ÉCOLE IMIE-PARIS

MÉMOIRE DE FIN D'ETUDE

DIPLÔME PRÉPARÉ: MANAGER DE SOLUTION DIGITALES ET DATA

TITRE DU MÉMOIRE:

MISE EN PLACE D'UNE SOLUTION D'AUTOMATISATION DES TESTS IHM

PRÉSENTE PAR :
DIA MOUHAMADOU MOUSTAPHA

REPRÉSENTANT ENEDIS:
LOISELET MATTHIEU

ENCADREUR:
MOHAMED AMINE EL AFRIT

2022/2023

Remerciement

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce mémoire.

Tout d'abord, je souhaite remercier le Bon Dieu pour sa grâce et sa guidance tout au long de ce parcours académique. Sans sa bénédiction et sa direction, cette réalisation n'aurait pas été possible.

Je tiens à exprimer ma reconnaissance envers mes chers parents pour leur amour inconditionnel, leur soutien indéfectible et leurs sacrifices. Leurs encouragements constants ont été ma source d'inspiration.

Un grand merci à mes tuteurs de formation, Jean Samuel ALLAIN, Alexandru SIRBU et Arnaud MONET, pour leur expertise, leurs conseils avisés et leur précieuse orientation tout au long de mes études. Leurs enseignements ont été une source de connaissance inestimable.

Je n'oublie pas de saluer chaleureusement mes collègues de travail, dont la collaboration et l'échange d'idées ont enrichi ma compréhension des concepts étudiés.

Mes amis de longue date, Limamou DIOP, Thierno Ibrahima KANE et Pape Malick Bâ, méritent également une mention spéciale. Leur amitié et leur soutien moral ont été un véritable moteur tout au long de ce parcours.

Un grand merci à mes frères, Youssouf DIA et Mamadou Sanou BA, pour leurs précieux conseils et leur influence positive dans le choix de ma formation.

Je remercie également mes amis : Chérif Amanatoulah SY, Anna DIAW, Dame Sall, Serigne Abdou Lat SARR, Racine SALL, Mamy NDIAYE, Racine NDIAYE, Daouda KOUNDOUL,Fatou DOUMBOUYA, Youssouf Ndao, Balla KANE.

Enfin, je tiens à remercier l'ensemble de ma famille, mes proches et tous ceux qui, de près ou de loin, ont contribué à la réussite de ce mémoire. Votre soutien a été inestimable.

Ensemble, vous avez été ma source d'inspiration et de motivation. Merci du fond du cœur pour votre confiance et votre soutien constant.

Avant-Propos

Ce mémoire explore en profondeur l'automatisation des tests Interface Homme-Machine (IHM) dans le contexte du développement logiciel. Les tests IHM jouent un rôle essentiel dans la garantie de la qualité des applications logicielles, et l'automatisation de ces tests présente de nombreux avantages en termes d'efficacité, de précision et de rapidité. Le mémoire débute par une analyse approfondie des concepts fondamentaux des tests IHM, en mettant en lumière leur importance cruciale dans le cycle de développement logiciel.

Il se penche ensuite sur l'architecture et les principes de conception du cadre d'automatisation des tests utilisé. L'étude couvre la création de cas de test, les stratégies d'intégration et les solutions pour relever les défis posés par les IHM.

En outre, cette recherche examine les défis et les risques associés à l'automatisation des tests IHM, en mettant l'accent sur l'importance d'une gestion proactive des risques. Des stratégies pour atténuer les complexités techniques, surmonter la résistance au changement et développer les compétences techniques au sein de l'équipe sont explorées en détail.

Un des thèmes centraux de ce mémoire réside dans le potentiel de l'automatisation pour accélérer le déploiement des applications logicielles et favoriser une détection rapide des anomalies, ce qui simplifie grandement le processus de correction.

Tout au long de cette étude, des connaissances pratiques et des exemples concrets sont partagés, offrant ainsi une guidance précieuse aux organisations souhaitant mettre en œuvre ou améliorer leurs processus de tests IHM.

En résumé, ce mémoire propose une vue d'ensemble complète de l'automatisation des tests IHM, couvrant son architecture, la gestion des risques, ainsi que son justification économique, en faisant de lui une ressource précieuse pour les professionnels du développement logiciel et les organisations cherchant à améliorer la qualité et la fiabilité de leurs applications logicielles.

Abstract

This research paper provides a comprehensive study on test automation for software quality assurance. The aim of this study is to investigate the benefits, challenges, and best practices associated with test automation and its impact on software development and delivery processes. The research explores various testing methodologies, tools, and techniques used in the industry to automate software testing and ensure high-quality deliverables.

The paper begins with an introduction to the significance of test automation in accelerating software development and deployment while maintaining high quality. It highlights the need for efficient and reliable testing processes to mitigate risks, enhance productivity, and meet customer expectations.

The study encompasses a detailed analysis of different aspects of test automation, including test planning, test case design, test execution, and results analysis. It investigates the integration of test automation into the software development lifecycle and emphasizes the importance of collaboration between stakeholders, including product owners, UX/UI specialists, and lead technical personnel.

Furthermore, the paper delves into the challenges and considerations associated with test automation, such as prioritizing test cases, maintaining test scripts, managing test data, and handling dependencies. It also addresses the integration of test automation with continuous integration (CI) pipelines to enable automated testing at each code change or software version release.

The research findings showcase the positive impact of test automation on software quality, time-to-market, and overall project success. It discusses the advantages of test automation in terms of early bug detection, faster feedback cycles, improved regression testing, and increased test coverage.

The study concludes with recommendations for implementing effective test automation strategies, including selecting appropriate test automation frameworks, defining clear test objectives, conducting regular code reviews, and continuously refining test cases based on feedback and evolving requirements.

Overall, this research provides valuable insights into the world of test automation and serves as a guide for organizations seeking to adopt and optimize test automation practices. It highlights the significance of test automation in ensuring software quality and streamlining the software development lifecycle, ultimately leading to enhanced

customer satisfaction and business success.

Sigles et Abbréviations

IHM - Interface Homme-Machine : L'interaction entre les utilisateurs et les systèmes informatiques, généralement à travers des interfaces graphiques.

MEP - Mise en Production : Le processus de déploiement d'une application ou d'un logiciel pour qu'il soit accessible aux utilisateurs finaux.

PO - Product Owner : Le propriétaire du produit dans le contexte de la méthodologie Agile, responsable de la gestion du backlog de produits et de la définition des exigences.

MR - Merge Request : Une demande de fusion (ou pull request) est une fonctionnalité permettant aux contributeurs de proposer des modifications à un dépôt de code source et de les fusionner une fois qu'elles ont été examinées et approuvées.

CI/CD - Intégration Continue et Déploiement Continu : Des pratiques de développement de logiciels qui automatisent le processus d'intégration du code source dans un référentiel partagé et le déploiement continu des applications.

Table des Matières

Table des figures

Chapitre 1

Introduction

L'automatisation des tests IHM (Interface Homme-Machine) est devenue un pilier essentiel dans le domaine du développement logiciel moderne. Face à des systèmes de plus en plus complexes et des attentes croissantes en termes de qualité, l'automatisation offre une approche prometteuse pour améliorer l'efficacité, la fiabilité et la cohérence des tests logiciels. Ce mémoire se plonge dans le contexte, les objectifs, les enjeux et les retombées d'un projet majeur : la mise en place d'une solution d'automatisation des tests IHM.

Les logiciels d'aujourd'hui se caractérisent par des interfaces utilisateur sophistiquées et variées, nécessitant une approche de test tout aussi avancée. Cependant, les méthodes de test traditionnelles montrent leurs limites en termes de rapidité et de couverture exhaustive. C'est dans ce contexte que l'automatisation des tests IHM se révèle être une réponse pertinente. Elle promet de réduire le temps nécessaire pour les tests manuels, d'accroître la couverture des tests et de détecter les anomalies plus tôt dans le cycle de développement.

Ce mémoire se penche sur la mise en œuvre d'une solution d'automatisation des tests IHM, mettant en lumière ses avantages ainsi que les défis inhérents. L'objectif fondamental de ce projet est d'améliorer la qualité des applications logicielles tout en optimisant les ressources et le temps consacrés aux tests. Les avantages potentiels englobent une réduction des erreurs humaines, une accélération des tests, une augmentation de la fiabilité des résultats et une meilleure adaptation aux cycles de développement agiles.

Cependant, le passage à l'automatisation des tests IHM ne se fait pas sans obs-

tacles. Les défis sont nombreux, allant de la complexité technique des applications à l'adaptation des équipes et des processus aux nouvelles méthodes de test. La résistance au changement, le besoin de compétences techniques spécifiques et la gestion des environnements de test sont autant de facteurs critiques à prendre en compte.

Ce mémoire examine donc les différentes dimensions de ce projet d'automatisation des tests IHM, de la planification à la mise en œuvre, en passant par l'analyse des risques et l'évaluation des résultats.

1.1 Présentation de l'entreprise

Enedis est une entreprise de distribution d'électricité et gestionnaire du réseau électrique. Elle est née le 1er janvier 2008 dans le contexte de l'ouverture du marché de l'électricité à la concurrence. Enedis se nommait alors ERDF. Elle adopte son nom actuel en 2016 pour mieux incarner sa posture d'acteur de service public nouvelle génération. En effet, l'innovation majeure du transport à longue distance de l'électricité, associée au développement des centrales de production, permettent de desservir un territoire plus important. Avec ces réseaux de longue distance, l'électricité s'étend partout. Ainsi, le marché de l'électricité s'organise entre : la production dans les centrales, le transport sur les lignes à haute tension et la distribution, qui utilise la basse tension pour acheminer l'électricité jusqu'à l'abonné. Le pôle Nexus est une division d'Enedis qui est en charge de la gestion des projets de transformation digitale et de modernisation du réseau électrique français. Ce pôle est responsable de la conception et de la mise en œuvre de solutions innovantes pour améliorer la qualité, la fiabilité et la performance du réseau électrique.

1.2 Contexte et problématique

Le contexte de ce mémoire est celui des applications informatiques qui ont une interface homme-machine (IHM), également appelée interface graphique ou interface utilisateur. Ces applications peuvent être des logiciels pour ordinateurs, des applications mobiles ou des sites web. L'IHM est l'élément clé de l'interaction entre l'utilisateur et l'application. Elle doit être intuitive, ergonomique et répondre aux besoins de l'utilisateur. Ainsi, il est essentiel que l'IHM fonctionne correctement et qu'elle ne présente pas de dysfonctionnements ou de bogues.

Cependant, la complexité croissante des applications IHM rend la tâche de tests manuels très difficile, fastidieuse et chronophage. Les tests manuels nécessitent une intervention humaine pour exécuter les scénarios de test, pour vérifier les résultats et pour rédiger les rapports de test. Cette méthode de test est donc très lente et coûteuse. De plus, les applications IHM évoluent rapidement et sont soumises à des mises à jour fréquentes, ce qui rend la tâche de tests manuels encore plus difficile.

Dans ce contexte, l'automatisation des tests IHM est devenue une solution de plus en plus populaire pour accélérer la phase de tests et pour réduire les coûts liés aux tests. Cette automatisation permet de simuler l'interaction d'un utilisateur avec l'IHM à travers des scénarios de test automatisés. Les outils d'automatisation permettent de détecter rapidement les anomalies, les dysfonctionnements et les bogues éventuels de l'IHM. Cependant, la mise en place d'une automatisation de tests IHM nécessite une étude approfondie des outils disponibles, des méthodologies de tests et des différentes contraintes à prendre en compte pour garantir l'efficacité de cette automatisation.

C'est donc dans ce contexte que se situe la problématique de ce mémoire : **Comment optimiser l'automatisation des tests IHM des applications très évolutives pour réduire leurs délais de mise en production ?**

Le but est de proposer une solution efficace d'automatisation des tests IHM pour améliorer la qualité des applications, réduire les coûts de tests et accélérer les délais de mise en production.

1.3 Initialisation du projet

1.3.1 Charte du projet

L'objectif de mettre en place une solution d'automatisation des tests IHM est de réduire le temps et les coûts liés à la validation manuelle de l'interface utilisateur des applications. Cependant, pour que cette automatisation soit efficace, il est tout aussi crucial que les scripts de tests automatisés soient compris et entretenus par l'équipe de développement. Cela signifie que les tests doivent être régulièrement revus par l'équipe et que toute modification apportée à une fonctionnalité couverte par un test automatisé doit être suivie d'une mise à jour du test correspondant afin de maintenir sa pertinence. L'automatisation des tests IHM permet de simuler les interactions de l'utilisateur avec l'interface graphique de l'application, détectant ainsi rapidement les anomalies éventuelles. Elle permet également de réaliser des tests de manière répétitive, ce qui augmente la couverture des tests et réduit les risques d'erreurs. En outre, elle contribue à améliorer la qualité globale du logiciel en détectant les bugs plus tôt dans le cycle de développement. Pour tirer pleinement parti de l'automatisation des tests IHM, il est crucial que les tests automatisés soient régulièrement revus et mis à jour par l'équipe de développement. Ainsi, lorsque des modifications sont apportées aux fonctionnalités, les tests correspondants doivent être ajustés en conséquence pour garantir leur pertinence continue.

