UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MISE EN OEUVRE D'INTÉGRATION CONTINUE IMPLIQUANT UNE LARGE BASE DE CODE PATRIMOINE

RAPPORT DE PROJET

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN GÉNIE LOGICIEL

PAR

ALEXANDRA LAPOINTE-BOISVERT

3 JUIN 2019

REMERCIEMENTS

Je tiens d'abord à remercier ma directrice de projet, Sylvie Trudel. Elle a été pour moi d'un immense soutien dans la réalisation de mon projet, présente du premier jour au jour J. Elle m'a comprise, a bousculé mes idées, m'a fait réfléchir et m'a permise, au fil de nos rencontres, d'aller plus loin dans ma démarche et de devenir une meilleure personne.

Je remercie également les généreux donateurs qui m'ont appuyée financièrement dans la poursuite de mon projet. Je remercie la Fondation de l'UQAM et Ubéker. Un merci tout spécial à Jean André de Ubéker qui m'a également offert son expertise dans le domaine.

Je remercie Broadsign qui m'a appuyée et donné l'opportunité de réaliser ce projet dans leur organisation ainsi que mes collègues m'ayant appuyée et soutenue dans mon projet.

Je remercie l'UQAM et les enseignants qui m'ont formée et m'ont donné envie de toujours vouloir aller plus loin dans mon parcours académique. Sans eux je ne serais pas rendue ici.

Finalement, un gros merci à ma famille et à mes proches qui avez su comprendre ma réalité et m'appuyer dans ce projet qui m'était cher.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS ET ACRONYMES

EP	*Execution Point*
HDMI	*High Definition Multimedia Interface*
H-P	Heures-personnes
J-P	Jour-personne
IT	*Information technology*
MVP	*Minimum Viable Product*
PO	*Product Owner*
POC	*Proof Of Concept*
ROI	*Return On Investment*
SM	*Scrum Master*
SP	*Story Point*
SUT	*System Under Test*

RÉSUMÉ

La rapidité de mise en marché est un facteur clé dans la performance des organisations œuvrant dans le développement de logiciels. Afin de répondre rapidement aux besoins des clients et de rester au-devant de la compétition, l'entreprise se doit d'avoir un processus de développement logiciel efficace pour permettre celle-ci.

Le passage d'un cycle de développement traditionnel vers un cycle de développement agile a permis à l'organisation d'augmenter sa réactivité à faire des mises en marché plus prédictives et régulières. Cependant, les tests de non-régression manuels restent un obstacle majeur à l'augmentation de la fréquence mises en marchés.

Ce projet vise à établir une approche d'automatisation des tests de non-régression dans une base de code patrimoine. Sous forme de preuve de concept, il aidera à la prise de décisions quant à l'investissement requis pour compléter le projet, à augmenter l'expertise de l'entreprise en automatisation, à démontrer la faisabilité du projet à l'interne et à évaluer le ROI potentiel de l'automatisation complète tout en ayant un impact concret sur la prochaine période de tests de non-régression.

Mon projet démontra qu'à l'aide d'un jeu de 15 tests de non-régressions il est possible de réduire l'effort des tests de non-régression de plus de 98%. Il sera également démontré que la poursuite du projet jusqu'à sa complétion permettrait de faire passer la durée de la période de tests de non-régression de huit semaines à moins d'une semaine, ce qui permettrait ainsi à la compagnie d'augmenter la fréquence des mises en marché.

Afin d'atteindre ces objectifs, un certain nombre de recommandations sont proposées.

INTRODUCTION

Le projet, effectué dans le cadre de la maîtrise en génie logiciel, vise à intégrer les pratiques de pointe du génie logiciel dans un contexte industriel. Plus particulièrement, le projet consiste à mettre en œuvre l'automatisation des tests de non-régression dans une large base de code patrimoine. En plus des défis techniques, le projet consiste à amener une culture plus favorable à l'acceptation de l'automatisation comme solution d'accélération des mises en marchés.

Le document est structuré en sept chapitres :

- Le premier chapitre consiste à introduire le plus fidèlement possible le contexte de l'entreprise en mettant l'emphase sur son cycle de développement, à décrire la problématique et les objectifs du projet ;
- Le second chapitre est une revue de l'état de l'art présentant les meilleures pratiques quant à l'automatisation de tests manuels et les différentes approches d'organisations ayant vécu des défis similaires ;
- Le troisième chapitre décrit l'approche qui sera utilisée pour résoudre la problématique telle que décrite au premier chapitre en fonction de la revue de l'état de l'art détaillée au second chapitre ;
- Le quatrième chapitre contient les résultats de l'approche effectuée dans l'organisation ;
- Le cinquième chapitre consiste en une analyse des retombées du projet ;
- Le sixième chapitre contient les recommandations pour la poursuite du développement des tests ;
- Le dernier chapitre contient la conclusion.

CHAPITRE I

CONTEXTE ET PROBLÉMATIQUE

1.1 Contexte

1.1.1 La compagnie

Broadsign est une compagnie de développement logiciel internationale. Fondée à Montréal en 2004 (Broadsign, 2017a), elle est présente dans plus de quarante pays (BroadSign). Passant de 30 à plus de 100 employés au cours des quatre dernières années, l'entreprise est en pleine expansion. Je contribue par ailleurs à cette expansion par mon embauche en 2015 à titre de *product owner* (*PO*), rôle que j'occupe toujours actuellement.

Broadsign offre une suite de logiciels d'affichage et de gestion de contenu numérique s'adressant principalement aux compagnies œuvrant dans la vente d'espaces publicitaires[1]. La clientèle de ces compagnies se composant principalement d'annonceurs[2] ou de leurs intermédiaires : les agences de publicité.

Le but des annonceurs est de d'atteindre un public ciblé « afin de l'inciter à adopter un comportement souhaité » (Publicité, 2018, 28 juin, 12 h 38). L'annonceur cible une audience, un secteur géographique et le tout selon des critères. Il s'intéresse aussi à la

---

[1] Un espace publicitaire est de manière générale un écran permettant d'afficher du contenu publicitaire.

[2] Personne, entreprise, organisme ou association, privé ou public, recourant à la communication pour sensibiliser une partie de la population (cible) à sa personne, à sa ou ses marques, à son ou ses produits ou à son enseigne. Jean-Marc Lehu, *L'encyclopédie du marketing : [commentée & illustrée]*, 2e éd., (2012).

rentabilité de sa campagne, il désire connaître les statistiques liées à sa diffusion : le nombre d'impressions, le nombre de répétitions, le nombre d'interactions, les caractéristiques de l'audience, etc. Ce sont tous des éléments devant être accessibles via un logiciel de gestion de contenu numérique tel que celui de Broadsign.

Le produit principal, le lecteur, est le logiciel qui permet de diffuser du contenu numérique dans un espace publicitaire. La diffusion du contenu est faite de façon dynamique en se basant sur une liste de critères fournie via le logiciel de gestion. Cela permet aux fournisseurs d'espaces publicitaires opérant de larges réseaux de répondre aux besoins de leurs clients, sans devoir gérer des listes de lectures manuellement.

Le marché des annonceurs, plus particulièrement les agences de publicité est très compétitif. Les acteurs doivent faire preuve de créativité, ce qui se solde régulièrement par une utilisation des plus récentes technologies en affichage numérique. La synchronisation entre plusieurs écrans, l'interactivité avec le spectateur ciblé, l'utilisation de contenu web, l'utilisation d'écrans à hautes résolutions en sont des exemples actuels mais de nouveaux besoins émergent continuellement. Améliorer la qualité des informations sur l'audience est aussi un besoin grandissant.

Cela constitue des défis importants pour l'organisation devant régulièrement ajouter des nouvelles fonctionnalités très rapidement, nécessitant souvent l'usage de technologies récentes devant fonctionner sur du matériel et des systèmes d'exploitation plus anciens puisque les lecteurs sont rarement remplacés sur le terrain.

La suite de produits comprend quelques autres produits, mais le cœur de la suite de produits est le lecteur (le client) ; le serveur et l'application de bureau de gestion. L'utilisateur peut installer le lecteur sur les plateformes Windows, Linux ou Android tandis que le logiciel de gestion s'installe sur Windows. Le serveur est situé dans un nuage privé géré par Broadsign. Le développement du cœur se fait majoritairement en

C++ et est pris en charge par une vingtaine de membres (développeurs, testeurs et gestionnaires) : l'équipe *CORE*.

1.1.2 Son historique

L'organisation ayant subi une forte croissante dans les dernières années a fait le constat que le cycle de développement en cascades avec lequel elle était habituée ne convenait plus, la gestion devenant de plus en plus difficile. L'organisation désirait avoir un cycle de développement plus flexible et plus adaptatif et favoriser l'auto-organisation des équipes. Broadsign a donc opté d'aller vers des méthodes de développement agiles et a choisi d'implémenter le cadre Scrum[3].

Afin d'entamer le changement, la culture organisationnelle, se situant plutôt à la droite sur le quadrant de Schneider (Schneider, 1994) , a dû être ajustée. Une culture de contrôle et de compétence se mariant difficilement avec l'agilité, la direction a donc initié et soutenu un décalage de la culture vers la gauche, valorisant davantage la collaboration entre les individus et leur développement.

La transition entraînant inévitablement des bouleversements pour les individus ; l'organisation a mandaté un *coach* agile pour les accompagner. Depuis le début de la transition en 2014, la résistance au changement est un défi constant pour l'organisation. Cependant, la majorité des individus se sont adaptés au nouveau paradigme et le *coach* a terminé son mandat en 2017.

---

[3] Cadre de travail établi et basé sur les valeurs du manifeste Agile qui prescrit des rôles, des livrables et des événements définis. Ken Schwaber et Jeff Sutherland, «The Scrum Guide™», (2016)

Bien que la majorité des projets de l'organisation utilisent SCRUM, certains projets tels le projet SWAT utilise plutôt le mode KANBAN dû à la haute fréquence de déplacement des priorités entre autres.

Les prochaines sections décriront les processus reliés au développement de l'équipe CORE. Comme il pourra être observé, la plupart des processus sont maintenant agiles. Par contre, certains sont toujours teintés par le modèle de développement en cascades.

1.1.3 Les projets

La charte d'équipe

Lorsqu'un nouveau projet (*release*) est démarré, l'équipe de *release* commence tout d'abord par réviser et ajuster au besoin la charte d'équipe. La charte d'équipe est un document écrit servant de ligne de conduite pour le l'équipe décrivant notamment le mode de fonctionnement, le mode décisionnel et la description des rôles. La création et/ou révision de la charte d'équipe dure entre 1 et 2 heures et se termine par l'approbation de la charte par consensus des participants.

La charte de projet

L'équipe de *release* se réunit ensuite pour composer la charte du projet. Cette charte contient grosso modo la vision du *release*, les risques accompagnés de leur plan de mitigation et/ou de contingence, la matrice de leviers[4] et des informations pertinentes à la réalisation du *release*. La création de la charte de projet se fait généralement en trois ou quatre rencontres de deux heures et doit être approuvée par le *sponsor* (son représentant) du projet avant d'entamer la prochaine étape.

Création du carnet de produit

La première étape consiste à créer les *epics*[4] qui devront par la suite être divisées en *user stories* plus tard afin de préciser l'estimation. Bien sûr, il y a toujours trop d'*epics* pour la capacité d'un *release* et le carnet de produit doit subir un *grooming*[5]. En se basant sur la vélocité moyenne du dernier *release*, le *PO* estime le nombre de *story points*[6] (sp) disponibles de manière optimiste et pessimiste et communique la portée à viser aux autres parties prenantes.

Pour planifier la portée, les différentes parties présentent leur liste de besoins recueillis pour le prochain *release*. Toutes les *epics* souhaitées sont expliquées et notées sur des *post-its* et une première estimation en équipe est faite. Les *epics* sont priorisées et apposés et les *post-its* sur un mur (voir figure 1.1). Le *PO* trace alors un trait vertical après la dernière *epic* qui permet d'atteindre la capacité maximale du *release*.

L'équipe classifie ensuite les *epics* en trois catégories :

A, B et C ayant les significations suivantes :

A : *MVP*[7] : le produit sera livré seulement si ces *epics* sont complétées ;

B : Très importantes : pouvant éventuellement être reportées au *release* suivant, mais pas au subséquent ;

---

[4] Les epics sont une description à haut niveau d'un problème à résoudre. Elles comportent assez de détail pour pouvoir être estimées grossièrement.

[5] Le *grooming* est une technique qui consiste à prioriser et ajouter ou retirer au besoin des éléments d'un carnet de produit.

[6] Unité de point couramment utilisée dans les équipes agiles permettant d'évaluer l'effort d'une story.

[7] *Minimum Viable Product* : Sous-ensemble minimal de fonctionnalités requises pour pour considérer l'incrément de développement suffisant pour être mis en marché.

C : Souhaitées : pouvant être reportées à un futur *release*.

C'est généralement ici que le gros du travail de préparation de l'équipe de *release* se termine, le *PO* prenant alors la relève avec l'équipe de développement.

Figure 1.1 *Epics* et user stories d'un *release* représentées sur des *post-its*

Division des *epics* en *user stories* et estimation

Subséquemment, le *PO* présente les *epics* à l'équipe de développement ; leur mission étant de les diviser en *user stories*. Cette étape mène souvent à des découvertes forçant l'équipe de *release* à réévaluer les *epics* considérées. L'équipe de développement procède ensuite aux estimations des *user stories* qui, additionnées, préciseront l'estimation des *epics*. Suite à cela, une dernière rencontre est faite avec l'équipe de

release qui officialise sa portée et produit le *release plan (*voir appendice A pour un exemple de *release plan*).

Déroulement des projets

Les sprints se succèdent à l'intérieur d'un projet qui contient typiquement entre 13 et 26 sprints ; les projets se complètent donc entre six et douze mois. Généralement, sur un projet de 13 sprints, 8 sprints seront consacrés au développement et les 5 derniers sprints seront alloués aux tests de non-régression et à la préparation de la livraison finale : la phase de stabilisation (voir 1.1.5). La transition se fait lorsque l'équipe atteint le *feature freeze*, c'est-à-dire le moment où la décision est prise de ne plus développer de nouvelles fonctionnalités et où la phase de stabilisation est entamée.

1.1.4 Le développement

Les membres de l'équipe de développement sont divisés en deux équipes de 7 à 9 membres, incluant le *PO* et le *Scrum master*. Chaque équipe possède ses objectifs de sprint distincts. Les objectifs de sprint sont établis par le *PO* et le *Scrum master* supporte et aide l'équipe dans l'atteinte de ces objectifs tout au long du sprint.

Les sprints ont une durée de deux semaines incluant l'équivalent d'environ une demi-journée consacrée aux activités suivantes :

- **2 sessions de design :** les sessions de design visent à réviser les solutions proposées aux problèmes décrits dans les stories à venir dans le prochain sprint ;
- **1 revue de sprint** : session permettant aux parties intéressées de voir le développement ayant été accompli dans le sprint ;
- **1 planification :** session ayant pour but de planifier le travail à faire dans le prochain sprint en tenant compte de la capacité de l'équipe en nombre d'heures ;

- **1 rétrospective de sprint :** session permettant à l'équipe de revenir sur les bons et moins bons coups du sprint dans le but de s'améliorer ;
- **9 mêlées quotidiennes :** session d'une dizaine de minute en matinée permettant à l'équipe de se synchroniser sur le travail à faire.

1.1.5 La phase de stabilisation

La phase de stabilisation (ou période de tests de non-régression) consiste à exécuter des tests de non-régression, à corriger les bogues trouvés via afin d'assurer la qualité du produit à livrer. Elle se termine par une série d'activités de préparation à la livraison qui ne seront pas illustrées dans ce projet.

Avant d'entamer mon rôle de *PO* pour l'organisation, j'ai tout d'abord été affectée à l'équipe de contrôle qualité (*QA*) pendant trois mois afin de me familiariser avec le produit. Ces trois mois m'ont permis de bien comprendre les processus et fonctionnements du département de *QA* et de pouvoir ainsi les décrire de manière détaillée.

L'équipe de *QA* est constituée de cinq testeurs dont le mandat est d'assurer le contrôle de la qualité en écrivant et exécutant différents scénarios de test ; mandat pour lequel une expérience en programmation n'est pas requise.

Les tests étant entièrement manuels, le processus d'exécution des tests de non-régression est long et fastidieux. Certaines tentatives d'automatisation ont été menées dans le passé sans toutefois être fructueuses, elles seront décrites en 1.3 . Les prochaines sections illustrent le processus.

Phase de préparation

Avant d'entamer l'exécution des tests de non-régression, les testeurs doivent s'assurer qu'ils ont tous l'environnement (logiciel et matériel) nécessaire pour effectuer tous les tests. Cela demande d'avoir autant d'installations physiques qu'il y a de plateformes à supporter, incluant tous les périphériques nécessaires (écran, clavier, câble réseau, etc.). Cette phase n'est pas comptabilisée dans la durée de la période de non-régression mais constitue tout de même un effort additionnel imposé par les tests de non-régression manuels.

Sélection et priorisation des tests

Ensuite ou parallèlement, les équipiers évaluent parmi le jeu de 5821 tests[8] manuels existants dans Testopia[9] (Broadsign, 2017b)[10], lesquels devront être exécutés pour le projet courant. L'équipe ne revisite pas les 5821 tests à chaque fois mais fait plutôt une sélection sur une *run*[11] préétablie qui contient en moyenne 833 tests (voir Figure 1.2) ; chiffre augmentant de projets en projets. C'est ce que Rothermel et Harrold (1997) ont décrit comme la « sélection de tests » ; stratégie consistant à réduire l'ensemble de tests à un sous-ensemble plus petit afin de maintenir les coûts de régression raisonnables.

Habituellement, deux membres de l'équipe complètent l'activité en quatre ou cinq rencontres s'échelonnant sur une ou deux semaines. Au terme de ces rencontres, une

[8] Les tests sont majoritairement des tests fonctionnels en boîte noire mais quelques-uns servent à évaluer la performance, la consommation de mémoire et d'autres qualités architecturales. Les tests sont écrits en anglais sous forme de liste numérotée d'une dizaine d'étapes en moyenne.

[9] Application Web de gestion de cas de tests. Ed Fuentetaja et Greg Hendricks, «Testopia», (2013)

[10] Source interne provenant de l'intranet Testopia de Broadsign, source non publique.

[11] Chez Broadsign, l'équipe nomme une *run*, suivant l'appellation dans Testopia, une série de tests sélectionnés qui seront exécutés pour une période de non-régression.

suite de tests ayant des priorités allant de 1 à 5 est établie. La priorisation a pour principal but de découvrir le plus tôt possible les régressions ; les plus prioritaires étant les tests les plus susceptibles d'en contenir basé sur l'expérience des testeurs ; technique ayant entre autres été documentée par Elbaum *et al.* (2004).

Exécution des tests (ou période de non-régression)

Les tests sont ensuite exécutés suivant leurs priorités, les priorités 1 étant exécutés en premier, les priorités 5 étant exécutés si le temps le permet. Historiquement, les priorités 4 et 5 ne sont jamais exécutés et 433 tests sont effectués en moyenne, c.-à-d. 52% des tests prévus. Valeur qui est de plus à la baisse ayant passé respectivement de 66%, à 59% et à 49,9% au cours des trois derniers *releases* (voir Figure 1.2).

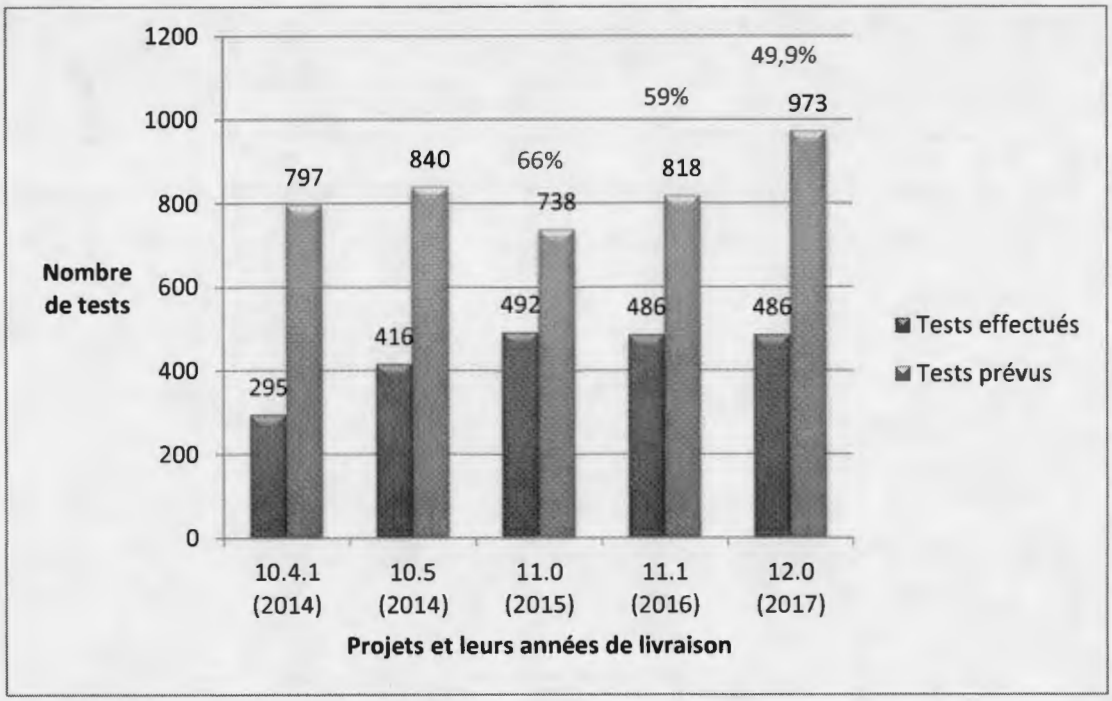


Figure 1.2 Nombre de tests prévus et effectués en période de non-régression

Un test manuel nécessite une intervention allant de 10 minutes à 4 heures. Lorsqu'un testeur est prêt à exécuter un test, il s'assigne d'abord au cas de test dans le logiciel de gestion de test Testopia. Il modifie ensuite le statut de la *story* dans JIRA[12] pour *En cours*. Une fois le test complété, c.-à-d. qu'il a été validé sur toutes les plateformes supportées, le testeur change le statut pour *Réussi* ou *Échoué* dans Testopia et liste les plateformes testées avec leurs versions. Il place ensuite, dans JIRA, la *story* dans le statut *Complété* ou la remet au statut *À faire* dans le cas d'un échec. De plus, si le test échoue, le testeur ouvre un billet de bogue pour la *story* dans JIRA et indique la procédure de reproduction ainsi que les plateformes en défaut. Un développeur peut alors s'assigner le billet de bogue, corriger le défaut et remettre la *story* en validation afin que l'équipe QA reprenne les tests. Ce processus est long et il y a beaucoup de va-et-vient entre les développeurs et les testeurs. Durant la régression chez Broadsign, un testeur exécute entre un et cinq tests manuels par jour (Luo, 2018), tout dépendamment de la nature du test.

Tout au long de l'exécution des tests, les régressions trouvées sont corrigées. Après vient une période où l'équipe juge que le code est stable et elle autorise le *code freeze*[13] ; c'est-à-dire que plus aucune modification ou correction n'est autorisée. À ce moment, l'équipe QA ré-exécute les tests de priorité 1, étape appelée *package validation* durant 1 semaine. Si tous les tests sont réussis ou qu'aucun *release blocker*[14], le produit peut alors être livré.

---

[12] Outil de gestion pour le développement logiciel, permettant entres autres le suivi de bogues et de projets. Atlassian, «JIRA Software», (© 2017)

[13] Le moment du *code freeze* n'est pas encore indiqué au moment du sprint 6, il est ajouté lorsque le *feature freeze* est atteint par souci de lisibilité.

[14] Certains bogues trouvés peuvent être jugés comme tolérables et pouvant être corrigés lors d'une prochaine mise à jour. Par contre, certains bogues, qualifiés de *release blockers* doivent par définition être corrigés à défaut de livrer le produit.

1.1.6 Améliorations récentes des processus

La transition vers des méthodes plus agiles a apporté de nombreux changements positifs pour l'organisation. L'embauche du *coach* agile à cette finalité a grandement aidé l'organisation à renforcer et maintenir ces changements.

Parmi ces changements, les plus notoires sont l'augmentation de la visibilité sur le processus de développement, l'augmentation de l'efficacité du processus de développement et l'amélioration de la communication entre les individus.

L'augmentation de la visibilité des processus est selon moi le changement qui a eu le plus d'impact au niveau de l'organisation, en faisant « apparaître » des problèmes. En réalité, ces problèmes étaient présents, ils ont tout simplement été rendus visibles, donnant à l'organisation une belle opportunité de les résoudre. L'opportunité fût saisie et soutenue.

Je présenterai ici quelques améliorations en découlant et auxquelles j'ai contribué, la liste n'étant toutefois pas exhaustive.

- **documentation du *release plan :* *document*** augmentant la visibilité sur les différentes échéances à atteindre au cours d'un projet (exemple à l'appendice A).
- **catégorisation des *stories* :** séparation des *stories* en quatre catégories (voir tableau B1 de l'appendice B pour une description des catégories) permettant de suivre la répartition du développement.
- **utilisation adaptée du *sunset graph* :** représentation graphique (voir figure 1.3) tenant sur une page illustrant à la fois la portée et ses changements ; la progression du projet en fonction d'un seuil de satisfaction minimum et les tendances pessimistes et optimistes quant à la date de livraison prévue (Boisvert et Trudel, 2011a).

