unitaires (on pouvait s'attendre à mieux !) et que 35% des répondants indiquent pratiquer du TDD.

ATDD / BDD – Acceptance Test Driven Developement / Behavior Driven Development – sont deux approches qui se situent au niveau des tests systèmes et d'acceptation. Il s'agit, dans les deux cas, de favoriser la compréhension partagée de l'expression du besoin, en formalisant les scénarios d'acceptation et les comportements à tester lors du raffinement de cette expression du besoin (donc du raffinement des User Story en Scrum). Le chapitre II.6, consacré à ces pratiques, précise sur la base d'exemples les nuances entre ces deux approches, dont la popularité s'accroît de façon importante et régulière dans les projets en Agile. Par exemple, dans l'enquête « Software Testing Profession – 2017-2018 » de l'institut Techwell, ATDD ressort comme utilisé par 45% des répondants, et dans le Top 5 des techniques de test au côté des tests dirigés par les exigences et les risques. Le succès des approches ATDD / BDD est lié au gain d'efficience de l'équipe, apporté par une scénarisation des tests d'acceptation réalisée au

moment de la définition du besoin. D'une part, cette pratique permet de gagner en précision dans la définition du besoin et, d'autre part, la conception des scénarios d'acceptation réalisée à ce moment permet une bonne couverture des User Story.

Un autre concept important de l'ingénierie des tests en Agile est celui de la **pyramide des tests**. Pour faire simple, l'idée est d'éviter la redondance des tests et de considérer que le premier filet de sécurité, pour détecter les régressions logicielles, commence par une bonne couverture du code par les tests automatisés unitaires et d'intégration des composants. Les tests des services au niveau système, plus difficiles à automatiser, doivent se focaliser sur les workflows Métier et les tests de l'IH/M sur l'usage. La pyramide désigne ainsi une base forte (de tests unitaires et d'intégration) et des tests à forte valeur Métier pour les niveaux systèmes et acceptation.
Enfin, terminons cette section des pratiques agiles par le **DevOps**, qui vise à intégrer les phases de développement et de production. Le DevOps est avant tout une philosophie visant à casser les silos entre les différents services. Là où l'Agilité traditionnelle a cassé les silos entre le besoin, le développement et le test, le DevOps (Dev pour Developpement et Ops pour Opérationnels) casse le silo entre la production et le développement. S'ensuit une accélération de la cadence des mises en production, qui constitue la première motivation du passage à l'Agilité (cf. chapitre I.2). Le DevOps trouve son prolongement naturel dans l'idée du « continuous delivery », pratique avec laquelle le DevOps est souvent confondu. Il s'agit de penser le déploiement des évolutions du logiciel en même temps que ces évolutions elles-mêmes, pour que le passage en production puisse se faire de façon la plus continue possible. Le test est bien sûr directement impacté par cette évolution DevOps, car le fait d'aller vers des déploiements en continu implique de suffisamment maîtriser les tests (automatisés donc) en continu. L'idée du DevOps est assez générale et s'applique à l'ensemble des systèmes ; en revanche, le déploiement en continu n'est clairement pas adapté à tous les contextes (vous feriez du déploiement en continu sur le logiciel critique de pilotage d'un robot médical en opération ?).

4- Comment se maintenir à jour quand on est un professionnel du test ?

Le domaine évolue vite. Le terme DevOps que nous venons d'évoquer date de 2009 et le framework SAFe évolue rapidement à partir des retours des déploiements. La combinaison Scrum et Kanban est devenue ScrumBan et la nouvelle approche agile, qui va sortir demain, n'est pas encore connue. Donc, pas le choix, il faut se former en continu quand on est un professionnel des tests logiciels en projet agile (les médecins font cela non ?) !

Voici quelques astuces pour ce maintien en condition opérationnelle du Testeur en continu (et sans parti pris ☺) :

1. Venez chaque année à la **JFTL** – Journée Française des Tests Logiciels – car c'est là que vous allez le mieux ressentir et découvrir les tendances importantes de la profession. Vous n'avez pas pu avoir de place car c'était complet ? Il faudra vous y prendre plus tôt l'année prochaine !

2. Lisez les blogs et le plus fréquenté d'entre eux dans l'espace francophone : **« La taverne du Testeur »**, car c'est une mine d'informations avec de nouveaux articles chaque semaine. Vous pouvez aussi partager votre expérience sur le groupe LinkedIn **« Le Métier du test »**, suivi par plus de 2500 membres.

3. Assistez aux événements qui se passent dans votre bassin d'activités, de plus en plus de **conférences, de soirées** ou de **meetups** dédiés aux tests et à l'Agilité ont lieu.

4. Formez-vous et **certifiez-vous avec le schéma CFTL/ISTQB** ! Au niveau Fondation, bien sûr, si ce n'est déjà fait mais aussi en tant que Testeur Agile et en considérant les niveaux avancés (Test Manageur, Analyste de Test et Analyste Technique de Test) et Automaticien de tests.


DURANT LE SPRINT, ON FAIT LA RÉUNION SCRUM QUOTIDIENNE DEBOUT !
TESTEUR, LEVEZ VOUS !
C'EST QU'IL A UN PEU DE MAL À SE REMETTRE DU TEST D'HIER...
SCRUM CHEF
SCRUM CRÈME
GÂTEAU 3.6.2
F. COINTE

I.4- L'organisation des tests en Agile

par Marc Hage Chahine

1- Introduction.

L'Agilité est une approche de développement logicielle totalement différente des approches classiques comme le cycle en V.

L'approche agile adoptant un état d'esprit (ou une vision) différent du développement des logiciels, tous les maillons de la chaîne de développement sont impactés et doivent s'adapter à cette nouvelle manière de procéder, de penser. Le test est également impacté en tant que maillon prépondérant de la chaîne du développement logiciel.

Afin de bien comprendre les principales divergences entre un cycle de développement dit classique et un cycle de développement agile, j'aime utiliser une analogie : celle de la construction d'un pont.

*Pour construire un pont en adoptant **une méthode comparable au cycle en V** :*

1. *On fait les plans, on fait intervenir les architectes et on vérifie la conformité aux normes (durée : 6 mois).*
2. *On commence la construction par les piliers (durée : 4 mois).*
3. *On finit la construction, on vérifie l'ensemble des normes de construction (durée : 7 mois)*
4. *On annonce l'ouverture et on prépare l'arrivée des automobilistes (durée : 1 mois)*
5. *Ouverture du pont*

***Bilan :** Le projet dure 18 mois. Les automobilistes peuvent donc raccourcir leur trajet au bout de **18 mois**.*

*Si on devait construire un pont en adoptant **une méthode comparable à une méthode agile** :*

Attention, cela n'est pas forcément instinctif. Il faut revenir à la base. Quel est le souhait des automobilistes ? Le souhait n'est pas d'avoir un pont ! Non, le souhait est de ne plus avoir à faire 40 minutes de trajet supplémentaire tous les matins.

Le projet de pont en méthode agile se présenterait donc ainsi :

1. *Faire venir (ou acheter) un ou plusieurs bateaux pour faire des allers-retours et permettre aux automobilistes de gagner l'autre rive – certes moins vite qu'avec un pont, mais beaucoup plus qu'en faisant le détour – (durée : 2 mois).*
   On récupère des informations de trafic afin de mieux connaître/comprendre les utilisateurs.

2. On fait les plans et on fait intervenir les architectes, on vérifie la conformité aux normes (durée : 6 mois) et on adapte aux retours des utilisateurs si besoin (ex : trottoirs si beaucoup de piétons, plus de voies si trafic important…).

3. Pendant que les bateaux font leurs allers-retours et que de l'argent commence à rentrer, le pont est construit, en commençant par les piliers (durée : 6 mois car il ne faut pas déranger les bateaux).

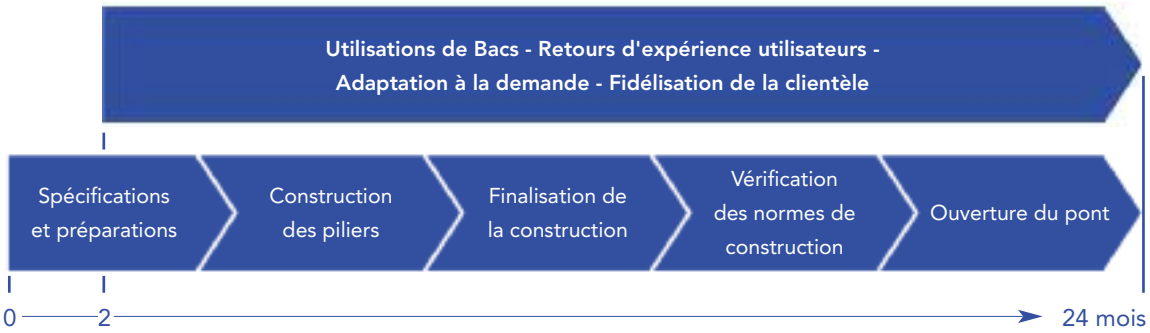
Les plans peuvent être modifiés en fonction de l'utilisation des bateaux (bacs)

4. On finit la construction, on vérifie l'ensemble des normes de construction à chaque étape (durée : 9 mois, toujours pour ne pas déranger les bateaux).

5. On annonce l'ouverture prochaine aux utilisateurs des bateaux (1 mois)

6. Ouverture du pont.

***Bilan :** Le projet dure 24 mois, soit 6 mois de plus que le cycle en V. Ce même projet coûte également plus cher car il a fallu acheter ou louer les bateaux. En revanche, le service pour faire traverser le fleuve a été opérationnel au bout de 2 mois au lieu de 18 (mais on a accès au pont au bout de 24 mois).*



Conclusion :

Il n'y a pas de bonnes ou de mauvaises méthodes. A périmètre égal et défini au début du projet, une méthode agile peut coûter plus cher qu'une méthode classique, notamment à cause des tests qui prendront une part plus importante (car exécutés beaucoup plus fréquemment). L'intérêt principal des méthodes agiles est d'arriver plus rapidement sur le marché pour pouvoir adapter son périmètre en fonction des retours et des besoins exprimés pas les utilisateurs. Un développement agile coûtera beaucoup moins cher qu'un développement classique nécessitant des reprises, car ce dernier ne correspond pas (ou plus) au besoin.

Nous voyons bien, à travers cet exemple, que la manière de fonctionner avec un esprit « agile » est nettement différente de la manière « classique ». Les tests dans une méthode agile vont prendre une place plus importante que dans une méthode classique. Le test d'un logiciel coûte plus cher en agile que dans un cycle en V ! Les raisons principales sont :

1. L'accès régulier à de nouvelles versions du logiciel par des utilisateurs finaux.
    - Cet accès régulier à de nouvelles versions « force » l'équipe agile à tester intégralement son produit avant chaque mise à disposition. Là où un cycle en V ne proposait qu'une seule phase de test, les méthodes agiles peuvent en proposer des dizaines !

2. Une construction itérative.

- Lorsque l'on construit de manière itérative un produit, on se retrouve à avoir comme matériau de base pour notre construction ce que l'on a déjà développé. Notre produit devient alors les fondations de notre produit futur.
- Pour avoir une construction solide, il faut partir sur de bonnes bases. Le produit étant la base, sa qualité (fonctionnelle ou non) ne peut être négociée et il faut donc des tests afin de s'assurer que le niveau de qualité suffisant est atteint.

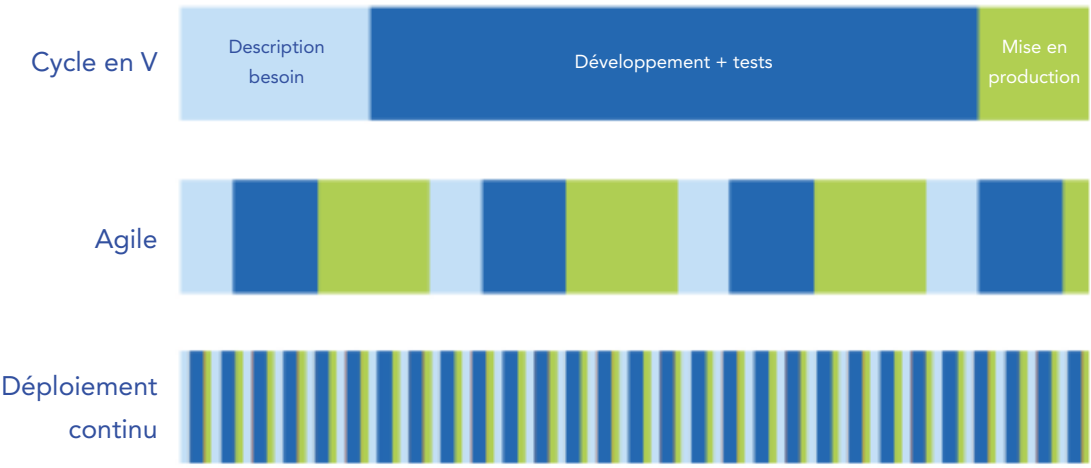
L'organisation des tests en Agile doit donc s'adapter à ces 2 points. Pour cela, il n'y a pas de solution unique. Néanmoins, grâce à l'apparition des équipes pluridisciplinaires et à une communication qui s'en trouve améliorée, il n'y a rien d'insurmontable !

2- Organiser les tests afin de pouvoir livrer de manière régulière de nouvelles versions du produit.

Comme décrit dans la partie précédente, l'Agilité permet de livrer régulièrement de nouvelles fonctionnalités d'un produit utilisable. Afin que cette promesse ne soit pas vaine, il faut :

- Assurer que les nouvelles fonctionnalités aient une vraie valeur ajoutée.
- Assurer que les nouvelles fonctionnalités soient vraiment utilisables.
- Assurer que les anciennes fonctionnalités soient encore utilisables.

Voici un schéma permettant d'appréhender la différence d'échelle entre les mises en service avec le cycle en V et avec des méthodes agiles :



Afin de répondre au premier point, l'Agilité propose une gestion des demandes à travers un « backlog » et sa priorisation. Les fonctionnalités à développer sont sélectionnées à partir de ce backlog. Le rôle des tests sur ce point est de questionner la pertinence de la fonctionnalité présentée lors des réunions permettant de décrire ces besoins.

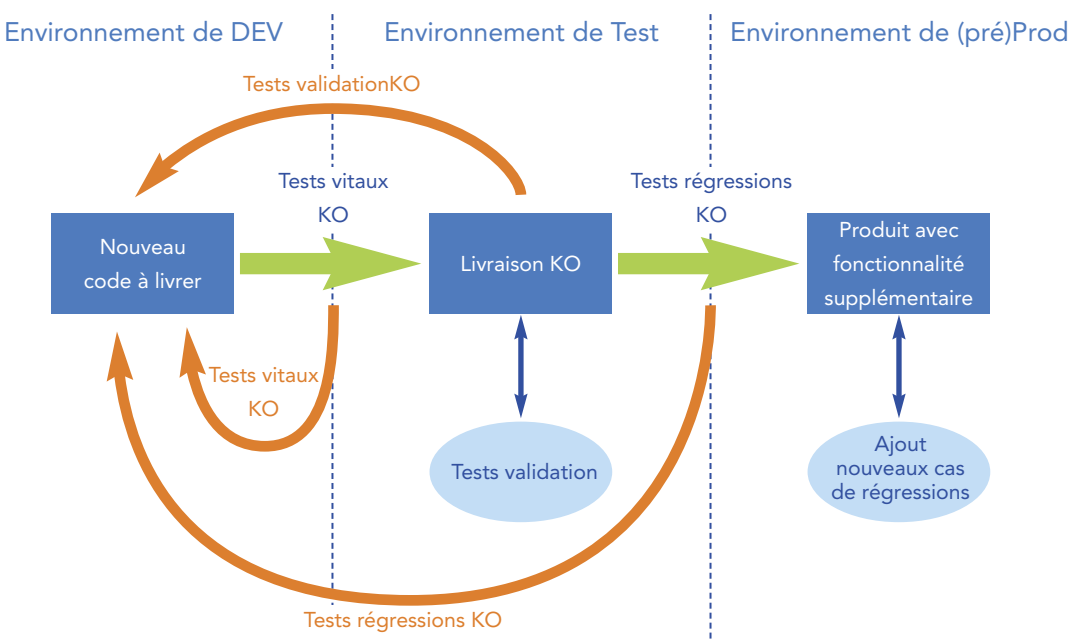
Les tests sont primordiaux pour répondre au second point. A chaque nouvelle livraison, un produit propose une ou plusieurs fonctionnalités ainsi qu'un ou plusieurs correctifs d'anomalies. Afin de répondre à ce besoin, il faut donc prévoir des tests pour valider ces correctifs et ces fonctionnalités.

Pour cela, il faut concevoir, écrire et exécuter ces tests pendant l'itération. Il devient donc nécessaire de pouvoir commencer à travailler sur les tests avant la livraison des spécifications (s'il y en a) et avant la livraison du code. Cela engendre naturellement de nouvelles pratiques, comme l'ATDD ou le BDD même si cela n'est pas forcément obligatoire. Ce qui est en revanche obligatoire, c'est que tous les membres de l'équipe soient capables d'aider sur des tâches du test. Comme dans tout développement, les charges de test sont sujettes à des pics, pendant lesquels le(s) testeur(s) aura(ont) besoin d'aide, et des creux pendant lesquels le testeur devra aider sur des tâches autres que les tâches de test.

Enfin, et c'est la partie la plus importante, les tests doivent également permettre d'assurer que ce qui a été construit lors des itérations précédentes est toujours en état de fonctionner. Les utilisateurs peuvent être compréhensifs lorsqu'ils n'arrivent pas à utiliser une nouvelle fonctionnalité implémentée, ils le sont en revanche beaucoup moins lorsqu'une fonctionnalité qu'ils ont l'habitude d'utiliser, devient inutilisable suite à une mise en service. Afin de limiter au maximum ces problèmes, l'équipe se doit de tester, au moins avant chaque mise en service, les anciennes fonctionnalités de son produit. Pour cela il faut exécuter les tests de régression (souvent appelés tests de non régression). Ces tests de régression sont fréquemment exécutés, il est donc intéressant de prévoir de les automatiser.

En fait, en Agilité, on se retrouve avec une organisation des tests qui peut ressembler à celle-ci :

Implémentation d'une User Story en Scrum



Dans l'image ci-contre, les tests vitaux correspondent à un sous-ensemble des tests de régression, les tests de validation sont les tests permettant de valider le comportement de la nouvelle fonctionnalité et les tests de régression correspondent à la campagne regroupant les tests utilisés, afin d'assurer que les nouvelles fonctionnalités n'ont pas introduit de régression majeure sur le produit.

Enfin, toujours dans le cadre de l'Agilité, il peut être très intéressant de mettre en place des sessions de tests exploratoires. Ces tests permettent de gagner du temps sur la conception et l'écriture mais aussi de lutter contre le paradoxe des pesticides en exécutant régulièrement des tests différents.

3- Organiser ses tests afin de gérer, sur le long terme, une construction itérative du produit.

Même si l'on parle de « sprint » avec le Scrum, un produit agile n'est pas quelque chose que l'on construit sur du court terme (à quoi bon faire de l'itératif si l'on peut construire un produit en moins de 2 mois ?). Un produit agile est un produit qui peut se construire sur plusieurs années (certaines applications mail mobiles ont largement dépassé les 5 ans !) et peuvent ainsi devenir complexes et comporter un très grand nombre de fonctionnalités. Cela oblige donc à avoir des campagnes de régressions performantes tout en maintenant une durée d'exécution, de maintenance et d'analyse restreinte.

C'est pour cette raison que la campagne des tests de régression est complexe à gérer et devient la source de nombreux problèmes pouvant amener à une fin de vie prématurée du produit. Afin d'avoir une bonne campagne de régression, il faut :

- Eviter de faire de la sur-qualité.

Il arrive régulièrement qu'une équipe agile, pas assez mature d'un point de vue test, souhaite faire de la sur-qualité en intégrant tous ses tests dans la campagne de régression. Cette démarche ne peut fonctionner qu'à court terme ! A moyen terme, le nombre de tests à maintenir, exécuter et analyser devient insoutenable car en constante croissance, l'équipe en arrive donc à un point où elle a le choix entre ne plus maintenir et exécuter les tests, ou passer tout son temps sur des tâches de test ! Les deux solutions sont mauvaises.

- Bien sélectionner ses tests de régression.

La sélection des tests de régression doit se faire dès la conception des tests d'une fonctionnalité. Les tests de régression étant les seuls à devoir être ré-exécutés ils doivent faire l'objet d'une attention particulière en permettant notamment une automatisation aisée. Pour rappel, afin d'avoir une automatisation aisée il faut, au minimum, avoir des scripts bien écrits (1 action => 1 (ou plusieurs), des résultats stables (ne dépend pas du jour ou de l'heure) et fiables (vérifie ce qu'il y a vraiment à vérifier).

Cette sélection doit être faite en fonction de la criticité du parcours ainsi que de sa probabilité de défaillance.
Enfin, comme toute sélection, on ne doit prendre qu'une partie des tests qui ont servi à valider la fonctionnalité.

- Maintenir sa campagne de tests de régression.

Une campagne de test non maintenue est pire que pas de campagne du tout ! La raison est simple : avec une campagne non maintenue, l'équipe garde l'illusion que son produit est protégé par des tests permettant de repérer les bugs majeurs ou critiques. Ce n'est malheureusement pas le cas ! Les membres de l'équipe sachant que leur campagne n'est pas maintenue n'analysent pas les tests constamment en échec. Pire, quand il y a trop de tests non maintenus, l'équipe n'analyse plus du tout la campagne, même si cette dernière est exécutée automatiquement.

Afin d'éviter ces problèmes, il est nécessaire de maintenir sa campagne de régression. Cette maintenance doit être faite de différentes manières, en voici quelques-unes :

– Mettre à jour ses cas de tests dès qu'ils sont impactés par un nouveau développement ajouté au produit.
– Lutter contre le paradoxe des pesticides : pour cela, les tests exploratoires sont une très bonne réponse mais pas forcément suffisante. Afin d'améliorer l'efficacité de la campagne de régression, il est conseillé de modifier des tests qui ne sont jamais en échec en proposant par exemple des jeux de données différents.
– Supprimer des tests s'avérant inutiles.
– Enrichir les campagnes aves des tests couvrant de nouvelles fonctionnalités
– Supprimer les tests les moins importants afin de garder un volume de tests que l'on peut exécuter, analyser et maintenir.

- Automatiser ses tests.

L'automatisation des tests devient une nécessité avec les méthodes incrémentales. Aucun testeur ne peut ré-exécuter manuellement plusieurs fois par semaine pendant des années les mêmes tests. Afin de préserver les testeurs (mais aussi le temps de l'équipe agile), il est nécessaire d'automatiser une partie de ses tests.
Pour cela, de nombreux outils et méthodes ont émergé. Il existe notamment l'ATDD et le BDD qui visent à faire des tests une documentation vivante du produit (et donc de n'avoir que les tests à maintenir plutôt que les tests et la documentation), mais aussi à faciliter l'automatisation par l'intermédiaire de mots clés et de l'homogénéisation de l'écriture des cas de test.

4- L'Agilité à l'échelle.

Construire un produit grâce aux méthodes agiles, c'est bien. Néanmoins, il n'est généralement question que de l'équipe agile. Or une équipe agile, c'est en théorie entre 5 et 9 personnes. Il est malheureusement impossible sur certains projets ou produits de ne faire travailler que 5 à 9 personnes.
Pour ce type de produit, plusieurs équipes doivent travailler ensemble. Afin de synchroniser ces équipes, il faut mettre en place des processus leur permettant de toutes travailler dans le même sens. Pour cela, diverses méthodes ont été créées, elles sont appelées «Agilité à l'échelle ».

Actuellement, les 2 modèles dominants sont les modèles « Spotify » et « SAFe ». Dans ce livre, vous avez également un retour d'expérience sur le « Scrum of Scrum ».

Exemple d'organisation sur l'Agilité à l'échelle :

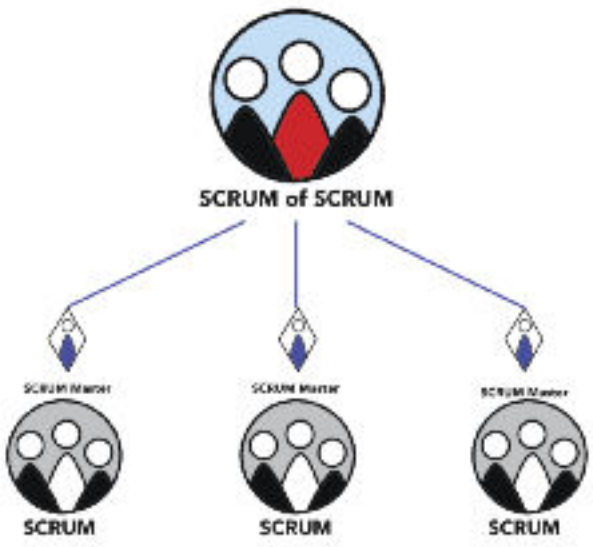


Image créée par Marcelo Kamenetz Szwarcbarg

Chacun de ces modèles a ses spécificités mais ils gardent tous des points communs qui sont :

- On ne négocie pas la qualité du produit.

Ce point ne nécessite pas forcément une forte adaptation. Néanmoins, chaque nouvelle fonctionnalité étant ajoutée sur le produit global, il est important d'avoir accès à des campagnes de tests d'autres équipes afin de vérifier si nos développements n'ont pas impacté négativement les autres équipes.

Afin de repérer ces impacts le plus rapidement possible, il peut être intéressant d'avoir une base de tests vitaux (voire plus) commune, voire une équipe dédiée à la supervision de ces tests de régression du produit.

- Il est bon de regrouper les personnes ayant les mêmes compétences au sein de groupes de compétences.

Plus les produits nécessitent de personnes pour être développés, plus il y a des compétences communes dispersées dans les différentes équipes. Permettre à ces personnes de se regrouper et d'échanger sur leurs expériences, problèmes et succès permet de faire gagner du temps à toutes les équipes.

Ces mêmes groupes peuvent travailler à la mise en place d'outils ou pratiques communs qui seront bénéfiques à l'ensemble des équipes agiles travaillant sur le projet.
Cela est vrai pour le test mais pas uniquement.

Exemple pour le Scrum of Scrum :

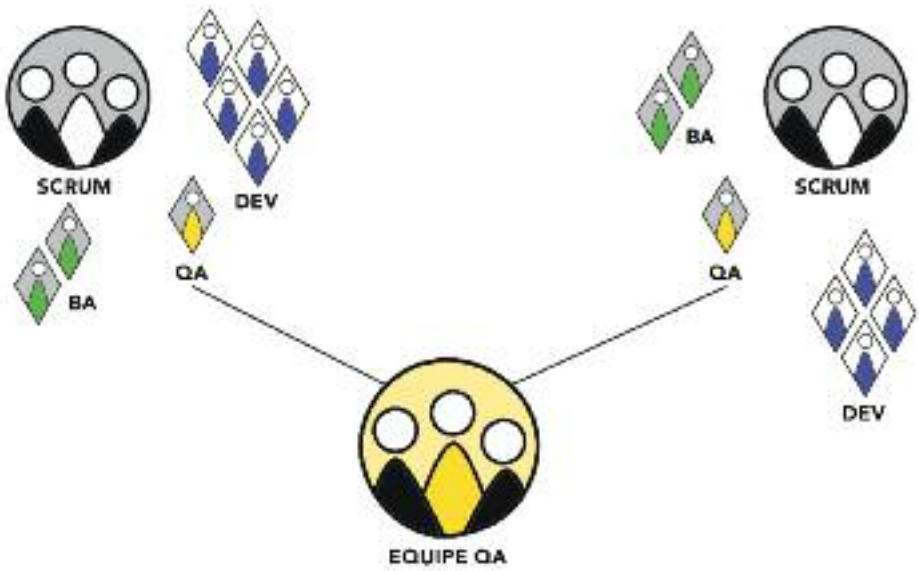


Image créée par Marcelo Kamenetz Szwarcbarg

• Les équipes agiles doivent communiquer entre elles car elles travaillent toutes sur le même produit.

Les équipes travaillant sur le même produit sont nécessairement dépendantes les unes des autres. L'équipe A peut se retrouver à attendre un développement de l'équipe B tandis que l'équipe C peut créer une régression dans une fonctionnalité développée par l'équipe B… Afin de limiter ces risques, les équipes doivent se synchroniser, savoir comment ces risques impactent les autres équipes mais aussi ce que les autres équipes attendent d'elles.

Sans communication et organisation, les équipes finissent par perdre beaucoup de temps avec des corrections d'anomalies, de l'attente de fonctionnalités voire des développements incompatibles nécessitant de redévelopper entièrement certaines parties du produit.

5- Conclusion.

L'organisation des tests en Agile n'a rien à voir avec l'organisation des tests dans les méthodes classiques. Le changement de méthode de développement induit un changement d'état d'esprit, la livraison de nouvelles versions du produit de manière très fréquente, de même que la création de produits de manière incrémentale engendre de nouveaux besoins. Ces besoins trouvent des réponses (processus, automatisation, tests exploratoires, ATDD/BDD…) qui sont d'un point de vue macroscopique très génériques mais qui vont devoir être adaptées à chaque équipe d'afin d'atteindre un niveau optimal.
Enfin, l'Agilité à l'échelle ajoute une couche supplémentaire de complexité et oblige les membres de chaque équipe agile à ne pas penser uniquement à elle mais à tout un environnement constitué de plusieurs équipes agiles.

I.5- Le rôle des testeurs dans les projets/équipes agiles

par Marc Hage Chahine

Avec l'Agilité, le cycle de développement est bouleversé. Le logiciel n'est plus livré en une fois mais en de multiples versions embarquant chacune une ou plusieurs nouvelles fonctionnalités. Les tests se font donc à tout moment du projet, il n'y a plus de « période » spécifique dédiée aux tests ou à certains types de tests.

Nous étudierons dans ce chapitre les changements apportés par l'Agilité pour les tests, notamment en nous attachant au fait que l'Agilité regroupe un ensemble de méthodes qui ont toutes quelques points communs éloignés d'un cycle en V traditionnel comme :

- La construction d'un logiciel orienté sur la valeur produit.
- La construction itérative d'un logiciel.
- La mise en relation, au sein d'une même équipe, de plusieurs « métiers » différents.

Nous aborderons ensuite les impacts de ces changements dans le travail de testeur en étudiant les nouveaux rôles des testeurs en agilité. Ces rôles sont :

- Etre garant de l'esprit de la qualité logicielle.
- « Chef de projet test ».
- Apporter son aide là où elle est la plus utile à l'équipe, même si ce n'est pas du test.

1- La construction d'un logiciel orienté sur la valeur produit.

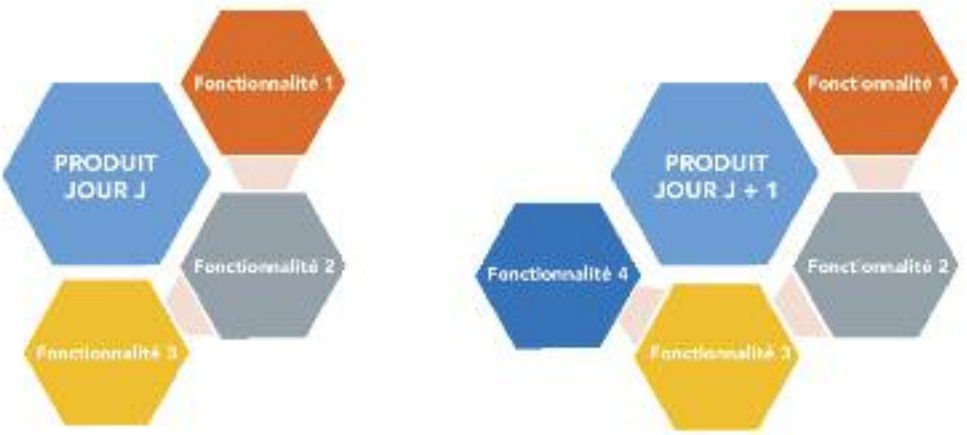
Par « construction d'un logiciel orienté sur la valeur produit », il faut comprendre que toute la construction du logiciel est basée sur ce que la fonctionnalité ajoutée peut apporter.

Le produit est donc :

- Un produit/logiciel, agglomération de fonctionnalités… Ces fonctionnalités pouvant évoluer, changer au cours de la vie du projet.

Le produit est développé par fonctionnalité. Chaque fonctionnalité est une « brique » qui s'ajoute au produit déjà existant. L'accumulation de ces briques forme un ensemble qui est le produit. Ce produit évolue et voit son nombre de briques augmenter tout au long de son développement.

On a donc un produit qui grandit, ce qui peut se résumer avec les images ci-dessous :



- Un produit/logiciel, construit en fonction des retours clients.

Un produit n'est pas totalement défini à l'avance avec les méthodes agiles. Pire, des fonctionnalités perçues comme importantes à un moment peuvent ne jamais voir le jour, alors que d'autres auxquelles personne n'avait pensé au début du projet s'avèrent être essentielles.

Il n'est pas rare d'avoir une différence entre ce à quoi l'on pensait et ce qui sera finalement développé.

Par exemple, entre 2 itérations, il est possible d'ajouter une fonctionnalité ou alors d'en modifier une existante.

Ce type de projet présente une grande différence entre le produit initialement pensé et le produit finalement conçu.

Les raisons de ces différences peuvent être multiples. Il y a, par exemple, la modification des besoins car il y a eu modification de l'environnement ou encore des demandes, utilisations ou retours des utilisateurs importants sur certaines fonctionnalités.

Un des points forts de l'Agilité est justement de pouvoir évoluer plus facilement que le cycle en V, particulièrement si l'on ne sait pas exactement ce qui est attendu, souhaité ou même apprécié.

- Présenté/Utilisé par des clients faisant des retours qui impactent la suite de son développement.

En Agilité, on n'attend pas que l'ensemble des développements soit achevé pour permettre aux utilisateurs finaux de travailler ou d'utiliser le logiciel. Une bonne pratique est de définir un « produit minimal » regroupant une partie des principales fonctionnalités. Ce « produit minimal » est alors considéré comme le produit utilisable regroupant le moins de fonctionnalités… Et donc la première version du produit utilisable par des clients.

Donner accès à cette version permet d'avoir rapidement des retours concernant l'accueil du produit, son utilisation et les demandes d'évolutions.

Les clients/utilisateurs agissent donc directement sur le produit et ses évolutions.

Certaines méthodes agiles proposent même des présentations du produit avant la finalisation de « produit minimal ». C'est le cas de la méthode agile la plus connue qui est la méthode Scrum. Dans ce cas, le produit est présenté de façon régulière, ce qui permet d'avoir des retours des futurs utilisateurs au plus tôt, de savoir rapidement si le produit correspond effectivement au besoin ou s'il faut modifier certaines fonctionnalités ou même orienter les développements dans une direction différente.

2- La construction itérative d'un logiciel.

La construction d'un logiciel dans les méthodes agiles est itérative.

Le logiciel est construit brique par brique, les briques représentant les fonctionnalités, comme vu précédemment.

Chaque itération peut regrouper entre une et plusieurs fonctionnalités, ces itérations pouvant être plus ou moins longues.

La durée des itérations varie énormément selon les méthodes agiles et les outils disponibles sur le projet. Cela peut varier de une itération avec plusieurs fonctionnalités toutes les 2 à 4 semaines (méthode Scrum) à une itération à intervalles irréguliers car itéré dès qu'une fonctionnalité est prête à être déployée. Ces intervalles peuvent être inférieurs à la minute grâce à l'utilisation d'outils permettant le déploiement continu.

Dans tous les cas, chaque itération a pour produit livrable un produit utilisable. Attention, « utilisable » ne veut pas forcément dire mis en production. Utilisable veut dire que les fonctionnalités développées depuis le début du projet sont bien implémentées et se comportent comme prévu.

I.6- Organisation d'une équipe de test

par Marc Hage Chahine

Qui dit Agilité dit équipes agiles… Et donc équipe pluridisciplinaire. Le concept d'équipe de test n'a donc pas, au premier abord, sa place dans les projets agiles.

Cela reste évidemment de la théorie pure et il est toujours bon, dans une entreprise ayant plusieurs testeurs, de pouvoir avoir une communauté, une équipe dédiée au test. De plus, un grand nombre d'entreprises éditrices de logiciels fonctionnait (et fonctionne toujours) sur des cycles de développement plus classiques, comme le cycle en V. De même, il n'est pas rare que, dans une même entreprise, il y ait des projets agiles et des projets en cycle en V.

1- Le rôle d'une équipe de test.

Les équipes, tout comme les testeurs, doivent donc s'adapter à ces changements afin de continuer à être efficaces et apporter de la valeur à l'entreprise.
Afin de savoir comment une équipe de test « agile » ou une équipe de test avec des projets agiles et des projets en cycle en V s'organise, il faut d'abord revenir au rôle d'une équipe de test. Ses rôles sont multiples :

a. Synchroniser les activités de test.
b. Gérer les plannings de test des projets.
c. Mettre en place des processus de test.
d. Mettre en place des outils communs de test.
e. Permettre aux membres de l'équipe de partager leurs expériences et leur savoir.
f. Travailler sur des projets communs ayant pour but d'améliorer les processus et les outils de test.

En détaillant ces rôles, on s'aperçoit qu'il y a des contradictions avec l'Agilité « pure ». En effet, voici des remarques que l'on peut se faire sur chacun des points précédents :

a. Synchroniser les activités de test.

En Agilité, les activités de tests sont déjà synchronisées dans chaque équipe agile. Il n'y a pas besoin d'avoir plus de synchronisation, chaque produit agile étant indépendant.

De même, dans le cas où les équipes agiles ont des dépendances entre elles (avec du SAFe ou du Scrum of Scrum), ces dépendances doivent être gérées grâce à la priorisation du backlog, des fonctionnalités à développer.

b. Gérer les plannings de test des projets.

En Agilité, le concept de planning est très vague car les itérations sont courtes et il est impossible sur plusieurs mois de savoir exactement ce qui devra être développé. En effet, le développement étant axé sur le produit ainsi que sur le retour des (futurs) utilisateurs et/ou des clients, le périmètre évolue constamment. Il est bon de noter que le seul périmètre stable est ce qui est en cours de développement. En Scrum, ce périmètre stable est en fait le contenu du sprint en cours.


DANS UNE ÉQUIPE AGILE, IL FAUT ÊTRE PLURIDISCIPLINAIRE ET SAVOIR INVERSER LES RÔLES
AUJOURD'HUI, TU NE VAS PAS FAIRE TESTEUR, TU VAS FAIRE LE BUG...
ON M'A PAS APPRIS ÇA DANS MON ÉCOLE...
F. GINTE

c. Mettre en place des processus de test.

Sur le papier, cela ne devrait pas être centralisé, chaque équipe agile devant adopter le processus qui correspond le mieux à ses besoins.

d. Mettre en place des outils communs de test.

Comme pour les processus, l'équipe agile est censée savoir ce qui lui correspond le mieux et donc quels sont les outils qui lui seront le plus utiles.

e. Permettre aux membres de l'équipe de partager leurs expériences et leur savoir.

L'ensemble de l'équipe est responsable de la qualité de l'application, ce sont donc les membres de l'équipe agile qui échangent et partagent leurs expériences. Le (ou les) testeur(s) de l'équipe sont évidemment le(s) membre(s) de l'équipe qui connaissent le mieux le sujet.

f. Travailler sur des projets communs ayant pour but d'améliorer les processus et les outils de test.

Un testeur dans une équipe agile est un membre de l'équipe agile à part entière. Il doit donc y consacrer 100% de son temps.

L'équipe de test, qui s'avère devenir une équipe transverse, doit savoir s'adapter aux remarques ci-dessus mais aussi répondre à certaines interrogations.

a. Synchroniser les activités de test.

L'équipe de test perd en effet une grande partie de son utilité sur ce point.

b. Gérer les plannings de test des projets.

Comme pour la synchronisation des activités de tests, l'équipe de test n'agit que peu, voire pas du tout, sur les plannings de test de projets. L'équipe de test peut néanmoins, de manière ponctuelle, proposer un élément de son équipe afin d'aider une équipe agile si besoin.

c. Mettre en place des processus de test.

L'équipe de test peut proposer des processus communs aux équipes ainsi que des lignes directrices à suivre dans les équipes agiles. Cela réduit, en effet, la liberté des équipes agiles. Cela permet néanmoins d'assurer que la stratégie de test est bien suivie et permet également une facilitation des montées en compétence de n'importe quel testeur de l'équipe de test intégrant une équipe agile.

d. Mettre en place des outils communs de test.

Tout comme les processus de test, proposer des outils communs permet à l'équipe de test de proposer aux équipes agiles des outils fonctionnels, maîtrisés par les testeurs et donc avec du support. Cela permet aussi de centraliser des données et donc de partager/communiquer plus facilement.
Une équipe agile pourra néanmoins utiliser d'autres outils supplémentaires, tout comme l'équipe de test peut proposer plusieurs outils différents pour une même activité (ex : UFT et Sélénium pour de l'automatisation des tests d'interface graphique).

e. Permettre aux membres de l'équipe de partager leurs expériences et leur savoir.

En termes de savoir, les équipes agiles ne peuvent pas remplacer toute une équipe de test. Les testeurs font régulièrement face à des difficultés et avoir une équipe de test avec des testeurs expérimentés permet de répondre à ces problèmes du quotidien de manière plus rapide et plus efficace.

f. Travailler sur des projets communs ayant pour but d'améliorer les processus et les outils de test.

L'amélioration des processus et le retour d'expérience sur les bonnes pratiques sont essentiels. Si une équipe agile améliore un outil, cela ne sert à rien qu'une autre équipe aux besoins identiques recrée un outil.
Les testeurs peuvent remonter les besoins, les difficultés et travailler ensemble afin d'améliorer tous ces processus communs. Ces projets bénéficieront à l'ensemble des équipes agiles.

Conclusion :

Une équipe de test avec des testeurs dispersés dans des équipes agiles voit ses responsabilités diminuées. Néanmoins, il reste certains points importants qu'elle peut encore améliorer. Ces points sont la mise en place d'outils et de processus communs, les échanges et l'entraide entre testeurs ainsi que l'optimisation, la mise en place et l'amélioration des outils et processus existant.

Sur l'ensemble de ces points transverses, l'équipe de test peut être d'un grand secours. Mais pour cela elle doit faire face au défi de la dispersion et également faire quelques entorses aux règles théoriques de l'Agilité.

2. Avoir une équipe de test transverse.

Il existe de nombreuses organisations avec des projets agiles et une équipe de test. Certaines méthodes ou organisations agiles ont d'ailleurs bien compris la plus-value d'une équipe transverse de test (ou de toute autre équipe). On peut entendre par exemple parler de « tribus » dans des entreprises agiles, ces « tribus » ont le même rôle qu'une équipe de test transverse.

Nous avons vu les rôles que l'équipe transverse peut avoir dans des organisations agiles ou ayant une partie de ses produit développée de manière agile.

Ces rôles sont principalement :

a. Mettre en place des processus de test communs afin de s'assurer que tout testeur changeant d'équipe agile (ou venant apporter son aide) puisse être efficace le plus rapidement possible. Ou encore s'assurer que la stratégie de test (au niveau entreprise) est bien suivie.
b. Mettre en place des outils de test communs et garantir une expertise de l'équipe sur ces produits.

c. Permettre aux membres de cette équipe d'échanger et de partager leurs expériences. Afin d'optimiser les résultats de l'ensemble des équipes, le savoir doit être partagé.
d. Travailler sur des projets communs ayant pour but d'améliorer les processus et les outils de test.

On peut également ajouter qu'avoir une équipe de test transverse peut permettre **d'assurer une meilleure indépendance des tests**. En proposant, par exemple, des sessions de tests exploratoires (après l'exécution des tests de régression et de validation) à des membres de l'équipe de test qui ne font pas partie de l'équipe agile.

Enfin, l'équipe de test transverse, comme nous l'avons précisé dans les deux premiers points énoncés, peut permettre une **meilleure gestion du turnover**. Le testeur arrivant dans l'équipe agile ne devant « que » monter en compétence sur sa connaissance du produit et non sur le produit, les processus et les outils.

Afin qu'une équipe comme celle-ci voie le jour et qu'elle soit efficace, il faut mettre en place une organisation spécifique pour permettre à ces testeurs de sentir qu'ils appartiennent tout autant à l'équipe agile qu'à l'équipe de test. Les testeurs font ainsi partie de 2 équipes ! Pour cela, l'équipe de test va devoir faire face à différents défis :

- Le premier défi est justement d'arriver à avoir une équipe de testeurs considérée par les membres de l'équipe agile comme appartenant à leur équipe.

Pour parvenir à relever ce défi, il faut d'abord sortir du mode agile théorique et permettre aux testeurs de consacrer une partie de leur temps à l'équipe de test, ce qui revient à dire qu'ils ne seront plus à 100% dans leur équipe agile. Cela peut paraitre un handicap pour l'équipe agile mais, à moyen terme, le retour sur investissement est évident avec la proposition de processus, du support sur des outils et surtout des solutions pragmatiques à des problèmes rencontrés par l'équipe agile.

Consacrer du temps à l'équipe de test n'est pas suffisant. Pour développer un sentiment d'appartenance, il faut également que les testeurs travaillent ensemble, aient un ou des buts communs. Pour cela, rien de tel que des projets transverses (comme le travail sur l'amélioration des processus et des outils) mais aussi une disponibilité de l'ensemble des membres de l'équipe. Que chaque membre soit prêt à aider son équipier, à lui apporter ses conseils et à partager son expérience.

- Un autre défi est d'optimiser le partage des connaissances.

Dans toute équipe, et encore plus dans les équipes dispersées, le problème du partage des connaissances est un point clé.

Avoir des connaissances est un bon point, mais si ces connaissances ne sont détenues que par une personne, lors de son absence (départ, vacances, maladie…) ces connaissances sont indisponibles voire perdues. L'équipe de test transverse doit particulièrement faire attention à cela en proposant un espace de partage avec de la documentation à jour, une matrice de

connaissance permettant de détecter les points à risque et donc une formation des membres de l'équipe sur ces points. L'équipe de test transverse peut également proposer des ateliers et des conférences afin de présenter des outils, des méthodes, des processus mis en place dans leurs équipes agiles.

- Il est également important de ne pas être considéré par les équipes agiles comme une nuisance !

Ce point est très important. L'équipe de test, si elle est acceptée par les testeurs, ne l'est pas forcément par les équipes agiles. Il faut faire attention à ne pas être trop chronophage. Afin de limiter ces problèmes, il est possible de jouer sur trois leviers.

Le premier est d'avoir un temps dédié à l'équipe de test limité (entre 10 et 20%) mais aussi de permettre au testeur de l'équipe agile de pouvoir être absent à certains moments car l'équipe agile a impérativement besoin du testeur à ce moment-là.

Le deuxième est de communiquer sur ce que fait l'équipe de test et ce que cela apporte aux équipes agiles.

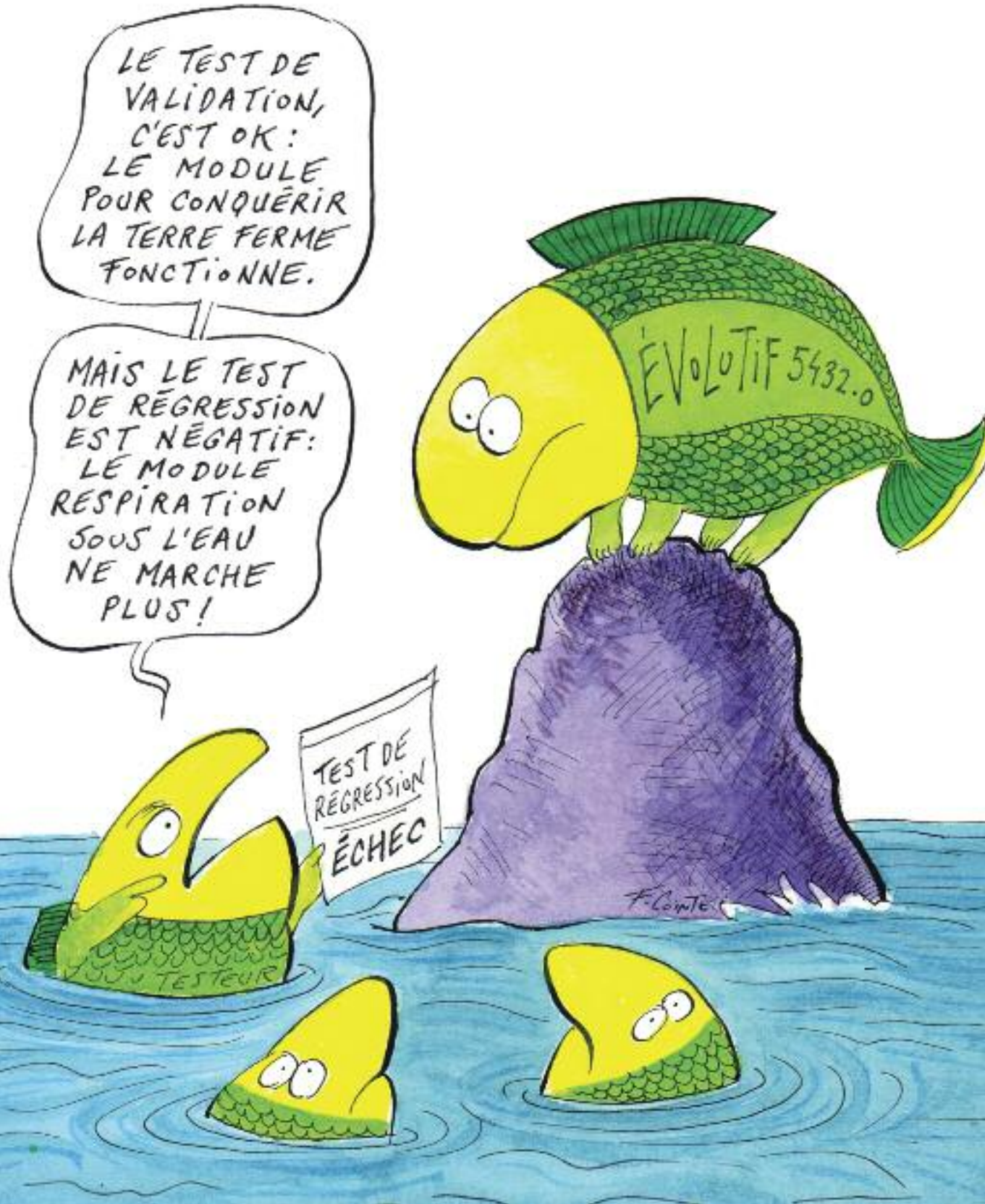
Enfin, le troisième levier est d'aller plus loin que la communication et de permettre à chaque équipe agile de « disposer » de plusieurs testeurs d'autres équipes afin d'organiser des campagnes de tests exploratoires ou simplement de les aider sur de l'exécution de test, pour pouvoir livrer au moment souhaité.

Conclusion :

Une équipe de test transverse doit être une équipe ! Les membres de cette équipe doivent pouvoir compter les uns sur les autres. Malheureusement, si cela est nécessaire, ce n'est pas suffisant car cette équipe doit « prouver » son intérêt. Pour cela, elle doit adopter une bonne communication afin de faire la démonstration de ses résultats, mais aussi savoir s'adapter à différents contextes en permettant aux équipes agiles de s'appuyer sur toutes les compétences et les connaissances de l'équipe de test.

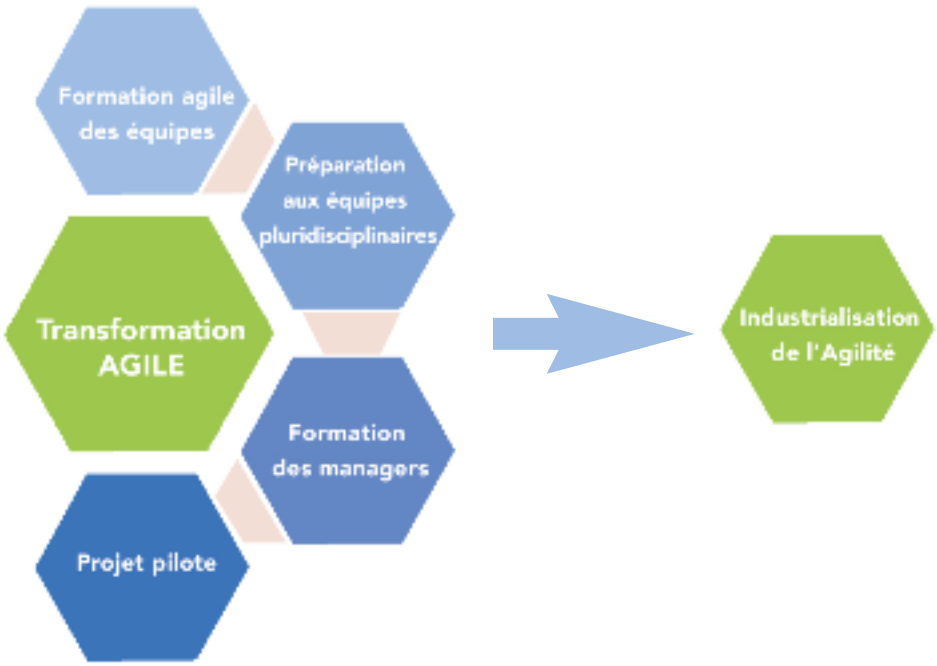
I.7- La transformation agile

par Marc Hage Chahine



La transformation agile n'est pas un long fleuve tranquille !

Pour réussir une transformation agile, il faut l'anticiper et travailler sur plusieurs points. Ces points sont :



1- La formation des futures équipes aux méthodes et à la philosophie agiles.

La première chose à faire est de s'assurer que les membres des futures équipes agiles soient formés à la méthode qui sera utilisée mais surtout à la philosophie agile.

Comme expliqué dans le chapitre sur le rôle du testeur agile, un membre d'une équipe agile, et même toute personne travaillant avec cette équipe, doit être capable de penser en produit. Le mode de pensée en projet avec des engagements précis à long terme sont contreproductifs en Agile. Ils sont même une raison fréquente de l'échec de ces changements.

La formation ne se fait malheureusement pas uniquement avec des certifications. Faire des ateliers agiles avec les membres de la future équipe est complémentaire et au moins aussi efficace. L'atelier « le jardinier agile », par exemple, est un atelier très efficace pour comprendre la vision produit.

Enfin, la formation à elle seule ne suffit pas : il faut non seulement comprendre « l'état d'esprit » nécessaire à la réussite des projets en mode agile, mais aussi adhérer à cette philosophie, ce qui inclut une préparation.

2- La préparation des futurs membres des équipes agiles à travailler dans des équipes pluridisciplinaires.

Savoir comment fonctionne l'Agilité, avoir intégré ses principes et sa philosophie n'est malheureusement pas suffisant (même si cela reste un point très important). Les futurs membres de l'équipe (ce qui inclut les testeurs) vont devoir se préparer à travailler dans une équipe pluridisciplinaire… ce qui revient à dire qu'ils vont devoir se préparer à ne pas travailler uniquement sur leur spécialité mais également sur ce pour quoi ils sont le plus utiles à l'équipe.

Les développeurs vont sûrement faire du test, les testeurs des spécifications et/ou du développement…

De même, les membres de l'équipe vont devoir s'habituer à converser directement avec des personnes qui ne font pas le même métier qu'eux (les silos entre ces les différentes équipes ne seront plus présents).

Pour assurer une bonne cohésion dans cette équipe, il faudra développer des compétences en « T », c'est-à-dire que chaque membre devra être capable de faire des tâches basiques normalement dévolues à d'autres métier (un développeur : exécuter des tests, un testeur : écrire une spécification ou développer des fonctionnalités basiques, etc.). Cette montée en compétence permettra de mieux comprendre les contraintes et besoins des autres métiers mais facilitera également la communication, chaque membre ayant une idée plus précise des problématiques techniques liées au métier de chacun.

Enfin, cette préparation théorique est rarement suffisante pour créer un esprit d'équipe. La mise en place d'ateliers ou de moments d'échanges permettant aux futurs membres de l'équipe de se connaître, de se comprendre et de se faire confiance, est toujours un plus non négligeable.

3- La formation des managers à ces méthodes et à l'évolution de leur rôle.

Cette partie est trop souvent oubliée. Elle reste cependant essentielle. Le manque de préparation/formation des managers est une raison courante de l'échec de la mise en place des projets agiles. Voici des questions courantes que les managers peuvent se poser :

- Comment puis-je gérer une équipe pluridisciplinaire ?
- Comment m'engager sur des résultats quand c'est l'équipe qui doit décider elle-même ?
- Comment rendre des comptes sur l'avancée du projet ?
- Quelle est ma place dans cette organisation ?
- Qu'attend-on de moi ? Suis-je encore utile ?

Toutes ces questions sont légitimes et traduisent une non-adaptation du manager à l'environnement agile. Le manager non accompagné dans la transformation agile va continuer à penser et travailler comme il le faisait avec les méthodes traditionnelles. Il va donc vouloir avoir une activité de suivi engageante, vouloir montrer des résultats… Bref, il continuera de penser en projet. Or, l'Agilité, c'est la fin de la pensée « projet » et l'avènement de la pensée « produit ».

Un manager/un chef d'équipe voit son rôle évoluer. Il doit donc lui aussi évoluer dans sa manière d'agir mais surtout dans sa manière de penser. Le chef d'équipe doit se muer en facilitateur, permettre aux équipes de travailler dans les meilleures conditions en apportant à l'équipe écoute, aide et conseils. On parle généralement de « servant leader », c'est-à-dire quelqu'un qui est là pour aider, appuyer, encourager son/ses équipe(s)… Ce rôle peut être comparé à celui d'une enzyme dans une réaction chimique. L'état initial et l'état final sont (quasiment) les mêmes, l'enzyme a en revanche permis à la réaction d'aller beaucoup plus vite.

De plus, chaque « chef d'équipe » va se retrouver avec « ses » ressources dispersées dans diverses équipes agiles. Les chefs d'équipe devront donc se préparer à voir leur travail au jour le jour évoluer fortement.

Prenons l'exemple d'un chef de projet test (test manager). Ce dernier ne s'occupe plus (ou très peu) de diverses activités comme :

- Estimer le coût d'un projet : c'est le testeur agile qui estime le coût des User Story.
- Travailler sur la « stratégie de test » (niveau projet) : c'est le testeur agile qui prend cette responsabilité avec l'équipe agile. Il est même possible de faire le rapprochement entre Definition of Done et stratégie de test.
- La coordination des activités de test : là encore, c'est le testeur agile qui a ce rôle dans une équipe agile.
- Contribuer à améliorer les processus de test : dans une équipe agile c'est l'équipe, à travers les rétrospectives, qui effectue ce travail. Néanmoins, le chef de projet test peut proposer des processus communs, voire « clé en main ». Ces processus peuvent en revanche être adaptés par chaque équipe en fonction des besoins spécifiques, l'important étant de garder une base commune.
- Evaluer la qualité du produit : c'est là aussi l'équipe agile qui est responsable de la qualité. Elle s'appuie sur le testeur agile pour déterminer et communiquer cette qualité.

Le chef de projet test ne se retrouve pas au chômage, loin de là ! Il a des responsabilités qu'il avait déjà et d'autres qui apparaissent avec cette nouvelle méthode de travail. On peut citer par exemple :

- Travailler à garder le sentiment d'appartenance des testeurs agiles à l'équipe test (par exemple en proposant des projets transverses).
- Travailler à avoir une « base commune » aux testeurs dans les différentes équipes agiles (outils, documents types, élaboration de la stratégie, processus, etc.).
- Gérer les ressources : il est plus compliqué de gérer 15 fois un testeur plutôt qu'une équipe de 15 testeurs.
- Assurer la montée en compétences et le partage de connaissances/retours d'expérience des différents membres de l'équipe.

En Agilité, j'aime parler de « rôle » du chef de projet test. Avec les équipes agiles, le chef de projet test se retrouve dans un environnement différent. Dans cet environnement, il est entouré de « mini-chefs de projets test » (les testeurs agiles). Son rôle s'oriente alors légèrement vers plus de coordinations et d'aide au partage de connaissances, des bonnes pratiques et des retours d'expérience.

Le métier de chef d'équipe a donc toujours de beaux jours devant lui (même en agile !), et, même si les membres de l'équipe agile prennent une partie du rôle qui leur était attribué dans les méthodologies classiques, les chef d'équipes/managers restent très importants.

4- Implémenter l'Agilité au fur et à mesure en proposant un projet pilote.

Cette partie est constamment rappelée lorsque l'on parle de transformation agile. Une transformation « Big Bang » est rarement une réussite. Si l'on veut réussir une transformation, il faut savoir avancer étape par étape. Il faut d'abord former. Il faut ensuite s'attacher à convaincre. Pour comprendre cela, rien ne vaut un exemple.

Il est préférable, pour démarrer dans une méthode agile, de prendre un projet « moyen ». Ici, « moyen » signifie que le projet garde un degré d'importance non négligeable mais qu'il n'est pas trop important en terme d'investissement.

Le « projet » se transforme alors en POC (Proof Of Concept), en prototype de l'Agilité pour l'entreprise et permet d'acquérir de l'expérience, d'essuyer les plâtres et de permettre une transformation plus aisée des autres équipes.
Il faut également se rappeler que la synchronisation entre une équipe agile et des équipes fonctionnant encore en cycle en V se révèle souvent compliquée. Je pense notamment à des problèmes comme :

- L'équipe du cycle en V qui n'arrive pas avoir la modification immédiate demandée car on est en cours de sprint (pour la méthode Scrum).
- L'équipe agile qui se plaint de ne pas pouvoir faire de démo car elle est en attente de la livraison d'une fonctionnalité qui a pris du retard dans un projet en V.
- L'équipe agile qui ne peut pas livrer rapidement car elle doit attendre des « slots » trimestriels de livraison…

Afin de limiter ces effets, il est préférable d'avoir un projet qui est le plus indépendant possible.