Agile Project Charter	
General Project Information	
Project Name	Mise en place d'une solution d'automatisation des tests IHM
Project Sponsor	Commenditaire
Organization	ENEDIS
Project Start Date	lundi 11 octobre 2021
Project End Date	mardi 10 octobre 2023
Project Details	
Mission	Le projet vise à mettre en place une solution d'automatisation des tests pour les interfaces homme-machine (IHM) afin d'améliorer la qualité et l'efficacité des tests logiciels.
Vision	La solution d'automatisation des tests IHM permettra d'automatiser les scénarios de test répétitifs et fastidieux, réduisant ainsi le temps de test manuel et augmentant la couverture des tests. Cela conduira à une meilleure détection des anomalies et à une livraison plus rapide et fiable des logiciels
Scope	Le projet couvrira l'automatisation des tests pour les IHM de l'application, en mettant l'accent sur les fonctionnalités essentielles et les scénarios de test critiques.
Sucess Metrics	Réduction du temps de test manuel
Defenition of Done criterias	La solution d'automatisation des tests IHM aura été intégrée avec succès dans l'environnement de test
Project Team	
Project Manager / Product Owner	Sebastien THIEBAUT
Scrum Master / Servant leader	Sebastien THIEBAUT
Technical lead	Alexandru SIRBU
Team members	Equipe de développement
	Product Owner / UX-UI
	Equipe de recette (Testeur Manuel)
Team Rules	
Duration of Sprint	Sprint de deux Semaines
Break Between Sprints	15 minutes
Duration of Sprint Planning Meeting	1 heure de temps
Duration of Daily Scrum meeting	15 minutes
Duration of Sprint review	45 minutes
Durattion of Sprint Retrospective	1 heure de temps

Figure 1.1 – Charte du Projet

1.3.2 Registre des parties prenantes

Le registre des parties prenantes est un outil essentiel utilisé dans la gestion de projet pour identifier, analyser et gérer les parties prenantes impliquées dans le projet. Il permet de recueillir et de consolider les informations clés sur chaque partie prenante afin de mieux comprendre leurs besoins, attentes, intérêts et influences.

Dans le tableau ci-dessous, nous pouvons observer les parties prenantes et leurs caractéristiques.

####	Nom	Type	Rôle	Intérêt	Pouvoir	Stratégie	Contributions	Attentes
P1	**Développeur**	Interne	Développer les différentes fonctionnalités de l'application	Elevé	Faible	**Garder Informé**	Ils participent à la mise en place des scripts de test automatisés. Ils doivent couvrir les nouvelles fonctionnalités de l'application.	Périmètre et délais
P2	**Recetteur**	Interne	Tester manuellement l'application avant déploiement	Elevé	Faible	**Garder Informé**	Ils doivent collaborer avec les testeurs pour s'assurer que les scripts couvrent toutes les fonctionnalités de l'application afin de tester ce qui n'est pas encore couvert .	Périmètre et Délais
P3	**QA Testeur**	Interne	Charger d'automatiser les scénarios de test	Elevé	Faible	**Garder Informé**	Ils participent activement à la mise en place des scripts de tests.	Périmètre et délais
P4	**Team management**	Interne	Manager	Elevé	Elevé	**Acteur Clé**	Informer et manager l'équipe	Périmètre
P5	**Commenditaire**	Interne	Manager	Elevé	Elevé	**Acteur Clé**	Planifier les réunions avec l'équipe de management pour suivre l'avance et y apporter des modification	Périmètre

Figure 1.2 – Registre des parties prenantes

1.3.3 Choix de la méthode de gestion du projet

La méthodologie agile scrum a été choisi pour réaliser ce projet. Cette méthode permet de s'adapter facilement aux changements tout en gardant un rythme de production soutenu. Elle est particulièrement adaptée aux projets qui nécessitent des développements itératifs et incrémentaux, ce qui est le cas de la mise en place d'une solution d'automatisation des tests IHM consiste à maintenir les tests tout au long du processus de développement.

En effet, le développement de cette solution peut nécessiter de nombreux ajustements et modifications en cours de route, notamment pour s'adapter à notre application et aux évolutions de ce projet. De plus, la communication entre les membres de l'équipe est essentielle pour assurer la qualité des tests, car il est important de bien comprendre les fonctionnalités à tester et les résultats attendus.

Figure 1.3 – Agile Scrum

Chapitre 2

Etat de l’art sur les tests IHM

2.1 Les différents types de tests

Il existe plusieurs types de tests logiciels qui sont utilisés pour s’assurer que le logiciel est fonctionnel, fiable et répond aux besoins des utilisateurs. Voici une liste des principaux types de tests et leur importance :

Tests unitaires : Les tests unitaires sont effectués sur chaque unité ou module de code pour vérifier si chacun d’entre eux fonctionne correctement. Ils permettent de s’assurer que chaque petite partie du logiciel fonctionne correctement avant d’être intégrée dans le système. Les tests unitaires permettent de détecter rapidement les erreurs de codage, réduisant ainsi les coûts et le temps nécessaires pour les corriger.

Tests d’intégration : Les tests d’intégration sont effectués pour vérifier que les différents modules du logiciel fonctionnent ensemble sans problème. Ils permettent de s’assurer que les interactions entre les modules sont correctes et qu’il n’y a pas de conflit entre les différentes parties du logiciel.

Tests fonctionnels : Les tests fonctionnels sont effectués pour vérifier que le logiciel fonctionne correctement selon les spécifications fonctionnelles et les besoins des utilisateurs. Ils permettent de s’assurer que le logiciel répond aux attentes des utilisateurs et qu’il fonctionne correctement.

Tests de performance : Les tests de performance sont effectués pour vérifier que le logiciel fonctionne correctement dans des conditions de charge élevée. Ils permettent de s’assurer que le logiciel répond aux exigences de performance et de temps de réponse.

Tests de sécurité : Les tests de sécurité sont effectués pour vérifier que le logiciel est sécurisé et qu'il ne présente pas de vulnérabilités qui pourraient être exploitées par des personnes malveillantes. Ils permettent de s'assurer que le logiciel est fiable et qu'il ne met pas en danger les utilisateurs ou les données.

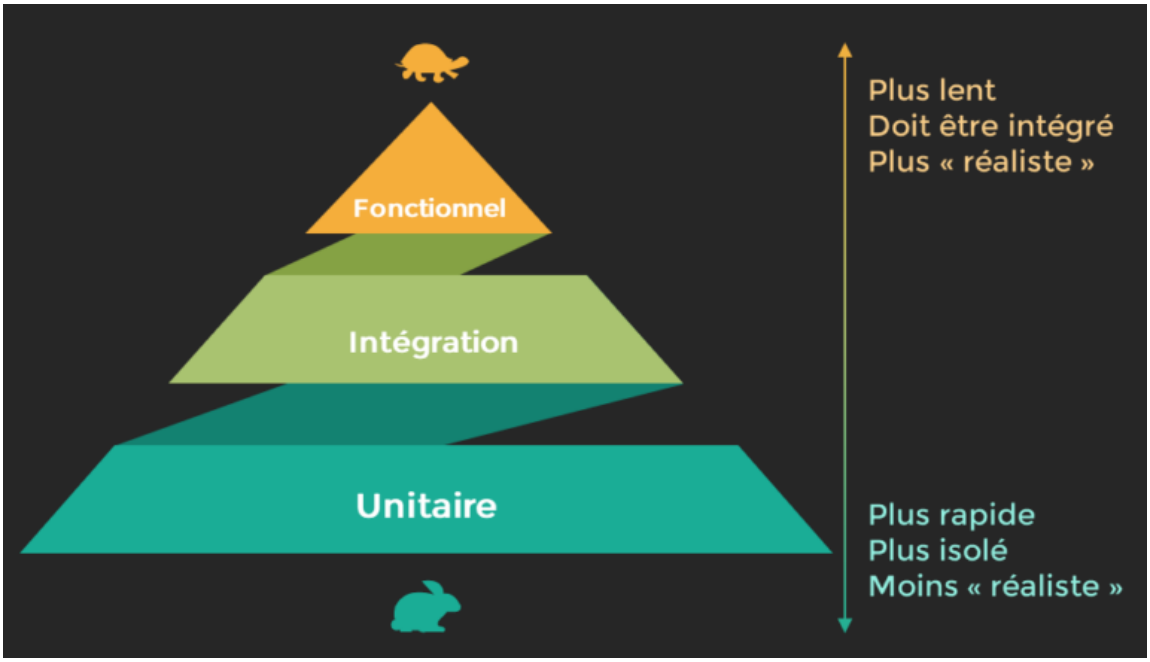


Figure 2.1 – Pyramide de tests

Le schéma ci-dessus met en évidence l'importance de respecter une proportion entre les différents types de tests. Les tests unitaires occupent la base de la pyramide car ils sont plus nombreux, moins coûteux et rapides à exécuter. En revanche, les tests fonctionnels, situés au sommet de la pyramide, sont moins nombreux car ils sont plus coûteux et prennent plus de temps à exécuter. Cette approche vise à optimiser les ressources en accordant une plus grande attention aux tests unitaires, qui peuvent détecter rapidement les erreurs à un niveau granulaire, tout en garantissant une couverture adéquate des tests fonctionnels pour valider le comportement global de l'application.Chacun de ces types de tests est important car ils permettent de détecter différents types de problèmes et de s'assurer que le logiciel est fiable et fonctionnel. En utilisant une combinaison de ces tests, les développeurs peuvent s'assurer que le logiciel est prêt à être utilisé par les utilisateurs finaux.

2.2 Les outils de tests IHM existants

Il existe de nombreux outils de tests IHM (Interface Homme-Machine) disponibles sur le marché, chacun ayant ses propres spécificités et avantages. Voici une liste de quelques-uns des outils de test IHM les plus courants :

Sélénium : Selenium est un outil historique largement utilisé pour l'automatisation des tests dans le domaine des applications web/Java. Cependant, il présente certains désavantages auxquels des outils plus récents ont tenté de remédier.

Cypress : Cypress est un outil de nouvelle génération en pleine expansion, qui offre des avantages considérables pour les tests IHM web. L'un de ses principaux atouts réside dans le fait que les développeurs peuvent écrire les tests dans le même langage que celui utilisé pour l'IHM, à savoir JavaScript ou TypeScript.

Appium : Appium est un outil open source de test d'interface utilisateur pour les applications mobiles. Il prend en charge les plateformes Android et iOS, ainsi que les langages de programmation tels que Java, Ruby, Python, etc. Il peut être utilisé pour automatiser des tests fonctionnels, de régression et de performance.

TestComplete : TestComplete est un outil de test d'interface utilisateur commercial qui prend en charge les tests sur les applications de bureau, mobiles et Web. Il prend en charge plusieurs langages de programmation, tels que JavaScript, Python, VBScript, etc., et peut être utilisé pour automatiser des tests fonctionnels, de régression et de performance.

Testim : Testim est un outil de test d'interface utilisateur commercial qui prend en charge les tests sur les applications de bureau, mobiles et Web. Il utilise une intelligence artificielle pour améliorer les tests et offre des fonctionnalités telles que la génération de tests automatisés, la détection de bogues et l'analyse de tendances.

Chaque outil de test IHM a ses propres avantages et spécificités. Le choix d'un outil dépend de nombreux facteurs, tels que les besoins de l'entreprise, les exigences du projet, les compétences de l'équipe, etc.