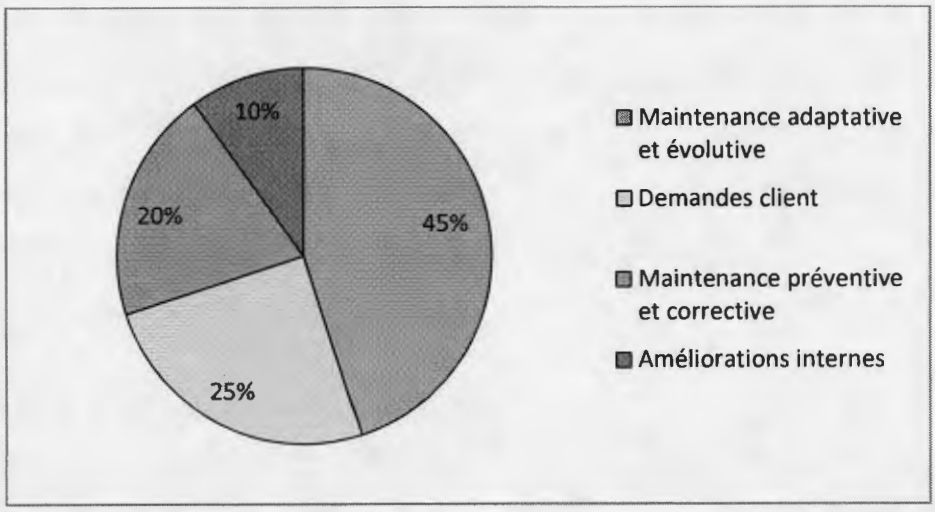


Figure 1.3 Répartition du développement dans un projet donné

Les couleurs permettent de voir rapidement en cours de projet où sont investis les efforts de développement. L'idée étant de respecter les ratios ayant été planifiés en début de *release*.

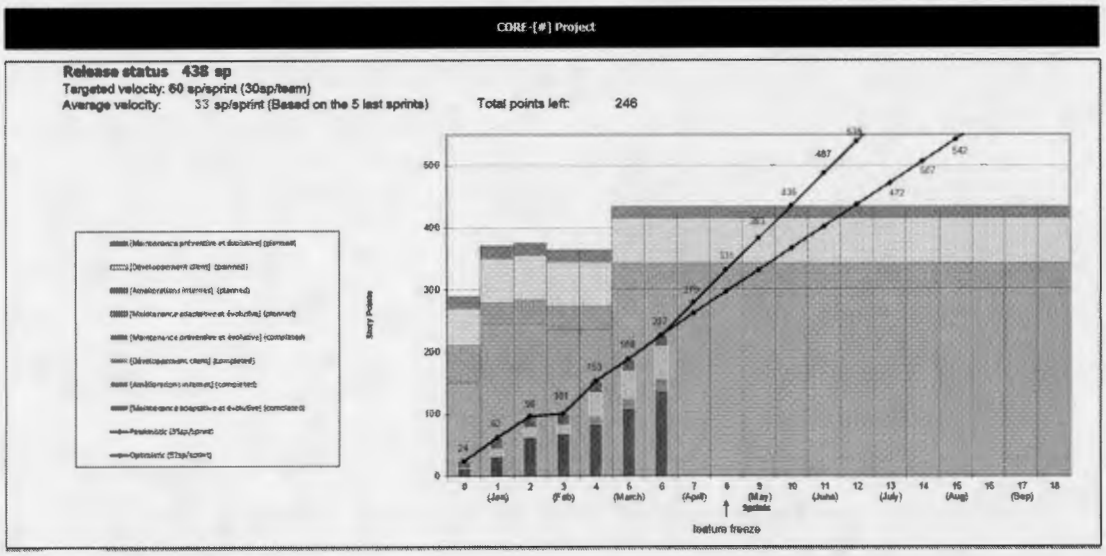


Figure 1.4 *Sunset graph* adapté aux besoins de Broadsign

Exemple du graphique adapté lors du sprint 6 d'un projet passé. L'arrière-plan représente la portée du release, les languettes représentent le développement complété et les droites représentent la projection pessimiste et optimiste quant à la complétion du développement par rapport à la date de livraison. Le graphique se distingue du *sunset graph* original des amélioration donnant de l'information plus précise.

1.1.7 Synthèse du contexte

En résumé, 3 ans après le début de la transition agile, l'organisation a pris beaucoup de maturité. La majorité des activités de développement de l'équipe CORE sont maintenant planifiées et exécutées de manière systémique et tous les rôles de l'équipe sont clairement définis et documentés. Les individus sont motivés, ont envie d'atteindre les objectifs, sont engagés envers leurs collègues et participent activement à l'amélioration des processus de développement.

1.2 Problèmes liés au processus

Malgré les nombreuses améliorations, certains aspects du processus restent problématiques.

1.2.1 Les tests de non-régression manuels

Le cœur du problème réside dans le fait que les tests de non-régression soient entièrement manuels. La principale conséquence est la longue durée d'exécution de ces tests. Tel que vu en 1.1.3, l'exécution nécessite généralement cinq sprints (2 mois 1/2)

pour un projet de 13 sprints (6 mois 1/2). Nous reviendrons sur les conséquences de cette longue durée en 1.2.2.

Deux autres conséquences importantes sont la gestion de plus en plus complexe des tests manuels et l'augmentation de la durée consacrée à la priorisation et la sélection des sous-ensembles de tests à exécuter.

Gestion des tests manuels

Les tests manuels sont difficiles à gérer, il est difficile d'avoir des tests complètement distincts, plusieurs scénarios s'entrecoupant entre les tests, créant des doublons et/ou allongeant inutilement chaque test. Au fur et à mesure que le nombre de tests augmente, il devient de plus en plus évident qu'il y a une perte de contrôle au niveau de la gestion des tests puisque tel que vu en 1.1.5, sur un peu moins de 6000 tests, il n'est possible d'en revisiter qu'autour de 850 et d'en exécuter la moitié.

Les tests étant écrit en langage naturel, ils comportent souvent des spécificités liées au testeur l'ayant écrit et/ou des ambiguïtés les rendant complexes et difficiles à comprendre. Combiné au fait que ces tests comprennent de nombreuses étapes (entre 2 et 22), la courbe d'apprentissage des nouveaux testeurs est lente et difficile ; la majorité des nouveaux testeurs ne sont pas autonomes avant 2 ou 3 mois.

Cela rend aussi l'estimation de l'effort d'exécution assez difficile, au point ou l'équipe ayant tenté d'estimer les tests en *story points* a dû abandonner le projet, évaluant que l'effort d'estimation était plus grand que le bénéfice engendré. Cependant, plusieurs tests possèdent toujours une valeur estimée en *story points* qui pourra servir afin de prioriser les tests à automatiser.

Temps consacré à la sélection et à la priorisation des tests

Tel que vu en 1.1.5, l'équipe de testeurs utilise à l'heure actuelle deux stratégies afin de pouvoir passer au travers de la période de non-régression : la « sélection de tests » et la priorisation de tests.

Bien qu'il s'agisse d'excellentes stratégies vu les circonstances, l'approche demeure néanmoins problématique. Premièrement, la durée consacrée à ces activités augmente de façon linéaire au fur et à mesure que le nombre de tests augmente. Deuxièmement, bien que le retrait (sélection d'un sous-ensemble) de tests de l'ensemble à exécuter raccourcisse la période de non-régression, sa contrepartie est la diminution de la capacité à trouver des régressions (Jiang *et al.*, 2009). Cette technique reste un compromis.

1.2.2 Longue période de tests de non-régression

De par sa durée et son décalage dans le temps, la durée de la période de non-régression entraîne plusieurs conséquences négatives.

Gel des demandes de changement

Le fait de regrouper l'ensemble des tests de non-régression en fin de projet force un *feature freeze*. Tel que décrit en Bien que la majorité des projets de l'organisation utilisent SCRUM, certains projets tels le projet SWAT utilise plutôt le mode KANBAN dû à la haute fréquence de déplacement des priorités entre autres.

Les prochaines sections décriront les processus reliés au développement de l'équipe CORE. Comme il pourra être observé, la plupart des processus sont maintenant agiles. Par contre, certains sont toujours teintés par le modèle de développement en cascades.

1.1.3, lorsqu'arrive cette étape, plus aucune fonctionnalité ne peut être développée, ce jusqu'à ce que le produit soit livré. Hors, cela équivaut à geler les demandes de changements pendant deux mois et demi. Ce point est l'irritant majeur puisqu'il retarde également tout nouveau développement pouvant être demandé pour la signature de nouveaux contrats clients.

Déséquilibre de la charge de travail entre les développeurs et les testeurs

Généralement, la période de tests de non-régression n'est pas assez riche en bogues pour occuper les développeurs à temps plein. Ceux-ci commencent alors généralement à coder de nouvelles fonctionnalités pour le nouveau *release* alors que les testeurs sont toujours en train de tester l'ancienne version. Cela crée un déséquilibre énorme de travail entre les développeurs et testeurs, ces derniers se retrouvant à commencer les tests du prochain *release* avec un important retard et bloquant la complétion des sprints.

Augmentation de la probabilité de régressions

À chaque fois qu'une correction est faite, cela augmente le risque qu'une nouvelle régression soit introduite. Comme les tests sont longs à exécuter, l'équipe n'a pas le luxe dans le temps qui lui est imparti de ré-effectuer l'ensemble des tests après chaque correction et doit faire une sélection de tests à exécuter. Cette sélection est établie en fonction des modules touchés par la correction, des modules qui seraient potentiellement affectés et grâce à l'instinct des membres les plus expérimentés de l'équipe de contrôle de la qualité. Malheureusement, malgré les nobles intentions de

l'équipe, ceci n'est pas suffisant et des régressions sont décelées en production dans la majorité voire la totalité des livraisons.

1.2.3 Autres constats

Adéquation aux principes agiles

L'organisation n'adhère pas au premier principe agile étant de livrer rapidement et régulièrement de nouvelles fonctionnalités à valeur ajoutée au client (Beck *et al.*, © 2001). Ce n'est pas non plus en adéquation avec le cadre Scrum mis en place par l'organisation ; cadre prescrivant qu'au terme de chaque sprint il devrait être possible de livrer un incrément de produit (Schwaber et Sutherland, 2016).

Production de *releases* intermédiaires

Afin de pouvoir livrer plus rapidement le produit à certains clients, l'organisation procède à la livraison de « *releases* intermédiaires ». Il s'agit d'une décision, basée sur la valeur d'affaire, prise en cours de projet de livrer une partie des fonctionnalités à une date antérieure à la date de livraison prévue ; contournant ainsi le processus de livraison complet. Pour résumer, une date est fixée, la branche intermédiaire est créée, une validation raccourcie (*package validation*) est effectuée puis le *package* est déployé sur le portail et accessible au client en ayant fait la demande.

La principale conséquence négative de ce procédé est l'augmentation de la probabilité de déceler des régressions en production étant donné une couverture de test largement diminuée. Les conséquences d'un tel risque sont généralement coûteuses ; le double du coût de correction par rapport au coût en phase de test (April et Laporte, 2011) ainsi

que de fâcheuses conséquences sur la réputation de l'entreprise (voir appendice C pour voir le coût moyen de correction par phase de développement dans l'industrie).

Couverture de test non connue

Présentement, la couverture de test n'est pas du tout connue. Il serait toujours possible de le faire en tentant de relier chaque test aux fonctionnalités qu'il couvre, cependant, il s'agirait d'un travail long et fastidieux dont l'effort dépasserait largement le bénéfice de la mesure. Par contre, tel que vu en 1.1.5, il peut être conclu que la couverture de test diminue de *releases* en *releases*.

1.3 Historique d'automatisation

1.3.1 Le cavalier solitaire

Une première tentative d'automatisation a été faite en 2015. L'organisation avait embauché une personne, appelons-la Robert, expert en automatisation, afin de réaliser le projet. Le projet fût considéré comme un échec.

Robert avait beaucoup d'expérience, il avait mis en place des revues systématiques afin de démontrer l'avancement du projet et d'offrir de la visibilité sur ce qu'il faisait.

Les membres de l'organisation ne s'intéressaient cependant pas au projet d'automatisation. L'équipe travaillait dur pour livrer une *release* comprenant plusieurs autres problèmes et c'était pour eux la priorité à ce moment-là. Ils ne se sentaient pas nécessairement concernés, encore moins impliqués ou responsables du projet d'automatisation.

De plus, aucune formation, ni aucun accompagnement n'a été offert à l'équipe afin de faire la transition vers les tests automatisés. Donc, lorsque Robert a terminé son mandat, les connaissances ne se sont pas transmises et le projet n'a pas été repris en main à l'interne.

1.3.2 Déploiement d'une équipe d'automatisation

Après avoir fait la revue de l'état de l'art (voir chapitre 2), j'ai fourni des recommandations à l'organisation et j'ai dû quitter pour un congé de maternité. Il était recommandé de ne pas investir dans des outils coûteux si l'inventaire des tests était un bourbier et de plutôt commencer par assainir l'ensemble des tests. L'objectif étant d'obtenir un rendement rapide afin de savoir si l'initiative serait viable.

Lors de mon absence, l'équipe d'automatisation a acquis des outils coûteux mais attrayants et a commencé à automatiser les tests d'interfaces utilisateurs à l'aide de *Gherkin*, *Cucumber*, *Squish* et *Python 2.7* dont certaines dépendances sont maintenant obsolètes. L'automatisation des interfaces utilisateurs s'est soldée par un échec pour deux raisons. La première, l'investissement n'était pas rentable puisqu'au bout d'une année de travail, le résultat a été une réduction d'environ quatre jours sur la durée de la période de non-régression. La deuxième, dès qu'un changement au niveau de l'authentification a eu lieu, cela a invalidé une partie importante des tests qui devait être refaite.

1.3.3 Ilôts isolés d'automatisation

Certains développeurs et testeurs ont de leur côté tenté d'automatiser certaines parties, soit par des scripts ne pouvant être facilement intégrés dans un cadre de test ou soit en intégrant des tests unitaires améliorant la qualité des composantes mais ayant peu d'impact sur la durée de la période de non-régression. Ces initiatives n'étaient pas

connues de la direction, il s'agissait plutôt d'initiatives personnelles sans vision d'ensemble.

1.4 Problématique

À la lumière des problèmes du processus et de l'historique d'automatisation, le cœur du problème est définitvement lié à la trop longue durée de la période de non-régression. Cette longue durée entraînant, entre autres, un gel des demandes de changements, un déséquilibre de la charge de travail entre les développeurs et les testeurs, une augmentation de la probabilité de régressions, une inadéquation aux principes agiles et une augmentation du risque lié aux *releases* intermédiaires.

Cependant, l'organisation a investi de nombreuses fois pour régler cette problématique mais sans succès. Malgré toutes les tentatives d'automatisation passées, la problématique est toujours présente. Cela indique qu'il y a également une problématique au niveau culturel dans l'organisation et c'est un aspect majeur de ce projet.

1.5 Objectifs

L'objectif ultime de mon projet serait de réduire, voire enrayer, l'effort manuel lié à l'intégration, c.-à-d. la phase de stabilisation, afin que celle-ci ne fasse plus partie de la colonne des contraintes lorsque de nouveaux projets sont envisagés. À long terme, l'objectif serait d'avoir le potentiel de livrer un incrément de produit à la fin de chaque sprint.

Dans le temps prescrit par le projet, mes objectifs sont les suivants :

O1. Réduire la durée de la prochaine période de non-régression

O2. Amener le changement de culture nécessaire à l'acceptation de l'automatisation comme solution

1.6 Approche proposée

Consciente des vestiges du processus cascades et de la difficulté d'intégrer les tests de non-régression dans un sprint, mon approche a été itérative et incrémentale afin de pouvoir régulièrement mesurer l'avancement et améliorer le processus d'atteinte des objectifs. J'ai mis l'accent sur l'économie de temps et d'effort tout en m'assurant le meilleur retour sur investissement possible (ROI).

Pour atteindre les objectifs du projet, j'ai fait une preuve de concept ayant pour principal but de démontrer la faisabilité d'un projet d'automatisation à l'interne avec un minimum d'outils technologiques et afin d'obtenir un ROI rapide. La preuve de concept est expliquée en plus de détails au chapitre 3.

Le projet consiste donc à automatiser une quinzaine de tests de non-régression existants. Les tests ont été priorisés afin d'effectuer les tests les plus bénéfiques à court terme et d'assurer un ROI rapide.

Certains risques pouvant compromettre le succès du projet ont été identifiés ; ils sont décrits dans la prochaine section.

1.6.1 Analyse des risques

<u>Résistance au changement</u>

La résistance au changement est selon moi le principal risque associé à ce projet. Tel que mentionné en 1.1.2, l'organisation a entamé une transition agile en 2014. Depuis,

les développeurs se sont bien adaptés aux nouvelles méthodes mais il semble que les testeurs ne s'y retrouvent toujours pas dans ce nouveau monde. Les testeurs étant les premiers individus à être concernés par l'implémentation de mon projet, il est important de comprendre leur réalité.

Parmi les testeurs, certains sont nouveaux et assez ouverts à de nouvelles pratiques. Par contre certains, ayant vécu les différents échecs d'automatisation passés et ayant déjà été beaucoup secoués par le changement de culture imposé par la direction sont naturellement plus méfiants.

Broadsign ayant pendant longtemps favorisé une culture de contrôle et de compétence, le succès d'un individu était déterminé par son niveau de pouvoir et son niveau hiérarchique. En décidant d'opter pour des méthodes agiles, la culture d'entreprise s'est graduellement mutée en une culture collaborative et d'accomplissement favorisant beaucoup plus les individus que les processus et la hiérarchie. Dans ces cultures, le succès repose plutôt sur l'atteinte d'objectifs d'équipe et sur la capacité d'un individu à s'améliorer.

Il est alors compréhensible que de passer d'une culture à l'autre demande une grande adaptation et est d'autant plus difficile pour les individus qui connaissaient un certain succès auparavant ; ces individus ayant l'impression de perdre leurs acquis et se sentant dévalorisés.

Dans ce contexte, il se crée alors une résistance aux changements qui va parfois jusqu'au sabotage (Boisvert et Trudel, 2011b). Ayant été une actrice principale dans la transition agile, je suis perçue comme menace au statu quo et mes intentions peuvent, et le sont chez ces individus, être mal interprétées et m'exposent à des risques supplémentaires. Il est donc primordial pour moi de bien comprendre ce risque et de trouver une façon de le réduire au maximum.

Manque d'expertise interne

« *...introducing test automation to such staff requires training or runs a high risk of failure (discussed in 8 sources)...* » (Garousi et Mäntylä, 2016)

« *...when competencies are lacking and there is no resources available for the training that is needed to acquire the skills, then it might be better not to automate.* » (Garousi et Mäntylä, 2016)

Tel que mentionné en 1.1.5, aucune expertise en programmation n'est requise actuellement pour être embauché comme testeur. Les testeurs actuels n'ont aucune formation officielle en programmation ni en automatisation.

Manque de temps et de ressource assignées au projet

L'équipe de testeurs subissant déjà beaucoup de pression et manquant de temps pour accomplir ses tâches normales, il y a un réel risque à ce que le projet n'ait pas la bonne allocation de ressources et de temps. Une des cause d'échec répertoriée est d'ailleurs d'assigner le projet à quelques personnes et leur demander de travailler sur le projet à leur rythme et à temps perdu (Pettichord, 1996). Le succès d'un projet d'automatisation des tests nécessite la mise en place d'une équipe multidisciplinaire dédiée (voir 2.2.6).

Peu ou pas d'information sur la couverture de test actuelle

Le fait de ne pas connaître la couverture de test peut mener l'équipe à concentrer l'effort d'automatisation au mauvais endroit. L'équipe peut se retrouver à répéter certains tests continuellement en omettant certains autres tests importants (Persson et Yilmazturk,

2004), cela peut mettre en danger la qualité du produit et faire échouer le projet d'automatisation.

Complexité des tests manuels actuels

Les tests actuels sont complexes et difficiles à comprendre, le testeur doit souvent se référer aux notes manuelles de ses collègues ou carrément aller les voir. Dans le cas où le testeur auteur du test ne travaille plus pour la compagnie, le nouveau testeur exécutera alors le test sans en comprendre la totalité et parfois exécutera un test qui n'a plus aucune valeur.

Support multiplateformes

L'organisation maintient à l'heure actuelle, le lecteur sur 11 plateformes différentes (Broadsign, 2015). Notamment plusieurs version de Windows et quelques distributions de Linux restreignant les choix d'outils et de librairies et rendant les tests beaucoup plus longs et laborieux.

Design actuel des tests

Les tests ne sont pas basés sur les spécifications telles que fournies par le client mais plutôt par ce que devrait faire la fonctionnalité selon les développeurs.

Ils sont plutôt orientés par la connaissance et l'expérience des individus, ce qu'Amannejad *et al.* (2014) réfèrent au « *exploratory testing* » ; « qui par nature s'effectue manuellement » [traduction].

1.6.2 Résumé

En résumé, mon approche est d'automatiser une quinzaine de tests afin d'obtenir un ensemble permettant de mesurer le ROI.

Il existe plusieurs risques pour la mise à terme de ce projet. Comme dans tous les changements organisationnels, la résistance au changement est l'un des premiers risques. Le manque d'expertise interne, le manque de temps accordé, le manque d'individus assignés au projet, le manque d'information sur la couverture de tests actuelle, la complexité des tests existants, le support de plusieurs plateformes et le design des tests en sont propres à ce projet d'automatisation. Mon approche permet également de faire ressortir ces risques s'il y a lieu.

CHAPITRE II

ÉTAT DE L'ART

En faisant une revue de l'état d'art, je vise à découvrir comment d'autres organisations, vivant des défis similaires, ont évolué. Quels ont été leurs bons coups et leurs moins bons coups, quels ont été leurs défis et comment je peux apprendre de ces expériences afin d'augmenter les chances de succès de mon projet. Je porte également une attention particulière à tout ce qui entoure le changement organisationnel et les façons de contrer les résistances au changement ainsi que les différents facteurs contribuant à la décision d'automatiser ou non et autres facteurs contribuant à la sélection des candidats à automatiser.

2.1 Automatisation des tests

Afin de réduire la durée de la période de tests de régression et d'augmenter la couverture de test, la technique la plus courante dans les dernières années est l'automatisation des tests de régression. Les tests de régression étant très répétitifs, leur automatisation permet une réduction significative du temps et de l'effort investi dans l'ensemble du processus lié aux tests de régression. Les vendeurs d'outils de tests automatisés font la promotion d'une augmentation de la couverture de tests ainsi qu'une augmentation de la fréquence de tests (Ramler et Wolfmaier, 2006). Plusieurs études de cas confirment aussi la réduction de l'effort et l'augmentation de la qualité du produit (Amannejad *et al.*, 2014; Berner *et al.*, 2005; Laapas, 2014) et certains affirment une plus grande réduction des coûts lorsqu'utilisée dans un cadre de développement agile (Laapas, 2014). D'autres vont même jusqu'à avancer que l'automatisation des tests de

régression est pratiquement un préalable au succès d'une pratique de développement incrémental (Persson et Yilmazturk, 2004).

Malgré sa popularité[15], l'automatisation des tests demeure une transformation majeure et nécessite une bonne planification. La stratégie doit être adaptée à chaque contexte (Bach *et al.*, 2002), et être en ligne avec les besoins en tests de l'organisation, l'architecture du produit et l'expertise des testeurs.

La question « Quand automatiser ? » est sans doute une des préoccupations les plus importantes parmi les chercheurs et praticiens face à l'automatisation des tests, suivie de la question « Quoi automatiser ? » qui sera discutée en 2.3.

2.2 Quand automatiser : les facteurs à considérer

Garousi et Mäntylä (2016) ont conduit une revue de littérature systématique dont une partie de leur recherche portait sur les facteurs à considérer afin de décider d'automatiser ou non. Leur revue, couvrant de 1997 à 2014, portait autant sur la littérature scientifique que la littérature grise[16], permettant ainsi de recenser aussi l'expertise des praticiens du milieu, autrement absente dans la littérature scientifique. Les facteurs seront présentés en ordre de popularité quant au nombre de sources sur le sujet, dans l'ordre suivant :

- Besoins en tests régression (2.2.1) ;

---

[15] Entre 2010 et 2014, 110 livres ont été publiés sur les tests logiciels dont 78 mettant l'accent principalement sur l'automatisation des tests et le nombre de publications générales augmente rapidement d'années en années. Vahid Garousi et Mika V Mäntylä, «When and what to automate in software testing? A multi-vocal literature review», *Information and Software Technology* 76(2016).

[16] Traduction libre de *grey litterature* signifiant tout autre littérature que celle scientifique dont les *white paper*, les articles de blogue et articles non publiés ou n'ayant pas subi de processus de revue formelle par les pairs.

- Facteurs économiques (2.2.2) ;
- Maturité du système à tester (2.2.3) ;
- Durée de vie du système (2.2.4) ;
- Maturité de l'entreprise (2.2.5) ;
- Compétences des testeurs (2.2.6) ;
- Types de tests (2.2.7).

2.2.1 Besoin en tests de régression

Comme mentionné en introduction de chapitre, les tests de régression étant répétitifs, leur automatisation est la plus bénéfique par rapport aux autres types de tests. Cependant, le besoin peut varier d'une organisation à l'autre, les entreprises passant beaucoup de temps à exécuter des tests de régression devraient davantage considérer l'automatisation.

2.2.2 Facteurs économiques

L'automatisation des tests étant un investissement important, évaluer la rentabilité permettra de justifier ou non la décision d'automatiser. Une méthode couramment utilisée est le ROI, consistant à diviser les bénéfices soustraits des coûts par les coûts (voir équation 2.1).

$$\text{ROI} = \frac{bénéfices - coûts}{coûts} \tag{2.1}$$

Cependant, bien qu'évaluer les coûts soit relativement facile (voir Tableau 2.1), cela est plus difficile pour les bénéfices, ceux-ci étant pour la plupart intangibles et/ou difficiles à mesurer. Quelques exemples démontrent aussi que certains coûts sont oubliés de l'équation, par exemple, le coût d'abandonner un test qui brise plutôt que de

le corriger (Ramler et Wolfmaier, 2006), malgré tout, le modèle suivant permet de comparer les coûts liés aux tests manuels versus aux tests automatisés.

Tableau 2.1 Évaluation des coûts (tests manuels vs automatisés)

$Coûts\ des\ tests\ automatisés$ $= Coût\ des\ licences + Coût\ du\ matériel$ $+ Durée\ de\ développement\ des\ scripts\ automatisés$ $+ Durée\ de\ maintenance * n + Durée\ d'exécution * n$ $Coûts\ des\ tests\ manuels$ $= Durée\ de\ développement\ des\ cas\ de\ test$ $+ Durée\ de\ maintenance * n + Durée\ d'exécution * n$ Où n représente le nombre de tests dans la suite

Plusieurs calculent le ROI uniquement en comparant le coût des tests manuels et des tests automatisés dans le temps ; un modèle largement utilisé est celui illustré à la Figure 2.1.