Un autre axe essentiel dans la sélection du projet/produit pilote est la sélection des membres de l'équipe associée à ce produit. Cet axe, qui est trop souvent ignoré, est peut-être le plus important. C'est l'axe humain : choisir les bonnes personnes au bon moment.

Le Manifeste Agile ci-dessous a été créé par des développeurs expérimentés :

- Les individus et leurs interactions plutôt que les processus et les outils.
- Des logiciels opérationnels plutôt qu'une documentation exhaustive.

- La collaboration avec les clients plutôt que la négociation contractuelle.
- L'adaptation au changement plutôt que le suivi d'un plan.

Comme l'écrit le Manifeste : les individus, les logiciels opérationnels, la collaboration et l'adaptation sont mis en avant.

Néanmoins, les processus et outils, la documentation, les contrats et les plans ne sont pas non plus oubliés et restent importants.

Afin de réussir à suivre efficacement ce Manifeste, il faut des personnes ayant assez d'expérience pour connaitre l'intérêt de chacun de ces points et pour savoir jusqu'à quel niveau il faut utiliser des processus, outils, documentation…

Une équipe agile pour un projet pilote se doit donc d'être une équipe composée de personnes expérimentées et matures. L'idée de mettre uniquement des ingénieurs fraîchement sortis de l'école pour commencer dans l'Agilité s'avère (quasiment) toujours être une erreur fatale au produit/projet.

Le projet pilote ne peut fonctionner qu'en choisissant un projet et une équipe appropriée.

5- L'industrialisation de l'Agilité.

Après avoir franchi toutes les étapes précédentes (formation de l'équipe et du manager, préparation des équipes (état d'esprit) et réalisation d'un projet pilote), il est alors temps d'industrialiser !

L'industrialisation ne peut se décréter et se faire comme en mode Big Bang et ce, même après avoir bien suivi le « mode d'emploi » donné dans le reste de ce chapitre. Les quatre premières étapes, bien que nécessaires, ne sont que le début d'un long processus de transformation qui peut prendre des mois voire des années, en fonction de la taille des équipes.

Le projet pilote et plus particulièrement les membres de l'équipe pilote, ont un rôle primordial. Ils doivent notamment permettre de repérer certaines difficultés et de résoudre les causes racines de ces problèmes. L'équipe pilote doit ensuite proposer un retour d'expérience continu.

Son rôle ne s'arrête pas à proposer ce retour d'expérience et à « essuyer les plâtres ». En effet, l'équipe pilote a au moins une autre responsabilité tout aussi importante : celle de répandre la « bonne parole », d'accompagner les nouvelles équipes dans la transformation et de leur faire partager leur expérience, tout en les laissant apprendre par elles-mêmes.
Cette responsabilité montre encore l'importance de bien sélectionner les personnes qui intègreront cette équipe.

Attention, l'équipe pilote seule ne peut rien. Afin de réussir une transformation agile, il faut aussi avoir un vrai soutien de la hiérarchie, des managers. Hiérarchie qui doit donc être prête à s'adapter à ses nouveaux rôles.

Une méthode efficace pour généraliser l'Agilité est d'intégrer des personnes dans l'équipe pilote et d'en faire des membres de l'équipe pendant quelques mois. Ces membres intégrés (ou simplement membre originels de l'équipe pilote) pourront ensuite être des référents dans de nouvelles équipes agiles et faire ainsi comprendre l'état d'esprit attendu par l'Agilité.

Conclusion :

Être agile ne se décrète pas. Une transformation agile réussie n'est pas simple et il existe de très nombreuses équipes nommées « agiles » qui, dans les faits, ne le sont pas du tout.

Afin d'être agile, il faut avoir le bon état d'esprit, changer de manière de penser. Pour cela, la formation, la preuve par l'exemple mais aussi un vrai travail de fond sont nécessaires. Ce travail se fait à moyen, voire à long terme et reste primordial. A défaut, on se retrouverait dans une méthodologie de travail reprenant les défauts des méthodes classiques et agiles, sans pour autant bénéficier d'une seule de leurs qualités.

PARTIE II
PRATIQUES DES TESTS EN AGILE


Hiiiiii
GRAOU
AAÂÂ
POUR LES TESTS EN CHAOS ENGINEERING, C'EST PLUS DRÔLE DE LÂCHER LES BUGS ET LES MONKEYS APRÈS AVOIR DÉBRANCHÉ TOUTES LES LUMIÈRES...
CHAOS GAME MASTER
AAÎÎÎH

II.1- Les tests d'acceptation en Agile.

par Reynald Stevens

1- Les nouveaux challenges du développement numérique.

a. Les nouveaux challenges côté utilisateurs.

Les utilisateurs du web sont aujourd'hui marqués par un usage cross-devices, c'est-à-dire par des usages alternés d'un smartphone, d'un ordinateur et éventuellement d'une tablette. En 2016, plus de 50% des transactions e-commerce étaient réalisées après un processus d'achat effectué sur deux terminaux ou plus[1]. 20% des achats sur ordinateur ont débuté par une recherche sur smartphone et 35% pour la réciproque.

Ce comportement pose de nombreux défis pour la conception d'un service numérique, notamment pour la portabilité du produit d'une plateforme à une autre.

La fragmentation du marché engendre des risques de portabilité.

Le secteur du digital avance vite, avec des millions de terminaux en constant développement qui n'ont pas tous les mêmes caractéristiques : format, résolution, OS (Windows, Android, iOS, Mac OS, Linux), version d'OS, mémoire... La fragmentation est telle qu'elle engendre des risques de portabilité, cette dernière étant la capacité d'un produit digital (sites web, Apps, IoT) à s'adapter à son environnement d'utilisation.

Ainsi, la fragmentation abyssale du marché va engendrer pour l'utilisateur des risques d'affichage, de performance, de connexion, d'ergonomie avec des contenus (texte, image, vidéo) qui ne s'affichent pas correctement selon les spécificités de chaque terminal.

Pourquoi ? Parce que le produit numérique n'a pas forcément été pensé et conçu pour être vu sur autant de devices. Tous ces risques doivent être maîtrisés en amont par les entreprises car le challenge est de proposer aux utilisateurs une solution numérique (application, site web) compatible sur le plus large panel de terminaux possible.

L'expérience utilisateur est centrale dans le succès d'un produit numérique.

Aujourd'hui, un produit qui fonctionne n'est pas suffisant. Il est crucial que l'utilisateur comprenne son fonctionnement au premier coup d'œil et prenne du plaisir à naviguer sur celui-ci. Diverses compétences se sont rapidement ajoutées aux équipes de développement pour s'assurer du succès de l'UX dans la conception de l'UI, notamment au travers de l'approche du Design Thinking. Ainsi, un produit numérique doit être :

- UTILE : il répond à un besoin business et à un besoin utilisateur. Il répond aux attentes de ce dernier (information précise, achat) et offre les fonctionnalités adéquates.
- ERGONOMIQUE : l'utilisateur doit pouvoir réaliser l'action sans effort.
- ENGAGEANT : l'utilisateur doit être impliqué dans sa navigation.

[1] : *Criteo, Rapport d'activité sur le e-commerce, 2ème semestre 2016*

Pourquoi un tel engagement dans la phase de conception ?

Prenons l'exemple du marché des applications mobiles. Le premier mois, 36% des utilisateurs utilisent une application mobile, le troisième mois, le taux de churn atteint 80% et 24% des utilisateurs abandonnent l'application après la première utilisation.

Les taux de churn sont devenus aujourd'hui extrêmement élevés, pour des coûts d'acquisition qui le sont tout autant.

Par conséquent, il est déterminant de s'assurer que le produit conçu correspond aux véritables attentes des futurs utilisateurs.

C'est une des raisons majeures pour laquelle est né le principe de l'Agilité et l'inclusion des tests d'acceptation dans celle-ci. Ne pas hésiter à pivoter rapidement, même pendant les phases de conception.

b. Les nouveaux challenges côté développeurs.

On le comprend, la manière de concevoir puis de maintenir un produit numérique s'adapte forcément à ces nouveaux challenges et les développeurs ont eux-mêmes leurs propres challenges. Ils doivent s'adapter à un environnement qui demande une plus grande rapidité d'exécution, de la souplesse et de l'amélioration continue.

Pendant longtemps, les développements ont été réalisés de manière méthodique en suivant le principe de la Cascade (Waterfall). L'eau d'une chute, une fois qu'elle a dévalé le flanc de la montagne, ne peut plus remonter. Dans le cadre d'un projet, dès qu'une étape est terminée, il n'y a pas de retour en arrière.

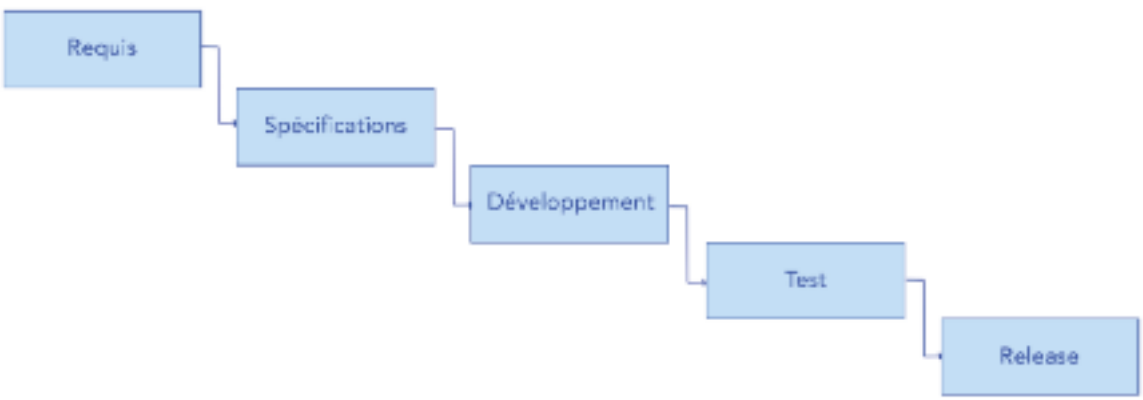


Figure : 1 : exemple de projet en Cascade

En gardant à l'esprit ce qui a été dit précédemment, il est clair que cette méthodologie peut conduire à des écarts entre la conception et les attentes du marché.

C'est pour cette raison que les organisations des développeurs ont évolué vers les cycles en V puis l'Agilité.

Les développeurs doivent aujourd'hui composer avec le changement en cours de projet.

La méthode agile repose sur le principe de l'**adaptation au changement** : elle offre une flexibilité d'intégration des spécifications tout au long du projet grâce au développement en sprints et permet de faire évoluer le projet dans des directions différentes selon les exigences ou problématiques rencontrées.

De manière synthétique, les développeurs doivent s'adapter à deux évolutions :

- Les User Story et sprints.

Les développeurs doivent travailler en sprints (itérations) définis autour de User Story (Parcours Utilisateurs). Les fonctionnalités sont développées autour de ces User Story. Le principe est d'ajouter des nouvelles fonctionnalités à chaque sprint pour faire grossir le projet.

- La transformation des équipes.

Le développeur doit apprendre à travailler au sein d'une équipe multi-compétences et autonome. Au sein de cette équipe, la place du responsable des tests est importante puisqu'il vient donner le GO pour le lancement des sprints et va éventuellement faire évoluer la suite des travaux.

c. Les nouveaux challenges du testeur.

L'assurance qualité fait ainsi partie intégrante du cycle de vie du produit.

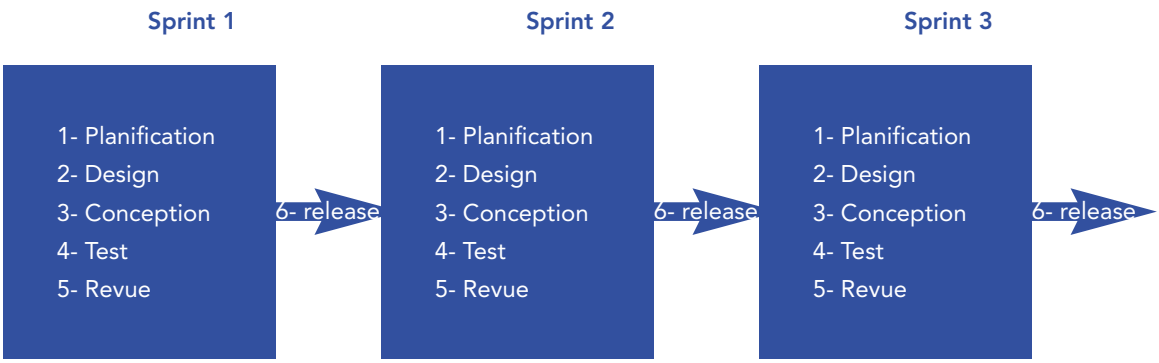


Figure : 2 : exemple de projet en Agilité

En effet, l'augmentation du rythme des mises en production induit mécaniquement :

1. le besoin de tester de nouvelles fonctions.
2. l'augmentation du risque des régressions.

A cela s'ajoute qu'en Agilité, l'organisation de la fonction du test est bouleversée. Dans les organisations traditionnelles, cette fonction est organisée autour du Test Manager et de testeurs. Le Test Manager est chargé, entre autres, de mettre en œuvre les tests de plusieurs projets, piloter les stratégies de test, gérer le budget et les plannings, piloter l'équipe de test, évaluer la qualité du produit. Le testeur est, quant à lui, chargé d'exécuter les plans de test.

Dans un projet agile, les testeurs sont intégrés aux équipes de développement. En conséquence, le rôle du Test Manager se trouve dilué, incombant souvent au testeur lui-même, qui prend donc en charge une large partie de ses responsabilités. Ou comme nous le verrons plus tard, il est dévolu à une entité à part entière.

Une collaboration étroite avec le marketing et le développement.

Ainsi, les nouveaux challenges du testeur agile vont bien au-delà de la réalisation de tests. Tout d'abord, le fonctionnement idéal en Agilité implique une collaboration qui devient de plus en plus étroite avec le marketing et le développement. Le testeur agile doit ainsi être en mesure de communiquer ses résultats au sein de son équipe et de contribuer pleinement au développement du produit et des sprints à venir.

Si la méthode offre des avantages, elle crée de nouveaux risques qu'il faut circonscrire :

1. Product Owner, développeur, testeurs : comment bien communiquer ?
L'équipe projet en Agilité est construite autour de compétences et de cultures différentes. Les tests accompagnent cette transformation managériale. En effet, les story et leurs tests vont remplacer les spécifications. Les discussions entre les membres de l'équipe vont donc s'appuyer sur ces dernières.

2. Aller au bout d'un sprint.
Le développement, le test puis la correction doivent impérativement faire partie du même sprint. Or, toute la difficulté va venir ici de la bonne planification et du respect des délais. Le testeur doit absolument être impliqué en amont puis être extrêmement réactif.

Si les prérogatives du testeur agile changent, les tests eux-mêmes et la façon de les conduire évoluent aussi. Les tests d'acceptation prennent une place importante en Agilité car ils permettent à la voix de l'utilisateur de s'exprimer pendant les développements.

2- Les tests d'acceptation en Agilité.

a. De la vérification au guide.

Quelle que soit la méthode de développement retenue ou l'organisation de travail mise en place, il y a forcément des phases de tests à prévoir. Les méthodes agiles ne changent pas l'importance de ces dernières, mais proposent simplement une nouvelle façon de les percevoir et de les intégrer.

En méthodologie classique, par exemple en V, le test a pour objectif de détecter des erreurs après le travail de développement. Dans ce cas précis, le test d'acceptation, ou recette, est une phase qui compare le fonctionnement du produit aux attentes spécifiées. Cette phase implique le déroulement d'étapes de test de manière rigoureuse pour identifier tout écart fonctionnel ou technique.

En Agilité, il y a un changement de prisme autour des tests. Les tests sont là pour guider les développements du produit. Pour s'en rendre compte, Brian Marick[2] a initié un modèle, qui a par la suite été développé par Lisa Crispin et Janet Gregory[3].

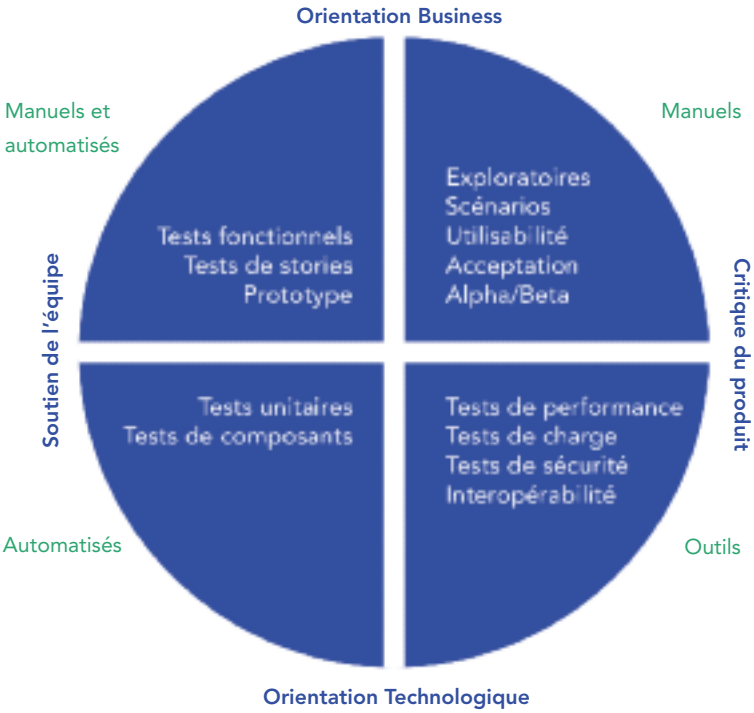


Figure : 3 : le quadrant du test agile

Au travers de cette conception, on se rend compte que les tests d'acceptation sont orientés clients (Business et critique du produit) et permettent d'accepter une story à la fin d'un sprint.

b. En pratique, comment ça marche ?

Les tests d'acceptation vont donc suivre des étapes logiques :

– Description du comportement attendu (les critères d'acceptation)
– Construction des storytests (cas de test)
– Conduite des storytests

Comme nous l'avons vu plus haut, les critères d'acceptation viennent valider une story et permettent de faire avancer le projet, notamment par la présentation d'une démonstration en fin de sprint. Leur « bonne » écriture est donc cruciale et c'est fréquemment un exercice collectif supervisé par le Product Owner. En général, il faut éviter la situation où c'est le PO ou le testeur agile qui écrivent seuls les critères d'acceptation sans concertation. En Agilité, la collaboration est essentielle au bon déroulement des sprints.

[2] : Brian Marick – Agile Manifesto
[3] : Lisa Crispin et Janet Gregory - Agile Testing: A Practical Guide for Testers and Agile Teams

A quoi ressemblent des critères d'acceptation ?

Ce peut être des User Story qui décrivent un comportement, toujours du point de vue utilisateur, souvent en suivant une structure inspirée du langage Gherkin : *Étant donné, quand, alors*.

Étant donné : l'état du produit avant l'exécution de la story.
Quand : un événement déclenche un processus.
Alors : l'état du logiciel après l'exécution.

Une approche agile et collaborative consiste à travailler sous la forme d'exemples. Dans ce dernier cas, on parle de SBE (Spécification by example) ou d'A-TDD (Acceptance Test-Driven Development). Cette approche vise clairement à guider la conception d'une fonction à travers ses tests d'acceptation. Dans le cas présent, on écrit d'abord le test, puis on développe de manière à ce qu'il n'échoue pas.

L'intérêt du SBE permet d'être très concret et d'avoir un langage compris par tous. Ainsi, on a :

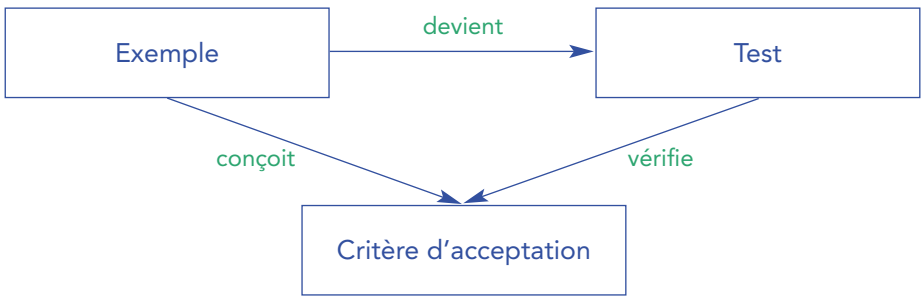


Figure : 4 : l'usage de l'exemple dans l'ATDD

L'A-TDD permet de clarifier de manière collaborative les exigences en les approfondissant grâce à une compréhension pragmatique et ce sont les tests qui permettent d'y parvenir.

c. Les avantages et les limites de l'A-TDD.

L'A-TDD permet, comme nous l'avons vu plus haut, d'exprimer un critère de validation de manière intelligible. Les tests sont vraiment écrits et ne sont pas réalisés à la légère puisqu'ils sont imaginés en début de méthode. Ils sont précis, non ambigus, reproductibles et exploitables.

L'approche par les tests permet en même temps d'écrire un cadre pour les futurs tests de non régression et donc de conduire vers l'automatisation. Or, dans un environnement agile, l'automatisation est devenue nécessaire pour pouvoir suivre le rythme des développements.

En revanche, du fait de son organisation, l'approche peut avoir des limites. D'abord, elle exige de tous les membres de l'équipe de sortir de leur zone de compétence, voire d'intérêt, pour travailler sur des cas de test. Ensuite, la limite à cette méthode concerne bien entendu les ressources : il est essentiel d'être accompagné par des experts du test, qu'ils soient internes ou externes.

3- Externaliser le test d'acceptation en Agilité.

Quelle que soit la méthode de développement retenue, si le test est indispensable, les entreprises se heurtent souvent à un principe de réalité : il faut des ressources en compétences, expériences, outils et devices (smartphones, tablettes, phablet, desktop).

C'est le prestataire qui va assumer et mutualiser sur plusieurs clients des infrastructures techniques et des compétences humaines, gourmandes en trésorerie. C'est également lui qui va maintenir les devices et OS, que ce soient les versions obsolètes comme les nouvelles versions.

a. Mieux gérer les montées en charge et avoir les bonnes compétences.

L'externalisation des tests permet donc une montée en charge rapide de l'activité QA et UAT d'une entreprise éditrice de services numériques. Vous disposez rapidement d'une équipe de testeurs pluridisciplinaire et impartiale. Il est en effet parfois complexe d'avoir toutes les compétences en interne : testeur agile, analyste, automaticien de tests et, par ailleurs, de les multiplier pour les intégrer dans plusieurs squads en cas de projets multiples. Si ces compétences font défaut, alors la qualité n'est pas au rendez-vous.

C'est la raison pour laquelle certaines entreprises privilégient l'externalisation de cette compétence pour des raisons de coûts et de maintenance.

Toutefois, l'externalisation en Agilité suppose de choisir un partenaire, plutôt qu'un prestataire. Comme nous l'avons vu plus haut, les tests en Agilité impliquent un travail collaboratif dans un esprit d'équipe. Par conséquent, le choix de la société tierce va se faire aussi sur sa capacité à travailler à distance mais en équipe, avec la capacité de fournir une relation de proximité.

Si l'impartialité est importante dans les tests pour apporter une prise de recul nécessaire, dans le cas d'un projet agile, l'équipe de test doit capitaliser sur l'expérience des itérations et parfaitement communiquer avec l'équipe interne. En conséquence, le prestataire de tests doit pouvoir mettre en place les outils et les fonctionnements permettant un suivi sans couture de son client.

b. Avoir à disposition un parc de devices représentatif.

Il est parfois impossible de disposer de l'environnement de tests, représentatif de son marché. Il est peut-être nécessaire de rappeler ici que les tests doivent se faire sur les terminaux cibles de votre marché et non sur ceux que vous avez à disposition. Or, comme nous l'avons vu précédemment, la fragmentation numérique fait qu'un service numérique doit être testé sur un grand nombre de configurations.

Appareils réels ou émulation ?

Un émulateur est une simulation informatique d'un appareil de test, tel qu'un mobile ou une tablette, qui peut être utilisé pour tester une application. Les émulateurs sont couramment utilisés lorsque les testeurs n'ont pas accès aux dispositifs de test dont ils ont réellement besoin.

Un émulateur peut simuler à la fois le matériel et le logiciel d'un périphérique, ce qui permet à une application de fonctionner sans modification sur cet émulateur. Un émulateur peut être une approche intéressante car elle peut être moins coûteuse à mettre en place. Elle permet de réaliser des tests d'interface, des tests fonctionnels, des tests d'utilisabilité et des tests de compatibilité.

Toutefois, l'émulation ne peut pas être une réponse à tout. Avec de réels devices, il est plus facile de tester une application dans des conditions réelles et ainsi de tenir compte de l'utilisation de la batterie, de la RAM et de l'espace de stockage (mémoire interne, carte SD), tout comme les interactions SMS/appel/voix et Bluetooth que les émulateurs ne peuvent pas prendre en compte.

Les testeurs avec de vrais appareils peuvent explorer une application pour mesurer les performances de chaque fonction dans une variété de conditions, afin de fournir une évaluation plus complète de l'application.

Les appareils réels facilitent l'examen de la mise en page d'une application en responsivité sur divers écrans, pour avoir une idée exacte de la façon dont l'application apparaît aux consommateurs. L'évaluation des applications sur une variété de tailles et de dimensions d'écran deviendra de plus en plus importante, au fur et à mesure que les marques expérimenteront des écrans courbés et flexibles.

De plus, les tests sur des appareils réels permettent d'évaluer la géolocalisation.

c. Les bons tests aux bons moments.

Les bénéfices d'une externalisation peuvent également résider dans la multiplicité des méthodes de tests à conduire, tout au long des itérations d'un développement de produit.

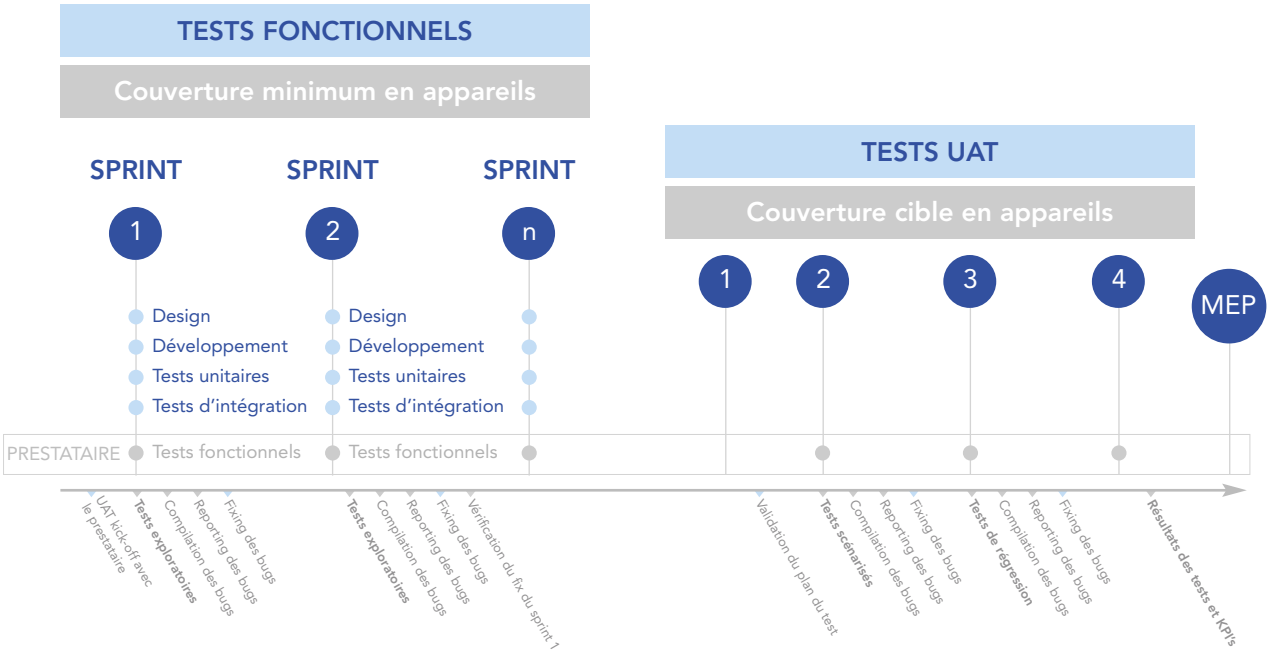


Figure : 5 : Exemple de couverture de tests au sein de sprints – Source : StarDust Testing

Le choix d'un partenaire qui couvre à la fois des tests manuels (exploratoires et scénarisés) et de l'automatisation, permet de créer une équipe agile externe. Si nous prenons l'exemple du modèle d'organisation de Spotify, le prestataire externe devient le chapitre QA de votre organisation.

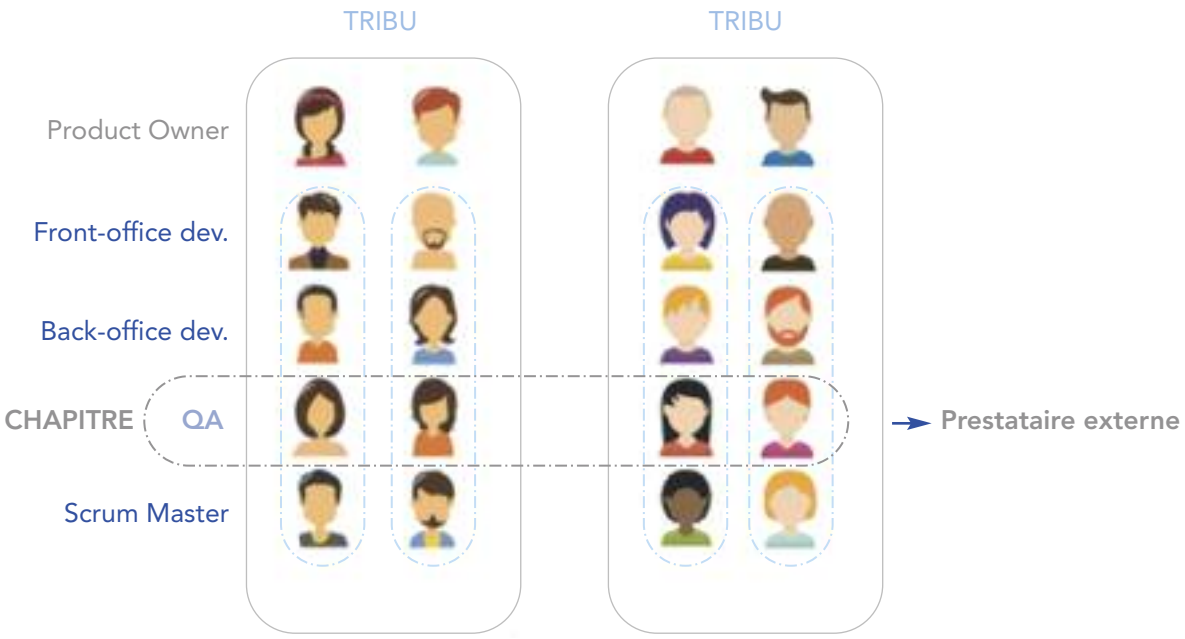


Figure : 6 : Exemple de couverture de tests au sein de sprints – Source : StarDust Testing

Le cas particulier du Crowdtesting

Le crowdtesting est une solution de tests basée sur une communauté de freelances répartis dans le monde. L'avantage de cette approche est l'effet « masse » qu'elle produit pour couvrir des territoires, des configurations et des compétences. Elle permet ainsi de gérer des problématiques de géolocalisation, de langues, de culture, de devices spécifiques. Toutefois, en Agilité, il est indispensable de s'adosser à un partenaire capable de piloter cette communauté, pour qu'elle s'intègre à vos sprints et qu'elle puisse capitaliser son expérience d'un sprint à un autre.

II.2- Comment réussir à faire collaborer le développement et les opérations dans un contexte agile ?

par Benjamin Carriou

1- Les impacts d'une organisation agile sur les opérations.

L'Agilité permet d'offrir une grande capacité d'adaptation aux imprévus et aux besoins des utilisateurs, en proposant une méthodologie de projet basée sur des cycles de développement itératifs, incrémentaux et courts qui permettent une étroite collaboration entre le développement et le métier, afin de mieux prendre en compte les besoins et retours des clients.

Cependant, l'approche agile ne se cantonnant qu'au développement et qu'au métier, cela entraînera, tôt ou tard, un ensemble d'impacts dont voici une liste non-exhaustive :

- Augmentation du nombre de livraisons.

Dans une approche agile qui amène une gestion de projet de type itérative, incrémentale et courte, il est nécessaire d'être en mesure de mettre en production un nouveau produit ou des évolutions apportées sur ce dernier dans des délais courts et de manière fréquente, pouvant aller de plusieurs fois par semaine à plusieurs fois par jour.

- Non-alignement du développement et des opérations.

Les opérations ont des besoins et contraintes antagonistes à ceux des développeurs. Les développeurs ont un besoin d'innovation forte, c'est-à-dire de pouvoir produire vite de nouvelles fonctionnalités et correctifs qui devront être livrés le plus rapidement possible en production. Or, les opérationnels ont, quant à eux, un besoin fort de stabilité sur ce qui est délivré en production et doivent donc s'assurer que ce qui est produit par le développement ne soit pas susceptible d'introduire des défaillances et régressions sur les environnements de production. Cependant, dans l'état actuel, les opérations ne sont pas en mesure de déployer rapidement un nouveau produit, une évolution ou un correctif en production. Cela s'explique souvent par le manque de confiance, parfois appelé « mur de la confusion », qui s'est instauré avec le développement. Et cela vient aussi du fait que les opérations ne disposent pas de l'outillage adéquat pour supporter cette cadence et cette charge de travail imposée par l'Agilité :

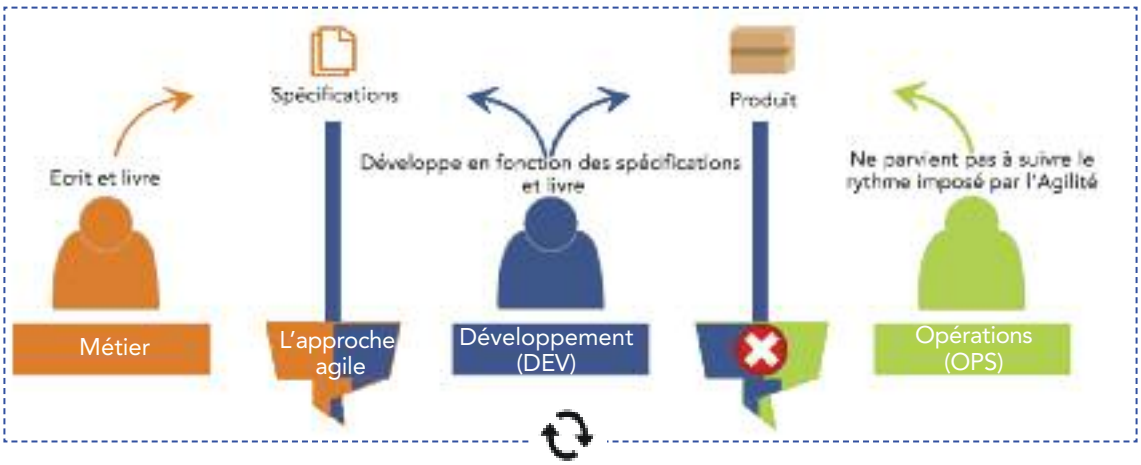


Figure 1: Non-alignement entre le développement (DEV) et les opérations (OPS)

- Manque de délégation de la part des opérations.

Du fait de ce non-alignement entre ces deux métiers, il découle un manque de délégation de la part des opérations, qui ne mettent pas à disposition des développeurs les outils pour qu'ils puissent gagner en autonomie sur des tâches qui leur sont pourtant nécessaires au quotidien. On peut, par exemple, citer le déploiement ponctuel d'environnements de test ou encore l'accès à des logs et des métriques de production, suite au déploiement d'une nouvelle version ou d'un correctif sur un produit donné.

La problématique qui se pose alors est de savoir comment réussir à faire collaborer le développement et les opérations dans un contexte agile, afin d'être en mesure de délivrer plus fréquemment et rapidement de la valeur au client sans devoir sacrifier la qualité de ce qui est produit.

2- Le DevOps, une approche de collaboration agile entre le développement et les opérations.

Au vu des impacts amenés par une organisation agile, il est nécessaire de casser les silos qui séparent le développement des opérations et c'est ce à quoi une approche dite « DevOps » tend à répondre.

En premier lieu, le DevOps permet de faire collaborer le développement (DEV) et les opérations (OPS) qui historiquement ne le faisaient pas, notamment en apportant aux opérations une vision produit afin qu'ils soient acteurs tout au long du cycle de vie de ce dernier et qu'ils participent aux décisions qui pourront les impacter, plutôt que d'être dans une vision projet où ils pourraient avoir le sentiment d'être « la cinquième roue du carrosse ». Mais il serait réducteur de ne cantonner le DevOps qu'à un « rapprochement entre les DEV et les OPS » puisqu'il s'agit avant tout d'une culture, d'un courant de pensée, d'une philosophie qu'il est nécessaire d'insuffler à une entreprise.

L'approche DevOps peut donc être considérée comme une philosophie et une approche logicielle axées sur la collaboration dont les pratiques associées permettent d'effectuer des déploiements plus fréquents et de qualité, en s'appuyant sur un outillage spécifique.

Ce qui est difficile lorsque l'on essaie d'appréhender cette approche, c'est que, contrairement à l'approche agile avec son Manifeste[1] et ses méthodes par exemple, elle ne propose pas de référentiel officiel permettant de spécifier ce qu'est cette dernière, et donc de la façon de procéder pour la mettre en œuvre. Cependant, certains tentent d'apporter des réponses afin d'être en mesure d'évaluer la maturité d'une organisation sur la mise en place de cette dernière. Il est alors possible d'utiliser, par exemple le modèle dit du « CALMS[2] » et ses cinq piliers (Culture, Automation, Lean, Measure et Share), ou encore celui du « The Three Ways[3] » et ses trois principes qui permettront d'avoir des éléments de réflexion pour identifier où l'entreprise se trouve actuellement et vers quoi elle pourrait tendre.

---

[1] : Le Manifeste Agile : https://agilemanifesto.org/iso/fr/manifesto.html
[2] : CALMS : https://devops.com/using-calms-to-assess-organizations-devops/
[3] : The Three Ways : https://itrevolution.com/the-three-ways-principles-underpinning-devops/

Du fait de cette liberté dans sa compréhension et dans sa mise en œuvre, chaque entreprise (ex : Oui.sncf[4], Blablacar[5], …) pourra donc avoir sa propre approche du DevOps, en axant davantage, pour certaines, leurs transformations sur l'aspect culturel de l'approche, tandis que d'autres préfèreront plutôt se focaliser sur l'aspect technique de cette dernière.

Tendre vers l'adoption d'une approche DevOps au sein de l'entreprise peut permettre de répondre aux impacts énoncés plus tôt et notamment :

- Redessiner les rôles du développement et des opérations.

Dans une approche DevOps, le développement et les opérations construisent une étroite collaboration basée sur la confiance, l'accompagnement et la compréhension du métier de l'autre. Il est possible par exemple, pour y parvenir, de former des équipes pluridisciplinaires pour favoriser l'échange et la collaboration, ou encore d'inviter les opérations à prendre part à l'élaboration des sprints, de participer aux daily meeting, ou encore aux rétrospectives. De plus, les opérations peuvent accompagner le développement afin qu'on obtienne un produit qui prenne en compte leurs contraintes de production en termes de sécurité, de scalabilité[6], de performance :

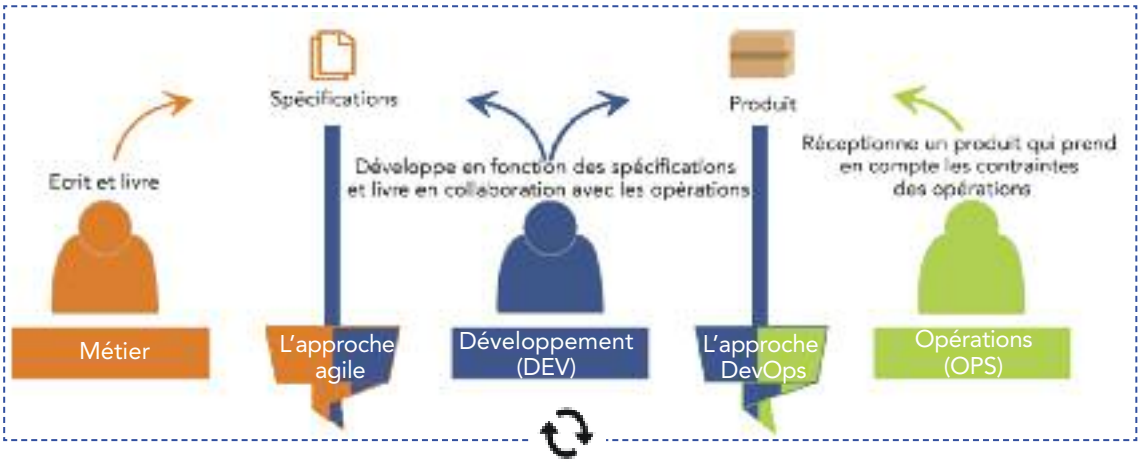


Figure 3 : Collaboration étroite entre les DEV et les OPS grâce à l'approche DevOps

- Faciliter l'autonomie des développeurs.

Le fait de redessiner les rôles du développement et des opérations va permettre de faire gagner en autonomie le développement sur certaines tâches pouvant être considérées comme à faible valeur ajoutée et qui étaient, jusqu'ici, sous la responsabilité des opérations. Cependant, la délégation de ces tâches doit se faire selon les règles et les contraintes apposées par les opérations et il est donc nécessaire qu'elles mettent à la disposition du développement les outils adéquats pour réussir cette « passe d'armes ». On peut parler ici d'un « catalogue de services » dans lequel le développement pourra venir se servir pour déployer à la demande des bases de données, des serveurs d'application, … en utilisation des solutions telles que la conteneurisation[7] avec

[4] : Oui.sncf : https://ouitalk.oui.sncf/blog/techno/devops-ou-l-agilite-de-la-production-chez-voyages-sncf-com
[5] : Blablacar : https://www.silicon.fr/blablacar-automatise-production-it-112053.html
[6] : Scalabilité : capacité d'un produit à s'adapter à un changement d'ordre de grandeur de la demande, en particulier sa capacité à maintenir ses fonctionnalités et ses performances en cas de forte demande
[7] : Conteneurisation : permet d'isoler une application avec ses dépendances et librairies sans système d'exploitation au sein d'une entité logique appelée « conteneur »

Docker[8] ou encore l'infrastructure en tant que code (IaC)[9] avec Ansible[10] pour venir masquer cette complexité et faciliter l'utilisation de ce catalogue.

• Délivrer des produits plus fréquemment et plus rapidement.

La mise en œuvre d'une approche DevOps va permettre d'avancer et d'innover plus rapidement en accélérant le rythme des livraisons, grâce à une meilleure maîtrise de la chaîne de bout en bout, de la conception du produit jusqu'à sa livraison en production. Cela est notamment possible grâce à l'automatisation des tests (qualimétrie, tests unitaires, tests d'intégrations, tests fonctionnels, tests de performance, tests de sécurité, …), au déploiement automatisé d'environnements à la volée à des fins de test et qui peuvent tendre à être iso-production, ou encore à un retour (feedback) facilité auprès des personnes concernées en cas d'erreurs.

3- L'industrialisation des tests au travers d'une usine logicielle.

Dès lors que l'on souhaite mettre en œuvre une approche DevOps en entreprise, il est essentiel d'avoir en tête au minimum ces deux axes de travail. Le premier, plutôt orienté sur l'aspect culturel de l'approche, doit permettre de traiter des éléments liés à la collaboration, l'organisation et les méthodologies de travail (cf. les piliers Culture, Lean et Share du CALMS). Le second doit, quant à lui, se positionner sur l'aspect technique de l'approche en mettant en œuvre notamment de l'industrialisation basée sur l'automatisation et la mesure (cf. les piliers Automation et Measure du CALMS) ; cet aspect va être détaillé par la suite.

L'industrialisation va consister à mettre en place ce qu'on appelle une « usine logicielle » qui s'inspire fortement de ce qui se fait dans les usines industrielles où l'organisation est découpée en chaînes de production et où les tâches répétitives sont automatisées et contrôlées. Ainsi, si une tâche ne retourne pas le résultat escompté (par exemple, un résultat de test en échec), la chaîne de production s'arrête et ne reprendra qu'après un retour à la normale de cette dernière. Pour créer cette usine logicielle, il est nécessaire de combiner un outillage à des pratiques d'ingénierie logicielle.

L'outillage va permettre de répondre à divers besoins comme du stockage centralisé, l'exécution automatique de tests, le déploiement d'environnements et bien plus, en s'appuyant sur une multitude d'outils. Les pratiques d'ingénierie logicielle feront appel à ces différents outils aux moments opportuns et de façon automatique, en fonction de l'état d'avancement du produit sur la chaîne :

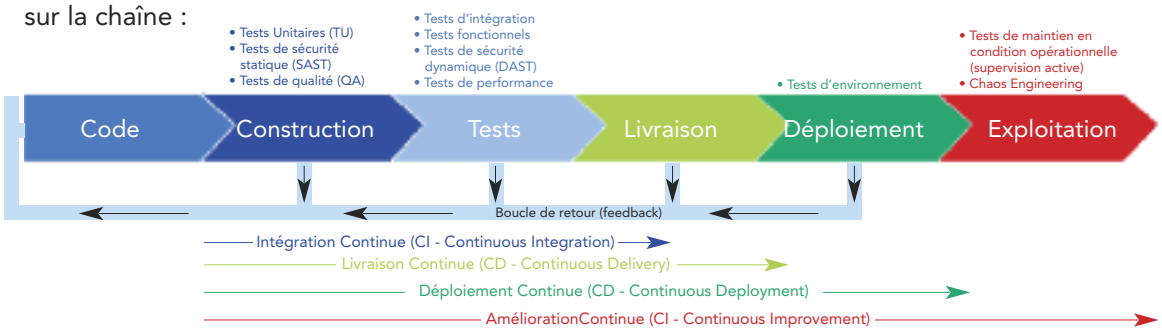


Figure 4 : Les pratiques d'ingénierie logicielle

[8] : *Docker : https://www.docker.com/why-docker*
[9] : *Infrastructure en tant que code : https://fr.wikipedia.org/wiki/Infrastructure_as_Code*
[10] : *Ansible : https://www.ansible.com/*

a. Pratique 1 : l'intégration continue (CI – Continuous Integration).

La pratique d'intégration continue permet, comme son nom l'indique, d'intégrer continuellement du nouveau code produit par le développement.

Coupler à un gestionnaire de code source comme par exemple Github[11], Gitlab[12], … cette pratique va consister principalement à s'assurer que le code se construit (build) correctement, qu'il respecte les critères de qualimétrie en vigueur et que l'exécution des différents types de tests au plus près de ce dernier réussisse. En voici une liste non-exhaustive :

- Test unitaire.

Teste le bon fonctionnement d'une partie précise d'un produit ou d'une portion d'un programme.

- Test de sécurité statique (SAST) de code.

Les SAST, signifiant « Static Application Security Testing », consistent à analyser statiquement du code (revue automatisée) dans le but de trouver des failles de sécurité.

- Test de qualimétrie.
  - Analyse statique du code.

Teste que le code est bien conforme aux bonnes pratiques de développement pour un langage de programmation donné.

  - Taux de couverture par les tests unitaires.

Remonte la quantité de code couverte par des tests unitaires.

  - Test de la Quality Gate.

Une « Quality Gate », pouvant être traduite par «portail de qualité» ou encore «niveau de qualité», représente un ensemble de seuils prédéfinis, comme par exemple le taux de couverture de code par les tests (ex : 80 %). Si cette dernière n'est pas respectée, cela indique que le code produit n'est pas conforme aux exigences de qualité en vigueur. Dans ce cas, la chaîne d'intégration continue s'arrête.

Par la suite, il va être nécessaire de déployer cet artefact nouvellement produit sur divers environnements (test, recette, pré-production, …), afin de pouvoir y effectuer une batterie de tests dont voici une liste non-exhaustive :

- Test d'intégration.

Teste qu'un module fonctionne avec un ou plusieurs autres modules du produit.

- Test fonctionnel.

Teste qu'une fonctionnalité est conforme aux spécifications fonctionnelles.

- Test de sécurité dynamique (DAST).

Les DAST, signifiant « Dynamic Application Security Testing », consistent à détecter des vulnérabilités dans un produit en cours d'exécution sur un environnement.

---

[11]: Github : https://github.com/
[12] : Gitlab : https://about.gitlab.com/

- Test de performance.
    - Test de charge.

Test dont l'objectif est la mesure du comportement d'un composant ou système avec une charge croissante, par exemple nombre d'utilisateurs et/ou nombre de transactions en parallèle pour déterminer quelle charge peut être gérée par le composant ou système.

- Test aux limites.

Test au cours duquel le comportement du système aux limites du modèle d'usage de l'application est vérifié.

- Test de robustesse.

Test pour déterminer la robustesse[13] d'un produit logiciel.

- Test de fiabilité.

Le processus de tests pour déterminer la fiabilité[14] d'un produit logiciel.

- Test de stress.

Un type de test de performance mené pour évaluer un système ou composant jusqu'aux limites ou au-delà des limites de ses charges de travail anticipées ou spécifiées, ou avec une disponibilité réduite de ressources telles que l'accès mémoire ou serveurs.

b. Pratique 2 : La livraison continue (CD – Continuous Delivery).

Dès lors que tous les tests auront été effectués avec succès, il sera alors possible d'effectuer une livraison de l'artefact nouvellement produit au sein d'un gestionnaire d'artefacts tels que Nexus OSS Repository[15], Artifactory[16], … afin de le stocker et de le gérer en configuration et en version.

Toutefois, le passage en production de ce dernier se fera **manuellement**.

c. Pratique 3 : le déploiement continu (CD – Continuous Deployment).

Cette pratique prête souvent à confusion avec la pratique précédente qu'est la Livraison Continue, puisque ces deux pratiques ont le même acronyme « CD ». La différence majeure entre ces deux pratiques est que le Développement Continu va jusqu'au déploiement automatique d'une livraison en production en utilisant des stratégies de déploiement adaptées, telles que du «blue / green deployment[17]» ou encore du «canary release[18]» par exemple.

Pour s'assurer que ces déploiements, montées de version ou mises à jour d'un produit se font dans les meilleures conditions, il est possible d'effectuer des tests d'environnement sur les environnements cibles afin de s'assurer qu'ils disposent bien des critères environnementaux nécessaires (ex : RAM, espace de stockage, CPU, services correctement démarrés, …) afin d'effectuer cette action dans les meilleures conditions.

---

[13] : Robustesse : Le degré pour lequel un composant ou système peut fonctionner correctement en présence de données d'entrée invalides ou de conditions environnementales stressantes [IEEE 610]
[14] : Fiabilité : La capacité d'un produit logiciel à effectuer les fonctions requises dans les conditions spécifiées pour des périodes de temps spécifiées, ou pour un nombre spécifique d'opérations [ISO 9126]
[15] : Nexus OSS Repository : https://fr.sonatype.com/nexus-repository-oss
[16] : Artifactory : https://jfrog.com/artifactory/
[17] : Blue / Green deployment : modèle par lequel les temps d'arrêt sont réduits pendant les déploiements en production d'un produit en ayant deux environnements de production ("blue" et "green")
[18] : Canary release : modèle pour déployer des versions vers un sous-ensemble d'utilisateurs ou de serveurs. L'idée est de déployer d'abord la modification sur un petit sous-ensemble de serveurs, de la tester, puis de la transférer au reste des serveurs.

c. Pratique 4 : l'Amélioration Continue.

Cette pratique va permettre de mettre en application un concept fort de l'approche DevOps qui est l'Amélioration Continue. Comme son nom l'indique, l'Amélioration Continue consiste à améliorer continuellement les pratiques, processus et outils en place pour optimiser la chaîne afin de délivrer le plus rapidement et le plus efficacement possible de la valeur aux clients finaux.
Outre la mise en place d'indicateurs de performance (KPI) ainsi que la collecte de métriques et logs pour de la supervision, il est possible d'effectuer ces types de test dont voici une liste non-exhaustive :

- Test de résilience.

Popularisé par le concept du «Chaos Engineering[19]», ce type de test consiste à choisir régulièrement, au hasard, des instances dans l'environnement de production et à les mettre délibérément hors service. En « tuant » régulièrement des instances au hasard, on s'assure d'avoir anticipé correctement la survenue de ce type d'incidents en mettant en place une architecture suffisamment redondée et scalable pour ne pas impacter l'existant. Cela permet aussi aux opérations de s'améliorer continuellement puisqu'elles pourront être confrontées à des pannes réelles qui n'auraient pas été identifiées ou encore à valider des procédures en place.

- Test de maintien en condition opérationnelle.

Une fois qu'un produit est en production, il est possible d'effectuer des tests de maintien en condition opérationnelle (MCO), aussi appelés «Healthcheck», qui consistent par exemple à s'assurer qu'une URL est toujours accessible, qu'un port HTTP/TCP est bien ouvert ou encore que tel mot est bien affiché sur une page web afin de s'assurer du bon fonctionnement du produit. Dans le cas où l'un des tests est en échec, il sera alors possible d'effectuer une action spécifique qui peut être automatisée en fonction des cas, afin de résoudre le problème.

4. Conclusion.

Lorsque l'on décide de faire de l'Agilité dans ses projets de développement, il est indispensable de penser au fait que cela aura un impact important, notamment sur les opérations. Ne pas y prêter attention, c'est prendre le risque d'instaurer un «mur de confusion» entre ces deux métiers, qui sera source de conflits, d'incompréhension, de manque de confiance et de collaboration et aura un impact non-négligeable pour les entreprises et notamment une augmentation du Time To Market[20] d'un produit, voire la non-satisfaction des clients.

C'est pourquoi il est indispensable de réussir à casser les silos qui séparent le développement et les opérations, afin que l'entreprise soit en mesure d'apporter continuellement de la valeur sur ce qui est produit, dans le but de satisfaire le client et c'est ce à quoi l'approche DevOps tend à répondre.

---

[19]: Chaos Engineering : http://principlesofchaos.org/
[20] : Time To Market : correspond au temps que met une idée pour se transformer en une fonctionnalité utilisée

En effet, cette approche a pour objectif de répondre aux impacts qu'entraîne l'Agilité en transformant l'entreprise, tant d'un point de vue organisationnel et culturel que d'un point de vue technique, pour qu'elle permette plus de collaboration, de partage, de confiance, de fluidité, d'autonomie et bien plus encore dans son fonctionnement.

Cependant, il va être nécessaire d'enclencher des chantiers, parfois transverses, comme la mise en place d'une usine logicielle pour industrialiser les tests et les livraisons, ou encore l'élaboration d'un catalogue de services pour déléguer et favoriser l'autonomie des équipes.

Mais cette transformation ne peut se faire du jour au lendemain et surtout sans l'adhésion de toutes les parties prenantes, c'est pourquoi il est nécessaire d'avoir une réelle stratégie de mise en œuvre de cette approche DevOps afin d'expliquer, de former et d'accompagner les personnes aux changements engendrés par cette dernière.

II.3- Les tests exploratoires, une obligation dans l'Agilité.

par Frédéric Assante Di Capillo

1- Introduction :

La méthode des Tests Exploratoires est devenue un incontournable des méthodes agiles pour plusieurs raisons :

- C'est une méthode complémentaire aux Tests classiques, qui ne perturbe donc pas le déroulement d'un «Sprint» ;
- Elle correspond aux principes agiles car elle ne demande pas une spécification lourde des Tests, en cela elle favorise le résultat plus que la spécification ;
- Elle permet de mixer diverses méthodes de validation (manuelle, automatique, fonctionnelle, non fonctionnelle) ;
- Elle met en avant l'humain (Expertise nécessaire) plus que le processus ;
- Elle permet d'augmenter la collaboration entre membres d'un projet (Scrum team) ;
- Elle permet un retour immédiat des problèmes rencontrés (collaboration développement-Qualiticiens).

Dans ce chapitre nous préciserons, tout d'abord, ce que sont les Tests Exploratoires (et ce qu'ils ne sont pas !) ; nous comparerons avec les Tests Manuels classiques ; nous donnerons quelques pistes pour expliquer pourquoi appliquer ce type de validation.

Puis, dans une seconde partie, nous donnerons un exemple de mise en pratique des Tests Exploratoires avec un canevas complet d'organisation. Enfin, nous donnerons les avantages et les inconvénients de cette méthode.

2- Définitions :

En préambule, il est important de préciser que les Tests Exploratoires ne sont pas un « type de tests » mais ***une façon de valider****. Cette notion est importante pour comprendre pourquoi nous devons utiliser cette méthode.*

Tests Exploratoires, qu'est-ce que ce n'est pas ?

Il y a souvent confusion entre différentes méthodologies de validation, qui font partie d'un même groupe. Toutes ces méthodes peuvent avoir leur intérêt, dépendant du type d'application sous validation, de la maturité de cette même application et de l'équipe de validation. Nous pouvons citer les méthodes suivantes (liste non exhaustive) :

Monkey Testing :

Les « Monkey Testing » : cette méthode consiste à donner un minimum de consignes aux personnes en charge de la validation. Les idées directrices sont :

- Cliquer partout pour éprouver la robustesse de l'application.

- Eprouver les parties potentiellement sensibles de l'application sans tenir compte de consignes particulières.
- Pas de préparation particulière.
- Pas de consignes ni de scripts.

Ad Hoc Testing :

Les « Ad Hoc Testing » : autre méthode qui consiste à effectuer une validation non formelle de l'application. Cette phase de validation serra complètement libre et totalement improvisée.

Error Guessing :

Cette autre méthode consiste à effectuer une validation basée sur les erreurs déjà trouvées dans l'application. Elle est basée sur l'expérience et l'intuition des équipes de validation, cela pouvant aller de pair avec une écriture de cas de tests.
Il existe d'autres méthodes de validation de même catégorie non abordées dans ce chapitre.

Tests Exploratoires, qu'est-ce que c'est ?

Une session de Tests Exploratoires est une phase de validation qui est organisée pour trouver des failles dans une application donnée. Cette méthode n'a pas pour vocation de remplacer d'autres types ou phases de tests. Elle viendra en complément de Tests Manuels classiques, d'une régression automatisée et de toute autre méthode de validation décrite précédemment (Monkey Testing, Error Guessing, etc.). Cette méthode est formelle et nécessite une organisation en amont et un suivi post-session (voir chapitre Organisation). Elle doit correspondre à un besoin et impose des contraintes.

Comme décrit précédemment, les tests exploratoires sont une méthode de validation avec :

- Une phase d'apprentissage pour bien connaître l'application ;
- Un questionnement sur la gestion des fonctions complexes ;
- Un accroissement itératif de la connaissance de l'application (session après session et entre les sessions).

Cela repose sur :

- **Un Animateur ou Chef d'Orchestre.** Celui-ci devra effectuer, en amont, la préparation de la session (en collaboration avec les experts de l'application) ; être le référent durant la phase de validation elle-même ; s'assurer du suivi de la session, notamment les rapports et la gestion des problèmes trouvés (voir chapitre Organisation)
- **L'expérience du (des) Testeur(s).** Ce point et une notion clé des Tests Exploratoires. En effet, toute la validation sera basée sur la connaissance et l'expérience des testeurs, qui seront à même d'investiguer toutes les parties de l'application et tout spécialement les parties connues comme sensibles. **Sans cette expérience, les Tests Exploratoires seront peu ou pas efficaces.**
- **Une préparation allégée.** Cette méthode n'est pas basée sur une écriture de scripts, contenant l'ensemble des étapes nécessaires à l'exécution d'un flux. Elle est basée sur la création de Chartes (voir chapitre sur les Chartes) permettant une description allégée de ce qui sera à tester et utilisant l'expérience des participants à la session.

• **Une limitation dans le temps.** Cette limitation est nécessaire pour une bonne organisation de la session et, notamment, pour le planning de cette session qui sera établi en amont par l'Animateur ou le Chef d'Orchestre.

• **Des Chartes.** Celles-ci permettront de définir clairement les objectifs et le cadre de la session, ainsi que tous les éléments nécessaires à une validation efficace (voir chapitre sur les Chartes).

Les Tests Exploratoires sont clairement définis par les organismes référents des méthodes de tests (ISTQB) et de l'Agilité (Agile Alliance), mais aussi Wikipédia (voir chapitre références)

3- Comparaison avec les Test Manuels classiques.

Même si les Tests Exploratoires n'ont pas pour vocation de remplacer les Tests Manuels classiques (ou les tests automatiques), nous pouvons comparer ces deux types de validation pour mettre en lumière le bénéfice des Tests Exploratoires.

Tests Manuels classiques	Tests Exploratoires	Explication
Exécution Unique, automatisable	Exécution Unique, complexe à automatiser	Dans les deux cas l'exécution sera unique, mais, pour les tests classiques, nous avons une définition des étapes, il sera donc plus facile d'automatiser les scripts.
Exigences/Spécifications, doivent être définies	Exigences non définies, support obligatoire	Dans le cas des Tests Exploratoires, il n'est pas obligatoire de lier ces tests à des exigences. En revanche, une référence est obligatoire pour savoir quoi tester.
Couverture plus claire	Couverture Floue	Contrairement aux Tests Manuels classiques qui sont liés aux exigences, les Tests Exploratoires, par définition, ne sont pas liés à ces mêmes exigences. De ce fait, la couverture de tests ne peut être connue.
Demande une Phase de Design Importante	Phase de Design peu Importante	Pour les Tests Manuels classiques, l'ensemble des étapes des tests devront être décrites, ce qui n'est pas le cas des Tests Exploratoires. Nous aurons donc une phase de design beaucoup plus succincte.
«Cerveau» utilisé avant la validation	«Cerveau» utilisé pendant la validation	La phase d'exécution des tests étant préparée en amont pour les Tests Manuels classiques (étapes décrites avec précision), le testeur ne peut que suivre ces étapes. Pour les Tests Exploratoires, comme rien n'est décrit, le testeur devra faire appel à son intelligence pour explorer l'application.