2.3 Les avantages et limites de l'automatisation des tests

Avantages et limites de l'automatisation des tests	
Avantages des test automatiser	Limites de l'automatisation
Gain de temps et d'effort: Les tests sont excutés rapidement et a chaque merge reques. Ce qui permet de résoudre les problèmes plus tôt dans le cycle de développement et d'éviter des erreurs coûteuses à corriger par la suite.	**Coûts de mise en place:** Les coûts peuvent être élevés, notamment en termes d'acquisition d'outils de test, de formation du personnel et de mise en place d'une infrastructure de test.
Réduction des coûts: La réduction des anomalies réduit également le volume de défauts à traiter, minimisant ainsi les dépenses liées à l'identification et à la résolution de problèmes	**Limitations de l'automatisation:** Les tests automatisés ne peuvent pas couvrir tous les aspects d'un logiciel, en particulier ceux qui nécessitent une analyse humaine
Amélioration de la qualité: L'automatisation des tests permet de garantir la qualité du logiciel en réduisant les risques d'erreur humaine.	**Maintenance des tests automatisés:** les tests automatisés doivent être maintenus régulièrement pour garantir leur fiabilité et leur pertinence. Les tests automatisés obsolètes ou inadaptés peuvent entraîner des résultats de test erronés et des coûts supplémentaires.
Augmentation de la couverture de test: L'automatisation permet de tester une grande variété de scénarios et de configurations qui ne peuvent pas être testés manuellement en raison de leur complexité ou de leur durée.	**Fausses perceptions de la qualité :** L'automatisation des tests ne garantit pas à elle seule la qualité du logiciel. Les tests doivent être conçus avec soin et en fonction des besoins spécifiques du logiciel pour garantir que les résultats des tests sont fiables et pertinents

Figure 2.2 – Avantages et Limites de l'automatisation

Chapitre 3

Conception de la solution et Planification du projet

3.1 Conception de la Solution et Choix techniques

Le choix d'utiliser Cypress comme outil de test automatisé a été motivé par plusieurs avantages qu'il offre ainsi que par sa compatibilité avec les besoins spécifiques du projet. Cypress est une plateforme de test nouvelle génération qui a gagné en popularité en raison de ses fonctionnalités avancées et de sa facilité d'utilisation.

L'idée principale derrière l'utilisation de Cypress était de sélectionner une technologie facile à comprendre et à utiliser, en particulier pour les développeurs chargés de mettre à jour les scripts de test et d'intégrer les nouveaux tests aux nouvelles fonctionnalités qu'ils développent. Cypress répond à ce critère en offrant une syntaxe claire et expressive qui simplifie l'écriture et la maintenance des tests. Les développeurs peuvent rapidement monter en compétence grâce à la facilité d'utilisation de Cypress et commencer à écrire des tests automatisés efficaces.

Une autre raison du choix de Cypress est sa compatibilité avec TypeScript. TypeScript est un langage de programmation populaire et largement adopté, et son intégration avec Cypress offre plusieurs avantages. Les développeurs peuvent tirer parti de la vérification statique des types offerte par TypeScript, ce qui permet de détecter plus facilement les erreurs de programmation et d'améliorer la qualité des tests.

La facilité d'utilisation de Cypress est également un atout majeur. L'outil fournit une interface graphique conviviale qui permet aux développeurs d'exécuter, de débo-

guer et de visualiser les résultats des tests en temps réel. Cette fonctionnalité accélère le processus de développement et de débogage, permettant aux développeurs d'itérer rapidement et de détecter les erreurs et les régressions plus rapidement.

Enfin, Cypress est bien documenté et dispose d'une communauté active, ce qui facilite l'apprentissage et la résolution des problèmes. L'outil s'intègre facilement avec les outils de développement modernes tels que Git et les pipelines CI/CD, ce qui facilite son adoption et son intégration dans le processus de développement existant.

En résumé, le choix d'utiliser Cypress comme outil de test automatisé repose sur sa compatibilité avec TypeScript, sa facilité d'utilisation, sa syntaxe fluide, son interface graphique conviviale, sa capacité de rechargement en direct et son intégration facile avec les outils de développement modernes. Ces caractéristiques permettent aux développeurs de monter rapidement en compétence, d'écrire des tests automatisés de manière efficace et de haute qualité, et d'intégrer les tests dans leurs processus de développement.

3.2 Plan de management de projet et gestion du périmètre

3.2.1 Définition des scénarios de tests à automatiser

La définition des scénarios de tests à automatiser dans ce projet suit une approche structurée pour faciliter l'intégration des tests automatisés. Voici les étapes clés :

Comprendre les fonctionnalités : Il est crucial d'avoir une compréhension approfondie des fonctionnalités de l'application. Cela implique d'analyser les exigences et les spécifications fonctionnelles afin d'identifier les principales fonctionnalités à couvrir par les tests automatisés.

Identifier les cas de test clés : À partir des fonctionnalités identifiées, il est important de déterminer les cas de test clés qui couvrent les différents flux d'utilisation, les conditions spéciales et les interactions importantes avec l'application. Cela permet de cibler les aspects les plus critiques et les plus susceptibles de présenter des problèmes.

Prioriser les scénarios de tests : Une fois les cas de test identifiés, il est essentiel d'établir une priorisation en fonction de leur criticité, de leur fréquence d'utilisation par les utilisateurs, de leur impact sur l'expérience utilisateur, de leur complexité et d'autres facteurs pertinents. Cela permet de focaliser les efforts sur les tests les plus

importants en premier.

Définir les données de test : Pour chaque scénario de test, il est nécessaire d'identifier les données nécessaires à son exécution. Cela peut inclure des données d'entrée, des données de référence ou des jeux de données spécifiques. Une bonne gestion des données de test garantit la reproductibilité des tests et permet de couvrir différents cas de figure.

Développer des scripts de test automatisés : Une fois les scénarios de test définis, il est temps de développer des scripts de test automatisés correspondants en utilisant cypress. Il est important de suivre les bonnes pratiques de développement de tests automatisés afin de garantir la maintenabilité, la réutilisabilité et la robustesse des scripts.

Valider les scénarios de test automatisés : Avant d'intégrer les scénarios de test automatisés dans le processus d'intégration continue, il est nécessaire de les valider en les exécutant et en vérifiant si les résultats obtenus sont conformes aux attentes. Cela permet de détecter d'éventuels problèmes et de procéder à des ajustements et des améliorations si nécessaire.

Rédiger les scénarios de test dans la bibliothèque de test : Une fois les scénarios de test automatisés prêts, il est recommandé de les documenter de manière structurée dans un système de suivi tel que JIRA. Chaque scénario de test doit être décrit en détail, en spécifiant les étapes à suivre, les données d'entrée nécessaires et les résultats attendus. Cette documentation facilite la visibilité des tests déjà automatisés et permet au recetteur de comprendre rapidement les cas couverts.

3.2.2 Gestion de l'échéancier

Cet échéancier illustre la progression de notre travail sur ce projet depuis octobre 2021. Il est clair que le nombre de tickets augmente à chaque sprint, principalement grâce à l'automatisation des tests, qui nous a permis d'inclure davantage de tickets dans chaque sprint.

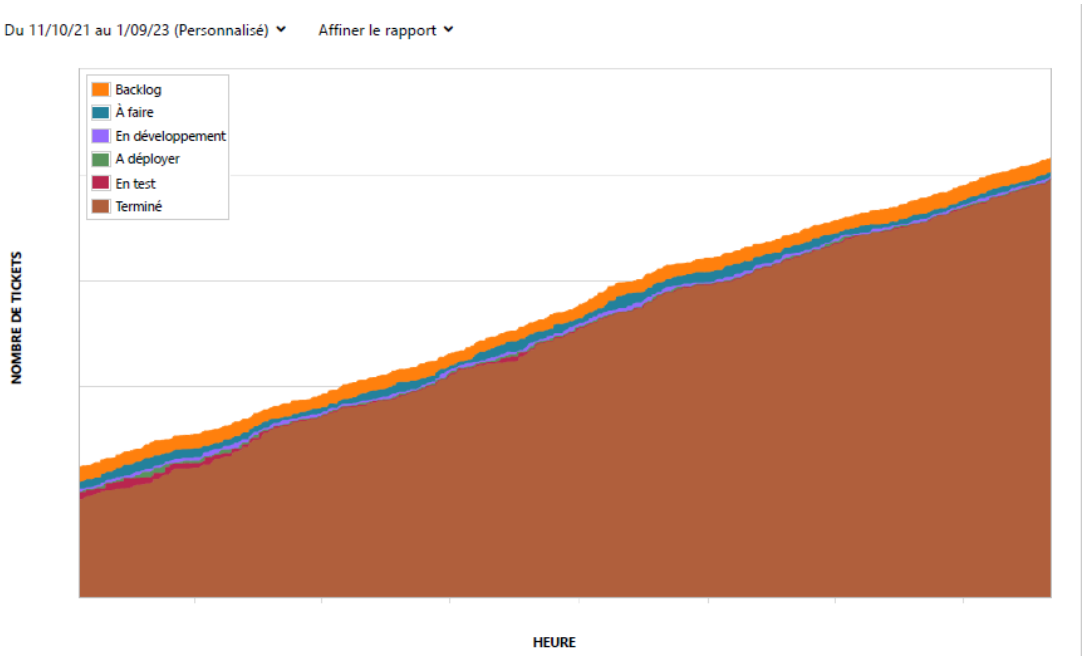


Figure 3.1 – Echéancier

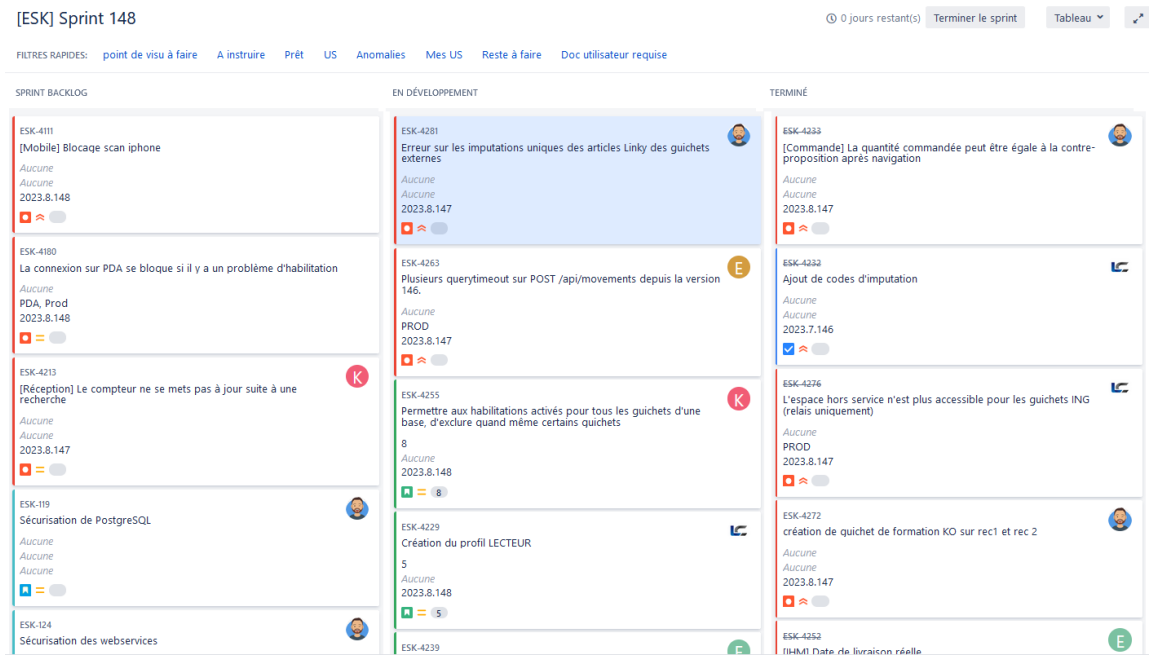


Figure 3.2 – Suivie du backlog

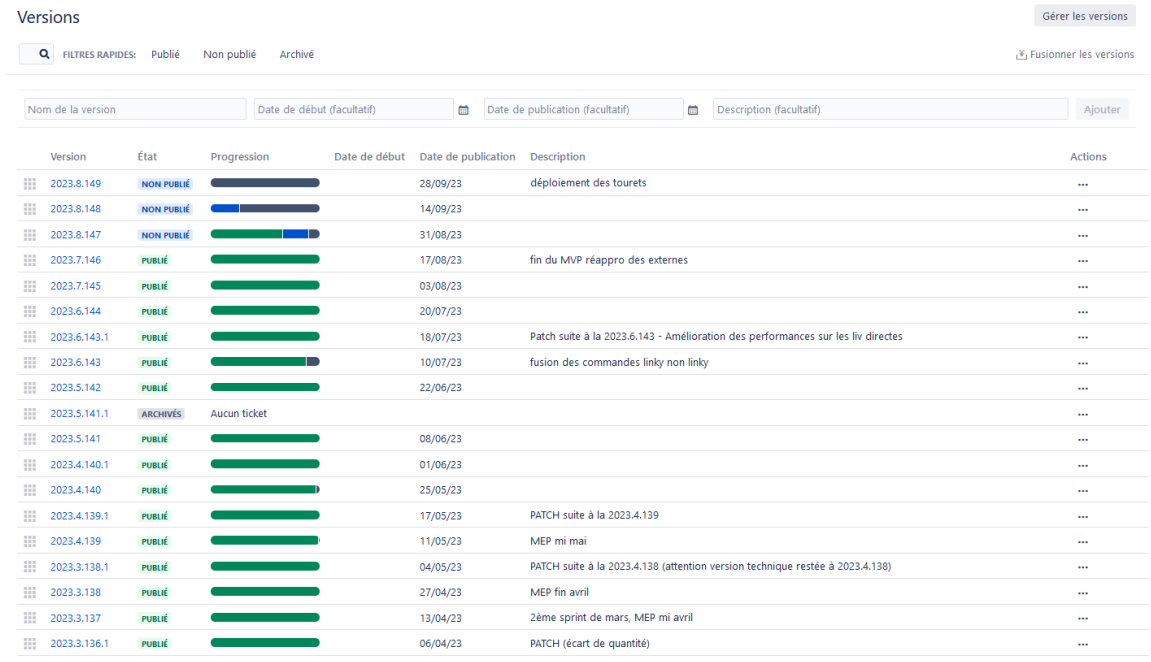


Figure 3.3 – suivie des versions

3.2.3 Gestion du Budget

Table 3.1 – Répartition du budget

Description	**Coût (€)**
Coût Automaticien (oct. 2021 - oct. 2023)	40 800
Coût des développeurs (oct 2021 - oct. 2023)	23 400
- Coût moyen par développeur/jour	450
- Nombre de jours de travail des développeurs	52
Total	**64 200**

3.2.4 Gestion des ressources

La gestion des ressources est un aspect essentiel de notre projet d'automatisation des tests IHM. Elle vise à optimiser l'utilisation des ressources disponibles, telles que les membres de l'équipe, les équipements et les outils, afin d'atteindre les objectifs du projet de manière efficace et efficiente.