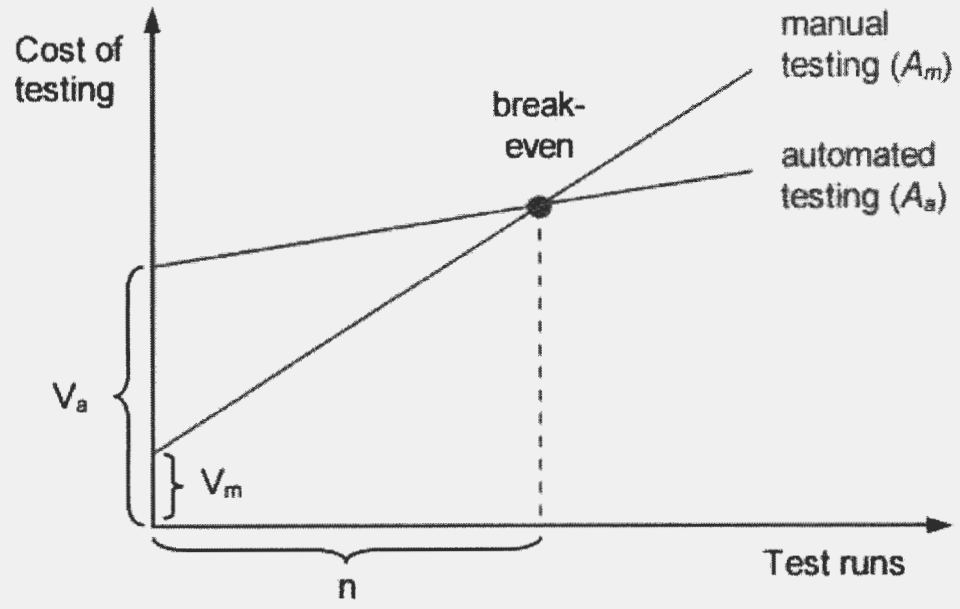


Figure 2.1 Point de rentabilité pour l'automatisation de test

Tiré de Ramler, R. et Wolfmaier, K. (2006). Economic perspectives in test automation: balancing automated and manual testing with opportunity cost. International Conference on Software Engineering (p. 85-91). Shanghai : ACM

Le graphique illustre qu'il existe, pour une suite de tests donnée, un nombre d'exécutions *n* devant être dépassé pour que l'automatisation soit considérée rentable ; *V* représentant l'ensemble des coûts tels qu'illustrés dans le tableau 2.1.

Selon la littérature, *n* se situerait généralement entre 2 et 20 dépendamment de différents facteurs. Certains praticiens de chez Microsoft affirment que le nombre se situerait généralement entre 15 et 17 (Garousi et Mäntylä, 2016). Essentiellement, en se basant uniquement sur les coûts, l'implémentation initiale de tests automatisés est plus dispendieuse que celle des tests manuels, cependant, cela demeure un investissement rentable à moyen terme : « *Introduction of test automation often increases cost for creating tests, however the cost of re-running a test decreases* » (Garousi et Mäntylä, 2016).

Le problème principal avec la méthode d'évaluation vue en Figure 2.1 est qu'elle est uniquement basée sur les coûts, « une bonne analyse de rentabilité devrait se baser à la fois sur les coûts et sur les bénéfices » [traduction libre] (Ramler et Wolfmaier, 2006). Elle ne permet pas non plus de s'intégrer à la réalité des organisations pour qui l'automatisation n'est pas entière mais réside plutôt dans un compromis entre les tests manuels et automatisés. Dans cette réalité, l'automatisation d'un test engendre généralement l'abandon d'un ou plusieurs tests manuels.

Le but du projet n'étant pas d'effectuer une analyse de rentabilité complète, j'ai relevé les principaux bénéfices qui pourront être utilisés pour une analyse future. Dans le cas d'une entreprise développant en modèle agile, le bénéfice principal est sans aucun doute de réduire le temps du cycle de développement et de pouvoir livrer plus rapidement.

Tableau 2.2 Bénéfices liés à l'automatisation des tests

Bénéfice	**Justification**
Amélioration de la qualité et de la profondeur des tests.	Les testeurs étant dispensés d'exécuter des tâches manuelles répétitives ont plus de temps pour améliorer les tests existants et pour en concevoir des meilleurs (Berner *et al.*, 2005).
Détection de bogue plus précoce	L'automatisation permet d'exécuter une suite de test immédiatement après l'introduction d'un changement dans le code, permettant ainsi de pouvoir détecter une anomalie dès son entrée (Laapas, 2014).
Accélération du processus de développement, réduction du temps de cycle	Allant de pair avec l'avantage précédent, sachant que plus un bogue est découvert tard, plus le temps de correction est long, sachant aussi que l'automatisation accélère certaines phases de test, le temps de cycle est réduit (Laapas, 2014). Yang (2016) présente plus ou moins le même avantage comme étant un gain de temps et d'argent.

Bénéfice	Justification
Feedback plus rapide aux développeurs (Alégroth *et al.*, 2013; Laapas, 2014) :	Alegroth et al. ont mené un sondage révélant que l'augmentation de la fréquence d'exécution des tests de régression fut très bénéfique pour les développeurs, étant donné un *feedback* beaucoup plus rapide.
Possibilité de tester le logiciel d'une façon que les tests manuels ne permettait pas (Hoffman, 1999; Laapas, 2014)	Les tests de performance et tests de charge ne peuvent être exécutés manuellement, cependant, ils apportent beaucoup d'information et sont bénéfiques pour assurer la qualité du produit.
Exécutions hors des heures de bureau	Les tests manuels ne peuvent être exécutés que lorsqu'un testeur est disponible. Les tests automatisés peuvent être roulés à n'importe quel moment même lorsqu'aucun testeur n'est disponible.
Augmentation de la qualité du produit (Stobie, 2009; Yang, 2016) ;	Un des bénéfices importants est une meilleure qualité pour le même coût.
Visibilité sur le logiciel augmentée (Hoffman, 1999) ;	L'automatisation des tests permet d'avoir une meilleure vue du logiciel, elle permet d'examiner celui-ci de plus près et sur un nouvel angle.
Augmentation du niveau de compétence et de professionnalisme des testeurs	L'automatisation demande une expertise complètement différente. Les testeurs passant d'un paradigme à l'autre augmentent inévitablement leur niveau de compétence.

2.2.3 Maturité du système à tester (*SUT*[17])

Un système immature et dont les fonctionnalités sont constamment ré-implémentées ou subissant énormément de changements crée un impact négatif majeur sur les tests automatisés. L'impact est dû au fait que les tests sont constamment brisés, le nombre de faux-positifs augmente et donc l'effort de maintenance et d'analyse des résultats

[17] *System Under Test*

augmente jusqu'à dépasser le bénéfice engendré par l'automatisation. Il est aussi important de prédire le nombre de changements à venir afin de ne pas être constamment en train de réécrire les tests.

2.2.4 Durée de vie du système

La durée de vie du système doit être considérée lorsque l'on fait le choix d'automatiser ou non. Plus un système a une longue durée de vie et plus l'automatisation devient une solution efficace [traduction libre] (Garousi et Mäntylä, 2016).

2.2.5 Maturité de l'entreprise

Il peut être dangereux d'essayer d'introduire un processus d'automatisation des tests dans une entreprise n'ayant pas la maturité suffisante, le risque étant de tomber dans un état de chaos plus élevé qu'avant l'introduction du processus (Persson et Yilmazturk, 2004).

2.2.6 Niveau de compétence des testeurs

L'automatisation demandera de nouvelles connaissances et compétences de la part des testeurs ; « le *scripting*, la programmation et les outils d'automatisation » [traduction libre] (Yang, 2016) en sont quelques exemples.

Le niveau de compétence et de productivité individuelle est important puisqu'il peut influencer grandement le coût de l'automatisation voir même le possible bénéfice de celle-ci. Des études ont démontré un facteur de 10 pour 1 au niveau de la productivité individuelle, c.-à-d. qu'un programmeur compétent peut en 1 heure développer autant de tests qu'un développeur moins productif le peut en 10 heures (Garousi et Mäntylä,

2016). Autre que le temps de conception, la qualité de conception des tests est tout aussi importante que pour n'importe quelle autre conception logicielle. Une bonne architecture favorisera la maintenabilité des tests, alors qu'une mauvaise entraînera inévitablement des coûts de maintenance plus élevés.

Il existe, comme pour le développement de fonctionnalités, certains « patrons d'automatisation de test » [traduction libre] (Graham, 2013) permettant d'orienter la conception en vue de favoriser des qualités architecturales telles que la maintenabilité, il pourrait être intéressant de les regarder.

Dans tout changement de processus de développement, on recommande de passer par une preuve de concept (POC) pour augmenter le niveau de compétences des individus, leur niveau de confiance et faire ressurgir les risques prématurément.

La POC doit permettre d'explorer des façons de faire et illustrer ce qui va bien et ce qui va moins bien en expérimentant afin que l'équipe puisse faire des choix éclairés par la suite.

Une autre manière d'adresser le problème est « d'assigner un *coach* qui sera en mesure de bien sensibiliser les membres de l'équipe sur l'importance des processus d'automatisation » [traduction libre] (Persson et Yilmazturk, 2004).

Une autre approche souvent envisagée par les décideurs est de confier le mandat à une équipe externe, une approche qui toujours selon Persson et Yilmazturk (2004), conduit généralement à l'échec. L'équipe doit pouvoir, tout au long de la mise en place du projet, augmenter son niveau de compétence et de savoir, afin que le projet puisse se maintenir au-delà de la mise en place initiale par des ressources externes.

Persson et Yilmazturk (2004) énoncent aussi une mise en garde quant à la formation d'une équipe d'automatisation constituée uniquement de testeurs et de développeurs : « *Forming a project with only testing or programming experience leads the project to*

a failure ». Ils recommandent plutôt de constituer « une équipe équilibrée, regroupant à la fois des testeurs, des programmeurs, des gestionnaires de projet et des ressources spécialisées en base de données, réseaux, systèmes d'opération, matériel, etc. » [traduction libre].

Il peut parfois aussi être nécessaire d'embaucher de nouveaux testeurs (Yang, 2016).

2.2.7 Types de tests

Les tests de régression peuvent aussi varier de par leur nature. Les tests de charge et de performance sont de très bons candidats à l'automatisation étant même souvent impossibles à exécuter manuellement (Garousi et Mäntylä, 2016). Les tests devant être réutilisés plusieurs fois, par exemple une authentification systématique avant de pouvoir tester une application web. Un nombre d'environnements élevés à tester où les mêmes tests seront exécutés sur plusieurs plateformes contribue aussi à augmenter la réutilisation et auront avantage à être automatisés. Plusieurs autres critères permettent aussi d'évaluer quels tests devraient être automatisés, ces critères seront discutés en 2.3.

2.2.8 Résumé

En résumé, la question du « quand automatiser » doit se poser en tant qu'organisation, il n'est pas vrai que l'automatisation améliore toujours les choses, il existe différents facteurs afin d'aider à la prise de décision. Ses facteurs se résument ainsi : l'analyse des besoins en tests de régressions, les facteurs économiques, la maturité du système à tester, la durée de vie du système, la maturité de l'entreprise, la compétence des testeurs et les types de tests.

2.3 Quoi automatiser : les différentes approches

Lorsque la décision d'automatiser un ensemble de tests manuels est prise, il reste à déterminer l'approche : Doit-on automatiser l'ensemble des tests ? Quels tests devraient être automatisés en premier ?

Les recherches tendent à démontrer qu'une automatisation complète des tests manuels est rarement souhaitable et même impossible (Hoffman, 1999; Software, 2007). On préconise plutôt d'aborder l'automatisation comme un complément aux tests manuels existants et de faire une priorisation des tests à convertir (Alégroth *et al.*, 2013) ou une priorisation des différentes étapes d'un test (Amannejad *et al.*, 2014). D'autres études démontrent aussi qu'il n'est pas toujours bénéfique d'automatiser un test manuel et proposent des facteurs à considérer avant d'en faire le choix. Dans l'ensemble, nul ne recommande une automatisation complète. Il est aussi à noter que l'automatisation ne permet pas de faire des tests exploratoires devant par nature être manuels.

2.3.1 Créer des tests automatisés basés sur la couverture

On recommande pour les tests unitaires de prioriser les fonctions les plus grandes, celles qui sont le plus souvent modifiées et celles dont le nombre de corrections est le plus élevé (Shihab *et al.*, 2011). On peut faire le parallèle avec les tests fonctionnels, les tests qui valident le fonctionnement des modules les plus souvent modifiés ou pour lesquels il y aurait le plus de corrections faites seraient alors ceux à prioriser. Cela va d'une certaine façon de pair avec les tests les plus souvent exécutés puisqu'ils seraient sélectionnés pour exécution, entres autres facteurs, lorsque l'on croit que le module

qu'il teste pourrait être affecté. Cette approche nécessite cependant de connaître la couverture de test existante.

2.3.2 Automatiser les tests manuels existants

Une autre approche, qui est par ailleurs bien documentée dans les études de cas, est l'automatisation des tests manuels existants. Dans cette stratégie, la question est se pose à savoir quels tests doivent être automatisés, comment et avec quelle priorité afin d'éviter de mettre des efforts sur des tests ayant peu ou moins de valeur.

> *"If you do not know which tests are the most important and which tests are the most applicable for automation, the tool will only help perform a bad test faster."(Rice et al., 2003)*

Ayant essayé cette approche d'automatisation stricte chez Saab AB, une compagnie dont les principaux tests validaient l'interface graphique, Alegroth et al. ont conclus que ce n'était pas la meilleure approche (2013). La première raison est par soucis d'efficacité, car il serait selon eux préférable de regarder uniquement la spécification afin de se concentrer sur le « quoi » tester et non sur le « comment ». D'autre part, comme l'automatisation vise à améliorer la vitesse d'exécution, « regrouper plusieurs actions ensemble » [traduction libre] permettrait plus facilement d'atteindre cet objectif.

Les pièges

Persson et Yilmazturk (2004) ont fait un recensement parmi la littérature d'une trentaine de pièges fréquents dans un processus d'automatisation des tests.

Tableau 2.3 Quelques pièges de l'automatisation

Numéro	Description
P1	*Uncontrolled introduction of test automation in immature organizations -"automated chaos yields faster chaos"*
P3	*Introducing too many test cases at once -defect overflow is extremely dangerous*
P8	*Defensive attitudes towards automated testing. Management is not committed to initiate introduction and establishment of automated testing*
P9	Poor or unknown test coverage of automated regression tests
P11	*The testware is not handled with the same care and professionalism as the shipped code - ignoring that test automation is software development*
P12	*Capture and Replay functionality of test tools encourage bad testware architecture by default. Test scripts are easily "hard coded" due to wrong use of "capture & play" features. This makes the test scripts sensitive to changes in user interface*
P23	*Reports produced by the tool might be useless, i.e. do not present the information needed by the organization.*
P29	*Lack of controlling the decisions to automate or not. To automate wrong can create costs without any benefits.*

Tiré de Society, I. C. (dir.). (2004). Establishment of automated regression testing at ABB: industrial experience report on avoiding the pitfalls'. 19th IEEE international conference on Automated software engineering.

Suite à leur recensement, Persson et Yilmazturk (2004) recommandent pour éviter ces pièges d'avoir « une excellente gestion de risque dont la principale source d'information serait la liste des pièges répertoriés ».

2.3.3 Facteurs pour la sélection et la priorisation des tests à automatiser

Plusieurs chercheurs et praticiens ont proposé des facteurs afin de prioriser les tests à automatiser qui seront présentés dans les prochaines sous-sections. Je les ai catégorisés comme suit :

- Fréquence d'exécution du test (2.3.3.1) ;
- Effort d'exécution et d'automatisation (2.3.3.2) ;
- Bénéfice du test (2.3.3.3) ;
- Nature du test (2.3.3.4) ;
- Effort de maintenance du test (2.3.3.5) ;

2.3.3.1 Fréquence d'exécution du test

Lorsque vient le temps de vérifier la fréquence d'exécution d'un test, il y a deux facteurs à considérer : la fréquence d'exécution passée et la fréquence d'exécution future.

Le nombre d'exécutions passées peut servir à illustrer l'importance d'un test et à en déduire sa priorité. Par contre, un test exécuté souvent peut aussi dénoter sa simplicité d'exécution : un test simple et rapide à exécuter aura moins tendance à être évincé de la *run* d'exécution.

La fréquence d'exécution future est quant à elle une estimation : il faut se demander s'il sera exécuté souvent ou sporadiquement ; dans le dernier cas, l'automatisation pourrait être plus coûteuse que les bénéfices associés. « Connaître la durée de vie de l'application et avoir une vue à haut niveau sur les changements à venir » [traduction libre] est d'ailleurs, selon Sampat et Jamwal (2007) un prérequis essentiel à la décision d'automatiser ou non. Néanmoins, à chaque nouvelle exécution d'un test automatisé, la rentabilité augmente (Berner *et al.*, 2005)

La fréquence d'exécution n'est qu'un facteur parmi d'autres, à elle seule elle ne permet pas de déterminer la valeur d'un test (Kaner et Fiedler, 2013). En résumé, il y aura bénéfice à automatiser les tests qui seront les plus souvent exécutés à condition que ceux-ci aient de la valeur, valeur étant mesurée indépendamment de la fréquence d'exécution.

2.3.3.2 Effort d'exécution et d'automatisation

Aranha et Borba (2007) prônent que les meilleurs tests à automatiser sont ceux qui nécessitent le plus d'effort d'exécution manuel et le moins d'effort à automatiser. Sur la même idée, Yang (2016) onsidère la complexité et la difficulté d'exécution comme un des incitatifs importants à l'automatisation d'un test.

En gros, les tests demandant le plus grand effort d'exécution manuel et le plus petit effort d'automatisation devraient être automatisés en premier (voir figure 2.2).

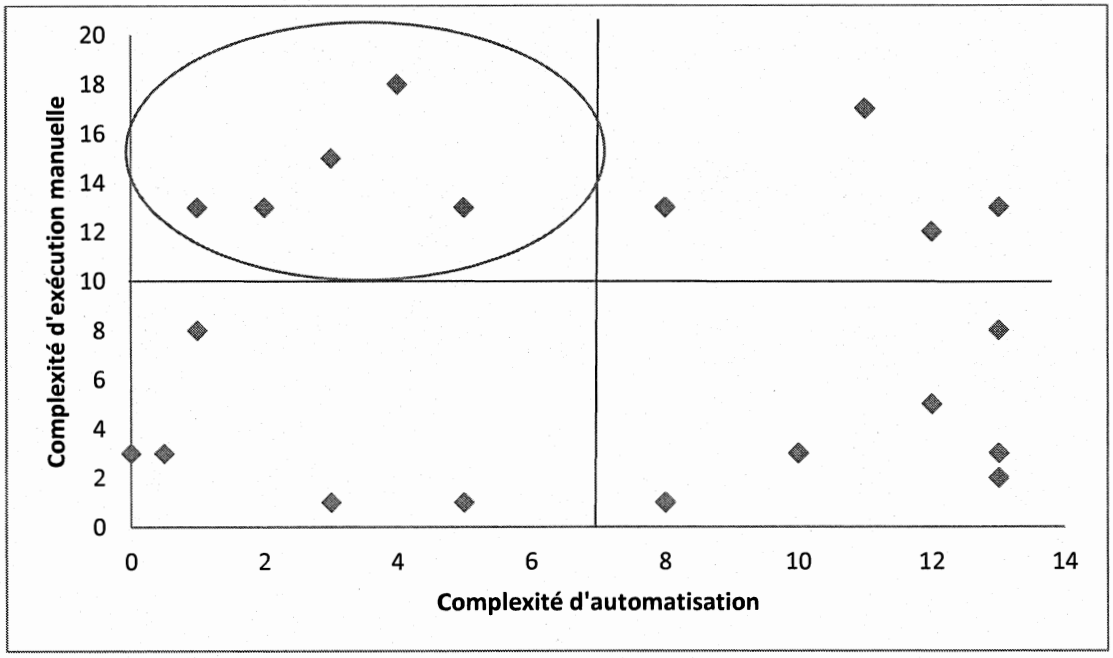


Figure 2.2 Répartition fictive de la complexité d'exécution de tests manuels en fonction de leur complexité d'automatisation

La figure 2.2 illustre des tests manuels répartis en tenant compte de leur complexité d'exécution (axe vertical), valeur pouvant se chiffrer en effort d'exécution (voir ci-après) et de leur complexité d'automatisation (axe horizontal) en effort d'automatisation. Les tests encerclés sont ceux qui devraient être automatisés en premier afin d'augmenter la rentabilité de l'automatisation.

L'effort d'exécution des tests manuels peut être obtenu de diverses façons. Aranha et Borba (2007) proposent d'assigner à chacun des tests un nombre de « points d'exécution » (*ep*[18]) représentant la taille et la complexité d'exécution d'un test. Dans ce système, un test de 1000 *ep* représenterait le double de l'effort par rapport à un test de 500 *ep*. Bien que cette technique ait certains avantages, un tel pointage n'a cependant aucune valeur du point de vue métrologique. Le parallèle peut être fait avec les *story*

[18] *Execution point.*

points pour lesquels il a été démontré qu'il existe plusieurs faiblesses. Une des principales étant qu'ils ne peuvent être normalisés, c.-à-d. que d'une équipe à une autre, même dans la même organisation, un pointage identique ne signifie pas le même niveau d'effort.

L'évaluation de l'effort pourra se baser sur des facteurs tels que le nombre de plateformes à tester, le nombre de matériels à tester, la clarté (ou l'existence) des spécifications, etc.

2.3.3.3 Bénéfice du test

Selon Ramler et Wolfmaier (2006), le bénéfice apporté par un test manuel est différent de celui apporté par un test automatisé. Par exemple, un test automatisé sera plus performant à déceler des régressions tandis qu'un test manuel permettra davantage la découverte de nouvelles façons de briser une fonctionnalité.

Un des bénéfices communs est la mitigation du risque. Selon Ramler et Wolfmaier (2006), il se calcule en terme d'exposition au risque. Plus la probabilité de perdre du temps est grande, et plus cette perte de temps est grande, alors plus le bénéfice du test est grand.

Un autre facteur pouvant être classé comme bénéfice est la contribution à la couverture de test mais qui n'est malheureusement pas toujours connue.

2.3.3.4 Nature du test

Tel que vu en 2.2.1, les tests pour lesquels le résultat varie très peu d'une version à l'autre et devant être exécutés fréquemment, tels les tests de régression, ont des coûts

de maintenance bas et permettent de détecter à moindre effort si un comportement a changé d'une version à l'autre.

Les tests de performance sont aussi des bons candidats à l'automatisation. Devant généralement se répéter plusieurs fois dans un cadre de temps, il peut être très coûteux de les effectuer manuellement. Aussi, certains tests devant valider des opérations atomiques, une requête par exemple, ne sont tout simplement pas possible dans un processus manuel. En réalité tout test prenant moins d'une seconde à s'exécuter est pratiquement impossible à valider par un être humain et est sujet à automatisation.

Les tests de stress et de charge tirent aussi bénéfice à être automatisés.

Cependant, Stobie (2009) rapporte que l'analyse des résultats peut être fastidieuse et que le compromis réside dans l'automatisation de la génération de la charge et du stress combinée à des tests d'exploration ne pouvant être automatisés par nature.

2.3.3.5 Effort de maintenance

L'effort de maintenance est important puisqu'il peut à lui seul éliminer tout le bénéfice de l'automatisation. Un des facteurs à considérer est le taux de changement du code testé, celui-ci augmentant directement le coût de maintenance (Stobie, 2009).

Il y a un danger par contre à prioriser ce facteur au dépend de la qualité d'un test. Par exemple, « une équipe pourrait décider d'automatiser des interactions avec l'interface en utilisant les raccourcis claviers plutôt que la souris, par souci de maintenabilité, mais passer complètement à côté de la valeur si les utilisateurs n'utilisent que la souris » [traduction libre] (Stobie, 2009).

L'autre point qui, selon moi, doit être considéré est que de prioriser les tests en fonction de ce facteur est contradictoire avec le fait de prioriser les tests qui valident les modules qui sont le plus souvent modifiés. Cette évaluation peut devenir alors assez difficile.

2.4 Alternatives à l'automatisation

Cette section discute de scénarios alternatifs à l'automatisation et de leurs possibles impacts.

2.4.1 Statu quo

La première possibilité est de ne prendre aucune décision et de continuer de la même façon que présentement, c.-à-d. de conserver le même nombre de testeurs qui continueront d'exécuter les tests manuellement.

Une des conséquences bien documentée est l'augmentation du risque au fur et à mesure que le nombre de nouvelles fonctionnalités augmente. De *releases* en *releases*, de nouvelles fonctionnalités sont ajoutées et de nouveaux tests de régression doivent être ajoutés. Par contre, étant donné un nombre de testeurs constant, la capacité à effectuer l'ensemble des tests diminue.

Le premier impact est généralement l'allongement de la période de régression, et de *releases* en *releases* elle est de plus en plus longue.

Un deuxième impact possible est l'augmentation de la pression sur l'équipe de QA. L'organisation voulant réduire sa durée de mise en marché, une pression sera mise sur l'équipe de testeurs afin que la période de non-régression soit la plus courte possible. L'équipe, pour arriver à entrer dans ces délais, utilisera alors diverses techniques telles

que vues en 1.1.5, ce qui augmentera le risque de déceler des régressions en production (voir Figure 2.3).