Planification de la validation	Validation Réactive	Contrairement à une validation classique où tout est planifié, les Tests Exploratoires permettent une meilleure réactivité aux détections de problèmes : en effet, lors de la validation d'une Charte, le testeur pourra insister sur un **«nid» de défauts**.
Pas besoin d'avoir de Compétences pour l'Exécution	Nécessite une bonne Connaissance de l'Application	C'est un des points clés des Tests Exploratoires. Il est nécessaire de composer son équipe avec des personnes ayant une bonne connaissance métier pour optimiser l'efficience de la session de validation.
Risques évalués avant la validation	Risques évalués pendant la validation	Une gestion des risques de la campagne de tests est faite en amont pour les Tests Manuels classiques. En revanche, elle sera directement évaluée pendant la phase de validation pour les Tests Exploratoires.
Validation plus Large	Validation plus en profondeur	Le but d'une phase de validation classique est de couvrir l'ensemble des fonctionnalités de l'application. Mais les Tests Exploratoires permettent de valider plus en profondeur ces mêmes fonctionnalités. C'est en cela que les deux méthodes sont complémentaires.
Temps important pour préparer la campagne	Temps assez court pour préparer la campagne	Lors de la phase de préparation, du fait de l'écriture de toutes les étapes des tests, une durée importante est nécessaire pour les Tests Manuels classiques, ce qui n'est pas le cas pour les Tests Exploratoires.
Focus et tactiques de validation statique	Focus et tactiques de validation dynamique	Alors que pour la validation classique, les méthodes utilisées sont statiques, pour les Tests Exploratoires, le peu de contraintes permettra plus de «dynamisme».
Pas efficaces pour les tests non fonctionnels	Assez efficaces pour les tests non fonctionnels	Pour la partie non fonctionnelle de la phase de validation classique, il est très difficile d'anticiper ce genre de validation. En revanche, lors d'une phase de Tests Exploratoires, il est plus facile d'imaginer sur le moment quelques tests non fonctionnels, même si cela n'est pas le meilleur moyen pour ce genre de validation.
Basé sur un suivi des risques connus	Basé sur une analyse des risques dynamiques	Les Tests Manuels classiques sont souvent basés sur une connaissance des risques passés. A contrario, les Tests Exploratoires, même si ils sont basés sur les risques connus, peuvent s'adapter dynamiquement si de nouveaux risques apparaissent durant la session de validation.

4- Quand organiser une session de Tests Exploratoires ?

Les Tests Exploratoires sont une méthode de validation relativement lourde à mettre en œuvre, donc assez coûteuse. Elle demande notamment une préparation assez conséquente (voir chapitre sur l'organisation). Elle demande aussi de mobiliser un certain nombre de personnes pendant une durée comprise entre un et deux jours et implique aussi la présence d'un Animateur et un suivi post-session. Pour être le plus efficace possible, il est conseillé :

- De n'appliquer cette méthodologie que sur des **applications matures**, c'est-à-dire que cette méthode n'est pas efficace sur des applications qui viennent de démarrer leur développement ou qui n'ont pas atteint une maturité suffisante pour être relativement stables ;
- De n'appliquer cette méthodologie que si nous avons un **groupe d'experts** suffisant pour être efficace lors de la session de validation. Sans ce « pool » d'experts, comme les personnes n'auront pas le niveau de connaissance nécessaire pour être efficaces, nous conseillons de prévoir d'autres types de validation comme le « Monkey Testing » ;
- D'être sûr d'avoir un **Animateur ou Chef d'Orchestre** pour gérer les différentes phases d'une session de Tests Exploratoires. Sans cet Animateur, il y a un risque de n'avoir pas l'ensemble des phases correctement organisées et donc une perte d'efficacité ;
- D'avoir le **soutien du Management**. Sans ce soutien, il y a un risque élevé de procéder à une session dégradée, donc moins efficace, et de ne pas avoir une implication complète du groupe de validation durant la session.

5- Qui inviter pour une session de Tests Exploratoires ?

Comme nous l'avons vu précédemment, une session de Tests Exploratoires nécessite un groupe d'experts. Nous pouvons évidement mixer ce groupe avec des gens plus 'novices' qui deviendront les futurs experts ! Il est à noter que les Tests Exploratoires ne sont pas le pré carré des équipes de validation. Pour une session de Tests Exploratoires, nous pouvons inviter :

- Les **Ingénieurs QA**, notamment ceux ayant une grande expérience de l'application ;
- Les **Analystes projets**, qui ont défini les fonctionnalités de l'application ;
- Des **Développeurs** qui ont participé à la conception de l'application ;
- Le **Product Owner** et le **Scrum Master**, si le Projet est en mode Agile ;
- Le **Chef de projet** ;
- Des **Représentants des Clients** ;
- Des **Utilisateurs** de l'application ;
- Toute personne ayant une bonne connaissance de l'application.

A noter que, pour les personnes susceptibles de participer à une session, une fiche de renseignement peut être créée pour faciliter l'attribution des Chartes (voir chapitre sur les Persona).

6- Comment organiser une session de Tests Exploratoires ?

Comme nous avons pu le voir précédemment, une session de Tests Exploratoires ne peut absolument pas s'improviser et nécessite une importante préparation. Nous allons proposer un découpage en quatre phases pour organiser une telle session.

Phase 1 : Préparation globale.

Cette phase est dédiée à la préparation des futures sessions de Tests Exploratoires. Elle doit être organisée et suivi par l'Animateur et va comporter plusieurs points :

- Une présentation initiale des sessions de Tests Exploratoires. Cette présentation peut être nécessaire si peu de personnes, formant l'équipe globale du projet, ont une connaissance des Tests Exploratoires. Elle consistera à expliquer ce qui va être fait durant les sessions et les avantages de cette méthode ;
- Une définition des Chartes (voir chapitre sur la Charte) : c'est-à-dire toutes les informations nécessaires devront être présentes dans cette Charte. Attention de ne pas donner trop d'informations, cela risque de brider les experts lors de la phase de validation ;
- Une définition des Persona (voir chapitre sur les Persona), qui devra comporter toutes les informations nécessaires à l'attribution des Chartes. Cela devra être fait en collaboration avec les responsables du projet ;
- Une définition du processus et des outils qui permettront la gestion des erreurs lors de la phase de Validation. Ces processus et outils peuvent différer par rapport aux processus classiques pour des raisons d'efficacité et de rapidité.

Phase 2 : Préparation de la session.

Cette phase va consister à préparer, en amont des Tests Exploratoires, tout le matériel nécessaire à une bonne organisation de cette session. Cette phase est capitale car une mauvaise préparation entraînerait une dégradation dans la qualité de la validation, le but étant de se concentrer sur la validation. Cette phase sera gérée par l'Animateur et la personne la plus à même de définir les besoins, le Product Owner étant souvent cette personne. Cette phase comprendra :

- Une sélection des participants à la session, qui seront choisis dans la liste des Personae définies lors de la phase 1 (voir chapitre 'Qui inviter') ;
- Une détermination du temps de la session, habituellement entre ½ journée et 2 jours pour optimiser la session. Cela dépend de la complexité de l'application à tester, de l'objectif attendu et de la disponibilité des participants ;
- Tout l'aspect matériel. Réservation d'une salle : pour augmenter l'efficacité, il est recommandé de regrouper tous les participants ; Envoie des mails d'invitation à tous les participants ; Réservation des repas (voir phase 3), etc. ;
- Une création/attribution des Chartes et leur sélection pour cette session. L'idée est de définir ce qui sera validé pendant la session, en s'appuyant sur la liste des Chartes et de les attribuer aux bonnes personnes, en se référant aux Personae. Un planning permettra d'optimiser la session ;

• Une création d'un tableau de bord pour le suivi de la session qui pourra être utilisé pour le rapport. Ce tableau de bord peut être inclus dans votre outil de suivi de tests (Octane, X-Ray, etc.) ou être physique, sur un tableau se trouvant dans la salle de la session ;

• Une préparation du rapport final avec les informations nécessaires, comme la liste de diffusion, les Chartes validées, la liste des participants, etc. ;

• La définition de la version qui sera validée ainsi que la plateforme qui sera utilisée.

Phase 3 : Session de Validation.

La session de validation est, évidemment, le point central des Tests Exploratoires. Cette session dépend entièrement de la phase précédente de préparation. Il est très important d'avoir un réel esprit de groupe pour cette session. Pour ce faire, l'Animateur aura un rôle central et devra entraîner avec lui tout le groupe mais aussi être le garant de la session, notamment en s'assurant que tout le monde comprenne bien les Chartes qui lui sont assignées, que les temps sont respectés, que les problèmes trouvés sont bien enregistrés (avec toutes les informations requises). Dans l'esprit d'avoir une réelle communication entre les différents participants, il est préférable de réserver une salle commune.

Proposition de planning pour une session de Tests Exploratoires d'une journée :

- 9h00 Café de bienvenue et accueil des participants.
- 9h30 Introduction par l'Animateur avec explication des objectifs de la journée (une explication des concepts des Tests Exploratoires peut être faite si les participants sont peu familiers avec cette méthode).
- 9h50 Distribution des Chartes et explications si nécessaire.
- 10h00 Début de la validation, basée sur les Chartes.
- 12h30 Première session de retour (discussion ouverte pouvant déboucher sur de nouvelles Chartes pour la prochaine session).
- 11h40 Repas en commun (pour renforcer la cohésion de l'équipe)
- 13h30 Distribution des Chartes de l'après-midi
- 15h30 Break
- 15h40 Continuation de la validation
- 17h30 Deuxième session de retour et collecte des informations et problèmes rencontrés
- 18h00 Clôture de la session, un mot du Management peut être le bienvenu.
- 18h15 After Works ? ◡̈

Il est à noter que les phases de retour sont très importantes, notamment pour :

• Améliorer l'organisation des prochaines sessions ;

• Créer de nouvelles Chartes si des parties de l'application ce sont montrées plus sensibles qu'attendu ;

• Compléter et mettre à jour les Persona ;

• S'assurer de la bonne remontée des problèmes ;

• S'assurer d'avoir toutes les informations nécessaires pour le rapport final ;

• Augmenter la cohésion des équipes en ayant des échanges directs ;

• Etc.

Phase 4 : Publication des résultats et suivi de la Session.

Une telle session n'a de sens que si on communique sur ce qui a été fait et sur les résultats obtenus. Après la collecte des informations, faite en fin de session, il est nécessaire de fournir un rapport complet sur les résultats. Ce rapport pourrait contenir :

- Le périmètre global de la validation ;
- La version du produit validé ;
- La liste de fonctionnalités couvertes (basées sur les Chartes) ;
- La liste des participants ;
- La liste des problèmes trouvés ;
- La liste des nouvelles Chartes créées, après passage des Chartes prévues ;
- Le retour des participants.

D'une part, les problèmes trouvés devront être suivis jusqu'à leur résolution. D'autre part, en fonction des potentiels enregistrements faits durant la session, la régression pourra être complétée par de nouveaux scripts automatiques.

Durant la rétrospective, un retour de cette session devra être initié par l'Animateur.

6- Avantages et Inconvénients.

Vous pouvez trouver, si dessous, des listes d'avantages et d'inconvénients. Ces listes ne sont pas exhaustives mais regroupent les points majeurs dont il faut tenir compte.

Avantages :

- Liberté pour le(s) testeur(s).
- Souligne les parties sensibles de l'application.
- Aide à trouver des problèmes sur les exigences/spécifications/Guide utilisateur/etc.
- Permet aux testeurs d'augmenter leur engagement.
- Gain de temps dans la conception des tests.
- Concentration sur l'expérience plus que sur le processus.
- Croissance itérative de la connaissance du produit.
- En accord avec les méthodologies agiles, utilisable dans d'autres méthodologies.
- Utilisable pour les tests non fonctionnels : performance, etc.

Inconvénients :

- Requiert des testeurs expérimentés.
- Nécessite une bonne connaissance du produit pour la conception initiale.
- Difficulté à suivre les progrès de la campagne de tests.
- Peu ou pas de couverture des besoins.
- Difficulté d'automatiser les tests.
- Scénarios de défaut non évidents à fournir.
- La charte de validation doit être clairement définie.

7- Charte.

La Charte est un élément très important des sessions de Tests Exploratoires. C'est avec les Chartes que l'Animateur, en collaboration avec les experts, va proposer les différentes parties de l'application à tester. Ces Chartes permettent aussi de rassurer le management sur la valeur d'une telle session car elle confère un côté maîtrisé et officiel !

Pour définir Les Chartes, nous pouvons nous baser sur plusieurs éléments :

- La connaissance des Ingénieurs Qualité qui connaissent les parties sensibles;
- Les doutes et connaissances du Product Owner;
- Les développeurs qui peuvent définir les zones de codes sensibles;
- Les informations remontant des « daily meetings » ;
- Les retours clients;
- Les retours des utilisateurs;
- Les retours de la phase de « Component Testing » ;
- Les précédentes sessions de Tests Exploratoires ;
- Etc.

Ces Chartes peuvent être des documents Word ou Excel mais peuvent aussi faire partie d'un référent de tests plus élaboré comme Octane ou X-Ray. Les différentes parties devront être définies dans la Phase 1 de l'organisation de la session.

Vous pouvez trouver, si dessous, un exemple de Charte :
Exemple 1 (Charte)

Objectif	Valider la partie prix de l'application
Durée	5 mai 2020 de 14 h à 16 h → 2 heures.
Périmètre	Se concentrer sur la partie prix de l'application. Après une sélection d'items à acheter, aller dans le panier et vérifier les prix ainsi que les remises. Les bons de réduction font partie de cette Charte. Les surcoûts en fonction des types de paiement aussi. Référence : • Guide Utilisateur • Tableau des prix fournis par le Client
Risques	L'accès au serveur des prix du client peut être limité ou lent
Méthode d'enregistrement	Utilisation de «sprinter»

8- Personae.

La notion de Personae est issue du monde du Marketing et n'est donc pas typique de la validation, ni des Tests Exploratoires. Elle a deux objectifs :

- Rendre la phase de Tests Exploratoires plus attrayante pour les participants (gamification).
- Connaître les compétences des participants pour mieux attribuer les Chartes et rendre la session le plus efficace possible.

Les Personae peuvent être très classiques, telles celles utilisées par le marketing (voir exemple ci-dessous), mais rien ne vous empêche de définir des profils plus «ludiques».

Exemple 2 (Persona Classique)

Nom : Terry Exbrayat	**Âge :** 42 
Profession : Product Owner	**Tel :** 06 06 06 06 06 **Mail :** MonMail@societe.com
Expériences : • 15 ans chez Société • Ingénieur QA depuis 8 ans	**Connaissances de l'application :** • Responsable depuis 5 ans • Bonne connaissance de l'Interface Utilisateur
Motivations : • Trouver des bugs dans l'IU • Travailler sur l'utilisabilité	**Connaissance en Tests :** • Expert ALM • Bonne connaissance de Sélénium

Les Personae peuvent aussi être définies par des caractères. Par exemple, donner des rôles à chacun comme :

- Le neutre, qui se focalise sur les fonctionnalités définies en amont ;
- Le négatif, qui va chercher les failles partout dans l'application ;
- Le positif, qui va valider le fonctionnement global de l'application sans chercher fatalement les erreurs ;
- L'émotionnel, qui utilise son ressenti pour aller débusquer les problèmes ;
- Le créatif, qui va imaginer des scenarios à la limite.

Ces rôles peuvent être inversés durant la session de tests.

Références :

- *ISTQB : http://glossary.istqb.org/search/exploratory%20testing*
- *Agile Alliance : https://www.agilealliance.org/glossary/exploratory testing/#q=~ (infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'exploratory*20testing))~searchTerm~'~sort~false~sortDirection~'asc~page~1)*
- *Wikipédia : https://en.wikipedia.org/wiki/Exploratory_testing*

II.4- Intégration Continue - Livraison Continue - Déploiement Continu

par Marc Hage Chahine et Mickael Carlin

L'Intégration Continue (I.C.), la Livraison Continue (L.C.) et le Déploiement Continu (D.C.) sont des notions souvent confondues avec l'Agilité ou le Devops et mal comprises.

L'I.C., la L.C. et le D.C. sont des pratiques démocratisées avec l'Agilité et le DevOps, bien que certaines de celles-ci soient adaptées/recommandées à tout logiciel développé et plus particulièrement avec des méthodes incrémentales. Les bonnes pratiques comme l'intégration fréquente du nouveau code, les tests précoces, l'automatisation des tâches répétitives ou la standardisation des projets devraient être obligatoires, peu importe la méthode de gestion du projet.

Les pratiques de l'I.C., de la L.C. et du D.C. sont mises en œuvre par un ensemble d'outils qui déclenchent une suite d'actions automatiques. C'est pour cela qu'on associe les termes de «chaîne outillée» et de «pipeline» à l'I.C. et ses extensions.

1- Définition de l'Intégration Continue, la Livraison Continue et le Déploiement Continu.

Intégration Continue :

L'Intégration Continue (I.C.) consiste à intégrer les branches de développement au code source d'une application le plus tôt et le plus fréquemment possible (plusieurs fois par jour) lors du cycle de développement.

On utilise pour cela un logiciel de gestion de versions et des flux de travail de branche adaptés, qui isolent les développements dans des branches de fonctionnalité. Les branches de fonctionnalités sont ensuite fusionnées au code source commun, via des demandes de fusion.

L'I.C. automatise des tests dont le rôle est de vérifier, à chaque modification du code source, que le résultat des modifications ne produit pas de régression visible au niveau du code, dans l'application développée, et que les nouvelles fonctionnalités sont implémentées correctement.

L'I.C. automatise aussi les étapes de construction du code qui contiennent les commandes nécessaires à la création d'un binaire installable de l'application (compilation). L'étape de construction peut aussi contenir la création automatique de documentation, l'enrichissement du journal de changement ou tout autre élément qui pourrait être généré automatiquement à ce stade.

L'ensemble de ces actions automatiques doivent être exécutées le plus souvent possible, sans pour autant nuire au temps d'exécution de l'ensemble des actions. Il s'agit donc de trouver à quel moment déclencher chacune des actions au sein des différentes phases du flux de travail.

Enfin, dans la plupart des flux de travail, une dernière action (création d'un tag, création d'une branche, fusion vers une branche de production…) permet de créer une version de l'application (Livraison Continue). Dans le cas du Déploiement Continu, cette action peut déclencher la mise en production automatique de cette version de l'application.

Chaque étape du flux de travail déclenche différentes étapes du processus d'Intégration Continue.

L'Intégration Continue automatise l'étape de « build » et certains tests.

Les tests vérifieront que chaque modification sur le code source ne produit pas de régression repérable par ces derniers dans l'application développée et que les nouvelles fonctionnalités sont implémentées correctement. Ils peuvent inclure les Tests Unitaires (T.U.), des checkstyle, des tests d'intégration, le respect des bonnes pratiques, des audits de sécurité automatique…

La phase de build automatise la compilation du code et le packaging, c'est-à-dire toutes les étapes requises pour que le code soit mis à disposition en tant qu'artefact déployable (aussi appelé objet binaire) sur les différents environnements.

Livraison Continue :

La Livraison Continue est une extension de l'Intégration Continue dans laquelle les artefacts créés sont automatiquement versionnés et livrés (ou facilement livrables, via un click, par exemple) dans un gestionnaire de dépôt d'objets binaires. L'application peut alors à tout moment être déployée par les membres des équipes d'opération (Ops) sur l'environnement de leur choix.

C'est l'équivalent de la construction du «.exe» pour un logiciel windows, du «.jar» pour un logiciel java ou de l'«apk» pour une application Android.

A ce stade, le déploiement des environnements de qualification ou de pré-production peut aussi être automatisé.

Déploiement Continu :

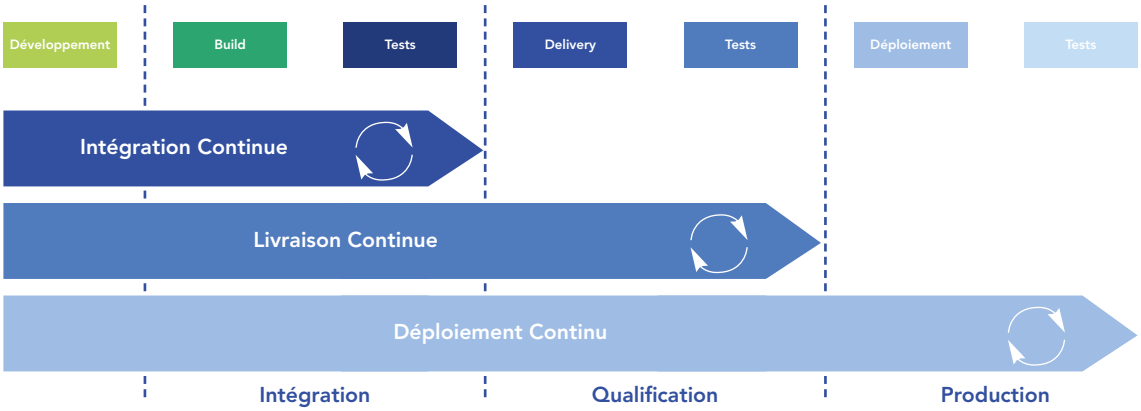
Le Déploiement Continu permet le déploiement automatique de chaque nouvelle version du code sur l'environnement de production avec un impact client immédiat. C'est donc une extension de la Livraison Continue sans qu'une action manuelle des Opérationnels (ou «Ops» de DevOps) soit requise pour la mise en production.

Le Déploiement Continu implique des méthodes de déploiement automatisées qui s'appuient généralement sur des logiciels de gestion de configuration.

Il implique, dans la majorité des cas, un niveau de test automatisé avancé à tous les niveaux de la chaîne outillée (tests unitaires, tests d'intégration, tests de sécurité, tests de charge, tests d'IHM, analyse de qualité…).

Enfin, il faudra prendre en compte des contraintes administratives liées à la gestion des changements, qui, selon les entreprises, auront plus ou moins de propension à accepter le D.C.

On peut facilement traduire ces définitions par ce schéma :



2- L'importance de l'Intégration Continue, de la Livraison Continue et du Déploiement Continu en Agilité.

L'I.C., la L.C. et le D.C. représentent un excellent choix d'outils dans la mise en œuvre de l'Agilité.

La qualité est essentielle en Agile. En testant systématiquement et fréquemment tous les ajouts au code source, on détecte au plus tôt les anomalies. Une bonne chaîne outillée d'I.C. automatise aussi des étapes d'audit de qualité du code (détection du code en doublon et des mauvaises pratiques) et de sécurité (recherche des failles connues). On assure donc une qualité optimale du produit. Produit qui sert de base pour tous les futurs développements.

De plus, l'Agilité prône des livraisons régulières de produits utilisables et fonctionnels. L'Intégration Continue et ses dérivés permettent de proposer ces livraisons régulières par le fait de fusionner les développements au code source plusieurs fois par jour et d'automatiser la construction des binaires et les tests. Dans la Livraison Continue et le Déploiement Continu, on rend même possible le déploiement de ces développements en production dans un délai très court, pouvant être de quelques minutes. Cela accroît la rapidité à laquelle les équipes de développement peuvent s'adapter au changement et aux demandes des clients.

La conséquence recherchée est une baisse considérable du temps de mise sur le marché d'une nouvelle fonctionnalité.

Enfin, l'Agilité encourage la communication et le travail avec des équipes pluridisciplinaires. Les chaînes d'Intégration, de Livraison ou de Déploiement Continus vont dans ce sens car elles utilisent des outils communs qui centralisent les développements, les tests, la construction de code, les déploiements, la documentation et souvent aussi l'aspect gestion de projet clairement orienté Agile (tableau agile, gestion des sprints, tableaux «burndown», «product backlog», gestion de la complexité…).

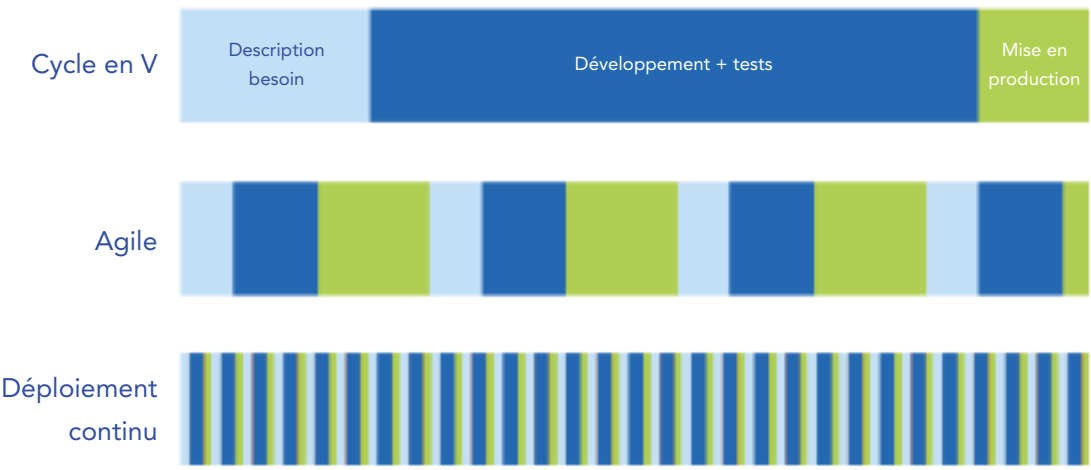
L'utilisation des mêmes outils par les développeurs, les testeurs et les Ops crée des interactions fortes entre tous les acteurs. Dans ce contexte, tout le monde peut se retrouver à développer un script participant à l'automatisation, tout le monde doit également prendre en compte les contraintes liées à la production ; dans la veine de l'Agilité et du DevOps.

3- La nécessité de l'automatisation des tests.

Vous l'avez compris, il n'existe pas de chaînes d'Intégration, de Livraison ou de Déploiement Continus dignes de ce nom sans tests.
Les tests associés à ces chaînes sont donc exécutés très fréquemment, à chaque commit, ce qui peut revenir à plusieurs fois par jour ! Cette intensité d'exécution des tests ne peut pas être soutenue par des tests manuels et des testeurs pour deux raisons principales :

- Le temps d'exécution est beaucoup trop long.
- Aucun testeur ne souhaite exécuter plusieurs fois par jours pendant des mois les mêmes tests !

Pour se rendre compte de l'intensité de la fréquence d'exécution des tests, on peut utiliser cette image (précédemment utilisée dans le chapitre I-4) :



4- Quels tests, à quel moment et pour quelle pratique ?

Comme décrit avec l'image de la première partie, l'Intégration Continue au sens large (I.C., L.C., D.C.) est indissociable des tests. Chaque niveau d'intégration engendre des tests (ou familles de tests) à ajouter.
Pour rappel, les tests dépendent du contexte. En fonction de l'impact des changements, différents types ou familles de tests sont nécessaires.

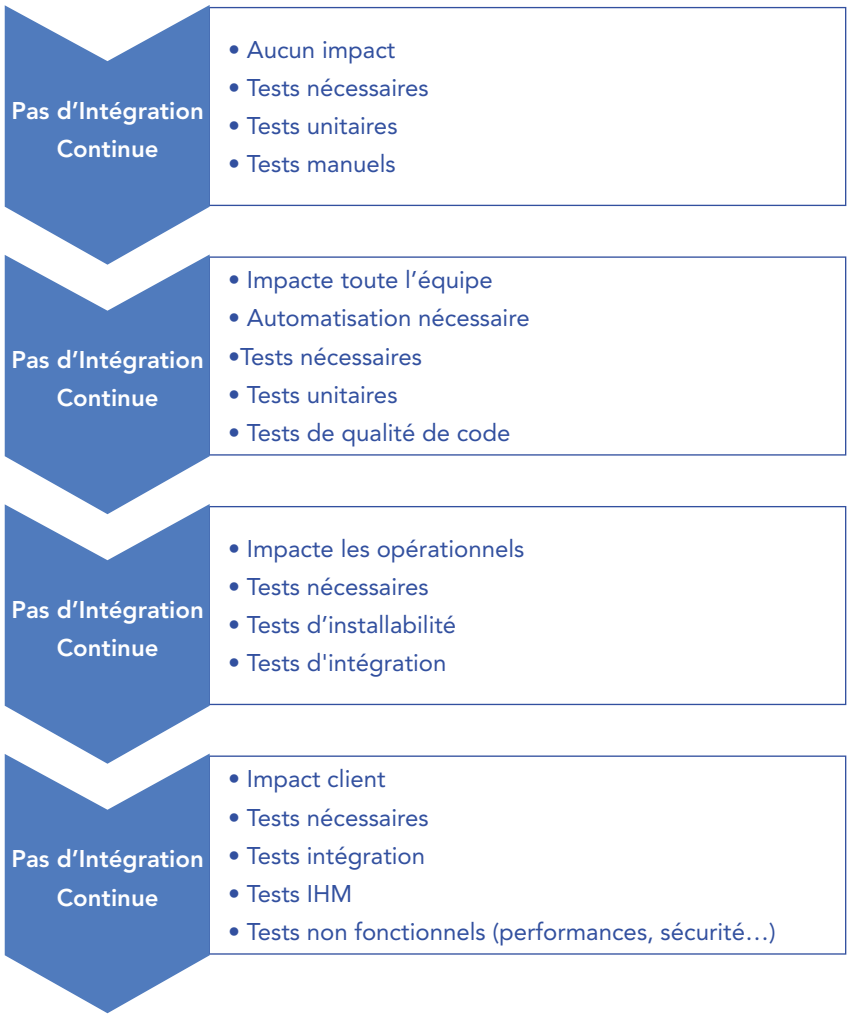
Comme vu au chapitre précédent, l'automatisation des tests est indispensable avec les chaînes d'Intégration Continue, les tests manuels atteignant très vite leurs limites en provoquant des interruptions dans le pipeline d'Intégration Continue qui est justement censé être continu !

Néanmoins, automatiser ses tests ne veut pas dire automatiser tous ses tests. Dans le cadre de l'intégration Continue, les utilisateurs finaux du produit ne sont pas impactés à chaque commit de code sur une branche de développement ou à chaque fusion (merge) de branche sur le code source. Dès lors, les tests nécessaires sont les tests utiles aux développeurs, à savoir les tests unitaires et les tests d'audit de qualité réalisés par des logiciels comme Sonarqube ou des tests de sécurité pouvant être réalisés par Fortify, par exemple. Idéalement, on automatisera aussi les tests d'intégration dès ce stade.

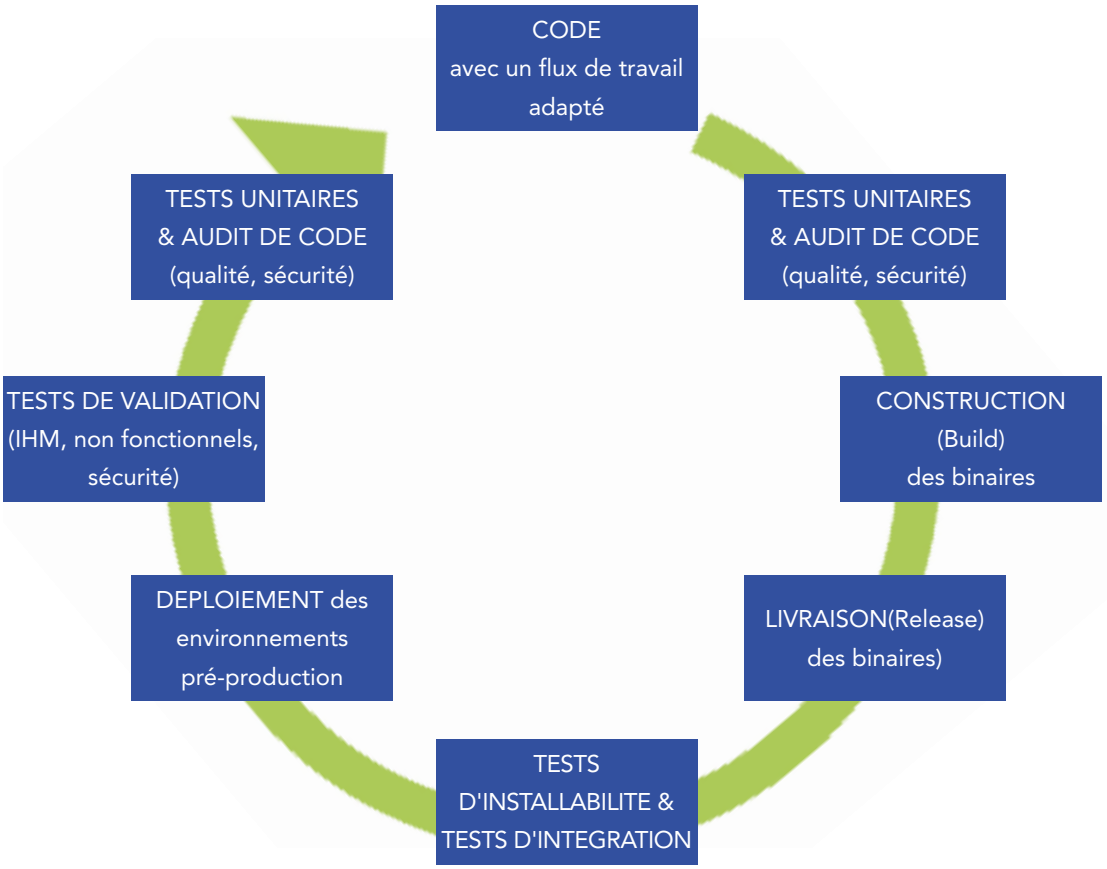
Dans le cas d'un impact client, il faut pouvoir assurer une qualité minimale lors de l'utilisation de l'application. Dans ce contexte qui est celui du Déploiement Continu, il faut automatiser les tests d'intégration et les tests de validation pouvant inclure, selon la nature du logiciel, des tests d'interface graphique, des test de sécurité, des tests non fonctionnels, des tests de performance...
Pour mettre en œuvre ces différents types de test, les chaînes de Déploiement Continu s'appuient généralement sur les environnements de qualification/pré-production et sur la conteneurisation (Docker) pour reproduire un contexte le plus proche possible de la production.

Enfin, entre l'Intégration Continue et le Déploiement Continu, il y a la Livraison Continue. La Livraison Continue doit proposer un « package » installable de l'application. Il existe donc un type de test particulièrement important pour la livraison, c'est le test «d'installabilité». Il est nécessaire de s'assurer que le « package » créé soit installable et utilisable sur l'ensemble des environnements ciblés. Si ce n'est pas le cas, la Livraison Continue perd tout son sens. C'est également à ce moment que l'automatisation des tests d'intégration n'est plus une option mais une nécessité.

On peut donc résumer les besoins en test avec cette image (chaque nouveau niveau nécessitant les solutions des niveaux précédents) :



5- Exemple de chaîne d'Intégration Continue.



CODE :

Développement des nouvelles fonctionnalités sur des branches de fonctionnalité. Fusion des branches de fonctionnalité sur une branche commune contenant le code source de l'application. Dans la majorité des cas, on utilisera l'incontournable Git en tant que gestionnaire de version. Le nombre, le nom des branches et la façon dont on les utilise peuvent changer en fonction du flux de travail utilisé. Le choix du flux de travail est primordial dans l'automatisation car il détermine à quel moment les différentes étapes du pipeline sont lancées. On peut citer GitFlow et GitlabFlow comme flux de travail adapté.

TESTS UNITAIRES et AUDIT DE CODE (automatisés dans l'I.C.) :

Déclenchement des tests unitaires et de l'analyse de qualité avec des outils comme Sonarqube. Les tests unitaires peuvent même être effectués dès qu'un commit est fait sur une branche de fonctionnalité, avant la fusion sur le code source. Tout est une question d'équilibre entre temps d'exécution d'un pipeline et fréquence de déclenchement de celui-ci.

CONSTRUCTION (automatisée dans l'I.C.):

Le code est compilé et conditionné en une version de binaire installable (un paquet debian, un .jar, une image docker ou n'importe quel autre format suivant le type de projet). Il est fortement recommandé de générer aussi la documentation, la release note ou tout autre élément

nécessaire à la création d'une nouvelle version de l'application. Généralement, cette nouvelle version est créée via un tag sur la branche contenant le code source, mais cela dépend du flux de travail utilisé.

LIVRAISON (automatisée dans la L.C.) :

La nouvelle version du binaire créée dans l'étape précédente est envoyée dans un dépôt d'objets binaire comme Artifactory ou Nexus. Le binaire est à présent téléchargeable et installable sur les différents environnements dont la production.

TESTS D'INSTALLABILITE et D'INTEGRATION (automatisés dans le D.C.) :

Tests d'installabilité des binaires et tests d'intégration sur un environnement le plus proche possible de la production.

DEPLOIEMENT des environnements pré-production (automatisé dans le D.C.) :

La nouvelle version du binaire est déployée sur l'environnement de qualification. On utilise idéalement un gestionnaire de configuration tel que Chef ou Ansible pour industrialiser les déploiements sur l'infrastructure et permettre le déploiement en un clic.

TESTS DE VALIDATION (IHM, non fonctionnels, sécurité) (automatisés dans le D.C.) :

Tests de validation, aussi poussés et divers que peuvent le nécessiter le produit ou la politique de l'entreprise.

DEPLOIEMENT EN PRODUCTION (automatisé dans le D.C.) :

Le binaire est enfin déployé en production de la même façon que sur les environnements précédents.

II.5- Automatisation des tests en environnement agile.

par Julien Van Quackebeke

1- Spécificité du test en environnement agile.

Les tests de qualité représentent une étape nécessaire et indispensable pour garantir la livraison d'une application moderne et robuste ayant le minimum possible de défauts : la satisfaction des utilisateurs et l'augmentation de la productivité des développeurs seront garantis.

Par ailleurs, afin de préserver leur productivité, il est indispensable aux équipes organisées en mode agile d'adopter l'automatisation des tâches répétitives, particulièrement les tests fonctionnels, étant donné qu'ils représentent généralement la tâche la plus lourde en terme de temps.

L'automatisation vient donc en faveur de l'Agilité grâce à la rapidité d'exécution, la flexibilité face aux modifications du code …

2- Rôle du testeur.

Le rôle d'un testeur dans une équipe agile inclut des activités qui génèrent et fournissent du feedback non seulement sur les statuts des tests, avancement des tests et qualité produit, mais également sur la qualité du processus.

Ces activités incluent :

- Comprendre, implémenter et mettre à jour la stratégie de test.
- Mesurer et reporter la couverture de test au travers de toutes les dimensions de couverture applicables.

- Assurer d'une utilisation correcte des outils de test.
- Configurer, utiliser et gérer les environnements de test et les données de test.
- Reporter les défauts et travailler avec l'équipe pour les résoudre.
- Coacher les autres membres de l'équipe sur les aspects pertinents du test.
- Assurer que les tâches de test appropriées sont planifiées pendant les planifications de release et d'itération.
- Collaborer activement avec les développeurs, les parties prenantes Métier et les Product Owners pour clarifier les exigences, spécialement en termes de testabilité, de consistance et de complétude.
- Participer pro-activement aux rétrospectives d'équipe, suggérer et implémenter des améliorations. Dans une équipe agile, chaque membre de l'équipe est responsable de la qualité produit et joue un rôle en réalisant des tests liés aux tâches.

3- ROI d'automatisation des tests.

Calculer le ROI, les gains en termes de qualité et de délai dans le cycle de développement, budgétiser l'investissement en matériel et ressources humaines.
L'étude d'opportunité permet de calculer les différents gains auxquels conduit l'automatisation des tests sur le périmètre choisi en termes de :

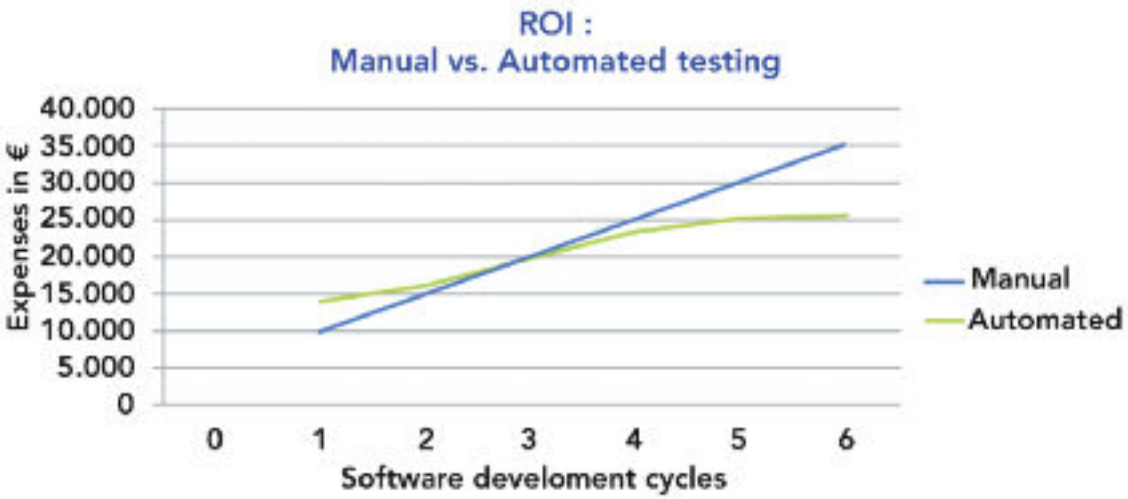
- Temps - Délai de mise en production.
- Qualité - Couverture de test, volume d'exécution de test, tests les plus opportuns à automatiser (stabilité fonctionnelle, technique, temps d'exécution manuel vs automatique, etc.).
- Financier – ROI.

Pour simplifier, le coût des tests peut être vu selon deux axes qui sont :

- La phase de mise en place. Cette phase regroupe l'écriture et la conception des cas ainsi que la plupart des procédures de sélection et n'a lieu qu'une fois. J'appellerai A le coût de la phase de mise en place. Le coût A est généralement nettement supérieur avec un test automatisé qu'avec un test manuel. En effet, le test automatisé ajoute le travail d'automatisation (et si besoin, l'implémentation de l'outil ou du framework).
- La phase d'exécution qui démarre après la première exécution du cas. Cette phase regroupe toutes les phases liées à l'exécution (exécution, analyse, bugs…) et la maintenance des cas. Cette partie, moins onéreuse que la première, peut être reproduite à l'infini. J'appellerai n le nombre d'exécutions du cas et B le coût de la phase d'exécution.

Le coût des tests peut donc est calculé après chaque exécution avec cette formule :

Coût = A + n*B



4- Meilleures pratriques d'automatisation des tests.

Pour un meilleur retour sur investissement et une meilleure qualité logicielle, voici les meilleures pratiques pour la création d'une infrastructure d'automatisation des tests d'UI :

- Comprenez l'application et sachez quoi automatiser.
- Choisissez le bon outil d'automatisation.
- Ne comptez pas SEULEMENT sur l'automatisation des tests d'interface utilisateur.
- Pensez à utiliser un framework BDD.
- Utilisez les modèles et les principes de conception de test (Page Object Model).
- N'utilisez JAMAIS Thread.sleep () à moins d'exigences de test spécifiques.
- N'exécutez pas TOUS les tests sur TOUS les navigateurs cibles.
- Séparez les tests d'infrastructure d'automatisation des tests.
- Rendez le framework d'automatisation de test portable.
- Prenez des captures d'écran pour une enquête d'échec.
- Créez des données de test efficaces.
- Tous les tests doivent être indépendants.
- Configuration des rapports de tests d'automatisation détaillés.

5- Stratégie d'automatisation des tests.

La stratégie de test d'automatisation est un élément primordial pour la réussite des projets d'automatisation. Elle permet notamment de déterminer ce qui est à automatiser, avec quels moyens et à quel moment du projet.

a. Automatisation des tests unitaires et d'intégration.

Objectif : permettre aux développeurs d'avoir un retour rapide sur la qualité du code.
Comment :

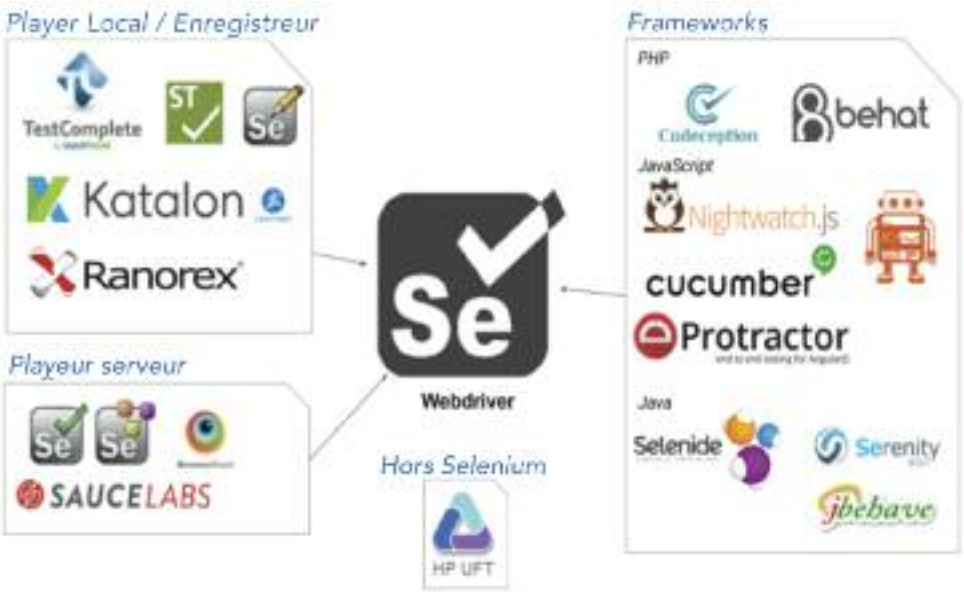
- Utiliser un framework de tests unitaires.

- Ne pas se contenter des nominaux.
  - Cas passants
  - Cas aux limites
  - Cas d'erreur
- Maîtriser les jeux de données Isoler un composant/une liaison pour le tester (bouchons)

b. Automatisation des tests fonctionnels.

Objectif : Le test automatisé a pour objectif de simplifier autant que possible les efforts de test grâce aux scripts, nous permettant ainsi de réduire le risque de régression.
Les automatiser permet de garantir une couverture constante des fonctionnalités.
Comment : Utiliser les outils suivants :

- **L'enregistreur de test** : Il permet d'enregistrer les actions faites dans l'application. Il permet de créer des scénarios de tests rapidement mais demande généralement d'être complété pour avoir une couverture satisfaisante.
- Les **«players»** de test permettent de jouer les tests. Il existe deux grands types de «players»,
- Les locaux : ils tournent sur la machine de l'utilisateur et bloquent (au moins partiellement) l'utilisateur (Ex : Selenium IDE, TestComplete, Katalon, Ranorex).
- Les serveurs : ils tournent sur une machine liée ou non au projet. Ils sont plus performants, ne bloquent pas l'utilisateur et peuvent s'intégrer dans un workflow de développement (Ex : Selenium webdriver, IBM rational functional tester, …). Les **actions** sont les différentes commandes qui seront jouées dans un test. Cela va du clic, à la gestion de variable, à la vérification d'attribut.
- Les **frameworks** sont des outils d'écriture de tests. Il en existe dans de nombreux langages. Ils ont différentes approches, allant du très technique au plus littéraire (Ex : Nightwatch (JS), Behat (PHP), CodeCeption (PH, jBehave (java), Specflow (c#))
- Les **drivers** sont les connecteurs entre le «player» de test et les navigateurs. Sur les «players» locaux, ils sont généralement intégrés mais dans le cadre de Selenium webdriver (le serveur de tests le plus répandu), ils sont à installer indépendamment.



6- Framework d'automatisation de test.

Un framework de test ou plus spécifiquement un framework d'automatisation de test est un environnement d'exécution pour les tests automatisés. C'est le système global dans lequel les tests seront automatisés. Il s'agit d'un ensemble d'hypothèses, de concepts et de pratiques qui constituent une plateforme de travail ou un support pour les tests automatisés.

Le framework de test est chargé de:

– Définir le format dans lequel exprimer les attentes.
– Créer un mécanisme pour connecter ou piloter l'application testée.
– Exécuter des tests
– Rapporter des résultats.

Propriétés d'un framework de test: Il est indépendant de l'application. Il est facile à développer, maintenir et perpétuer.

a. Type de framework d'automatisation de test.

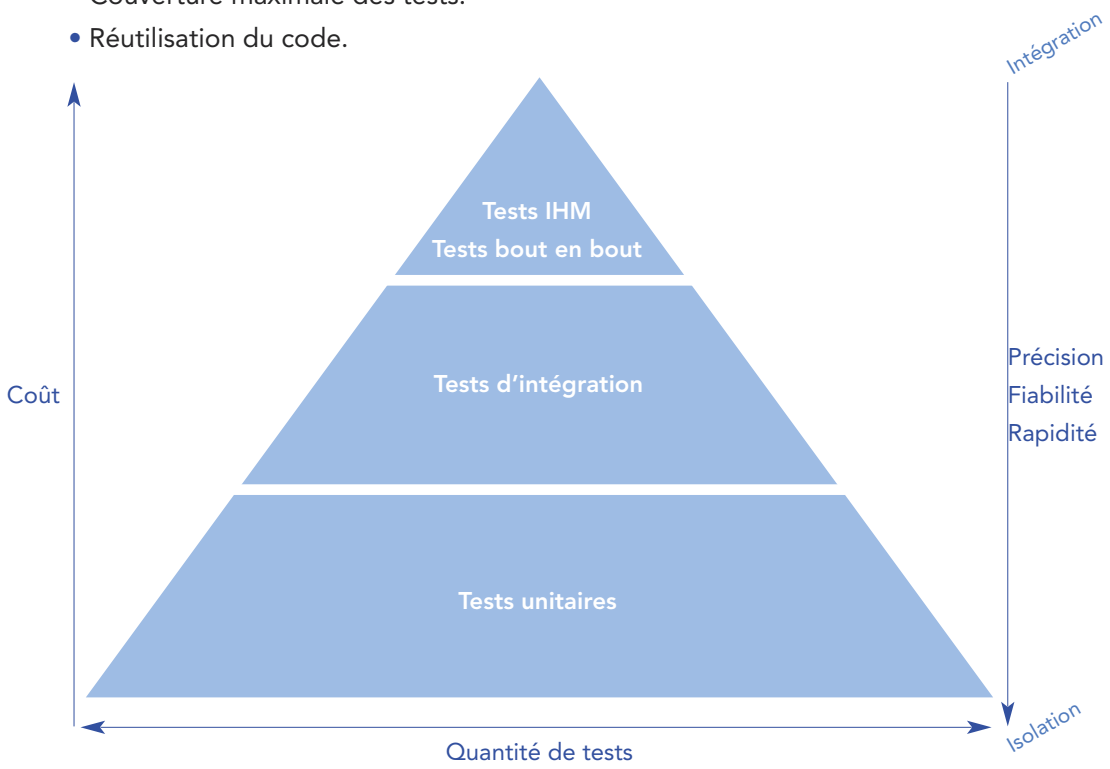
Il existe six types de framework d'automatisation des tests, chacun avec sa propre architecture et ses avantages et inconvénients. Lors de l'élaboration d'un plan de test, il est important de choisir le cadre qui vous convient.

- Linear Automation Framework
- Modular Based Testing Framework
- Library Architecture Testing Framework
- Data-Driven Framework
- Keyword-Driven Framework
- Hybrid Testing Framework

b. Avantages des framework d'automatisation de test.

L'utilisation d'une structure pour les tests automatisés augmentera la vitesse et l'efficacité des tests d'une équipe, améliorera la précision des tests, réduira les coûts de maintenance des tests et diminuera les risques. Les framework d'automatisation de tests sont essentiels à un processus de test automatisé efficace pour plusieurs raisons :

- Amélioration de l'efficacité du test.
- Réduction des coûts de maintenance.
- Intervention manuelle minimale.
- Couverture maximale des tests.
- Réutilisation du code.



7- Pyramide de tests.

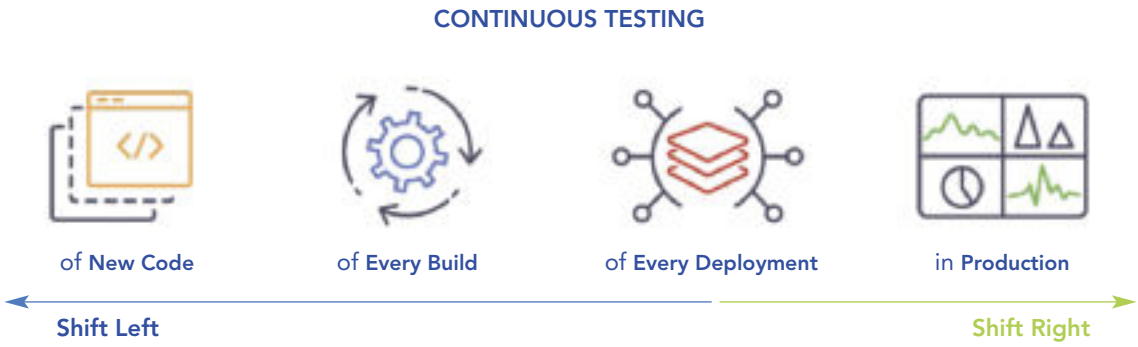
- On investit massivement sur les tests unitaires qui vont fournir une base solide à la pyramide, avec un feedback précis et rapide. Couplés à l'intégration continue, les tests unitaires fournissent un véritable harnais contre les régressions, indispensable si l'on souhaite maîtriser notre composant à moyen/long terme.
- En haut de la pyramide, on réduit la quantité de tests de bout en bout ou IHM au strict minimum : validation de l'intégration du composant dans le système global (configuration, connectivité), composants graphiques faits maison, éventuellement quelques *smoke tests* (en guise de tests d'acceptance).
- Et au milieu de la pyramide, les tests d'intégration nous permettent de valider le composant (intégration interne) et ses frontières (intégration externe). Le principe est là encore de privilégier les tests internes au composant (en isolation du reste du système) par rapport aux tests externes, plus complexes à mettre en œuvre.

8- Shift Left Testing.

Le Shift Left testing est une approche utilisée pour accélérer les tests de logiciels et faciliter le développement en déplaçant le processus de test à un stade antérieur du cycle de développement. Shift Left est une référence au déplacement des tests vers la gauche sur une timeline.

Le Shift Left est conçu pour proposer un meilleur modèle pour le développement de changement de gauche (voie rapide) car les modèles de test traditionnels, qui attendent plus tard dans le cycle de développement, peuvent constituer un goulot d'étranglement.

Les tests de Shift Left traditionnels se concentrent sur les tests unitaires et d'intégration via des tests API et des outils de test modernes.



9- Les outils du marché.

Les meilleurs outils d'automatisation des tests pour 2018 :

- Selenium
- Watir
- Protractor
- Robot Framework
- Katalon Studio
- Unified Functional Testing(UFT)
- TestComplete
- Ranorex

10- Tendances du test.

Aujourd'hui, on entend souvent parler de RPA et IA appliquées aux tests.

a. Robotic Process automation RPA.

Robot Process Automation (RPA) constitue un nouvel outil puissant pour les testeurs et les utilisateurs métier effectuant des tests, notamment dans le contexte de la mise en œuvre de systèmes standard à grande échelle.

RPA peut être mis en œuvre pour automatiser tout processus répétitif effectué sur un ordinateur. Ce processus peut être lié à n'importe quel outil de bureau ou application Web .

Les 3 meilleurs outils RPA pour 2018

- Automation anywhere
- UI path
- Blue prism

b. Intelligence Artificielle appliquée aux tests.

L'Intelligence Artificielle appliquée aux tests learning est une IA basée sur des statistiques et sur l'apprentissage automatique.
L'IA est excellente pour trouver des modèles dans les données. Elle peut ensuite utiliser les modèles identifiés par vos algorithmes ML pour prévoir les tendances futures.

Voici dans quels objectifs utiliser l'IA testing :

- Automatiser la priorisation, la sélection et l'ordonnancement des tests de non-régression à exécuter pour un run donné, en utilisant l'apprentissage à partir des résultats d'exécution des tests sur la plateforme d'Intégration Continue.
- Analyser les résultats de test pour induire la gravité des tests en échec et produire des recommandations sur ces résultats (smart analytics).
- Recommander les mises à jour des tests en fonction des données de production et fournir un analytique de la couverture fonctionnelle.
- Diversifier et variabiliser les données de tests à partir de l'apprentissage couplé des exécutions de tests dans l'Intégration Continue et des traces d'usage en production du système sous test.
- Automatiser la maintenance des tests de non-régression par l'apprentissage sur l'usage du système en production et l'inférence des parcours les plus pertinents de façon itérative et incrémentale.

Les outils qui utilisent l'IA dans le test se concentrent aujourd'hui sur les tests unitaires, intégration, affichage et générations de données de test et se positionnent comme outils d'aide aux testeurs : réparation automatique des scripts, analyse de risque, etc.
Exemple des outils IA :

- INFER
- EvoSuite : IA- Test unitaire- env java
- IntelliTest for .NET

II.6- ATDD/BDD – les tests aux services de l'expression du besoin en Agile.

par Laurent Py et Bruno Legeard

1- ATDD/BDD, une adoption en forte croissance.

Les équipes de développement logiciel sont confrontées à des exigences de plus en plus fortes de rapidité, de qualité et d'innovation. Cette course effrénée pour rester pertinent pousse ces équipes à produire rapidement de nouvelles fonctionnalités, tout en s'assurant que celles-ci soient en phase avec le marché et donc avec les besoins de l'utilisateur final.

C'est dans ce contexte que l'adoption des approches Acceptance Test Driven Development (ATDD) et Behavior Driven Development (BDD) connaissent une forte croissance ces dernières années.

> ***Un peu d'histoire.***
>
> On doit l'approche BDD à Dan North, qui l'a popularisé dans un article de 2006 dans le magazine « Better Software ». L'idée de départ est de permettre une expression partagée du besoin entre Product Owner, testeurs, développeurs et analystes métier : « we decided to apply all of this behaviour-driven thinking to defining requirements. If we could develop a consistent vocabulary for analysts, testers, developers, and the business, then we would be well on the way to eliminating some of the ambiguity and miscommunication that occur when technical people talk to business people.» – *Dan North-Mars 2006.*

Les approches ATDD/ BDD ont pour objectif d'aligner tous les membres d'un projet autour d'un même objectif, en instaurant une compréhension partagée des fonctionnalités à développer. Cet alignement se crée grâce à des exemples concrets et explicites écrits sous la forme de «scénarios d'acceptation» qui complètent les User Story et les critères d'acceptation. Tel un fil rouge, ATDD/BDD permet d'assurer que le code qui sera produit correspond parfaitement à la vision définie par les équipes métier et produit.

Le but est non seulement de développer un produit de qualité mais aussi et surtout le «bon» produit, c'est-à-dire celui correspondant aux attentes et aux besoins des utilisateurs.

Plusieurs enquêtes récentes montrent une adoption très rapide de ces approches dans les équipes agiles :

– Dans l'enquête annuelle «The State of the Software Testing Profession» de l'institut Techwell, 45% des répondants indiquent utiliser une approche ATDD en 2017/2018. Elle apparaît ainsi dans le top 5 des approches de test les plus utilisées (au côté des classiques requirements based testing, risk-based testing ou exploratory testing).

– L'enquête «State of BDD» menée par HipTest pour la deuxième année consécutive, montre une progression d'usage du BDD de 38% en 2017 à 44% en 2018.

Les motivations invoquées pour utiliser les approches ATDD/BDD, et que l'on retrouve dans les enquêtes et les blogs sur le test, sont à 3 niveaux : 1/ utiliser la scénarisation des tests pour clarifier l'expression des besoins en lien avec les User Story et les critères d'acceptation, 2/ définir de façon collaborative des scénarios de test d'acceptation, validés entre les parties prenantes (Product Owner - PO, Testeurs, Développeurs et analystes Métier), donnant ainsi une compréhension commune et exécutable des fonctionnalités à développer dans l'itération, 3/ faciliter l'automatisation des tests en associant une scénarisation de haut niveau avec des mots clés d'automatisation.

Quelle différence entre ATDD et BDD ?

ATDD et BDD sont de proches cousins et visent le même objectif: créer une compréhension partagée avant de commencer le développement, en favorisant la collaboration amont au sein de l'équipe. ATDD se focalise principalement sur la création de cas de tests d'acceptation pour capturer les exigences. Le test est l'artefact pivot de l'approche. Il n'y a pas à ce jour de standard qui s'est imposé dans la façon d'écrire ces tests. BDD, en revanche, utilise le mot «behavior» (comportement) plutôt que test. L'utilisation du mot test présuppose que nous connaissons le comportement. Or, précisément BDD se focalise sur la découverte du comportement. Les comportements sont structurés en pré-conditions, actions et post-conditions pour capturer les exigences avec l'utilisation de la syntaxe Gherkin «Etant donné – Quand – Alors» qui s'est imposée dans la plupart des équipes utilisant BDD. Ces comportements/scénarios sont ensuite utilisés pour générer des tests et piloter les développements.

Les principes d'application de ATDD/BDD sont simples (mais on verra plus loin que le diable se cache dans les détails et qu'il faut être attentif aux bonnes pratiques pour obtenir les bénéfices attendus) :

a. La scénarisation des tests est réalisée lors des sessions d'expression du besoin : l'affinage des User Story et la définition des critères d'acceptation s'accompagnent d'une scénarisation des différents cas d'usage spécifiques de la feature.

b. Cette scénarisation des tests d'acceptation est collaborative, impliquant les testeurs, le PO, les analystes métiers concernés par la features et les développeurs et permettant de challenger les différents usages de façon concrètes dans différents scénarios.

c. Les scénarios sont exprimés à partir des utilisateurs types (les personnae) de façon à se projeter le plus possible en situation réelle : les exemples d'usage permettent de décrire le comportement attendu.

d. Ces scénarios sont exécutés en test d'acceptation des User Story et servent de base à l'automatisation pour les tests de régression.