Dans le cadre de la gestion des ressources humaines, nous avons soigneusement identifié les compétences nécessaires pour mener à bien notre projet. Notre équipe est composée de différents membres, chacun apportant des compétences spécifiques et complémentaires.

Nous avons un chef de projet qui supervise l'ensemble du projet et assure la coordination entre les différentes parties prenantes. Nous avons également quatre fonctionnels/PO (Product Owners) qui ont une connaissance approfondie des besoins des utilisateurs et qui peuvent guider le développement en fonction de ces besoins.

Deux testeurs sont responsables de la mise en place des tests et de la validation de la qualité du produit.Ils commencent par analyser en détail les exigences fonctionnelles et non fonctionnelles du projet afin de comprendre les attentes et les critères de qualité. Cette analyse approfondie leur permet de concevoir des cas de test pertinents qui couvrent le maximum de scénarios d'utilisation possibles. Ils veillent à ce que chaque fonctionnalité soit testée de manière exhaustive et à ce que tous les cas de test soient documentés de manière claire dans la bibliothèque de tests. ils travaillent en étroite collaboration avec l'équipe de développement pour mettre en place l'environnement de test nécessaire à l'exécution des cas de test. Cela peut inclure la configuration de serveurs, la création de jeux de données de test. Ils écrivent ensuite

les scripts de test automatisés pour les fonctionnalités qui peuvent être testées de manière automatisée. Ces scripts automatisés leur permettent de gagner du temps lors de l'exécution des tests et garantissent une plus grande précision et une meilleure couverture.

Nous comptons neuf développeurs et DevOps qui sont responsables du développement des fonctionnalités et de la mise en place de l'infrastructure nécessaire pour les tests automatisés. Ils maîtrisent les langages de programmation et les outils de développement nécessaires.

Deux spécialistes UX/UI sont chargés de l'ergonomie et de l'interface utilisateur du produit. Leur expertise contribue à la conception d'une expérience utilisateur fluide et intuitive.

Enfin, nous avons deux référents en conduite du changement qui assurent la gestion du changement au sein de l'organisation. Ils travaillent à sensibiliser les utilisateurs finaux et à faciliter l'adoption du produit automatisé.

Cette répartition des compétences nous permet de bénéficier de l'expertise de chaque membre de l'équipe et de travailler de manière complémentaire. Nous sommes en mesure de couvrir tous les aspects du projet, du développement à la validation en passant par l'expérience utilisateur. Cela contribue à la qualité globale du produit et à sa conformité aux attentes des utilisateurs.

En ce qui concerne les ressources matérielles, nous avons déterminé les équipements nécessaires pour l'automatisation des tests IHM. Dans le cadre de ce projet, nous utilisons Jenkins pour la mise en place de l'intégration continue (CI) et du déploiement continu (CD). Jenkins nous permet de lancer les tests de manière automatisée, à des moments prédéfinis tels que chaque jour ou à chaque fusion de code (merge) au minimum.

L'utilisation de Jenkins pour le CI/CD nous offre plusieurs avantages. Tout d'abord, cela permet d'automatiser le processus de construction, de test et de déploiement du logiciel, ce qui augmente l'efficacité et réduit les erreurs potentielles. Les tests sont exécutés de manière régulière et systématique, garantissant ainsi une détection précoce des problèmes et une rétroaction rapide à l'équipe de développement.

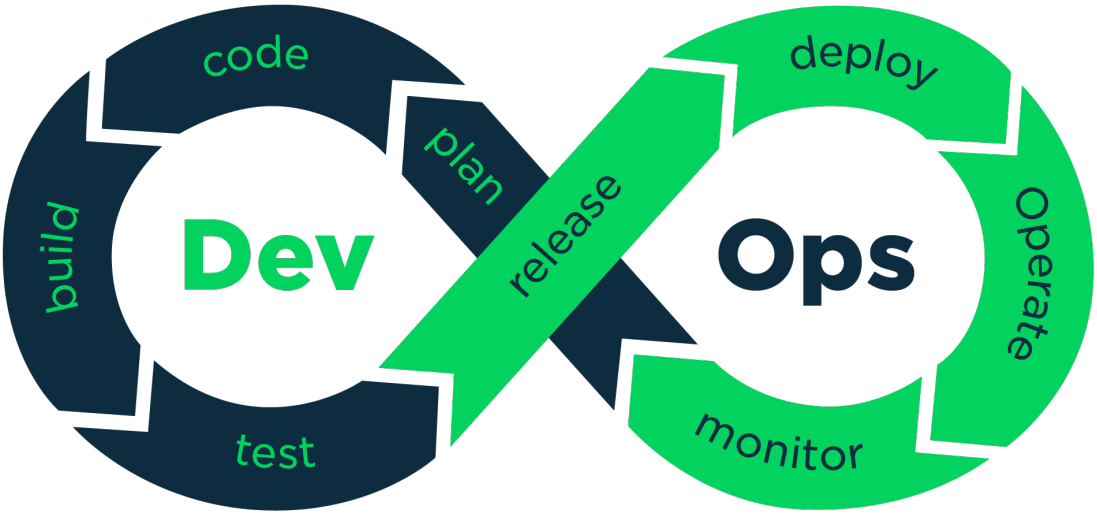


Figure 3.4 – CI/CD

Enfin, nous utilisons également des outils et des technologies spécifiques pour optimiser la gestion des ressources. Par exemple, nous utilisons des outils de gestion de projet tels que Jira pour suivre l'avancement des tâches et gérer les ressources de manière centralisée. Nous utilisons également des outils de gestion de code source comme git pour faciliter la collaboration et la gestion des versions du code.

3.2.5 Gestion des risques

L'automatisation des tests IHM présente incontestablement des bénéfices notables en matière d'efficacité, de précision et de rapidité dans l'exécution des tests. Cependant, ce processus n'est pas sans risques. Nous allons examiner dans le tableau ci-dessous les risques potentiels et tenter de proposer des mesures correctives appropriées.

ID	Nature de risque	Description	Gravité	Probabilité	Criticité	Conséquence si	Tendance	Responsable	Actions préventives	Actions correctrices
R1	Technique	**Complexité technique : La complexité des applications IHM peut rendre l'automatisation des tests plus difficile et potentiellement**	Mineur	Peu probable	**Modéré**	Délais	Stable	Team Dev	Réaliser une analyse approfondie des applications IHM pour identifier les fonctionnalités complexes avant le début du projet. Impliquer des experts techniques dans la planification de l'automatisation pour anticiper les défis potentiels.	Mettre en place des sessions de brainstorming avec l'équipe pour trouver des solutions aux problèmes techniques rencontrés. Si nécessaire, envisager de réorganiser certaines tâches pour surmonter les obstacles
R2	Humain	**Manque de compétences techniques : L'automatisation des tests IHM nécessite des compétences techniques spécifiques qui peuvent ne pas être présentes au sein de**	Grave	Peu probable	**Critique**	Délais	Baisse	Team Dev	Identifier les compétences techniques requises dès le début du projet et prévoir des formations internes ou externes pour développer les compétences nécessaires	Recruter des experts techniques si nécessaire ou prévoir une période de formation continue pour combler les lacunes.
R3	Intrinsèque	**Instabilité de l'environnement de test : Des environnements de test instables peuvent perturber l'exécution des tests automatisés et entraîner des**	Grave	Probable	**Très critique**	Délais	Stable	Team Dev	Mettre en place un environnement de test dédié et surveiller régulièrement sa stabilité.	Collaborer avec l'équipe de développement pour résoudre rapidement les problèmes d'instabilité
R4	Intrinsèque	**Changements fréquents des exigences : Des changements fréquents dans les exigences de l'application peuvent nécessiter des mises à jour**	Mineur	Très probable	**Critique**	Délais	Stable	Equipe recette	Mettre en place un processus de gestion des changements bien défini pour évaluer l'impact des modifications d'exigences sur l'automatisation des tests.	Adapter les scripts de test aux nouvelles exigences en collaboration avec les parties prenantes concernées.
R5	Humain	**Gestion des données de test : La gestion des données de test pour les tests IHM automatisés peut être un défi, en particulier pour les applications avec une grande**	Grave	Peu probable	**Critique**	Qualité	Baisse	Team Dev	Mettre en place une stratégie de gestion des données de test, y compris la création de jeux de données de test réalistes	Réviser la gestion des données de test pour identifier les erreurs et mettre en œuvre des améliorations.

Figure 3.5 – Registre des risques

R6	**Technique**	**Temps de maintenance : Les tests automatisés doivent être entretenus et mis à jour régulièrement pour s'assurer qu'ils restent pertinents et**	Majeur	Peu probable	**Critique**	Qualité	Stable	Team Dev	Prévoir des ressources dédiées à la maintenance régulière des tests automatisés dès le début du projet	Planifier des périodes de maintenance régulières pour s'assurer que les scripts de test restent pertinents et fonctionnels.
R7	**Technique**	**Non-prise en compte de tous les scénarios de test : Il peut être difficile de couvrir tous les scénarios de test possibles avec des tests automatisés, laissant certains aspects sans**	Mineur	Probable	**Critique**	Qualité	Stable	Team Dev	Mettre en place une stratégie de test complète et définir clairement les scénarios à automatiser.	Réviser la stratégie de test pour identifier les scénarios manquants et planifier leur automatisation.
R8	**Technique**	**Difficultés de test sur les interfaces complexes : Les applications avec des interfaces complexes et dynamiques peuvent nécessiter des approches**	Mineur	Probable	**Critique**	Délais	Stable	Team Dev	Réaliser une analyse approfondie des interfaces complexes pour identifier les défis potentiels liés à l'automatisation des tests	Collaborer avec l'équipe de développement pour développer des méthodes d'identification et de sélection des éléments d'interface plus robustes. Mettre en place des scénarios de test spécifiques pour les interfaces complexes et revoir régulièrement leur efficacité
R9		**Ne pas corriger rapidement les tests cassés**	Catastrophique	Peu probable	**Critique**	Qualité	Baisse	Team Dev	Mettre en place une politique de gestion des tests cassés qui inclut des délais pour la correction des tests défaillants. Définir des rôles et des responsabilités clairs au sein de l'équipe pour s'assurer que les tests cassés sont rapidement signalés et attribués à un membre de l'équipe pour correction. Mettre en place des outils de suivi et de reporting pour surveiller l'état des tests et les délais de correction.	Dès qu'un test est identifié comme cassé, le responsable de la correction doit être informé immédiatement. Il doit analyser la cause de l'échec du test, apporter les corrections nécessaires et soumettre le test corrigé à une nouvelle exécution. Il est essentiel de documenter les raisons de l'échec et les mesures prises pour la correction, afin d'éviter les mêmes erreurs à l'avenir.

Figure 3.6 – Registre des risques

Figure 3.7 – Matrice des risques

Chapitre 4

Exécution du projet : Mise en œuvre de la solution

4.1 Pratiques agiles

La gestion de la communication agile est un élément essentiel de notre projet, visant à assurer une collaboration fluide et efficace entre les membres de l'équipe. Elle repose sur des réunions planifiées tout au long du processus de développement :

Réunion de planification : Cette réunion marque le début de chaque sprint et permet de définir les objectifs de livraison à atteindre. L'équipe analyse le backlog des tâches et les priorise en fonction des besoins et des ressources disponibles.