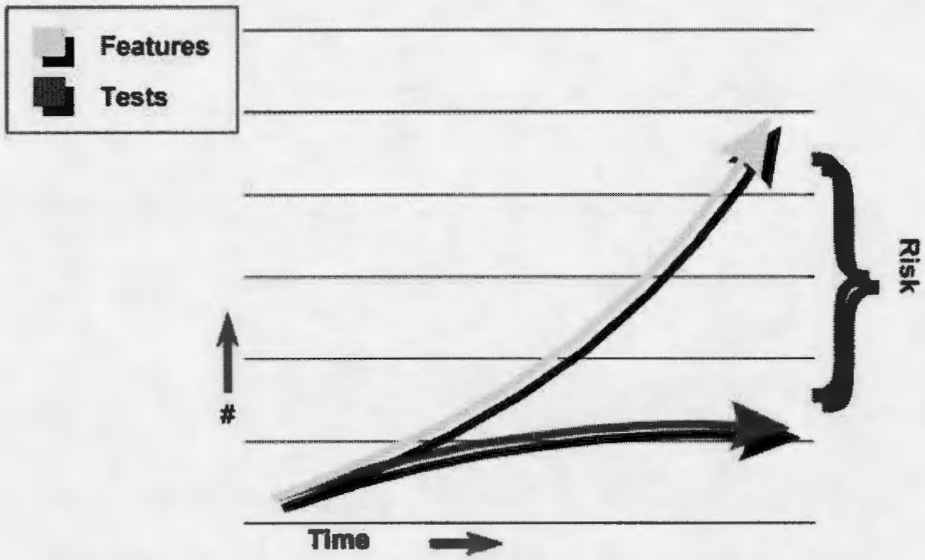


Figure 2.3 Risque associé à un processus de test fautif

Tiré de Hayes, L.G. (2004).

Le graphique illustre l'augmentation du risque associée à un nombre de fonctionnalités augmentant rapidement pendant que le nombre de tests exécutés tend à la baisse.

2.4.2 Augmenter le nombre de testeurs

Une seconde alternative est d'augmenter le nombre de testeurs afin de maintenir la capacité de l'équipe de QA à effectuer l'ensemble des tests de non-régression.

Cela fonctionne généralement à court terme, cependant, il peut être assez difficile d'avoir un nombre doublé ou triplé de testeurs en période de régression uniquement.

Que devraient faire ces testeurs pendant le restant de l'année. On voit rapidement qu'il se pose des enjeux de gestion importants.

D'autres part il n'est pas toujours possible d'engager autant de testeurs qu'on le voudrait, ces ressources étant parfois limitées sur le marché.

2.4.3 Améliorer la testabilité du système

Certaines recherches démontrent qu'en améliorant la testabilité du système, c'est-à-dire la facilité avec laquelle il expose ses bogues, que les coûts reliés aux tests seront réduits (Singh *et al.*, 2013). Cela ne peut cependant pas se réaliser à court terme et pourrait s'avérer difficile à faire, selon les compétences disponibles.

2.5 Technologies

2.5.1 Écriture des tests

Le choix d'un langage « naturel » balisé permet d'établir une norme dans l'écriture des tests (Aranha et Borba, 2007), il permet aussi de réduire les ambiguïtés et d'éliminer l'effort d'interprétation, augmentant par le fait même la maintenabilité des tests. L'écriture de tests devient aussi plus facile, elle s'accompagne généralement d'un modèle à suivre, permettant aussi la réutilisabilité (Grabinski et O'Hare, 2014). Elle devient aussi accessible aux autres parties prenantes telles que les analystes d'affaires, *product owners*, clients, généralement tout ceux étant impliqués de près dans la gestion des exigences.

Le moment de l'écriture des tests est aussi important ; l'idée ici est de migrer vers un processus unique mariant à la fois le développement et les tests, où la collaboration

étroite entre développeurs et testeurs est de mise et qui serait fait antérieurement au développement.

Les tests devraient être écrits avant que la fonctionnalité ne soit développée. Le développeur peut alors s'assurer lui-même de l'exactitude de ce qu'il développe. Cela réduit grandement le nombre d'échanges entre le développement et le QA et diminue ainsi le temps de développement.

2.5.2 Automatisation des tests

2.5.2.1 Automatisation des tests d'interface graphique

Plus une interface change souvent et plus les coûts de maintenance des tests automatisés seront élevés

> *If you automate UI testing early and the UI changes every day in multiple ways, the best case is you keep up with the UI in your automation. The worst case is you quickly fall hopelessly behind. (Stobie, 2009)*

Les tests qui vérifient l'utilisabilité ne peuvent généralement être automatisés (Stobie, 2009) et lorsqu'ils le peuvent, les coûts sont généralement beaucoup plus élevés. Cependant, l'automatisation de tests d'interface a beaucoup évolué dans les dernières années et il est beaucoup plus facile aujourd'hui et de moins en moins coûteux d'automatiser ceux-ci.

2.5.2.2 Choix des outils

Il convient de bien dresser la liste des besoins avant de faire le choix d'un outil. Par contre, il peut être assez difficile de bien connaître les besoins en début de projet, ceux-ci se clarifiant généralement au fur et à mesure que le projet progresse. Ayant un si bas niveau de détail en amont fait souvent glisser une équipe dans le piège de choisir un outil ne supportant pas certaines fonctionnalités. Il faut néanmoins y investir un effort minimal en gardant cela en tête.

Un élément important à considérer avant de faire l'acquisition d'un nouvel outil d'automatisation est de s'assurer d'avoir et de maîtriser son processus de test. Une des leçons transmises par Bach *et al.* (2002) est qu'une stratégie de test confuse mène inévitablement à une augmentation de la confusion lorsque vient le temps d'automatiser.

2.6 Changement organisationnel

L'introduction de l'agilité dans un environnement de développement crée plusieurs changements organisationnels, l'automatisation des tests affectant le plus les testeurs. L'utilisation de nouveaux outils est nécessaire et la technique d'écriture des tests change complètement (Hoffman, 1999). Les testeurs sont amenés à automatiser les tests et à les incorporer dans un nouveau cadre de test, ce qui représente de nouvelles habilités (Coram et Bohner, 2005). Ce changement amène parfois l'organisation à devoir réaffecter les testeurs qui ne cadrent plus dans une nouvelle équipe, donnant toutefois une opportunité aux juniors de redémarrer et de développer une nouvelle expertise (Coram et Bohner, 2005).

2.6.1 Changement de rôle des testeurs

Les testeurs seront amenés à développer de nouvelles habiletés et une nouvelle expertise. Les habilités pour des testeurs manuels étant plutôt l'intuition et les observations passées (Stobie, 2009), ils devront dorénavant développer des tests d'une manière différente.

En introduisant une méthode de test plus agile, les testeurs devront aussi devenir partie intégrante de l'équipe de développement et non plus une unité fonctionnelle à part, ce qui demandera une adaptation à de nouveaux rôles (Crispin et Gregory, 2009).

2.6.2 Gérer la résistance au changement

La peur est généralement responsable de l'insuccès d'une transition organisationnelle. C'est un élément que l'on doit adresser rapidement afin que la transition ne soit pas paralysée par de la résistance au changement (Crispin et Gregory, 2009).

Il faut s'assurer de comprendre et dissiper les peurs, une des peurs les plus répandues étant celle de perdre son emploi ou de ne pas pouvoir développer de nouvelles compétences. Les testeurs seront aussi amenés à sortir de leur zone de confort et l'on doit leur assurer le soutien et l'accompagnement nécessaires afin qu'ils puissent réussir.

Crispin et Gregory (2009) parlent aussi de « barrières à l'adoption de l'Agilité dans les équipes de QA » : la perte d'identité, l'ajout de nouveaux rôles, le manque de formation, l'incompréhension des concepts agiles et les expériences passées en sont des exemples.

Pour contrer ces barrières, ils proposent d'abord d'informer les différents acteurs concernés sur les étapes d'un changement, « d'illustrer le modèle du changement[19] » pour éviter que l'équipe ne confonde l'étape du chaos avec le nouveau statu quo.

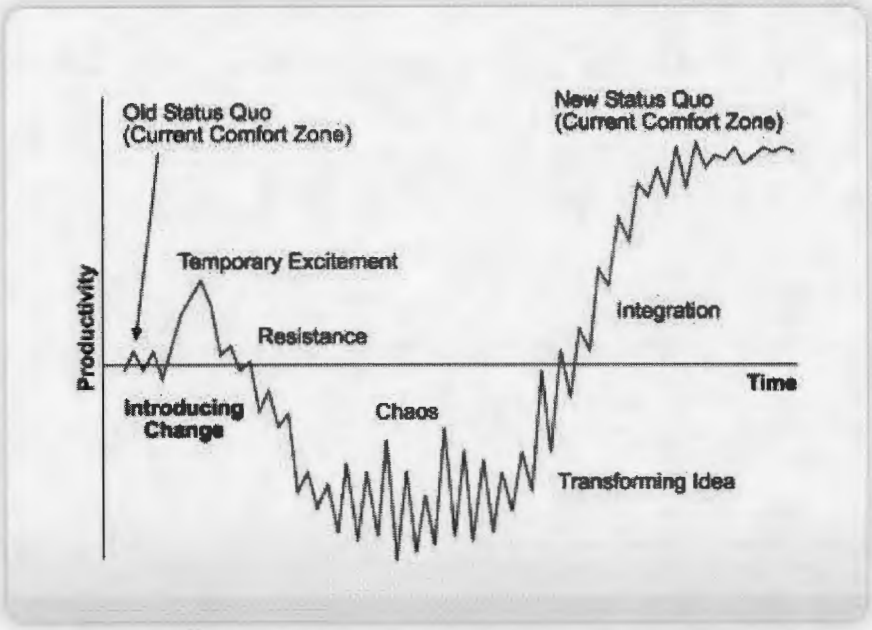


Figure 2.4 Le modèle de changement de Satir

Tiré de https://www.pinterest.ca/pin/451697037597561946

Le modèle du changement illustre qu'il se produit diverses phases après l'introduction d'un changement organisationnel majeur. Tout d'abord l'excitation liée à l'introduction d'un élément nouveau ; ensuite la résistance face à ce nouvel élément ; une période de chaos où tout devient pire qu'avant l'introduction du changement ; une période d'adaptation où les idées s'orientent positivement ; l'intégration de ces idées

[19] L'auteur fait référence au modèle du changement de Satir (voir Figure 2.4), nommé d'après sa conceptrice : Virginia Satir, célèbre psychothérapeute américaine. Le modèle a été utilisé et repris dans de nombreux ouvrages de gestion pour illustrer les étapes d'un changement organisationnel.

puis le nouveau *statu quo* où l'on a évolué comparativement à où l'on se situait avant l'introduction du changement.

Un problème qui survient généralement dans les changements organisationnels est l'abandon d'un nouveau processus durant la période de chaos. Les individus offrant beaucoup de résistance et étant mal informés croient que le changement nuit et est néfaste et préfèrent retourner à leur état antérieur.

Crispin et Gregory (2009) proposent également d'autres facteurs de succès qui ne seront pas inconnus pour ceux ayant déjà vécu une transition agile :

- utiliser les rétrospectives afin que les individus puissent librement exprimer leurs peurs et donner du *feedback* ;
- célébrer les succès de l'équipe ;
- s'assurer d'avoir l'engagement et le soutien des gestionnaires.

Permettre une meilleure communication entre les testeurs et les clients favorisera aussi le changement de paradigme. Que ce soit une communication directe ou par un intermédiaire, cela permettra à l'équipe de QA de s'améliorer en apprenant des forces et faiblesses du client et vice-versa (Crispin et Gregory, 2009).

Afin de bien gérer les attentes des gestionnaires, on recommande d'offrir des mesures régulières de l'avancement, par exemple en les illustrant à l'aide d'une *burndown chart* après chaque itération.

2.7 Mesure de l'avancement

Il existe différentes façons de mesurer l'avancement d'un projet d'automatisation.

2.7.1 Couverture de test

En optant pour une stratégie d'automatisation basée sur la couverture de test, l'objectif étant généralement d'atteindre x % de couverture ou d'augmenter de x % celle existante, la mesure régulière de la couverture est un bon indicateur d'avancement. Par contre, elle demande une bonne connaissance de la couverture existante ou de l'ensemble des fonctionnalités existantes. Certains outils fournissent ce calcul en comparant le code source avec le code de test (ex. SonarQube[20]).

2.7.2 Durée de la période de régression

En optant pour une stratégie visant à raccourcir le temps à exécuter des tests manuellement, l'objectif est de réduire l'effort lié à la régression sans nécessairement viser d'augmentation de la couverture des tests.

2.7.3 Nombre de tests écrits

En optant pour une stratégie visant à raccourcir la durée d'exécution des tests manuellement, l'objectif est de réduire l'effort lié à la régression sans nécessairement viser d'augmentation de la couverture des projet.

2.7.4 Vélocité

En optant pour l'atteinte d'un certain nombre de tests pré-déterminés à être automatisés, la vélocité peut être un indicateur un peu plus précis que le nombre de

---

[20] SonarQube (précédemment Sonar2) est un logiciel libre permettant de mesurer la qualité du code source en continu. SonarQube, «Dans Wikipédia, l'encyclopédie libre», (2019, 27 avril, 22 h 55)

tests. L'effort d'automatisation des tests (ou de leur complexité) peut bien se prêter à l'estimation en *story points*. Tout comme pour un projet de développement agile, l'estimation en *story points* devient de plus en plus linéaire au fur et à mesure que l'équipe prend de la maturité. Cela permet d'estimer de manière plus précise la date de complétion du projet, bien que cette « mesure » soit reconnue comme étant subjective.

2.8 Synthèse

En résumé, il s'avère que la majorité des organisations aient préconisé que la prochaine priorité pour réduire la période de régression est l'automatisation des tests.

Il est important d'être conscient de l'effort requis pour l'automatisation *versus* le bénéfice acquis par cette automatisation. L'automatisation des tests est une activité qui consomme beaucoup de temps (Aranha et Borba, 2007). Comme les ressources et le temps sont limités, le choix des tests à automatiser est important. Il faut d'ailleurs bien définir les attentes auprès des investisseurs.

CHAPITRE III

MÉTHODOLOGIE

Travaillant dans un environnement où le travail d'équipe et la collaboration sont de mise, mon approche initiale était d'impliquer le plus possible différentes parties prenantes dans le projet d'automatisation. Comme l'organisation favorise l'autonomie et la responsabilisation de chaque individu, mon rôle aurait été d'accompagner, de guider et de supporter l'équipe afin que les objectifs de l'automatisation soient atteints avec succès. Les décisions auraient pu être prises en équipe afin de favoriser l'engagement de celle-ci et mon travail aurait consister à informer au mieux l'équipe afin qu'elle puisse prendre des décisions éclairées.

Cependant, bien que cette démarche souhaitable ait été démarrée vers la fin de 2017, un nombre considérable d'embûches a fait en sorte qu'il n'a pas été possible de poursuivre dans cette voie (voir le chapitre V pour une discussion à ce propos). Par conséquent, et pour mieux contrôler les tenants et aboutissants de mon projet de synthèse, il a été convenu avec ma direction de projet que je fasse moi-même une preuve de concept avec un sous-ensemble des tests à automatiser, et ce avec une implication minimale de l'équipe.

3.1 La preuve de concept

Étant donné plusieurs tentatives d'automatisation des tests infructueuses par le passé, le projet d'automatisation est considéré risqué et avec un haut potentiel d'échec par différents membres de l'organisation. Bien que l'automatisation soit souhaitée, il y a

un faible engouement à investir davantage dans l'automatisation et la question d'externaliser le projet se pose. Mon projet servira donc de preuve de concept (POC) au projet d'automatisation.

La POC aura pour but :

- de permettre l'atteinte des deux principaux objectifs de ce projet (voir 1.3) ;
- de faire ressurgir rapidement les risques pouvant potentiellement se réaliser et de les évaluer ;
- d'augmenter mon niveau d'expertise sur l'automatisation des tests ;
- de me familiariser avec les différents outils d'automatisation ;
- d'augmenter la crédibilité de l'approche d'automatisation auprès de la direction ;
- de démontrer la faisabilité d'un projet d'automatisation à l'interne.

Cette POC servira d'ailleurs à démontrer à la direction qu'il est possible de mener un projet à terme et avec succès afin d'avoir l'appui nécessaire pour la poursuite du projet d'automatisation. Le but sera aussi d'expérimenter et faire ressortir les risques le plus rapidement possible.

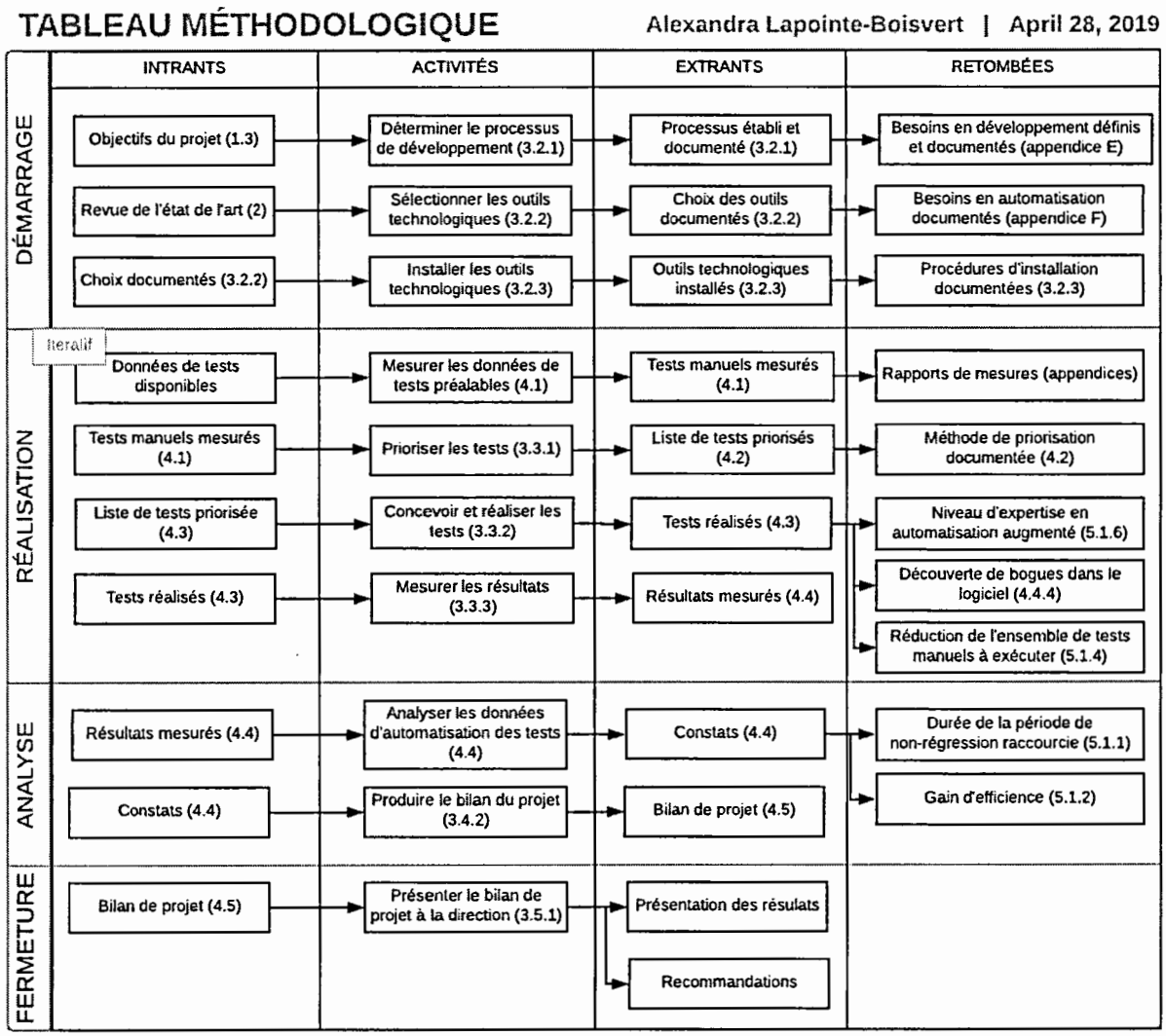
Cadre de la preuve de concept (portée du projet)

La POC se limitera à l'automatisation de 15 tests candidats découlant de la priorisation qui sera décrite en 3.3.1. Je ferai la conception, la réalisation des tests et la mesure des résultats.

Le nombre de tests est expressément défini à 15 pour deux raisons. La première : le projet doit être réalisé dans le cadre d'un projet de maîtrise ayant une portée somme toute limitée. La deuxième : le nombre doit être assez grand afin de pouvoir obtenir des résultats significatifs, soit une réduction de la durée d'exécution des tests. La *POC* se

déroulera sur une période de 3 mois, c.-à-d. 12 sprints d'une durée d'une semaine (voir appendice).

Tableau 3.1 Tableau méthodologique



3.2 Phase de démarrage

Les étapes suivantes ont été réalisées dans le sprint 0 (voir appendice D, tableau D.1).

3.2.1 Déterminer le processus de développement

En fonction des besoins de développements (voir appendice G, tableau E.), j'ai comparé les processus de développement et choisi une approche hybride entre Scrum et Kanban en optant pour des sprints d'une durée d'une semaine. Ce processus m'a permis régulièrement de bien montrer l'avancement du projet, notamment à l'aide d'un *sunset graph* (Boisvert et Trudel, 2011a), disponible dans le bilan de projet (voir section 4.5).

3.2.2 Sélectionner les outils technologiques

J'ai tout d'abord dressé une liste des besoins en automatisation (voir appendice F, tableau F.1). J'ai ensuite comparé différents outils et sélectionné les mieux appropriés pour mon projet et pour les projets futurs en automatisation (voir appendice F, tableau F.2 pour une liste des outils).

J'ai également déterminé que les tests seraient écrits dans le langage *Python*. Le choix du langage a été basé sur une combinaison de facteurs : sa simplicité d'apprentissage, la facilité de trouver des testeurs connaissant ce langage et sa rapidité d'écriture et d'exécution.

3.2.3 Installer les outils technologiques

L'installation des outils a consisté à bien configurer les différentes applications pour les besoins de mon projet et à en documenter les étapes dans le Confluence de Broadsign.

Les outils ont en général été installés en suivant les directives et standards recommandés du fabricant. Pour l'installation de Git, j'ai suivi les directives de configuration globales établies par Broadsign (Broadsign, 2016).

Au niveau du langage *Python*, j'ai utilisé la dernière version disponible au moment de commencer mes tests, c.-à-d. la version 3.7.2. Certaines librairies ont été utilisées, notamment *unittest*, *requests* et *Pil*, la liste de celles-ci se retrouvent dans le fichier « *requirements.txt* » de mon projet d'automatisation. Celui-ci se trouvant sous contrôle de version dans le dépôt de la compagnie.

3.3 Phase de réalisation

La phase de réalisation s'est déroulée entre les sprints 1 et 10 : l'étape 3.3.1 a été réalisée au sprint 1, les étapes 3.3.2 à 3.3.5 itérativement du sprint 2 à 10.

3.3.1 Prioriser les tests

Définir la méthode priorisation

La méthode de priorisation devait permettre d'obtenir un sous-ensemble de tests candidats qui amènerait rapidement le plus de valeur à Broadsign et de répondre aux objectifs du projet.

Afin de prioriser les tests à automatiser, j'ai d'abord établi des critères de priorisation (voir appendice G, tableau G.1). J'ai ensuite utilisé ces critères et filtré les tests suivant le modèle illustré à la figure 3.1 afin de réduire l'ensemble des tests à un ensemble prioritaire à automatiser durant mon projet de maîtrise.

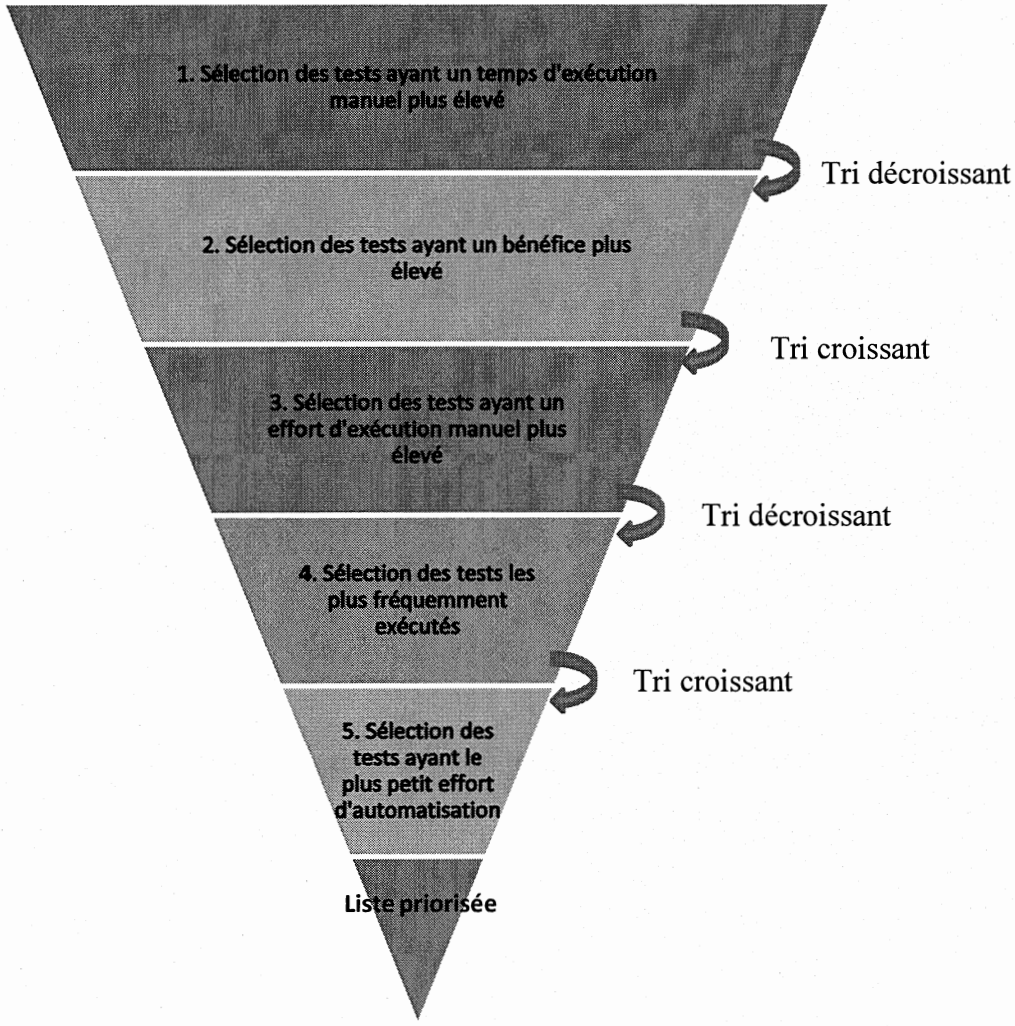


Figure 3.1 Méthode de priorisation pour la sélection des tests candidats

La figure 3.1 illustre que la sélection des tests sera faite en cinq étapes suivant l'entonnoir, c.-à-d. qu'à chaque étape, l'ensemble de tests sera réduit jusqu'à obtenir la liste finale de tests candidats.