Dans ce chapitre, nous présentons et illustrons deux façons différentes de mettre en œuvre ces approches :

- BDD fondé sur la structure textuelle «Etant donné – Quand – Alors».
- ATDD visuel fondé sur une représentation graphique des parcours applicatifs.

2- BDD en pratique.

L'approche BDD consiste à aligner l'équipe de développement et le métier en décrivant en amont des exemples de la fonctionnalité à implémenter. La technique «d'example mapping» va permettre de structurer la discussion amont entre PO, développeur et testeur afin d'identifier les différentes règles métier d'une User Story, les exemples associés ainsi que les questions encore ouvertes.

Dans le cas de l'application d'un distributeur de billets pour la fonctionnalité «retrait», nous avons plusieurs règles métiers :

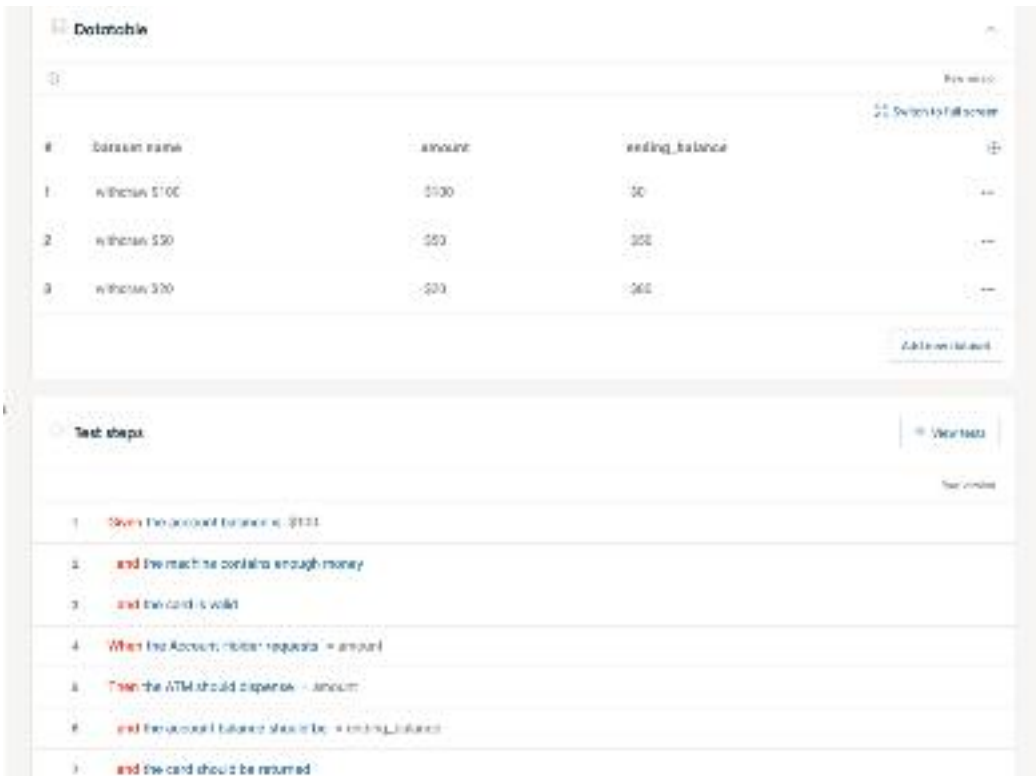
- Le compte a suffisamment d'argent.
- Le compte n'a pas suffisamment d'argent.
- La carte est invalide…

Chaque règle métier peut donner lieu à un ou plusieurs exemples sous forme de scénarios BDD.

- Feature : Account Holder withdraws cash
- Scenario : Account has sufficient funds
- Given the account balance is $100
- And the card is valid
- And the machine contains enough money
- When the Account Holder requests $20
- Then the ATM should dispense $20
- And the account balance should be $80
- And the card should be returned

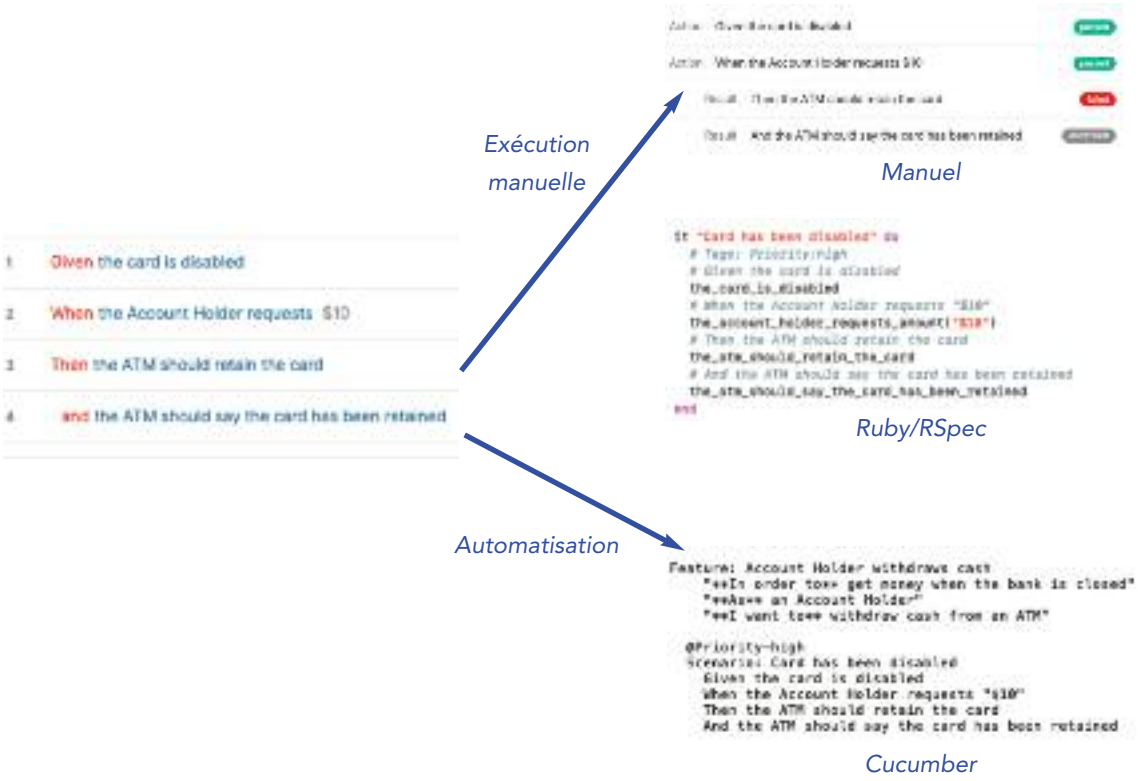
Ces scénarios sont décrits en utilisant la syntaxe Gherkin (*Given - When - Then, en français Etant donné – Quand – Alors)*. La section Given décrit le contexte, la section When les événements et la section *Then* les résultats à observer.

Ces scénarios BDD peuvent également contenir des paramètres (Scenario Outline) et une datatable d'exemples afin de générer plusieurs tests: avec un solde de $100, que se passe- t-il si je retire $20, $50 ou $100?



Une fois les différents exemples définis, le développeur peut commencer à coder avec une compréhension claire et partagée de la fonctionnalité.

Ces exemples sont généralement automatisés et permettent de guider le développement. Il n'est pas nécessaire d'utiliser Cucumber pour l'automatisation. Tout type de framework peut convenir (Ruby/RSpec, JavaScript/Jest…). Enfin, dans certains cas, l'exécution des scénarios peut être manuelle et gérée par des outils de gestion de tests.



Enfin, lorsque les scénarios sont exécutés avec succès, cela signifie que l'implémentation répond aux critères d'acceptation définis au préalable par l'équipe.

3- ATDD visuel.

L'idée de l'ATDD visuel est de scénariser graphiquement les scénarios d'acceptation pour représenter les comportements à tester dans le contexte d'un workflow applicatif.

Prenons un exemple d'application de gestion des recrutements (typiquement fondée sur un progiciel configuré), dont le processus global de recrutement est donné par la figure suivante (ici réalisée avec l'outil Yest de conception visuelle des tests).

Ce workflow donne la vision globale, partagée entre le Product Owner, les Analystes Métiers, les développeurs et les testeurs du projet. Le workflow de la phase de sélection est indiqué en bleu car c'est sur cette phase que portent les User Story de l'itération en cours, et sur lesquelles se focalisent les scénarios d'acceptation d'ATDD visuel.

Sur cette itération, considérons la User Story suivante, avec ses critères d'acceptation :

• User Story SELECTION-1 : Phase de sélection - Évaluation des candidats.

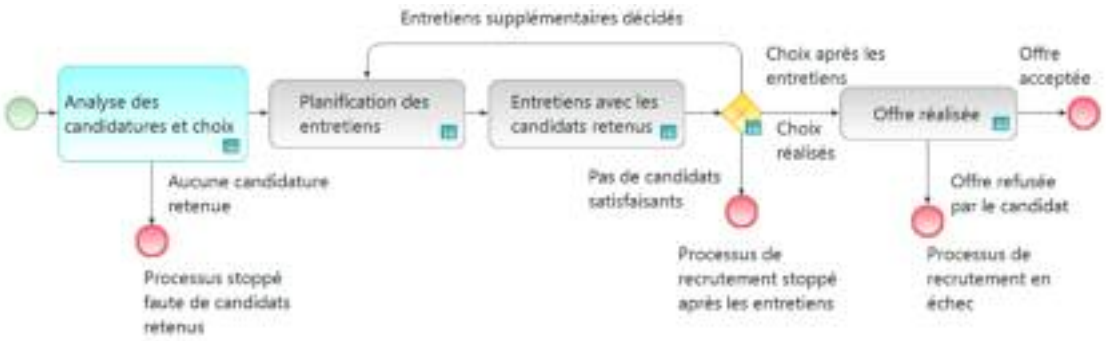
– *En tant que* Équipe RH, *je peux* suivre l'évaluation des candidats sur la base de leur CV par l'équipe Métier concernée *afin de* procéder au classement des candidats.

– Critères d'acceptation :

- Le système permet un classement des candidats sur la base de l'évaluation.
- L'option CV anonyme peut être choisie ou non.

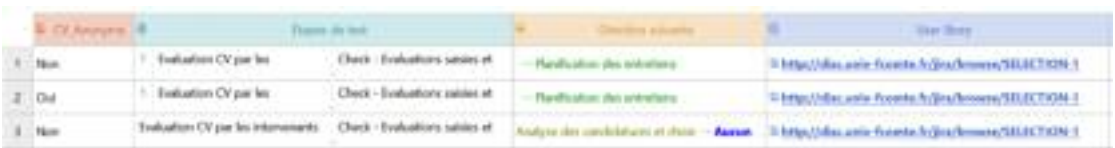
User Story dans JIRA.

La User Story, créée dans le Backlog du produit dans JIRA, est affectée à l'itération courante. Lors du raffinement du Backlog, les parcours applicatifs couvrant les User Story de l'itération sont représentés pour scénariser les tests d'acceptation. Dans le cas de la User Story HRM-1, c'est la « Phase de sélection » du workflow global qui est détaillée sous la forme du parcours wapplicatif concerné et d'une table formalisant les critères d'acceptation, les données et étapes de test et les liens de traçabilité vers les User Story.



Parcours applicatif de la phase de sélection des candidatures.

L'activité en bleu «Analyse des candidatures et choix» est celle qui correspond à la User Story SELECTION-1 vue précédemment et dont la couverture des critères d'acceptation est réalisée par la table suivante :



Chacune des lignes de la table précédente correspond à la couverture de critères d'acceptation de la User Story. Par exemple, la ligne 2 permet de tester le cas où le CV anonyme est choisi et la ligne 3 le cas d'absence de candidat, ou la possibilité qu'aucun candidat ne satisfasse la sélection sur CV.

ATDD visuel vs BDD.

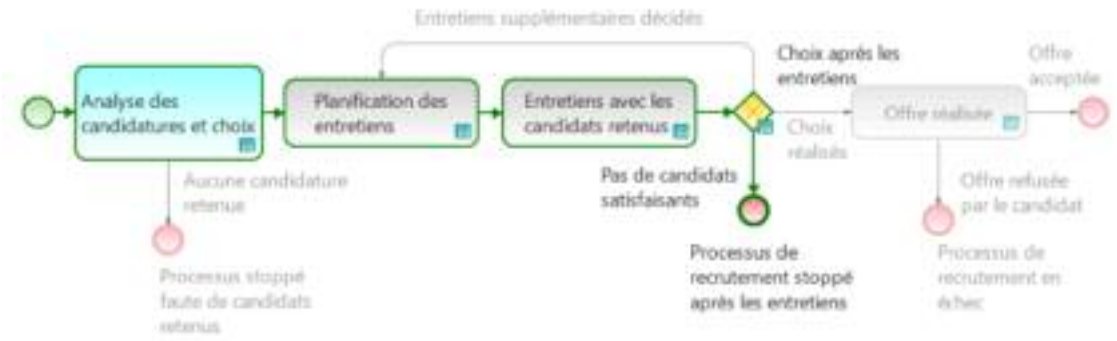
Les différences principales entre les deux approches se situent à deux niveaux :

• La représentation visuelle des parcours applicatifs sous la forme de flots d'activité Métier sur l'application avec l'ATDD visuel, plutôt que la scénarisation des comportements en langage de Gherkin «Etant donné – Quand – Alors».

• L'idée de mise en contexte de la scénarisation des tests d'acceptation est un point fort de l'approche visuelle. Il s'agit, lors des tests des User Story, de les resituer dans un workflow Métier et dans les cas d'usage pertinents du point de vue du Product Owner et des Analystes Métier.

Comme le BDD, l'ATDD visuel vise à renforcer l'alignement des membres de l'équipe projet (PO, Analystes Métier, Testeurs et Développeurs). La scénarisation des parcours applicatifs peut être portée par les testeurs, ou par le PO en lien avec les testeurs. La traçabilité est réalisée sur la User Story dans JIRA.

La génération des scénarios d'acceptation se fait par couverture des critères d'acceptation. La figure suivante montre un cas de test générés sur le parcours applicatif pour couvrir la ligne 2 de la table précédente, dans le cas où le processus s'arrête après la phase des entretiens.



Les scénarios de tests d'acceptation documentés sont exportés dans l'outil de gestion de tests, pour le suivi des tests manuels d'acceptation de l'itération et/ou générés sous forme de scripts automatisés pour l'automatisation à base de mots clés.

4- Les Do/Don't.

L'un des principaux problèmes menant à l'échec d'une transition ATDD/BDD est le manque de soutien des équipes managériales et décisionnelles. Il est nécessaire de prévoir le temps et l'accompagnement des équipes pour cette transition. C'est un investissement qui aura un coût initial et toutes les expérimentations ne seront pas fructueuses.

a. Le facteur temps sous-estimé.

Dans une approche ATDD/BDD, le Product Owner, des Analystes Métier, des représentants des développeurs et bien sûr les testeurs de l'équipe projet, vont participer aux sessions d'écriture des scénarios et tests. Ceci implique donc de libérer du temps en amont pour les équipes afin d'assurer que la vision des nouvelles fonctionnalités est partagée entre tous les membres du projet.

Cela peut parfois être perçu comme des tâches supplémentaires plutôt qu'une façon d'améliorer la qualité des nouvelles fonctionnalités. C'est pourtant cette phase de découverte du problème et de la solution (appelée 3 amigos session) qui permettra ensuite d'améliorer considérablement le processus de développement.

Comme toute nouvelle approche, il est important d'anticiper ce temps d'adaptation.

b. Le manque d'accompagnement.

Que l'adoption de l'ATDD/BDD soit une volonté de l'équipe managériale ou des équipes techniques, l'impact du changement radical de paradigme est souvent sous-estimé. Il s'agit d'un changement organisationnel et de processus.

Différentes solutions sont possibles pour anticiper et réussir au mieux cette conversion ATDD/BDD : la formation, l'accompagnement par des experts et coachs, l'intégration d'une compétence supplémentaire familiarisée avec les pratiques ATDD/BDD choisies. Il convient également de mettre en place les bons outils collaboratifs pour aider à la transition, à la fois pour le partage des scénarios et pour leur exécution via un outil d'intégration continue.

c. Des problèmes pratiques pour l'écriture et l'automatisation des scénarios/tests.

Les écueils organisationnels ne sont pas les seules embûches à surmonter lors d'une transition vers une approche ATDD/BDD. La définition des fonctionnalités et l'écriture des scénarios d'acceptation nécessitent de trouver un équilibre permettant à toutes les équipes (produit, code et qualité) d'avoir une vision commune des fonctionnalités à développer.

On retrouve généralement 2 anti-patterns chez les équipes adoptants ATDD/BDD.

- Des scénarios certes automatisés mais trop techniques écrits par les développeurs. Ces scénarios sont généralement sans intérêt pour les équipes Métier et Produit car ils ne donnent pas d'information sur le véritable comportement des fonctionnalités. Ils décrivent de façon trop précise les interactions d'un utilisateur avec le produit et ne mettent pas en valeur le cas d'usage et les avantages procurés par la fonctionnalité à l'utilisateur final.

- Des scénarios (ou des parcours applicatifs dans le cas de l'ATDD visuel) écrits par le Product Owner et trop abstraits. Ces scénarios décrivent le comportement attendu des fonctionnalités mais peuvent être difficiles à automatiser par manque de définition.

La solution est l'écriture des scénarios/tests par les équipes produit (PO) et par les testeurs et développeurs **réunis**. C'est le principe clé : la communication, l'échange entre toutes les parties prenantes. Seule cette collaboration efficace permet de créer des scénarios compréhensibles **ET** plus faciles à automatiser.

d. Des outils pas toujours adaptés.

Une autre complication freinant le changement se situe dans le manque de partage des scénarios ATDD/BDD entre les différentes équipes. Lorsque ces scénarios ont été implémentés et automatisés, ils assurent le rôle de documentation vivante de l'application.

Certaines équipes rencontrent des difficultés à trouver les outils permettant de faciliter la collaboration autour des scénarios d'acceptation. Par exemple, les scénarios écrits et stockés dans des outils de gestion de code, utilisés quotidiennement et facilement par les développeurs, sont difficilement exploitables par des équipes «produit» ou qualité.

Autre cas d'usage : il arrive régulièrement que les scénarios soient définis à l'origine par l'équipe «produit» dans un Wiki, puis dupliqués dans le dépôt de code par les développeurs. Le risque de schisme est omniprésent dans cette approche, puisque les modifications apportées dans l'un des outils ne sont pas forcément répercutées dans l'autre. L'équipe risque ainsi de perdre la documentation vivante, qui est l'un des intérêts principaux de l'approche.

e. Une approche possible pour éviter les risques du big-bang.

Pour piloter cette transition, démarrer «petit» et itérer est souvent un facteur clé de succès, permettant de capitaliser graduellement ce savoir-faire tout en construisant ses propres bonnes pratiques.

L'approche idéale consiste à tester d'abord cette méthodologie sur une simple et unique «user story» par un premier sous-groupe de l'équipe, afin que les différentes personnes impliquées dans le développement puissent adopter les bonnes pratiques et bons outils. Ensuite, la méthodologie pourra être appliquée à une seconde «User Story», impliquant des personnes ayant déjà travaillé sur la première séquence et des nouvelles personnes découvrant ATDD/BDD.

Le but est d'amener les personnes concernées à adopter progressivement l'approche, en commençant par un petit groupe de personnes, en y greffant d'autres membres dans le temps, en mixant les novices et les avertis, pour, au final, répandre les bonnes pratiques et fédérer tout le monde autour de cette nouvelle technique.

Cette méthode par tâche d'huile demande, certes, plus de temps mais cet investissement en ressources humaines et immatérielles facilite la formation des collaborateurs et l'appropriation de la terminologie métier, qui sera utilisée dans les différents projets pour l'écriture des scénarios ATDD/BDD.

5- Les bons outils ATDD/BDD.

Les outils pour l'ATDD/BDD combinent les outils pour la conception des tests, le test management et les frameworks d'automatisation adaptés à ces approches.

a. Les plateformes BDD.

HipTest et **BehavePro** sont deux solutions spécialisées permettant aux équipes agiles de concevoir de façon collaborative les scénarios BDD. HipTest est une solution indépendante s'intégrant également avec Jira. Behave Pro est un plug-in natif Atlassian disponible dans Jira. Toutes deux proposent une édition avec réutilisation des steps, afin de favoriser l'émergence d'un langage commun au sein de l'équipe. Ces deux solutions s'intègrent à Git et proposent une documentation vivante, générée à partir des tests.

b. Les outils de test management.

De nombreux outils de test management intègrent désormais un premier niveau de support BDD. Cela se limite généralement à l'édition de tests avec la syntaxe Gherkin. Parmi les solutions les plus populaires: **Zephyr for Jira, TestRail, Xray.**
Il est possible de télécharger les scénarios BDD pour l'automatisation mais aucune de ces solutions ne propose d'intégration Git ou de documentation vivante.

c. Les frameworks d'automatisation basés sur Gherkin.

Cucumber, Specflow, Behat, Behave, JBehave sont les frameworks BDD les plus populaires utilisés par les développeurs pour l'automatisation des tests BDD. Ces frameworks supportent nativement la syntaxe Gherkin.

d. Les outils de conception des tests pour l'ATDD visuel.

Yest de Smartesting, **ARD** de CA et **MaTeLo** de All4Tec sont des exemples de solutions permettant de décrire les parcours applicatifs à tester sous une forme graphique, de préparer les données de test et de produire à la fois les tests manuels documentés dans l'outil de test management et de générer les scripts dans le format du framework d'automatisation.

Il est aussi possible, avec ces outils, de faire le lien entre ATDD visuel et BDD en produisant des tests au format Gherkin, à partir des scénarios d'acceptation issus des parcours applicatifs.

6. Conclusion.

Les approches ATDD/BDD sont en train de changer la façon de produire du logiciel : définir le besoin, c'est définir en même temps la façon d'accepter la solution (c'est-à-dire le produit) apportée au besoin.

Le séquencement et la séparation historique entre les phases d'analyse du besoin, de développement et de test sont très peu efficaces, car cela oblige chaque phase à découvrir le sujet.

*Définir le besoin et la validation du besoin **en même temps** est un grand progrès !*

Les approches outillées ATDD/BDD qui permettent le partage des scénarios d'acceptation et leur automatisation, permettent d'incarner cette promesse de façon efficace.
Mais comme vu en section 4, ce changement des pratiques (comme tout changement) doit être réfléchi, sponsorisé par le management et organisé avec l'ensemble des acteurs du projet en Agile.

II.7- Les tests dans SAFe®

Par Mette Bruhn-Pedersen (traduction par Bruno Legeard)

1- Introduction.

L'Agilité en tant qu'approche du développement de systèmes logiciels a évolué depuis sa popularisation dans les années 2000. Les organisations ont généralement introduit l'Agilité par étapes, en commençant par des équipes agiles de petite taille, positionnées de façon indépendante. Dans les grandes organisations, la nécessité d'une collaboration entre les équipes agiles, qui contribuent au système d'information global, a donné naissance à différentes formes de coordination.

La phase suivante, qui concerne de nombreuses entreprises aujourd'hui, consiste à rendre plus Agile l'environnement dans lequel les équipes agiles coordonnées opèrent. Il s'agit notamment d'aligner le niveau stratégique sur les principes agiles. Ainsi, la gestion de portefeuilles applicatifs, la gestion des ressources humaines, la gestion des fournisseurs et d'autres domaines connexes se réorganisent pour devenir plus agiles.

Que signifie Agile à l'échelle ?

Scrum, qui est le Framework agile le plus populaire, décrit comment une petite équipe indépendante, peut rapidement et efficacement développer et livrer des produits et solutions de valeur. Cependant, lorsque le produit en développement est trop important pour être réalisé par une seule petite équipe, il devient nécessaire d'avoir plus d'équipes travaillant dessus pour réduire le temps de mise en production. Une autre raison pour laquelle davantage d'équipes collaborent est que le produit (ou la solution) est si complexe qu'il exige de nombreuses expertises différentes, que les membres d'une équipe agile ne peuvent pas toutes couvrir. Dans ces cas, l'Agilité à l'échelle devient pertinente. Au fur et à mesure que le nombre de personnes et la complexité de l'organisation augmentent, le guidage Scrum devient insuffisant.

L'Agilité ne concerne pas seulement le fonctionnement de l'équipe agile. Ses valeurs et principes influencent la structure et le fonctionnement de l'organisation dans son ensemble. En 2010, plusieurs propositions de Frameworks pour l'Agilité à l'échelle ont été lancées, dont Scaled Agile Framework®, en abrégé SAFe®.

Qu'est-ce que SAFe® - Scaled Agile Framework ?

Voici la façon dont Dean Leffingwell définit SAFe® dans sa version 4.6[1] :

> *«SAFe® for Lean Enterprises est une base de connaissances de principes, pratiques et compétences éprouvés et intégrés pour le Lean, l'Agile et le DevOps.»*

La vue d'ensemble SAFe® montre les différents niveaux, les cinq compétences clés ainsi que les éléments les plus importants du Framework. Si vous ne les connaissez pas, vous pouvez consulter le site *https://www.scaledagileframework.com*.

[1]: *Welcome to Scaled Agile Framework® 4.6 !, Scaled Agile®, www.scaledagileframework.com/about/*

SAFe® se compose de quatre niveaux : Équipe, Programme, Solution globale et Portefeuille. Il est configurable.

La plupart des organisations commence avec la configuration de base de SAFe®, appelée *Essentielle*, pour aider à gérer des équipes d'équipes en Agile, en utilisant principalement le niveau Equipe et Programme. Selon la taille et la nature de l'organisation, certains commencent à utiliser le niveau *Portefeuille* pour modifier d'autres parties de l'organisation. C'est ce qu'on appelle la configuration *Portfolio*. D'autres organisations utilisent le niveau *Large Solution* pour organiser des équipes d'équipes d'équipes. Pour les grandes entreprises, la configuration complète comprenant les quatre niveaux peut être utilisée.

SAFe® combine le Lean, l'Agile, le DevOps ainsi que les principes et pratiques d'autres disciplines comme la pensée systémique. Au niveau de l'équipe, il intègre des Frameworks et des méthodologies comme Scrum, XP et Kanban. Comme il s'agit d'un cadre, il fournit une bonne structure pour la mise à l'échelle, mais les organisations doivent organiser sa mise en œuvre et utiliser les pratiques qui ont du sens dans leur contexte.

Pourquoi SAFe® et pas un autre cadre ?

Outre SAFe®, plusieurs Frameworks pour l'Agilité à grande échelle ont vu le jour, par exemple[2] :

- Disciplined Agile Delivery (DAD)
- Large-scale Scrum (LeSS)
- Nexus™ [3]
- Scrum@Scale®

La raison pour laquelle ce chapitre est focalisé sur SAFe® est qu'il a gagné en popularité et est cité comme la méthode actuellement la plus populaire pour l'Agilité à grande échelle.

2- Défis typiques pour les activités de test dans l'Agilité à grande échelle.

Comme décrit précédemment, l'Agile à l'échelle se produit lorsque plusieurs équipes construisent et livrent ensemble, un produit ou une solution. Pour les clients et les utilisateurs, le produit est censé fonctionner de manière cohérente et homogène. La façon dont le produit est fabriqué n'a pas d'importance. Mais il faut trouver une organisation efficiente entre les activités propres aux équipes et les activités inter-équipes.
Ainsi, l'Agilité à l'échelle n'est pas seulement liée aux activités d'assurance qualité (QA) et de test, mais à toutes les activités tout au long du cycle de vie du développement logiciel.

Des activités complexes de QA et de tests concernant :

- Des activités qui couvrent plusieurs User Story qui, ensemble, forment une fonctionnalité.
- Les niveaux de test plus élevés tels que les tests système, les tests d'intégration système, les tests de bout en bout et les différents types de tests d'acceptation.
- Tous les types de tests fonctionnels, non fonctionnels et de conformité.

[2]: La liste n'est pas exhaustive
[3]: 12e Rapport annuel sur l'état d'Agile, VersionOne Inc, 09-Avr-2018, https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report

L'une des difficultés qui est le plus souvent évoquée est de savoir comment gérer l'assurance qualité et les activités de tests qui couvrent plusieurs équipes de façon transverse. Même si les autres équipes travaillent également de manière agile, il est parfois complexe de planifier, coordonner et réaliser des activités de QA et de test en collaboration avec les autres équipes. Sans surprise, les difficultés augmentent en général lorsque certaines des équipes font partie d'une autre entreprise.

En effet, d'une part, les équipes agiles sont responsables de leurs résultats, de la gestion de leur travail et de l'obtention de résultats qui répondent aux besoins et aux attentes des clients. Cela comprend la définition, la réalisation, le test et le déploiement des User Story et de features dans des environnements de pré-production et de production. D'autre part, les équipes agiles peuvent dépendre de systèmes ou de services fournis par d'autres personnes ou équipes de l'organisation ou d'autres entreprises. Les équipes agiles ne sont ni responsables ni garantes de ces systèmes ou services. C'est là que le problème de la responsabilité à l'égard des résultats et des activités en commun peut se poser.

La gestion de l'assurance qualité et des activités de test couvrant plusieurs équipes comprend l'identification, la planification et la réalisation des activités de test. Si le rôle classique de Test Manager n'existe pas ou a été supprimé dans le cadre de la transition vers l'Agilité (ou vers l'Agilité à l'échelle), il faut aussi s'entendre sur qui est responsable et qui est garant des différentes activités de QA et de tests.

Avant de discuter de la façon de gérer les tests inter-équipes à l'aide de SAFe®, il est important de comprendre ce que nous attendons des équipes agiles dans ce contexte.

Responsabilités des équipes agiles en matière de QA et de tests.

Dans SAFe®, une équipe agile se compose de développeurs, de testeurs, d'un Scrum Master et d'un Product Owner qui possèdent ensemble les compétences et les connaissances nécessaires pour délivrer la valeur attendue.
L'une des valeurs fondamentales de SAFe® est la qualité intégrée : c'est-à-dire une culture générale de la qualité dans l'organisation, qui couvre cinq aspects spécifiques : flux, qualité de l'architecture et de la conception, qualité du code, qualité du système et qualité de la livraison[4].

Plus concrètement, les équipes agiles sont censées utiliser des pratiques telles que Test Driven Development (TDD), Behaviour Driven Development (BDD), Specification By Example (SBE), Acceptance Testing Driven Development (ATDD), la gestion de configuration et la gestion des données de test, le bouchonnage (mocks), l'automatisation des processus et les pratiques des tests en Agile plus généralement[5].
Pour appuyer les équipes agiles, SAFe® dispose d'une équipe agile spécifique appelée équipe Système. La responsabilité première de l'équipe Système est de construire et de maintenir l'environnement global de développement agile, y compris les outils. Souvent, l'équipe est également responsable de l'intégration de la solution et des tests de bout en bout. Cela peut inclure des tests non fonctionnels et la création, l'exécution et la maintenance de suites de tests automatisées[6].

[4]: Built-In Quality, Scaled Agile®, https://www.scaledagileframework.com/built-In-quality
[5]: Team and Technical Agility, Scaled Agile®, https://www.scaledagileframework.com/team-and-technical-agility
[6]: System Team, Scaled Agile®, https://www.scaledagileframework.com/system-team

Si les pratiques de base en matière de qualité en Agile ne sont pas en place au niveau des équipes agiles, il sera difficile de gérer correctement les activités d'assurance qualité et de tests inter-équipes. Dans ce cas, les équipes agiles pourront être en difficulté pour délivrer les fonctionnalités attendues dans un sprint et n'auront pas le temps de faire quoi que ce soit d'autre.

Pour de nombreuses équipes dans l'Agilité à l'échelle, l'apprentissage, l'utilisation et l'obtention de la valeur des pratiques mentionnées ci-dessus est un cheminement qui peut être difficile. Mais avec SAFe®, les équipes agiles sont guidées. SAFe® utilise le principe intitulé Agile Release Train (ART[7]) et Solution Train pour aider les équipes et les organisations à créer des synergies entre les équipes et à gérer les dépendances entre elles. Avant d'entrer dans les détails, il est important de comprendre ce qu'est un Agile Release Train.

Synchroniser avec le principe du Agile Release Train (ART).

Dans SAFe®, l'unité organisationnelle, qui orchestre la collaboration de plusieurs équipes agiles et d'autres contributeurs est appelée Agile Release Train (ART). C'est une organisation virtuelle composée de 50 à 125 personnes. Outre les équipes agiles, l'ART comprend les rôles suivants pour aider à maintenir le train sur la bonne voie :

- Product Manager,
- System Architect/Engineer,
- Release Train Engineer,
- Business Owners,
- Et bien sûr les clients.

Cette organisation transverse inclut aussi, en général, une équipe Système, certains services partagés (rôles spécialisés non requis à temps plein) et parfois diverses Communautés de Pratiques[8].

L'ART a de nombreuses caractéristiques qui sont similaires à celles d'une équipe agile. Il est entièrement multifonctionnel, ce qui signifie que toutes les personnes, compétences et connaissances nécessaires pour développer, exploiter et maintenir un produit ou une solution font partie de l'ART. La plupart des personnes sont affectées à temps plein et peu d'entre elles peuvent être partagées entre plusieurs ARTs. Comme une équipe agile, l'ART est auto-organisé et responsable de la planification et de l'exécution de toutes les activités nécessaires. L'ART fonctionne en itérations plus longues appelées "Program Increment - PI" et comporte des événements de base ou des activités similaires à celles d'une équipe agile[9] :

[7]: *Nous avons choisi de conserver en anglais ce nommage spécifique à SAFe®*
[8]: *Agile Release Train, Scaled Agile®, https://www.scaledagileframework.com/agile-release-train*
[9]: *Note du Traducteur - Nous avons volontairement laissé en anglais les figures de ce chapitre pour éviter de distordre le vocabulaire couramment utilisé par les équipes (évidemment au risque d'un jargon franglais).*

Comme cela apparaît dans le tableau précédent, SAFe® propose plusieurs types d'événements pour la synchronisation des équipes agiles : Scrum of Scrums est la réunion des Scrum Masters, PO sync est la réunion des POs et ART sync combine les POs, Scrum Masters. L'équipe Système participe à ces événements.

3- Les tests dans SAFe®.

Les sections suivantes traitent des questions liées à la planification des tests, à la coordination des tests, à l'organisation de la QA et aux responsabilités en matière de tests.

Les approches du test et leur bonne prise en compte.

La qualité du système livré s'établit dès le début lorsque l'on définit les attendus sur l'ensemble des niveaux. Les Epics, les indicateurs sur le système et les exigences non-fonctionnelles définissent la qualité attendue et influencent les tests à réaliser. Il est également judicieux de préciser dès le départ les approches de tests qui seront mises en œuvre et de réfléchir en amont sur la façon de garantir une bonne testabilité du système. Valider la définition de ce que l'on attend du système fait aussi partie des tests. Dans ce contexte d'Agilité à l'échelle, les approches peuvent cependant différer du test fonctionnel «classique» et impliquer différentes compétences. Avoir un rôle de type «Enterprise Architect», avec une bonne compréhension des enjeux, de la qualité et des pratiques de test au niveau du portefeuille applicatif, est un gage de succès.

Intégration de la planification des tests dans la planification PI.

Comme plusieurs équipes peuvent travailler sur une même fonctionnalité, la planification PI permet les échanges sur les tests système, les tests d'intégration de systèmes et même les tests de bout en bout. Les tests d'intégration de systèmes et les tests de bout en bout peuvent nécessiter une collaboration avec d'autres ARTs ou des parties externes. Si les représentants de ces parties ne sont pas présents ou s'il n'est pas possible de s'entendre sur le travail requis, une partie de la planification des tests peut être considérée comme un risque et demander une coordination spécifique pendant l'exécution du PI. Le tableau du programme (Program Board[10]) peut également contenir une zone séparée pour chaque partie externe, ce qui rend les dépendances visibles et compréhensibles pour tous les membres de l'ART.

Dans la figure ci-dessous, voici un exemple de la façon dont vous pouvez étendre votre Program Board avec un couloir dédié pour les activités de test qui couvrent plusieurs équipes. Les «étiquettes vertes» indiquent qu'une activité de test coordonnée doit avoir lieu avant que les fonctionnalités ne soient publiées et utilisées par les clients. Certaines fonctionnalités peuvent être implémentées de manière indépendante par une équipe. Ces fonctionnalités peuvent, dans ce cas, faire partie ou non d'une activité de test inter-équipes.

Au cours de la planification PI, les activités de test sont définies et planifiées. Si les lignes rouges sur les «étiquettes orange» viennent de la droite, vous avez un problème car la fonction n'est pas prévue pour être finalisée avant que le client n'en ait besoin. Dans ce cas, il faut analyser la façon de faire différemment ou encore déplacer la version ou le jalon Métier.

[10]: Note du traducteur – Nous avons conservé le terme en anglais, actuellement, c'est ainsi que cela est pratiqué le plus souvent.

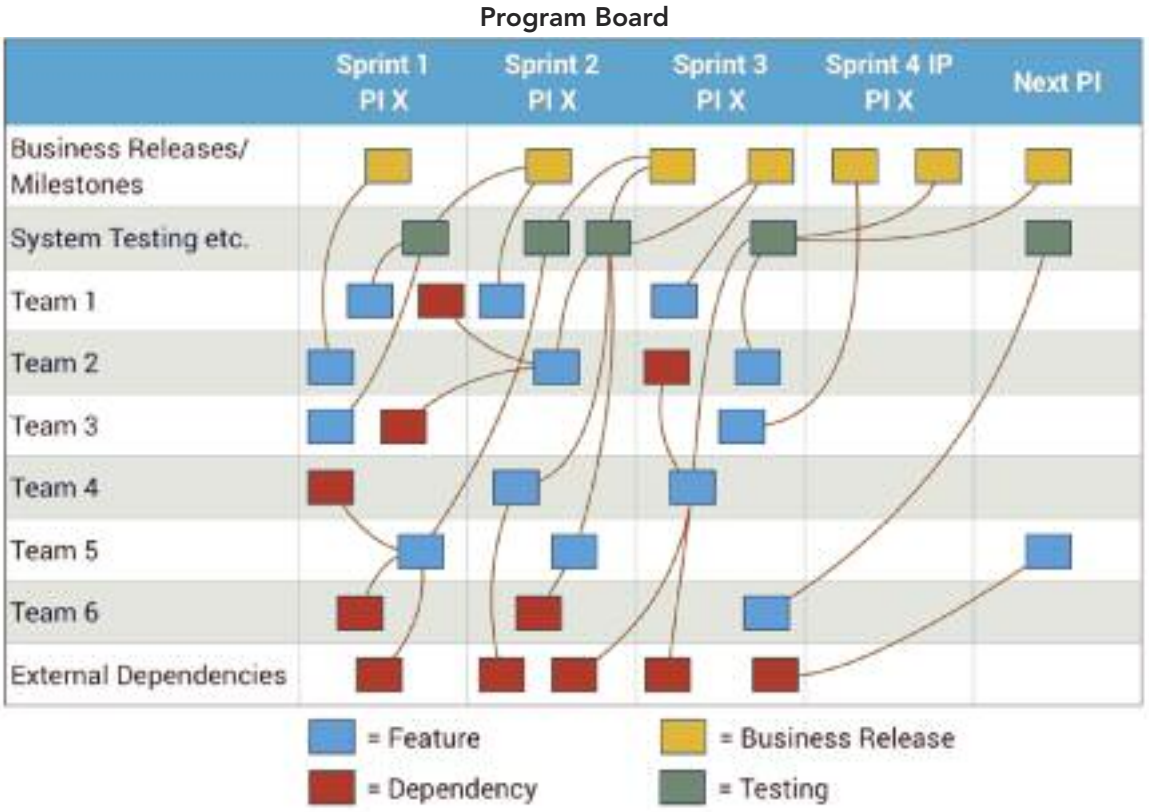
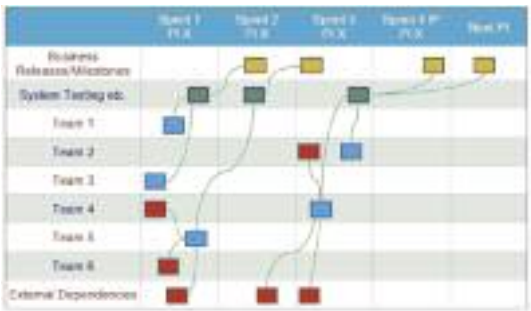
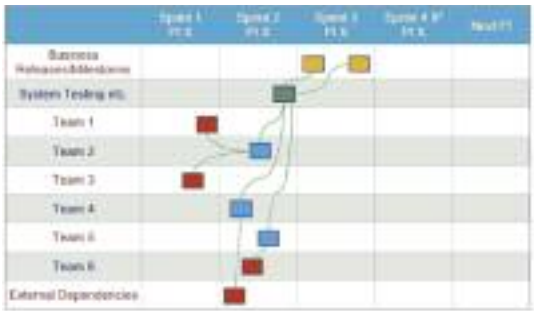


Figure 1 - Exemple de Program Board

Voici des extraits du Program Board ci-dessus montrant les différentes implications dans les activités de tests inter-équipes et des exemples de dépendances entre activités.





La ligne dédiée aux activités de tests inter-équipes (indiquée « System Testing etc. » dans les figures précédentes) peut également aider les équipes agiles à planifier les moments où elles doivent interagir avec d'autres équipes, projets ou parties externes qui pourraient avoir un calendrier fixe pour leurs activités. En ajoutant une carte séparée pour ces derniers dans le couloir des tests inter-équipes, l'ART peut mieux définir ses activités de test dans la planification du PI. Si l'ART dispose d'une équipe Système dédiée, vous pouvez utiliser l'une des lignes de l'équipe pour ce même usage. Cependant, les autres fonctionnalités fournies par l'équipe du système sont ensuite mélangées aux activités de test.

En incluant la planification des tests dans la planification de PI, les équipes agiles et tous les membres de l'ART peuvent s'assurer que les activités de qualité intégrée sont planifiées et que la confiance des membres de l'ART dans l'organisation globale augmente.

Coordination de haut niveau des tests durant le PI.

Pendant le PI, chacun doit faire ce qui a été convenu lors des différents événements (tels que PO sync, ART sync, Scrum of Scrums). En fonction de la fréquence et du rythme des équipes agiles, il peut être judicieux que d'autres parties prenantes soient présentes et participent activement lorsque c'est nécessaire. Il peut s'agir de gestionnaires de versions, d'architectes/ingénieurs système et de représentants d'autres parties prenantes dont sont tributaires les équipes agiles. Si vous avez des dépendances externes, il peut être utile de les inviter à participer de temps en temps. Une autre solution consiste à confier à l'une des équipes agiles la coordination avec la partie externe.

Puisque vous avez inclus les activités les plus importantes de tests inter-équipes dans votre Program Board, il est essentiel de signaler les problèmes, retards et risques liés à ces activités et réalisations de test. Cela peut se faire dans des réunions spécifiques.

Coordination fine des tests pendant le PI.

Parce que les tests inter-équipes peuvent être complexes à réaliser, avoir des synchronisations spécifiques pour cela entre les équipes agiles est nécessaire. Voici un exemple de la façon dont la mise en place d'une Communauté de Pratiques (CdP) sur les tests peut être utilisée pour l'alignement des équipes au niveau des tests.

Dans un ART, nous avons décidé d'avoir une Communauté de Pratiques pour les tests (appelée CdP Testing). Nous avons convenu qu'au moins un membre de chaque équipe participerait. Certaines équipes ont envoyé leur testeur, d'autres ont envoyé leur Scrum Master, un développeur ou un analyste Métier. C'est aux équipes de décider et c'est l'intérêt des personnes participantes pour la qualité et les tests qui était important, pas le rôle.

Le groupe s'est réuni chaque semaine pour discuter plus en détail de la planification et de l'exécution des tests, à la fois dans le cadre d'un ART et pour chaque équipe. L'ART effectuait des itérations de 2 semaines. La synchronisation a été effectuée au milieu de la semaine. Au cours de la première semaine, les équipes agiles ont planifié leur sprint et ont ensuite utilisé la synchronisation des tests pour résoudre les problèmes liés à leur planification et dépendances de test. Au cours de la deuxième semaine, les équipes agiles ont discuté de l'état d'avancement des tests et ont pu coordonner les tests restant à réaliser.

Les activités de la CdP Testing comprenaient aussi le partage des connaissances et des discussions sur les méthodes de test.

JOUR	SEMAINE 1	SEMAINE 2
Lundi	Planification du sprint équipe	
Mardi	Planification du sprint équipe	
Mercredi	Testing Sync	Testing Sync
Jeudi		
Vendredi		Revue & Démo du Système

Figure 2 - Exemple de synchronisation des tests inter-équipes

4- Réorganisation de l’assurance qualité et des responsabilités de test.

SAFe® propose que l’équipe Système soit responsable des tests Système, des tests d’intégration de systèmes, des tests de bout en bout et des tests non-fonctionnels, en collaboration avec les autres équipes agiles de l’ART. Lorsque les tests inter-équipes s’avèrent complexes et que vous n’avez pas d’équipe Système, il est fort utile d’en mettre une en place. Les équipes dédiées sur un composant ou une fonctionnalité peuvent difficilement traiter ces tests inter-équipes. Leur priorité est sur la livraison des fonctionnalités dont l’équipe est responsable et ces équipes peuvent manquer d’expérience et de savoir-faire pour les tests inter-équipes.

Si vous avez une équipe Système mais que l'infrastructure de développement est peu mature, l'équipe Système pourrait ne pas avoir la bande passante nécessaire pour effectuer les tests. Elle sera trop occupée à créer des environnements, à construire des pipelines de build automatisé, etc. Dans ce cas, les autres équipes agiles doivent assumer la majeure partie, sinon la totalité, de la responsabilité des tests système, des tests d'intégration système, etc. Comme alternative, certaines organisations choisissent d'avoir une équipe de test séparée pour compléter l'équipe système. C'est un bon moyen de passer progressivement d'une organisation de test plus «classique» à une organisation ART virtuelle. Cela donne aux personnes impliquées le temps de développer l'équipe, les connaissances et les compétences au sein de l'équipe agile sans que le train ne s'arrête.

SAFe® suggère également d'avoir des personnes possédant une expertise spécifique en matière d'assurance qualité et de tests, en tant que service partagé utilisé par plusieurs équipes et potentiellement par plusieurs ARTs. Il s'agit de personnes qui ne sont pas affectées à plein temps à un ART. Cela peut concerner un groupe d'appui pour les tests de performance, de charge et de stress qui aide les équipes agiles à démarrer. Ce groupe peut aussi être responsable de l'infrastructure de test qui est partagée par plusieurs ARTs. Un autre exemple est celui des spécialistes de la sécurité intervenant sur plusieurs ARTs.

Bien que SAFe® ne mentionne pas explicitement le rôle de Test Manager ou de Test Lead, de nombreuses organisations en disposent. Il est primordial de déterminer si et comment ces responsabilités sont transférées à de nouveaux rôles, comment les personnes qui occupent ces nouveaux rôles acquièrent les connaissances et l'expertise nécessaires pour améliorer la qualité et les pratiques de test. Les Test Managers et le Test Lead peuvent aussi avoir besoin d'acquérir des compétences et d'assumer de nouveaux rôles en tant que membres d'une équipe agile, d'un ART ou d'une équipe tiers. Enfin, il peut être pertinent d'inclure un représentant de l'organisation «classique» des tests dans l'équipe de transformation agile. Non seulement cela aide l'organisation de test "classique" à s’adapter à l’Agilité, mais cela permet également de renforcer la culture qualité de l'organisation. En effet, la qualité est la responsabilité de tout le monde, même lorsque nous passons à l’Agilité à grande échelle.

II.8- Indicateurs agiles pour les testeurs agiles.

Par Cyril Tardieu

1- Les métriques de test pour l'Agile.

Le syllabus ISTQB propose quatre mesures à évaluer pour les activités de test.
Qu'en est-il en Agile ?

a. Avancement par rapport au calendrier et au budget prévus.

- Le testeur est un équipier comme les autres.

L'avantage en Agile, c'est que l'on fonctionne à budget fixe. Donc le testeur consomme le budget d'un équipier et cela reste stable tout au long de l'avancement du projet.
Pas d'inquiétude ici, donc.

- Le temps de cycle de test.

La seule vraie contrainte qu'il doit surveiller ici, c'est de ne pas être le goulot d'étranglement pour terminer les User Story. Comme toute bonne définition d'une User Story terminée inclut que tous ses tests doivent être passés avec succès, qu'en est-il si le testeur n'a pas encore décrit tous les tests alors que les codeurs ont tous fini leurs développement ? Rien d'insurmontable pour une bonne rétrospective mais l'approche de la stratégie de test doit quand même être remise en cause. Où en est l'automatisation ? Couvre-t-elle les bons niveaux de tests ? Et si on essayait le Behavior Driven Development ? Bref, le testeur doit être force de propositions pour ne pas se retrouver à la traîne de ses camarades qui sont généralement 4 à 8 fois plus nombreux que lui dans une équipe agile. Ici, le métrique est clair. Il s'agit du temps entre le moment où le codeur dit qu'il a fini son développement et le moment où le testeur lui fait un retour (positif ou négatif). Si ce temps est supérieur à une journée, alors il y a quelque chose à améliorer. En effet, après une journée, le codeur est passé à autre chose et se remettre dans son code va coûter cher en temps de bascule (swap time). Pour éviter ce gaspillage de temps, le testeur doit lui fournir un retour (feedback) rapide pour qu'il puisse apporter les correctifs si nécessaire. On peut, par exemple, calculer ce temps en faisant la moyenne (ou le médian) entre le moment où le codeur clôt sa tâche de développement et le moment où le testeur clôt sa tâche de test (tests en succès) ou bien ré-ouvre celle du développeur (tests en échec).

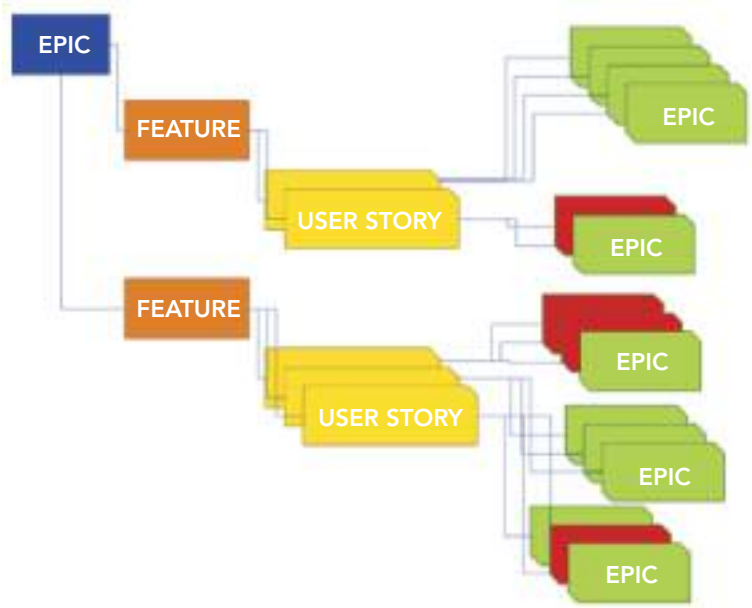
- Mini cascade.

Un écueil à éviter en Scrum, c'est que le testeur se retrouve avec toutes les User Story à tester en fin de sprint. Pour mesurer cela, il faut regarder le temps que passe le codeur à développer sa partie de la User Story par rapport à la durée du sprint. Si elle dépasse le 1/6ème du sprint (par exemple plus de 2,5 jours pour un sprint de trois semaines) alors c'est que les User Story sont trop grosses et pas assez bien découpées. Le testeur se doit de proposer de faire l'exercice *Elephant carpaccio* à la prochaine rétrospective ! Pour une meilleure efficacité, il faut viser la taille d'1/10ème du sprint. Ainsi, le testeur ne se retrouvera pas dans l'effet mini-tunnel qui reproduit le modèle de développement en cascade sur un sprint.

b. Qualité actuelle de l'objet de test.

- Taux de tests en erreur.

C'est l'indicateur classique des tests. On joue une campagne de tests et on obtient le pourcentage de tests qui ont échoué par rapport au nombre de test prévus. Mais est-ce encore possible en agile ? C'est possible, oui, mais avec une complexité supplémentaire. Non seulement nos tests doivent couvrir des fonctionnalités (features) mais également des critères d'acceptation de User Story (User Story en scrum, Work en Kanban, Story en XP). En fait, on peut même ramener cela au nombre de User Story qui ont échoué aux premiers tests. Pourquoi ? Et bien parce qu'il nous est demandé de vérifier la qualité de chaque sprint. Et comme le produit d'un sprint est un ensemble de User Story, une bonne mesure de la qualité de production est le nombre de User Story qui ont été retournées aux codeurs pour correction, car certains de leurs critères d'acceptation n'étaient pas vérifiés. On pourrait objecter ici que de toute façon, cela étant corrigé en cours de sprint, les User Story livrées à la fin du sprint sont toutes terminées avec tous les tests (re)passés avec succès. Mais cela serait manquer l'objectif principal : l'Amélioration Continue ! Effectivement, si ce taux de User Story en erreurs est élevé (au-delà de 20%), cela veut dire qu'il y a une déficience en amont. Seule une bonne rétrospective pourra trouver la cause de ce mauvais résultat.



Dans ce sprint, une Epic est développée grâce à deux Features découpées en cinq User Story, elles-mêmes couvertes par quinze tests. On peur ici remonter un taux de test en erreur de 4/15 soit 26,7 %. Mais également un taux de User Story en erreur de 3/5 soit 60 %. On pourrait pousser le vice jusqu'à dire que 100 % des Features sont en erreur ! Il est temps de faire la rétrospective !

- Taux de retests en erreur.

Ce métrique est plus difficile à capturer mais il est un grand révélateur de dysfonctionnement. C'est le nombre de tests qui ont été corrigés mais qui sont encore en échec. Si le correctif ne fonctionne toujours pas, alors il faut remonter à la source du problème grâce à une analyse des causes racines. Par exemple, sur les quinze tests d'un sprint, quatre sont en erreur dès la première exécution. Les développeurs reviennent en disant que c'est corrigé mais le testeur découvre que deux de ces quatre tests échouent encore. Alors le taux de retest en erreur est de 2/4 soit 50 %.

Ce qui est beaucoup trop. La rétrospective mettra surement en avant une mauvaise compréhension des critères d'acceptation par les équipiers lors des séances d'affinage. Il est temps de se mettre à l'Example Mapping !

- Taux de régression en erreur.

Ce n'est pas parce qu'un test est passé avec succès dans un sprint qu'il le sera encore dans les sprints suivants. Tous les testeurs le savent et c'est pourquoi ils mettent en place des campagnes de test de régression. Elles vérifient que les tests restent bien verts. L'Agilité ne faisant pas exception, tout test qui passe du vert au rouge est un signe qu'un effet de bord a cassé quelque chose qui fonctionnait avant. Si votre batterie de tests de régression contient 200 tests et que 5 tests sont passés au rouge lors d'un sprint, alors le taux de régression en erreur est de 2.5%. Cela peut paraître faible, mais cela peut aussi vouloir dire que 5 Features ne marchent plus ! Et comme ce sont les Features les plus importantes qui sont développées en premier, cela implique que des fonctionnalités critiques sont probablement impactées. Il faut vite retrouver le code qui a mis ces tests en erreur et le corriger, voire le retirer. Bien entendu, ces tests doivent être automatisés et tourner tous les jours mais cela sera abordé plus loin.

- Dette technique.

Il est important de bien surveiller la dette technique et particulièrement en agile. Comme il est très tentant de prendre des raccourcis pour finir un sprint à temps, le testeur doit rester vigilant sur la perte de qualité de code que cela peut entraîner. Heureusement, il existe aujourd'hui de très bons outils comme SonarQube, qui donnent des indicateurs factuels quant aux nombres de jours de travail nécessaires pour revenir à un code propre. Ils listent aussi précisément les lignes de code défectueuses, sujettes à plantage et les bonnes pratiques pour y remédier. Le testeur doit être capable d'expliquer que réduire la dette technique n'est pas là juste pour avoir du joli code mais pour qu'il soit maintenable dans le temps et que les fonctionnalités qui seront ajoutées ne coûtent pas de plus en plus cher à cause de ce code incompréhensible. Présenter à la revue de sprint la tendance de la dette technique permet de réaliser si la qualité du code s'améliore ou bien s'enfonce dans les eaux boueuses du code spaghetti.

- Performances.

Les tests de performances (ou dit de charge, d'endurance ou de stress selon leur point d'attention) ont parfaitement leur place dans un projet agile. Seulement ici, ces tests doivent faire partie intégrante de l'Intégration Continue. Impossible de faire des campagnes de tirs qui durent plusieurs semaines. Il faut qu'une campagne de tir puisse être jouée toutes les nuits sur le nouveau code afin de détecter rapidement les baisses de performance éventuelles. Les outils de test de performance d'aujourd'hui sont parfaitement adaptés à ces besoins. Il faut avoir défini les seuils d'acceptance des temps de réponse et autres contraintes non-fonctionnelles dans les critères d'acceptation des User Story (si spécifiques) ou de la définition de fini (si générales). Ainsi, on garde une réactivité à la journée en cas de dégradation de performance.

c. Pertinence de l'approche de test.

Il est souvent difficile de donner des indicateurs parlants aux non-spécialistes du test. Trouver les bons mots est important pour que tous les intervenants se rendent bien compte du travail effectué par le testeur.

- Taux de couverture.

Dans un sprint, la couverture sera mesurée sur les User Story. Toutes les User Story devant être testées, l'objectif est donc une couverture des User Story de 100%. Mais toutes ne le sont pas par des tests formels (manuels ou automatisés). Si une User Story a une valeur métier haute, alors l'investissement dans des tests formels est rentable. Ces tests seront repris dans les sprints suivants pour éviter toute régression sur ces parties critiques. Mais il faut aussi rattacher ces tests aux Features qui sont développées par ces User Story. En effet, une User Story a une durée de vie d'un sprint puis elle part à la poubelle. Mais les fonctionnalités demandées restent. Et les tests restent aussi ! Mais pour que ces tests aient un sens fonctionnel, il faut qu'ils couvrent les Features de l'application. Pour ce qui est du produit, c'est le taux de couverture des Features qui compte. Une manière élégante pour résoudre cela en agile est la documentation vivante. En faisant en sorte que les tests soient aussi les règles métier décrivant les fonctionnalités, le rapport d'exécution des tests devient la documentation de votre application. A ce moment-là, un taux de couverture de 100% est nécessaire et suffisant. Et cet indicateur vous permet de savoir si cet objectif est atteint, voire atteignable, dans les délais imposés par la boite de temps du sprint.

Pour la régression, le testeur va s'appuyer sur le taux de couverture des tests automatisés pour couvrir les Features des sprints précédents. Mais aussi sur des tests manuels formels ou exploratoires, sur les parties qu'il juge critiques ou bien susceptibles de défaillance (regroupement des défauts). L'intelligence et l'expérience du testeur sont encore de mise.

- Taux d'automatisation (TU et TS).

En agile, il ne faut pas se poser de question sur ce sujet. Tous les tests formels doivent être automatisés. Il est impossible de suivre le rythme des sprints avec des tests formels manuels, ne serait-ce qu'en termes de régression. Il faut donc avoir dès le départ une stratégie d'automatisation sur tous les niveaux de test en respectant la pyramide des tests automatisés. Pour les tests unitaires, ce taux d'automatisation représente le nombre de lignes de code couvertes, l'objectif étant entre 80% et 100%. Pour les tests systèmes, le taux d'automatisation dépendra beaucoup de la maturité de l'équipe. Si le testeur est le seul à le faire, alors le taux sera compris entre 20% et 50%, le reste étant l'objet de tests manuels. Si tous les équipiers participent avec du Behavior Driven Development par exemple, alors l'objectif de 100% de tests automatisés est facilement atteignable, le testeur pouvant consacrer plus de temps sur des tests manuels exploratoires à haute valeur ajoutée.

- Taux de faux positifs et faux négatifs.

Il ne suffit pas d'avoir des batteries de tests pour garantir la qualité d'une application. Encore faut-il que les tests soient robustes au changement et efficaces pour trouver les anomalies. C'est là que rentrent en jeu les faux positifs (test en échec à tort) et les faux négatifs (test en succès à tort).

Le premier peut déterminer si les tests ont un faible coût de maintenance. En effet, la cause principale des faux positifs est que le test doit être mis à jour pour tenir compte de l'évolution de l'application. Si le taux de faux positifs dépasse les 5%, alors c'est qu'il y a un problème de conception des tests qui sont trop sensibles aux données, à la configuration ou à ses dépendances. Par exemple, si sur deux cents tests joués, dix sont en échec mais que sur ces dix il y en a deux qui ne sont pas dûs à l'application mais aux tests eux-mêmes, alors le taux de faux positifs est de 20% ! Une analyse fine doit permettre de voir si les bouchons sont bien en place aux bons niveaux de test et si la pyramide des tests automatisés est bien respectée.

Le second est plus subtil à mesurer. Il s'agit de déterminer si les anomalies détectées n'auraient pas dues l'être par un test dans un niveau de test inférieur. Par exemple, si un test système trouve une exception *java.runtime.exception*, c'était clairement aux tests unitaires de trouver ce genre d'anomalie. Il faut rechercher pourquoi les tests unitaires qui couvrent la partie du code ont laissé passer cette erreur technique. C'est un indicateur très important en Agile pour s'assurer que chaque niveau de test trouve bien les anomalies qu'il doit trouver. Sinon, il faut renforcer la sensibilité de ce niveau de test en revoyant sa technique de couverture. Pour réussir à calculer le taux de faux négatifs, il faut reporter les anomalies détectées dans le niveau de test où elles auraient dû être trouvées. Par exemple, trente tests d'intégration sont en échec à cause d'anomalies qui auraient dues être trouvées pendant les tests systèmes. Ces tests systèmes ont eux-mêmes cent quatre-vingt-quatorze tests en succès. Le taux de faux négatifs est dont de 30/194 = 15,4%.