Réunion quotidienne (Daily Scrum) : Ces réunions courtes et quotidiennes permettent à chaque membre de l'équipe de partager son travail accompli, ses défis rencontrés et ses objectifs pour la journée. Cela favorise la transparence, la coordination et la résolution rapide des problèmes.

Revue de sprint : À la fin de chaque sprint, une réunion de revue est organisée pour présenter les résultats obtenus. L'équipe démontre les fonctionnalités développées et échange avec les parties prenantes pour recueillir leurs commentaires et suggestions.

Rétrospective de sprint : Cette réunion intervient après la revue de sprint et permet à l'équipe de réfléchir sur le sprint écoulé. Elle vise à identifier les points forts, les points à améliorer et à définir des actions pour optimiser les prochains sprints.

Afin de faciliter la communication au sein de notre équipe, nous avons mis en place des outils de collaboration performants. Slack est utilisé comme plateforme de mes-

sagerie instantanée, offrant une communication rapide et fluide entre les membres de l'équipe. Nous avons créé des canaux dédiés à différents sujets, favorisant ainsi l'organisation des discussions et la coordination des tâches. Cela nous permet de partager des informations en temps réel, de poser des questions et d'échanger des idées de manière efficace.

Pour la planification des réunions, nous avons adopté Teams. Cet outil nous permet de fixer les dates et les heures des réunions, d'envoyer des invitations aux membres de l'équipe et de créer des rappels. Il s'avère particulièrement utile pour les réunions à distance, garantissant ainsi la participation de tous les membres, indépendamment de leur présence physique.

Par ailleurs, nous utilisons Jira pour la création des tickets qui représentent les différentes tâches du projet. Cela nous permet de suivre l'état d'avancement de chaque ticket, en réalisant des tests ou en les marquant comme terminés. Cela offre une vision globale sur l'état du sprint et facilite la gestion du projet.

En complément de ces outils de communication, nous avons créé un dossier dans la bibliothèque de tests de Jira. Ce dossier détaille l'ensemble des étapes de tests automatisés que nous avons développées. Il permet à l'équipe de recette d'être informée des fonctionnalités couvertes par les tests automatisés. Cela évite de retester manuellement ces cas de test, ce qui se traduit par un gain de temps et une réduction des risques d'erreurs.

Grâce à cette combinaison d'outils de communication et de gestion de projet, nous favorisons une collaboration efficace au sein de l'équipe, améliorant ainsi la productivité et la qualité des résultats obtenus.

4.2 Installation et configuration de l'environnement

Pour installer et configurer l'environnement avec Cypress, suivez les étapes ci-dessous :

Prérequis : Vérifiez que votre système remplit les prérequis pour exécuter Cypress. Assurez-vous d'avoir Node.js installé sur votre machine.

Installation de Cypress via npm : Dans votre terminal, exécutez la commande suivante pour installer Cypress en tant que dépendance de développement :

```
npm install cypress --save-dev
```

Cela téléchargera et installera Cypress localement dans votre projet.

Installation de Cypress via thread :

```
cd /your/project/path
```

```
yarn add cypress --dev
```

Téléchargement direct de cypress : en cliquant sur le lien suivant vous pouvez télécharger la dernière version de cypress directement : Télécharger Cypress directement.

Configuration de Cypress : Une fois l'installation terminée, exécutez la commande suivante pour configurer Cypress :

```
npx cypress open
```

Cela lancera l'interface graphique de Cypress et créera le répertoire de configuration et d'exemples de tests.

Configuration supplémentaire : Vous pouvez personnaliser la configuration de Cypress en créant un fichier cypress.json à la racine de votre projet. Ce fichier permet de définir des options spécifiques telles que les chemins des fichiers de tests, les URL de base,un raccourci pour les commandes cypress,etc.

Exécution des tests : Dans l'interface graphique de Cypress, vous pouvez sélectionner les tests que vous souhaitez exécuter ou exécuter tous les tests en cliquant

sur le bouton "Run all specs". Vous pouvez également exécuter les tests en ligne de commande en utilisant la commande :

```
npx cypress run
```

Configuration de l'environnement : Selon votre projet, vous devrez peut-être configurer des environnements spécifiques pour les tests. Vous pouvez utiliser des fichiers de configuration supplémentaires tels que cypress.env.json pour définir des variables d'environnement spécifiques aux tests.

4.3 Programmation des scripts de tests

La programmation des scripts de tests est une étape importante dans l'automatisation des tests IHM. Elle conciste à écrire du code pour définir les actions à effectuer sur l'interface utilisateur.

Avant que les développeurs ne commencent à travailler sur une User Story (US), plusieurs étapes préliminaires sont nécessaires pour assurer une compréhension claire et une planification efficace. Voici les étapes habituelles avant le démarrage du travail sur une US dans ce projet :

Étape de cadrage : Cette étape consiste en une discussion entre le Product Owner (PO) et les parties prenantes pour clarifier et définir les objectifs et les exigences de l'US. Le PO explique le contexte, les attentes et l'impact commercial de l'US. Des échanges peuvent avoir lieu pour affiner les détails et les critères d'acceptation.

Conception : Une fois que l'US est cadrée, une étape de conception est entreprise. Elle implique souvent une collaboration entre le PO, les spécialistes de l'expérience utilisateur (UX/UI) et le Lead Technique. Les spécialistes UX/UI contribuent à la conception de l'interface utilisateur, en veillant à l'ergonomie et à l'expérience utilisateur optimale. Le Lead Technique apporte son expertise pour évaluer l'impact technique potentiel sur d'autres fonctionnalités du projet.

Rédaction du ticket : Une fois que l'US est cadrée et que la conception est finalisée, le PO rédige un ticket détaillé pour l'US. Ce ticket inclut généralement une description précise des fonctionnalités attendues, des critères d'acceptation spécifiques et d'autres informations pertinentes pour les développeurs. Des captures d'écran ou des maquettes peuvent également être incluses pour faciliter la compréhension.

Revue du ticket : Avant que les développeurs ne prennent en charge l'US, le ticket est généralement examiné par les membres de l'équipe de développement, y compris le Lead Technique. Ils s'assurent que toutes les informations nécessaires sont incluses, que les exigences sont claires et compréhensibles, et posent des questions ou demandent des clarifications si nécessaire.

Étape de chiffrage : Après avoir rédigé et revu le ticket de l'US, une étape de chiffrage est effectuée. Cette étape consiste généralement en une discussion au sein de l'équipe de développement afin d'estimer la complexité et l'effort requis pour mettre en œuvre l'US. Le chiffrage peut être basé sur des points de complexité, des unités de temps de travail dans le sprint ou d'autres méthodes adaptées à la méthodologie de développement utilisée. Après cette analyse, un vote est généralement organisé, auquel seuls les membres de l'équipe de développement participent, pour obtenir une idée du temps et des ressources nécessaires pour réaliser l'US

Analyse de la nécessité d'un test d'intégration :Après avoir chiffré l'US, l'équipe de développement procède à une analyse de la nécessité pour la mise en place d'un test pour cette fonctionnalité. L'objectif est de vérifier que l'US fonctionne correctement en interaction avec les autres composants ou systèmes du projet, que ce soit par le biais d'un test end-to-end, d'un test d'intégration ou des deux. Cette analyse est basée sur divers facteurs tels que la complexité de l'intégration, les risques potentiels ou les dépendances avec d'autres fonctionnalités. Si l'équipe détermine qu'un test d'intégration est nécessaire, celui-ci sera planifié et inclus dans le processus de développement de l'US.

Phase de réalisation de l'US : Une fois que l'US a été estimée, analysée et planifiée, l'équipe de développement se lance dans la phase de réalisation. Pendant cette étape, les développeurs codent et mettent en place la fonctionnalité décrite dans le ticket. Des revues de code sont effectuées pour vérifier la qualité du code et identifier d'éventuels problèmes. Les développeurs s'efforcent de produire un code propre et facilement maintenable. Une fois que la fonctionnalité est entièrement développée, l'étape suivante consiste à intégrer un test automatisé.

L'intégration d'un test automatisé se fait généralement pendant la phase de réalisation de l'US. L'équipe de développement identifie les scénarios de test clés et crée des scripts de test automatisés afin de vérifier le bon fonctionnement de la fonctionnalité développée.

Une fois les tests automatisés écrits, ils sont intégrés au processus d'intégration

continue (CI). Cela signifie que chaque fois qu'une modification du code source est effectuée ou qu'une nouvelle version du logiciel est publiée, les tests automatisés sont exécutés automatiquement pour vérifier si la fonctionnalité développée répond aux attentes.

L'intégration d'un test automatisé permet de détecter rapidement les régressions ou les erreurs introduites lors de la réalisation de l'US. Cela garantit une meilleure qualité du logiciel et facilite la détection précoce des problèmes, ce qui permet de les résoudre plus rapidement.

4.4 Stratégie mise en place pour l'intégration des tests automatisés dans le processus de développement

L'intégration des tests automatisés dans le processus de développement de ce projet revêt une importance cruciale pour garantir une livraison de haute qualité et accélérer le déploiement. La stratégie mise en place pour intégrer les tests dans ce projet comprend les éléments suivants :

Les tests sont priorisés en fonction de la criticité des fonctionnalités couvertes. Nous avons commencé par automatiser en priorité les tests de priorité P0, tandis que les tests qui requièrent l'expertise humaine, tels que les tests visuels et exploratoires, sont réservés aux interventions humaines. Il est essentiel d'évaluer les coûts et les avantages de l'automatisation, en prenant en compte la criticité des fonctionnalités testées.

Identification des fonctionnalités clés à automatiser et détermination du niveau de couverture souhaité (intégration ou End-to-End).

Couverture du code dans la limite du raisonnable.

Maîtrise du temps d'exécution, où la rapidité du feedback est primordiale.

Maintenance et correction des tests en cas d'erreur.

Évitement d'une forte dépendance des tests à une hiérarchie d'abstraction complexe.

Enrichissement et correction des tests en réponse aux anomalies remontées.

Réalisation de tests fiables et reproductibles.

Intégration de l'intégration continue (CI) pour exécuter tous les tests automatiquement, de manière rapide, à chaque commit et merge au minimum.

Mise en place de la configuration des déclencheurs dans le pipeline afin d'effectuer automatiquement les tests à chaque modification du code source ou lorsqu'une nouvelle version du logiciel est disponible. Une fois que les tests automatisés sont exécutés, il peut s'avérer bénéfique de recevoir une notification ou une alerte pour connaître les résultats. Pour ce faire, une intégration avec Slack, un outil de communication d'équipe, a été établi pour informer les membres de l'équipe des résultats des tests. Cette approche permet à toute l'équipe de rester informée en temps réel et de réagir rapidement en cas d'échec des tests..

Maintenance continue afin de refléter les changements apportés à l'IHM et aux fonctionnalités du logiciel. Les scénarios de test automatisés sont mis à jour en fonction des nouvelles fonctionnalités et des modifications de l'IHM.