Mesurer les données préalables des tests manuels

Chaque mesure recueillie vise l'un des deux, ou les deux objectifs suivants :

- pouvoir mesurer l'atteinte du premier objectif (réduire la durée de la période de non-régression)
- aider à la priorisation des tests à automatiser.

Les mesures suivantes ont été prises préalablement au développement de la preuve de concept et documentées dans la section 4.1 du chapitre IV.

- durée d'exécution des test manuels de non-régression ;
- effort d'exécution d'un test manuel de non-régression ;
- bénéfice d'un test manuel de non-régression ;
- effort pour l'écriture d'un test manuel ;
- effort d'automatisation d'un test manuel de non-régression ;
- fréquence d'exécution des tests manuels de non-régression.

Prioriser les tests à automatiser

Les tests ont été priorisés en appliquant la méthode de priorisation définie un peu plus haut. Ces tests ont été maintenus dans mon carnet de produit dans Google Sheets.

3.3.2 Concevoir et réaliser les tests

Planifier les tests à automatiser dans le sprint

À chaque début de sprint j'ai choisi un ensemble de tests à automatiser basé sur ma capacité évaluée en *story points*. Au fur et à mesure que je complétais des sprints, j'étais en mesure de connaître ma vélocité et ainsi d'ajuster mon carnet de sprint de manière

plus précise. L'estimation me permettait également de prévoir la date de livraison du projet actuel ou futur.

Normalement, les tests devraient être exécutés suivant leur priorité, toutefois, la découverte de bloquants ou d'imprévus pourrait m'amener à reprioriser les tests à automatiser en cours de projet.

Réaliser les tests

La conception de test s'est faite en suivant ces critères de qualité : la maintenabilité, la lisibilité et la réutilisabilité. Les tests ont été placés sous contrôle de version dans le Bitbucket de l'organisation.

3.3.3 Mesurer les résultats

Durant la réalisation des tests, des mesures supplémentaires ont été prises pour chaque itération :

- nombre de tests manuels automatisés ;
- nombre de tests automatisés générés ;
- nombre de bogues découverts ;
- nombre de story points complétés ;
- effort d'automatisation (en heures) ;
- durée d'exécution des tests automatisés (en secondes) ;
- point de rentabilité des tests automatisés.

Ces mesures m'ont aidé à prendre conscience de mon avancement et à pouvoir m'améliorer mais surtout permettre de déterminer si les objectifs du projet seraient atteints.

3.4 Phase d'analyse

3.4.1 Analyser les données d'automatisation des tests

La phase d'analyse a été entamée lorsque l'exécution des tests s'est terminée. Elle a consisté à analyser les données récoltées en 3.3.3.

Cela m'a permis de déterminer l'atteinte ou non des objectifs, d'évaluer le retour sur investissement, et d'extrapoler les résultats potentiels de la complétion du projet d'automatisation.

La phase d'analyse a été exécutée au sprint 11 et s'est poursuivie pour toute la durée du sprint.

3.4.2 Produire le bilan de projet

Le bilan de projet a été produit au Sprint 12. Le bilan de projet est un rapport sommaire de trois pages résumant l'ensemble des données du projet (voir section 4.5).

3.5 Phase de fermeture

3.5.1 Présenter le bilan de projet à la direction

La présentation du bilan de projet consistera à adapter le bilan de projet sous une forme visuelle afin de le présenter aux différentes parties prenantes lors d'une rencontre. Pour se faire, j'utiliserai Google Slides et produirai une présentation d'une dizaine de page illustrant le contenu du bilan de projet. Le but est d'offrir une visibilité complète aux

décideurs afin qu'ils puissent prendre une décision éclairée quant au choix de l'automatisation des tests de non-régression comme solution.

CHAPITRE IV

RÉSULTATS

4.1 Mesures préalables

Les prochaines sections illustrent les données recueillies préalablement au développement de la preuve de concept.

4.1.1 Durée d'exécution des tests manuels

La durée d'exécution des tests manuels a été mesurée en nombre de semaines par module, soit l'information la plus granulaire et fiable disponible. La figure 4.1 illustre la durée moyenne pour les derniers *releases* ayant eu lieu de 2014 à 2017.

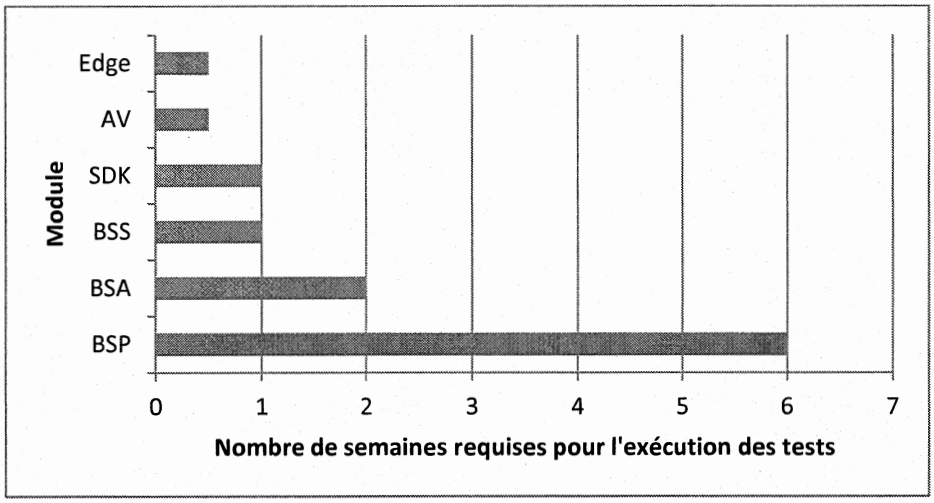


Figure 4.1 Durée requise pour l'exécution des tests de non-régression par module

La mesure démontre que la durée d'exécution des tests du module BSP est la plus longue. Généralement, le module BSP est testé en premier suivi du module BSA exécuté en parallèle avec les autres modules. La période de régression totale est donc d'une durée moyenne de 8 semaines alors que la durée pour tester le module BSP est 6 semaines, soit 75% de la durée.

4.1.2 Effort d'exécution des tests manuels de non-régressions

Deux données d'effort étaient à ma disposition : l'effort par test estimé en *story points*[21] par l'équipe et l'effort total nécessaire pour accomplir l'ensemble des tests de non-régressions (en jour-personnes).

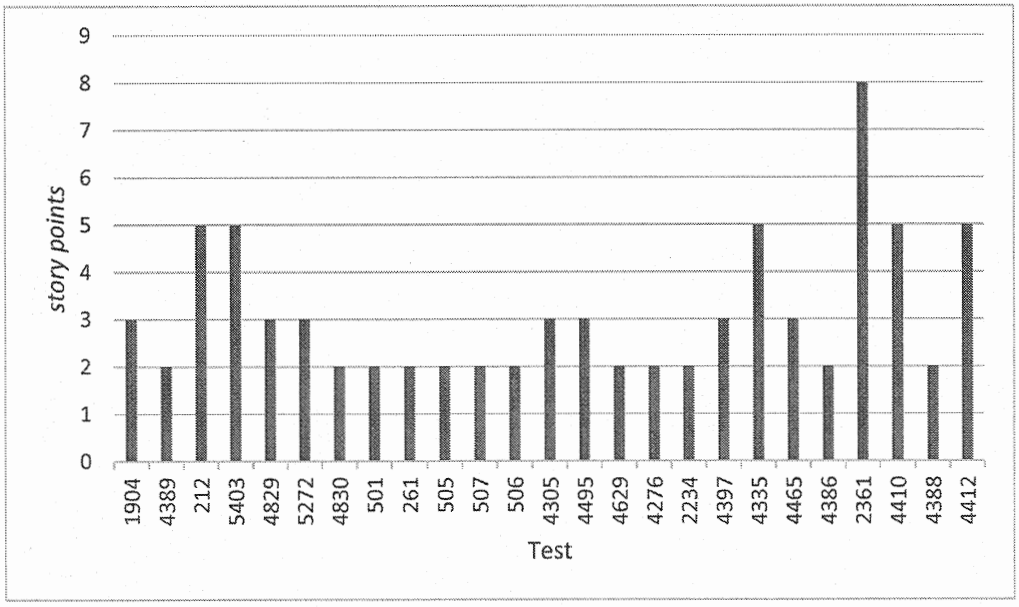


Figure 4.2 Effort estimé d'exécution manuel par test en *story points*

[21] Malgré que cette valeur ne réponde pas aux critères de métrologie (voir 2.3.3.2) et soit très subjective, elle permettra tout de même de comparer les tests entre eux et de déterminer lesquels semblent les plus ardus à exécuter manuellement.

L'effort estimé d'exécution d'un test manuel varie entre 2 et 8 *story points*.

Tableau 4.1 Effort consacré à la régression comparé au nombre de tests effectués par version

Version	Effort (jours-personnes)	Effort (heures-personnes)	Nombre de tests effectués
13.1	55	440	100
13.0	59,5	476	243
12.1	75,6	604,8	235
12.0	92	736	279

Sachant que chaque test est différent et qu'ils ne peuvent être comparés en terme d'effort, j'ai tout de même calculé la durée et l'effort moyens d'exécution d'un test pour fin de comparaison.

Tableau 4.2 Effort et durée moyens par test et par version

	12.0	12.1	13.0	13.1	Moyenne
Effort par test (h-p)	2,64	2,48	2	4,4	2,88
Durée par test (heures)*	21,12	19,84	16	35,2	23,04

*Chez Broadsign, une journée de travail comprends 8 heures.

4.1.3 Effort pour l'écriture d'un test manuel

Aucune donnée n'étant compilée sur le temps d'écriture d'un test manuel, j'ai opté pour un sondage auprès des membres de l'équipe de QA. Sur 5 personnes interrogées, trois ont répondu. Les trois testeurs ont répondu que l'écriture d'un test prenait en moyenne entre une et trois heures dépendamment de la clarté des spécifications, c.-à-d. une moyenne de deux heures par test.

4.1.4 Évaluation du bénéfice des tests

L'objectif ici était d'évaluer le bénéfice des tests. Plus un test est susceptible de révéler des régressions en production et plus son bénéfice sera haut.

Le bénéfice a été évalué à partir de deux autres mesures : les bogues catégorisés par phase de développement dans laquelle ils ont été découverts et les bogues catégorisés par la composante affectée. Les mesures se sont limitées aux bogues du module BSP corrigés entre 2016 et 2018

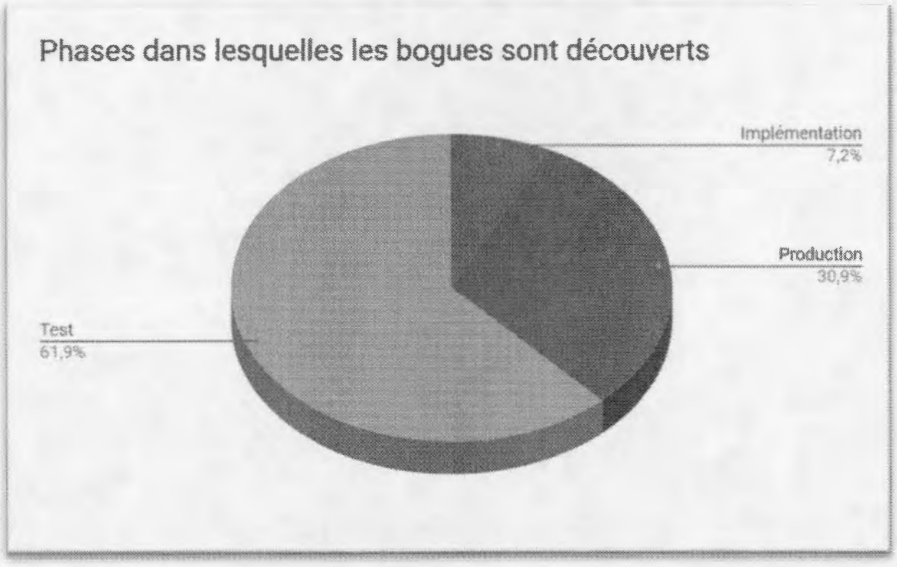


Figure 4.3 Bogues découverts par phase de développement

La figure 4.3 illustre que la grande majorité des bogues (61,9%) sont découverts dans la phase de test. Chez Broadsign, cette phase correspond à la période de tests de non-régression. La seconde majorité (30,9%) sont découverts en production. Le reste des bogues (7,2%) sont découverts durant l'implémentation, c.-à-d. durant les sprints de développement. Au total, il y a eu 192 bogues (voir appendice J pour la liste détaillée).

Le sous-ensemble des bogues découverts en production a ensuite été catégorisé selon la composante affectée tel qu'illustré à la figure 4.4 (voir appendice I pour une description des différentes composantes).

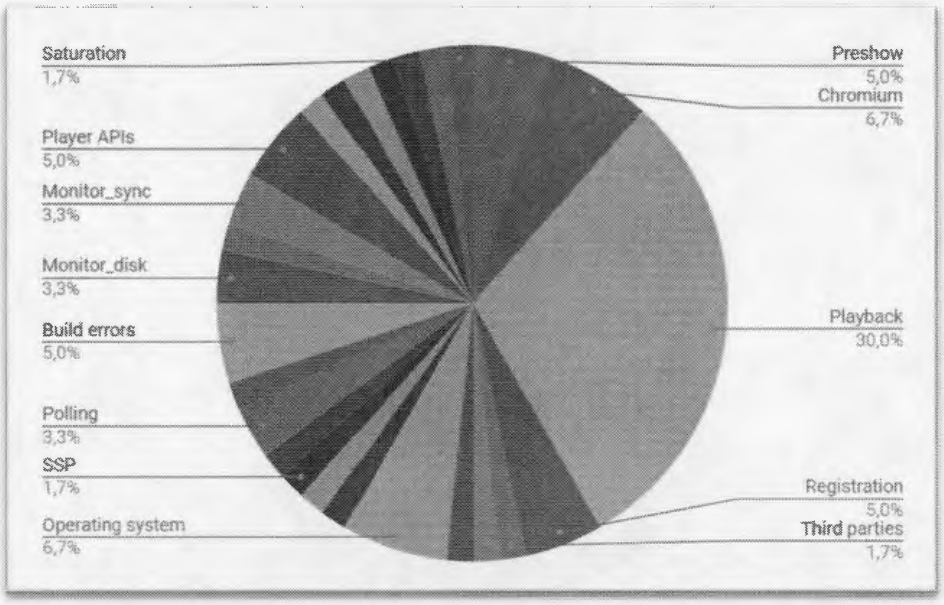


Figure 4.4 Répartition des bogues selon la composante touchée

La figure 4.4 illustre que la majorité des bogues trouvés en production (30,0%) sont liés au *playback.* À ex aequo ensuite se retrouvent les bogues des composantes *operating system* et *chromium* (6,7%). De nouveau à égalité, les bogues des composantes *Player APIs*, *Preshow*, *Builds errors* et *Registration* (5%) viennent ensuite.

Les tests validant la lecture de contenu (*playback*) seront donc ceux retenus comme ayant le bénéfice le plus élevé.

L'ensemble des données suivantes ont été récupérées de Testopia et colligées dans un tableau contenant tous les tests mesurés se trouvant à l'appendice K.

4.1.5 Fréquence d'exécution

La fréquence d'exécution varie entre 1 et 61. Néanmoins, la majorité des tests ont été exécutés entre 2 et 10 fois (voir figure 4.5).

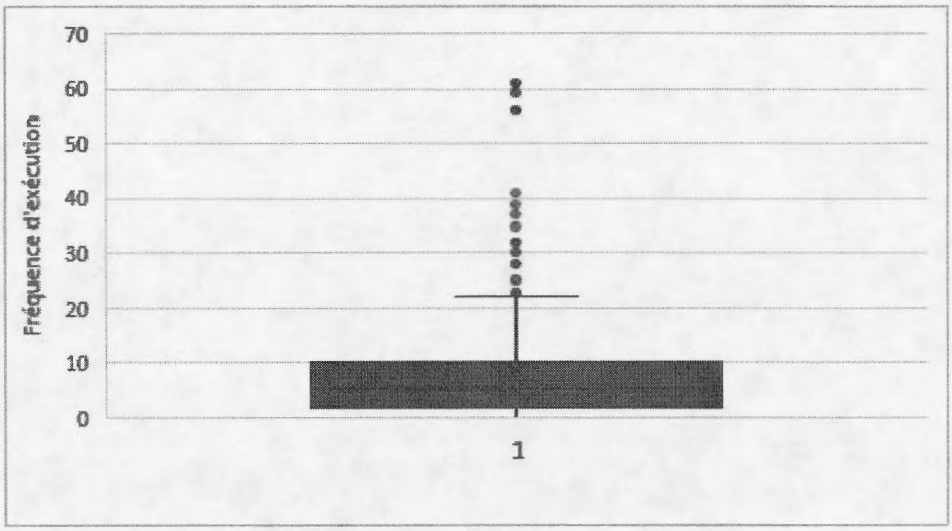


Figure 4.5 Fréquence d'exécution des tests de non-régressions

4.1.6 Effort d'automatisation des tests

Afin d'estimer l'effort d'automatisation en *story points*, je me suis pairée à un membre de l'équipe d'automatisation. Cette activité étant coûteuse en temps pour

l'organisation, nous avons donc limité le nombre de tests à estimer à ceux retenus dans la dernière étape de priorisation.

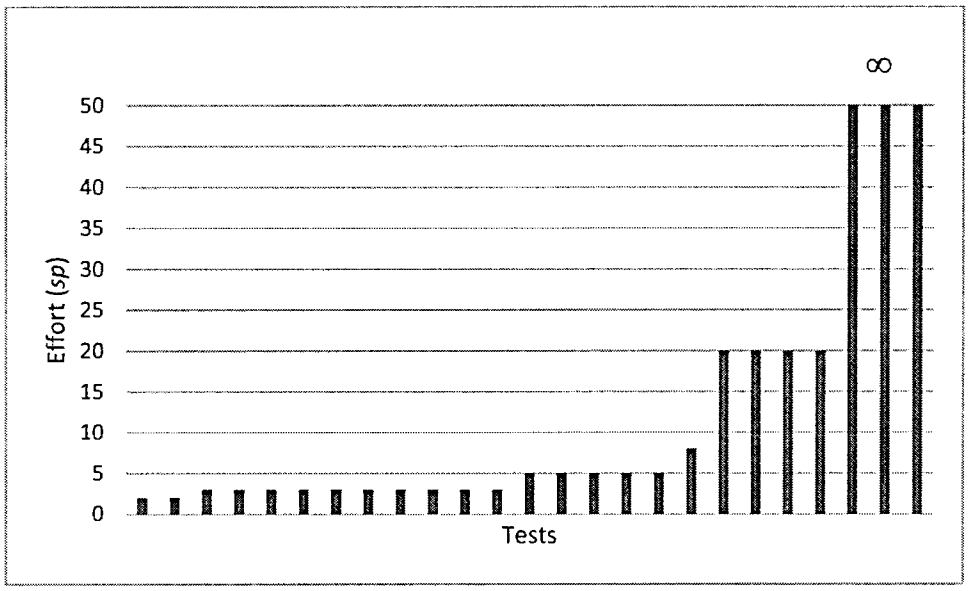


Figure 4.6 Effort estimé d'automatisation par test

L'effort d'automatisation varie de 2 à 20 *story points*. Cependant, 3 tests ont été estimés à la valeur ∞, signifiant que nous n'étions pas en mesure de les estimer puisqu'à priori nous ne voyions pas comment nous pouvions les estimer.

4.2 Liste priorisée des tests à automatiser

En appliquant la méthode de priorisation sur les données recueillies en 4.1, j'ai obtenu la liste des tests prioritaires à automatiser (tableau 4.3).

Malgré que la preuve de concept vise à automatiser une quinzaine de tests, j'ai tenu à avoir un peu de marge de manœuvre en faisant une sélection un peu plus grande, soit 25 tests. Le but étant de pouvoir pallier à des imprévus.

Tableau 4.3 Liste priorisée des tests à automatiser

Priorité	Référence	Description sommaire
P1	1904	Clean install
P2	5403	All Media Types test
P3	4465	Trigger Synchronization: Synchronization Sets
P4	4829	Gapless playback and Video Transitions
P5	4388	Trigger Synchronization: Video Sync on Multiple Systems II
P6	5272	Performance and stability
P7	4389	Screen Orientation
P8	4495	Partial Booking
P9	4386	Trigger Synchronization: Video Sync on Multiple Frames
P10	4629	Automatic Retargeting VII
P11	4335	Remote Geometry
P12	4830	Simple Screen Control
P13	501	Playlist over-saturation (transform_over_saturate)
P14	4276	Playback upon network disconnect
P15	261	Reservations and campaigns
P16	2234	Resume Default loop schedule
P17	212	Fullscreen bundle
P18	4397	Conditions using remote_action utility
P19	505	Playlist under-saturation (transform_under_saturate)
P20	507	Under-saturation with loop segmentation
P21	4412	Transitions between different media types
P22	506	Under-saturation with category separation
P23	4475	Time Synchronization
P24	4305	Over and Under-saturation with same Category II
P25	2361	Playback of multiple triggered content in a Slave frame

La liste est ordonnée de la plus importante (P1) à la moins importante (P25). La référence est le numéro de référence dans le logiciel de test Testopia.

4.3 Tests automatisés

À partir de la liste priorisée, j'ai été en mesure d'automatiser 15 tests manuels complètement, ce qui s'est résulté en 32 tests automatisés.

4.4 Résultats mesurés

Le tableau 4.4 illustre les mesure recueillies telles que décrites en 3.3.3.

Tableau 4.4 Compilation des mesures des tests automatisés

Test	Nombre de tests auto-matisés	Prio-rité	Durée d'exécution manuelle (s)**	Durée d'exécution automatisée (s)	Effort d'automati-sation (heures)	Point de rentabilité (n)
1904	1	s/o	1940	12,135	7,42	10*
5403	1	P2	5265	39,266	5,16	3
4389	5	P7	8060	46,106	1,02	0
5272	1	P6	7450	20,401	0,16	0
4495	3	P8	4826	114,239	12,36	8*
501	1	P13	5486	22,519	7,40	4
4830	4	P12	7041	170,883	9,85	5
4629	2	P10	7794	11,439	3,19	1
261	2	P15	4055	32,430	1,18	1
505	1	P19	3000	30,255	2,36	0
2234	3	P16	1925	128,881	1,14	0
212	2	P17	5485	27,819	3.02	1
507	2	P20	1525	48,898	3,49	4
506	2	P22	1267	47,857	0,98	0
4305	2	P24	2806	89,106	1,10	0
Moyenne	s/o	s/o	4528	55,066	3,12	1
Total	32	s/o	67925	842,234	60,30	2

* Les mesures n'étaient pas cohérentes et prises de manière différente d'une personne à l'autre, je les ai tout de même conservées car elles seront discutées au chapitre V.

** Les données des tests ont été retirées de la moyenne étant des valeurs extrêmes et non valides pour le calcul de la moyenne.

Deux choses sont à noter par rapport à ces résultats :

Premièrement, l'ordre de priorisation n'a pas été suivi à 100%. Dans quelques cas, (4465, 4829, 4388, 4386), soient les priorités 3, 4, 5 et 9, l'effort s'est révélé être beaucoup plus grand que prévu et j'ai dû les laisser tomber (pour le moment) car je n'aurais pas été en mesure de les compléter dans le cadre de mon projet. J'ai aussi dans certains cas (4495) dû faire du *refactoring* nécessaire pour le test mais s'appliquant à tout le *framework* de test, ce qui biaise un peu l'effort d'automatisation de ce test. Dans

d'autres cas (4335), soit la priorité 11, j'ai entamé le test mais j'ai dû arrêter en cours de chemin étant bloquée par un bogue du logiciel. Le test sera complété une fois le bogue corrigé, après mon projet. Sinon, de manière générale, l'ordre a été changé parce que j'étais bloquée temporairement sur un test de plus haute priorité.

Il est aussi à noter que les premiers tests ont en général été beaucoup plus longs à automatiser. Ceci s'explique par deux raisons : la première, j'étais exposée à un nouveau langage de programmation et la durée d'automatisation inclut ma courbe d'apprentissage. La deuxième, les premiers tests incluent la construction de mon *framework*, effort rendu très faible ou parfois inexistant à partir du 8^{e} test.

Je suis également en mesure de constater, qu'une fois la courbe d'apprentissage complétée, que la durée d'automatisation d'un test manuel se situe entre 1 et 3,5 heures, ce qui est très similaire à l'effort d'écriture d'un test manuel se situant, tel que vu en 4.1.3, entre 1 et 3 heures.

Le calcul du point de rentabilité est expliqué dans la prochaine section.

4.4.1 Calcul du ROI

Pour chacun des test automatisés, j'ai calculé le ROI tel qu'expliqué au chapitre II. Par exemple, pour le test 5403 *All media type test*, j'ai calculé le temps total consacré au test, d'exécution en exécution, ce qui m'a permis d'obtenir le résultat à la figure 4.7).