- Taux de défaillance en production

C'est là le verdict suprême. Il est important pour le testeur de voir quelles défaillances sont passées entre les mailles de son filet. Cela lui permet de s'améliorer en perfectionnant sa couverture de test sur les périmètres les plus défaillants. L'objectif étant bien entendu d'obtenir un taux de défaillance en production le plus proche de zéro, ce qui est parfaitement atteignable. Une façon simple de calculer ce taux est de diviser le nombre de défaillances constatée par le nombre de jours entre deux mises en production. Par exemple, pour des mises en production de tous les sprints de trois semaines (soit vingt-et-un jours), si trois défaillances sont découvertes en production entre deux mises en production, alors le taux de défaillance est de 3/21 soit 0,14 défaillance par jour.

d. Efficacité des activités de test par rapport aux objectifs.

- Coûts versus bénéfices.

Il est toujours difficile d'évaluer concrètement la valeur ajoutée du test qui est trop souvent perçue comme un coût. La valeur métier donnée à chaque User Story peut être un bon indicateur quoique souvent relative. Lorsque le testeur trouve un bug, une manière de faire consiste à demander au PO combien cela aurait coûté de l'avoir en production. Si la somme de ces coûts évités est supérieure à ce que coûte le testeur et ses outils, alors on peut dire qu'il est rentable. Cela peut même être directement calculé par les tests automatisés qui cumulent leur valeur

d'impact à chaque fois qu'ils passent au rouge. A la fin de chaque sprint, le testeur peut ainsi montrer les économies qu'il a permis de réaliser. Mais à cela, il faut aussi ajouter le fait que l'application aurait été inutilisable en production sans ces tests et qu'ainsi, le coût des autres équipiers a aussi été sauvé.

- Coût de la sur-qualité.

Se pose toujours aussi le coût par rapport au bénéfice de l'exécution de tests supplémentaires. Combien de tests sont trop de tests ? Là aussi, l'Agilité répond par l'empirisme. On ne peut pas définir une valeur absolue à l'avance, mais la bonne quantité de test est réajustée en permanence à chaque sprint. En fonction des résultats obtenus, bons ou insuffisants, le testeur va ajouter ou supprimer des tests. Des tests automatisés qui n'ont pas trouvé de bug depuis deux sprints peuvent passer d'un rythme quotidien à un rythme hebdomadaire afin de garder un build dans les dix minutes (une pratique recommandée par l'eXtrem Programming). Ainsi, on garde une campagne de test automatisée plus exhaustive qui ne sera jouée que le weekend, son délai de Feedback étant plus long mais compensé par sa faible probabilité d'échec. L'objectif du testeur ne doit pas être d'avoir toujours plus de tests, même automatisés, mais juste ce qu'il faut de tests pour avoir la meilleure qualité à coût constant. La meilleure qualité étant bien sûr un coût de défaillance en production inférieur au sien.

- Et les rapports d'anomalies alors ?

Et bien, il n'y en a tout simplement pas en Agile. Le testeur va voir directement le codeur pour qu'il corrige le défaut et lui montre le problème. Pas le temps de remplir un rapport d'anomalie. C'est un gros changement par rapport à une méthode de développement non-agile. Cela retire du travail administratif mais prive le testeur d'un métrique important pour montrer la valeur de son travail. Il lui faudra mettre en avant plutôt les métriques présentés précédemment, surtout si les défaillances sont détectées par les tests automatisés. C'est le résultat du principe du test agile d'aider à construire un meilleur produit plutôt que de critiquer un produit fini.

2- Les métriques agiles pour le test.

Il est important que le testeur comprenne très bien les métriques utilisées en Agile et ce qu'elles signifient pour lui.

a. En Scrum.

Même si le Scrum Guide ne mentionne que les «burn-downs, les burn- ups ou les cummulative flows», il est deux métriques principales reconnues en Scrum.

- Burn-down.

Tous les scrum masters vous diront que le Burndown Chart sert à savoir si l'on est en retard pour terminer toutes les User Story du sprint : ce n'est pas le cas. Son plus grand intérêt est de nous monter si notre équipe a un processus Scrum dysfonctionnel. En effet, la courbe ne peut descendre que lorsqu'une User Story est finie. C'est-à-dire que le testeur a dit qu'elle est finie

car il a pu la tester avec succès (entre autres définitions de fini). Or, si la courbe est un escalier à trois marches et que celles-ci sont dans les derniers jours du sprint, cela veut dire que le testeur se retrouve à tout tester à la dernière minute et subit l'effet mini-cascade. La rétrospective doit mettre en lumière les problèmes techniques et organisationnels qui sont la cause de ce dysfonctionnement. Le testeur a un rôle important à jouer dans la bonne tenue de cette métrique car sinon il sera débordé tôt ou tard.

- Vélocité.

La deuxième métrique clé en Scrum est la quantité moyenne de Story points que l'équipe peut terminer en un sprint. Le travail du testeur doit être inclus dans cette mesure. L'attribution d'une valeur de Story points lors du Scrum Card Game ne doit pas venir uniquement des codeurs. Elle doit inclure la participation de tous les équipiers, testeur inclus, qui sont impliqués pour finir une User Story. Ainsi, on obtient une vision réaliste et juste de la complexité du travail à fournir. Sans cela, il sera impossible d'avoir une vélocité significative pour prédire ce qui peut être pris dans un sprint.

b. En Kanban.

Le Kanban vient des méthodes de l'industrie automobile, bien plus matures que celles de l'informatique. Elles s'adaptent très bien au développement logiciel mais demandent une certaine rigueur et une remise en question des méthodes traditionnelles.

- WIP.

Limiter le travail en cours est un facteur clé de réussite en Agile. Moins il y a de tâches ouvertes en même temps, plus le débit sera fort. Cela veut dire que l'équipe sera plus productive. Et tout cela en travaillant moins ? Pas vraiment : cet indicateur nous permet d'éviter de perdre notre temps à basculer d'une tâche à l'autre. Au-delà de deux tâches en parallèle, notre productivité s'effondre, jusqu'à atteindre 20 % pour cinq tâches simultanées. Cela veut dire ici que 80 % de notre temps peut être gaspillé en activités inutiles. Le WIP (Work In Progress) mesure le nombre de tâches commencées et par encore terminées. C'est-à-dire tout ce travail effectué qui ne rapporte encore rien ! Comme seule une tâche terminée apporte de la valeur, il faut minimiser le nombre de tâches en cours. Le testeur doit donc faire très attention à tester une seule User Story à la fois jusqu'à ce qu'elle soit finie pour de bon. Cela peut sembler très difficile car de nombreux blocages peuvent survenir. Il faut résister à la tentation de basculer sur une autre User Story et laisser celle-ci en plan en attendant que le blocage se résorbe tout seul ! Il faut, au contraire, solliciter au plus tôt ses équipiers pour résoudre ensemble le blocage.

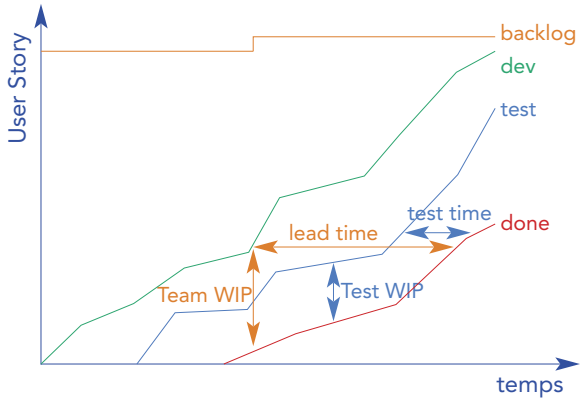
Le WIP se mesure sur la totalité de l'équipe. Le WIP total ne doit pas dépasser le double du nombre d'équipiers. Une bonne équipe aura un WIP entre x1 et x2. Une équipe performante aura un WIP inférieur au nombre d'équipiers. Par exemple, s'il y a six équipiers, un WIP de trois indique une équipe travaillant en binômes, pratiquant sûrement le pair-programming (une technique très efficace pour augmenter la qualité d'une application entre autres).

- Lead time.

Cette mesure indique le temps qu'une User Story met entre le moment où elle entre dans le backlog (le besoin est identifié) et le moment où elle est en production (l'utilisateur peut s'en servir). C'est le time to market si cher au Product Owner. Donc plus c'est court, mieux c'est. Le testeur y apporte sa contribution dans le sens où il y participe de sa productivité. Moins une User Story passe de temps en test, plus le Lead time sera court et plus vite elle pourra partir en production. Si le temps passé par les User Story en test est trop long, c'est qu'il y a des pistes d'amélioration comme l'automatisation ou une meilleure coordination avec les autres équipiers. C'est un bon objectif d'amélioration continue que de chercher à réduire le Lead time, même s'il semble convenable. C'est facilement mesurable (passer de 5h à 4h par User Story par exemple) et participe à un objectif global d'équipe. Les outils d'aujourd'hui le calculent automatiquement, ce qui facilite grandement son utilisation.

- Cumulative Flow Chart.

Le CFD est un diagramme qui montre le cumul de la progression des différentes activités de développement, le test étant l'une d'entre elles. Il permet de visualiser facilement les deux indicateurs que sont le WIP et le Lead time. Mais il donne aussi aux testeurs de précieux indices sur leur activité au sein du processus de développement.



Par exemple, si l'écart entre les trois courbes augmente avec le temps, cela veut dire que de plus en plus de tâches sont commencées mais pas finies. Il y a donc des blocages à résoudre rapidement si l'on ne veut pas se retrouver avec de plus en plus de User Story commencées mais jamais finies! Ce diagramme est très riche en enseignements et rentrer dans les détails dépasse le cadre de cet ouvrage. Le testeur doit retenir que la tendance de la courbe de test par rapport aux autres courbes lui donne des informations pertinentes et lui permettent de s'améliorer.

II.9- Le test basé sur les risques – Une approche classique s'intègre dans le monde agile.

Par Anne Kramer

1- L'enjeu.

Nous l'avons tous compris : les tests exhaustifs sont impossibles. Sauf exception, il faut faire un choix et concentrer les activités d'assurance qualité sur les aspects les plus critiques du produit ou du projet. Tandis que les méthodes agiles adressent le risque #1 de tout projet – les besoins pas clairs ou volatiles - elles ne répondent pas vraiment à la question «Quoi tester et dans quelle mesure ?». De plus, les méthodes comme Scrum ont tendance à mettre l'accent sur les tests unitaires automatisés et à négliger les tests d'intégration et de système. S'il y en a, les testeurs d'intégration et de système ont souvent du mal à suivre le rythme de changement continu qui est inhérent à chaque méthode agile.

Le test basé sur l'analyse de risque représente une stratégie de test adaptée à ces problèmes. Il permet de bien cibler les efforts d'assurance qualité. De façon générale, c'est la stratégie idéale si l'on manque de temps, d'argent ou tout simplement si le projet de test lui-même est à haut risque (par exemple, parce que les besoins ne sont pas bien définis). Il n'est donc pas spécifique aux projets agiles, mais s'intègre bien dans ce contexte.

2- Le principe.

Dans son glossaire CFTL/ISTQB des termes utilisés en tests de logiciels, l'ISTQB® définit qu'un risque est «un facteur qui pourrait résulter dans des conséquences négatives futures, généralement exprimé comme un impact et une probabilité». Pour identifier les risques, il faut donc non seulement connaître les problèmes et leurs conséquences potentielles, mais aussi évaluer leurs impacts et leurs probabilités.

Nous distinguons deux types de risque. *Les risques produit* sont directement liés à l'objet de tests. Ils portent sur la fonctionnalité, la sécurité des utilisateurs, la sécurité des données et autres exigences non-fonctionnelles. Les *risques projet* sont liés à la gestion du projet, notamment au triangle «qualité – délai – coût». En principe, chaque risque produit est aussi un risque projet, mais nous allons voir dans un instant que la façon d'évaluer ces risques diffère.

Exemples	
Risques liés au projet : • exigences changeantes • dates limites irréalistes • problèmes politiques • manque de personnel • manque d'encadrement	Risques liés au produit : • logiciel défectueux • logiciel n'offrant pas les fonctionnalités voulues • dommages corporels • manque de sécurité des données

Tableau 1: Exemples de risques produit et projet

Le test basé sur les risques est défini comme *une approche de test visant à réduire le niveau des risques du produit, informer les parties prenantes de leurs statuts et commençant dans les stades initiaux d'un projet*. Cette approche *comprend l'identification des risques du produit et l'utilisation de niveaux de risque pour guider le processus de test*. Autrement dit : chaque décision concernant la priorité et l'intensité des tests est basée sur les risques associés à la fonctionnalité en question.

Suivant la définition de l'ISTQB®, le test basé sur les risques ne porte donc que sur les risques produit. Or, un gestionnaire de test n'est rien d'autre qu'un chef de projet (de test) et devrait donc également appliquer les méthodes de la gestion de projet. Là, nous trouvons le même principe. Un bon chef de projet identifie et évalue les risques de projet pour ensuite pouvoir prendre des contre-mesures. Plus un risque est élevé, plus il est important d'agir. Idéalement, chaque risque est traité par une mesure, si ce n'est qu'une réserve de temps ou d'argent. Nous allons donc étendre le concept et inclure les risques projet dans nos considérations, tout en sachant que cela ne correspond pas tout à fait à la doctrine pure.

3- La méthode AMDEC.

Tandis que le principe du test basé sur les risques est facile, la pratique l'est moins. Comment savoir quels sont les risques et comment les aborder ? Parmi les différentes techniques d'analyse de risques, une méthode est particulièrement répandue : l'analyse des modes de défaillance, de leurs effets et de leur criticité (AMDEC).

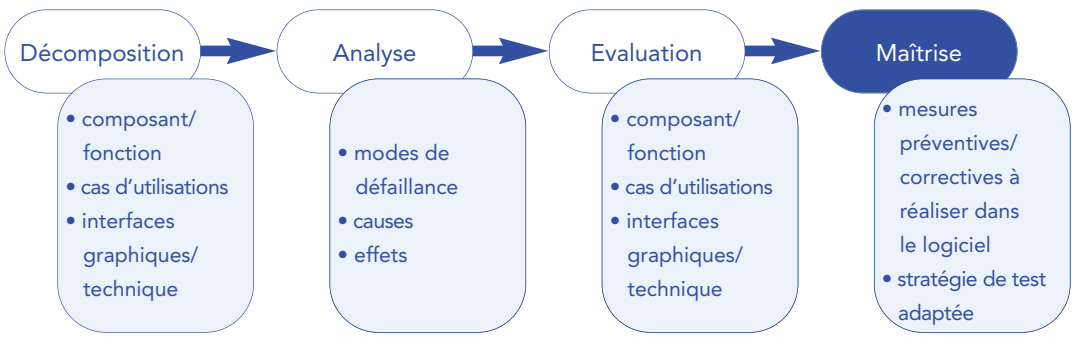


Figure 1: Les étapes de l'AMDEC

Étape 1 : Décomposition.

L'AMDEC consiste en différentes étapes : décomposition, analyse, évaluation et maîtrise. La première étape sert à établir une systématique qui nous aidera à ne rien oublier. Souvent, le système sous test est décomposé en composants et chaque composant en fonctions. La liste des fonctions sera ensuite la base de l'analyse des modes de défaillance. Le choix de la décomposition doit être adapté au système sous test (un contrôleur embarqué n'a pas d'IHM), mais il doit surtout permettre une analyse exhaustive. Par exemple, il faut également penser aux scénarios d'utilisation moins évidents comme l'installation, la maintenance ou l'archivage de données.

***Attention** : Le niveau de détail défini par la décomposition influence beaucoup l'effort nécessaire et l'efficacité de l'analyse de modes de défaillance.*

En Agile, la meilleure décomposition est évidemment basée sur les éléments du carnet de produit : épopées et récits d'utilisateurs. La méthode AMDEC peut être utilisée d'une part pour prioriser et planifier les versions (releases) et d'autre part pour gérer les activités de test pendant une itération.

Étape 2 : Analyse.

Après avoir dressé notre liste de fonctions à considérer, nous devons réfléchir aux modes de défaillance. Si l'AMDEC est appliqué dans le contexte du test basé sur les risques au début d'un sprint, il suffit de regarder les défaillances du système. Cela inclut tous les problèmes liés aux besoins non-fonctionnels comme la performance, la stabilité ou la sécurité des données. Sinon, il faut également penser à l'utilisateur comme source de défaillance. Dans ce cas, les résultats de l'AMDEC serviront pour la conception du logiciel et serons intégrés dans le carnet de produit ou le carnet de sprint.

Pour chaque mode de défaillance identifié, nous déterminons les causes et les effets. Parfois, la même défaillance peut avoir plusieurs causes qu'il est utile d'énumérer individuellement. Cela nous permettra par la suite de prioriser les causes les plus probables. Quant à l'effet, on ne considère généralement que l'effet le plus grave – sauf s'il est possible de déterminer des probabilités différentes selon la situation.

L'analyse de risques est un travail collectif effectué par plusieurs personnes. Il est important d'inclure des experts côté logiciel aussi bien que côté utilisation du système. Beaucoup d'entreprises avec une gamme de produits homogène établissent une liste de risques produit identifiés dans le passé et se servent de cette check-list comme point de départ pour les nouveaux produits. L'approche systématique de l'analyse de modes de défaillances devrait être complétée par des méthodes plus créatives comme le remue-méninge (brainstorming). Cette méthode permet d'obtenir un grand nombre d'idées en peu de temps. Grâce à l'atmosphère détendue et joyeuse, les participants s'inspirent mutuellement. À la fin, les idées farfelues sont jetées mais il reste généralement beaucoup d'idées utiles. Dans le contexte de l'analyse de risques, une variante du remue-méninge, le *brainstorming inversé*, donne également d'excellents résultats. L'idée consiste à discuter l'inverse du problème, ce qui permet de se séparer des idées préconçues du genre «cela n'arrive jamais». Au lieu de demander «Comment sécuriser nos données ?», la séance de remue-méninges porte sur la question «Comment rendre la vie du pirate informatique plus facile ?». Rien que le fait que la question soit absurde libère l'esprit. Bien entendu, les solutions proposées sont finalement à nouveau inversées.

***Attention** : Pour les risques liés à la sécurité des données, la méthode AMDEC n'est pas suffisante. Il est fortement conseillé de s'appuyer sur un modèle de menaces et de chercher systématiquement les vulnérabilités (par exemple, en suivant la classification STRIDE de Microsoft®).*

Étape 3 : Évaluation.

Afin de pouvoir prioriser les risques, il faut connaître leur criticité. Pour cela, nous allons évaluer l'impact et la probabilité d'occurrence de chaque risque.

L'impact dépend uniquement de la gravité de l'effet. La probabilité d'occurrence dépend de deux aspects : la fréquence d'apparition de la cause et la probabilité de détection du problème avant que le risque ne produise son effet. Parfois, ces deux probabilités sont évaluées séparément, mais pour plus de simplicité, nous allons combiner les deux probabilités et parler uniquement de la probabilité d'occurrence.

Prenons l'exemple d'une banque en ligne.

- ***Impact*** : Un bug dans la messagerie obligeant le client à se rendre à sa succursale bancaire est moins grave que le faux virement.
- ***Occurrence*** : Le risque de bug dans la fonction «commande de chéquier» est probablement moins élevé que le risque de bug dans la fonction «achat d'un produit» pour les raisons (fictives) suivantes :
    1. La fonction «commande de chéquier» existe depuis longtemps et la plupart des bugs a déjà été éliminée, alors que la fonction «achat d'un produit» vient d'être complètement remaniée. Il est très probable que le nouveau code contienne aussi de nouveaux bugs.
    2. La fréquence d'utilisation des deux fonctions est différente. Il n'y a probablement plus autant de chéquiers commandés alors que le produit flambant neuf est censé connaître un succès fou.
    3. Le client verra assez vite que son chéquier n'arrive pas, alors qu'il ne connaît pas vraiment bien le nouveau produit et attendra peut-être assez longtemps avant de se poser des questions.

Tout comme l'analyse, l'évaluation des risques est un travail collectif. Pour obtenir un résultat homogène, il est important de définir des critères bien précis en amont. Ces critères vont permettre aux participants d'une séance d'AMDEC d'évaluer l'impact et la probabilité de façon relativement objective. Ils sont spécifiques à chaque projet et doivent être définis par les responsables.

Les tableaux 2 et 3 montrent des exemples.

Effet	... sur le coût	... sur les délais	... sur la qualité	Valeur
Mineur	pas ou peu de surcoûts (<5% du budget)	sans impact sur la date de livraison	• limitation insignifiante d'une fonctionnalité • produit est utilisable, mais facilité d'utilisation légèrement réduite • aucune fonction critique pour la sécurité n'est affectée	1

Moyen	surcoûts sensibles, mais qui ne mettent pas le succès du projet en danger (5% < x ≤ 15%)	date de livraison en danger, mais écart accepté par le client	• atteinte à la fonctionnalité de base • produit peut être utilisé, mais facilité d'utilisation est limitée	4
Majeur	surcoûts mettant le succès du projet en danger (15% < x ≤ 30%)	date de livraison probable pas tenue; écart éventuellement pas accepté par le client	• atteinte à la fonctionnalité de base • produit ne peut pas être utilisé tel quel	7
Important	surcoût ayant de l'impact sur d'autres projets (>30% du budget)	date de livraison probable pas tenue; écart définitivement inacceptable pour le client	• produit complètement inutilisable • fonctions critiques pour la sécurité affectées • de graves dommages sont possibles	10

Tableau 2 : Critères pour l'évaluation de l'impact d'un risque (exemple)

Occurrence	…pour les risques projet	…pour les risques produit	Valeur
Exceptionnelle	ne peut pas être complètement exclu (<10%)	une ou deux fois sur l'ensemble du cycle de vie du produit	1
Rare	concevable, mais ne se produira plutôt pas (<40%)	une fois par an	4
Possible	se produira plutôt qu'il ne se produira pas (<70%)	une fois par mois	7
Fréquente	il est quasiment certain que le risque se produira	une ou plusieurs fois par semaine	10

Tableau 3 : Critères pour l'évaluation de l'occurrence (exemple)

***Attention** : La définition des critères pour l'occurrence requiert un soin particulier. Il est utile de se demander ce que « toujours ? » signifierait en chiffres absolus, car le nombre de produits vendus ou installés joue également un rôle. La probabilité que la rougeole entraîne des complications est de 1/15 000. En termes de probabilité, c'est peu, mais cela correspond toutefois à 1,6 enfant par an qui risque de mourir ou de subir de dommages irréversibles. Donc, si votre produit est dangereux, pensez aussi au nombre absolu de cas.*

Les résultats de l'analyse de modes de défaillance et de leurs effets sont généralement documentés sous forme de tableau.

Étape 4 : Maîtrise.

Une fois que les risques sont identifiés et évalués, nous pouvons réagir et définir des contre-mesures. Afin d'éviter les risques produit identifiés par l'AMDEC, le propriétaire du produit va probablement définir des récits utilisateur supplémentaires ou compléter les critères d'acceptation d'un récit utilisateur existant. Pour éviter les risques projet, différentes mesures sont imaginables, dont le test basé sur les risques qui constitue le sujet de cet article. (Bien entendu, il est également possible de prévoir un plan B soit sous forme de réserve de temps ou d'argent, soit sous forme de plan de contingence qui permet de faire marche arrière ou de contourner le problème.)

Il s'agit donc de définir une stratégie de test basée sur les risques, qui permettra de concentrer les efforts d'assurance qualité sur les risques les plus élevés.

4- La stratégie de test basée sur les risques.

L'idée de la stratégie de test basée sur les risques consiste à définir l'intensité de test à accomplir en fonction du niveau de risque. Malheureusement, il est souvent difficile de définir des critères pour l'intensité de test. Une bonne pratique consiste à appliquer des critères de couverture de classes d'équivalence, de valeurs limites, de tables de décisions ou d'éléments structurels d'un modèle.

Définir les niveaux de risques.

Pour connaître le niveau d'un risque, nous calculons son *Indice de Priorité des Risques* (IPR). L'IPR n'est rien d'autre que le produit des valeurs que nous avons associé à l'impact et l'occurrence. Prenons, par exemple, les critères définis par les tableaux 2 et 3. L'IPR d'un risque grave (= 7) mais rare (= 4) est de 28.

Reste la question : «Que faire avec un risque d'un IPR = 28 ?». Pour répondre à cela, nous devons associer chaque IPR possible à un niveau de risque et définir une stratégie de test adaptée à ce niveau.

Occurrence / Impact	1	4	7	10
1	1	4	7	10
4	4	16	28	40
7	7	28	49	70
10	10	40	70	100

Figure 2 : L'Indice de Priorité des Risques et leur niveau de risque associé (exemple)

La figure 2 montre tous les indices possibles pour les critères définis par les tableaux 2 et 3. Ils sont divisés en trois classes (les couleurs d'un feu de trafic – vert, orange et rouge). Ces trois niveaux de risque expriment notre niveau d'acceptation d'un risque. Tous les risques avec IPR inférieur ou égal à 16 sont tellement peu critiques qu'en cas de doute, nous pouvons aussi vivre avec alors que les risques avec IPR supérieur à 28 sont inacceptables (trop grave, trop probable) et justifient un niveau élevé d'assurance qualité.

***Attention** : Les tableaux 2 et 3 ainsi que la figure 2 sont des exemples. Les critères et les niveaux de risque dépendent fortement du contexte. Chaque organisation doit définir ses propres critères. Il est également possible de travailler avec d'autres chiffres et plus de niveaux de risque.*

Une stratégie de test basée sur les risques définit des règles claires pour la conception des cas de tests en fonction du niveau de risque. Le tableau 4 montre comment différentes techniques de test boîte-noire peuvent être associées aux niveaux de risque que nous avons définis précédemment.

Niveau de risque	Indice de priorité	Technique de test boîte-noire à appliquer
Bas	IPR inférieur ou égal à 16	Partitions d'équivalence
Moyen	IPR entre 17 et 28	Analyse des valeurs limites
Élevé	IPR supérieur à 28	Test de tables de décision

Tableau 4: Stratégie de test basée sur les risques (exemple)

Il est relativement simple d'appliquer ces règles aux tests unitaires car leur portée est limitée à quelques fonctions et paramètres. Cependant, il est beaucoup plus difficile de garder la vue globale sur les cas de test à choisir dès qu'il s'agit des tests d'intégration et système. À cela s'ajoute une difficulté supplémentaire : les méthodes agiles favorisent les changements fréquents et le développement incrémental. Il en résulte que les testeurs d'intégration et de système ont souvent du mal à suivre le rythme des itérations. Une stratégie de test basée sur les risques leurs permettrait de bien concentrer leurs efforts, mais comment mesurer si les tests spécifiés sont suffisants ou pas ? Dans ce contexte, la visualisation des processus à tester a fait ses preuves.

Visualiser les tests pour pouvoir choisir les cas de test adaptés.

La figure 3 en page suivante montre la visualisation du processus d'un virement bancaire. Elle correspond au récit d'utilisateur suivant : «En tant que client particulier de la banque, je veux virer un certain montant d'argent sur un compte de mon choix afin de payer mes factures». Un dysfonctionnement de la fonction associée aurait un impact majeur et un bug est au moins possible. Donc, le niveau de risque est élevé et le test exhaustif s'impose.

Toutefois, il n'est pas nécessaire de tester l'abandon du virement avec n'importe quel type de compte, n'importe quel montant, avec ou sans motif. Le diagramme de flux nous permettra de définir les cas de test les plus pertinents. Avec treize cas de test, une bonne couverture du récit d'utilisateur est possible.

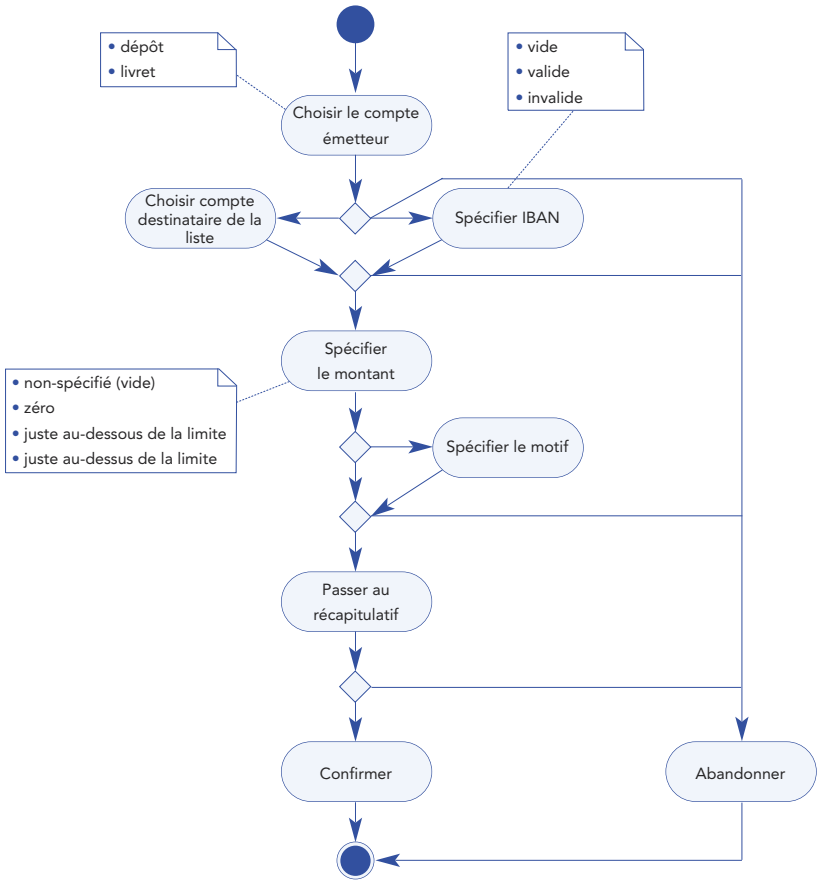


Figure 3 : Visualisation du processus d'un virement bancaire avec classes d'équivalence des paramètres

La visualisation du processus à tester est toujours utile. Dans l'Agile, elle représente une stratégie de survie pour les testeurs de système car elle permet de suivre le rythme parfois effréné du développement itératif et incrémental. Comme la vision globale ne se développe que progressivement, il est difficile de créer des cas de test système appropriés à un stade précoce. Or, la représentation graphique peut également être développée de manière incrémentale. De nouveaux cas de test peuvent être dérivés pour chaque sprint, qui reflètent l'état actuel du système.

5- Quelques conseils.

L'AMDEC est une méthode systématique et puissante. Elle s'avère utile pendant l'entretien du carnet de produit car elle permet de prioriser les récits d'utilisateurs et de spécifier les contre-mesures nécessaires à réaliser dans le produit. Elle peut également être employée pendant la planification d'un sprint pour avoir une idée plus claire de l'effort de test requis pour ce sprint.

Dans la pratique – dans le contexte agile comme dans le contexte séquentiel – l'AMDEC se heurte souvent à un certain nombre de problèmes :

- perte de temps avec des discussions des aspects trop détaillés ;
- surestimation de l'impact d'un défaut du produit par les développeurs ;
- difficulté d'estimer la probabilité d'occurrence d'un bug.

Les conseils suivants peuvent vous aider à éviter ces pièges :

- N'allez pas trop dans le détail. Restez au niveau système avec l'AMDEC. S'il est vraiment nécessaire de déterminer la fonction spécifique du logiciel qui peut causer un risque, optez pour l'arbre de causes (une autre méthode d'analyse de risques) afin de trouver les causes primaires.
- Faites évaluer l'impact par un expert. En général, ce sera le propriétaire du produit. Si possible, questionnez aussi les utilisateurs. Cela vous aidera aussi à approfondir votre connaissance du contexte dans lequel votre produit sera utilisé.
- Partez du principe que la probabilité d'un bug est 1 (c'est-à-dire 100%) tant que vous n'avez pas testé l'aspect spécifique identifié par l'analyse de risques.

Il est important de limiter le temps consacré à l'analyse et l'évaluation de risques. Dans ce but, Eric van Veenendaal propose de jouer une variante du planning poker pour estimer l'impact et la probabilité de chaque risque. Adaptez donc les cartes mais pas les règles. Cela vous évitera de longues discussions.

6- Spécificités de l'approche dans le contexte agile.

Comme nous l'avons dit au début, le test basé sur les risques n'est pas une méthode spécifiquement agile. Tout au contraire, la méthode est souvent associée au monde «classique» et donc perçue comme étant obsolète. Or, il serait dommage de jeter cet outil de gestion de test avéré par-dessus bord.

Le tableau 5 résume encore une fois les aspects spécifiques à l'Agilité.

Sujet	Aspects spécifiques à l'Agilité	Résultat
Motivation	Permettre aux testeurs d'intégration et de système de suivre le rythme des itérations	Stratégie de test ciblée
Domaine d'application	Définition / raffinement des éléments du carnet du produit (par exemple pendant le toilettage du carnet du produit)	Récits utilisateur supplémentaires / critères d'acceptation d'un récit utilisateur existant
	Priorisation des tests pendant l'itération (généralement effectuée pas des testeurs dédiés intégrés ou associés à l'équipe de développement)	Ensemble des cas de test nouveaux pour chaque sprint correspondant à l'avancement actuel du système
Décomposition du système	Éléments du carnet de produit (épopées ou récits d'utilisateur)	Analyse de risques systématique
Réalisation	Application du principe de la boîte de temps ; planning poker avec cartes adaptées	Réunions plus efficaces

Rôles	Propriétaire du produit	Expertise en matière d'application du produit
	Equipe de développement	Expertise technique en matière d'implémentation
	Maître de mêlée	Expertise en matière de méthodologie AMDEC, modération de réunions
	Testeur d'intégration / de système (si présent)	Expertise en matière de conception et de visualisation de tests

Tableau 5 : Spécificités du test basé sur les risques dans le contexte agile

Notez que le rôle du testeur dédié n'est pas prévu par la théorie de Scrum, mais s'est avéré utile dans la pratique.

7- Conclusion.

Pour résumer, le test basé sur les risques est une approche classique qui s'intègre très bien dans le monde agile. Combiné avec la visualisation des processus à tester, cette approche permet d'obtenir rapidement des tests d'intégration et système et de mieux cibler les efforts d'assurance qualité.

II.10- Retour Individuel et Gamification : comment améliorer la qualité des tests ?

Par Xavier Blanc et Sébastien Pannetier

1- L'engagement des développeurs dans la qualité des tests.

L'intérêt des tests n'est plus à démontrer [Myers 11]. Tester une application permet de révéler les fautes avant qu'elles n'arrivent en production et ne perturbent les utilisateurs. Révélées au plus tôt, il est alors possible de corriger leurs causes et de gagner efficacement en qualité.

Pour autant, on le sait, les tests doivent être de qualité pour apporter une réelle plus-value !

C'est pour répondre objectivement à la question de la qualité des tests que plusieurs mesures quantitatives ont été proposées. La plus connue, notamment pour les tests unitaires, est celle de la couverture de code qui permet de mesurer le taux de code source ciblé par des tests [Miller 63].

Des études empiriques ont d'ailleurs montré l'existence d'une corrélation négative entre la couverture de code et le nombre de bugs [Andrews 06] [Mockus 09]. Un code faiblement couvert est faiblement testé et a donc de fortes chances de contenir des bugs. Améliorer la couverture de code entraîne donc une amélioration de la qualité des tests et ainsi une réduction du nombre de bugs.

Certaines organisations ont même érigé en principe le fait d'avoir un taux minimal de couverture de code. Un taux de 80% ou 85% est souvent choisi [Williams 01], garantissant un bon niveau de qualité, même si de tels seuils sont encore discutés [Inozemtseva 14].

Pour autant, force est de constater qu'il reste compliqué pour une organisation de mettre en place une stratégie d'amélioration de la qualité des tests.

Pour améliorer la qualité des tests, il faut au minimum mettre en place les outils de mesure de la qualité (couverture de code) et cibler un objectif de taux (80 ou 85%). Mais surtout, il faut engager les développeurs dans une démarche agile d'Amélioration Continue pour atteindre le taux ciblé comme objectif et c'est bien là que le bât blesse.

En effet, les développeurs ont trop souvent une mauvaise perception de la qualité des tests. De plus, les indicateurs comme la couverture de code restent trop abstraits ou trop loins de leurs préoccupations. C'est d'autant plus le cas dans les projet legacy où la couverture se mesure parfois avec deux chiffres après la virgule (par exemple 77,46%). On comprend qu'il est très difficile pour un développeur de réaliser l'importance de faire augmenter ce taux de quelques centièmes (le faire monter à 77,58% augmente-t-il réellement la qualité ?) alors que ces quelques centièmes représentent certainement des dizaines de lignes de code.

C'est pour faire face à cette problématique de l'engagement des développeurs dans l'amélioration de la couverture du code, que nous avons mis en place une approche faisant levier sur deux

concepts très connus : le retour individuel [Nadler 79] et la gamification [Deterding 11]. L'objectif étant d'impliquer durablement les développeurs et de les positionner au centre de la stratégie de qualité, tout en renforçant les valeurs de la programmation agile.

Dans ce chapitre, nous présentons notre approche et illustrons les avantages qu'elle apporte. Nous commençons par mieux présenter le concept de couverture de code grâce à un exemple simple de code source. Nous présentons ensuite les deux concepts sur lesquels notre approche fait levier : retour individuel et gamification. Enfin, nous décrivons les résultats que nous avons obtenus sur un cas concret réalisé par Sopra Steria.

2- Couverture de code et Qualité.

La figure 1 présente un exemple très simple d'un code JavaScript et de son test unitaire. Le code permet de créer un Customer (function createCustomer). Le test consiste à créer un Customer et à vérifier son prénom.

```
1   //Customer.js
2   function createCustomer(first, last) {
3       console.log("a customer was created");
4       return {first, last};
5   }
6
7   //TestCustomer.js
8   function testCreateCustomer() {
9       var customer = createCustomer("Bob","Sponge");
10      assert(customer.first).equals("Bob");
11  }
```

Figure 1 : Version initiale (v1) du code et de son test

L'exécution du test couvre toutes les lignes de la fonction createCustomer. En mesurant la couverture du code de cet exemple avec un outil tel que istanbul[1], on peut alors voir que la couverture de code est de 100%.

La figure 2 montre ce même code après que deux développeurs (Camille et Bob) ont effectué des modifications :

```
1   //Customer.js
2   function createCustomer(first, last) {
3       if (! _.isString(first)) {
4           throw "Can't createCustomer, first is not a String";
5       }
6       return {first, last};
7   }
8
9   //TestCustomer.js
10  function testCreateCustomer() {
11      var customer = createCustomer("Bob","Sponge");
12      assert(customer.first).equals("Bob");
13      assert(customer.last).equals("Sponge");
14  }
```

Figure 2 : Version modifiée (v2) du code et de son test

[1]: *https://gotwarlost.github.io/istanbul/*

Camille a modifié le code du test en vérifiant non seulement le prénom mais aussi le nom. Bob, quant à lui, a modifié le code de la fonction createCustomer en ajoutant une vérification du premier paramètre (vérifier que la valeur passée est bien une chaîne de caractères).

Dans cette nouvelle version, le test ne couvre pas toutes les lignes. En effet, le test n'exécutera jamais la ligne 4. La couverture du test devient alors de 75% (il manque une ligne sur les quatre de la fonction). Cette information sur la couverture informe que la qualité du test peut être améliorée.

Cet exemple, bien que très simple, illustre plusieurs points sur la gestion de la qualité des tests. Déjà, il est important de rappeler que la couverture de code est corrélée négativement avec le nombre de bugs. La figure 3 illustre la relation entre le nombre de bugs et la couverture de code. Plus la couverture est grande, moins il y a de bugs. Une telle relation est surtout observable sur des grands projets (legacy). De plus, soulignons que cette relation est surtout vraie quand les tests cherchent réellement à détecter des fautes. Si les tests sont trop simples et ne sont écrits que pour faire augmenter la couverture de code, alors la couverture n'est pas corrélée avec le nombre de bugs.

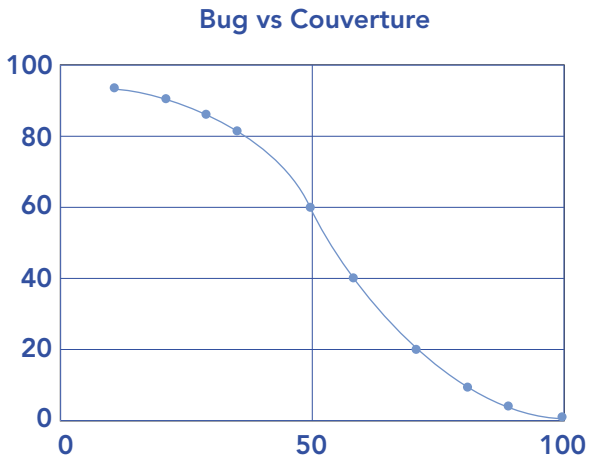


Figure 3 : Illustration de la corrélation négative entre Bug et Couverture.

L'exemple montre aussi la distance de la couverture de code avec les développeurs. En effet, les modifications effectuées par Camille et Bob ont toutes les deux de la valeur ajoutée. Cependant, en regardant l'axe de la couverture, la contribution de Camille ne change pas la couverture alors que la contribution de Bob la fait baisser. Idéalement, il faudrait donc que Bob ait l'information de son impact négatif sur la couverture au plus tôt, cela lui permettrait ainsi de modifier le test pour prendre en compte ses modifications sur le code.

Dans la plupart des organisations, la couverture de code est calculée dans une étape du cycle d'intégration. Les approches agiles vont, quant à elles, s'intéresser à ces informations à certains temps forts : mêlée quotidienne, rétrospective de sprint, etc. Il est ainsi fort probable que Camille et Bob n'aient l'information que bien après avoir réalisé leurs modifications. On peut donc imaginer que d'autres modifications effectuées par d'autres développeurs aient eu, elles aussi,

d'autre impacts négatifs ou positifs, éloignant encore plus l'impact des modifications de Camille et Bob. L'exemple, même simple, montre qu'il est difficile d'engager Camille et Bob dans un objectif d'atteindre 80% ou 85% de couverture de code.

3- Retour Individuel.

Pour améliorer l'engagement dans la qualité des tests, notre approche se base sur le concept du retour individuel. Le constat que nous avons fait est que les mesures de qualité, telles que la couverture de code, portent sur le produit logiciel dans sa globalité. De fait, il est difficile de rapprocher ces mesures des développeurs, ce qui leur permettrait pourtant de déclencher des actions d'amélioration.

Nous avons alors proposé de porter les mesures de qualité sur les modifications (commit) plutôt que sur le code dans sa globalité. Dans le cas de la qualité des tests, notre propos est de mesurer la couverture de code pour chaque commit, ce qui permet alors de créer un lien entre la mesure et le développeur qui a réalisé la modification. Plus précisément, nous avons défini le concept d'**action** qui est réalisée par un développeur lorsqu'il a effectué des modifications. Dans notre exemple, nous avons alors deux actions sur la qualité des tests, une action réalisée par Camille et une par Bob.

De plus, afin d'améliorer la perception de l'impact des actions par les développeurs, nous avons défini trois niveaux de sémantique pour les actions. Pour la qualité des tests, une action est :

- **Saine** lorsque la couverture de code est supérieure au taux cible (i.e. 80%) et que l'action n'a pas réduit la couverture de code.
- **Nocive** lorsque l'action a réduit la couverture de code ou, si la couverture de code est inférieure au taux cible, qu'elle ne l'a pas changé.
- **Réparatrice** lorsque la couverture de code est inférieure au taux cible (i.e. 80%) et que l'action a augmenté la couverture de code.

Sur notre exemple, cette sémantique s'illustre de deux façons différentes selon que Camille effectue ses modifications avant Bob ou après (nous considérons que le taux ciblé est de 80%).

Si Camille effectue ses modifications avant Bob :

- Camille réalise une action saine sur la qualité des tests. Son action ne change pas la couverture de code qui est au-dessus du taux ciblé.
- Bob réalise une action nocive sur la qualité des tests. Son action réduit la couverture et, de plus, la fait passer sous le taux ciblé.

Si Bob effectue ses modifications avant Camille[2] :

- Bob réalise une action nocive sur la qualité des tests. Son action réduit la couverture et, de plus, la fait passer sous le taux ciblé.
- Camille réalise une action nocive sur la qualité car elle ne change pas la couverture qui est sous le taux ciblé.

[2] : *Dans ce cas, il faut que Camille modifie aussi le code source pour que la couverture soit recalculée.*

Le concept d'action et la sémantique associée permet de donner un retour individuel à chaque développeur. Ce retour individuel est fonction des modifications réalisées par le développeur mais aussi de l'état dans lequel se trouve le projet lorsque les modifications ont été réalisées.

La figure 4 montre la vue graphique de l'outil que nous avons réalisé, nommé Themis[3], et qui supporte ce concept de retour individuel. La figure montre l'interface graphique proposée par Themis pour Bob (chaque développeur dispose de son propre cockpit). Bob peut ainsi voir la dernière action qu'il a réalisée. Il voit qu'il a réalisé une action nocive sur les tests. Il sait quel fichier est la cause de cet impact négatif.

Figure 4 : Le cockpit de Bob montrant une action nocive

Grâce à notre exemple, on voit que le retour individuel va engager Bob à rectifier l'action nocive qu'il a réalisée. En effet, voyant que son action a fait baisser la couverture de code, il sera plus enclin à effectuer une correction. Lorsqu'il aura réalisé la correction, il verra alors apparaître une nouvelle action qui sera qualifiée comme étant réparatrice. Pour Camille, le retour individuel lui permettra d'être soit rassurée et de voir qu'elle a fait une action saine (si elle a effectué ses modifications avant Bob), soit de voir qu'elle peut améliorer le test qu'elle vient de toucher (si elle a effectué ses modifications après Bob).

4- Gamification.

Pour faciliter et accroître l'engagement des développeurs dans la qualité des tests, nous avons intégré une couche de gamification à notre approche. La gamification c'est «l'application des concepts du jeu dans un contexte professionnel» [Deterding 11]. L'idée est d'ajouter des points, des badges, des quêtes, etc., dans l'objectif de rendre les tâches professionnelles plus ludiques et donc plus intéressantes. L'objectif est d'accroître le fun et donc l'engagement dans des comportements considérés comme peu intéressants. Notons qu'il est important de ne pas confondre la gamification avec le serious game qui consiste, lui, à apprendre en jouant.
Dans notre approche, la gamification est optionnelle. Chaque développeur peut créer un « salon de jeux » et proposer à d'autres joueurs d'y participer. A tout moment, un développeur peut quitter un salon s'il désire ne plus jouer. Les développeurs qui participent à un salon jouent tous aux mêmes jeux et gagnent des niveaux et des badges.

[3] : *https://promyze.com/themis/*

Les niveaux sont obtenus en effectuant des actions saines ou réparatrices. Les actions saines et réparatrices permettent de gagner des points. Il faut atteindre un certain nombre de points pour gagner un niveau. Notons que les actions nocives font perdre des points. Pour autant, il n'est pas possible de perdre un niveau dès lors qu'il a été acquis.

Les badges sont, quant à eux, gagnés en effectuant des séries d'actions. Par exemple, le badge «constant» s'obtient en réalisant une série d'actions saines alors que le badge «samaritain» s'obtient en réalisant plusieurs actions réparatrices. Notre couche de gamification propose cinq types de badge avec, à chaque fois, la possibilité de l'avoir en bronze, argent ou or. Enfin, le badge «accompli» s'obtient quand on a obtenu tous les autres badges.

Chaque développeur qui participe à un salon voit ses propres scores et peut voir les scores des autres participants du salon. Le classement est établi en fonction des badges (nombre de badges d'or, puis d'argent et bronze). En cas d'égalité dans les badges, le niveau est alors utilisé pour départager les joueurs. Enfin, il est possible de réinitialiser les scores des badges ce qui revient à faire une nouvelle partie. La figure 5 montre la vue de Camille qui participe à un salon et est actuellement deuxième.

Figure 5 : La vue gamification de Camille avec ses scores et le classement des joueurs

Sur notre exemple, on peut imaginer l'émulation que peut apporter la gamification et l'effet positif qu'elle peut avoir sur les développeurs tels que Camille et Bob pour améliorer la qualité des tests. On comprend aussi que la gamification doit être animée afin de garder un esprit positif et ludique sur l'équipe.

4- Cas concret : Sopra Steria.

Nous avons mis en œuvre notre approche chez Sopra Steria, dans un groupe d'une cinquantaine de collaborateurs sur le site de Bordeaux. Ce groupe s'occupe de la maintenance d'une application legacy pour le compte d'un grand client.

Notre approche a été déployée en 2017 dans l'objectif d'améliorer la gestion de la dette technique et des tests. Le déploiement a été progressif avec quelques développeurs au début puis l'intégralité des développeurs après quelques mois. Notons que l'équipe utilise encore l'outil quotidiennement.

Nous avons effectué une étude qualitative pour mesurer les avantages et les limites de notre approche. Cette étude a consisté à réaliser des interviews avec une dizaine de développeurs. Suite à ces interviews, nous avons regroupé et recoupé les avis afin d'élaborer une synthèse des ressentis.

Concernant le retour individuel, notre étude montre que les développeurs y trouvent de nombreux avantages. Le retour individuel est précis car il cible le développeur, ses dernières actions et l'impact de celles-ci sur la qualité des tests «On voit vraiment ce qu'on a fait, nous, c'est plus personnel donc c'est mieux. (d11)». Il offre une évaluation personnelle permettant aux développeurs de passer plus facilement à l'action : «J'utilise beaucoup le fil d'actualités, pour voir et corriger mes erreurs et me permettre de progresser (d11)».

Un point intéressant est que le retour individuel permet une prise de conscience des développeurs sur la qualité des tests «ça m'a plutôt aidé à prendre conscience (d14)». A l'usage, le retour individuel fait changer la façon dont les développeurs commitent : «Ça me force à être plus concentré quand j'écris du code (d14)». Enfin, au-delà de l'aspect individuel, cela favorise la communication au sein de l'équipe : «Ça permet de discuter avec l'équipe pour disséminer les bonnes pratiques sur certaines règles (d15)».

Sur la gamification, il faut noter que la plupart des développeurs ont trouvé que cela ajoute un aspect ludique et donc renforce l'esprit d'équipe : «Les badges ne sont pas un objectif en soi, mais c'est surtout amusant d'obtenir ceux qu'on a jamais eus (d16)», «L'ambiance de la compétition est bon enfant, chacun essaie de faire le mieux possible (d15)». Notons cependant que tous les développeurs ne sont pas sensibles à la gamification et trouvent même parfois cela contre-productif : «Cela ne sert pas à grand-chose. (d5)».

Enfin, la gamification promeut une amélioration de la qualité grâce à la volonté de progresser dans le classement : «La gamification nous pousse à regarder pourquoi on a fait une action nocive et à la corriger (d12)», «Le jeu pousse vers le haut et le climat de la compétition est amical (d16)».

5- Conclusion.

L'étude que nous avons menée permet de mesurer les avantages du retour individuel et de la gamification. Notre analyse est que le retour individuel est un accélérateur favorisant l'engagement des développeurs dans la qualité des tests. La gamification est, quant à elle, l'accélérateur de l'accélérateur.

Notre expérience nous a permis de remarquer qu'il fallait impérativement prendre en considération le rôle du management dans cette stratégie de l'amélioration de la qualité. Si notre approche est dédiée aux développeurs, il est important de préciser que les informations véhiculées par le retour individuel et la gamification ne seront pas utilisées pour mesurer la performance des développeurs.

Notons de plus que le retour individuel et la gamification nécessitent une animation qui tire vers le haut et garantit un climat positif. Lors de notre expérimentation avec Sopra Steria, le

management s'est mis explicitement en retrait du dispositif en responsabilisant les développeurs et ces développeurs ont animé spontanément cette approche. Ce rôle d'animateur est absolument nécessaire et doit être clairement identifié.

Enfin, il est important de noter que notre approche met en valeur les valeurs du Manifeste Agile. D'abord, elle est centrée sur l'individu grâce au concept de retour individuel mais favorise fortement leurs interactions grâce à la gamification. Ensuite, en promouvant la qualité, son objectif est bien de fournir un logiciel qui fonctionne sans que l'utilisateur ne souffre d'aucun bug. Enfin, elle facilite la conduite du changement en mettant en place des mécanismes itératifs d'amélioration de la qualité.

Bibliographie.

[Andrews 06] Andrews JH, Briand LC, Labiche Y, Namin AS (2006) Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria. IEEE Transactions on Software Engineering 32(8):608-624.

[Deterding 11] Deterding S, Dixon D, Khaled R, Nacke L (2011) From Game Design Elements to Gamefulness: Defining "Gamification". In: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, ACM, New York, NY, USA, MindTrek '11, pp 9-15

[Inozemtseva 14] Inozemtseva L, Holmes R (2014) Coverage is Not Strongly Correlated with Test Suite Effectiveness. In: Proceedings of the 36th International Conference on Software Engineering, ACM, New York, NY, USA, ICSE 2014, pp 435-445

[Miller 63] Miller JC, Maloney CJ (1963) Systematic Mistake Analysis of Digital Computer Programs. Commun ACM 6(2):pp58-63

[Mockus 09] Mockus A, Nagappan N, Dinh-Trong TT (2009) Test coverage and post-verification defects: A multiple case study. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, pp 291-301

[Myers 11] Myers GJ, Sandler C, Badgett T (2011) The Art of Software Testing. Wiley. ASIN: B005PETXRM

[Nadler 79] Nadler DA (1979) The effects of feedback on task group behavior: A review of the experimental research. Organizational Behavior and Human Performance 23(3):309-338

[Williams 01] Williams T. W., Mercer M. R., Mucha J. P. and Kapur R., Code coverage, what does it mean in terms of quality?, Annual Reliability and Maintainability Symposium. 2001 Proceedings. International Symposium on Product Quality and Integrity Philadelphia, PA, USA, 2001, pp. 420-424.

II.11- Test des systèmes IoT.

Par Abbas Ahmad et Hamza Baqa

1- Introduction.

Cela fait plus de dix ans que le terme Internet des Objets (IoT pour «Internet of Things» en anglais) a été introduit. Faisant son chemin sous les projecteurs, il est maintenant devenu populaire. Prudemment, des entreprises leader du marché adoptent vigoureusement le terme dans leurs produits et services les plus avancés. Il est en vogue et reflète l'état de l'art. Cependant, comme le terme est plus largement utilisé, ses interprétations se diversifient. Certains appelleraient n'importe quel appareil connecté un IoT, tandis que d'autres ne feront référence qu'aux analyses de données volumineuses «Big Data». Les applications s'étendent à un grand nombre de domaines, tels que les villes intelligentes, les maisons intelligentes, la santé, etc. Le Groupe Gartner estime à 21 milliards le nombre d'objets connectés d'ici 2020. Le grand nombre d'objets connectés introduit des problèmes, tels que la conformité et l'interopérabilité en raison de l'hétérogénéité des protocoles de communication et de l'absence d'une norme mondialement acceptée. Le grand nombre d'utilisations introduit des problèmes de déploiement sécurisé et d'évolution du réseau des IoT pour former des infrastructures de grande taille. Ce chapitre aborde la problématique de la validation de l'internet des objets pour répondre aux défis des systèmes IoT.

Pour cela, nous verrons dans un premier temps les systèmes IoT en terme général, ainsi qu'une introduction plus détaillée des problématiques qui lui sont liées. Nous aborderons ensuite les problématiques de tests sémantiques, tests de sécurité et finalement tests de montée en charge avant de conclure.

2- Systèmes IoT.

Définition de l'union de télécommunication internationale (ITU) [1] : «L'Internet des Objets est une infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution».

Le système complet d'un objet connecté intègre quatre composants distincts :

a. Les capteurs : Tout d'abord, les capteurs récoltent les données de leur environnement. Cela peut être une simple lecture de température comme une lecture complète d'un flux vidéo.

b. Connectivité : Ensuite, l'information est envoyée dans le cloud, mais elle besoin d'un moyen d'y parvenir. Les capteurs peuvent être connectés au cloud au travers d'une variété de méthodes incluant : réseau cellulaire, satellites, WIFI, Bluetooth, les réseaux LORA ou une connectivité directe à internet via ethernet.

c. Traitement de l'information : Une fois l'information dans le cloud, des logiciels réalisent un traitement sur ces dernières. Le traitement peut être très simple, il peut s'agir de vérifier si la température lue est comprise dans une fourchette – ou plus compliqué, comme par exemple définir des emplacements de parking libre à partir d'un flux vidéo.

d. Interface utilisateur : L'information est tout d'abord rendue en temps réel utile et lisible pour l'utilisateur final. Elle peut prendre vie sous forme d'alerte à l'utilisateur (email, notification, etc.).

En fonction du champ d'application de l'objet connecté, le flux d'information n'est pas toujours un envoi à sens unique. L'utilisateur peut aussi être en mesure de déclencher une action sur le système. Par exemple, l'utilisateur peut ajuster la température de la chambre froide lorsqu'il décèle une vague de chaleur grâce aux capteurs, directement depuis son téléphone.

Comment assurer la fiabilité d'un système IoT, base de la confiance des utilisateurs ?

Il faut le tester afin de pouvoir assurer :

- Que les données stockées sont conformes au modèle de données (vérification syntaxique et sémantique) et valides (qualité de l'information).
- Que le système assure le niveau de performance attendue, en particulier lors d'une montée en charge.
- Que les tests de sécurité – considérés ou non comme du test fonctionnel – qui sont critiques dans le monde en émergence de l'IoT, doivent être abordés de façon différente des systèmes classiques en raison des aspects non monolithiques et hétérogènes des systèmes IoT.

Ainsi, les tests sont cruciaux. Pourtant, ils sont quelquefois négligés car fastidieux pour les équipes de développement. Il est donc nécessaire de mettre en place un système de test automatisé qui présente de nombreux avantages :

- Le gain de temps dû à un allègement de la charge de travail des équipes.
- L'augmentation de la productivité.
- La fiabilité et la qualité des tests effectués.
- L'accroissement du niveau de complétude des tests réalisés.
- La démobilisation des équipes sur les tâches répétitives au profit de tâches à plus grande valeur ajoutée (analyse des résultats, définition des cas de tests, planification…).

On favorise l'approche boîte noire, qui se focalise sur la réponse des systèmes en ne considérant que leurs interfaces d'entrée-sortie, sans se préoccuper de la façon dont les fonctions sont implémentées en interne. Cette approche présente l'avantage d'être orientée vers les fonctionnalités finales tout en laissant la liberté aux développeurs d'avoir leurs propres stratégies de vérification au niveau du code (test boîte blanche). Ce type d'approche a prouvé sa validité pour des architectures micro-services [2].

L'automatisation peut intervenir :

- Lors de la génération des cas de test : dans ce cas, un modèle de haut niveau du système (modèle de test) est développé sur la base de son périmètre fonctionnel. Ce modèle permet alors la génération de cas de test, suivant des objectifs fournis à l'outil de génération (par exemple : taux de couverture des transitions ou des exigences).
- Lors de l'exécution des cas de test : les tests fonctionnels s'intègrent naturellement dans le cadre d'une architecture d'Intégration Continue. Elle permet de s'assurer qu'une modification, même mineure, ne provoquera pas une erreur de la part du logiciel, dans une situation jamais envisagée jusque-là.

Disposer d'un environnement de test est aussi le point de départ à la mise en place d'un système de labélisation qui permet de valider qu'une application, un composant, un modèle ou un flux de données sont conformes aux exigences de l'écosystème.

3- Tests sémantiques.

L'Internet des Objets nécessite non seulement le développement d'une infrastructure, mais également le déploiement de nouveaux services capables de prendre en charge de multiples applications (scalabilité), ainsi que la possibilité d'interconnecter ces derniers (l'interopérabilité). Avec l'augmentation rapide du nombre de données provenant de l'IoT et la topologie des données IoT (hétérogène), le problème de l'interopérabilité est devenu l'un des sujets phares de recherche dans le but d'atteindre le plein potentiel du domaine. Les organismes de standardisation tels que W3C[1], ETSI[2] et oneM2M[3] sont également à leur tour intéressés par cette problématique à laquelle ils contribuent activement. Dans le domaine du Web sémantique, la sémantique des données est décrite par des ontologies, ensembles structurés des termes et concepts représentant le sens d'un champ d'informations, que ce soit par les métadonnées d'un espace de noms ou les éléments d'un domaine de connaissances. Après des années de recherche, des méthodologies de développement d'ontologies, des meilleures pratiques et des directives ont été proposées pour aider les gens à développer ou à utiliser une ontologie en (créant) adaptant les ontologies (non) existantes. Dans cette perspective, l'évaluation de ces ontologies ou de ces données sémantiques est essentielle pour déterminer si elles sont interopérables avec d'autres ontologies, en particulier les ontologies de référence reconnues par les organismes de standardisation tels que l'ontologie SSN [6], l'ontologie SAREF [7] et l'ontologie de base oneM2M [8].

La validation sémantique.