4.5 Intégration des tests automatisés dans le processus de développement

L'objectif de cette partie est de montrer comment les scripts de test automatisés sont généralement écrits et intégrés dans le processus de développement. Pour cela, nous allons prendre l'exemple de l'écriture d'un script de test automatisé avec l'extension .spec.ts. Dans ce fichier, nous décrirons les différentes étapes du test en utilisant des assertions pour vérifier les résultats attendus. De plus, selon le type de test, nous utiliserons des fichiers JSON pour les données de test dans le cas des tests frontaux, ou des fichiers SQL pour les tests backend. Ces fichiers contiendront les données nécessaires pour exécuter les tests et vérifier les résultats obtenus. En intégrant ces scripts de test automatisés dans le processus de développement, nous pourrons garantir une meilleure qualité et efficacité du logiciel.

exemple de script cypress

```
describe('Tests sur la sortie du stock', () => {

  context('avec un article géré à la qté', () => {

    it('[${IntegrationLabel.IT09}][ESK-3199] test sur les boutons + et - pour
     un article géré à la qté', () => {
      cy.cleanAndLoadDb({
```

```
  interne: '/sql/sortie/IT9.sql'
});

cy.loginOnMobileAs('admin-fonc-exploitation-avec-office');

cy.dataCy('go-to-sortie')
  .click()
  .dataCy('btn-sortie-branchement')
  .click()
  .location()
  .should((location) => expect(location.pathname).to.contain('/mouvement
  /sortie',"La page de sortie n'a pas été chargé"));

//GIVEN
let itemnumAScanner = scanQteFixture.items[0].itemnum; // S6861352
let orderMultiple = scanQteFixture.items[0].orderMultiple; // 3

cy.interceptUrl('GET', 'api/stock/list?s=' + itemnumAScanner + '
    &typeScan=SCAN_CHECKOUT**', {
      fixture: 'mouvements/ESK-3199/scan-article-gestion-qte.json'
});

//WHEN
let qteSaisieArticleGestionQte = 0;
cy.openScan();
cy.scan(itemnumAScanner);
cy.closeScan();
qteSaisieArticleGestionQte = qteSaisieArticleGestionQte + orderMultiple;

cy.get('esk-mouvement-mobile-card').find('[data-cy=common-card-itemnum]')
    .should('contain.text', itemnumAScanner);

cy.dataCy('ctrl-qte-add').click();
qteSaisieArticleGestionQte = qteSaisieArticleGestionQte + orderMultiple;
```

```
        cy.get('esk-mouvement-mobile-card:has([data-cy="common-card-itemnum"]' +
            ':contains(' + itemnumAScanner + '))').contains(
          qteSaisieArticleGestionQte
        );

        cy.dataCy('ctrl-qte-add').click();
        qteSaisieArticleGestionQte = qteSaisieArticleGestionQte + orderMultiple;
        cy.get('esk-mouvement-mobile-card:has([data-cy="common-card-itemnum"]' +
            ':contains(' + itemnumAScanner + '))').contains(
          qteSaisieArticleGestionQte
        );

        cy.dataCy('ctrl-qte-sub').click();
        qteSaisieArticleGestionQte = qteSaisieArticleGestionQte - orderMultiple;
        cy.get('esk-mouvement-mobile-card:has([data-cy="common-card-itemnum"]' +
            ':contains(' + itemnumAScanner + '))').contains(
          qteSaisieArticleGestionQte
        );
      });
  });
```

Le code ci-dessus utilise le framework de test Cypress pour exécuter des tests automatisés sur la fonctionnalité de sortie du stock. Plus précisément, il teste le comportement des boutons "+" et "-" pour un article géré à la quantité. Voici un résumé des étapes du test :

Configuration initiale : Le code effectue une configuration initiale en nettoyant la base de données et en chargeant des données de test spécifiques. Connexion et navigation : Le test simule une connexion en tant qu'utilisateur spécifique et effectue des clics sur des éléments de l'interface utilisateur pour accéder à la page de sortie. Préparation des données : Le test récupère les informations nécessaires pour l'article à tester, telles que le numéro d'article et le multiple de commande. Interception de requête : Le code intercepte une requête GET spécifique pour fournir une réponse prédéfinie à partir d'un fichier JSON de données de test. Exécution des actions : Le test effectue une série d'actions, telles que l'ouverture du scan, la numérisation de l'article, puis la fermeture du scan. Il enregistre également la quantité saisie de l'article

géré à la quantité. Vérification des résultats : Le code vérifie si les résultats attendus sont présents dans l'interface utilisateur après chaque action effectuée. Cela inclut la vérification de l'affichage du numéro d'article, des valeurs de quantité saisie, et de l'interaction avec les boutons "+" et "-". En résumé, ce code exécute un test automatisé pour vérifier le bon fonctionnement des fonctionnalités de sortie du stock et des boutons "+" et "-" pour un article géré à la quantité.

exemple de jeu de donne JSON pour ce test

```
  {
"barcodeType": "QUANTITE",
"items": [
  {
    "itemnum": "S6861352",
    "description": "MANCHON ANCRAGE BROCHE T RETCOMP ABT 54L",
    "refStatus": "ENABLED",
    "orderMultiple": 3.0,
    "orderUnit": "pcs",
    "gestionUnitairePossible": false,
    "qty": 16.0,
    "category": {
      "id": 0,
      "label": "SANS CAT GORIE",
      "color": "#FFFFFF"
    },
    "itemStoreId": 593,
    "itemnumInStore": true,
    "sn": null,
    "snStatus": null,
    "snRefStatusForTransfer": null,
    "tobeAdded": false,
    "itemsConfig": {
      "id": 219243,
      "itemnum": "S6861352",
      "storeId": 593,
```

```
        "qtyMin": 3.0,
        "qtyMax": 9.0,
        "readOnly": false
      },
      "consigne": null,
      "lot": null,
      "gtin": null,
      "qtyLot": 0,
      "productManagementType": "QTY",
      "containerQty": 0.0
    }
  ]
}
```

Le code JSON présent est utilisé dans le script de test pour récupérer les données de l'article lors d'un test frontal. Il est appelé via une fonction d'interception pour simuler une réponse de l'API contenant les données nécessaires. Cette approche permet d'obtenir les informations spécifiques de l'article sans avoir à les saisir manuellement à chaque exécution du test.

Il convient de noter que ce script de test peut être adapté pour être utilisé dans différents types de tests, tels que les tests frontaux ou les tests end-to-end. Dans le cas d'un test end-to-end, où des requêtes SQL peuvent être utilisées pour simuler des données, le même jeu de données présent dans le fichier JSON peut être utilisé dans un fichier SQL. Cela évite de répéter la même logique de test et assure une cohérence entre les différents types de tests.

```
INSERT INTO public.entreprises(id, name)values (1, 'INEO');

INSERT INTO public.organizations (id, name, parent_id, code, imputation_code,
division_code, company_id,
                                  courriel_e_plan)
VALUES (1, 'TESTROOT', null, '9513 ERDF', 'A21', null, null, 'iep-IFE@e-plans.fr');

INSERT INTO public.suppliers (id, name, code, distributor, phone, status, system,
tech_creation_date, last_update,
                              last_update_by, code_pf, prefix_num_commande)
VALUES (1, 'OA92 - SERVAL Nancy', '0100', 'OA92 - SERVAL', '', 'ENABLED', true, '
```

```
    2019-02-20 00:00:00.000000',
10              '2019-02-20 00:00:00.000000', 'SERVAL_11C_REF01', null, 'NE');
11
12  INSERT INTO public.guichets (id, name, organization_id, initialized, suppliers_code,
     department_number, type,
13                                   type_metier_name, entreprise_id, uuid)
14  VALUES (1, 'Guichet Test 1', 1, true, '0100', '01', 'PART', 'EXPLOITATION', 1, '9
    ffc4eeb-e8a2-4671-9542-148df69738ac');
15
16  INSERT INTO public.users_cache (nni, firstname, lastname, email, current_guichet_id,
     buyer_group_code, accepted_gdpr,
17                                      acceptation_date_gdpr, last_connection_date,
    entreprise_id)
18  VALUES ('AS45B74N', 'Test 1', 'User 1', 'alexandru-externe.sirbu@enedis.fr', 1, null
    , true, null, null, 1);
19
20
21  INSERT INTO public.guichets_configs (id, category_id, color, guichet_id)
22  VALUES (1, 1, null, 1);
23
24  INSERT INTO public.guichets_contacts (id, name, phone, guichet_id)
25  VALUES (100, 'Stocky A', '0123456789', 1);
26
27  INSERT INTO public.stores (id, guichet_id, name, default_store, type,
    show_fill_from_order, show_imputations,
28                                 enable_fifo_alerts, sort_order, affaire_id)
29  VALUES (1, 1, 'Stock Test 1', true, 'MAIN', false, true, true, null, null);
30
31  INSERT INTO public.items_config (id, itemnum, guichet_id, qty_min, qty_max,
    read_only, warning_day_delay,
32                                       expiration_day_delay, tech_creation_date,
    status, description, category)
33  VALUES (1, 'S6861352', 1, 5, 10, false, null, null, '2017-01-01 00:00:00.000000', '
    ENABLED',
34              'VIS BOIS RZ FILENT ACZB  5X30', 0);
35
36  INSERT INTO public.items (id, qty, itemnum,item_config_id, store_id, qty_min,
    qty_max, valorisation,
37                                last_activity_date, over_valorisation,
    avg_consumed_qty)
38  VALUES (1, 10, 'S6861352',1, 1,  0, 70, 0, '2018-05-01 00:00:00.000000', 0,    1);
39
```

```
INSERT INTO public.products (sku, upc, description, status, chemicals, dflt_category, typology, linky, linky_specific,
                                mgnt_type, resuppliable, i_raw_unit, i_raw_status, i_raw_mgnt_type, i_mgnt_relevancy,
                                system, tech_creation_date, last_update, last_update_by, hierarchy, consigne_ref_metier)
VALUES ('S6861352', 'S6861352', 'VIS BOIS RZ FILENT ACZB  4X40', 'ENABLED', null, 0, 'OTHER', false, false, 'QTY', true,
            null, null, null, null, false, '2021-03-26 11:51:26.943410', '2021-03-26 11:51:26.943410', 'SERVAL', null,
            null);

INSERT INTO products_suppliers (products_sku, suppliers_code, order_status, last_update_by,order_multiple)
VALUES ('S6861352', '0100', 'ENABLED', 'TEST',3);
```

4.6 Démonstration de l’exécution du script de test

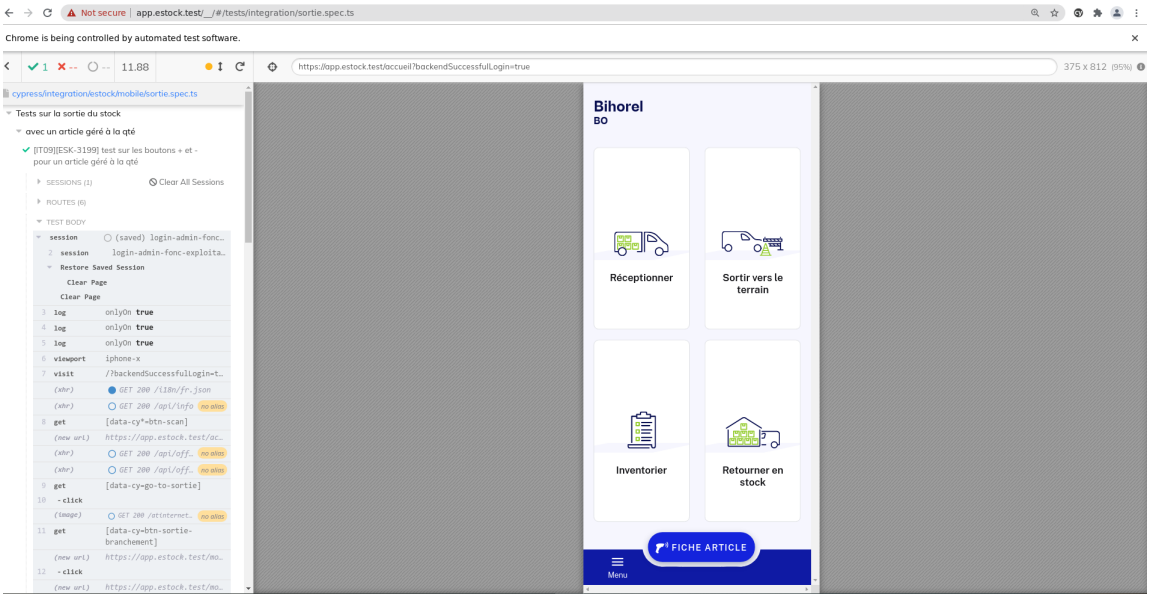


Figure 4.1 – Démonstration Etape 1

Cette capture d’écran représente les étapes effectuées lors de l’exécution du test. Tout d’abord, les données du test sont chargées pour assurer un environnement de test propre et cohérent. Ensuite, l’utilisateur s’authentifie sur l’application à l’aide du nom d’utilisateur et du mot de passe appropriés. Enfin, l’utilisateur clique sur le menu "Sortir vers terrain" pour accéder à la fonctionnalité de sortie.