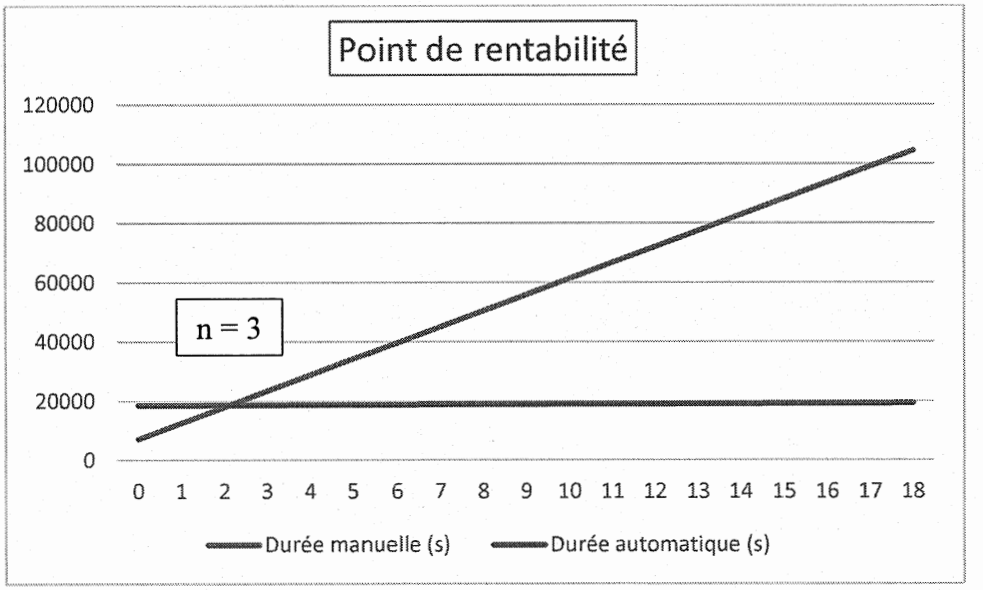


Figure 4.7 Point de rentabilité du test « 5403 All media type test »

J'ai additionné à la durée d'écriture du test manuel, sa durée d'exécution à chaque exécution, donnant ainsi une droite augmentant de façon linéaire (droite bleue). Pour le test automatisé, j'ai additionné la durée requise pour automatiser le test et la durée d'exécution pour chaque exécution (droite rouge). Cette durée étant très négligeable par rapport à la durée d'écriture, la droite obtenue augmente à peine au fil des exécutions. Le point de croisement des deux droites correspond au point de rentabilité, soit le nombre minimal d'exécutions d'un test manuel pour que son automatisation soit rentable. Dans l'exemple ci-haut, l'automatisation devient rentable après seulement trois exécutions.

Dans l'ensemble, les tests automatisés sont rentables en moyenne après seulement une exécution (voir tableau 4.4).

4.4.2 Calcul du gain d'effort

L'effort moyen d'exécution par test manuel étant de 2,88 heures-personnes, nous pouvons estimer que l'effort lié à l'exécution de 15 tests manuels avoisine 43,2 heures-personnes (voir tableau 4.7). L'effort d'exécution automatisée total est quant à lui de 14 minutes-personnes ou 0,23 heures-personnes.

Sachant cela, nous pouvons extrapoler un gain d'effort de 43 heures-personnes pour la prochaine période de tests de non-régression.

4.4.3 Calcul du gain d'efficience

En comparant la durée d'exécution totale des 15 tests manuels, soit 67 925 secondes, à la durée d'exécution totale des tests automatisés, soit 842 secondes (voir tableau 4.4), je suis en mesure de constater un gain d'efficience de 98,8% (voir équation 4.1).

$$\text{Gain d'efficience} = \frac{\textit{Durée d'exécution manuelle}}{\textit{Durée d'exécution manuelle} - \textit{Durée d'exécution automat.}} \quad (4.1)$$

J'abstrais ici l'effort d'écriture d'un test manuel et l'effort d'automatisation puisque tel que vu précédemment, les deux efforts sont sensiblement équivalents.

4.4.4 Bogues découverts

11 bogues ont été découverts durant l'automatisation des tests : 4 bogues dans la documentation (site web), 1 bogue lié à l'infrastructure (INC0016781), 4 bogues liés à l'API et 2 bogues liés à BSA (voir l'appendice N pour une liste détaillée des bogues découverts). Sur les 11 bogues découverts, 4 étaient à ce point importants qu'ils ont été corrigés sur le champs et les autres seront priorisés dans la version 13.2.

4.5 Bilan de projet

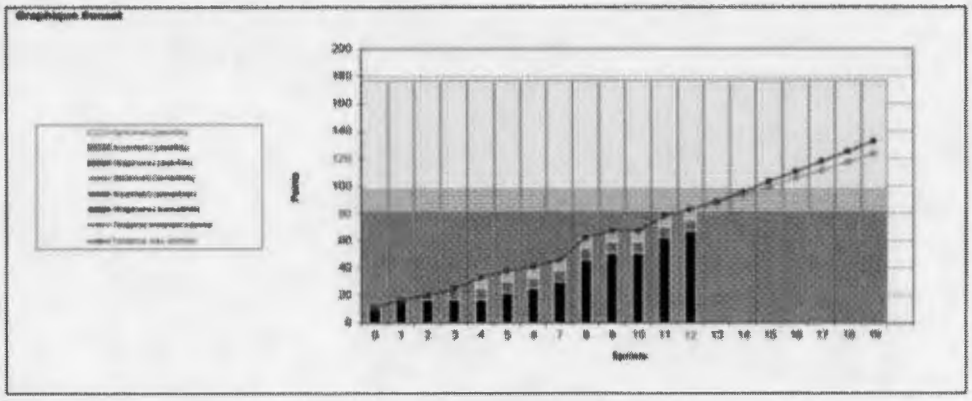
Le projet n'a pas été exécuté comme il avait été initialement prévu. Tel que mentionné en début de chapitre, l'idée était de faire le projet en équipe et d'automatiser 25 tests candidats. Il a plutôt fallu que je replanifie tout mon projet et que je prenne la charge complète. Malgré cela, j'ai été en mesure avec seulement 15 tests candidats automatisés d'obtenir des résultats importants qui m'ont permis d'atteindre les objectifs du projets.

Vision:	Le projet consiste en une preuve de concept sur l'automatisation de tests de régressions clés qui permettront de réduire la durée de la période de régression.	**Date:**	2019-01-07
But:	**Donner confiance à l'organisation d'investir dans l'automatisation en démontrant qu'il est possible de réduire significativement la durée de la période de régression par l'automatisation.**	**Date de début:**	2019-01-07
Critères de succès:	La période de régression est réduite en temps	**Date de fin:**	2019-04-05

Engagements:

	Liste des items:	Statut
1	Processus de développement déterminé	Terminé
2	Approche d'automatisation sélectionnée	Terminé
3	Choix des outils documentés	Terminé
4	Outils installés	Terminé
5	Liste de tests priorisés	Terminé
6	Tests réalisés	En Cours
7	Résultats mesurés	Terminé
8	Rapports de sprint	Terminé
9	Améliorations au processus documentées	Terminé
10	Bilan de projet	Terminé

État d'avancement:



État de la qualité:

Risques	Description	**Statut**	**Résolution**		
Obstacles	**Description**	**Découvert à l'itération :**	**Résolution prévue à l'itération :**	**Responsable**	**Statut**
Plan d'amélioration:	Valider les données avant l'envoi vers l'open API Intégrer le bsp quickstart afin de pouvoir démarrer le lecteur à partir d'un fichier json				

Coût pour compléter	**Dans le pire scénario**		**Dans le meilleur**
En USP	94 USP	70,5	117,5
En itérations	15 itérations	11 itérations	18 itérations

Figure 4.8 Bilan de projet

CHAPITRE V

DISCUSSION

Ce chapitre s'intéresse aux retombées de mon projet d'automatisation. Les prochaines sections décriront les retombées positives mais également les différentes embûches ayant affecté ce projet.

5.1 Retombées positives du projet d'automatisation

5.1.1 Période de tests de non-régression raccourcie

Le gain d'effort établi en 4.4.2 a été évalué à 43 h-p, c.-à-d. un peu plus d'une semaine-personne.

En prenant comme hypothèse que la prochaine régression pour la version 13.2 sera similaire à celle de la version 13.1 en terme d'effort, soit 55 j-p (voir tableau 4.1), cela me permet de conclure à un gain d'effort de près de 10% pour la prochaine période de tests de non-régression.

1,85 testeurs étant affectés à la régression en moyenne lors d'une période de test de non-régression, cela signifie une réduction de la durée de 23,24 heures, soit presque 3 jours.

Et ce, en émettant l'hypothèse que la compagnie aurait payé pour l'écriture des tests automatisés, ce qui n'a pas été le cas (voir 5.2.2).

5.1.2 Gain d'efficience

Tel que vu en 4.4.3, une retombée intéressante est le gain d'efficience dans l'exécution des tests de non-régression. On constate que l'exécution automatisée des 15 tests manuels entraîne un gain d'efficience de 98,8%.

5.1.3 Très bon retour sur investissement

En considérant l'effort requis pour l'automatisation de chacun des tests, j'ai aussi pu être en mesure de déterminer le retour sur investissement pour chacun d'eux.

Le retour sur investissement moyen pour les tests exécutés se fera après une seule exécution. Sachant que ces tests sont en moyenne effectués 4,6 fois par année. Cela signifie que les tests seront rentables dans le courant de la première année, fort probablement dans le premier quart de l'année.

Le nombre d'exécutions variant entre 1 et 10 pour atteindre le ROI, il est intéressant de noter que plus de la moitié des tests manuels automatisés sont rentables après 1 ou une exécution, signifiant que plus de la moitié d'entre eux sont déjà rentables ayant tous été exécutés minimalement une fois au cours de ce projet.

5.1.4 Diminution importante du nombre de tests à effectuer pendant la période de non-régression

L'automatisation des tests candidats a démontré qu'il n'est pas souhaitable d'automatiser exactement ce qui était couvert par le test manuel. Dans certains cas il n'était pas possible de le faire, par exemple débrancher une prise HDMI, dans d'autres

cas, j'ai pu ajouter de la valeur. Par exemple en vérifiant systématiquement les preuves d'impressions dans la base de données.

Cela a aussi mis la lumière sur des scénarios qui n'avaient plus de valeur à être testés, lesquels je n'ai non seulement pas investi de temps à transposer mais ayant également été retirés des tests de non-régressions à exécuter. La baisse du nombre de tests entre la version 13.0 (243 tests) et 13.1 (100 tests) est d'ailleurs due à l'élimination de tests désuets, dupliqués ou sans valeur. Initiative ayant été déclenchée par le constat découvert par mon projet d'automatisation.

Cette baisse diminue également la portée du projet d'automatisation pour lequel seulement une centaine de tests devront être automatisés pour le module BSP. Comme 15 l'ont déjà été, il en reste autour de 85, soit 85%.

5.1.5 Réduction de la courbe d'apprentissage des nouveaux testeurs

Un des points soulevés dans la description de la problématique est la longue courbe d'apprentissage, 2 à 3 mois en moyenne (voir section 1.2.1). Le projet illustre qu'après l'automatisation de huit tests sur une période d'environ deux semaines, un développeur junior est en mesure de développer des tests automatisés de façon efficiente.

5.1.6 Niveau augmenté d'expertise en automatisation

Après le développement des 15 tests, mon niveau d'expertise est beaucoup plus élevé, du fait même celle de Broadsign. Je serai également en mesure de partager mes apprentissages avec l'équipe en plus de la documentation déjà fournie par le projet.

5.1.7 Changement de culture face à l'automatisation

Tout au long du projet, j'ai été en discussion avec les testeurs mais également avec les développeurs de l'équipe *CORE*. J'ai démontré de manière informelle l'avancement de mon projet et j'ai dénoté un fort enthousiasme à vouloir mettre en place le processus d'automatisation complet au sein de l'équipe. J'ai également eu des offres de soutien de la part des testeurs et développeurs en fin de projet. Cela n'a pu être mis à profit pour ce projet, cependant, cela favorisera grandement la poursuite du projet d'automatisation.

5.2 Autres constats

5.2.1 Perte du soutien de l'organisation

Le plus gros obstacle a été la perte du soutien de l'organisation. Le projet qui devait d'abord être effectué par une équipe d'automatisation de cinq individus a dû être revu afin d'être réalisé seul. Cela m'a fait perdre énormément de temps en planification et replanification, en plus d'avoir réduit la portée (nombre de tests à automatiser).

5.2.2 Changement de direction et de mandat

Lors de mon absence (congé de maternité), un cadre nouvellement employé a été affecté comme responsable du projet d'automatisation. Celui-ci a décidé d'opter pour une approche incompatible avec l'approche que je proposais et a débuté le projet avant mon retour. L'affectation d'individus à temps partiel par exemple, le choix des priorités et des outils en sont quelques exemples. L'approche de cette équipe n'a par ailleurs pas été un succès et l'équipe a été démantelée en décembre 2018.

J'ai donc dû me retirer du projet, redéfinir mon propre projet et travailler sur une preuve de concept en solitaire en parallèle à ce que l'équipe démantelée en 2018 faisait. Par ailleurs, j'ai dû faire tout cet effort d'automatisation en dehors de mes heures normales de travail, n'ayant pas le soutien de la nouvelle direction.

5.2.3 Difficulté à obtenir des données fiables

Afin d'obtenir des données, j'ai dû consulter plusieurs sources de données : Testopia, Jira, Confluence, Google Sheets, etc... J'ai constaté qu'il y avait beaucoup de duplication mais qu'aussi les données divergeaient d'une source à l'autre. Il manquait aussi des données importantes comme l'effort consacré à l'écriture d'un test manuel. Cette donnée aurait précisé à la hausse la rentabilité des tests automatisés.

5.2.4 Forte résistance au changement

Plusieurs expressions de résistance au changement avaient été dénotées depuis l'annonce du projet d'automatisation. La première fût le non-suivi des recommandations offertes par mon projet, allant même jusqu'à l'inverse des directives recommandées et s'étant soldé par un échec.

Ce projet a démontré que la poursuite de ces recommandations a non seulement mené à l'atteinte des objectifs du projet mais a également démontré un potentiel de réduire la période de tests de non-régression à moins d'une semaine.

5.2.5 Expertise interne quasi-inexistante en automatisation

Les individus ont été positionnés dans des rôles nouveaux, inconnus et pour lesquels ils n'avaient pas l'expérience, voire pas le profil. De plus, aucune formation ni expertise

externe n'était disponible pour l'équipe. Cet état permet d'expliquer en grande partie les nombreuses résistances au changement et l'échec du projet d'automatisation entamé pendant mon congé de maternité.

5.3 Résumé

En résumé, le projet d'automatisation a généré plusieurs retombées positives, une réduction d'une semaine de l'effort lié à la période de non-régression, une réduction de 3 jours sur la durée, un gain d'efficience de 98,8% au niveau de l'exécution des tests de non-régression, un point de rentabilité moyen de une exécution, une réduction dans le nombre de tests de non-régression à exécuter, une réduction de la courbe d'apprentissage des nouveaux testeurs, une augmentation de l'expertise en automatisation chez Broadsign et un changement de culture face à l'automatisation des tests de non-régression.

Certains défis ont dû être surmontés durant le projet : une perte de soutien de la nouvelle direction, un changement de direction en milieu de projet, une difficulté à obtenir des données fiables, une forte résistance au changement ainsi qu'un manque d'expertise en automatisation.

CHAPITRE VI

RECOMMANDATIONS POUR LA POURSUITE DE L'AUTOMATISATION DES TESTS

Les prochaines sections décriront mes recommandations suite à la complétion de mon projet d'automatisation.

6.1 Poursuite de l'automatisation des tests de non-régression

Ma première recommandation est la poursuite de l'automatisation des tests de non-régression. Plus concrètement je propose les points suivants :

- continuer de prioriser les tests en terme de valeur ;
- allouer du temps pour que les individus de l'équipe puissent s'impliquer dans l'automatisation (testeurs, développeurs, *product owners*) ;
- définir l'automatisation comme objectif départemental ;
- offrir du soutien aux testeurs voulant automatiser des tests, que ce soit par l'offre de cours de perfectionnement ou de séances *lunch & learn* ;
- pour les fonctionnalités à venir, rendre la complétion du test automatisé *sine qua non* à la complétion d'une *story* (modifier la *definition of done*) lorsque possible.

6.2 Révision des critères d'embauche des nouveaux testeurs

Je recommande également de revisiter les critères d'embauches des testeurs afin de favoriser les candidats ayant un background en automatisation, une connaissance de langages scriptés et/ou avec un intérêt pour l'automatisation.

CONCLUSION

Un an et demi après le début de mon projet, et après avoir terminé ma preuve de concept, le climat est beaucoup plus propice à l'automatisation chez Broadsign. Dans l'équipe CORE, deux testeurs sont maintenant dédiés à 100% à l'automatisation. De plus, des séances hebdomadaires de *lunch & learn* sur l'automatisation avec *Python* ont été initiées par un des deux testeurs en automatisation afin d'offrir de la formation aux autres membres de l'équipe de testeurs. Les testeurs s'y joignent sur une base volontaire. En moyenne deux ou trois testeurs assistent aux séances. Également, certains développeurs de l'équipe CORE ont développé des modules afin d'améliorer la testabilité du système qui pourront accélérer l'automatisation des tests.

Retour sur les objectifs du projet

Avec ces constats, je peux donc assurément conclure à un changement de culture positif par rapport à l'acceptation de l'automatisation comme solution d'accélération des mises en marché et donc, à l'atteinte du deuxième objectif de mon projet. Quant au premier objectif, soit de réduire la prochaine période de non-régression, il a été illustré en 5.1.1 que la prochaine période de non-régression serait réduite de 3 jours, ce qui confirme l'atteinte de cet objectif.

Degré d'atteinte des objectifs de la preuve de concept

Les objectifs de la preuve de concept ont pour la plupart tous été atteints, je serai en mesure de valider la réussite ou non du 5^{e} objectif après avoir présenté le bilan de projet à la direction dans le prochain mois.

Tableau 7.1 Degré d'atteinte des objectifs de la preuve de concept

Objectif	**Statut**
1. Permettre l'atteinte des deux principaux objectifs de ce projet (voir 1.3).	
2. Faire ressurgir rapidement les risques pouvant potentiellement se réaliser et de les évaluer.	
3. Augmenter mon niveau d'expertise sur l'automatisation des tests.	
4. Me familiariser avec les différents outils d'automatisation.	
5. Augmenter la crédibilité de l'approche d'automatisation auprès de la direction.	
6. Démontrer la faisabilité d'un projet d'automatisation à l'interne.	

Résumé

En résumé, les deux principaux objectifs du projet ont été atteints et ainsi que 5 objectifs sur 6 au niveau de la preuve de concept (le 6^{e} étant en voie de l'être). La réalisation de mon projet a permis d'automatiser un sous-ensemble de 15 tests priorisés des tests manuels (15% de l'ensemble des tests exécutés pour le module BSP). L'automatisation a permis une réduction de l'effort d'exécution des tests de non-

régression de 10%, une réduction de 3 jours sur la période de tests de non-régression et un gain d'efficience de 98,8%. À la lumière des résultats présentés, la compagnie peut-elle se permettre de ne pas automatiser les tests de non-régressions?

APPENDICE A
EXEMPLE DE PLAN DE LIVRAISON

This document provides a high level planning for release of 11.1.

Target **full release date: ~~June 17th~~, July 15th, 2016**

Hypothesis (based on 11.0)

Pessimist velocity: 25sp/sprint (6 dev, 3 qa)

Optimist velocity: 35sp/sprint (6 dev, 3 qa)

50% of the story points will emerge in the course of the release

Summary and Release Targeted Dates

Month	Sprint	Theme and Objectives	NOTES
December	0	• Spikes • Git Migration (internal improvement) • [*Client feature n*]	No development between December 24th and January 4th Sprint 0 will starts on December 4th and ends on December 23th (This gives us 2w3d for sprint 0)
January	1,2	• [*Epic n*] • [*Internal improvement*] • [*feature n*] • [*Client feature n*] • [*Client feature n*] • [*Client feature n*]	Decision was taken to release an intermediate version ##.#.# where the main goal is to deliver [redacted] We decide to also include [*Client feature n*], [*Client feature n*] and [redacted]

			features for [redacted].
February	3,4	• [*Epic n*] • [*Feature n*] • [*Feature n*] • [*Service Feature n*] • [*Feature n*] • [*Client feature n*]	CentOS support was added [redacted].
April	7,8	• [*Feature n*] (8sp) • Epic *n features*] (~58sp) • [*Feature n*] (12sp) • [*Feature n*] (10sp) • [*Client feature n*] (8-11sp) • Bug fixes	Feature freeze should happen at the end of sprint 8 in order to let two months for the regression
May	9,10	• [*Feature n*](10sp) • [*Feature n*] • [*Feature n*] (5sp) • [*Feature n*] (21sp) • [*Feature n*] (18sp) • 64bit support (17sp) • Bug fixes	[*Module A*] feature freeze at the end of Sprint 10: (**May 20th**) Regression will also include CentOS 7.2 (and may include [redacted]) [*epic n*] was removed from the release [*epic n*] was added to the release 64bits support was added to the release

June	11,12	• Stabilization
		  • [*Module A*] Regression
		  • Bug fixes
		[*feature n*] in [*Module B*] (4sp)
		[*feature n*] (7sp)
			[*Module A*] regression starts on Sprint 11 (3 sprints)

		[service *epic n*] (17sp) Stabilization • [*Module A*] Regression • Bug fixes	[*Module B*] feature freeze at the end of Sprint 12: (**June 17th**)
July	13,14	Stabilization • [*Module A*] Regression • [*Module B*] Regression • [*Module C*], [*Module D*], [*Module E*], [*Module F*] Regressions • Bug fixes	[*Module A*] regression ends on Sprint 13 [*Module B*] regression starts and ends on Sprint 13 [*Module B*], [*Module C*], [*Module D*], [*Module E*] and [*Module F*] regressions should be completed in Sprint 14 This leads us to releasing on July 15th, 2016 (14 sprints, 434-468sp)

APPENDICE B
CATÉGORIES DE DÉVELOPPEMENT

Tableau B.1 Description des différentes catégories de développement

Élément	Description
Maintenance adaptative et évolutive	Le volet évolutif de cette catégorie consiste à développer de nouvelles fonctionnalités pour soit satisfaire de nouveaux besoins ou attirer de nouveaux clients. C'est aussi ici qu'entre le développement des innovations. Le volet adaptatif quant à lui consiste plutôt à développer des fonctionnalités permettant de s'adapter aux nouvelles technologies ou de se mettre à niveau afin de pouvoir éventuellement répondre aux futurs besoins des clients.
Maintenance préventive et corrective	Les demandes de changements dans cette catégorie sont majoritairement exprimées par les membres du département de service[22]. Ce sont des demandes visant, comme la catégorie l'indique, à réduire la probabilité de défaillances dans le logiciel mais aussi à mitiger les dommages lorsque la probabilité est rencontrée. On pourrait par exemple y retrouver un changement au niveau de la gestion des logs. Il y a aussi un volet correctif mais il ne faut pas confondre ce que fait l'équipe S.W.A.T[23], équipe complètement dédiée à la correction de bogues trouvés en production.

[22] Équipe de support à la clientèle chez Broadsign. Ils sont en contact quotidien avec les clients de l'organisation et reçoivent énormément de feedback utile afin d'améliorer le logiciel.

[23] L'équipe S.W.A.T. est une équipe distincte, ayant une mission distincte, et travaillant en mode Kanban* (voir Laurent Morisseau, *Kanban pour l'IT*, 2e éd., (: Dunod, 2014).). Les demandes de changement exprimées ici sont plutôt des bogues irritants n'étant jamais priorisés par l'équipe S.W.A.T.

Élément	Description
Développement client	Certains des clients de Broadsign contribuent financièrement au développement et l'organisation s'entend avec eux sur un budget de points à leur accorder, c'est ce qui est catégorisé ici. Pour faciliter la planification, le budget leur est fourni en début de *release*. Les clients présentent ensuite leurs carnets de produits afin que l'équipe de développement puisse leur fournir une première estimation à haut niveau. Suite à cela, les clients priorisent leurs *stories* et l'équipe estime plus précisément un nombre de *stories* suffisant pour remplir deux sprints. Le client décide ensuite, en accord avec Broadsign, les *stories* à développer dans le prochain sprint, visant idéalement 25% des *points* d'un sprint, mais qui peut être ajusté d'une semaine à l'autre. À chaque fin de sprint, une version beta[24] de l'application est livrée aux clients afin qu'ils puissent valider les nouvelles fonctionnalités demandées et retourner du *feedback* à l'équipe de développement.
Améliorations internes	Les améliorations internes concernent tout ce qui permet d'améliorer l'efficacité du processus de développement. Les priorités de développement sont complètement gérées par l'équipe et non par le *PO* uniquement. La seule contrainte imposée est de ne pas dépasser le 10% de points accordé et le *PO* décide lorsque les *stories* priorisées entreront dans un sprint.

[24] Chez Broadsign, une version beta est une version dont les fonctionnalités ont été testées mais sur laquelle il n'y a pas eu de tests de régression d'exécutés.

APPENDICE C
COÛTS DE CORRECTION DES DÉFAUTS PAR PHASE DE DÉVELOPPEMENT

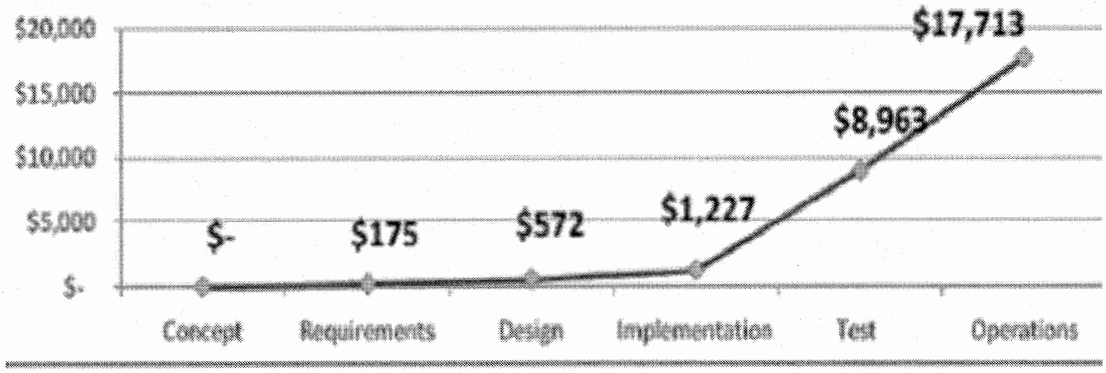


Figure C.1 Coûts de correction des défauts en dollars 2009

Tiré de https://substance.etsmtl.ca/

APPENDICE D
PLAN DE RÉALISATION DE LA PREUVE DE CONCEPT

Tableau D.1 Plan de réalisation de la preuve de concept

Jalon	**Élément de réalisation**	**Dates**
Sprint 0	• Déterminer le processus de développement • Déterminer les critères pour la priorisation • Déterminer la méthode de priorisation • Installer les outils	28 février au 3 mars 2019
Sprint 1	• Recueillir les données de test • Prioriser les tests • Estimer l'effort d'automatisation des tests	4 mars au 10 mars 2019
Sprint 2 à 10	• Planifier les tests à automatiser • Réaliser les tests planifiés • Mesurer les données de test	11 mars au 14 avril 2019
Sprint 11	• Analyser les résultats mesurés	15 avril au 21 avril 2019
Sprint 12	• Produire le bilan de projet	22 avril au 28 avril 2019

APPENDICE E
SOMMAIRE DES BESOINS EN DÉVELOPPEMENT

Tableau E.1 Besoins en développement

Besoin	**Description**
BD1	Minimiser la résistance au changement.
BD2	Faire ressortir les risques du projet rapidement.
BD3	Pouvoir m'améliorer et acquérir de l'expertise.
BD4	Assurer une uniformité avec le reste des processus de l'organisation.
BD5	Offrir une visibilité accrue, régulière et constante sur l'état d'avancement du projet.
BD6	Permettre de prédire la durée et le coût.