A partir des caractéristiques des données sémantiques qui suivent le modèle «Resource Description Framework» (RDF) et utilisent l'ontologie de référence, on distingue trois niveaux de validation :

a. Vérification lexicale : ce niveau de vérification consiste à valider l'exactitude de la sérialisation RDF par rapport au type déclaré. Par exemple, les données sémantiques sont marquées en

[1] : *https://www.w3.org/*
[2] : *https://www.etsi.org/*
[3] : *http://www.onem2m.org/*

représentation XML (ex: spécifié dans le suffixe du fichier) alors que l'annotation sémantique est sérialisée en JSON. Ou le document est détecté en XML mais contient une erreur qui provoque une erreur d'analyse et la vérification lexicale échoue.

b. Vérification syntaxique : après les vérifications lexicales de base, la vérification syntaxique consiste à contrôler l'exactitude de la «syntaxe» des triplets RDF représentés par le format de sérialisation souligné, plus précisément :

- Ressources et littéraux non typés. Ici, la ressource fait référence aux instances d'une classe et la valeur littérale renvoie à une valeur textuelle ou numérique. Le type de ressource ou littéral est le lien d'une annotation vers l'ontologie qui active les capacités sémantiques. Tout élément non typé présenté dans une annotation est problématique pour l'interopérabilité sémantique.
- URIs mal formés. L'URI est essentiel et critique pour l'identification d'une ressource. Les URIs doivent être vérifiés par rapport à la RFC3968 qui définit la syntaxe générique de l'URI.
- Préfixes et espaces de noms problématiques. Les espaces de noms servent à lier l'annotation aux ontologies et vocabulaires de référence. Si l'URI de l'espace de noms est problématique (par exemple un URI erroné, l'URI contient un caractère illégal), cela peut amener les autres à interpréter incorrectement la sémantique et les types de données. Le préfixe est une référence unique pour remplacer les espaces de noms dans le fichier local. Un mapping un-à-un entre le préfixe et l'espace de noms est essentiel et doit être vérifié pour assurer une référence correcte.
- Classes et propriétés inconnues. Une condition préalable de l'interopérabilité sémantique est que toutes les ressources utilisent un vocabulaire commun et convenu. En conséquence, si une ressource utilise dans son annotation une classe ou une propriété qui n'est pas définie dans l'ontologie de référence, les autres ressources n'auraient aucun moyen de la comprendre, de sorte que l'interopérabilité sémantique est impossible.

c. Vérifications sémantiques. Suite à une validation syntaxique réussie, la vérification sémantique consiste à vérifier la cohérence logique de l'annotation sémantique par rapport à l'ontologie de référence :

- Incompatibilité de cardinalité. Si l'ontologie définit que la classe A peut avoir une et une seule instance de la classe enfant B et que, dans l'annotation, il y a deux instances de B liées à une instance de A, il y a un problème.
- Relation ou héritage problématique. Suivant la relation définie dans l'ontologie de référence, si une instance d'une classe A est annotée à tort comme étant en même temps une instance de classe B disjointe de la classe A, il y a un conflit et l'instance ne peut être résolue par le moteur sémantique.

Le résultat de la validation dépend du résultat de chacun des tests ci-dessus. Pour conclure qu'une annotation sémantique est validée, une vérification complète de tous les contrôles ci-dessus doit être effectuée et transmise. Cependant, comme plusieurs tests sont indépendants des autres (par exemple, les points «c.» n'ont pas d'impact les uns sur les autres), plusieurs «profils validés» peuvent être définis comme un sous-ensemble de tous les aspects à vérifier.

Certification sémantique : approche FIESTA.

Le projet H2020 FIESTA-IoT[4] consiste à chercher à fédérer les TestBed de différents horizons aux niveaux de données : quel que soit le format de données au départ, tous les TestBed fournissent leurs données en les annotant par une ontologie commune (Fiesta Ontology) pour réaliser l'interopérabilité. Dans le cadre de Fiesta-IoT, nous avons mis en place un **outil en ligne de tests de conformité sémantique** [9] pour une première étape vers l'interopérabilité sémantique.

Cet outil vérifie les trois aspects liés à la conformité sémantique expliqués dans la section précédente. Une conclusion de conformité est compilée à partir des résultats de chaque test. Cet outil intervient à la plateforme FIESTA-IoT aux deux endroits (les numéros correspondent aux numéros dans la Figure 1) :

a. Certification de conformité de données sémantiques fournies par le TestBed avant que le TestBed ne soit connecté à la plateforme fédérée. Avant de rejoindre la plateforme, le fournisseur de données sémantiques doit montrer un certificat qui prouve que les données sont conformes à l'ontologie de référence. Ce certificat est obtenu seulement quand tous les tests de conformité implémentés par l'outil de validation sémantiques sont passés avec succès.

b. Une fois le Testbed certifié et ayant rejoint la plateforme, il pousse des données sémantiques périodiquement. Une vérification de conformité sémantique, par cet outil de validation sémantique sur les données reçues par la plateforme, est exécutée au fur et à mesure pour assurer le niveau de conformité. Si une erreur est détectée à ce stade-là, la donnée qui contient cette erreur est rejetée et toutes les données émises par le même fournisseur de données seront à vérifier à chaque fois, jusqu'au moment où le succès est maintenu pendant une durée donnée.

Le consortium FIESTA exige que tous les fournisseurs de données démontrent le certificat et que toutes les données reçues par la plateforme, quelle que soit la source, soient soumises à la vérification à la volée (n°3 dans Figure 1).

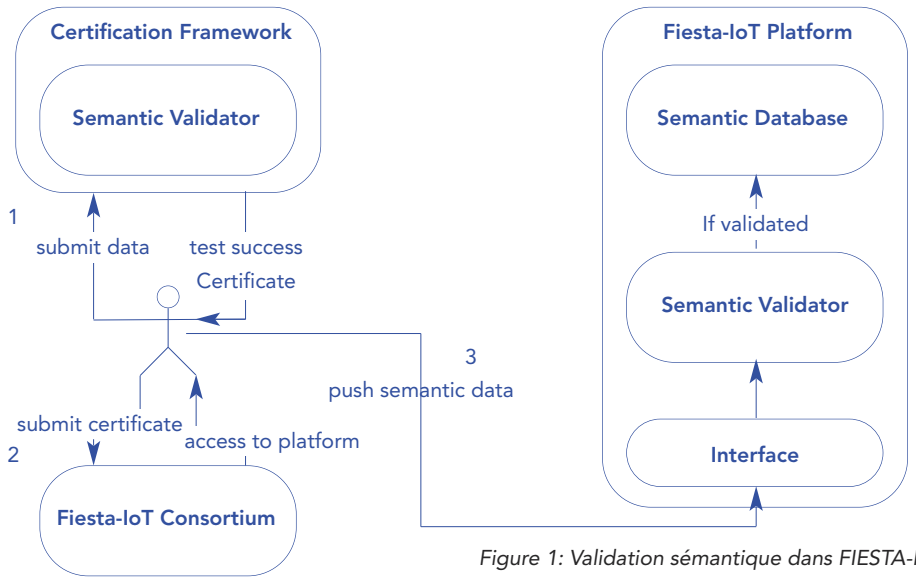


Figure 1: Validation sémantique dans FIESTA-IoT plateforme

[4] : *http://fiesta-iot.eu/*

4- Tests de montée en charge.

La question peut paraître obsolète car désormais, une ou plusieurs phases de tests de montée en charge font partie intégrante de la majorité des processus de développement d'un système IoT.

Cependant, des incertitudes subsistent, en particulier pour les plateformes sémantiquement basées car peu de références existent sur le marché. Il est donc nécessaire d'évaluer la dégradation de performance face à un nombre d'utilisateurs et un volume de données croissants.

Quel est le meilleur moment pour réaliser un benchmark ?

Il est souvent trop tard lorsque l'on s'interroge au sujet de la mise en place de tests de montée en charge. En principe, on se pose des questions lorsqu'un problème est survenu ou lorsque l'on émet des doutes sur le bon fonctionnement de son architecture. Dans le cas présent, il est impératif d'initier les tests de montée en charge le plus tôt possible, afin de valider les choix technologiques réalisés dans la conception du système.

De nombreux aspects techniques émergeant des architectures logicielles sont à prendre en compte :

- Centralisation des traitements : contrairement au client/serveur, il faut être en mesure de gérer un partage de ressources, les accès concurrents, des pools d'accès aux bases de données, etc. Problématique de dimensionnement : nombre de serveurs, puissance des processeurs, taille mémoire, espace disque, bande passante, etc.
- Complexité des architectures : les architectures contiennent de plus en plus de maillons techniques pour une seule transaction, il faut tester chaque partie afin d'assurer le fonctionnement de l'ensemble.
- Choix techniques : les services web sont-ils plus performants que des échanges HTTP/XML ? Quel est le serveur d'applications le plus performant dans notre contexte ?

La mise en œuvre de tests de montée en charge permet, si les tests sont complètement maîtrisés, de répondre à la majorité des questions posées.

Les technologies de cloud (virtualisation, containers) sont des moyens de gérer une montée en charge.

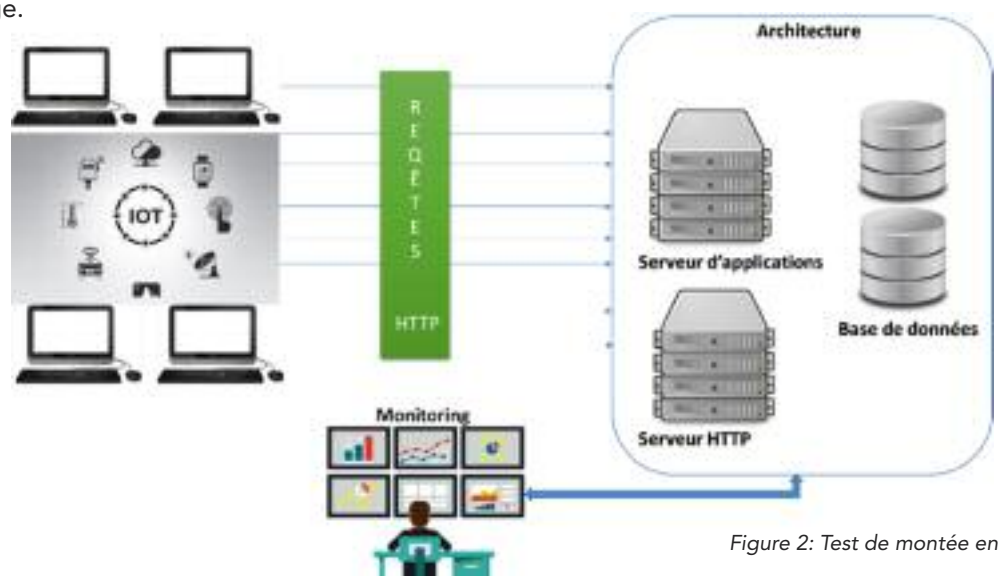


Figure 2: Test de montée en charge

Comme le représente la Figure 2, un test de montée en charge consiste à solliciter (mettre en charge) une architecture, un site ou une application Web de la même manière qu'en situation d'exploitation. Concrètement, il s'agit de reproduire le comportement d'un nombre important d'utilisateurs à l'aide d'outils spécialisés, de manière à mesurer les performances de l'architecture testée.

On propose une méthodologie qui est résumée par ces différentes étapes :

- Formalisation du besoin : choix du type de charge, du type de test, de l'outil, etc.
- Préparation à la mise en œuvre : mise en place de l'architecture, sensibilisation des intervenants et écriture des scripts.
- Exécution des tests : respect de certaines règles fondamentales et monitoring de l'architecture.
- Analyse des résultats : 4 axes d'analyse (cartographies, ressources, performances et fiabilités).
- Proposition d'optimisations et accompagnement pour les appliquer et établir un transfert de connaissances.

Un article du blog de Redline13 [5] montre les différentes fonctionnalités des divers outils open-source de test de montée en charge. Nous ne nous intéresserons qu'aux solutions proposant toutes les fonctionnalités du comparatif, c'est-à-dire les solutions distribuées, avec un recorder (outil permettant d'enregistrer un scénario), proposant des graphiques, des plugins et pouvant être intégréés avec Jenkins. On peut donc s'intéresser à :

- Gatling : développé en Scala et basé sur Akka et Netty.
- JMeter : développé à 100% en Java.
- Grinder : Également développé en Java.

Les experts en performances sont conscients que le plus difficile reste l'interprétation des résultats de ces tests, dans le but de trouver la cause du potentiel problème de performance. C'est pourquoi dans le choix des outils, l'un de nos critères est tout simplement la qualité du rapport final de chaque test. Ensuite, un autre critère est la capacité de la technologie à passer à l'échelle et à générer un grand nombre d'utilisateurs simultanément.

5- Tests de sécurité.

L'Internet des Objets promet de connecter facilement des milliards d'appareils à Internet. Si l'on examine les technologies et les différents terminaux constituant l'IoT, nous retrouvons les smartphones, le Wi-Fi, le Bluetooth, la gestion de données, le stockage et, bien sûr, la sécurité. Les échecs de projets M2M/IoT sont souvent liés à des problèmes de cybersécurité ou aux défis de la gestion de déploiement à grande échelle. Il y a encore quelques années, un identifiant utilisateur et un mot de passe suffisaient. De nos jours, plusieurs couches de sécurité, comprenant une authentification et un chiffrement allant du niveau des composants de l'appareil jusqu'au cloud, sont nécessaires pour garantir la protection des données critiques.

Voici notre approche pour réaliser une évaluation complète du système IoT :

Il faut effectuer une évaluation de haut niveau pour comprendre où les dispositifs IoT se situent dans le réseau, comment ils interagissent et partagent des informations avec d'autres appareils et quel type d'informations ils envoient en dehors du réseau. Une des approches pratique est d'utiliser la modélisation (MBT). Le but de cette activité est d'identifier toutes les surfaces d'attaque possibles (définies par OWASP ou autres) qu'un attaquant pourrait exploiter pour voler des données, causer un déni de service ou, dans le pire des cas, prendre le contrôle intégral du périphérique. Une fois la phase d'évaluation terminée, il est possible de mettre en évidence tous les points d'intérêt liés à la sécurité concernant l'environnement IoT du système, puis d'évaluer s'il est nécessaire de concentrer l'attention sur une seule partie de l'environnement, ou de passer en revue l'intégration entre différents composants.

6- Conclusion.

Le déploiement d'un système IoT requiert préalablement une phase de tests, élaborée d'après une méthodologie rigoureuse. Les tests doivent fournir des informations précises sur la qualité d'un produit ou service, en rapport avec le contexte dans lequel il est censé fonctionner. Les tests doivent également fournir une vue objective et indépendante du système, afin de permettre aux entreprises d'évaluer et de comprendre les risques impliqués dans la mise en œuvre du système. Le système doit valider et vérifier qu'un produit répond aux exigences commerciales et techniques convenues lors des premières étapes et il doit fonctionner comme prévu.

Les tests ne permettent jamais de mettre complètement en évidence tous les bugs et défauts d'un produit. Au lieu de cela, ils identifient la qualité et le fonctionnement du produit dans un environnement particulier.

Chaque produit a un public cible et quand une entreprise développe ou investit dans un produit logiciel, elle doit s'assurer que le produit répondra aux besoins de ses utilisateurs finaux, de son public cible et de ses acheteurs.

Bibliographie

[1] ITU : Définition du terme IoT. https://www.itu.int/en/ITU-T/ssc/resources/Pages/topic-001.aspx

[2] Ting Miao, Nak-Myoung Sung, Jaeyoung Hwang, Naum Spaseski, György Réthy, Elemer Lelik, Jaeseung Song and Jaeho Kim, Development of an Open-Sourced Conformance Testing Tool – oneM2M IoT Tester, User Conference on Advanced Automated Testing, UCAAT 2016, October 26-28, 2016, Budapest, Hungary

[3] World Wide Web Consortium: https://www.w3.org/

[4] ETSI: https://www.etsi.org/

[5] Redmine13 blog : https://www.redline13.com/blog/

[6] M. Compton et al., « The SSN Ontology of the W3C Semantic Sensor Network Incubator Group », p. 10.

[7] L. Daniele, F. den Hartog, et J. Roes, « Created in Close Interaction with the Industry: The Smart Appliances REFerence (SAREF) Ontology », in Formal Ontologies Meet Industry, vol. 225, R. Cuel et R. Young, Éd. Cham: Springer International Publishing, 2015, p. 100 112.

[8] ETSI oneM2M Base ontology (oneM2M TS-0012 version 2.0.0 Release 2: https://www.etsi.org/deliver/etsi_ts/118100_118199/118112/02.00.00_60/ts_118112v020000p.pdf

[9] M. Zhao, N. Kefalakis, P. Grace, J. Soldatos, F. Le-Gall, et P. Cousin, « Towards an Interoperability Certification Method for Semantic Federated Experimental IoT Testbeds », in Testbeds and Research Infrastructures for the Development of Networks and Communities, vol. 177, S. Guo, G. Wei, Y. Xiang, X. Lin, et P. Lorenz, Éd. Cham: Springer International Publishing, 2017, p. 103 113.

II.12- Chaos Engineering : et si on testait en production ?

Par Christophe Rochefolle

1- Contexte.

L'importance de la Qualité de Service (QoS).

Avec la complexité IT qui s'accroît, les incidents sont inévitables pour toutes les entreprises. Une étude intitulée « Masters of Machine III », réalisée par le cabinet d'analyse Quocirca[1], démontre qu'une entreprise européenne moyenne perd des millions d'euros chaque année, pour une moyenne de trois incidents par mois, avec un coût moyen pour chaque incident IT de 115 034 €. Au-delà de l'aspect financier, les indisponibilités ont un impact sur l'image de l'entreprise car elles sont désormais très rapidement connues et massivement partagées sur les réseaux sociaux. Un des exemples récents les plus marquants est l'indisponibilité de l'espace « Prime Day » d'Amazon, un événement annuel avec une forte visibilité, pendant 75 minutes, représentant une perte estimée à 90 millions de dollars[2].

Ainsi, pour la première fois, **les indisponibilités arrivent en tête des sujets d'inquiétude des responsables informatiques, devançant ainsi la sécurité.**

Et pourtant avec l'Agilité, la Qualité est non négociable !

Un des postulats de base des démarches agiles est que **la qualité est non négociable**, il est incarné par le principe agile : *« Un logiciel opérationnel est la principale mesure d'avancement*[3] *».* Ainsi, on préfère livrer moins (en quantité) mais livrer bien (en qualité). La conséquence est que les tests sont ainsi au cœur des responsabilités de l'équipe applicative, avec notamment l'avènement de démarche TDD, BDD. On a donc assisté à ce qu'on appelle un ***« shift left »*** des tests par la prise de conscience du principe des tests : « tester au plus tôt ». La démarche DevOps a par ailleurs renforcé la nécessité d'automatiser les tests afin d'accompagner les pipelines de déploiement et de permettre le déploiement en continu.

Si les pratiques de tests ont évolué pour s'adapter à ces nouveaux modes de fonctionnement, **la nécessité de tester n'est plus un combat** et nous pouvons, en tant que communauté de tests, nous en réjouir. Cependant, l'accélération des livraisons a clairement challengé nos pratiques de tests bout en bout, et encore plus celles des tests techniques (performance, résilience, etc.).

Elle a spécialement mis l'accent sur un problème systémique aussi ancien que les pratiques de tests : **comment tester dans un environnement suffisamment représentatif de la production, voire iso-production ?** Cette problématique complexe n'a jamais été résolue et la difficulté s'accentue, sans parler de l'arrivée de l'Intelligence Artificielle et des ordinateurs quantiques qui vont contribuer à la rendre insoluble. Pour citer Werner Vogels, CTO d'Amazon, *« Comment tester dans un environnement comme celui d'Amazon ? Devez-vous construire un autre Amazon*

[1] : *https://www.splunk.com/pdfs/infographics/splunk-master-of-machines-III-infographic.pdf*
[2] : *https://techcrunch.com/2018/07/18/amazon-prime-day-outage-cost/*
[3] : *https://agilemanifesto.org/iso/fr/principles.html*

pour les tests quelque part, qui aurait le même nombre de machines, le même nombre de centres de calcul, de clients et les mêmes tables et fichiers ? ».

Après le shift left, le shift right ?

Si tester en environnement proche de la production devient de plus en plus utopique, que pouvons-nous envisager ? Afin de dépasser ces limites, il faut pouvoir aller plus loin et dépasser le cycle habituel d'un projet en poussant les tests jusqu'à la production. On appelle cela le ***« shift right »***. Le test ne se cantonne donc plus au *« cycle projet »* mais va au-delà pour atteindre le « cycle de vie du produit ».

Le shift right permet également d'avoir des retours en temps réel et de pouvoir répondre plus rapidement aux problématiques des clients. Ainsi, il ne s'agit plus de se mettre à la place de l'utilisateur pour tester mais d'utiliser la « vraie vie » dans nos stratégies de tests.

Nous voyons donc émerger de plus en plus de pratiques de tests en production :

- ***Au niveau fonctionnel :***

  – ***A/B test :*** Il ne s'agit plus seulement de vérifier qu'il n'y a pas de régression dans les fonctionnalités proposées, mais également qu'elles produisent une valeur business identique, voire meilleure que la version précédente. On évalue ainsi en production la valeur business (taux de transformation, panier moyen, …) d'une ou plusieurs versions partiellement ou totalement différentes d'une même page en la comparant à la version originale.

  – ***Beta test :*** On propose à un échantillon représentatif et/ou privilégié de nos utilisateurs un accès à la prochaine version du logiciel ou service pour recueillir ses feedbacks. On peut ainsi adapter, corriger à partir de ces retours avant de généraliser à l'ensemble des utilisateurs.

- ***Au niveau technique :***

  – ***Canary test :*** les tests de performance devenant de plus en plus complexes, de nombreux acteurs du web, principalement les GAFA, ont opté pour des stratégies de déploiement progressif accompagnées de mesure des principaux indicateurs de performance. Ce pattern est notamment utilisé par Facebook, qui déploie ses mises à jour dans un premier temps à l'ensemble de ses employés, puis à tous les utilisateurs si tout se passe bien pendant une journée.

  – ***Chaos Engineering :*** expérimenter pour s'assurer que l'impact d'une panne est atténué.

Si les pratiques d'A/B test, de Beta test et de Canary test ne sont pas vraiment récentes, elles sont de plus en plus présentes dans nos entreprises.. Le Chaos Engineering, lui, est plus nouveau comme moyen d'améliorer la confiance et la qualité de nos systèmes. La suite de ce chapitre va donc se concentrer sur ce sujet.

2- Chaos Engineering.

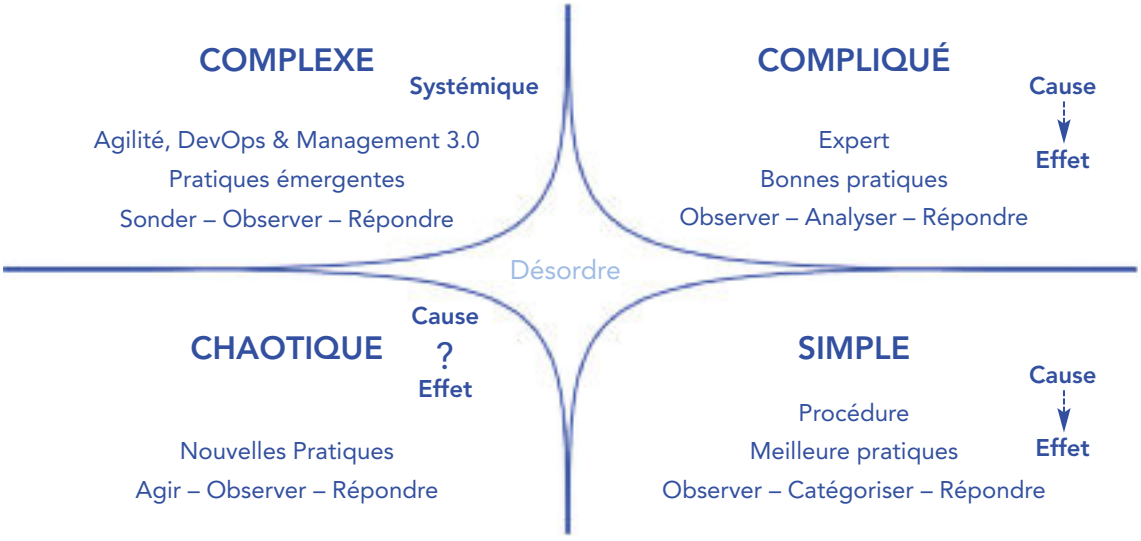
Notion de complexité et chaos.

En mathématiques, le chaos est défini comme le comportement des systèmes dynamiques très sensibles aux conditions initiales, un phénomène généralement illustré par l'effet papillon. Des différences infimes dans les conditions initiales (comme des erreurs d'arrondi dans les calculs numériques) entraînent des résultats totalement différents pour de tels systèmes, rendant en général toute prédiction impossible à long terme.

Dans le modèle Cynefin, l'état chaotique est l'état d'un système qui complète les trois états du domaine de la logique – le Logos : les systèmes Simple, Compliqué et Complexe. C'est un système dans lequel aucune relation entre cause et effet ne peut être déterminée. Ce système a besoin d'être stabilisé. Les pratiques agiles ont notamment permis d'adresser la complexité des systèmes et des projets.

Le modèle Cynefin.

Ce modèle a été créé chez IBM en 1999 pour permettre aux personnes prenant des décisions de donner du sens à leurs approches et pratiques. Le nom est issu du gallois Cynefin qui signifie *Habitat*.
Pour pouvoir appréhender le désordre, il définit quatre états d'un système :



• **Simple :** le lien de cause à effet est direct : quand on fait tomber un domino, il fait tomber le suivant. Dans ce système, il suffit d'***observer*** l'environnement, de ***catégoriser*** l'action puis d'***agir***. On y applique des procédures, les meilleures pratiques.

• **Compliqué** : le lien de cause à effet est indirect, des experts sont nécessaires pour comprendre le mécanisme. Ici, il s'agira donc d'***observer*** l'environnement, d'***analyser*** avant de ***répondre*** par une bonne pratique, connue par les experts. Lancer une fusée est compliqué, mais avec un bon manuel et les connaissances nécessaires, c'est atteignable par tous.

• **Complexe** : dans ce système, les liens de causes à effets ne peuvent être appréhendés sans prendre en compte l'ensemble du système et ne sont transparents qu'a posteriori. Seule une approche systémique peut permettre par ***expérimentation***, d'***observer*** un résultat pour pouvoir ***répondre*** de manière pertinente. On y retrouvera ici les pratiques émergentes : Agilité, DevOps, Management 3.0.

• **Chaotique** : aucune relation de cause à effet ne peut être déterminée. L'objectif n'est donc plus de comprendre, mais d'***agir*** au plus vite pour revenir à la stabilité, d'***observer*** le résultat des actions menées pour ***répondre*** et s'adapter.

En résumé, on retiendra qu'il s'agit d'un état d'une extrême complexité, qu'une infime variation des conditions initiales peut conduire à un résultat radicalement différent. Il nous faut donc de nouvelles pratiques telles le Chaos Engineering pour être en mesure de revenir le plus rapidement possible à un état stable.

Définition.

« Everything fails all the time » – Werner Vogels, CTO d'Amazon.

A chaque instant, quelque chose, quelque part, tombe en panne. Il ne s'agit plus de croiser les doigts pour que ça n'arrive pas, mais de mettre en place la résilience nécessaire dans nos systèmes pour tolérer les pannes et de s'assurer que cette résilience est opérationnelle et fiable en production.

La discipline a été initiée par Netflix et définie ainsi par Casey Rosenthal, Responsable Trafic et Chaos Engineering :

« Discipline de l'expérimentation sur un système distribué afin de renforcer la confiance dans la capacité du système à résister à des conditions turbulentes en production[4]. »

Elle tente de répondre à la question :
« A quel point votre système est-il proche du précipice et peut sombrer dans le chaos ? »

Les grandes dates du Chaos Engineering :

– 2004 - Amazon – Jesse Robbins. Master of disaster, créateur des Chaos Gameday.

– 2010 - Netflix – Greg Orzell. @chaosimia - Première implémentation d'un Chaos Monkey pour renforcer l'usage de services stateless en auto-scale.

– 2012 - NetflixOSS met à disposition en open source la Simian Army.

– 2016 - Création de Gremlin Inc.

– 2017 - Sortie du livre Netflix *«Chaos Engineering[5]»*
Projet Open Source *«Chaos toolkit».*

– 2018 - Déploiement à l'international des concepts Chaos et la première Chaos Conf[6] en septembre 2018 à San Francisco.

[4] : *https://principlesofchaos.org/*
[5] : *http://www.oreilly.com/webops-perf/free/chaos-engineering.csp*
[6] : *https://chaosconf.splashthat.com/*

Principes.

Il s'agit bien d'effectuer des tests en production :

- *Expérimenter pour éprouver nos systèmes :* plutôt qu'attendre une panne, il s'agit d'en introduire une pour tester la résilience du système.
- *Expérimenter pour apprendre :* il ne s'agit pas de provoquer le chaos pour s'amuser, mais bien pour découvrir des faiblesses inconnues de nos systèmes.

Il s'agit, en revanche, d'expérimenter en production sur un système stable et performant. Il ne s'agit pas d'un jeu, mais de la vie réelle, avec des impacts humains et financiers potentiellement importants. L'ingénieur du Chaos n'est pas un savant fou, c'est un explorateur en recherche de connaissances sur le système qu'il étudie.

Dans le livre *Chaos Engineering, Building Confidence in System Behavior through Experiments*[7], les ingénieurs du Chaos de Netflix ont proposé ce protocole d'expérience :

- Définir quelle est la **question** que l'on souhaite poser au système : souhaite-t-on tester la résilience d'un composant, d'une application, d'une organisation ?
- Définir le **périmètre** de l'expérience : est-ce tout ou partie de la production ? Est-ce uniquement l'environnement technique seul ou cela inclut-il également les interventions humaines (surveillance, exploitation, support).
- Identifier précisément les **métriques** qui permettront de valider l'expérience et éventuellement de l'arrêter instantanément en cas d'impact critique.
- **Communiquer**, prévenir l'organisation de l'existence de l'expérimentation – pour éviter l'escalade en cas d'incident critique.
- **Réaliser** l'expérience.
- **Analyser** les résultats, mettre en place les éventuels plans d'action nécessaires.
- Elargir le scope pour la prochaine expérience.

Pour garantir la pertinence de l'expérience, il est essentiel de s'outiller tant pour l'injection des pannes que pour l'observation des effets. Faire un test une fois permettra de se rassurer sur la résilience du système, mais avec des changements permanents, la seule façon de bien dormir la nuit est d'automatiser l'expérience pour qu'elle se réalise en continu afin de suivre l'évolution du système.

3- Exemple d'expérimentation.

Chaos Monkey.

Netflix s'est doté d'un outil appelé Chaos Monkey pour tester la résilience de sa plateforme.

Cet outil stoppe aléatoirement des services systèmes et des instances de machines virtuelles. Les pannes induites permettent de tester la redondance de l'infrastructure et la reconfiguration dynamique du système dont le fonctionnement ne doit pas être altéré du point de vue des utilisateurs. La société présente ses objectifs de la façon suivante[8] :

[7] : *http://www.oreilly.com/webops-perf/free/chaos-engineering.csp*
[8] : *http://techblog.netflix.com/2014/09/introducing-chaos-engineering.html*

« Notre philosophie reste inchangée en ce qui concerne l'introduction de pannes en production, qui nous permet de nous assurer que nos systèmes sont tolérants aux pannes. Nous testons sans arrêt notre aptitude à survivre aux pannes qui n'arrivent qu'une fois, tous «les trente-six du mois». Comme manifestation de notre engagement dans cette philosophie précise, nous souhaitons doubler la mise sur la création de chaos, autrement dit l'injection de panne. Nous nous efforçons de simuler de manière contrôlée les types de pannes qui sont possibles dans notre environnement de production. Nous attendons de nos ingénieurs qu'ils écrivent des services qui résistent aux pannes et dégradent leur fonctionnement avec élégance autant que nécessaire. En continuant ces simulations, nous sommes en mesure d'évaluer et d'améliorer les vulnérabilités de notre écosystème. »

Début 2011, Netflix a créé et intégré la *Simian Army* qui présente d'autres types d'interruption :

- *Chaos Kong* qui fait tomber une région complète du cloud Amazon,
- *Latency Monkey* qui permet de tester la tolérance à la perte de performance d'un composant externe.

Chaos Gameday.

Pour tester la résilience de l'organisation et l'entraîner à réagir en cas d'incidents, Jesse Robbins, ex-pompier, ex– « Master of Disaster » chez Amazon, a mis en place le concept de Gameday qui consiste à simuler des pannes pour tester la capacité des équipes à réagir et revenir à une situation nominale.

Les objectifs d'une journée de GameDay AWS sont multiples :

- Partager les bonnes pratiques d'automatisation sur le Cloud AWS.
- Résoudre des incidents et des situations de crise dans un environnement de simulation similaire à celui de production.
- Identifier des axes d'amélioration pour assurer la résilience et la scalabilité de son application.
- S'améliorer collectivement, tout en apprenant dans un contexte ludique.
- Promouvoir l'organisation DevOps au sein des équipes par la mise en pratique.

Concrètement, le Gameday AWS s'appuie sur un environnement de simulation. Différents scénarios de panne sont lancés sur cet environnement et les équipes tentent de résoudre au plus vite les incidents, comme en situation de production. A l'issue de chaque scénario, un debriefing permet de travailler sur les axes d'amélioration : auto-scaling groups, security groups, outils de monitoring, etc.

OUI.sncf a décliné le concept GameDay en créant les Days-of-chaos[9] à destination de toutes les équipes IT et visant à l'entraînement à la détection, au diagnostic et à la résolution des incidents de production. Les Feature Teams jouent sur leur véritable environnement hors production, sur leur application et non sur une application virtuelle comme Unicorns.Rentals pour les Gamedays Amazon AWS[10].

[9] : *http://days-of-chaos.com/quest-ce-quun-doc/*
[10] : *https://aws.amazon.com/fr/twitch/gameday-essentials/*

4- Conclusion : par où commencer ?

Le Chaos Monkey de Netflix est éprouvé : c'est une valeur sûre pour ancrer la démarche du Chaos Engineering dans l'entreprise, en servant d'exemple simple. Il correspond cependant à un certain type de perturbations qui ne s'appliquent pas à tout le monde et qui ne permettent pas un apprentissage de tous types de faiblesses (sécurité par exemple). Avec le déploiement de cette démarche, de plus en plus d'outils sortent régulièrement et la plupart sont open-source. Ils viennent d'entreprises qui avaient un besoin adhoc (PowerfulSeal[11] de BloomBerg) ou d'une communauté de passionnés, comme Pumba[12] ou le ChaosToolkit[13] de ChaosIQ . Des offres SaaS comme celles de Gremlins Inc[14] commencent à atteindre une maturité permettant de démarrer plus facilement.

Un exercice de Chaos Gameday, dans sa version la plus simple est également une bonne façon d'aborder le sujet avec votre organisation. Nous vous invitons à consulter les articles de la **Paris Chaos Engineering Community**[15] sur Medium[16] pour des exemples de mis en place.

Nous espérons vous voir rejoindre rapidement cette communauté pour partager vos expériences.

[11] : *https://github.com/bloomberg/powerfulseal*
[12] : *https://github.com/alexei-led/pumba*
[13] : *https://chaostoolkit.org/*
[14] : *https://www.gremlin.com/*
[15] : *http://meetu.ps/c/3BMlX/xNjMx/f*
[16] : *https://medium.com/paris-chaos-engineering-community/*

PARTIE III
RETOURS D'EXPERIENCE


POUR LES TESTS D'ACCEPTATION DE LA PIZZA TEAM,
D'ACCORD POUR QU'OBÉLIX TESTE LES PIZZAS,
MAIS À CONDITION QU'IL EN LAISSE UN PEU POUR LES AUTRES!
J'AI ENCORE FAIM.
KING SIZE PIZZA
F. COINTE

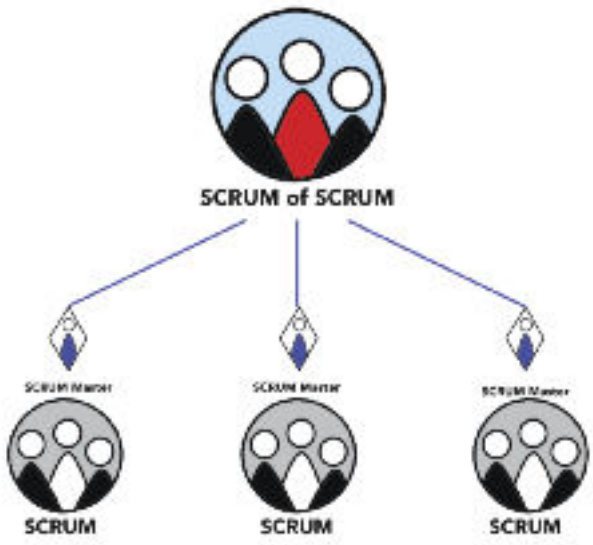
III.1- Retour Experience - Ingénieur QA agile dans l'Agilité à l'échelle.

Par Marcelo Kamenetz Szwarcbarg

En rejoignant Amadeus, j'ai eu l'opportunité de rejoindre un projet récent qui proposait un changement majeur: la migration vers les méthodes agiles.
Le fait de débuter avec une nouvelle méthodologie dans un nouveau projet est intéressant, car cela permet d'innover sans cesse, sans impacter négativement le projet.

Le projet étant trop important et complexe, il ne pouvait être effectué par une seule équipe SCRUM. Une solution a été adoptée : l'Agilité à l'échelle avec le « Scrum of Scrum ».
Nous avons donc commencé avec plusieurs équipes Scrum, initialement une par domaine fonctionnel. Pour orchestrer et centraliser les avancements, un « Scrum of Scrum » a été mis en place.
Le « Scrum of Scrum » est une méthode d'Agilité à l'échelle qui permet d'avoir sur un projet plusieurs Scrum (équipes composées de 8 personnes en moyenne) travaillant sur des modules différents tout en restant synchronisés.

Une équipe Scrum composée des Scrum Masters, des représentants de leur équipe, est formée avec un Super Scrum Master qui organise et coache ; et un Super Product Owner qui gère les livraisons à un niveau projet.



Dans ce Scrum au-dessus du Scrum, nous avons un backlog de tâches, qui est ensuite adressée aux Product Owner des Scrum afin de donner une meilleure granularité aux développements nécessaires en User Story.

Ce que j'ai apprécié dans cette organisation, c'est la capacité de prévoir les dépendances des livraisons entre les équipes et la mise en avant de la collaboration à grande échelle.

Chaque point remonté par le Scrum Master était discuté au « Scrum of Scrum » et les solutions étaient partagées entre les équipes. Tous les intervenants participaient aux évolutions du projet.
De plus, l'organisation des livraisons devenait simple : toutes les User Story sont reliées à une tâche de la « Scrum of Scrum ». En un point, nous avons la liste des développements d'une livraison avec les dépendances et des critères d'acceptation moins unitaire (tests End to End, par exemple).

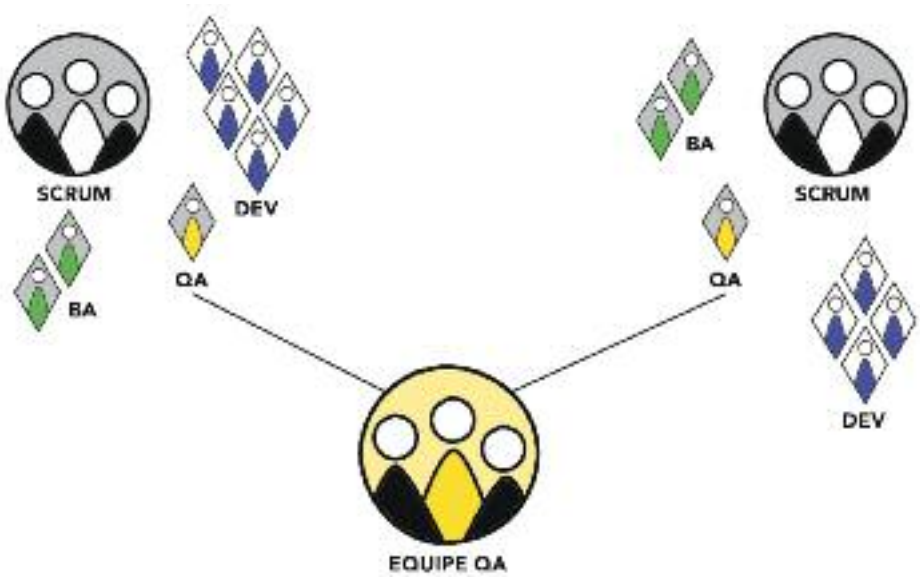
Le risque de la méthodologie Scrum sur un projet constitué de plusieurs équipes est la perte de la vision globale du projet/produit. Chaque Scrum se focalise sur son domaine fonctionnel et peut s'isoler. Ceci peut donner des dérives avec des développements de deux équipes SCRUM qui deviennent incompatibles.

Dans mon cas, le Super Scrum Master et les différents Scrum Master ont bien compris ce risque et ont toujours accordé beaucoup d'importance aux phases d'intégration des composants et aux tests end to end.

En parallèle de l'installation des méthodes agiles, l'entreprise procédait également au recrutement d'experts en qualité afin d'accompagner les équipes. Ce changement a reçu une appellation interne : « la QA transformation ». Et c'est ici que j'interviens.

En tant qu'ingénieur en qualité, j'ai rejoint le projet peu après son lancement, en intégrant une des Scrum de la « Scrum of Scrum » dans une équipe de huit personnes, composée de développeurs et d'experts en définition de produits (analyste métiers).

Dans une Scrum, nous sommes tous développeurs. Cependant, l'organisation a maintenu une gestion par domaine de compétences en terme hiérarchique.
En tant qu'expert en qualité logiciel, j'appartiens à mon équipe Scrum mais aussi à une équipe d'ingénieurs qualité qui est elle-même divisée entre les équipes Scrum. Il en est de même pour les autres domaines d'expertises.
Chaque équipe d'expert (Développeurs – DEV, Business Analystes – BA) envoie donc ses membres pour composer l'équipe Scrum. Un seul expert qualité était assigné par Scrum.



Le point fort de cette organisation en communauté est la collaboration entre experts du même domaine. Une idée émergeant d'une Scrum peut facilement être partagée et propagée. Avoir une communauté d'experts permet d'introduire des incubations d'idées ou des réunions dédiées à l'innovation. Ce point est illustré un peu plus tard avec l'évolution d'outils et de processus suivis.

Le risque d'une telle organisation, qui est visible sur le schéma précédent, est la ségrégation des domaines d'experts. Une Scrum n'étiquette pas ses membres en fonction de leur domaine d'expertise. Nous ne parlons pas d'analyste business ou de testeur, mais de membres du Scrum (ou juste DEV en cas d'abréviation).
Ici, chaque membre de la Scrum s'identifie dans un rôle, ce qui est contradictoire à l'esprit de la méthode Scrum et plus généralement des méthodes agiles.

Cela peut introduire des tensions, notamment quand un expert se voit attribuer une tâche qui n'entre pas dans son domaine.
J'ai fait face à plusieurs reprises à ce rejet quand les tâches de tests étaient attribuées à une personne non-incluse à l'équipe QA. Le conseil que je peux donner pour régler ce problème est l'utilisation d'une matrice de compétences, afin de convaincre les plus récalcitrants.

Montrez aux membres de la Scrum qu'ils sont capables d'effectuer des tâches de tests en rappelant les règles du Scrum. Ne soyez pas trop conciliant, sinon les tâches ne seront pas réalisées. Soyez diplomate afin de ne pas aggraver les tensions.
Et surtout : remontez le problème s'il est impactant au Scrum Master et aux Managers. Ils sauront soutenir les principes du Scrum afin que le projet se déroule normalement.

Ayant rejoint le projet après son départ et étant le seul ingénieur qualité de la SCRUM, mon premier défi a été d'évaluer et de comprendre les choix et implémentations effectués jusqu'à présent. En termes de produit mais aussi de procédures et outils sélectionnés par l'équipe.

J'accorde énormément d'importance sur les choix effectués, ils représentent les objectifs et la mentalité de l'équipe : des outils simples à utiliser mais demandant beaucoup de tâches manuelles montrent une volonté de simplicité.
Pour une meilleure efficacité, il faut avoir des outils automatisés qui sont en revanche plus complexes à utiliser et demandent plus de connaissances techniques.
En début de projet, à l'exception des outils imposés dans la stratégie de tests, les outils étaient simples et manuels. Notre solution avait un nombre de tests limités et maintenables manuellement. La multiplication de ces tests au fur et à mesure du projet a forcé le changement de certains outils.

En parallèle, j'ai été responsable de l'introduction de quelques nouveaux outils de suivi et de rapports de qualités en utilisant notamment un outil comme ALM HP.
Avec une équipe QA, il est plus aisé de s'aligner sur les outils de rapports de tests. Une harmonisation des rapports permet une réduction des mails grâce à la mutualisation. De plus, les échanges avec les personnes impliquées dans le projet deviennent plus faciles, les mêmes références sont utilisées indépendamment du Scrum.

L'installation de nouveaux outils dédiés pour une activité dans la Scrum s'est révélée être assez délicate, en raison de la résistance aux changements de certains membres des équipes ou de la réticence à donner de la visibilité sur la qualité du produit dès sa conception.

Mon astuce afin de contrer cela est de bien prendre le temps d'expliquer, avec des exemples concrets, ce que ces outils peuvent apporter au projet et à l'équipe.
Donner de la visibilité de la qualité permet une meilleure planification du sprint à venir. Les éléments non validés et bloquants sont priorisés en bonne et due forme. Les équipes impactées sont rapidement informées afin de ne pas s'engager sur un développement en début de sprint qui sera bloqué pendant ce dernier.

Mon travail d'expert qualité s'introduit parfaitement dans cet échange entre le résultat des tests et les réunions de début de sprint ou des plannings poker. Une User Story ayant un risque élevé ou ayant une dépendance sur une fonctionnalité dont les tests ne sont pas encore passés, peut avoir une estimation revue à la hausse.

Au fur et à mesure que le projet avance, la complexité des fonctionnalités et des tests s'accroît. Il est indispensable de revoir constamment les outils et procédures utilisés.
Par exemple, pour le stockage des tests, nous utilisions CVS. L'outil est facile à prendre en main, intégré à l'explorateur Windows et propose une gestion de version simple, basée sur des descriptions.

Cependant, l'outil ne permet pas de visionner facilement les changements effectués dans un test et ne donne aucune visibilité sur les revues faites pour les tests.
Certains tests automatiques se sont dégradés à cause de revues ignorées.

Afin de pallier ces deux soucis, j'ai proposé dans mon équipe Scrum une migration vers Git pour la gestion des tests. Ce qui est appréciable avec Git est son intégration avec des outils tels que BitBucket (Altassian), qui permet d'avoir une vision en ligne des documents ainsi que des changements effectués sur ces derniers. De façon à partager avec un groupe ou une entité (exemple : une équipe) via une application web simple et ergonomique. De plus, Git était déjà utilisé par les développeurs, cela a donc permis de limiter le nombre d'outils et d'inclure plus facilement les développeurs dans les tâches de test.
Ainsi, nous avons pu ajouter des vérifications automatiques sur le respect des procédures, par exemple la revue. Une modification opérée se doit d'avoir : un titre, une description, l'identification de l'auteur, l'approbation d'une personne autorisée par l'outil ainsi que son identification.
Tout changement est suivi et peut être facilement inversé ou changé car rapidement identifiable.

Travailler en Scrum se révèle être très enrichissant. Contrairement à mes autres expériences où mes compétences de testeur étaient suffisantes, le Scrum requiert des compétences humaines. Communication, compréhension, adaptabilité, persévérance et gestion de soi seraient les cinq compétences que j'assimilerais avec le métier de testeur agile.

A chaque jour de travail, nous collaborons avec une équipe hétéroclite. Savoir discuter pour avoir des informations sur le développement en cours est tout aussi important que savoir donner de son temps pour expliquer ou aider ses coéquipiers en Scrum.
Le fait d'être testeur agile ne se limite pas au simple fait d'être testeur. L'unique contrainte est la priorisation.

Précédemment, j'ai évoqué le fait qu'une personne non experte en qualité pouvait prendre des tâches sur la qualité. Il en en est de même pour le testeur qui peut se voir dans la contrainte, ou plutôt l'opportunité, de s'aventurer sur un domaine où il est compétent mais non expert.
Il m'est arrivé de mettre le test de côté pour aider dans la spécification du produit. Un membre du Scrum peut participer à toutes les tâches auxquelles il se sent capable de fournir une valeur ajoutée.

Pouvoir travailler en dehors de son domaine de confort est vraiment positif. J'ai pu développer de bonnes compétences en compréhension du produit et en communication. Pour certains, cela permet même de se découvrir des talents. Par exemple, j'ai eu l'occasion de travailler avec un développeur qui s'est intéressé aux métiers liés à la qualité logicielle en menant plusieurs tâches de tests.

Attention cependant à ne pas prendre un travail pour lequel vous n'avez pas de compétences suffisantes, surtout si la partie est critique ou urgente. L'objectif reste de faire avancer le projet, une personne experte sera plus adéquate pour les tâches les plus sensibles afin de ne pas impacter les livraisons. Je me suis déjà retrouvé en situation de difficulté en prenant une tâche de définition de produit bien trop compliquée pour mon niveau. Heureusement, nous ne sommes jamais seuls en Scrum.

Nous travaillons dans les mêmes bureaux et nous nous côtoyons quotidiennement. Cela forge un esprit d'équipe fort. A plusieurs reprises, j'ai constaté que les membres d'une même Scrum s'entraident au travail et se lient d'amitié. Pour ma part, un grand nombre de mes collègues sont également des amis.

Cela semble être un détail, mais cette collaboration poussée permet de mieux appréhender les difficultés. De plus, cela permet de discuter ouvertement sur des problématiques, communes ou non, et d'avoir de meilleurs points de vue.

Nous ne pouvons pas nous entendre avec tout le monde, mais quand nous instaurons un esprit d'équipe fort dans une Scrum, les concessions sont plus faciles à faire pour tout le monde : on fait cela pour le bien du projet, et pour le bien de la Scrum.

Si je devais décrire ma journée type en tant que testeur agile :

J'arrive souvent en premier, tôt le matin. Les environnements de tests sont peu utilisés, donc plus rapides. Je profite également de ma solitude à cette heure-là pour vérifier les résultats de la campagne de non régression exécutée la nuit, en buvant mon café. Certains préfèreront lire le journal, mais mes actualités au travail sont les nouvelles erreurs remontées par mes tests.

Aux alentours de dix heures, une fois tous mes collègues présents, je participe au stand up, avec toujours un petit mot sur l'état de notre backlog de bug. J'ai toujours été intransigeant sur la résolution de bug : il ne faut jamais laisser un bug critique non urgent devenir critique et urgent. La matinée peut varier selon les tâches à exécuter : en première semaine de sprint, quand peu de code est délivré, je travaille sur l'optimisation de nos processus et de notre régression. On peut toujours mieux tester sans obligatoirement tester plus. Je m'impose également une pause-café avec des testeurs d'autres domaines fonctionnels afin de discuter tranquillement sur des sujets variés, dont les méthodologies de tests. Les discussions sont toujours intéressantes.

Les autres semaines, je priorise le test des développements en cours ou terminés pour le sprint. Un retour rapide est toujours apprécié, même si les développeurs restent déçus s'il y a un bug. Mon objectif dans le test n'est pas de trouver absolument un bug. Pour moi, un testeur agile doit couvrir dans ses tests tous les cas critiques et importants pour le client et être sûr que le délivrable n'apporte pas de risques.

Mes collègues m'ont rarement dit : « Si tu es souriant, c'est que tu as trouvé un bug ». Même si j'étais satisfait de trouver un de ces bon bugs bien impactants avant le client, on est loin d'un cas de schadenfreude (joie malsaine).

Le repas du midi se devait d'être partagé avec des collègues de la Scrum ou d'autre testeurs. C'est le meilleur moment de la journée pour échanger des idées et des avis sur tout, y compris les projets ou nos processus. Bien que, souvent, nous parlons plus de sports et technologies.
Les après-midi ressemblent à mes matinées. Certains jours, nous avons des cérémonies Scrum, comme le raffinement pour discuter des User Story.

Sur l'ensemble de la journée, je n'hésite pas à inviter à une pause-café un petit nombre de collègues pour discuter rapidement de problèmes ou afin de demander des éclaircissements.

En conclusion, travailler en Agile, dans une Scrum intégrée dans une Scrum of Scrum, est très intéressant en raison des défis qu'une telle organisation peut apporter.

Il est important de garder un esprit agile, mais non incompatible avec une hiérarchisation par domaine d'expertise si les équipes comprennent leur rôle dans la Scrum : ne pas être qu'un expert dans un domaine, mais être un membre à part entière dans une équipe qui s'adapte et évolue.

Le travail du QA reste important comme dans tout projet, mais la communication y est une qualité indispensable. Il faut pouvoir discuter avec les Scrum master et les membres de son équipe.
Je reste entièrement satisfait de mon expérience dans cette organisation qui m'a apporté énormément, tant au niveau professionnel qu'au niveau personnel.

III.2- Retour Experience - Echec d'un projet/produit agile.

Par Marc Hage Chahine

Dans ce chapitre, nous nous attarderons sur un projet agile qui fut un « échec ». Par échec, je ne veux pas dire pas que le produit n'a jamais vu le jour mais plutôt que le projet, ou devrais-je dire le produit, a :

- Explosé les coûts prévus.
- Explosé le temps prévu (près de 18 mois au lieu des 6 annoncés initialement).
- Eté beaucoup moins efficace que le même type de projet en cycle en V.

Afin de savoir comment cela est arrivé – et donc de vous permettre d'éviter ces erreurs – nous allons étudier ce cas en le présentant, en resituant le contexte mais surtout en analysant les erreurs commises.

Il est important de noter qu'un grand nombre de ces erreurs ont été remontées et corrigées au cours et à la fin du projet. Ce projet, qui était le projet pilote, aura finalement été très instructif et aura permis de beaucoup mieux implémenter l'Agilité sur les projets d'après.
Les différentes parties de ce chapitre seront :

1. Le projet.
    a. Contexte du projet.
    b. L'équipe projet.
    c. L'implémentation de l'Agilité à l'équipe.
    d. Les dérives liées au « cycle en V ».
    e. L'impact de la non-qualité.
    f. Epilogue du projet.
2. L'analyse des erreurs.
    a. Pourquoi ces retards ?
    b. Echec sur l'état d'esprit.
    c. Ce que ce projet a apporté.
3. Conclusion.

1. Le projet.

Tout d'abord, je tiens à préciser que le terme « projet » est voulu. Dans les méthodes agiles, on parle généralement de produit mais dans ce contexte nous verrons que l'aspect projet avec un cycle en V classique a fortement impacté de nombreuses décisions qui se sont avérées, sur le moyen terme, de graves erreurs pour un produit agile.

a. Contexte du projet.

Ce projet s'est déroulé au début des années 2010, il y a plus de 5 ans au moment où j'écris ces lignes. Le but du projet était de proposer une application mail mobile pour tablettes.

Nous étions dans une équipe expérimentée qui avait une forte expérience de ce type d'application. Cette même équipe avait, à ce moment, déjà développé avec succès des web-applications ainsi que des applications natives qui recevaient un très bon accueil de nos clients.

Forts de cette expérience et profitant du succès des autres applications développées, nous avons commencé le projet en toute confiance, en ayant décidé de développer cette application avec une méthode très récente et qui semblait prometteuse : le Scrum.

Nous avons commencé par nous former, étudier les différentes cérémonies, la gestion du backlog, des sprints,…

Suite à cela, nous avons commencé le projet.

b. L'équipe projet.

Nous étions conscients que cette application servait également de projet pilote pour cette méthode de développement et avions à cœur de bien faire. Pour composer l'équipe agile, nous avons fait appel à des développeurs expérimentés sur des application mails avec une lead tech (oui, notre lead tech était une femme, très compétente et consciencieuse), notre business analyste le plus compétent et expérimenté ainsi que moi-même. J'étais le testeur attitré car j'avais travaillé sur toutes les autres applications au fonctionnel similaire. Nous avions même un « ingénieur DevOps » – en fait l'équivalent d'un Ops – qui assistait à nos cérémonies et travaillait en partie sur le projet. En revanche, le langage de programmation était nouveau pour tout le monde.

Néanmoins, nous pouvons voir que, comme je l'ai conseillé dans le chapitre sur la transformation agile, nous n'avions pas donné ce projet pilote à une équipe de débutants.

Par rapport à la constitution d'une équipe « pluridisciplinaire » l'équipe était déjà habituée à travailler dans le même bureau et à beaucoup échanger, certaines bonnes pratiques agiles sont aussi des bonnes pratiques pour le cycle en V. En tant que testeur agile, j'ai travaillé sur deux activités principales : la revue des spécifications et les tests, mais aussi sur d'autres projets, si besoin de testeurs pour des campagnes. Nous touchions dès lors un problème sur « l'équipe agile » car, à part les développeurs, les autres membres (le testeur, le DevOps, le business analyste) n'étaient pas à plein temps dans l'équipe. De même, lors des campagnes (manuelles) de test, l'automatisation s'étant avéré un échec, j'étais accompagné… d'autres testeurs, au lieu des autres membres de l'équipe agile.

c. L'implémentation de l'Agilité à l'équipe.

L'Agilité devait permettre de livrer plus vite l'application, le « temps nécessaire » alloué aux différentes fonctionnalités avait alors été diminué par rapport aux estimations de notre lead Tech afin de pouvoir livrer un produit complet dans le temps disponible (6 mois).

Bon, ça ne faisait pas très « Agile » :

- Les membres de l'équipe n'étaient pas totalement dédiés à l'application.
- il y avait des membres « temporaires » pour les campagnes de test.
- la durée totale ainsi que le périmètre final étaient déjà prévus.

Nous avions remarqué ces points, les avions fait remonter de manière informelle à notre hiérarchie mais avions tout de même décidé de commencer dans ces conditions. Le point fort de l'Agilité n'est-il pas justement de savoir s'adapter ?

Nous avions donc décidé de commencer à travailler sur notre projet. Nous avions six sprints de quatre semaines chacun qui étaient prévus. Chaque sprint était déjà pré-rempli avec des User Story à livrer pour ces sprints. Bref, nous avions donc un périmètre (en terme de fonctionnalité et de temps) bien défini. Cela ressemblait déjà fortement à un cycle en V, ou tout du moins à six mini-cycles en V. Cette ressemblance s'est vite avérée être davantage que cela, ce qui a été la source d'un grand nombre de nos maux.

d. Les dérives liées au « cycle en V ».

C'est ici que le bât blesse. Le cycle en V que nous maîtrisions parfaitement a refait surface et a débordé un peu partout sur notre projet.

Certes, nous faisions des daily meeting quotidiens. Ces daily meeting étaient d'ailleurs bien implémentés et ne duraient jamais plus de quinze minutes, chacun parlait de ses tâches, de ses difficultés, … Nous avions également une présentation des User Story pour présenter le sprint backlog.

Néanmoins, le contenu du sprint était déjà convenu à l'avance et il était impossible de le modifier. De même, par manque de temps, nous n'avons pas fait de rétrospectives. A quoi bon ? On échangeait tous les jours, s'il y avait quelque chose à faire, il suffisait de le dire, non ?

Bref, nous nous sommes retrouvés dès le premier sprint avec un grand nombre d'US à implémenter. Ces US ne pouvaient pas être décalées car le deuxième sprint était déjà plein. Les développeurs ont donc développé aussi vite qu'ils ont pu, pour livrer l'ensemble des US à quelques jours de la fin du sprint. Comme pour du cycle en V, les tests se retrouvaient à la fin. Et bien évidemment, comme pour un cycle en V avec des temps estimés trop faibles, les tests ont « retardé » la fin du projet… pardon, du sprint !

De plus, les développements ayant été faits trop vite, les tests ont remonté un fort taux d'anomalies plus ou moins bloquantes. Un certain nombre de ces anomalies ont été corrigées dans le premier sprint qui aura finalement duré six semaines (au lieu de quatre), les autres bugs, quant à eux, ont été ajoutés au sprint 2.

Sprint 2, l'histoire se répète avec un retard supplémentaire, dû au périmètre encore plus important que le périmètre suivant.

Et cela a continué jusqu'à la fin du sprint 4. A ce moment-là, l'impact de la non-qualité ne pouvait plus être ignoré !

e. L'impact de la non-qualité.

En Agile, on a coutume de dire que la qualité est non négociable. La théorie explique que la dette fait un effet boule de neige, que l'on est vite dépassé et qu'à la fin, on ne peut plus rien garantir ou maintenir.

La pratique va encore plus loin !

Souvenez-vous du paragraphe précédent. « Certains bugs » étaient remis à plus tard, les développements étaient faits aussi rapidement que possible. Dans notre cas, le travail des développeurs s'est mué en 100% de correction de bugs. Malheureusement, chaque correction engendrait de nouveaux bugs… De nouveaux bugs étaient détectés à chaque utilisation, je n'avais besoin que de dix secondes pour faire crasher l'application, le composeur n'envoyait pas tous les caractères que nous avions écrits… Bref, les nouvelles fonctionnalités ne pouvaient plus être développées, nous étions totalement bloqués avec l'impossibilité d'avancer sur le produit. La solution, nous l'avions identifiée. Mais malheureusement trop tard. Elle s'avérait en fait assez simple. Cette solution était de mettre en échec les US qui n'étaient pas finies au sprint 1 pour les mettre au sprint 2. De revoir le backlog du sprint 2 en fonction et de faire une rétrospective. Cette solution n'était malheureusement plus implémentable et nous avions atteint un point de non-retour : six mois pour s'apercevoir que nous étions totalement bloqués !

f. Epilogue du projet.

Nous avons finalement décidé de repartir de (quasiment) zéro mais en étant, cette fois-ci, intransigeants sur la qualité, en gardant une durée de sprint toujours égale et en modifiant le périmètre si besoin. Nous avions également suivi, entre temps, des formations et ateliers de formateurs agiles compétents. L'expérience des six premiers mois a aussi permis à l'équipe de mieux appréhender le langage qui n'était pas connu au départ… En mettant réellement en place l'ensemble des cérémonies, car nous avions appris – au prix fort – leur utilité.