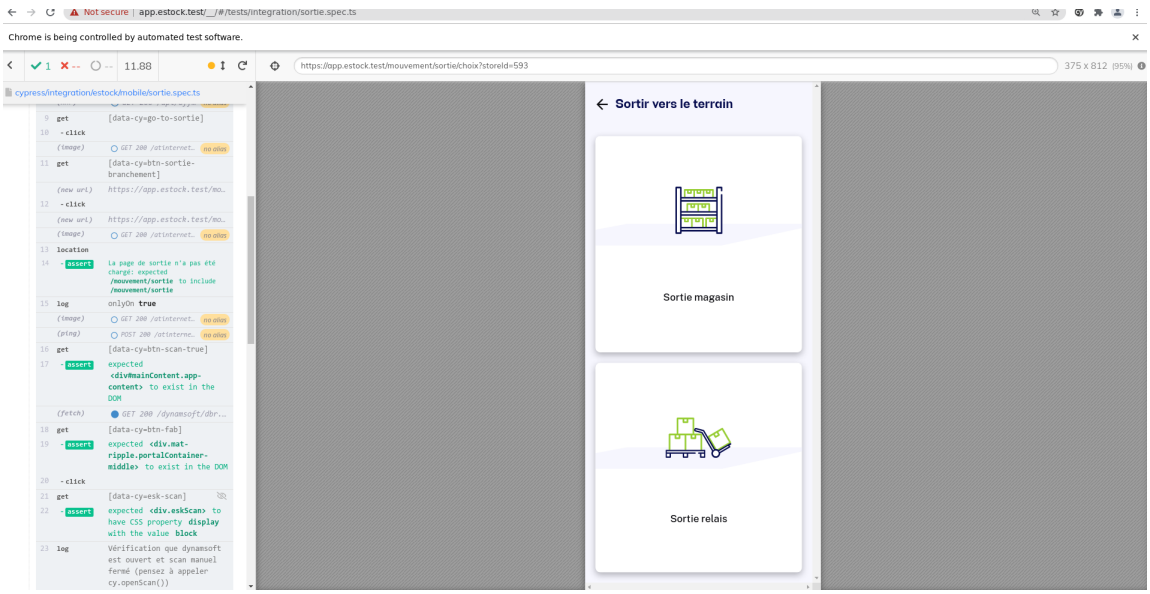


Figure 4.2 – Démonstration Etape 2

Dans cette capture, nous effectuons une vérification de l'URL pour nous assurer que nous sommes bien arrivés à l'écran de sortie après avoir cliqué sur le bouton "Sortir vers terrain" sur l'écran précédent. Une fois sur l'écran de sortie, nous procédons en cliquant sur le bouton "Sortie magasin" afin de pouvoir scanner l'article correspondant. Ces étapes sont essentielles pour vérifier que la navigation entre les écrans se déroule comme prévu et pour préparer l'application à la prochaine action de scan.

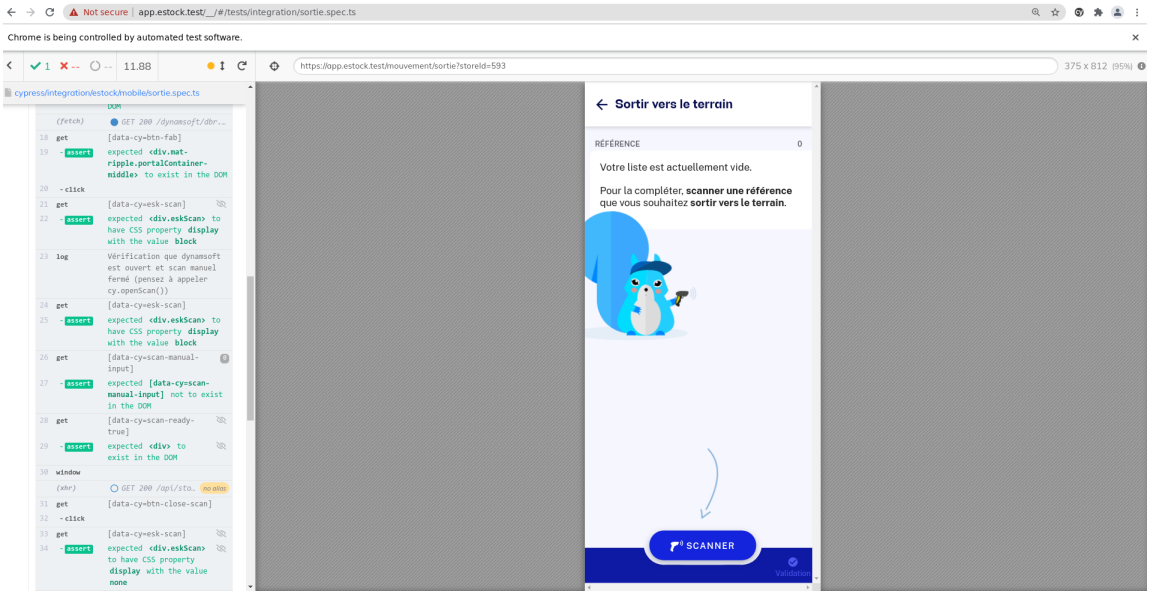


Figure 4.3 – Démonstration Etape 3

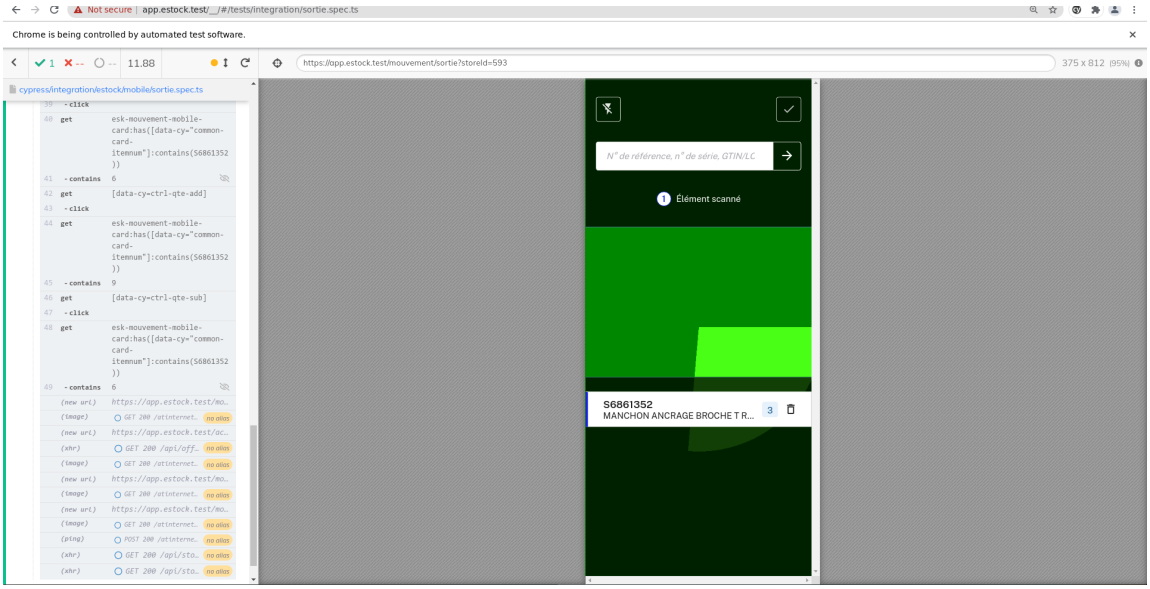


Figure 4.4 – Démonstration Etape 4

Dans cet écran, nous procédons au scan de l'article afin d'afficher ses détails et ainsi pouvoir tester les éléments spécifiques associés à celui-ci. En effectuant le scan, nous obtenons les informations nécessaires pour réaliser les vérifications et les manipulations requises dans l'écran suivant.

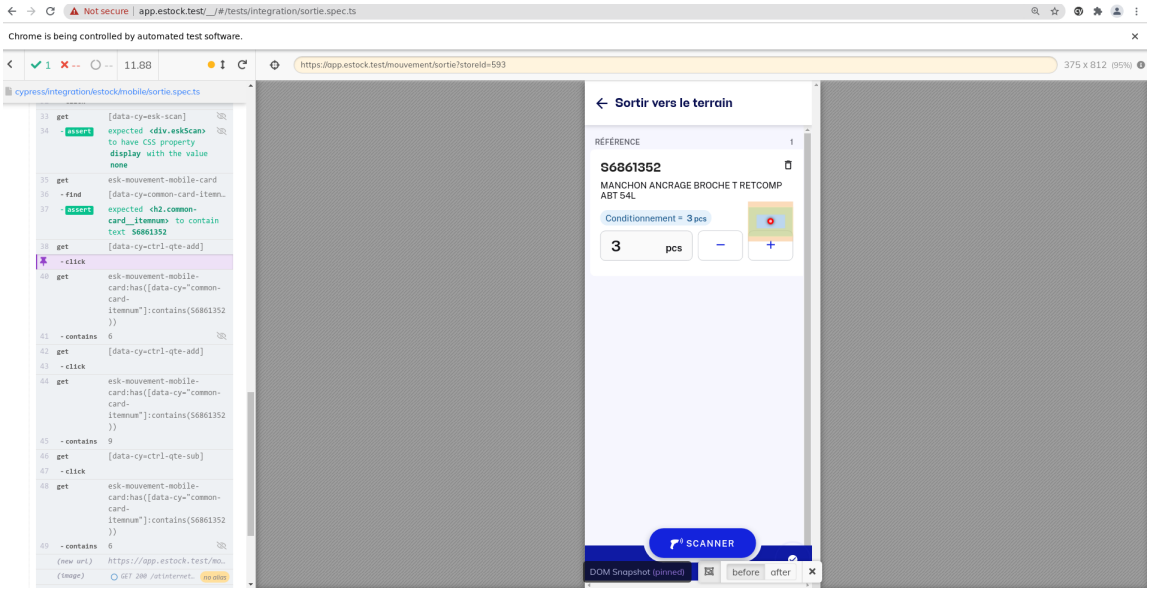


Figure 4.5 – Démonstration Etape 5

Dans cette étape, nous effectuons la vérification des quantités de l'article en testant également les boutons "+" et "-" pour vérifier si les quantités sont correctement mises à jour après chaque clic sur les boutons correspondants. Cela nous permet de valider le bon fonctionnement de la fonction de mise à jour des quantités et de s'assurer que les changements sont bien reflétés à l'écran.

Chapitre 5

Suivi et pilotage du projet avec la gestion de la valeur acquise

Pour faire un planning pour l'intégration des tests automatisés dans un projet, plusieurs éléments doivent être pris en compte. Voici la planification pour l'intégration des tests automatisés dans notre projet : Analyse des besoins et identification des fonctionnalités à tester : Cette étape consiste à examiner les besoins du projet et à identifier les fonctionnalités qui doivent être couvertes par les tests automatisés. Cela peut être réalisé en collaboration avec les parties prenantes et l'équipe de développement. Durée estimée : 1 semaine.

Sélection des outils et technologies : Dans cette étape, il faut choisir les outils et les technologies adaptés pour l'automatisation des tests. Cela peut inclure l'évaluation des différents frameworks, la sélection des outils de test appropriés, et la configuration de l'environnement de test. Durée estimée : 1 semaine.

Conception des scripts de test : Une fois les fonctionnalités identifiées, il faut concevoir les scripts de test automatisés correspondants. Cette étape implique la création des scénarios de test, la définition des étapes de test et des assertions, ainsi que la préparation des jeux de données de test. Durée estimée : variable en fonction de la complexité des fonctionnalités à tester.

Implémentation des scripts de test : Dans cette étape, les scripts de test sont développés et codés en utilisant les outils et les technologies sélectionnés. Les tests sont intégrés au framework de test automatisé et sont configurés pour être exécutés de manière automatique. Durée estimée : variable en fonction du nombre de scripts de

test et de leur complexité.

Exécution des tests et correction des erreurs : Une fois les scripts de test implémentés, ils sont exécutés pour vérifier le bon fonctionnement des fonctionnalités du projet. Les résultats des tests sont analysés et les erreurs identifiées sont corrigées. Durée estimée : variable en fonction du nombre de tests et de la complexité des erreurs.

Intégration continue et exécution régulière des tests : Pour assurer la qualité continue du projet, il est recommandé d'intégrer les tests automatisés dans le processus d'intégration continue (CI). Cela permet d'exécuter les tests automatiquement à chaque modification du code source et d'obtenir rapidement des rétroactions sur les erreurs éventuelles. Durée estimée : continue tout au long du projet.

Maintenance et évolution des tests : Les tests automatisés doivent être régulièrement mis à jour pour s'adapter aux nouvelles fonctionnalités et aux changements dans le projet. Il est important de consacrer du temps à la maintenance et à l'évolution des scripts de test afin de garantir leur efficacité à long terme. Durée estimée : continue tout au long du projet.