APPENDICE F
SOMMAIRE DES BESOINS EN AUTOMATISATION

Tableau F.1 Besoins en automatisation

	Besoin	**Description sommaire**
BA1	Pouvoir écrire des tests automatisables	Les tests devraient pouvoir être écrits par n'importe qui dans l'équipe incluant le *PO*, voire parfois les clients. L'écriture devrait donc être le plus près possible du langage naturel et elle ne devrait pas nécessiter de connaissances en programmation.
BA2	Contrôler les versions et centraliser les tests	Pouvoir suivre les changements d'une version à l'autre et de restaurer des versions antérieures au besoin. Des tests centralisés favorisent l'accessibilité à l'ensemble des parties prenantes et facilitent la maintenance.
BA3	Les tests et leurs données (effort, priorité, etc.) sont sauvegardés	Les tests doivent être sauvegardés et protégés en cas de bris logiciels ou matériels.
BA4	Automatiser l'exécution	Pouvoir en un clic déclencher l'exécution d'une série de tests. Également pouvoir facilement sélectionner un sous-ensemble de tests à exécuter. Les tests doivent également pouvoir être exécutés localement, par exemple sur le poste d'un développeur afin de ne pas engorger le système de *build* inutilement et par n'importe qui dans l'équipe incluant le *PO*. L'outil peut aussi supporter un grand volume de tests.

	Besoin	Description sommaire
BA5	Utiliser des outils populaires	L'outil offre un bon soutien technique (ex. soutien en ligne, forum, communauté en ligne, etc.). L'outil est mis à jour et corrigé rapidement en cas de problème. L'outil est facile à apprendre et la documentation est disponible et complète.
BA6	Utiliser des outils compatibles avec les outils existants	L'outil doit être compatible avec JIRA, où on doit pouvoir facilement lier une *story* avec un ou plusieurs tests. La priorité des tests peut être facilement changée afin de prioriser les tests ayant une plus grande valeur. Les tests doivent pouvoir être catégorisés facilement, par exemple par module, afin de pouvoir sélectionner des sous-ensembles de tests à exécuter.
BA7	Offre des rapports d'exécution des tests	En plus de l'automatisation, il est important de pouvoir recueillir diverses données quant à l'exécution et de façon automatisée le plus possible. Entre autres : • le nombre de tests réalisés à un moment donné de l'exécution ; • le nombre total de tests exécutés ; • le nombre de tests réussis ; • le nombre de tests échoués.

Tableau F.2 Outils sélectionnés en fonction des besoins

Outil	Description sommaire
Docker	Permet d'empaqueter l'application de test ainsi que toutes les dépendances dans un conteneur logiciel. Permet également d'abstraire l'*OS* sur lequel on travaille et facilite le travail collaboratif.
Git	Outil de contrôle de version permettant le suivi des changements entre les différentes versions des tests et permettant rapidement restaurer une version ou l'autre. C'est l'outil utilisé par l'équipe CORE.
Jenkins	Jenkins est un outil permettant d'automatiser l'exécution des tests. Il a été choisi pour sa compatibilité avec Git et Bitbucket. Il est également supporté par une large communauté et possède une documentation complète sur son site web (Jenkins community, s. d.).
JIRA	JIRA étant le principal outil utilisé par les équipes de développement de l'organisation, il a été sélectionné pour y gérer les *stories* d'automatisation et leurs données. L'idée était, en plus de les conserver dans un endroit centralisé, de pouvoir les prioriser, les estimer tout en offrant une visibilité à plus de parties prenantes possible.
PyCharm	Environnement de développement intégré spécialisé pour le développement avec le langage python.
Python	Python est un langage de programmation qui a comme principaux avantages sa rapidité d'écriture et sa capacité à intégrer différents systèmes ensemble (Python community (s. d.)).

APPENDICE G
CRITÈRES POUR LA PRIORISATION DES TESTS AUTOMATISÉS

Tableau G.1 Critères pour la priorisation des tests automatisés

Critère	**Explication**
Fréquence d'exécution	Les tests qui sont le plus exécutés sont probablement ceux qui le continueront. Des tests plus souvent exécutés sont plus faciles à maintenir et le coût d'automatisation est absorbé plus rapidement.
Complexité d'automatisation	Un test complexe à automatiser ne devrait pas figurer parmi les tests prioritaires à moins que sa complexité d'exécution manuelle soit supérieure ou que sa valeur soit immensément grande.
Complexité d'exécution manuelle	Un test étant complexe à exécuter manuellement devrait se retrouver dans la liste des tests prioritaires pour l'automatisation.
Impact d'une faute qui serait détectée par ce test	Plus l'impact d'une faute détectée par un test et grand et plus il devrait être prioritaire. Les fautes avec un grand impact sont souvent coûteuses à corriger, d'autant plus lorsque l'on est avancé dans le cycle. L'impact sur le client est aussi à considérer quoique qu'il soit difficile à mesurer en termes de coût.
Temps d'exécution d'un test manuel	Plus un test est long à exécuter et plus son automatisation amènera un gain de temps important.

APPENDICE H
MODULES TESTÉS ET LEUR NOMBRE MOYEN DE TESTS EXÉCUTÉS

Tableau H.1 Modules testés par l'équipe CORE et leur nombre moyen de tests exécutés

Module	Description	Nombre de tests exécutés
BSP	Module client.	197
BSA	Interface utilisateur	164
BSS	Serveur.	103
AV	Lecteur de contenu.	11
SDK	Interface de programmation (*API*[25]).	15
Edge	Serveur de mise en cache de contenu.	13
Total		**503**

Le tableau illustre et décrit l'ensemble des modules développés et testés par l'équipe CORE. Le nombre de tests exécutés est une moyenne des tests ayant été exécutés au cours des cinq derniers *releases*, ayant eu cours de 2014 à 2017.

[25] *Advanced programmation interface.*

APPENDICE I
PRINCIPALES COMPOSANTES DANS LESQUELLES LES BOGUES BSP SONT RAPPORTÉS

Tableau I.1 Principales composantes dans lesquelles les bogues BSP sont rapportés

Composante	**Description**
Lecteur (Playback)	Regroupe tous les bogues liés à l'affichage du contenu. Par exemple, du contenu qui vacille, qui coupe ou dont la transition à un autre contenu est saccadée ou il s'affiche du temps d'écran noir.
APIs du lecteur (player's APIs)	Regroupe les bogues liés aux différentes API disponibles du lecteur (Triggers, Conditions, Data Capture, etc.).
Preshow	Regroupe des bogues liés à la fonctionnalité Preshow ou à l'intégration avec Jnior.
Registration	Regroupe les bogues liés à l'enregistrement de la licence du lecteur.
Chromium	Regroupe tous les bogues qui sont liés au moteur de rendu Chromium. Par exemple, des bogues liés à la lecture de contenu HTML, ou encore des irrégularités introduites par des nouvelles versions de Chromium.
Monitor_sync	Regroupe les bogues liés à la technologie monitor_sync.
Monitor_disk	Regroupe les bogues liés à la technologie monitor_disk.

Composante	**Description**
Erreurs de construction (Build errors)	Regroupe les bogues causés par des problèmes au niveau du *build system*, par exemple des librairies manquantes affectant le comportement de BSP.
Systèmes d'exploitation (*Operating systems*)	Regroupe les bogues liés au système d'exploitation. Par exemple, des mises à jour de l'OS affectant notre application, des failles de sécurité devant être corrigées, etc.
Polling	Regroupe les bogues liés au « *poll* » entre le lecteur et le serveur.
SSP	Regroupe les bogues introduits par l'intégration du nouveau produit Reach (anciennement *SSP*).
Autres parties tierces (Third parties)	Regroupe les bogues des parties tierces telles que liés à Postgres, Qt, etc…

APPENDICE J
BOGUES BSP DÉCOUVERTS SELON LA PHASE DE DÉVELOPPEMENT

Tableau J.1 Bogues BSP entre 2016 et 2018 selon la phase de développement où ils sont découverts

# JIRA	Sommaire	Phase de développement
CORE-11688	assert in client_registration.exe	Implémentation
CORE-10126	Missing LD_LIBRARY_PATH export in psql client binary	Implémentation
CORE-14191	Tizen - Lecture (Playback) master does not recover from crashed NaCl until next poll	Implémentation
CORE-14274	Tizen - playlist is reset upon poll timeout even when poll hasn't changed	Implémentation
CORE-13584	Tizen - Files larger than 1GB not downloading, producing CTP2 error.	Implémentation
CORE-13325	Tizen - Incorrect data returned in BSS simple poll	Implémentation
CORE-9706	SSL handshake fails when connecting to a MEP or External POP server on Fedora 18	Implémentation
CORE-13608	Tizen - Player refuses to load Poll JSON data	Implémentation
CORE-13605	Tizen - Fix content URL mapping in SSSP4	Implémentation
CORE-11613	boost 3rdParty won't build with gcc 5.4 on ubuntu 16.04	Implémentation
CORE-13582	Tizen - 60fps Video content not playable, creates errors in Lecture (Playback) of other content.	Implémentation
CORE-13586	Tizen - Player does not Lecture (Playback) any video with a resolution higher than 1080p	Implémentation
CORE-14284	Tizen - blacklisted content logs a lot of traces	Implémentation
CORE-11599	[Jnior Cineplex] Preshow plays previously generated loop	Implémentation
CORE-10649	Preshow: Loop Generator executes and builds next loop based on Campaigns Schedules active at showtime (Instead of the time of the actual Preshow).	Production
CORE-10354	[BSP] Chromium sandbox blocks timezone detection on linux	Production

# JIRA	Sommaire	Phase de développement
CORE-10119	HTML content flickers as it loads	Production
CORE-11517	MAC addresses Filter is saved at the user account Level not the Local Machine Level	Production
CORE-10125	Black spot between creatives in external playlist is still visible	Production
CORE-12187	BSP - Make bundle randomization works for manually triggered Campaign.	Production
CORE-9716	BSP/CentOS7 : OpenGL video strategy outputs scrambled colours for video on DE45 H/W	Production
CORE-13038	Transitions with images do not work	Production
CORE-13065	Chromium touch events coordinates are not translated relative to the screen when using Orientation.	Production
CORE-12963	Flicker of previously played content between Video and HTML5 package on Windows	Production
CORE-12872	BSP fails to compute the "Last-Modified" for Creator and External Ad Copies if unrecognized date/time format at OS.	Production
CORE-12000	Add new mount point	Production
CORE-12451	BSP crashes when playing .pls streams	Production
CORE-13323	BSP under perform Chrome when rendering HTML5 content (enable on screen rendering)	Production
CORE-13347	BSP doesn't register click - touch events when using IE	Production
CORE-13884	BSP ignores Campaigns/Bundles with multiple conditions and multiple bundles in Preshow (Ubuntu only)	Production
CORE-14275	BSP continues to use Xorg even if it executes a sleep() application ad copy	Production
CORE-12775	BSP package should not provide nor depend on its bundled libraries	Production
CORE-12018	Preshow - Discarded showtime performances will count as a missed loops making bundle rotation inconsistent.	Production
CORE-10548	[BSP] Freezing in last frame when using transition (Crossfade, Star Wipe and Page Turn), if the Campaign length is longer than the External Ad Copy duration.	Production
CORE-12090	Task - script to filter out 4G dongles MAC addresses	Production

# JIRA	Sommaire	Phase de développement
CORE-11935	XMLHttpRequests Blocked by Chromium51 When Run From Local Files on Linux	Production
CORE-12333	Websocket/JSON "show_loop" only returns the current and very last item in the current loop.	Production
CORE-9787	BSP doesn't kill orphans	Production
CORE-10490	BSP: Video to HTML crossfade transition inconsistent on slave players	Production
CORE-13268	SWAT (PMV on release/12.1) - SBC Hardware acceleration and transitions	Production
CORE-13001	BSP package should not provide nor depend on its bundled libraries	Production
CORE-13565	BSP: Force renewal of serial port connection as a preventative measure	Production
CORE-10014	BSP: black spots between creatives from an external playlist	Production
CORE-13688	[SBC] - BSP under-performs when playing Video content using Video API	Production
CORE-11036	BSP: Player API "show_loop" and "now_playing" will only return one Frame's content of a multi Frame DU	Production
CORE-9811	BSP: pls stream Ad Copy causes Unexpected Shutdowns in BSP 11.1	Production
CORE-11228	back port CORE-9142 to 11.1.XX	Production
CORE-13965	Nonplaying Conditional Multibundle Campaigns prevent Preemptibles from playing	Production
CORE-11680	BSP : Missing "libxss1" will cause BSP11.1+ to install partially on Ubuntu 12 and fail to report correct version	Production
CORE-10084	Screen tearing on videos in smaller frame (CentOS)	Production
CORE-10750	BSP : Uses and abuses of the /tmp partition by BSP on Socle.	Production
CORE-10899	BSP: Crossfade from/to SWF and Ad Copies that supports transition is inconsistent with network synched players (master or slave)	Production
CORE-10789	[BSP] Slow Player API and Websockets errors	Production
CORE-12152	BSP slave will display a black screen when using external ad copies and TCP network triggers	Production

# JIRA	Sommaire	Phase de développement
CORE-9709	Intermittent error "The signature of the certificate is invalid" when using MEP or external POP with SSL	Test
CORE-9995	BSP quividi migration	Test
CORE-10268	BSP hangs when DU timezone is set to Africa/Timbuktu on Windows	Test
CORE-12411	Screen is black after Fade-in/Fade-out transition to Flash on Linux	Test
CORE-11093	Campaign with video and image displays black w/ image transition config.	Test
CORE-11449	Fix Chromium 56 build on Windows	Test
CORE-9794	No transition for HTML package ad copy on Chromium	Test
CORE-10622	Auto Upgrade/Downgrade hangs on exit in Windows 12.0 BSP	Test
CORE-11114	Update script for Android builds	Test
CORE-12077	Flicker on transition img->flash on ubuntu	Test
CORE-12051	Fix 3rdParty build of Curl	Test
CORE-11477	Audio is cut for every 2nd video with Crossfade Transition	Test
CORE-10512	Degraded resolution when Serv SSP content plays with transition	Test
CORE-10840	Setting a timezone beyond UTC on DU causes incorrect rep_per_hour calculation	Test
CORE-10279	No IANA timezones available on Android	Test
CORE-12086	[Jnior] Second preshow does not play while "Toggle movie_on during the first preshow"	Test
CORE-11516	Player does not reboot when screen reconnected if BSP restarted while off.	Test
CORE-11531	bspsh crashes upon player reboot if there is no screen connected.	Test
CORE-11760	crash_reporter hangs if bsp crashes multiple times therefore delaying the restart of bsp.	Test
CORE-12119	[Preshow] Message "Skipping system time adjustment." is not showed in bsp log if No clock skew adjustments during preshow.	Test

# JIRA	Sommaire	Phase de développement
CORE-11021	Setting no time limit on MEP bundle allows filler to play in a fully booked loop	Test
CORE-12251	BSP must initiate TLS v1.2 when connecting to the creator server over HTTPS	Test
CORE-13592	CentOS build size increase	Test
CORE-13002	configuration w/ min 12.1.6 version does not display Quividi settings	Test
CORE-12810	BSPE cannot find libprotobuf.so.15	Test
CORE-12728	screenshot is not deleted when upload fails	Test
CORE-13447	Latest linux builds go into a restart loop	Test
CORE-13603	BSP PubNub transaction error logging does not respect delay.	Test
CORE-13700	Multi Frame with Fade I/O causes bsp to hang on windows	Test
CORE-13922	video Lecture (Playback) stutters, frame drops/skips (ubuntu16)	Test
CORE-13908	bsp logs "Premium Add-On is not enabled, cannot connect to PubNub." when premium is enabled.	Test
CORE-14277	Extra protobuf libs in centos 32b package	Test
CORE-12808	SBC board outputs push_screenshot image twice	Test
CORE-13598	Build fails when Db connection is lost.	Test
CORE-11901	Message errors in bsp.log when using Chinese build.	Test
CORE-14007	[BSP] Freezing in last frame when using transition, if the Campaign length is longer than the External Ad Copy duration with Time-Based Sync.	Test
CORE-14302	[SBC] - Chromium content occasionally plays black	Test
CORE-14025	[SBC] - Stutter at the end of a video Lecture (Playback)	Test
CORE-14301	[SBC] - HTML5 Package briefly visible after fade in / fade out	Test
CORE-13319	Chromium does not play HTML content on windows	Test
CORE-13806	Missing Roboto font license	Test
CORE-9955	BSP chromium black frame	Test
CORE-11640	Gestures on Touchscreen slow on Google Maps	Test

# JIRA	Sommaire	Phase de développement
CORE-12079	Preemptive campaign not taking precedence over filler	Test
CORE-12314	Polipo is not installed when dedicating a new player	Test
CORE-12071	Dupe mac with identical secondary mac	Test
CORE-11797	CentOS 32 bits support - post-install script fails on "chown: cannot access '/opt/broadsign/suite/bsp/bin/chrome-sandbox': No such file or directory"	Test
CORE-12262	Slave does not hold last frame	Test
CORE-12242	Screen is black during html Lecture (Playback) after transition flash -> html on windows with chromium	Test
CORE-12165	SSL error: The signature of the certificate is invalid	Test
CORE-12241	Flicker of previously played image between video and html on windows	Test
CORE-12459	HTML packages are skipped and bsp logs "Error while playing ad copy id" in the logs.	Test
CORE-12457	"content was not completely pre-buffered before being displayed" message are logged every time a html content is played.	Test
CORE-12428	Crash when pre-buffered page is not ready on Windows non-fullscreen	Test
CORE-12396	Ubuntu: SSP Preshow holds frame on video after preshow end	Test
CORE-12440	SSP Master-Slave, HTML.zip files are skipped when SSP skips	Test
CORE-9944	BSP installer WAAAAAAAAAAY too huge (163 MB)	Test
CORE-12700	BSPE Test app log messages don't appear in release builds	Test
CORE-10571	Autrichian Player Debugging	Test
CORE-10283	Lecture (Playback) not scaled with external playlist on Linux	Test
CORE-10365	Automate and include the new "date_time_zonespec.csv" when releasing new BSP packages	Test
CORE-13083	Domain attribute for Premium Add-On is not checked for existing BSP	Test

# JIRA	Sommaire	Phase de développement
CORE-10580	Last frame of a video remains displayed on video to html transition with Chromium Browser and network triggers.	Test
CORE-10579	When using monitor external playlist; applying mac filtering does not clear the external.db	Test
CORE-13499	Fix builds to include debug info	Test
CORE-10879	xpress outputs "Powering off screen for stand-by" when turning screen on	Test
CORE-10251	Remove Crash Reporter	Test
CORE-13591	Pass Remote Control configuration to provider_feed_script	Test
CORE-13579	Chromium is packaged in CentOS32	Test
CORE-11246	Builds fail on clean command	Test
CORE-11211	[BSP] Android build boots to black screen	Test
CORE-11805	CentOS 32 bits support - Lecture (Playback) performance issues.	Test
CORE-13046	windows self registration menu appears behind bsp	Test
CORE-13926	CentOS build size 310MB	Test
CORE-13892	self-registration window missing new icon	Test
CORE-14096	players fail to poll due to missing lib reference	Test
CORE-11622	Cannot see available resources for player self registration	Test
CORE-14149	Image Ad Copy plays black after Flash and HTML content when using image transitions.	Test
CORE-14198	BSP does not play HTTPS content with proxy enabled.	Test
CORE-14063	Pubnub transaction consumption	Test
CORE-11758	Retrieving flash / javascript variable with MEP creatives fails	Test
CORE-11756	CentOS 32 bits support - cef*.pak should not be included in the rpm package.	Test
CORE-14311	Default of 13.0.0 Configuration should be Creator instead of Publish	Test
CORE-11763	BSP is unresponsive while fetching all criteria from its domain	Test

# JIRA	Sommaire	Phase de développement
CORE-11619	Transparent HTML ad copy is not displayed on CentOS with Compton	Test
CORE-12038	Chromium - Keyboard tab key does not work in BSP full screen mode	Test
CORE-12085	[Jnior] Preshow restart playing after end_preshow while "Toggle movie_on during the pre_show"	Test
CORE-12058	BSP querying wrong creator URL	Test

APPENDICE K
ENSEMBLE DES TESTS DU MODULE BSP

Tableau K.1 Ensemble des tests du module BSP

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
292	Obeying timezone from DU V	10	n/d	
730	Honoring Timezone	9	n/d	
501	Crossfade transitions with duration under 1 second	41	2	3
5649	Chromium Transition Endurance	4	n/d	
5644	Gesture Support For Chromium	6	n/d	
5407	Validation of db_log_sink and filesink parameters	5	n/d	1
4276	Monitor_remote for a web socket server	40	2	5
5686	Web redirect playback with Chromium using proxy	3	n/d	
124	All Media Types test	39	0,5	2
4882	Android: Ensure apps not included in image	11	n/d	
442	Obeying timezone from DU VI	10	n/d	
288	Obeying timezone from DU III	8	n/d	
1719	No Debug References	8	n/d	
2108	Obeying timezone from DU when GMT support DST	8	n/d	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4873	Proxy exceptions: BSP	8	n/d	
5343	JavaScript Blob in Chromium	7	n/d	
261	Multiple frames with audio	33	2	3
2347	Correct display of timezone	6	n/d	
2834	Polling with timezone changed.	6	n/d	
5034	Playing HTML with flash content using Chromium Browser	5	n/d	
5754	Scrollbar flag in Chromium	5	n/d	3
2234	Http and network errors skipped	30	2	5
5406	No Windows error dialog boxes for Chromium crashes	4	n/d	2
5421	Chromium cache (Browsing in offline mode)	4	n/d	3
5959	Stability during playback with Chromium	4	n/d	
4487	USB rescue mode: Restore Player Triggers	3	n/d	
5342	Chromium remote debugging	3	n/d	
5465	Chromium Flags	3	n/d	1
5516	Ad Copy Commands	3	n/d	
5517	Ad Copy Commands with Network Triggers	3	n/d	
5581	Proxy And Proxy exceptions: Chromium Browser	3	n/d	
5925	No Flashing White Page in Chromium before content plays	3	n/d	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
5272	Secondary Category separation	26	3	3
4389	Content playback with custom geometry and OpenGL	25	3	2
5128	Chromium Browser Support - HTML5 content	1	n/d	
5810	Monitor External Playlist: Parsing playlist_url protocol	1	n/d	
5768	/etc/broadsign/bsp script new location	16	n/d	
381	Obeying timezone from DU - Device control time spans	14	n/d	
4878	Android: System UI error pop up box	11	n/d	
212	Moving hosts between display units	25	2	2
4495	Schedule date range	21	3	
277	Obeying timezone from DU I	9	n/d	
4932	Android - Shutdown without user interaction	9	n/d	
2387	Clear database	8	n/d	2
276	Poll timestamps when using timezone from DU	7	n/d	
4397	Resuming playback when incidents disabled	20	2	5
4509	Events: content looping	7	n/d	
4410	Fullscreen on Multiscreen	17	8	inf
2368	Remote reboot	6	n/d	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4623	Bundle Duration and Filler	17	1	
5417	Logging - Rotation by days	6	n/d	
5641	Web Content Window Transparency with Chromium	6	n/d	
4025	Program Schedule respects timezone	5	n/d	
4346	USB rescue mode: restore player	5	n/d	
4847	Logging on system upgrades	5	n/d	
5371	Creator loop playback order with over-saturation	5	n/d	
5408	Compression of logs	5	n/d	2
5416	Logging - Rotation by log size	5	n/d	2
5420	Logging - Archived files kept	5	n/d	2
3131	Timestamp value when custom timezone is set	4	n/d	
5403	Native Interactivity	16	n/d	3
5468	Chromium Version in Field Report	4	n/d	1
5698	Geotargeting support for Chromium Browser	4	n/d	
5961	Chromium gapless playback	4	n/d	
3115	Timestamp value when no custom timezone is set	3	n/d	
4703	Invoking BroadSignPlay()	3	n/d	
5349	Flash Ad Copy with Remote Action Stop	3	n/d	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4386	Heavy flash playback on Windows	16	3	20
5384	Set proper environment variables in linux cmd-line scripts	3	n/d	
5399	Geo Targeting - Condition Schedule and Geofencing	3	n/d	
5415	CEF product version	3	n/d	
505	BSP on multi head Linux players	16	2	3
4465	Interactivity with transparent flash and custom geometry	15	5	20
5732	Under-saturation with Vistar filler	3	n/d	
5812	BSP install on CentOS with no bsp user home directory	3	n/d	
5847	BSP does not discard Creator contents upon restart	3	n/d	
4890	Vistar remnant ads – IV	2	n/d	
5383	Share of Loop Special cases I	2	n/d	
5387	Android: certificates folder not present	2	n/d	
4629	Flash performance	15	3	5
507	Secondary separation collision	15	2	3
5398	Geo Targeting - Activate/deactivate conditions based on external script	2	n/d	
5527	Chromium Dev Tools	2	n/d	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
5575	Stats reported on slave player with video synchronization	2	n/d	
5705	Ignoring day parts with a future start date	2	n/d	
5829	Monitor External Playlist: Midnight Behavior	2	n/d	
5842	BSP hides windows taskbar after playing application ad copy	2	n/d	
5856	Monitor External Playlist: Transform External Playlist	2	n/d	
5859	Monitor External Playlist: Reporting POP Stat	2	n/d	
5907	Monitor External Playlist: Padding	2	n/d	
4630	BSP process on Transitions	15	1	3
5375	Share of Loop conditional campaigns with transform removed	1	n/d	
5389	Android: network self healing with static IP	1	n/d	
5402	Geo Targeting - Activate/deactivate conditions based on os location	1	n/d	
2359	Video and Audio Playback	15	1	
5582	Expiration and update of external Ad Copy with synchronization	1	n/d	
5634	Independently update chromium on BSP 11.1 and higher	1	n/d	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
5813	Monitor External Playlist: parameter validation	1	n/d	
5816	Monitor External Playlist: fetching behaviour	1	n/d	
5817	Monitor External Playlist: fetching connection timout and https connections	1	n/d	
5821	Monitor External Playlist: pending fetches	1	n/d	
5827	Monitor External Playlist: playlist caching	1	n/d	
5828	Monitor External Playlist: no caching on error	1	n/d	
5846	Monitor External Playlist: Creative Caching	1	n/d	
5854	Monitor External Playlist: transform_external_playlist with oversaturation and undersaturation	1	n/d	
5858	Monitor External Playlist: Segmentation & Seperation	1	n/d	
5861	Monitor External Playlist: filler, overriding bundle slot and fully booked loops	1	n/d	
5897	Monitor External Playlist: Hold last frame	1	n/d	
5909	Monitor External Playlist: Padding duration and min_threshold	1	n/d	
5929	No gap playback for multistage monitor_showtime	1	n/d	
4411	HTML does not plays on top of another Bundle that uses Full	15	1	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
	Screen Custom Geometry on the same Z-layer			
6011	Dynamic Feed: Creating a Feed Message	1	n/d	
6109	External POP - Verbose Mode	1	n/d	
6110	External POP - Report upon playback completion	1	n/d	
6111	External POP - Error handling	1	n/d	
6113	External POP - Periodic report	1	n/d	
6118	External POP - Non-Overlap behavior	1	n/d	
6119	External POP with Monitor External Playlist	1	n/d	
6120	External POP - Max POPs per request	1	n/d	
6123	External POP - HTTPS connection	1	n/d	
6126	No BSP deadlock when DST changes occur at 12AM	1	n/d	
6129	No clock skew adjustments during preshow.	1	n/d	
6137	Touch event recognized on triggered flash ad copy	1	n/d	
4625	Unique ad copy filler rule: time and day of filler	15	0	
4829	Unique ad copy filler rule: respect campaign saturation	14	5	3
4412	Flash Playback with Internet Explorer	14	2	inf