Le projet est alors reparti sur de bonnes bases, les tests ont été le plus possible automatisés, les tests vitaux communiqués aux développeurs et le produit a pu être développé et proposer une version au niveau de qualité souhaité.

En revanche, certains reliquats du cycle en V sont restés. Nous considérions avoir atteint un produit minimal mais n'avons été autorisés à proposer le produit au client qu'une fois l'intégralité du périmètre développé. De plus, le « métier » qui n'avait pas assisté aux démos (je sais, ce n'est pas agile non plus) voulait avoir sa session de test (la partie tout en haut à droite du cycle en V) à la toute fin du projet.

Néanmoins, ce projet arriva à une fin heureuse… Mais une fin beaucoup plus tardive que prévue et un projet beaucoup plus onéreux également.

2. L'analyse des erreurs.

a. Pourquoi ces retards ?

Ce projet d'une durée prévue de six mois a finalement eu un an de retard. Il y a plusieurs raisons à cela, certaines ne sont pas liées à l'Agilité et sont :

- Le fait d'avoir dévalué les estimations de notre Lead Tech (qui se sont avérées, à terme, exactes à plus de 90%). Ce problème n'est pas lié directement à l'Agilité mais plutôt à la dérive liée au fait de passer à l'Agilité : sortir l'application plus tôt ne signifie pas sortir l'application avec le même périmètre plus tôt !

- Avoir négligé la qualité dans le but de proposer plus rapidement quelque chose.
- Avoir un périmètre figé, ce qui est contraire à l'esprit agile. En Agilité, la variable, c'est le périmètre !

b. Echec sur l'état d'esprit.

Le principal problème sur ce projet vient justement du fait que l'application était considérée comme un projet et non comme un produit. Nous avons travaillé comme dans un cycle en V mais dans un carcan Scrum.

Le résultat a été désastreux. Ce qui peut être acceptable sur du cycle en V – c'est-à-dire faire des concessions sur la qualité avant une mise en service – est totalement incompatible avec le Scrum et les sprints où chaque concession se retrouve à coûter beaucoup plus à moyen, voire à court terme.

Un autre problème est que par « adaptabilité » de l'Agilité nous avions compris que l'on pouvait faire tout et n'importe quoi, que le modèle d'adapterait. Là encore, cela s'est avéré totalement faux. L'Agilité est là pour s'adapter au changement, aux retours des utilisateurs. L'adaptabilité n'est pas au niveau de la qualité, de la méthodologie ou même des processus (qui doivent être décidés par l'équipe et qui restent modifiable lors des rétrospectives) mais au niveau du périmètre. Malheureusement, nous avions implémenté exactement le contraire, tout était adaptable à volonté… sauf le périmètre.

c. Ce que ce projet a apporté.

Ce projet pilote, même s'il s'est avéré être un échec d'un point de vue projet, a finalement été une mine d'informations. Il a joué son rôle de projet pilote en exposant les difficultés que tout autre projet aurait rencontré dans notre contexte.
Nous y avons notamment appris :

- **L'importance de penser produit et pas projet** : penser projet induit une dérive cycle en V avec les tests à la fin, garder toujours le même périmètre, utiliser le délai ou la qualité comme variable d'ajustement.
- **L'importance d'automatiser les tests** : l'exécution des tests prend très vite beaucoup de temps. L'Agilité « force » une automatisation des tests qui peuvent revenir plusieurs fois dans la semaine.
- **L'importance de bien gérer ses tests** : et savoir trier entre les tests utilisés une seule fois et ceux qui seront utilisés.
- **La puissance des tests exploratoires** : avec la possibilité, après avoir ciblé des risques, de faire des sessions pour les couvrir sans concevoir et écrire à l'avance des tests qui ne seront exécutés qu'une unique fois.
- **L'intransigeance sur la qualité** : là encore, cela parait basique, on nous le répète à volonté, cela fait même partie des piliers de SaFe… Néanmoins on n'adhère généralement à ce concept qu'après avoir subi le « traumatisme » de la non-qualité. Cela est encore plus important pour toutes les méthodes incrémentales (et donc toutes les méthodes agiles). On ne construit pas sur des fondations friables. En Agilité, les fondations, c'est le produit lui-même !

- **La flexibilité sur le périmètre** : C'est le vrai point fort de l'Agilité. On s'adapte aux retours des utilisateurs, à leurs demandes, on peut définir un produit minimal pour être en production plus rapidement.
- **L'importance d'avoir une formation appropriée** : on ne devient pas agile en lisant des livres. Une mise en pratique et une formation appropriées, notamment à travers des ateliers, sont un passage primordial et permettent d'éviter un grand nombre d'erreurs.
- **L'importance de chaque cérémonie Scrum**… Et notamment la rétrospective qui aurait permis, dès le sprint 1, de rectifier certains problèmes mais surtout de ne pas se retrouver dans l'impasse de la fin du sprint 4.

Au-delà de cet apprentissage, ce projet nous a permis de mieux cerner la « philosophie agile ». Là où nous considérions, comme beaucoup, que le Manifeste Agile pouvait se traduire par l'abandon des processus et de la documentation, nous avons découvert qu'en fait ces derniers restaient très importants mais l'étaient moins que les interactions entre individus ou avoir des logiciels fonctionnels. Pour aller plus loin, je pourrais même dire que nous étions plus agiles avec notre pratique du cycle en V qu'avec notre implémentation initiale du Scrum !

3. Conclusion.

Une transformation agile ne se décrète pas. Une transformation agile n'est pas simple. Une transformation agile prend du temps !

On ne devient pas agile du jour au lendemain.

De plus, il est important de noter que l'Agilité n'est pas toujours la meilleure solution. Dans notre contexte, un cycle en V aurait objectivement été une bien meilleure solution (si on ne prend pas en compte le fait que l'on avait besoin d'un projet pilote) pour les raisons suivantes :

- Les contraintes de temps se sont avérées plus informelles que vraiment présentes.
- Pas de nécessité de sortir « avant les autres » pour investir un marché.
- Nous avions un périmètre défini et connu à l'avance (c'était particulièrement vrai avec notre expérience sur les autres applications au fonctionnel similaire).
- Nous maîtrisions très bien le cycle en V.

Enfin, l'Agilité ne doit pas être prise pour ce qu'elle n'est pas.

- Agile ne veut pas dire plus rapide : la rapidité est une rapidité d'arrivée sur le marché (le TTM ou Time To Market), en revanche, on ne sait pas avec quel produit. Ce qui est sûr, c'est que cela ne sera pas un produit final comme on peut le voir avec du cycle en V.
- Agile ne signifie pas non plus dire adieu à la documentation, aux processus et à la rigueur. L'Agilité requiert une très grande rigueur (plus que le cycle en V) et ne permet pas d'avoir un produit défini à l'avance plus rapidement et moins cher qu'un cycle en V.

L'Agilité c'est la possibilité d'avoir :

- Une mise sur le marché plus rapide.

- Des mises à jour régulières.
- Une adaptation plus réactive aux retours des utilisateurs.
- Une gestion des changements fonctionnels liés à différents contextes.

III.3- Retour d'expérience sur les tests dans le cadre de SAFe chez ORANGE à la DTSI.

Par Claude Barrau et Fabrice Grimbert

Introduction.

Par Sandrine BREBANT,
Directrice de l'Ingénierie et de l'Expertise du Système d'information (DIXSI).

Le test nous apparaît comme un pilier incontournable lorsqu'on veut proposer de nouveaux services de qualité à nos clients et répondre à leurs attentes.

Les nouvelles méthodes de travail (Agile, SAFe) et la rapidité avec laquelle nous évoluons aujourd'hui nous confortent dans l'idée que l'automatisation des tests est un élément clé.
Les technologies sont en perpétuelle évolution et les outils à notre disposition nous permettent de relever ces nouveaux challenges.

Une démarche très structurée peut être suivie pour adapter la stratégie de tests au nouveau contexte amené par SAFe…

1- Contexte Orange.

Nous sommes un centre d'expertise du test, de l'intégration et du développement au service de la Direction Technique du Système d'Information (DTSI) chez Orange France.

C'est dans notre ADN de fournir l'ensemble des outils et services nécessaires à tout projet de la DTSI dans la délivrance de leur application, tout en garantissant la qualité.

Les projets n'ont pas vocation à être experts des outils mis à leur disposition, ils doivent avoir la liberté de délivrer leur application dans les temps avec la meilleure qualité possible, en utilisant un écosystème d'outils cohérent, interfacé et le plus « facile » d'usage possible.

L'arrivée de « l'Agile Coordonné » reposant sur la méthodologie SAFe (Scaled Agile Framework) a demandé quelques adaptations pour que nous puissions continuer à apporter notre expertise aux projets. Il nous faut continuer à satisfaire les applications de notre Legacy et proposer de nouvelles solutions pour ce Framework.

Tous les projets ne partent pas en méthodologie SAFe : sur les quelque 1200 projets que compte le SI d'Orange France, seulement 400 sont réellement concernés par ce Framework.

Beaucoup de solutions existent et peuvent couvrir les besoins SAFe, les méthodologies de tests agiles sont adaptées, mais il faut garder à l'esprit que mettre ensemble des projets agiles ne garantit pas la réussite. Il faut intervenir à plusieurs niveaux.

Nous avons donc constitué une équipe de coachs du test assistée d'automaticiens de tests pouvant intervenir sur n'importe quelle problématique de tests, tant sur la stratégie que sur le développement de tests automatisés, mais aussi capable d'accompagner les équipes en place

pour leur permettre d'aborder SAFe dans les meilleures conditions et de monter en compétence le plus rapidement possible.

Voici donc un retour d'expérience d'un train SAFe chez Orange France.

2- Retour Expérience Train Orange.

a. Contexte Projet.

De nouvelles offres sont proposées aux multiples clients Orange (Mobiles, Internet, TV, Objets connectés, Bank, …) à travers différents canaux tels que les boutiques Orange ou via les sites web institutionnels d'Orange.

Le développement d'une offre ou d'un nouveau service nécessite la mise en commun de multiples applications de l'écosystème Orange et des équipes multi-compétentes.

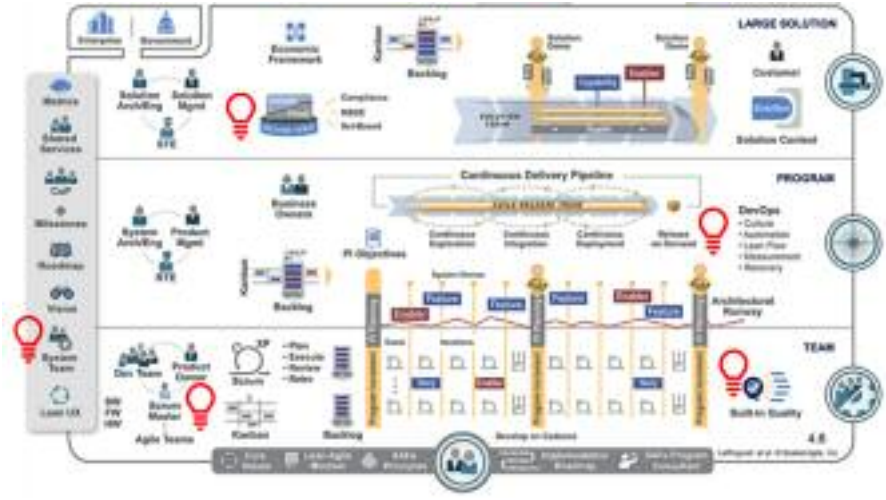
Dans ce contexte, le virage agile des nouveaux projets s'est organisé autour du DevOps et du Framework SAFe sous le concept d'« Agile Coordonné ».

b. Agile Coordonné (cadre SAFe) dans le contexte Test.

Les réflexions qui suivent portent sur des éléments clés des activités de Testing, identifiés dans le cadre SAFe mis en œuvre sur les projets.

Une structuration en trois niveaux :

- Partie Sponsor Business, identifiant les évolutions d'offres et de services, représentée par le niveau Solution.
- Partie ART (Agile Release Train), identifiant le projet de pilotage de la mise en œuvre de la solution, représentée par le niveau Program.
- Partie Développement, identifiant la réalisation de la solution, représentée par le niveau Team. Ce niveau Team étant multiplié par autant d'équipes de développement intervenant dans le Train.



Organisation tout à fait typique au cadre SAFe avec la mise en place d'une équipe transverse au Train/Team's, la célèbre ***« System Team »***.

c. Problématiques dans la mise en œuvre de SAFe.

Le tableau suivant liste les principaux problèmes rencontrés dans l'utilisation du cadre SAFe.

Niveau	Artefacts	Problème
Solution	les principaux artefacts issus de ce niveau sont : • les CAPABILITIES (ou EPICS suivant le langage JIRA) • L'architecture de la Solution • Le MBSE (Model-Based Systems Engineering) : Modélisation de la solution système.	Equipe business/marketing orientée sur le côté « Business Value » des nouvelles offres. Cela a pour conséquence un manque dans l'aspect de définition de l'architecture de la solution système et de vision globale de la solution en terme de déroulement d'activités.
Program	L'objectif principal est de délivrer la Solution, via l'ART (Agile release Train), sur la base de PI (Program Increment). Les principaux artefacts sont : • les Features/Enablers dans le Backlog du Train • L'architecture Technique de la Solution • System Demo et la mise à disposition aux End users Ce train est organisée autour des profils RTE, PM et SA en lien avec les businessowners de la solution.	Acculturation de l'équipe du Train à la démarche de testing. Décliner une stratégie de test dans le contexte du train en adéquation avec la stratégie de test de chaque Team. Mise en place d'un Environnement de test de la solution au niveau du train.
Teams	L'équipe agile a pour objectif de définir, concevoir, tester et délivrer les composants de la solution sur la base du Backlog de User Story propre à l'équipe. Les principaux artefacts sont : • les User Story en relation avec les Features de la solution • La conception des composants de chaque User Story • Le test des User Story • La livraison au Train. En application des bonnes pratiques Agile/DevOps.	Différentes maturités de développement agile au niveau des équipes. Processus de test limité et mené de façon séquentielle (V) sans automatisation. Concept de Built In Quallity très disparate suivant les équipes, voire inexistant. Evaluation du Niveau de Qualité des livrables impossible.
System team (ST)	La System Team est une Agile Team transverse à l'ART, ayant les responsabilités suivantes : • Intégration de la solution • Test de la solution • Solution démo • Releasing de la solution.	Constituer une organisation de la System Team pour répondre aux objectifs et responsabilités de la ST.

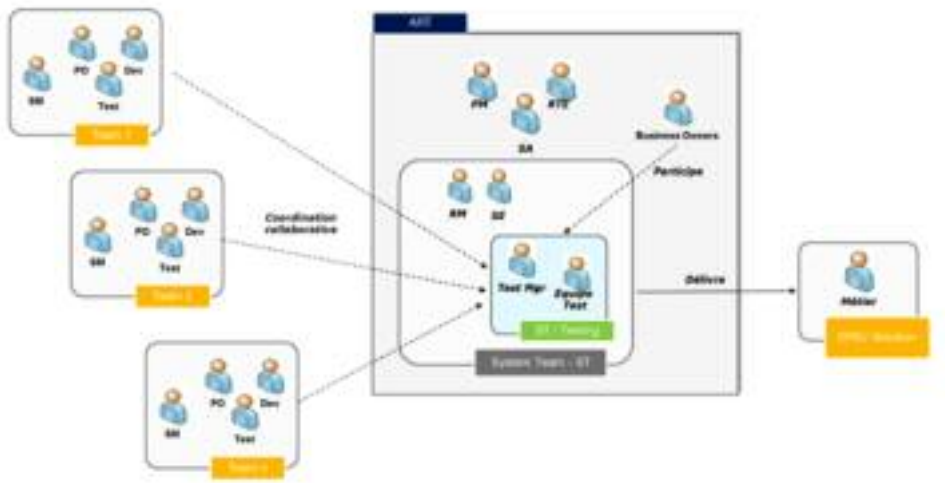
d. Approche Testing adoptée.

- Organisation équipe System Team.

Il faut remarquer que dans le Framework SAFe, il n'est fait mention d'aucune organisation (équipe, spécialiste, coach, …) dédiée aux aspects de test, hormis le chapitre « Agile testing » et le fameux quadrant des tests.

Le premier point d'intervention concerne l'intégration de la démarche de test dans l'équipe ST (System team). Un des principaux objectifs de la ST concerne l'intégration et les tests E2E (End To End) de la solution, par la mise en place d'une équipe de test train dont les objectifs sont de :

– Définir et dérouler la stratégie de test appliquée au Train pour les tests Intégration et E2E appelée : Qualification Test Transverse du Train (QTT).
– Appliquer les bonnes pratiques de test ISTQB pour s'assurer de la qualité de chaque PI (Program Increment) livré au métier.
– Identifier les environnements de Test nécessaires à cette QTT.
– Identifier et piloter la gestion des anomalies dans un contexte solution multi-applicative.

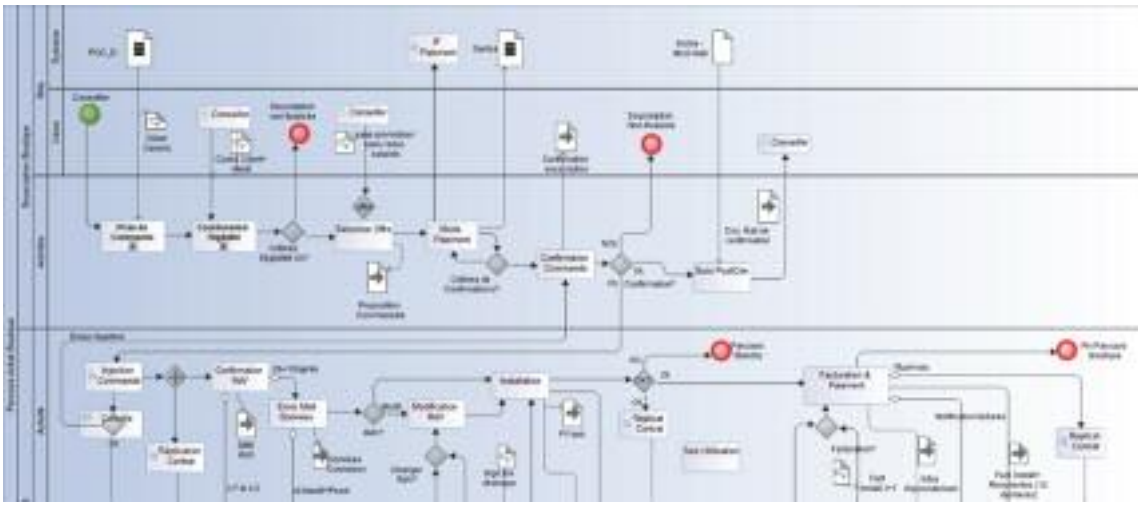


- Modélisation.

La première activité nécessaire dans la stratégie de test QTT est de formaliser les EPIC/Features de la solution, sous une représentation qui soit compréhensible aussi bien par le métier que par l'équipe de test. La modélisation de processus (BPM) et plus particulièrement le MBT (Model based testing) permet de trouver ce compromis de compréhension du système. En effet, il ne faut pas oublier qu'en ce qui concerne le Train, nous sommes dans ce que l'on appelle un Système de Système au niveau de l'écosystème Orange.

– Première Action : modélisation BPM.

Pour pallier ce manque de vision globale de la Solution, la première action porte sur une modélisation commune de type BPM permettant de structurer et représenter l'enchaînement des activités de la solution.



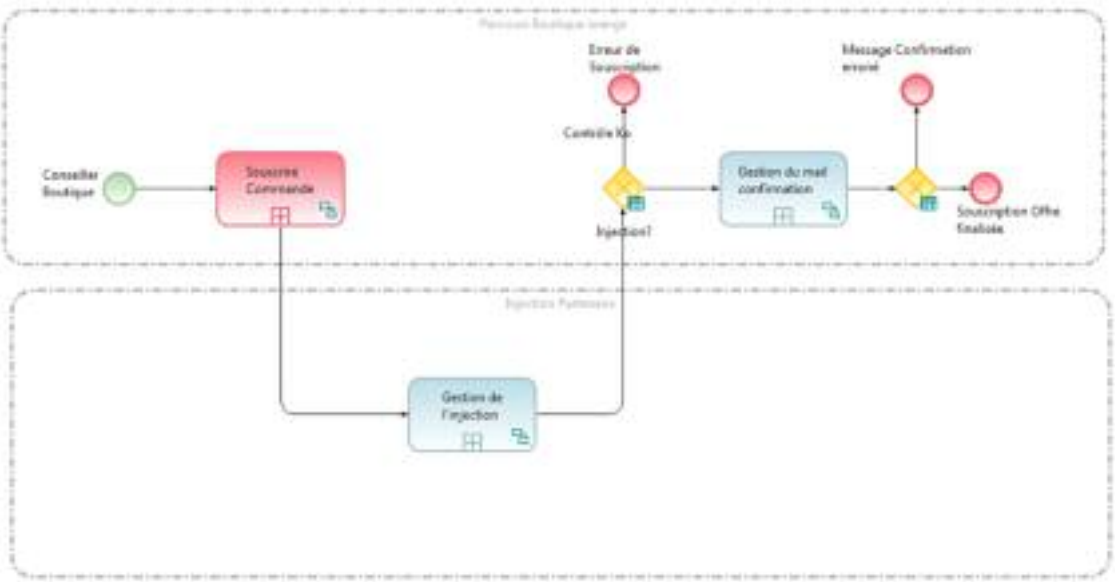
Le principal avantage de cette modélisation est la compréhension par chaque organisation (métier, test, dev, user, …) de ce que le système « Fait, Quand et avec Quoi ».

Cette modélisation peut ensuite être utilisée pour chaque Activité, comme support dans chaque PI des évolutions de Features et des User Story qui sont associées.

– Deuxième Action : modélisation MBT.

Sur la base du BPM, l'équipe de test transverse de la system team identifie les différents scénarios de test, en déclinant le processus sous la forme d'un MBT : Modeling Based Testing. On retrouve chaque activité du BPM avec une description cette fois-ci orientée testing, avec la déclinaison pour chacune des différents cas de figure de traitement.

Modèle de haut niveau : Souscription d'une Offre.



Activité « Souscrire Commande » détaillée avec ses différents cas de fonctionnement.

La génération des scénarios de test de bout en bout (E2E) se fait suivant les différentes tables de décisions identifiées dans chacune des sous-activités.

Cette transformation du BPM en MBT permet de se focaliser de façon rigoureuse et précise sur les aspects de traçabilité et de couverture fonctionnelle, car chaque table de décision est faite sur la base des critères d'acceptance identifiés pour chaque Feature, lors de chaque scénario. (Respect des bonnes pratiques de l'ISTQB en ce qui concerne les principes de traçabilité bi-directionnelle entre les spécifications et les tests).

- Acculturation Agile Testing.

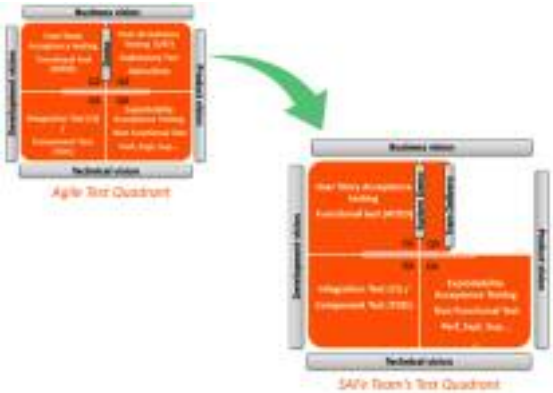
Le cadre SAFe (ou Agile Coordonné chez Orange) nécessite d'avoir une vision de test commune mais néanmoins adaptée aux objectifs de chacune des équipes : Dev team, System Team et Business team. Les problèmes de maturité et d'acculturaton testing des différentes équipes impose de redéfinir et de formaliser la déclinaison du Quadrant Agile Testing pour chaque équipe.

– Agile Testing niveau Dev Team.

Au niveau Dev Team, la mise en œuvre des tests est conforme au quadrant Agile pour les quadrants Q1 et Q2, qui correspondent à la vision du développement des User Story en lien avec les Features. Au niveau du quadrant Q4, les tests sont focalisés sur les aspects d'exigences

non fonctionnelles liées à la sécurité, la performance au niveau des User Story ainsi que la prise en compte des besoins de supervision pour les OPS (ce qui est nécessaire dans un contexte DevOps). Le quadrant Q3, quant à lui, est limité à la seule participation à la system démo sur le contour des User Story validées pour l'itération.

Pour Q1 et Q2, la maturité disparate dans l'application des bonnes pratiques de test et surtout d'automatisation, nécessite au niveau de la System Team de positionner un Coach Testing (similaire au Coaching DevOps mais focalisé sur les activités de test et d'automatisation) auprès des équipes pour atteindre les niveaux de qualité et d'efficacité.

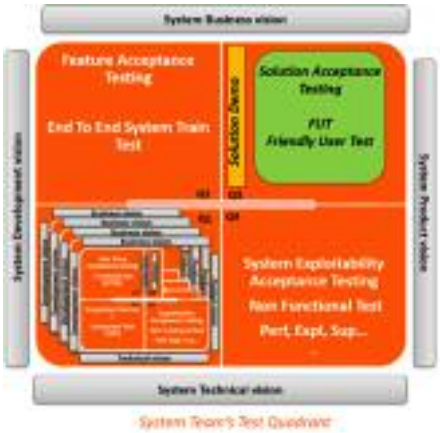


– Agile Testing dans la System Team.

Comme identifié dans la partie Organisation, c'est réellement lors de cette étape que le plus gros effort d'acculturation de test est mené, pour atteindre les objectifs pour ce qui concerne le test du Train.

La vision des tests du Train est orientée système par les besoins :

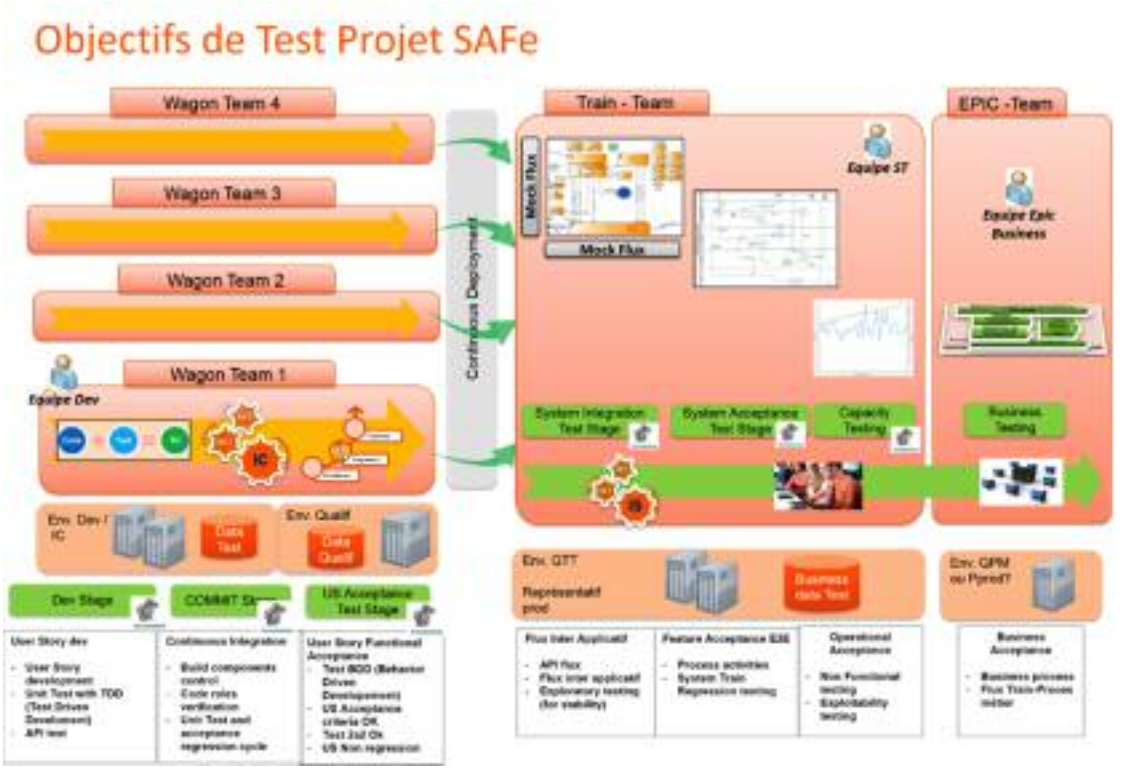
- niveau Q1 :
  - d'intégrer l'ensemble des User Story issues des Teams nécessitant un processus de delivery maîtrisé.
  - de vérifier la disponibilité de la plateforme de test par des tests Exploratoires.
- niveau Q2 :
  - de réaliser le test des Features sur la base des critères d'acceptance identifiés en collaboration entre les PM, Business owners et Test Manager.
  - d'effectuer les tests de bout en bout (E2E) au niveau du Train de la solution.
- Niveau Q4 :
  - de piloter les tests d'exploitabilité en relation avec les équipes dédiées (métrologie, supervision, ...).



• Stratégie globale de test.

– Objectifs de test.

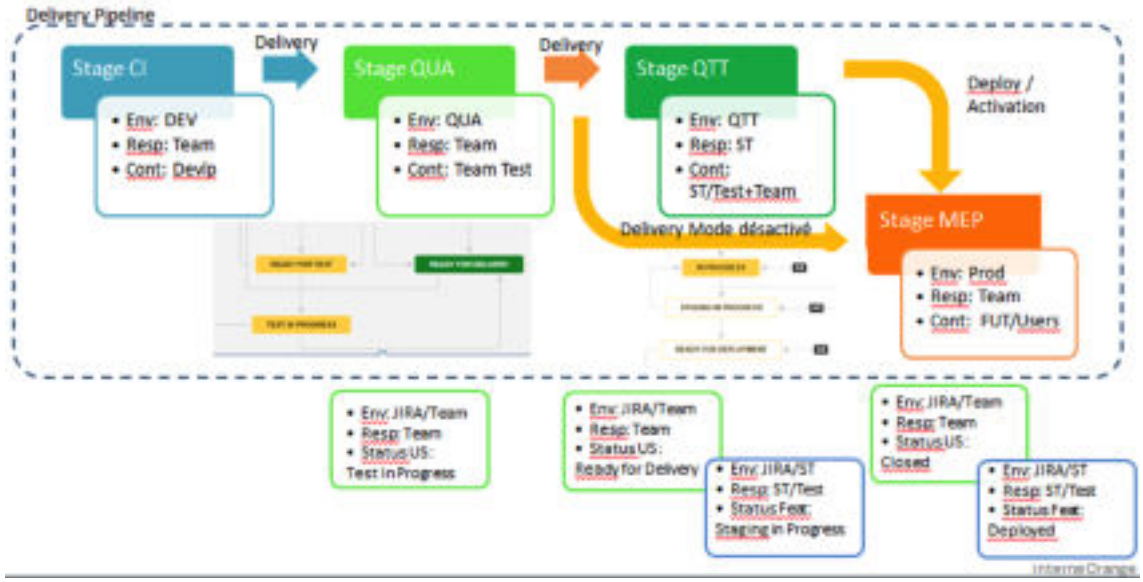
Sur la base de l'organisation Dev Teams/System Team et des objectifs Agile Testing de chacun, la stratégie globale de test pour le Train est définie de la façon suivante :



Un environnement spécifique QTT (Qualification Test Transverse) permet de disposer d'une plateforme de Test de Qualification Transverse pour chaque Team. Celle-ci est mise à jour via le déploiement régulier entre les Teams et le Train.

– Organisation Delivery et plateforme de test.

La mise en place d'une plateforme de Delivery automatisée est nécessaire afin de gagner en efficacité et en qualité à chaque étape.
Cette plateforme est généralisée pour chaque Team sur la base d'un Pipeline (Jenkins, GitLab) et de Test automatisés au niveau QUA (Qualification Applicative) et QYY (Qualification Test Train).



5- Outils et Agilités.

La mise en œuvre d'outils est une activité transverse nécessaire et doit couvrir aussi bien les équipes Dev Teams, ART que Solution métier. L'utilisation de JIRA/Confluence apporte le support de formalisation et de communication pour la description de la Solution, par l'utilisation d'un Backlog partagé par tous, implémentant la structure : Epic/Feature/User Story. L'amélioration du dialogue entre le métier et le Dev passe par l'utilisation d'outils BPM permettant de formaliser les parcours utilisateur et ainsi fournir une base pour la pratique du MBT (Modeling based testing) via des outils tels que YEST qui offre la faculté de décomplexifier les processus utilisateurs en activités testables via des scénarios de test.

Pour le Continuous Testing (CT) et le Continuous Delivery (CD), il est impératif de mettre des solutions homogènes sur toutes les équipes afin de gagner en efficacité et traçabilité, par exemple : XRay pour la gestion des tests, RobotFramework/Selenium pour l'automatisation des tests d'acceptance et Jenkins/Pipeline pour le déploiement.

6- Conclusion.

« L'Agile Coordonné » s'appuie sur un écosystème d'outils (backlog, développement, gestion de conf, intégration, tests, déploiement, defect, modélisation, reporting, management, communication, ...).

L'automatisation et l'interfaçage sont essentiels à un fonctionnement optimal. Nous ne nous sommes concentrés dans ce REX que sur le Test, les autres aspects ont été masqués volontairement.
L'approche doit se faire sur l'écosystème afin de prendre en compte les besoins Projets et Trains dans leur ensemble et ainsi proposer des solutions cohérentes, fluides et interconnectées, permettant aux utilisateurs de rester concentrés sur l'objectif premier :

« Délivrer leurs produits dans les temps, avec la meilleure qualité ».

III.4- Retour d'expérience : tous testeurs chez AXA !

Tester au plus tôt pour livrer en continu, la Guilde Test au cœur de DevOps.

Par Emilie-Anne Guerch, Lionel Brat, Pascal Cugnet, David Esteves Da Fonte, Vincent Dugarin, Emmanuel Lehmann et Bruno Pedotti

1- Contexte de la transformation AXA : pourquoi se transformer ?

Pour faire face à des enjeux business grandissants dans le secteur de l'assurance, AXA France lance en octobre 2017 un plan stratégique centré sur ses clients : Easy AXA, « Gagnons en simplicité pour nos clients ».
A chaque instant, à chaque opération, à chaque projet, chaque collaborateur doit se demander si cela apporte de la valeur et de la simplicité à nos clients.

Forte de ses équipes agiles, de son partenariat avec le Métier et de son savoir-faire technologique, la Direction du Système d'Information (DSI) est un pilier de ce plan. Elle doit apporter une expérience simple et fiable à l'ensemble des utilisateurs du système d'information.

La transformation agile de son organisation, le déploiement progressif de pratiques DevOps, l'accompagnement des collaborateurs sur ce changement culturel doit permettre à la DSI de relever ce challenge.

2- L'organisation agile à l'échelle de la DSI AXA France.

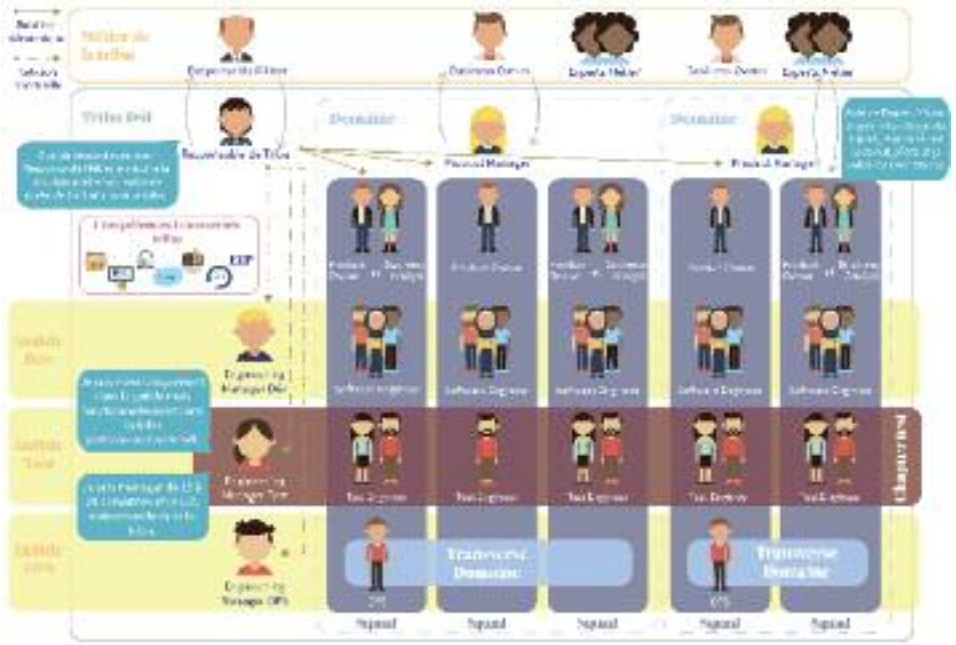
L'histoire de cette transformation commence dès 2011 avec la création du Webcenter de Lille, où sont développées les applications web métiers et grand public d'AXA France. Fin 2014, 50% des équipes de la DSI travaillent en Scrum.

En 2015, la DSI construit et expérimente pendant un an, auprès de 400 collaborateurs, un modèle inspiré de Spotify et du framework SAFe. Ce modèle est validé, déployé et généralisé au reste de la DSI deux ans après, soit auprès de 1200 personnes. Il s'organise en mode Feature Team, avec un découpage standardisé basé sur la mise en place de tribus / domaines / squads.

Des équipes agiles stables, autonomes, multi-compétences et polyvalentes sont mises en place avec des Product Owners, Software Engineers et Test Engineers.

Grâce à une plus grande collaboration entre les équipes IT et les équipes Métiers, celles-ci pensent maintenant « Produit » plutôt que « Projet » et se focalisent sur la livraison de valeur visible pour le client, plus vite et plus régulièrement par petites itérations.

Description d'une Tribu :



En parallèle, la Guilde Test est lancée. Direction transverse à l'ensemble des tribus, elle 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 AXA France.

La Guilde Test s'organise autour d'équipes de test qui sont managées par des engineering managers test. Chaque équipe est composée de Test Lead et de Test Engineers intégrés dans les tribus.

L'Engineering Manager Test est garant de la qualité des livrables et des engagements de la guilde. Il porte la responsabilité du staffing de son équipe, déploie les bonnes pratiques et mesure l'efficacité des plans de progrès mis en place.

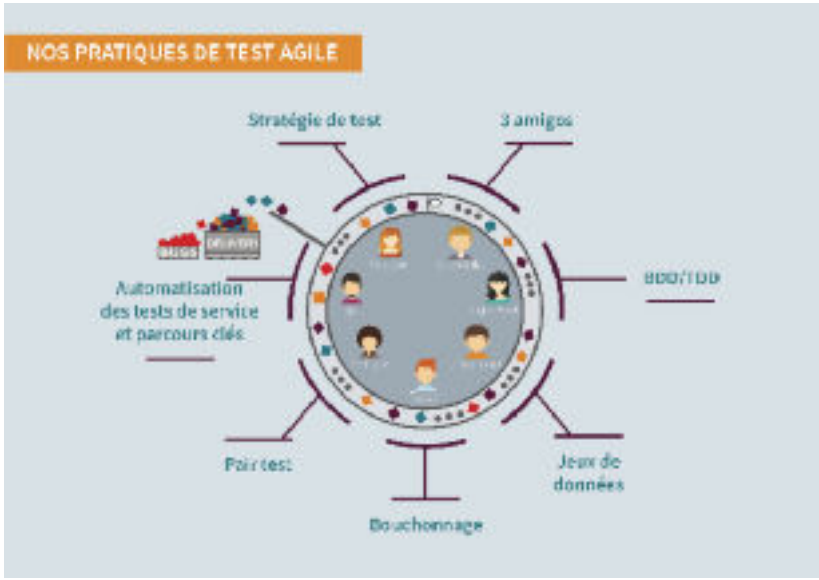
Le Test Engineer contribue à la définition de la stratégie de test de son périmètre. Il assure la conception, l'automatisation et l'exécution des tests en mettant en œuvre les bonnes pratiques de test. Il réalise et communique de manière régulière le reporting d'avancement de l'activité.

En plus des activités du Test Engineer, le Test Lead anime un chapter test et assure la montée en compétence des Tests Engineers. Il est le référent de son périmètre pour le déploiement des bonnes pratiques et la montée en compétences des différents acteurs.

Une équipe Ingénierie complète l'organisation en prenant en charge l'animation et la montée en compétence des collaborateurs. Elle définit et maintient le cadre de référence test de la guilde (méthodes, outils, conseil expertise).

3- Les pratiques de test.

Les pratiques de test agile que nous mettons en premier plan :



La stratégie de test se décline à deux niveaux : tribu et produit.
La stratégie de test tribu décrit le contexte de la tribu, son organisation, les moyens mis en œuvre, l'approche des tests à mettre en place pour la validation du plan d'actions de la tribu. Elle est validée avec le responsable de tribu.
La stratégie de test produit décline la stratégie tribu pour chaque release d'un produit. Elle aborde principalement le périmètre, la couverture des tests, les éléments d'organisation spécifiques et le planning de réalisation. Elle est partagée et validée avec l'équipe et le Métier.

Des ateliers « **3 amigos** » permettent d'associer systématiquement le trio Product Owner, Software Engineer et Test Engineer dès l'apparition d'un nouveau besoin pour renforcer sa compréhension très en amont d'un sujet. Pendant ces ateliers, Test Engineers et Software Engineers questionnent le Product Owner pour comprendre le besoin, extraire les règles Métier et définir ensemble les exemples illustrant ces règles. Ces exemples deviendront les critères d'acceptations de la User Story. Ces exemples seront détaillés par les développeurs en utilisant le langage Gherkin pour développer la solution selon la pratique de développement BDD et automatiser autant que de possible.
Les **parcours clés** sont identifiés et définis avec le Product Owner. Ils représentent les parcours utilisateurs de bout en bout les plus critiques. Les tests de ces parcours sont automatisés et sont joués à chaque déploiement de l'application.

Ces tests automatisés sont intégrés dans la PIC (Plateforme d'Intégration Continue) du produit en cours de développement. L'automatisation est faite par un Test Engineer automaticien qui utilise le framework de développement utilisé par les Software Engineers pour faciliter les échanges, l'entraide et le partage des bonnes pratiques (revue de code, analyse qualité, …) au sein de la squad/tribu.

Lorsque ces tests sont exécutés dans une phase dite d'intégration, le test engineer automaticien s'appuie généralement sur des services virtuels (bouchons ou mock services) pour gérer ses jeux de données.

Le Test Engineer organise des séances de **pair test** avec un Software Engineer dans un environnement proche du développement (poste du Software Engineer ou de DEMO). Le Test Engineer accompagne le Software Engineer pour jouer manuellement les cas de test des Story associés aux US « fraîchement » développées, complétées par des tests exploratoires basés sur l'expérience. Lors du pair test, les anomalies identifiées sont corrigées en séance et re-validées. Si ce n'est pas possible, la User Story est bloquée, l'anomalie détectée fera l'objet d'un correctif à valider lors d'une prochaine séance de pair test.

Enfin, les **tests de sécurité** sont effectués à deux niveaux : pendant le développement en s'appuyant sur des outils d'analyse de code et en phase de préproduction par des experts sécurité (tests de pénétration).
Les **tests de performance** sont également effectués à deux niveaux : par les Software Engineers lors des phases amont du développement et par des experts en phase de préproduction. Des tests de montée en charge d'endurance et aux limites sont généralement effectués.

4- Déploiement et mise en œuvre des pratiques.

Depuis 2011, les collaborateurs ont dû modifier leurs pratiques (agiles et ingénierie), ils ont vu leur organisation et leurs rôles évoluer. Si ces changements ont modifié leur quotidien, ils ont surtout dû acquérir un nouveau mindset.
En effet, une transformation agile est avant tout une transformation culturelle qui s'accompagne d'une forte conduite du changement. Cette transformation est basée sur les valeurs et principes agiles, en privilégiant la collaboration directe et l'amélioration continue. Afin d'être plus engagés au quotidien, les collaborateurs doivent gagner en autonomie, en responsabilisation et saisir le lien entre leurs activités et le bénéfice utilisateur apporté.

Pour déployer cette transformation, AXA a créé un Agile Center d'une quinzaine de coachs agiles qui accompagnent tribus et guildes sur les changements culturels et sur l'évolution du processus de delivery. Les coachs interviennent opérationnellement dans le delivery quotidien des squads et supportent les Responsables de Tribus et les Product Managers dans le cadrage des besoins avec le Métier et la définition de la roadmap produit. Ils dispensent aussi des formations agiles à l'ensemble de la DSI.

A l'instar des autres métiers d'ingénierie de la DSI, la Guilde Test met en place pour ses collaborateurs des formations spécialisées sur son métier, afin de renforcer les connaissances et les expertises sur les méthodologies (formations ISTQB, Agile) et sur les outils (webcast, dojos et accompagnements sur les automates ou la mise en œuvre des services virtuels). Ces formations individualisées se complètent par des communautés de pratiques sur la tribu d'intervention (Chapter) ou sur l'expertise (automatisation et méthodologie de test).

Pour garantir un certain niveau de qualité et l'homogénéité des activités de test, un cadre de référence a été établi, basé sur des bonnes pratiques à mettre en œuvre.

Pour prioriser et coordonner les actions à mettre en œuvre, chaque Engineering Manager Test évalue le niveau de maturité de son périmètre et co-construit un plan d'action avec le responsable de la tribu.
Ce niveau de maturité dépend également du contexte (niveau de maturité agile de la tribu et des autres guildes).

5- Difficultés VS gains.

La mise en œuvre des pratiques a été progressive pour permettre l'adaptation et la montée en compétence des équipes de développement et de test.

La mise en place des pratiques agiles (3 amigos et Pair-Test principalement) a permis de réduire de moitié le nombre de bugs en recette et d'accentuer la collaboration au sein des équipes : « On ne démarre pas un projet sans un Test Engineer intégré », affirment désormais les Software Engineers.

Le déploiement du TDD couplé à la mise en œuvre des parcours orientés clients a renforcé notre approche de « shift left testing ».

Enfin, l'intégration du BDD et l'automatisation des tests d'acceptation par les développeurs a limité les phases de recette et a permis de bâtir un harnais de test des scénarios exécutés lors de chaque Build. La recette se concentre désormais sur les tests de bout en bout et l'exécution automatisée des tests des parcours clés.

La mise en place du modèle crée ipso facto des silos contre lesquels il faut lutter en intensifiant les échanges au sein des guildes. Nous avons donc mis en place des communautés de pratiques sur l'automatisation et la méthodologie et avons nommé des référents pour faciliter l'appropriation et l'homogénéité des pratiques. Il faut également s'assurer via des mesures régulières de la bonne mise en œuvre des pratiques. Un modèle de maturité permettant d'en suivre l'évolution au sein des squads et des tribus a été déployé. Ces actions nécessitent un fort investissement en formation et coaching des équipes de développement et de test.

Les équipes sont focalisées sur les engagements opérationnels et il est toujours difficile de les mobiliser sur les travaux d'amélioration continue.

Un des bénéfices importants a été la valorisation du métier du test. La DSI AXA France a engagé une internalisation des compétences et un plan de recrutement ambitieux a été lancé pour faire face à la montée en charge de l'activité du test.

En synthèse et à titre d'exemple avec le site internet public axa.fr :

- Nombre de bugs par US réduit de 80% en 6 mois (passage de 1 à 0,2 bugs/US).
- Time to Market inférieur à 2 mois.
- Fréquence de livraison en production hebdomadaire.
- Réduction du nombre d'incidents majeurs en production.

6- Et demain ?

Le déploiement du modèle Feature Team met en exergue la nécessité de renforcer la polyvalence des acteurs d'une squad. En effet, l'équipe étant de taille réduite, il est primordial que le Test Engineer puisse être remplacé par un software engineer pour l'automatisation, que le Test Engineer puisse remplacer un Business Analyst ou un Product Owner pendant ses congés et réciproquement. Cela nécessite un accompagnement des collaborateurs de la squad sur les domaines de compétence ou les changements de rôle sont réalistes : Test Engineer Automaticien / Software Engineer ; Test Engineer/ Business Analyst ou Product Owner ; Test Engineer / Scrum Master.

Nous avons initié le « shitf left » et renforcé les tests en amont du cycle de développement. Il reste encore beaucoup de travail pour optimiser la pyramide des tests et limiter la durée des phases de recette et de préproduction à leur strict minimum. Cela passe par une généralisation de l'Intégration Continue et de DevOps ; de l'accroissement de l'automatisation des tests pour réduire au maximum la nécessité du test manuel qui force une rupture dans le process de livraison. Bien sûr, cet effort est à pondérer au regard de la nécessité Métier de livrer rapidement et souvent : ce besoin de réactivité n'est évidemment pas le même entre notre espace client et une application back-end de gestion, par exemple.

Le monde de l'assurance s'intéresse également de plus en plus aux objets connectés dans le domaine de la santé et aux chatbots dans les centres d'appels. Des stratégies de tests spécifiques seront à mettre en œuvre pour faire face à ces technologies.
L'Intelligence Artificielle devrait également nous permettre d'optimiser nos plans de test et de maximiser la couverture de test à moindre coût.

Et bien sûr, l'évolution technologique rapide sur les métiers du test (outils et pratiques) doit être suivie pour que nous adaptions nos outils et nos pratiques. Une veille technique permanente est indispensable.

III.5- Processus de test fondé sur la conception visuelle des tests : Retour d'expérience sur un projet en Agile.

Par Elodie Bernard

Introduction

Dans ce chapitre, nous décrivons un retour d'expérience sur la conception visuelle des tests en contexte agile. Nous commençons par décrire les motivations qui ont conduit à utiliser une approche de type ATDD – Acceptance Test Driven Developement – à une description visuelle des cas de test, puis nous détaillons la mise en œuvre effective sur notre projet et la chaîne outillée utilisée.

Nos résultats se basent sur six mois de projet, avec une mise en place de l'approche dès le premier sprint. Nous détaillons la façon dont la mise en place d'un outil de conception visuelle des tests nous a permis d'améliorer notre processus de test et d'appliquer au quotidien une approche d'ATDD agile.

1- Les problématiques autour de l'Agilité.

À la suite de nos expériences sur des projets en Agilité, nous avons identifié plusieurs difficultés dans la mise en place et le maintien d'un cycle de test stable au fil des itérations. Au travers de ces expériences et des enseignements que nous pouvons en tirer, nous proposons une démarche ayant pour objectif d'améliorer la gestion du cycle des tests en Agile sur différents aspects.
Notre démarche s'attache à l'amélioration de la gestion des tests selon trois axes : l'analyse des tests et la communication, la conception et la génération des tests et enfin, leur automatisation.

Analyse et communication

Conception et génération

Automatisation

Tout d'abord, nous avons cherché à améliorer la phase d'analyse ou nous rencontrions des difficultés à identifier le périmètre à tester, notamment au niveau des connexions entre les diverses fonctionnalités.
Cette phase est nécessaire au bon déroulement du sprint : sans cette analyse, les équipes ne sont pas en mesure d'identifier les dépendances pouvant exister entre les User Story (US) et donc de prioriser les développements et la conception des tests pour permettre l'exécution des

tests au plus tôt. Cette phase était difficile à réaliser, car les outils en notre possession ne permettaient pas d'identifier rapidement les fonctionnalités liées entre elles sans devoir consulter l'ensemble des US et nous n'étions pas en mesure de représenter ces interactions de manière visuelle. Auparavant, nous le faisions à travers un tableau, mais ceci était long et ne nous permettait pas d'avoir la vue d'ensemble sur le périmètre en cours.

Ce manque de visibilité entraînait inévitablement des difficultés de communication entre les différentes parties prenantes du projet (i.e. développeurs, testeurs et client) et nous avons cherché une solution afin de faciliter la communication et répondre aux attentes de chacun. Les développeurs ne comprenaient pas les besoins des testeurs en termes de priorisation des développements, par manque de visibilité sur la trame Métier et sur les dépendances des US entre elles.

La connaissance de la trame Métier est importante dans les contextes agiles car à chaque fin de sprint, un cérémonial agile de revue de sprint a lieu. Il intègre une démonstration des fonctionnalités développées au cours du sprint. Cet exercice, bien que mis en place, était souvent difficile à réaliser, incomplet et décousu par manque de positionnement des User Story dans la vision globale du système et de connaissance Métier des développeurs et des testeurs.

L'utilisation d'une conception visuelle des tests et la définition des cas de test au plus tôt dans une démarche ATDD visuel ont fortement contribué à clarifier et à contextualiser le périmètre à chaque sprint. Cela permet ainsi une meilleure visibilité du client sur les travaux effectués à chaque sprint.

Les caractères incrémentaux et itératifs des méthodes agiles induisent le fait que les fonctionnalités évoluent ou s'enrichissent au fur et à mesure des sprints : il est donc nécessaire d'identifier les tests qui sont déjà présents et qui seront impactés par les évolutions. Ce travail est fastidieux et complexe dans les référentiels de test traditionnel type Squash™ ou Hp ALM. En effet, il est nécessaire de consulter chaque cas de test, d'identifier s'il doit faire l'objet d'une évolution ou non.

Vient ensuite la phase de conception, où les méthodes agiles préconisent d'accueillir favorablement toute demande de changement : nous avons toutefois pu constater que l'évolution des règles Métier ne se faisait pas sans heurt. Ainsi, lors de la modification d'une règle Métier déjà implémentée et testée, il fallait être en mesure d'identifier les changements opérés dans les cas de test et les mettre à jour. Comme évoqué précédemment, c'est dans la phase d'analyse que nous réalisions ce travail d'identification et que nous rencontrions des difficultés. Il restait ensuite la mise à jour des cas de test qui demandait un effort manuel important pour éditer chaque cas de test impacté.

Le dernier point, et non le moindre, est celui de l'automatisation. Les retours d'expériences que nous avons de mise en place de l'automatisation ont pour bilan d'être complexes et/ou coûteuses.

Ceci s'explique, entre autres, par les changements et les évolutions des règles Métier qui nécessitent la retouche des scripts d'automatisation de nombreux tests, en plus de la production de nouveaux. De plus, le périmètre des tests automatisés n'est pas toujours connu et les

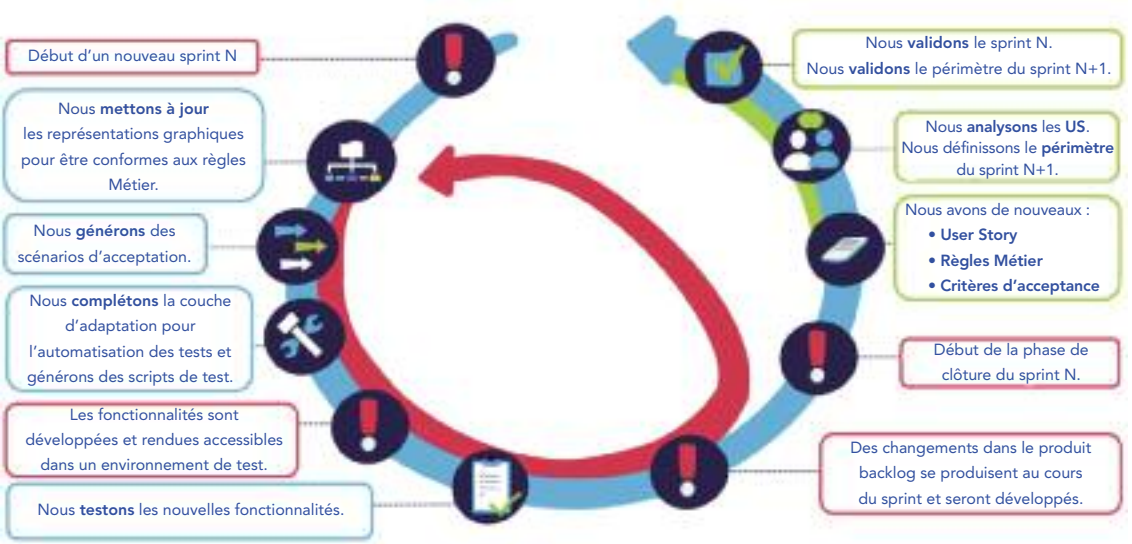
fonctionnalités testées dans leur détail non plus. Ceci s'explique par le fait que les automaticiens de test réalisaient des scripts de test totalement détachés des cas de test existant avec peu de documentation. Avec ces difficultés d'automatisation, la réduction des efforts des testeurs manuels n'était pas palpable, car ceux-ci n'étaient pas en mesure de déterminer la couverture fonctionnelle obtenue par l'automatisation.

2- Une approche fondée sur la conception visuelle des tests.

Afin de décrire l'approche mise en place sur le projet, nous présentons le contexte dans lequel nous travaillons et les éléments mis à notre disposition afin de produire les cas de test.
Le développement du projet adopte un cycle projet agile avec des sprints de quatre semaines. Avant chaque début de sprint, le périmètre des fonctionnalités à implémenter est arrêté et explicité à travers un ensemble de « User Story » (US). Ces US sont regroupées par fonctions et sous-fonctions et contiennent un ensemble d'exigences. La gestion de projet et du backlog est assurée avec *Jira*, dans lequel les US sont enregistrées en tant que *Récit* et les exigences en tant qu'*Exigence*.
Ces deux éléments sont la base pour la conception de nos cas de test qui sont générés avec l'outil *Yest* de conception visuelle des tests. Après la phase de conception, nous exportons ces cas de test dans *Squash*^{TM}, notre outil de gestion des résultats d'exécution de tests.

3- Les activités de test revisitées.

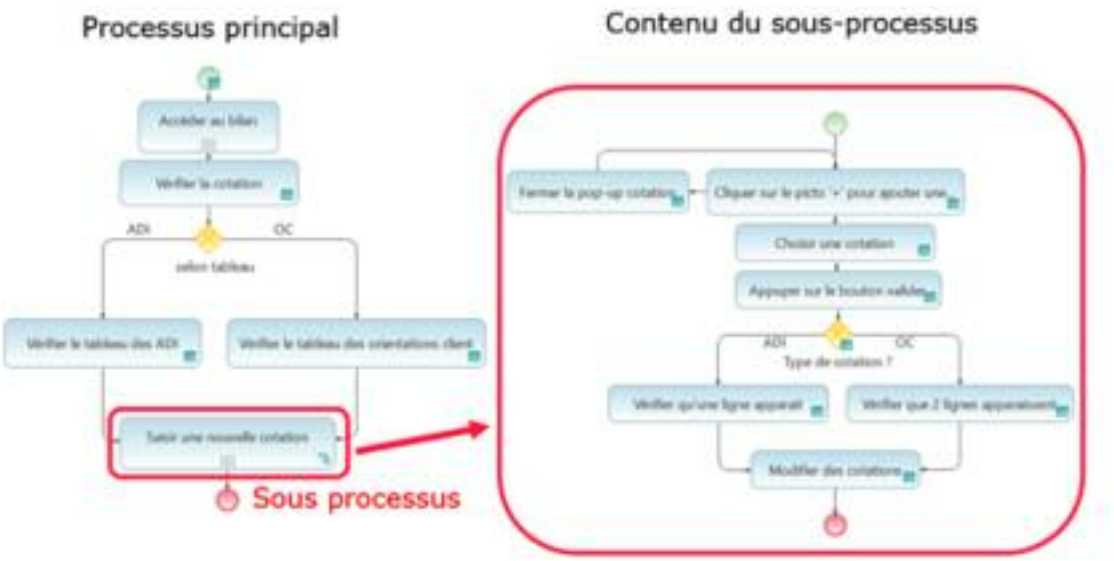
Notre approche, fondée sur la conception visuelle des tests et l'ATDD (c'est-à-dire la conception au plus tôt des tests d'acceptation des User Story), nous a conduits à revisiter en profondeur le processus de test.
Le schéma qui suit représente notre démarche en amont et à travers le sprint.



Avant le sprint, nous sommes en mesure d'analyser les US et de définir le périmètre des fonctionnalités à tester. Dans le cadre du sprint 1, cette phase d'analyse se fait durant un sprint 0, ayant pour objectif de définir le périmètre pour le sprint 1. Dans les sprints courants du projet, ce travail est fait en fin de sprint N pour le sprint N+1. L'analyse est réalisée à travers une représentation visuelle constituée des US et des interactions existantes entre elles. Ensuite vient le début du sprint, où nous mettons à jour nos représentations graphiques des tests et générons de nouveaux cas de test ou mettons à jour les tests. Via le référentiel de test existant, nous complétons une couche d'adaptation nous permettant de générer des scripts de tests. Tests manuels et scripts de tests automatisés sont ensuite exécutés suivant les fonctionnalités disponibles sur l'environnement de test. Si des changements ont lieu, nous mettons à jour les cas de test afin de répondre aux nouvelles règles Métier et recommençons le cycle précédemment énoncé jusqu'à la validation et fin de sprint. Les sections qui suivent décrivent comment, pour chaque étape, la conception visuelle des tests améliore et facilite le processus des tests en Agile.

4- Conception visuelle des tests : de quoi parle-t-on ?

Dans une approche classique de conception des tests, l'analyste de test définit graphiquement les parcours applicatifs à tester ainsi que les données logiques et les règles Métier au travers de tables représentant les cas pertinents pour le test. Cela permet de visualiser la couverture de test des différentes User Story.



La capture ci-dessus présente un exemple de description de parcours applicatif à tester : les tâches (dans les encadrés bleus) décrivent les activités de test, synthétisant ainsi de façon visuelle les différents scénarios. Cette présentation est structurée (avec des sous-parcours) de façon à faciliter la conception des tests et leur communication aux membres de l'équipe. Ces représentations des workflow Métier permettent de produire l'ensemble des cas de test requis qui sont ensuite exportés dans des outils d'exécutions de test (suivant qu'il s'agit de tests manuels ou de tests automatisés).