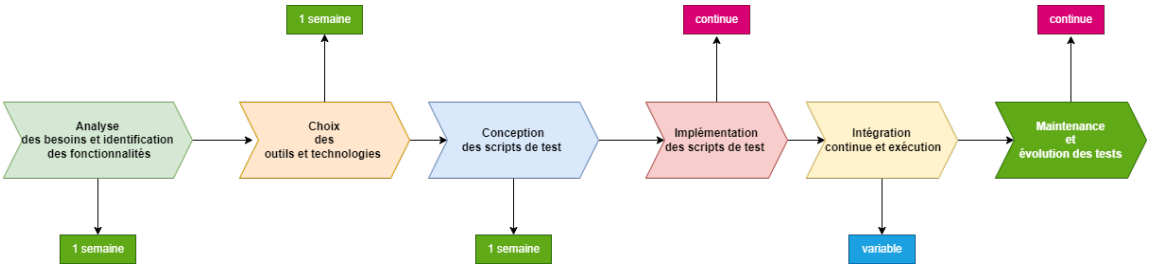


Figure 5.1 – Planning pour l'intégration des tests automatisés

Chapitre 6

Évaluation de la solution

L'évaluation de la solution constitue une étape essentielle dans le processus de développement logiciel. Elle vise à vérifier si la solution développée répond aux exigences et aux attentes des utilisateurs. Cette partie de mon mémoire se focalisera sur les différentes méthodes et approches utilisées pour évaluer la solution, notamment à travers les tests et la validation. L'objectif est d'assurer la qualité, la performance, la fiabilité et la convivialité de la solution, en identifiant et en corrigeant les éventuelles anomalies ou incohérences détectées. Cette évaluation permettra de garantir le succès et l'efficacité de la solution mise en place

6.1 Tests et validation de la solution

Les tests et la validation de la solution jouent un rôle vital dans le processus de développement de logiciels, assurant que la solution élaborée répond de manière adéquate aux exigences du projet, qu'elles soient fonctionnelles ou non fonctionnelles. Pour ce faire, plusieurs étapes essentielles sont mises en œuvre. Initialement, la planification des Tests Non Régression (TNR) sert à tester l'application avant son déploiement, ce qui permet de détecter rapidement les anomalies pour qu'elles puissent être corrigées. Les problèmes identifiés lors des TNR sont classés en fonction de leur priorité dans le sprint suivant, favorisant ainsi une correction rapide.

Tout d'abord, la création des scénarios de test intervient pour couvrir les divers aspects fonctionnels de la solution. Ces scénarios décrivent en détail les démarches à suivre, les données nécessaires en entrée ainsi que les résultats attendus. Un testeur

manuel s'emploie alors à exécuter ces scénarios pour garantir que l'application ne présente pas de dysfonctionnements.

Ensuite, les tests d'intégration sont exécutés pour évaluer l'interaction entre les différents modules de la solution et pour assurer leur bon fonctionnement. Ces tests sont effectués à chaque fusion de code afin de certifier qu'aucun test n'est compromis avant la fusion sur la branche principale. En outre, des tests de régression sont entrepris pour s'assurer que les ajustements effectués sur la solution n'ont pas introduit de nouveaux problèmes.

Une fois les tests achevés, la validation de la solution est menée pour vérifier si elle satisfait les besoins et les attentes des utilisateurs. Cela s'opère par l'utilisation de tests d'acceptation, des démonstrations ou des revues exécutées à la fin de chaque cycle de développement. Les problèmes identifiés lors des tests sont consignés, analysés et rectifiés par l'équipe de développement.

En se basant sur cette approche, une fois les tests automatisés réalisés, ils sont intégrés dans la bibliothèque de tests Jira afin de définir les étapes nécessaires pour la validation par le recetteur. Si des anomalies ou des incohérences sont détectées dans les données ou les résultats des tests, un ticket d'anomalies lié aux tests concernés est créé et remonté à l'équipe de développement. Ce ticket permet aux développeurs de corriger les tests en question. Cette démarche assure la prise en compte des problèmes identifiés et favorise une communication efficace entre les équipes impliquées dans le processus de validation de la solution. En intégrant cette boucle de rétroaction et de correction, les tests automatisés deviennent un outil puissant pour améliorer la qualité et la fiabilité de la solution développée.

6.2 Analyse des résultats obtenus

L'analyse des résultats obtenus dans le cadre des tests automatisés est une étape cruciale pour évaluer la qualité du logiciel et identifier d'éventuels problèmes ou anomalies. Cette analyse permet de prendre des mesures correctives et d'améliorer la fiabilité et la robustesse de l'application.

Lors de l'exécution des tests automatisés,un rapport contenant les résultats sont générés pour chaque scénario de test. Ces résultats comprennent des informations sur les étapes exécutées, les données d'entrée utilisées et les résultats obtenus. Il est essentiel d'analyser ces résultats de manière approfondie pour évaluer la conformité

de l'application par rapport aux attentes.

Dans cette analyse, plusieurs éléments peuvent être pris en compte. Tout d'abord, il est important de vérifier si les résultats obtenus correspondent aux résultats attendus. Si des écarts sont détectés, il convient d'identifier la cause de ces écarts, qu'il s'agisse d'une erreur dans le code source, d'un problème d'environnement ou d'autres facteurs.

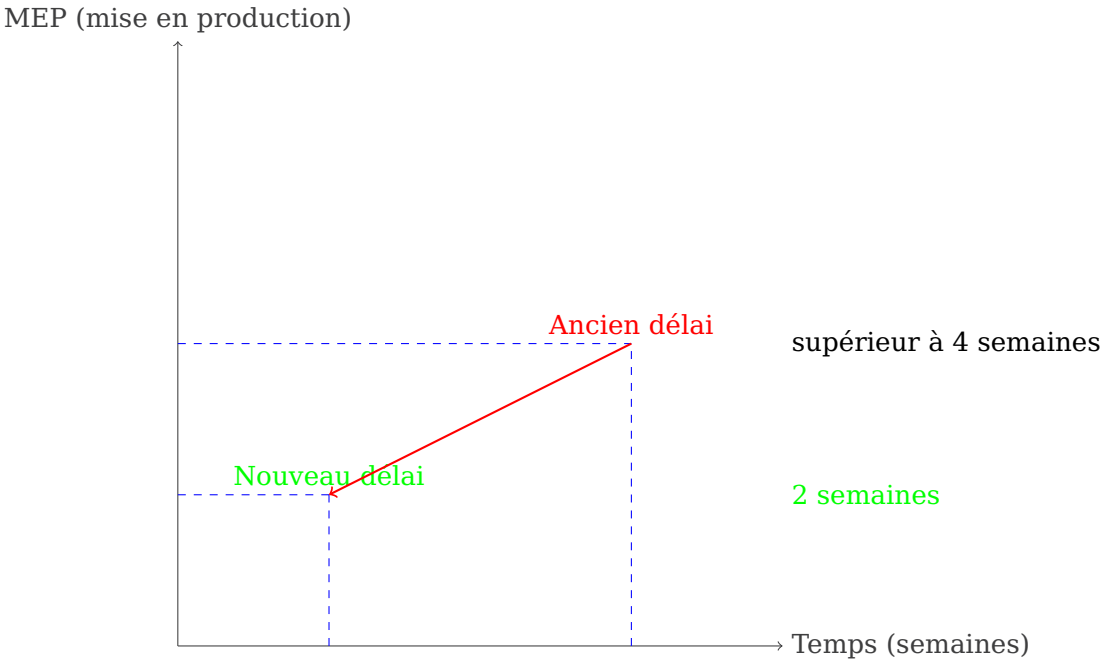
Il est également essentiel d'analyser les tendances et les patterns dans les résultats des tests. Cela permet d'identifier des problèmes récurrents ou des zones de l'application qui nécessitent une attention particulière. Par exemple, si certains scénarios de test échouent régulièrement, cela peut indiquer des problèmes de stabilité ou de fonctionnalité spécifique.

En parallèle de l'analyse des résultats, Nous documentons les problèmes identifiés dans un système de suivi tel que JIRA. Cela facilite le suivi et la résolution des problèmes par l'équipe de développement.

6.3 Évaluation des gains en termes de délais de mise en production

L'intégration des tests automatisés dans le processus de développement présente plusieurs gains en termes de délais de mise en production, de détection d'erreurs, de réduction des anomalies et de réduction des coûts.

Tout d'abord, en utilisant des tests automatisés, le délai de mise en production a été considérablement réduit. Avant l'automatisation des tests, il fallait généralement entre 4 et 6 semaines pour effectuer les tests manuels nécessaires avant la mise en production. Cependant, grâce aux tests automatisés, ce délai a été réduit à seulement 2 semaines. Cela permet une livraison plus rapide des fonctionnalités et une mise en production plus agile.



En détectant les erreurs en cours de développement, les tests automatisés permettent une correction rapide des problèmes. Plutôt que d'attendre les phases de test traditionnelles pour identifier les erreurs, les tests automatisés sont exécutés régulièrement pendant le processus de développement. Cela permet de repérer les erreurs dès qu'elles surviennent, ce qui facilite leur correction immédiate. En corrigent les erreurs rapidement, on évite ainsi les répercussions négatives sur les étapes ultérieures du développement.

L'intégration des tests automatisés contribue également à la réduction des anomalies. Les tests automatisés permettent de vérifier régulièrement et de manière exhaustive le fonctionnement de l'application, ce qui permet de détecter les anomalies potentielles plus tôt dans le processus. Cela permet d'éviter que les anomalies ne se propagent dans le code et d'économiser du temps et des efforts pour les résoudre.

De plus, les tests automatisés permettent de lancer les tests plus rapidement et de manière répétée. Au lieu de compter sur des ressources humaines pour effectuer les tests manuellement, les tests automatisés peuvent être exécutés à tout moment de manière rapide et cohérente. Cela permet de gagner du temps et de maintenir un flux de travail continu, même avec des mises à jour fréquentes du logiciel.

Enfin, l'intégration des tests automatisés réduit les coûts liés au recrutement de testeurs supplémentaires. Étant donné qu'une grande partie des tests est automatisée, il devient moins nécessaire de recruter du personnel supplémentaire pour effectuer des tests manuels. Cela permet d'économiser des ressources financières et de les réallouer à d'autres domaines du projet.

En somme, l'intégration des tests automatisés offre des gains significatifs en termes de délais de mise en production, de détection d'erreurs, de réduction des anomalies et de réduction des coûts. Cela permet une livraison plus rapide des fonctionnalités, une correction rapide des erreurs, une meilleure qualité du logiciel et une optimisation des ressources disponibles.

Chapitre 7

Conclusion et perspectives

En conclusion, la mise en place d'une solution d'automatisation des tests IHM s'est avérée être une étape cruciale pour améliorer l'efficacité, la qualité et la fiabilité des tests dans le processus de développement logiciel. Ce mémoire a exploré en détail les différentes facettes de cette initiative et a mis en évidence à la fois les succès obtenus et les défis rencontrés.

7.1 Bilan de la solution proposée

L'automatisation des tests IHM a permis d'accélérer les cycles de développement et la mise en production en réduisant le temps nécessaire pour exécuter des tests répétitifs. Cela a également ouvert la voie à une couverture plus large des scénarios de test, y compris ceux qui étaient autrefois difficiles à réaliser manuellement. La réduction des erreurs humaines a été un autre avantage significatif, ce qui a contribué à des résultats de test plus fiables et cohérents. Toutefois, il est important de noter que cette transition n'a pas été sans ses challenges.

7.2 Limites et perspectives

La complexité technique des applications IHM a présenté des défis lors de l'automatisation des tests, nécessitant des efforts supplémentaires pour résoudre les problèmes techniques inattendus. De plus, la maintenance constante des scripts de test automatisés a été soulignée comme une préoccupation majeure, exigeant une atten-

tion continue pour assurer la pertinence et l'efficacité des tests au fil du temps. La résistance au changement a également été identifiée comme un obstacle, soulignant l'importance de la communication, de la formation et de la sensibilisation pour faciliter l'adoption de la solution.

7.3 Recommandations pour l'automatisation des tests IHM

À l'avenir, il est recommandé de continuer à investir dans la formation et la montée en compétences de l'équipe pour surmonter les défis techniques et garantir la durabilité de la solution. Une approche proactive de gestion du changement sera essentielle pour assurer une adoption fluide et maximiser les avantages de l'automatisation des tests IHM. En définitive, la mise en place d'une solution d'automatisation des tests IHM constitue un pas important vers l'amélioration continue des processus de développement et de test, contribuant ainsi à la qualité et à la fiabilité des produits logiciels.

Chapitre 8

Bibliographie

https://openclassrooms.com/fr/courses/6100311-testez-votre-code-java-pour-realiser-des-applications-de-qualite/6440061-choisissez-les-bons-tests-automatises-avec-la-pyramide-de-tests, Date de consultation : 15/04/2023.

https://blog.myagilepartner.fr/index.php/2018/08/17/ceremonies-sprint-scrum/, Date de consultation : 02/05/2023.

https://www.atulhost.com/what-is-ci-cd, consultation : 24/04/2023.

Réunion planifier avec SIRBU Alexandru pour échanger sur le choix du framework pour l'implémentation des scripts de test auto. Date du réunion : 16/06/2023.

Réunion planifier avec Yassine OTMANI point d'échange pour définir la mise en place des critère d'acceptance des tickets. Date du réunion : 17/06/2023.

Article : DUEL : TESTS IHM VS TESTS AUTOMATISÉS, PUBLIÉ LE 11 décembre 2020 par Marc Hage Chahine : https://latavernedutesteur.fr/2020/12/11/duel-tests-ihm-vs-tests-automatises/

Livre : "Le guide pratique de l'automatisation des tests" par Daniel Dargent