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4624	HTML recovers after being covered by frame with customize-geometry	14	0	
6172	SSP - Validate Communication Packets	1	n/d	
6175	SSP - Proof of Play	1	n/d	
6176	Under-saturation with SSP filler	1	n/d	
6177	SSP Geometry override	1	n/d	
6179	SSP - AdRequest Throttling	1	n/d	
4388	OpenGL video playback monitoring on DE45 H/W	13	5	inf
506	BroadSign SSP - Playback with Vistar	13	2	3
6184	Share UDP port binding	1	n/d	
6191	External Saturation with Preemptive SSP Campaign	1	n/d	
6199	Next Expected Poll with "Let player to set system time"	1	n/d	
6201	Chromium - Keyboard and mouse support	1	n/d	
6202	Dynamic Pacing Campaign stops after total plays reached	1	n/d	
6204	Dynamic Pacing player MIA	1	n/d	
6206	Dynamic Pacing - Separation and secondary Separation	1	n/d	
6207	Dynamic Pacing – segmentation	1	n/d	
1353	Suspend/Resume screen(s) with Blocking day part	13	0	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
2354	BroadSign SSP Transition	13	0	
6226	Crash Reporter GUI disabled on JCD OS	1	n/d	
6227	rpm postinstall installation argumenst	1	n/d	
4626	replacing bundle content	13	0	
5526	Chromium Browser WebCam Support	0	n/d	
4628	Transitions between Images	13	0	
4335	Player system volume and Ad copy volume	12	3	8
1807	Clean install	25	8	
4422	Monitor_showtime: Preshow interruption with JNIOR	7	8	
5250	Monitor_showtime: Multi-stage preshow interruption	3	8	
5317	Share of loop with exceptions	3	8	
4475	Gapless playback and Video Transitions	12	2	
2355	Trigger Synchronization: Video Sync on Multiple Systems II	12	0	
2356	Honouring opening hours	12	0	
5251	External Ad Copy Sync	8	5	
5116	Preshow max duration override	4	5	
4482	Atomic/Progressive FTP Folder Sync Interruption	17	5	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
2357	Trigger Synchronization: Synchronization Sets	12	0	
4427	Proxy Exceptions: External Ad Copies	13	5	
4561	Active popup prevention	8	5	
5165	Application Ad copy as Admin User	4	5	
5359	Linux Crash Report Submittal	4	5	
5119	Preshow max duration override on Preempltible Campaigns	3	5	
5114	Creator Message Synchronization - isReady	2	5	
4463	Dependencies Requirements	19	5	
4476	Monitor Sync: If-Modified-Since Support	16	5	
4768	External Triggers and Play Next	10	5	
4421	Monitor_showtime: Preshow with JNIOR	3	5	
5006	Quividi - Data Collection	3	5	
5360	Xpress Crash Report Submittal	3	5	
4627	Applications as Ad Copies - playback	12	0	
5248	Monitor_showtime: Multi-stage preshow with JNIOR	2	5	
5320	Share of Loop with preemptive campaigns	1	5	
5345	Share of Loop with overbooking and carrying over time	1	5	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4830	Screen Orientation	11	3	3
4305	Remote Geometry	11	2	3
2361	Flash transitions	11	2	20
2358	Triggering Transitions on various content types	11	1	
4344	External AdCopy Updates	22	3	
2353	Transitions and Triggers	11	0	
4309	Pushing Flash Upgrades / Downgrades	8	3	
119	Transitions and Multi-head Triggers	10	n/d	
4900	Simple Screen Control	10	3	
4538	Multicast synchronization: Network Trigger Variations	10	3	
4414	Performance and stability	10	2	20
4186	Monitor Sync with Authentication	22	3	
287	Obeying timezone from DU II	16	3	
2362	Trigger Synchronization: Video Sync on Multiple Frames	10	2	
5035	Automatic Retargeting VII	10	1	
215	No stat reported on interrupted content	13	3	
5218	Player poll after DST change	13	3	
4963	Trigger tool encryption and password	12	3	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4436	Transitions with enforced Resolutions	10	0	
4387	Scheduled Condition	10	0	
2360	Remote Hardware Acceleration	10	0	
2364	Scheduled Condition Offset	10	0	
4858	Improved slot rotation controls - Bundle ordering	10	0	
156	Partially booked filter on Generated Frames	9	n/d	
5278	Application Ad copy working path is correct	5	3	
5347	Sigterm / Sigkill configurable for application as adcopy	4	3	
5331	xorg memory leak	3	3	
4792	Under-saturation Offset with Pre-emptible Campaigns	9	3	
4479	No Firewall Exceptions for BSP/BSES process	38	3	
5162	TLS Version Verification	36	3	
4490	Atomic/Progressive FTP Folder Sync: FTP updates	25	3	
4413	Partial Booking	9	3	
4481	Atomic/Progressive FTP Folder Sync	14	3	2
4569	Invoke stop API to Flash	11	3	
4604	Network synchronization and scheduled conditions - I	9	3	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4791	Loop Query on multiple frames	9	3	
4776	Campaign Interactivity	9	3	
4910	Atomic/Progressive FTP Folder Sync Config Profile Version	8	3	
5178	Custom Geometry - V – Playback	9	2	
415	Loop Query Output	9	1	
2874	Frame Interactivity	9	0	
3973	Manual Adjustment Transform and multiple bundles	8	5	
4770	Play Next timeout	5	3	
5101	Creator Message Synchronization	5	3	
5276	Reboot if system Nand is unmounted	5	3	
4902	External Saturation Schedule - II (Refresh Rate & Sync)	8	3	
4835	DIAPI - No playback when condition not met	8	3	
4500	Improved slot rotation controls - Bundle percent share with multiple campaigns	8	3	
5362	Stats on interrupted content with stat_completion_percentage	3	3	
5080	Monitor_showtime: Preshow with JNIOR over RS232	2	3	
5123	Creator Message Synchronization - isReady and Backup Master	2	3	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4611	Share of Loop with Triggered content	8	3	
4728	Under-Saturation Spacing	8	3	
5112	Creator Synchronization and Multiframes	1	3	
5249	Monitor_showtime: Modifying multi-stage stage_align_to_showtime	1	3	
5170	Share of Loop Underbooked with filler/programs	8	2	
4948	Share of Loop with over and under saturation	8	2	
4908	Share of Loop with Separation and Segmentation	8	1	
4832	Share of Loop with conditional campaigns	8	0	
4901	External Saturation Schedule - IV (Value Errors)	8	0	
4839	MAC Filtering	38	2	
4464	Playback upon network disconnect	8	0	
4294	Inserting Media Layer	40	2	
4361	Emergency Messaging	61	2	
2326	Scheduled Condition multiple day parts and opening hours	7	n/d	
454	Access to http sites	17	2	
4944	Time Synchronization	7	3	
5144	HTML Package content	7	2	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
2248	Caching flash_web_adapter content	6	n/d	
5084	Upgrade / downgrade BSP while playing different media types	23	2	
4565	Enforce Variables (Javascript)	19	2	
2113	Transitions between different media types	6	n/d	
2214	Local time utc offset provided for field report	12	2	
4458	Transitions and aspect ratio	6	n/d	
4794	HTML Allow local file access	6	3	
4864	Inserting Media Layer	7	2	
1804	File Structure and Configuration Differences	56	2	
4532	Emergency Messaging and Orientation	28	2	
4470	Upgrades while disconnected from Monitor	14	2	
5092	Player self registration	5	2	
5100	Player self registration – Scoping	5	2	
4619	Prebuffer delay per mime type	6	2	
149	DB Pickup	30	2	1
4158	Stats on Triggered Playback	25	2	
2861	Playlist under-saturation (transform_under_saturate)	6	2	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
3986	Appending all variables to an .htm file	14	2	
4340	Appended HTML id with Monitor Sync in Config Profile	13	2	
4777	Over and Under-saturation with same Category II	6	2	
4549	External Ad Copies: Expiry and Fillller	10	2	
4909	Monitor Remote Encryption and Security - III	10	2	
5364	Custom Geometry - VI - Playback with Z order	6	1	
5138	Chromium Cache Updates	8	2	
4539	External Ad Copies: used as slave content with network triggers	7	2	
5240	BSP Retries to Connect to Multicast Grou	5	2	
5104	Creator Synchronization and Trigger Features	3	2	
391	Disk space incident on content still needed	32	2	
4536	Fullscreen bundle	6	0	
4969	Playlist over-saturation (transform_over_saturate)	5	3	
4334	Advanced Configuration	37	2	
5171	Reservations and campaigns	5	3	
4544	Resume Default loop schedule	5	3	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4748	Conditions using remote_action utility	5	3	
5235	Under-saturation with loop segmentation	5	2	
4965	Under-saturation with category separation	5	0	
5021	Playback of multiple triggered content in a Slave frame	5	0	
5386	Frame synchronization on an unlimited bundle in a Slave frame	4	n/d	8
4011	Broadcasting triggers over a network	9	2	
4959	Campaign with multiple schedules on BSP with local time that differs from UTC	9	2	
4984	Append Location Code to Monitor Sync FTP Path	9	2	
5223	BSP startup time	9	2	
4556	Feed Expiry and Timezones	7	2	
5404	Triggering slave bundle when category duration = 0	4	n/d	
4502	Network synchronization and scheduled conditions - II	4	n/d	
5252	Appends variables to first request only in Chromium	6	2	
4552	External Ad Copies: used in an event	5	2	
4602	Enforce Condition	5	2	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
5351	Improved slot rotation controls - Manual Transformation	4	8	
4905	Improved slot rotation controls - Triggered content	4	3	
5236	Poll Retry	4	2	
5202	Improved slot rotation controls - Bundle percent share with multiple schedules	4	3	
4972	Improved slot rotation controls - Switching Rotation mode	4	2	
4973	Improved slot rotation controls - Segment and Separator transformations	4	2	
4868	Fullscreen on monitor disconnect / reconnect	4	1	
5028	External Under-Saturation Offsets	4	0	
5405	BSP continues to run while disconnected from Monitor	3	n/d	
5097	Player self registration – Options	2	2	
5277	Android system bar	2	2	
5353	Registration GUI on existing Players	2	2	
5374	Under-Saturation and Conditions	3	n/d	
5346	Share of Loop bundle durations	1	2	
5576	Flash-Video Transitions	3	n/d	
3941	Executing the dedication script	59	1	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
5721	Respecting playlist playback order	3	n/d	
4966	Playing HTML with video content	3	3	
4967	Automatic Retargeting I	3	2	
4393	Screensaver inactive on BSP	28	1	
1970	monitor_sync on secure https	10	1	
4916	Show loop API	10	1	
5036	Chromium Browser Support - BSP process	9	1	
4457	Events: Event playback	8	1	
1905	Installation wizard	38	1	
136	Missing Ad Copy incident stress test	35	1	
4874	Android: BSP Installed packages	20	1	
4877	Android: BSP startup	20	1	
513	Upgrade channel active check	15	1	
4970	Frame synchronization interruptions	3	2	
4971	Slave content triggered after changes to trigger category	3	2	
5336	Playback priority of triggered content	3	2	
5356	Playback endurance	3	2	
5769	Monitor Remote Encryption and Security - II	3	2	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
5059	Creator content preview in Administrator - Filler	8	1	
4833	Broadcast triggers using trigger.exe	3	0	
4912	Monitor Sync recover when sync folder removed	5	1	
5042	Chromium Browser Support - HTML Fills frame	5	1	
4618	External Saturation Schedule – I	3	0	
5357	Under-saturation spacing with creator	0	1	
4977	Deactivated bundles stop playing	2	n/d	
4871	About shows correct version	37	0,5	
2089	ca_cert's expiry date	56	0,5	
4881	Options menu	41	0,5	
190	Push and clear requests	35	0,5	
4875	Android: Hide the system UI	20	0,5	
4876	Android - Sytem Orientation kept	18	0,5	
5418	Enforced Resolution	2	n/d	
5393	Triggers and Video Transitions	2	n/d	
5395	Flash Transitions with various content	2	n/d	
5127	Android: O/S Firmware Upgrade	20	0	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4978	Trigger Synchronization: Video Sync on Multiple Systems I	2	5	
227	Timezone in field report	17	0	
291	Obeying timezone from DU IV	12	0	
278	Obeying timezone from DU - Network controls	11	0	
5358	Slave bundles with multiple master triggers	2	3	
4775	External Triggers and Play Next on Multiframe layouts	9	0	
5323	Multicast synchronization	2	3	
4709	TCP triggering	4	0	
5113	Creator Synchronization and Emergency Messaging	4	0	
5348	Video Transitions and Multi-head	2	3	
4480	Restricted Telnet access to BSP/BSES	40	0	
5354	Automatic Retargeting III	2	2	
4913	Automatic Retargeting III	1	n/d	
5489	Removing frames from an existing reservation	1	n/d	
5982	Playback in a Master frame	1	n/d	
6145	Automatic Retargeting IV	1	n/d	
6146	Automatic Retargeting VII	1	n/d	
6158	Playback in a None frame	1	n/d	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
6161	Playing content from a Reservation in a Slave and Master Frame	1	n/d	
6180	Playing content from a Reservation in a Slave and None Frame	1	n/d	
6182	Automatic Retargeting V	1	n/d	
6211	Playback without a Master frame	1	n/d	
6215	Frame synchronization accuracy	1	n/d	
6228	Playback when a Master frame contains an unlimited bundle duration	1	n/d	
5321	Offline mode exception for http server	1	3	
4790	External Incidents – II	9	0	
5322	Trigger Synchronization: Backup Master	1	3	
5181	Hardware watchdog	8	0	
4869	Monitor Remote Encryption and Security - I	6	0	
5326	Improved slot rotation controls - Bundle percent share	1	3	
5327	Rebookable Campaigns – Targeting	1	3	
4553	External Ad Copies: used in a bundle with multiple Ad Copies	4	0	
4554	External Ad Copies: direct-to-slot scheduling	4	0	

Numéro de test	Description	Fréquence d'exécution	Effort manuel (*sp*)	Effort d'automatisation (*sp*)
4769	External Triggers on Backup Master	4	0	
4845	RS-232 API - Remote action device operations	4	0	
4550	External Ad Copies: used with multiframe triggers	3	2	13
4551	External Ad Copies: used in a program	3	0	
4981	Applications as Ad Copies – triggers	1	0	
5005	Quividi - Dynamic Conditions	1	n/d	
5007	Quividi - Data Collection with changing viewer count	1	0	
5009	Quividi – Timeout	1	0	
4983	Applications as Ad Copies - fullscreen	0	0	
5016	RS-232 API - Custom actions and external devices - I	0	0	
5017	RS-232 API - Custom actions and external devices - II	0	0	
5193	Creator migration after Upgrade	0	0	
2239	DST calculation for timezone Greenwich Mean Time	8	0	
4907	LSR: Multi-Bunlde Campaigns with Conditions	0	0	

APPENDICE L
BOGUES DÉCOUVERTS PENDANT LE PROJET D'AUTOMATISATION

Tableau L.1 Bogues découverts pendant le projet d'automatisation

Date	Référence	Description	Statut
2019-03-16	INC0016781	Can't access testopia.broadsign.net anymore since new VPN	Corrigé
2019-03-19	DOC-611	Documentation: (swagger) should specify that premium poll is only available with Broadsign Live	Corrigé
2019-03-20	DOC-612	Documentation: Missing network controls section of poll	Corrigé
2019-03-22	CORE-15897	Premium poll & screenshots should not be blocked by maximum poll limit	Ouvert
2019-03-23	DOC-613	Documentation: Incorrect relational schema	Corrigé
2019-04-21	CORE-15923	"method" option missing for Screenshot Remote Action	Ouvert
2019-04-03	CORE-15966	REST: resource_query/v3/query_by_id doesn't work	Ouvert
2019-04-04	CORE-15983	Invalid error message on REST upload fail (content/v11/add)	Ouvert
2019-04-04	DOC-610	Documentation: REST use case to add player to DU not up-to-date	Ouvert
2019-04-05	CORE-15808	OS volume is muted when closing BSA	Ouvert
2019-04-05	CORE-16109	BSA times with a "Connection failure" error out even if no expiration time is set	Ouvert

BIBLIOGRAPHIE

Alégroth, E., Feldt, R. et Olsson, H. H. (2013). *Transitioning manual system test suites to automated testing: An industrial case study* IEEE.

Amannejad, Y., Garousi, V., Irving, R. et Sahaf, Z. (2014). *A search-based approach for cost-effective software test automation decision support and an industrial case study* IEEE.

April, A. et Laporte, C. (2011). *L'assurance qualité logicielle 1 : concepts de base* (vol. 1). Paris :.

Aranha, E. et Borba, P. (2007). Test effort estimation models based on test specifications. *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007*, 67-71. doi: 10.1109/TAIC.PART.2007.29

Atlassian. (© 2017). *JIRA Software*. Récupéré de https://fr.atlassian.com/software/jira

Bach, J., Pettichord, B. et Kaner, C. (2002). *Lessons learned in software testing: a context-driven approach* John Wiley & Sons.

Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (© 2001). *Principes sous-jacents au manifeste*. Récupéré de http://agilemanifesto.org/iso/fr/principles.html

Berner, S., Weber, R. et Keller, R. K. (2005). *Observations and lessons learned from automated testing* IEEE.

Boisvert, M. et Trudel, S. (2011a). Gestion de projet. Dans *Choisir l'agilité: du développement logiciel à la gouvernance* (chap. 6, p. 100-104). Dunod.

Boisvert, M. et Trudel, S. (2011b). Gestion des individus. Dans *Choisir l'agilité: du développement logiciel à la gouvernance* (chap. 17, p. 277-282).

Broadsign. (2015). Charter - Version 11.1.

Broadsign. (2016, 21 avril 2019). *Git usage at Broadsign - Development - Atlas-Confluence*. Récupéré le 21 avril 2019 de https://atlas01.broadsign.com/confluence/x/BQBp

Broadsign. (2017a). *Broadsign | Linkedin*. Récupéré le 21 mars 2017 de https://www.linkedin.com/company/broadsign

Broadsign. (2017b). *Number of manual test cases*. Récupéré le 21 mars 2017 de https://testopia.broadsign.net/tr_list_cases.cgi?current_tab=case&case_status_id=2&product_id=1&product_id=16&product_id=22&product_id=4&product_id=2&product_id=3&product_id=5&product_id=17&product_id=14&product_id=8&product_id=25&author_type=substring&author=&andor=0&default_tester_type=substring&default_tester=&summary_type=allwordssubstr&summary=&tcaction_type=allwordssubstr&tcaction=&tceffect_type=allwordssubstr&tceffect=&script_type=allwordssubstr&script=&requirement_type=allwordssubstr&requirement=&tags_type=anyexact&tags=&case_id=&plan_id=&run_id=&limit=25&qname=%23%20of%20confirmed%20test%20cases%20for%20BSP%2C%20BSA%2C%20BSS%2C%20BSSDK%2C%20BSES%2C%20AV%2C%20BMB%2C%20BM%20Mobile%2C%20BSUM%2C%20Crash%20Reporter%20and%20Feature&qname=Number%20of%20manual%20test%20cases

BroadSign. (s. d.). *Who We Are - BroadSign Cloud-based Digital Signage Software*. Récupéré le 21 mars 2017 de https://broadsign.com/who-we-are/

Coram, M. et Bohner, S. (2005). *The impact of agile methods on software project management* IEEE.

Crispin, L. et Gregory, J. (2009). *Agile testing: A practical guide for testers and agile teams* Pearson Education.

Elbaum, S., Rothermel, G., Kanduri, S. et Malishevsky, A. G. (2004). Selecting a cost-effective test case prioritization technique. *Software Quality Journal*, *12*(3), 185-210.

Fuentetaja, E. et Hendricks, G. (2013). *Testopia*. Récupéré le 21 mars 2017 de https://sourceforge.net/projects/testopia/

Garousi, V. et Mäntylä, M. V. (2016). When and what to automate in software testing? A multi-vocal literature review. *Information and Software Technology*, *76*, 92-117.

Grabinski, L. et O'Hare, J. (2014). How to Implement Efficient Test Automation in an Agile Project. *Agile Business Conference*. Récupéré de http://www.agileconference.org/wp-content/uploads/2014/10/How-to-Implement-Efficient-Test-Automation-on-an-Agile-Project-Lukasz-Grabinski-John-OHare.pdf

Graham, D. (2013). *Test automatisation patterns*. Récupéré le 12 août 2017 de http://www.dorothygraham.co.uk/patterns/desPatterns/index.html

Hoffman, D. (1999). Cost benefits analysis of test automation. *STAR West*, *99*.

Jiang, B., Zhang, Z., Chan, W. K. et Tse, T. (2009). *Adaptive random test case prioritization* IEEE Computer Society.

Kaner, C. et Fiedler, R. L. (2013). *Foundations of Software Testing* Context-Driven Press.

Laapas, A. (2014). Cost-benefit analysis of using test automation in the development of embedded software.

Lehu, J.-M. (2012). *L'encyclopédie du marketing : [commentée & illustrée]* (2e éd.).

Luo, J. (2018). *Regression Data*. Récupéré le 10 février 2019 de https://docs.google.com/spreadsheets/d/1hPyo7XWEvxR7FwdU4qBbuLHSRkwQ-ZT5FUpsRqf3EPA/edit?usp=sharing

Morisseau, L. (2014). *Kanban pour l'IT* (2e éd.) Dunod.

Persson, C. et Yilmazturk, N. (2004). *Establishment of automated regression testing at ABB: industrial experience report on'avoiding the pitfalls'*

Pettichord, B. (1996). *Success with test automation*

Publicité. (2018, 28 juin, 12 h 38). *Dans Wikipédia, l'encyclopédie libre*. Récupéré le 26 août 2018 de https://fr.wikipedia.org/wiki/Publicit%C3%A9

Ramler, R. et Wolfmaier, K. (2006). Economic perspectives in test automation: balancing automated and manual testing with opportunity cost. *International Conference on Software Engineering* (p. 85-91). ACM.

Rice, R. W., CSQA, C. et Rice Consulting Solutions, L. (2003). Surviving the top ten challenges of software test automation. *CrossTalk: The Journal of Defense Software Engineering*, 26-29.

Rothermel, G. et Harrold, M. J. (1997). A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *6*(2), 173-210.

Sampat, N. et Jamwal, A. (2007). *Test automation for effective post deployement testing*. Récupéré le 12 août 2017 de http://docshare01.docshare.tips/files/24682/246825158.pdf

Schneider, W. E. (1994). *The reengineering alternative: A plan for making your current culture work.*

Schwaber, K. et Sutherland, J. (2016). *The Scrum Guide™*. Récupéré de http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf

Shihab, E., Jiang, Z. M., Adams, B., Hassan, A. E. et Bowerman, R. (2011). Prioritizing the creation of unit tests in legacy software systems. *Software: Practice and Experience*, *41*(10), 1027-1048.

Singh, P. K., Sangwan, O. et Sharma, A. (2013). *A systematic review on fault based mutation testing techniques and tools for aspect-j programs* IEEE.

Software, S. (2007). *Automated Testing Best Practices*. Récupéré

SonarQube. (2019, 27 avril, 22 h 55 3 janvier 2019). *Dans Wikipédia, l'encyclopédie libre*. Récupéré le 27 avril 2019 de https://fr.wikipedia.org/wiki/SonarQube

Stobie, K. (2009). *Too much automation or not enough? When to automate testing*

Yang, L. (2016). *Factors to Consider When Implementing Automated Software Testing*. SPAWAR Systems Center Atlantic Charleston United States.