5- Analyse et conception visuelle des tests dans l'Agilité.

Dans notre approche, nous privilégions l'analyse des US en amont des sprints pour faciliter le travail de conception. Une fois cette analyse opérée, nous adaptons les tests existants et créons de nouveaux tests en fonction des nouvelles User Story implémentées dans le Sprint.

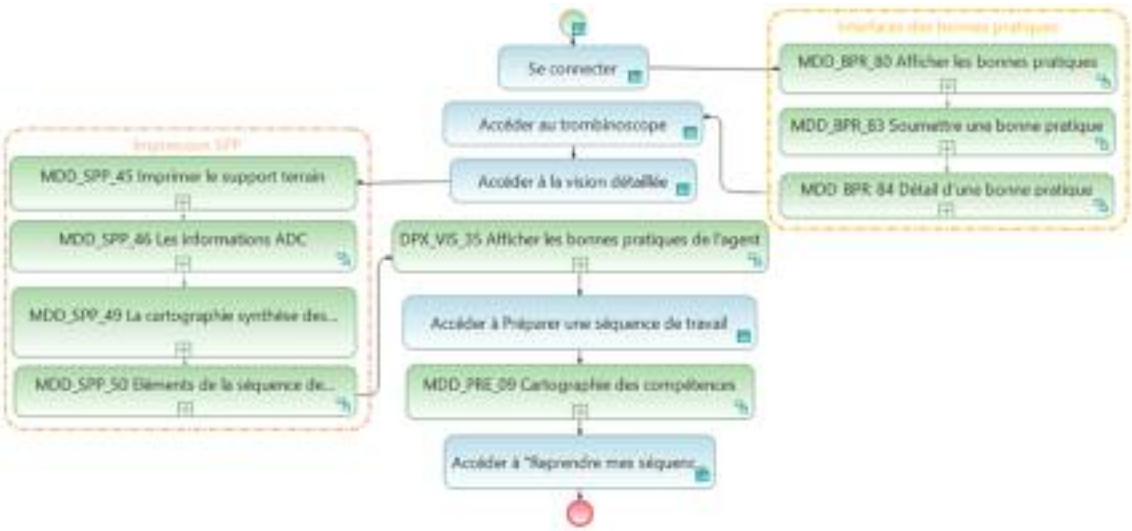
a. La phase d'analyse des US.

Cette phase est primordiale dans la conception des cas de test, elle consiste à étudier le périmètre du sprint à venir, repérer les dépendances entre US et prioriser les développements et les tests. Nous avons constaté que cette phase était souvent coûteuse en temps (plus de 50% du temps alloué à l'analyse des US et du périmètre durant la conception des cas de test), ceci prolongeait donc la phase de conception des tests et reportait l'exécution des tests au plus tôt.

Le référentiel des US en tant que tel, même classé par fonctions et sous-fonctions, ne permet pas sans analyse de déterminer les liens entre les différentes fonctionnalités. À travers la conception visuelle des tests, nous avons trouvé une solution pour faciliter cette analyse : nous avons noté un gain de 20% sur le temps de conception. La section qui suit décrit l'approche que nous avons mise en place.

Une fois le périmètre à développer connu (liste des US), nous définissons graphiquement les parcours applicatifs à tester qui couvrent l'ensemble des US permettant de :

– Faciliter la communication entre les parties prenantes du projet.
– Identifier les liens entre les fonctionnalités et prioriser les développements et tests.
– Définir une trame pour le scénario de démonstration.



Dans cet extrait de parcours applicatif couvrant plusieurs US, on fait apparaître en bleu les tâches purement dédiées au parcours Métier (« se connecter », « accéder au trombinoscope », etc.) et en vert les US qui s'intègrent dans le processus. Ainsi, nous pouvons communiquer facilement autour de l'intégration des US dans le processus métier existant. Pour chaque sprint, nous

adaptons les parcours applicatifs pour mettre en avant les évolutions du sprint et nous faisons apparaître au besoin les US précédemment développées et impactées par les changements.

Afin d'assurer une traçabilité entre les US apparaissant dans les parcours applicatifs et Jira, nous nommons les tâches dans la conception visuelle des tests comme dans Jira et nous créons des liens dans les tables associées, comme cela est présenté ci-dessous :

	US	Lien Jira
1	MDD_PRE_09 Cartographie des compétences	https://agiliti.dsiv.fr:8443/secure/Ra...

Nous avons remarqué que la représentation par les parcours applicatifs des interactions entre les US permettait de **faciliter la communication entre les parties prenantes du projet** : que cela soit entre testeurs ou entre testeurs et développeurs. Entre testeurs, nous utilisons la représentation visuelle pour suivre notre avancement au niveau de la conception, mais aussi au niveau du suivi de ce qui est développé. Ainsi, nous pouvons identifier si de mauvais choix sont faits en termes de priorisation de développement, qui pourraient être bloquants pour réaliser les tests au plus tôt. Ceci nous permet d'**identifier les liens entre les fonctionnalités et prioriser les développements et tests** à travers une démarche ATDD de test au plus tôt.
Enfin, nous construisons, à partir des parcours applicatifs, le scénario de démonstration pour le client.

Nous avions remarqué que lors de la revue de sprint, certaines fonctionnalités n'étaient pas présentées et la trame de présentation n'était pas fluide, présentait des lacunes et des discontinuités. Ceci s'expliquait par le manque de temps pour réaliser cette tâche et le manque de visibilité sur l'enchaînement des différentes fonctionnalités.

La mise en évidence des interactions entre les différentes US via les parcours applicatifs nous a permis **d'extraire une trame pour le scénario de démonstration.** En se basant sur ces parcours et les cas de test réalisés au fil du sprint, les testeurs conçoivent le scénario de démonstration. Ainsi, en fin de sprint, ils disposent du scénario complet et peuvent l'exécuter afin de valider la possibilité de dérouler la trame métier attendue. Il est, de plus, possible de personnaliser la publication de ce scénario : un scénario dans l'outil SquashTM pour l'exécution et un scénario au format tableau Excel, constitué de la suite des étapes à exécuter, pour la démonstration, formant ainsi la trame demandée. Nous avons pu réduire de 40% le temps alloué à la réalisation de ce scénario de démonstration et avons pu tracer son exécution en le publiant dans SquashTM.

b. La phase d'analyse et de conception des cas de test.

Cette phase consiste, entre autres, à étudier le référentiel de tests et à déterminer si des cas de tests doivent être mis à jour afin de garantir leur non-obsolescence. En règle générale, cette analyse peut être compliquée par le manque de visibilité sur les cas de test existant déjà.

Par expérience, avec l'enchaînement des sprints, nous constatons un risque de perdre en visibilité sur l'ensemble du référentiel de tests et nous ne sommes plus en mesure d'identifier efficacement les cas de test à modifier pour conserver le référentiel des tests en phase avec l'application sous test.

Comme évoqué précédemment, à chaque sprint, un ensemble de User Story est à analyser et de nouveaux cas de test doivent être conçus afin de couvrir les nouvelles fonctionnalités. Dans l'outil Yest, nous construisons l'arborescence des cas de test par fonction et sous-fonction. Ainsi quand une nouvelle US impacte une fonction ou sous-fonction existante, nous identifions facilement dans quel dossier nous devons mettre à jour nos parcours applicatifs. La possibilité d'avoir le référentiel de test dans l'outil de conception de test est un point fort, car nous pouvons ainsi mettre à jour les cas de test dans l'outil de conception et synchroniser directement les changements opérés dans notre outil d'exécution Squash™. Le succès de cette méthode repose sur le fait que les cas de test dans l'outil de conception des tests (Yest) et dans l'outil de gestion des tests (Squash™) sont identiques : on retrouve exactement les mêmes dossiers et les mêmes cas de test dans chacun des deux outils. Nous n'avons ainsi aucune difficulté à identifier les périmètres de changement.

6. Créer et maintenir des tests automatisés.

La mise en place de l'automatisation s'appuie sur la conception visuelle des tests permettant de générer et maintenir les scripts d'automatisation à base de mots clés. Dans notre outil de conception des tests, nous avons le référentiel de nos cas de test : pour l'automatisation, nous choisissons un ensemble de cas de test que nous souhaitons automatiser et nous les plaçons dans une campagne.

Ce travail est effectué par les testeurs fonctionnels intégrés à l'équipe Projet, permettant ainsi de choisir les cas de test à automatiser en priorité (selon les critères de temps d'exécution, de risque, de fréquence d'exécution, etc.) et en prenant en compte la faisabilité de l'automatisation (en présentant les cas de test à un automaticien). Nous basons donc notre automatisation sur des cas de test existants et en visibilité de l'ensemble de l'équipe Projet, permettant ainsi d'avoir une vision partagée du périmètre qui va être automatisé.

En termes de méthode d'automatisation, nous utilisons de façon classique un système de Keyword-driven où chaque étape de test va être liée à un ou plusieurs keywords. Une fois chaque étape d'un cas de test liée à des keywords, nous pouvons générer un script en Selenium ou Ranorex. Le système de Keyword-driven réduit l'effort d'automatisation : en effet, dès qu'un keyword est implémenté et lié à une étape de test, si cette étape de test est utilisée dans plusieurs cas de test, la liaison entre étape de test et keyword se fera automatiquement et ne demandera pas de nouvelle intervention d'automatisation. Ainsi, plus nous avançons dans l'implémentation des keywords, plus la production des scripts de test automatisé devient rapide. Après la génération des scripts, nous les publions dans notre outil d'exécution de test automatisé.

Synthèse et conclusion :

Les itérations, les cycles courts et l'acceptation des changements sont de réels défis à relever pour la mise en place d'un processus de test stable dans un contexte agile. Souvent, de nombreux aléas retardent le planning des tests et entraînent l'exécution des tests seulement en fin de cycle de développement, ce qui n'est pas souhaitable. Dans notre démarche d'ATDD avec la conception visuelle de tests, nous produisons les cas de test en amont des développements et testons les nouvelles fonctionnalités dès qu'elles sont disponibles. Nous gérons actuellement un référentiel de test de près de 400 cas de test avec l'exécution à chaque sprint d'une moyenne de 90 cas de test couvrant les évolutions.
Le cycle défini dans ce chapitre prend un ensemble d'artefacts constitué de User Story et d'exigences en entrée et produit un ensemble d'artefacts, constitué des cas de test couvrant les nouvelles fonctionnalités, et des cas de test couvrant le périmètre impacté par les nouvelles fonctionnalités en sortie. La conception visuelle des tests montre la couverture des processus Métier simples ou complexes en fonction des cas.

La phase d'analyse nous permet de prendre en compte très tôt le périmètre du sprint à venir au travers des parcours applicatifs que nous communiquons au sein de l'équipe : ceci a nettement amélioré l'organisation de nos sprints et la communication entre chaque partie prenante du projet.

La conception des cas de test a été facilitée à différents niveaux, autant au niveau de la conception même, à travers une approche visuelle se détachant des méthodes traditionnelles de rédaction, qu'au niveau de l'organisation du référentiel de test, réalisable au niveau de l'outil de conception. Nous avons ainsi nettement amélioré la traçabilité entre les artefacts de tests (US, exigences et cas de test) dans nos différents outils : Squash™, Jira et Yest. Enfin, nous avons facilité l'automatisation en rendant les concepteurs fonctionnels acteurs de celle-ci, en leur donnant la visibilité sur les cas de tests automatisés et la possibilité d'étendre le périmètre des cas de tests à automatiser via un publisher.

III.6- Les tests d'acceptation en Agile, ou quand la « pizza team » doit sortir de sa bulle !

Par Dina Rabenjarivelo et Tata Tandjigora

Le modèle de la bulle agile est assez séduisant : une équipe pluridisciplinaire, autonome et relativement préservée des contraintes techniques des grandes organisations. Cependant, la réalité est souvent plus nuancée.

Un projet agile doit à un moment « brancher » ses développements au reste du Système d'Information (SI) ou s'accoster à d'autres développements réalisés en cycle en V et/ou en Agile, voire avoir recours à un fournisseur externe, ou encore simplement tester ses développements sur des environnements plus proches de la production, sur lesquels l'équipe n'a pas la main. Et là, les ennuis commencent…

Les tests d'acceptation sont justement le moment où l'équipe agile doit sortir de sa « bulle » pour se confronter aux contraintes du monde extérieur.
Savoir traiter cette phase importante est un atout majeur pour ne pas perdre le bénéfice de l'Agilité.

Ce chapitre présentera trois modèles d'organisation des tests d'acceptation dans les projets agiles, illustrés à travers trois retours d'expérience au sein de BNP PARIBAS CARDIF.

1- Les enjeux de l'organisation des tests d'acceptation en Agile.

L'objectif des tests d'acceptation est de valider la conformité fonctionnelle métier des applications en vision bout en bout. Ces tests nécessitent l'appréhension d'un certain nombre de contraintes, suite à l'intégration avec le reste du Système d'Information. L'enjeu est d'insérer le produit développé et testé dans la bulle agile, dans un écosystème complexe.

a. Les contraintes à la mise en œuvre des tests d'acceptation : environnement, interaction avec des partenaires extérieurs et avec les autres projets.

Les deux principales contraintes à la mise en œuvre des tests d'acceptation sont la gestion des environnements et la collaboration avec des équipes ne faisant pas partie de la « pizza team ».

Pour les tests de bout en bout, il faut avant tout déterminer quelles sont les applications à prendre en compte, sans pour autant tout tester. Le Risk Based Testing (Tests Basés sur les Risques – RBT), partagé et validé avec le Métier et l'Information Technology (IT), permettra de savoir jusqu'où aller.

La réservation des environnements correspondants doit être effectuée, ainsi que leur préparation : branchement entre eux et accessibilité avec la gestion des habilitations, ouverture de flux, etc. Ces travaux doivent être pensés et préparés en amont et non en parallèle des travaux de l'équipe agile. En effet, il faut prendre en considération les contraintes des équipes externes qui sont en charge de ces travaux, dont les procédures et délais peuvent être incompatibles avec les contraintes des projets en Agile.

La gestion de ces environnements en cours de test doit être prise en compte : l'équipe peut être amenée à gérer deux environnements de test différents selon la stratégie adoptée, en fonction du type de projet et des applications liées. Un environnement pour les tests spécifiques à l'Agile, avec la validation des User Story (US), et l'autre pour les tests d'acceptation avec une recette de bout en bout.

Au-delà des problèmes d'environnement, il est également nécessaire de prendre en considération le mode de développement du projet : projet full Agile, avec une seule équipe ou plusieurs squads ; ou un projet hybride avec une partie des développements en cycle en V.

Si plusieurs équipes interviennent, la stratégie sera différente en fonction de l'appartenance de ces équipes : issues de la même entité, structure ou partenaires extérieurs. Effectivement, il peut arriver que certains développements/tests soient délégués ou externalisés.
La difficulté est de consolider les plannings de chacune de ces équipes pour déterminer le « bon moment » pour effectuer des tests de bout en bout significatifs et surtout le « comment ».

b. Comment insérer les tests d'acceptation dans le projet : release ou pas release ?

Les deux principaux modèles pour effectuer les tests d'acceptation adviennent soit à chaque fin de sprint, soit après un ensemble de sprints représentant une release. D'autres schémas peuvent en découler, en mixant ces deux modèles ou encore en jouant sur les différents environnements avec des tests de bout en bout décalés par rapport aux sprints.

Ces tests de bout en bout doivent être anticipés, réfléchis et organisés dès la phase de cadrage du projet : les effectuer au moins une fois avant la livraison du projet, pour s'assurer que la version qui partira en production est conforme à l'attente de l'utilisateur final.

Le choix du modèle est fonction des technologies utilisées sur le projet, ainsi que du nombre d'application impactées et nécessaires dans la phase de bout en bout. Un projet mono-applicatif, sur du web uniquement, ne demandera pas nécessairement de faire des tests de bout en bout à chaque fin de sprint, contrairement à un projet avec plusieurs applications et équipes travaillant sur différentes technologies. Plus il y a d'applications entrant en jeu, plus il est important d'effectuer des tests de bout en bout en amont.

Le choix du modèle dépend également de l'organisation, de la composition et de la maturité de l'équipe. Les tests d'acceptation doivent intégrer l'ensemble des évolutions développées et testées par tous les acteurs du projet – s'il comprend plusieurs équipes, internes ou externes, en Agile ou en cycle en V. Ils doivent être effectués par une équipe « transverse », ayant une vision globale du sujet et la capacité de comprendre le projet dans son ensemble et des applications connexes.
Ces tests doivent être préparés en amont de cette phase de recette, dès l'élaboration de la stratégie de test. Idéalement, la phase de recette doit être prise en charge par la même équipe qui a déroulé tout ou partie des tests de sprint car, pour être plus efficace, sa conception doit se faire en parallèle de l'exécution des tests système.

c. Comment garantir l'indépendance des tests tout en adoptant la logique d'équipe pluridisciplinaire ?

Le principe de l'Agile est d'avoir une équipe pluridisciplinaire, où chacun est responsabilisé sur les tests et la qualité. Mais chacun est aussi influencé par les enjeux du respect des objectifs du Scrum Master, avec un risque de la perte d'indépendance des testeurs embarqués dans l'équipe. Le testeur n'a plus la vision globale du projet et se laisse influencer par les arbitrages projet.

Au cours des tests, au sein même de l'équipe agile et toutes phases confondues, on entend souvent ce genre d'échanges :

> – *Développeur : Il ne faut pas tester ce cas de test, ce périmètre. Tu en fais trop.*
> – *Testeur : Et si ça arrive en production ?*
> – *Développeur : Mais non ! Cela n'arrivera pas en production. On n'y a pas touché.*
> – *Testeur : Tu es prêt à mettre ta main à couper ?*
> – *Développeur : Euh ... finalement on y a à peine touché … mais bon, je pense que tu en fais trop quand même. Mais … à toi de voir !*

Le testeur ne doit pas perdre de vue que, s'il appartient à une famille qui est celle de l'équipe agile, il est avant tout garant de la qualité globale du produit vis-à-vis du métier. Il doit toujours avoir l'esprit critique et garder une vision d'ensemble. Dans la mesure du possible, cette phase de test doit être confiée à une équipe indépendante de l'équipe agile : les mêmes interlocuteurs mais avec une casquette différente, ou de nouvelles personnes en dehors de la « pizza team ».

d. Comment maintenir une répartition de l'effort de test dans une logique « shift left » ?

Le principe du « shift left testing » (tester au plus tôt) signifie en Agile que l'effort de test doit être le plus possible intégré dans les sprints, au plus proche des développements. Mais pour les tests d'acceptation de bout en bout, cela est parfois difficile car ils nécessitent des environnements et des contributions plus diverses. La mise en œuvre de tests de bout en bout après les sprints agiles est souvent une nécessité mais cela est également coûteux, notamment en termes de délai. Anticiper ces tests d'acceptation, pour en intégrer tout ou partie à l'intérieur des sprints, devient un enjeu de réduction des délais. Le risque est de percevoir les tests de bout en bout comme un effet tunnel qui s'ajoute en fin de sprint et qui viendrait anéantir le bénéfice de l'Agilité.

La dynamique de sprint doit être exploitée en intégrant des tests de bout en bout soit directement dans le sprint, soit en parallèle de celui-ci. Ces tests peuvent être effectués partiellement, en fonction des applications et développements disponibles avant la release. C'est une notion que nous appellerons « pré-bout en bout ».
Cette séquence d'anticipation des tests multi-applications permet de sécuriser la réalisation du bout en bout, qui doit se dérouler sur la période la plus courte possible. La séquence de tests de bout en bout étant réduite à son minimum, il est primordial de l'optimiser et de s'affranchir des problèmes d'interconnexion des environnements et de cohérence des jeux de données. Les tests de pré-bout en bout serviront notamment à détecter en amont ces problèmes d'environnement.

e. Comment piloter la couverture de test globale et le dispositif associé ?

Les tests de bout en bout représentent une rupture par rapport au contexte de la bulle agile. Lors de cette phase, il est nécessaire d'appréhender certaines contraintes que l'équipe Scrum a temporairement mis de côté dans ses sprints pour se focaliser sur son périmètre. Le passage à un environnement non bouchonné, connecté au reste du SI, la prise en compte de la dépendance avec d'autres développements ayant une dynamique différente… toutes ces contraintes nécessitent un pilotage de la couverture de test par un acteur ayant une vision globale du projet et un certain recul par rapport au contexte de la bulle agile.

Cette phase de test d'acceptation doit être pilotée par un Test Leader ayant une vue globale des tests. Il est le garant de la couverture et de la cohérence de l'ensemble des tests, par rapport à ce qui a été défini avec le Métier et les risques remontés dans le RBT. Le Test Leader ne doit pas être intégré dans la « pizza team », pour pouvoir garantir cette prise de recul par rapport aux objectifs de l'équipe et ne pas se laisser absorber par les tests système. En effet, la « pizza team » a naturellement tendance à se concentrer sur son périmètre.

En cours de sprint, le Test Leader doit prendre en compte les grands principes et évolutions pour chacun des périmètres, chacune des applications et équipes entrant dans le projet, afin de les intégrer dans le bout en bout. Les plans de tests effectués pour les tests système doivent être réfléchis pour être exploitables pour la recette de bout en bout, avec une vision en processus Métier.

2- Trois modèles proposés pour organiser les tests d'acceptation, illustrés par trois retours d'expérience.

Si la phase de tests d'acceptation de bout en bout peut constituer une rupture par rapport au contexte de la bulle agile, il est nécessaire de bien penser en amont dans la stratégie de test globale du projet, du modèle d'organisation des tests à mettre en place. Nous avons ainsi défini trois modèles pour aider les Test Leaders à cadrer l'organisation des différents niveaux de test et particulièrement les tests d'acceptation.

a. En parallèle des sprints : réalisation des tests d'acceptation sur une release figée qui a vocation à partir en production ; poursuite des sprints pour le développement d'une autre release.

Ce modèle a été utilisé sur un projet multi-applications, faisant intervenir différentes technologies, sur plusieurs releases. Il s'agit d'un projet de refonte d'un parcours client avec une forte interaction avec le reste du SI, mélangeant les deux modes de développement : agile et cycle en V. Au total 2,5 ETP (équivalent temps plein) sont intervenus, sur une période de 3 ans.

Le choix de ce modèle a été motivé par la prise en compte des contraintes planning de toutes les équipes, avec le risque de devoir attendre certains développements pour exécuter les tests de bout en bout.
Pour absorber ces décalages d'avancement entre les développements de l'équipe agile et les autres et détecter au plus tôt les anomalies, des tests de pré-bout en bout ont été intégrés après chaque validation de sprint. Et en l'absence de l'ensemble des développements, certains d'entre eux ont été effectués en mode bouchonné.

Ces phases de pré-bout en bout et bout en bout ont été déroulées sur un environnement de recette dédié, différent de celui des tests de sprint, avec un décalage de version.

Ces phases de recette ont permis de détecter des anomalies spécifiques au bout en bout, entraînant parfois une adaptation de la solution par rapport à son intégration dans le SI. Les phases de pré-bout en bout ont également limité la découverte des anomalies d'environnement sur la dernière phase de recette, ce qui a permis de se concentrer sur le déroulement fonctionnel des tests selon les macroprocessus définis avec le métier.

Notons que pendant les recettes de bout en bout, il n'y a pas de sprint réalisé. En effet, toute l'équipe agile est concentrée sur cette phase : correction des anomalies et adaptations éventuelles de la solution.

b. A l'intérieur des sprints : intégration des tests d'acceptation à l'intérieur de chaque sprint.

La configuration du projet suivant nous a fait adopter le modèle standard d'intégration des tests d'acceptation au sein de chaque sprint, pour anticiper le bout en bout. En effet, il s'agit d'un projet de mise en œuvre d'un outil de simulation d'offre d'assurance, mono-applicatif, avec peu d'interaction avec le reste du SI, mais une forte combinatoire pour les jeux de données. 1,5 ETP a été dédié à ce projet, sur 3 mois.

Sur ce projet, l'automatisation a été privilégiée pour dérouler un maximum de combinatoire, dès les tests de sprints, pour alléger le poids de la non-régression.
Des tests d'acceptation ont également été exécutés et anticipés sur chacun de ces sprints, en prenant en compte les développements des sprints précédents. Notons que les derniers sprints ont été consacrés à des ajustements sur le produit.

Cette stratégie a permis d'alléger la validation finale sur la phase de bout en bout déroulée sur un environnement dédié.

c. En dehors des sprints agiles : à la fin des développements en sprint, réalisation d'une recette « classique » sur un environnement dédié .

Ce modèle a été appliqué sur les deux premières activités du programme ci-dessous ; programme multi-applications, utilisant différentes technologies et qui fait intervenir plusieurs équipes, internes et externes au groupe. Dans ce programme, neuf ETP sont intervenus sur trois ans et répartis sur trois activités :

– Trois équipes agiles sur la première activité, avec sept ETP
– Une équipe sur la deuxième activité, avec un ETP
– Et la troisième activité, toujours en Agile, a été externalisée, avec un ETP.

Les trois activités sont interdépendantes, chacune avec une forte interaction avec le reste du SI et intégrant les deux modes de développement : agile et cycle en V.

Le choix de la réalisation d'une recette d'acceptation uniquement à la fin de l'ensemble des sprints, a été motivé par deux critères différents.

Sur un des périmètres de la première activité, l'équipe agile et l'application étaient matures. Les risques étaient limités sur le SI. L'analyse du RBT a fait apparaître la non-nécessité d'effectuer des tests d'acceptation en cours de sprint.

La deuxième activité, gérée en externe de BNP PARIBAS CARDIF, ne nous a pas permis d'intervenir en cours de sprint. Les tests d'acceptation sont réalisés à la fin du sprint final pour valider la conformité fonctionnelle et Métier en vision bout en bout. A noter également que, pour limiter l'effet « tunnel » et mitiger les risques en recette d'acceptation, des tests système ont été réalisés à travers l'exécution de cahiers de tests fournis par l'éditeur, à l'issue de plusieurs sprints.

d. Les variantes de ces modèles.

Toujours sur le même programme, sur la première activité, nous avions trois équipes agiles intervenant sur des périmètres différents et chacune avec des organisations différentes pour les tests d'acceptation.

Outre le modèle évoqué dans le paragraphe précédent, un autre modèle a été appliqué : Exécution des tests d'acceptation à la fin de chacun des sprints sur la première release, sur le même environnement. Puis, sur les releases suivantes, les tests d'acceptation ont été déroulés sur un environnement dédié, en parallèle des sprints en cours. La version testée correspond à l'ensemble des sprints validés précédemment par les tests système et donc avec un sprint de décalage.

Les évolutions apportées sur le produit entre les deux releases, plus complexes, et le manque de maturité de l'équipe ont entraîné une modification de la stratégie des tests d'acceptation. Cela a permis de mieux détecter les anomalies de non-régression.

Conclusion

Il est essentiel de bâtir une stratégie de test qui pose le modèle le plus adapté aux besoins et contraintes du projet. Et de s'attacher au pilotage du dispositif de test.

Pour les projets dont tout ou partie des développements sont réalisés en Agile, les tests d'acceptation de bout en bout sont une phase délicate, dont la mise en œuvre est difficile et qui représente un risque d'effet tunnel. Pour l'équipe agile, c'est souvent le sentiment désagréable de revenir à du cycle en V. Pour l'équipe de test en charge de ce bout en bout, le risque est d'apparaître aux yeux du projet comme des « anti-Agilité », refusant de se fondre dans la démarche agile.

L'organisation des tests de bout en bout vient ainsi challenger les limites des modèles de développement agile. Il est donc nécessaire de partager la problématique avec l'équipe agile et de trouver le modèle d'organisation le plus adapté au contexte du projet. Le choix du modèle dépendra des caractéristiques du projet et surtout de ses contraintes : nombre d'applications impactées, technologie utilisée, mode de développement, etc. Le modèle doit être choisi dès le départ. En phase critique du projet, il est difficile de changer de modèle en cours.

Pour assurer la couverture de test et avoir un recul suffisant par rapport à l'équipe agile et ses objectifs, les tests d'acceptation doivent être pilotés par une personne ayant une certaine indépendance par rapport à la bulle agile. Etant indépendant, il aura assez de recul pour veiller à la cohérence d'ensemble du dispositif de test et accepter les éventuels arbitrages, en exposant les risques, notamment en vue de l'insertion des développements agiles dans le Système d'Information. Le rôle du Test Manager prend alors tout son sens dans ce type de projet. Il aura à charge d'adapter son dispositif et de rationaliser les tests de bout en bout pour ne pas perdre les bénéfices de l'Agilité, tout en maîtrisant les risques de disqualité lors de l'insertion du produit dans le système d'information.

III.7- Nous avons dit OUI à l'Agile !

Par Michael Granier et Stéphane Batteux

Quand le site s'appelait encore voyages-sncf.com, la situation était semblable à beaucoup de projets de développement :

- Un cycle en V pour gérer une application assez monolithique.
- Une équipe de testeurs séparée et délocalisée, à Nantes,
  - dont un automaticien pour réaliser l'automatisation des tests bout en bout.
- Différentes équipes Métier pour valider la recette client.
- Environ quatre releases majeures voyages-sncf.com par an avec de nombreuses phases de tests.
- Des équipes de développement de plus en plus importantes, pour le même produit, à Paris.

Avec cet état des lieux, l'amélioration du Time To Market paraissait chose compliquée mais loin d'être impossible.

En 2014, la méthode agile montait en puissance et quelques équipes en interne, sur d'autres projets, avaient commencé à adopter l'approche Scrum.

Au vu des résultats et des réponses qu'apportait la méthode face aux problématiques abordées, un grand chantier de transformation se lançait en 2014 afin d'adopter les « Feature Team » pour le projet voyages-sncf.com.

Le passage à l'Agile : chaque page est portée par une Feature Team Agile.

Il a été alors décidé :

- De découper la totalité du site (applicatif monolithique long à mettre en production) en plusieurs applicatifs représentant chacun une page du dialogue de vente, pour mieux maîtriser des mises en production plus régulières.
- De donner la responsabilité du développement de ces nouveaux applicatifs à des Feature Team autonomes d'une dizaine de personnes, responsables de leur périmètre.

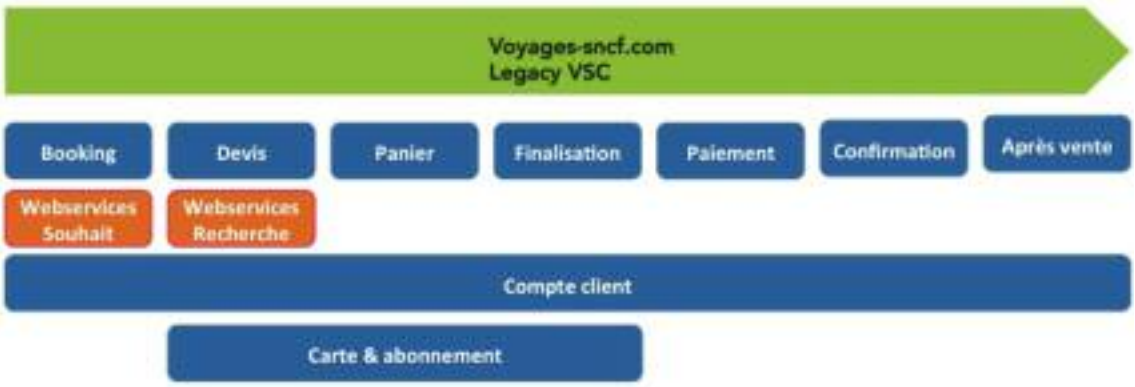
La première page à faire partie de ce grand plan de « Sortie du Legacy » fut la page « Devis » (la page où vous sélectionnez votre tarif) et l'équipe EWOK, la première Feature Team du site.

Cette première expérimentation a alors permis de faire de nombreuses constatations :

- Possibilité de réduire le TTM en mettant en production des incréments plus régulièrement (sprint de 2 semaines).
- Activer différents critères d'éligibilité pour permettre un accès maîtrisé à la nouvelle page devis depuis le Legacy.

- Avoir un modèle efficace d'équipes, beaucoup moins importantes et plus responsabilisées, pour développer un applicatif autonome.
- De l'amélioration continue et partagée dans les process et outillages, permettant expérimentations et innovations.

On a alors constaté que ce changement était possible et que le processus plaisait aux équipes. C'est ainsi que toutes les pages du site ont été attribuées à tout autant de Feature Team pour permettre un passage complet à l'Agile.



Découpage du site : Chaque bloc est porté par une Feature Team

L'autonomie des Feature Team a permis de voir apparaître plusieurs types de fonctionnement agile.

- En Feature Team utilisant Scrum ou Kanban, chacune intégrant le test dans son Definition of Done et rajoutant une colonne spécifique à la gestion des tests dans leurs boards.
- En Feature Team sans testeur, sur des projets bien spécifiques avec des développeurs sensibilisés au test et ayant opté pour une démarche BDD avec automatisation des features spécifiées avec le P.O.
- En tribu, permettant de créer des Squads, équipes applicatives éphémères, pour le développement de lots de Features.

Chaque équipe gère son Definition of Done des US (avec ou sans prise en compte de l'automatisation ou de la non-régression) et met en place les Dashboards nécessaires, liés à leurs activités de tests.

Cette autonomie est une valeur que la direction du Delivery Oui.sncf met en avant et souhaite conserver pour permettre l'innovation dans les équipes.

Le test et le testeur dans l'organisation agile.

La sortie du Legacy n'étant pas immédiate, l'Agile – et ses nouvelles briques – coexiste alors avec les Releases Legacy.

Les tests de ces dernières sont gérés par une équipe plus réduite et se concentre sur des évolutions mineures et de la maintenance.

Les testeurs participent à la vision bout en bout du produit, en exécutant des campagnes de non-régression bout en bout ainsi que de compatibilité entre le Legacy et les composants agiles. L'effort d'automatisation principal a été réalisé sur la non-régression bout en bout et a ainsi permis d'optimiser les cycles de Release de trois à un mois.

Pour les Feature Team agiles, l'intégration d'un à plusieurs ingénieurs qualité a permis de répondre à la charge de test des User Story de nouvelles évolutions pendant les sprints, mais aussi de la validation de la non-régression du sprint précédent, avant sa mise en production.

Le testeur, intégré dans l'équipe, participe donc à de multiples tâches :

- La rédaction des How to Test avec le PO (acceptation) et les tests des nouvelles US.
- Le chiffrage de la complexité des US avec le reste de l'équipe agile.
- La validation des évolutions sur les plateformes d'assemblage et recette avant mises à disposition au métier.
- La non-régression du périmètre applicatif de l'équipe.

On voit alors une évolution des métiers du test.

Testeur Proxy Product Owner : plus proche du PO, du Métier et donc du fonctionnel.

Il assiste le P.O. dans la rédaction des User story et des How to Test de ces dernières (Acceptation). Il rédige de même des User Story automatisation pour prioriser, dans chaque sprint, les tests à faire automatiser par les développeurs ou automaticiens.

Testeur Automaticien : profil plus technique, le testeur automaticien va développer, dans son équipe, l'automatisation des tests spécifiés avec le P.O. ou le testeur PPO.

Il est également le point d'échange privilégié avec l'équipe transverse afin de faire évoluer les outils de test communs.

Une équipe dédiée à l'automatisation des tests !

Le changement d'organisation a impliqué une forte automatisation des process, avec de l'Intégration Continue et des cycles de validation bien plus courts.

Ce qui a fait émerger un besoin fort de pouvoir automatiser certains tests.
A la fois pour ajouter de la plus-value à ces process et pour pouvoir outiller efficacement les testeurs.

Ainsi, pour épauler les différentes équipes, une nouvelle Feature Team dédiée à l'automatisation a été créée fin 2014 pour répondre à ce besoin.

Leur première mission a été de reprendre le Framework Selenium auparavant développé par les

équipes de développement, dans le but remplacer les tests QuickTest Pro de l'équipe de test. L'objectif était de rapprocher les développeurs et les testeurs autour d'une seule base de tests et d'outils communs.

Cela s'est passé en deux étapes :

- Développement des tests et du moteur dans l'équipe dédiée.
- Accompagnement et formation des automaticiens dans les équipes de développement.

En parallèle, l'équipe met également à disposition des projets de test pour les sites, applications mobiles et web services dans différentes technologies et propose une plateforme pour exécuter les tests automatisés.

Le but de cette plateforme est de faciliter l'intégration de nouvelles équipes et technologies tout en conservant une certaine qualité dans les activités de tests :

– Remontées des résultats et preuves de test dans les référentiels des équipes (ALM ou Jira/Xray).
– Rapports de tests et commentaires interprétables par tous.
– Exécutions des tests web sur Windows (Représentatif de notre audience).

Cela permet aux équipes de disposer d'un outillage clé en main pour faciliter l'entrée dans ses besoins d'automatisation.

Début 2016 et la généralisation des Feature Team, l'objectif de l'équipe a évolué.

Il s'agit alors de transmettre les process, outillages et le développement des tests à ces équipes, afin d'assurer leur autonomie sur la validation bout en bout.

L'équipe dédiée s'est alors spécialisée dans de nouvelles activités :

– La formation et la gestion transverse des projets de tests automatisés.
– La maintenance de la plateforme transverse d'automatisation des tests.
– L'accompagnement dans les process et bonnes pratiques autour de l'automatisation.
– La veille technologique sur les outils de tests et le partage des pratiques lors de clubs de tests.
    ° MForTest.
    ° Matinales de l'innovation (Altran).

En centralisant l'exécution des tests automatisés sur une plateforme, cela permet non seulement de profiter des outils de Big Data pour analyser les grandes tendances mais également de structurer l'information globale.

Aujourd'hui, l'équipe cherche à valoriser les données d'exécution des tests dans le domaine de l'analyse automatique des résultats de tests. En effet, les tests bout-en-bout en environnement dé-bouchonné sont beaucoup plus sensibles que les tests unitaires. Les résultats sont donc bruités par des tests faussement en erreur, ce qui demande un travail d'analyse.

En s'appuyant sur ces analyses et les techniques de machine learning, il a été possible avec l'aide

d'un stagiaire en Data Science de réaliser un prototype d'analyseur automatique de tests. Les résultats se sont avérés très positifs et l'équipe souhaite maintenant approfondir et généraliser cette solution.

N'hésitez pas à consulter notre présentation faite aux JFTL 2017 pour avoir notre retour d'expérience, plus graphique, sur la gestion de l'automatisation dans un contexte agile.

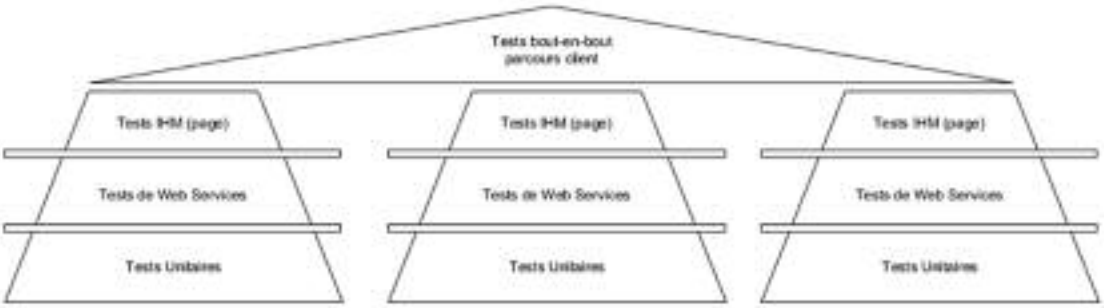
Vers une évolution des outillages.

L'évolution de l'outillage pour les activités de test reflète parfaitement notre transformation vers l'Agile.

Tout d'abord, concernant le référentiel de test, l'ensemble de nos tests fonctionnels était rassemblé dans une unique base HP ALM pour tout le site. Ces tests ont depuis été redistribués dans les différentes équipes qui ont décidé de les associer à leurs projets Jira via Xray, de sorte que les exigences fonctionnelles puissent de nouveau être associées à leurs tests.

Concernant les solutions d'automatisation des tests, la situation est plus complexe car d'une certaine manière, nous sommes passés d'un produit : le site web, à plusieurs produits : un par page.

Nous passons donc d'une pyramide de tests à plusieurs pyramides partageant le même sommet.



Ainsi, pour les tests de bout en bout, sommet de la pyramide, le projet de test est resté le même qu'avant notre transformation vers l'Agile. A la différence, non négligeable, que les tests développés sont à la charge des équipes.

En revanche, pour les autres niveaux de la pyramide, que ce soit sur l'IHM ou les web services, nombre de solutions ont vu le jour. Les équipes, autonomes sur leurs choix d'outils, se sont engagées dans le développement de projets d'automatisation spécifiques à leurs besoins.

On note également que les tests automatisés bout en bout prennent désormais moins de place que les tests de fonctionnalités ou composants, ce qui va dans le sens de la bonne pratique, représentée par la pyramide des tests.

En résumé, on peut donc dire qu'à l'image de notre architecture logicielle, nous sommes partis d'une stratégie centralisée sur l'équipe de test à une stratégie distribuée dans les équipes de développement. Mais si la forte autonomie des équipes a permis d'innover et de responsabiliser, cela a également créé des solutions très hétérogènes. Après une phase centrée sur la liberté des équipes, il y a maintenant une recherche d'équilibre entre liberté des équipes – valeur portée par l'entreprise – et efficience à l'échelle de l'organisation.

Problématiques d'une gestion globale du produit Oui.sncf.

Chaque équipe est autonome mais, au final, très liée à chaque composant en amont de son applicatif. Cependant, avec la responsabilisation essentiellement sur son périmètre, un effet silo s'est créé entre les équipes et a impacté la vision bout en bout du produit.

Chaque Feature Team s'assure que son produit est de qualité, mais son intégration à l'écosystème du site est portée par l'équipe gérant le Legacy.

Ce même Legacy étant voué à disparaître, ainsi que l'équipe associée, il sera crucial que cette vision soit, à terme, portée au sein des équipes.
Cela impacte directement le projet de tests automatisés bout en bout, collégialement porté par les équipes. Cependant, ces dernières ont des difficultés à fournir l'effort et la coordination pour leur maintenance et leur bon fonctionnement, dûes à un manque de stratégie de tests commune sur le produit complet.

Cycle de Release : Avec autant d'équipes agiles et d'applicatifs associés, en plus du Legacy, le site a maintenant un calendrier chargé avec presque une mise en production par jour.
Cette hétérogénéité des cycles de Release rend complexe l'estimation d'un niveau de confiance et de qualité du produit global final.

Pour répondre à ces problématiques, une nouvelle organisation est envisagée :

- Rassemblement autour de domaines fonctionnels structurants.
- Une gestion agile du Legacy jusqu'à sa disparition complète.
- Refonte de la stratégie de tests pour le site principal, portée avec les équipes.
- Liaison forte avec des communautés de pratiques transverses pour le partage.

Répondre au manque d'harmonisation ? Au niveau des process : refonte de la stratégie de tests.

En 2018, une grande ambition sur la stratégie de tests a été proposée aux Feature Team.

L'idée était de refondre, de formaliser et de mettre en œuvre une nouvelle stratégie de tests afin d'avoir une meilleure visibilité sur les validations des différents niveaux de tests et ainsi, mieux placer le curseur de la qualité pour les différents applicatifs.

L'objectif est de proposer des modèles pour une mise en commun d'indicateurs, de pratiques et d'outillages mais en gardant une souplesse permettant aux équipes de continuer à innover dans leur activité de test.

La première étape a été de rassembler et de faire collaborer la dizaine d'équipe en charge des différentes pages du site, à travers cette ambition commune.

Des représentants ont participé tout au long de l'année à des groupes de travail autour des niveaux de tests pour proposer des indicateurs, des expérimentations, en vue de les généraliser à tous les applicatifs.

La stratégie ainsi définie se décline autour de trois niveaux de tests :

– Les tests unitaires : s'assurer que la couverture de code est optimale par rapport au contexte de l'applicatif testé. La partie est gérée côté développeur avec un partage de pratiques dans la communauté associée et les indicateurs dans Sonar.
– Les tests de Feature : portés par chaque équipe pour valider essentiellement leur périmètre applicatif, que ce soit IHM ou Webservices.
Des indicateurs de couverture de fonctionnalités sont disponibles, générés grâce au référentiel de tests (Jira / Xray).
On voit même certaines équipes proposer la démarche BDD pour spécifier leurs US avec les comportements attendus, rédigés avec le Product Owner.
– Les tests bout en bout : portés par toutes les équipes afin de valider les cas business indispensables sur plusieurs critères :
° Trafic : il faut savoir que 50 cas utilisateurs représentent presque 80% du trafic en page paiement.
° Politique, juridique ou image : des sujets d'importance, même si les cas d'utilisation sont limités et peu nombreux
° Métier : plus ciblé sur des fonctionnalités Métier indispensables.

L'initialisation et les premières expérimentations aboutissent fin 2018 et la mise en œuvre doit à présent être accompagnée pour que nous puissions constater les gains.

Répondre au manque d'harmonisation ? Au niveau de l'outillage.

La réponse au manque d'harmonisation se trouve également dans l'outillage. En effet, dans notre situation où les équipes ont libre choix, il est important que les solutions proposées soient les plus simples et les plus pertinentes possibles. Ces solutions se retrouvant en concurrence directe avec les solutions commerciales ou open source disponibles sur Internet.
Nous travaillons également à la mise en place d'un catalogue de solutions d'automatisation, à l'image des centres logiciels présents dans les entreprises. Ainsi, l'objectif est que le catalogue

soit suffisamment large pour que la majorité des besoins soit comblée par les solutions communes identifiées. Pour les besoins plus exotiques, il sera bien sûr possible de recourir à de nouvelles solutions.

Répondre au manque d'harmonisation ? Une communauté des pratiques de tests.

Le partage des différentes pratiques de tests dans le groupe se fait en 2018 par le biais d'une communauté de pratiques ou Com'Test !

Elle permet de promouvoir les réalisations autour du test dans les différentes équipes :

- Outillages (automatisations…).
- Process (BDD…).
- Résolutions de problématiques tests.

Mais aussi de transmettre des informations sur les pratiques observées dans d'autres SI.

Cela permet de créer une dynamique de partage autour du test dans son ensemble et la création récente d'un référentiel des pratiques du SI.
L'objectif de ce dernier sera de rassembler, dans le même outil, les pratiques de tests, retours d'expérience de toutes les Feature Team.

La communauté participe aussi à une partie plus RH, où il est important de définir de manière claire les missions de l'ingénieur de tests dans les équipes agiles et de mettre en place des outils permettant une meilleure gestion des compétences tests.

La communauté dispose d'un backlog de réalisations et d'objectifs propres, afin de donner de la visibilité à ses actions et de mesurer les gains réalisés.

En évolution constante.

Notre transformation vers l'Agile nous a permis d'atteindre les objectifs que l'entreprise s'était fixés et notamment la réduction du Time To Market, en toute confiance. Les nouvelles fonctionnalités sortent de manière plus fluide et, grâce à l'automatisation à tous les niveaux, les mises en production sont en phase de devenir des non-événements.

Mais si personne ne déplore la fin du cycle en V chez Oui.sncf, il serait faux de dire que notre modèle actuel n'a que des vertus. En effet, notre mise en place de l'Agilité a créé une grande hétérogénéité des process et des outils de tests.

Il convient maintenant de continuer l'harmonisation des pratiques en profitant du meilleur des deux mondes. C'est-à-dire en conservant la rapidité et l'innovation permises par le niveau d'autonomie actuel, tout en rationalisant les ressources grâce à la mise en commun des solutions.

Le défi est de taille.

PARTIE IV

LES CONTRIBUTEURS

Abbas AHMAD

Ingénieur de recherche et validation, Abbas a effectué une thèse CIFRE au sein d'Easy Global Market où ses axes de recherche sont concentrés dans le domaine du Model-Based Testing (MBT), avec une approche orientée tests de système IoT, méthodes et outils.
Il dispose de deux certifications ISTQB® (International Software Testing Qualifications Board) niveau foundation level et Model-Based Tester foundation et d'une certification en automatisation de tests TTCN-3. Sa connaissance étendue du monde logiciel et des applications de test lui a permis d'être désigné FIWARE friendly tester, période de 4 mois durant laquelle il a testé le portail FIWARE et certains Generic Enablers.

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.

Hamza BAQA

Hamza BAQA effectue sa thèse au sein d'Easy Global Market, poursuivant ainsi son Master en Ingénierie des Télécommunications et Technologies de l'Information obtenu au sein de l'INPT Maroc, dont une année de partenariat avec Mines Saint-Etienne.
Au sein d'Easy Global Market, il est un fort contributeur du projet H2020 FIESTA, avec notamment des travaux de recherche dans les domaines de l'IoT, de la sémantique et du big data. Depuis mi-2016, il est également impliqué dans le projet WISE-IoT, collaboration Europe-Corée, contribuant à l'évaluation de qualité de l'information dans le monde des objets connectés.

Claude BARRAU

Responsable du centre d'expertise du test, de l'intégration et du déploiement.
Issu des filières technologiques en Electromécanique/Electronique, Claude entre chez Orange en 1984 dans le cadre du développement du réseau cuivre des P&T. C'est en 1994 qu'il tombe dans la marmite de l'informatique. Il exerce les métiers d'exploitant, administrateur système, DBA, expert système puis développeur et Intégrateur et pour finir testeur puis manageur. Depuis 2006 il assure la responsabilité de différents services sur le du test.

Stéphane BATTEUX

Développeur de formation, Stéphane met ses compétences au service du test et de la qualité logicielle. Après 7 ans d'experience, il propose des outils d'automatisation correspondant aux besoins de test des applicatifs.
Il aime répondre aux défis par des solutions innovantes.

Elodie BERNARD

Elodie BERNARD est Référente testing à Sogeti Lyon (France) et doctorante en Test logiciel à l'Institut FEMTO-ST (Besançon, France).
Elle travaille avec une nouvelle approche de conception de tests visuels pour l'automatisation des tests et la gestion du processus de test.

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.

Lionel BRAT

Engineering Manager Test au sein de la Guilde Test de la DSI AXA France.

Avec onze ans d'expérience dans le métier du test, Lionel manage une équipe de 25 test engineers. Il est garant de la qualité du delivery et de la maturité des pratiques sur les périmètres Data/RPA et sur les fonctions Transverses.
Certifié ISTQB au niveau « Avancé » et participant à la JFTL depuis 2014, il s'intéresse particulièrement aux enjeux et à la transformation actuelle du métier dans un contexte agile.

Sandrine BREBANT

Directrice de l'Ingénierie et de l'Expertise du Système d'Information.

Ingénieure en systèmes et réseaux informatiques, Sandrine Brébant a débuté sa carrière il y a 23 ans en tant que consultante dans les domaines d'administrations de systèmes et réseaux.

Chez France Telecom/Orange depuis 2002, toujours motivée par l'humain et la technique, elle a eu la responsabilité de divers domaines autour du SI et des plateformes de service.

Mette BRUHN-PEDERSEN

Mette Bruhn-Pedersen a travaillé comme Testeur, Test Manageur, Business Analyst, Scrum Master, Coach Agile et travaille actuellement comme leader de la transformation agile avec son entreprise Safe Journey.

Depuis 2014, Mette accompagne les équipes dans la transformation agile et la mise en œuvre des tests en SAFe®. Elle est co-auteure de son premier livre électronique « Quality and Testing in Scaled Agile Framework for Lean Enterprises », paru en 2018. Le livre a été recommandé comme un incontournable par Scaled Agile®.

Mickaël CARLIN

Ingénieur DevOps chez Altran, le rôle de Mickaël est de réfléchir, mettre en place et faire évoluer des chaînes outillées d'Intégration, Livraison et Déploiement Continus chez ses clients. Il s'agit d'un double travail humain et technique, puisqu'il faut mettre en place l'environnement technique en s'adaptant aux besoins du client mais aussi apporter de nouvelles façons de travailler, former et accompagner les équipes dans le changement.

Benjamin CARRIOU

Consultant DevOps et responsable de cette activité au sein de l'entreprise KEREVAL, Benjamin Carriou accompagne les entreprises dans leur transformation DevOps, tant d'un point de vue culturel que technique. Il porte un réel intérêt à des technologies telles que Docker, Kubernetes ou encore Rancher. Benjamin s'intéresse aussi de plus en plus à l'Agilité ainsi qu'au Software Craftsmanship afin de couvrir l'ensemble des besoins auxquels le DevOps ne saurait répondre.

Pascal CUGNET

Responsable de l'ingénierie au sein de la Guilde de test de la DSI AXA France.

Manager d'une équipe d'expert outils et méthodes depuis 3 ans, il accompagne plus de 200 testeurs de la DSI. Certifié ISTQB au niveau « Avancé », il partage régulièrement l'expérience d'AXA au sein de diverses communautés du test.

Olivier DENOO

Olivier est le président du CFTL.

Fort d'une carrière de plus de 25 ans dans la qualité logicielle, il est aussi le vice-président de ps_testware SAS, une ESN spécialisée dans le conseil et la formation.

Il occupe depuis plusieurs années un rôle clé au sein de l'ISTQB en tant que Governance Officer et des fonctions de liaison avec divers schémas de certification (IREB, IQBBA, Tmmi...).
Il participe régulièrement en tant qu'orateur à de nombreuses conférences internationales et a publié de nombreux articles dans la presse en ligne et sur le web.

Vincent DUGARIN

Engineering Manager Test au sein de la Guilde Test de la DSI AXA France, garant de la qualité du delivery et de la maturité des pratiques sur le périmètre des assurances collectives.

Vincent a plus de 10 ans d'expérience dans le test, il a animé des présentations et ateliers sur la méthodologie de test en Agile lors d'événements comme les JFTL ou l'Agile Tour.

David ESTEVES DA FONTE

Engineering Manager Test au sein de la Guilde Test de la DSI AXA France.

Manager d'une équipe de testeurs intégrés dans les équipes agiles et leader de la communauté des Engineering Manager Test de la DSI AXA France, il est garant de la qualité des projets livrés en production pour les Métiers : Offre Epargne, Prévoyance et Distribution.

Il est certifié ISTQB au niveau « Avancé » et passionné par le test depuis 15 ans.

Michaël GRANIER

Passionné de tests depuis 10 ans chez Oui.sncf, Michaël Granier participe activement à l'amélioration des process et outillages de tests en transverse, avec les différentes Feature Team Agile et testeurs, afin de tenter de répondre aux problématiques qualité du S.I.

Il participe également à l'animation de communautés actives autour des pratiques et outils de tests.

Fabrice GRIMBERT

Expert Test et Ingénierie logicielle.

Ingénieur en électronique et informatique, Fabrice Grimbert a débuté sa carrière à la SNECMA (SAFRAN) dans la division Etudes Avancées sur le développement de la régulation du moteur M88 du Rafale. Après plusieurs expériences dans les domaines aéronautique, naval et spatial, il s'est orienté vers le consulting des activités support au développement : qualité logicielle, gestion de configuration et déploiement, test et automatisation. Co-fondateur du CFTL en 2005, son activité est aujourd'hui focalisée sur les démarches de test et l'automatisation DevOps.

Emilie-Anne GUERCH

Coach à l'échelle au sein de la DSI AXA France.

Emilie-Anne Guerch est coach à l'échelle depuis 2016 au sein de la DSI AXA France. Elle coache notamment la Guilde Test, sur la transformation agile, organisationnelle et culturelle.

Marc HAGE CHAHINE

Passionné par le test logiciel, Marc s'investit sur ce sujet depuis le début de sa vie professionnelle.
Il travaille maintenant sur des missions d'expertise et partage ses connaissances et expériences à travers des articles et maintenant ce livre.
Il a également le plaisir d'animer le groupe LinkedIn « Le métier du test », ainsi que le blog « latavernedutesteur.fr », en collaboration avec Bruno Legeard.

Marcelo KAMENETZ SZWARCBARG

Ingénieur en Assurance Qualité pour Amadeus.
Marcelo est un passionné d'informatique. Assurer et promouvoir la qualité dans les produits distribués, c'est son métier.
Il est certifié ISTQB / CFTL Test Manager ainsi que Professional Scrum Master.

Le test est pour lui une évidence, trop souvent implicite et incomplète.

Anne KRAMER

Dr. Anne Kramer est Consultante Testing pour sepp.med, une SSII allemande spécialisée en assurance qualité de produits complexes dans des domaines à sécurité critique (médical, automobile, pharmaceutique, etc.). Chez sepp.med, Anne est à la fois conseillère en processus, chef de projet et enseignante. Elle est co-auteure de deux livres, dont un au sujet du test basé sur les modèles.

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. Il a publié plusieurs livres sur les pratiques des tests logiciels. 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.

Emmanuel LEHMANN

Coach Agile spécialisé en pratiques de développement au sein de la guilde de développement de la DSI AXA France.
Précédemment développeur expert en technologie Microsoft puis manager d'équipe à la DSI d'AXA France, Emmanuel intervient régulièrement au niveau des DSI et des tribus pour aider à la transformation vers DevOps.

Sébastien PANNETIER

Sébastien Pannetier travaille chez Sopra Steria depuis 9 ans. Il a d'abord été directeur de projet (il a piloté des projets nécessitant plusieurs dizaines de collaborateurs) puis responsable des Ressources Humaines. Il est désormais responsable commercial.

Bruno PEDOTTI

Responsable de Guilde Test AXA France.
Depuis près de 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, il 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.

Laurent PY

Avec plus de 15 ans d'expérience dans le domaine du logiciel, Laurent Py est désormais VP of products à HipTest et SmartBear Software. Il dirige notamment le développement produit chez HipTest, une plateforme pour les équipes agiles et DevOps qui utilisent BDD et ATDD.

Dina RABENJARIVELO

Titulaire d'un Master en Marketing Opérationnel, Dina Rabenjarivelo a évolué du métier Marketing vers le monde du testing au sein de Bnp Paribas Cardif. Forte de son expérience de plus de 11 ans sur les tests dans le métier Assurance, elle est actuellement Test Manager au sein du département Centre de Test. Elle a piloté plusieurs recettes en Agile et a contribué au cadre de référence du Groupe BNPP sur les tests en Agile.

Eric RIOU du COSQUER

Eric RIOU du COSQUER, expert international en Qualité et Test des Logiciels et Systèmes d'Information est membre du Bureau du CFTL, dont il a été président, et représente la France à l'ISTQB. Après une quinzaine d'années chez des éditeurs de logiciels, ESN, ou grands comptes comme Orange, il exerce maintenant comme auditeur, consultant et formateur. Fondateur et dirigeant de la société Certilog, il est également « Lead Assessor » TMMi et a réalisé la certification des premiers centres de test accrédités TMMi en France.

Christophe ROCHEFOLLE

Ingénieur Qualité & Sûreté de fonctionnement de l'École des Mines de Nantes (aujourd'hui IMT Atlantique), il a construit son expérience par la mise en place d'équipes qualité logicielle chez des éditeurs et start-up, notamment dans des environnements multisites et offshore.
Après avoir contribué à mener une démarche Agilité & DevOps pour le premier site de ventes événementielles, il a rejoint le groupe OUI.sncf afin de poursuivre la transformation DevOps par une approche d'Excellence Opérationnelle complétée par du Chaos Engineering.

Reynald STEVENS

Reynald Stevens est aujourd'hui le directeur marketing de StarDust Testing.
Grâce à ses expériences au marketing de Bouygues Telecom, puis à ses fonctions de fondateur et directeur d'une agence digitale, il possède une bonne expérience de la conception numérique en Agilité.

Tata TANDJIGORA

Arrivée en 2004 au sein d'une équipe Métier en tant que rédactrice gestion prévoyance chez Bnp Paribas Cardif, elle découvre le monde du test à l'occasion d'un mission d'expertise Métier sur un projet de refonte du SI prévoyance.
Depuis 2011, Tata intègre officiellement la famille du test. Elle est intervenue sur le premier projet agile réalisé chez Cardif. Actuellement Test Manager au sein du Centre de Test, elle pilote les tests d'un programme stratégique réalisé en Agile.

Cyril TARDIEU

Avec plus de 13 années d'expérience en tant que Test Manager, Coach Test et Agilité, Cyril accompagne aussi bien les startups en pleine croissance à rester agiles que les grands comptes à se transformer vers plus d'Agilité et de qualité. Installé dans le bassin genevois depuis 2018, il met ses compétences techniques et humaines au service de la réussite des équipes pour les aider à atteindre leurs objectifs. Il donne également des formations ISTQB pour diffuser le métier du test au sein des entreprises.

Alexis TODOSKOFF

Enseignant-chercheur à Polytech'Angers (ex ISTIA), Alexis est depuis 1999 à la tête d'une formation bac+5 (Ingénieur / Master) en Qualité Logicielle qui a diplômé plus de 300 étudiants en France. Depuis 10 ans, il dirige également les Relations Ecole-Entreprises de l'école d'ingénieurs. Il a rejoint le comité d'administration du CFTL en 2010 et coordonne depuis 5 ans les Relations avec les formations en France en Qualité et Test Logiciel. Il co-dirige, enfin, le Master 2 ITVL Ingénierie du Test et Validation Logicielle, formation continue à distance co-diplômante avec l'Université de Besançon et Polytech'Angers permettant de passer jusqu'à 7 certifications.

Julien VAN QUACKEBEKE

Julien Van Quackebeke est le fondateur de la société de consulting ALL4TEST, l'une des premières sociétés spécialistes de la qualité et du test logiciel en France (pure player).
La société a été créée en 2006 à Sophia-Antipolis. Elle est désormais présente à Paris et possède une test factory NearShore à Tunis afin de permettre à ses clients de monter facilement des plateaux de test pour leurs applications digitales.
Passionné par le sujet, Julien Van Quackebeke est également responsable de la commission « Test et qualité » de Telecom-Valley à Sophia-Antipolis et animateur du Meetup « Software tester club ».

ISBN : 978-2-9567490-0-4
Dépot légal 2019
Imprimé en France
Charte graphique et mise en page : Socreate