Mise en place des tests d'acceptation automatisés dans un environnement agile

Étudiant : **SABOUNI Ahmed**

Encadrant de recherche : **BASSON Henri**

ING3 Informatique

Étude bibliographique

École d'Ingénieurs du Littoral-Côte-d'Opale

9 février 2023

Table des matières

1 Introduction

Les tests d'acceptation sont un élément clé de la qualité des produits logiciels. Ils permettent de vérifier que le produit répond aux exigences du client et peut être utilisé de manière efficace. Dans un environnement agile, où la livraison rapide et la flexibilité sont des priorités, il est important de mettre en place une approche efficace pour les tests d'acceptation.

L'objectif de cet article bibliographique est d'explorer les différents aspects de la mise en place des tests d'acceptation dans un environnement agile. Nous examinerons les avantages et les défis de l'intégration des tests d'acceptation dans une méthodologie agile, ainsi que les meilleures pratiques pour les mettre en œuvre avec succès. Nous explorerons également les outils et les technologies disponibles pour soutenir les tests d'acceptation agiles.

Cet article bibliographique aidera les professionnels du développement logiciel qu'ils soient programmeur, testeurs ou experts métiers qui cherchent à comprendre les tests et leurs raport avec les méthodes agiles.

2 Les tests logiciels et leurs automatisations

2.1 L'interêt des tests logiciels

Les tests logiciels sont un processus systématique de vérification du logiciel pour s'assurer de son fonctionnement correct et de la conformité aux exigences spécifiées. Les tests logiciels sont importants pour garantir la qualité et la fiabilité du produit logiciel. Ils permettent de détecter les erreurs et les bugs avant la livraison, ce qui peut réduire les coûts liés à la correction des erreurs en production. Les tests logiciels peuvent également renforcer la confiance des clients et des utilisateurs finaux dans la qualité du produit, ce qui peut améliorer la réputation de l'entreprise et augmenter les chances de vente. Bien que les tests logiciels nécessitent un investissement en temps et en ressources, ils peuvent finalement permettre de réaliser des économies durant toute la durée de vie des produits (AZEEM UDDIN, 2019).

2.2 Classification des tests logiciels par (UMAR, 2020)

Les tests logiciels peuvent être classifiés en fonction de différents critères tels que la catégorie, le niveau, la technique et les types. Chacun de ces critères a ses propres avantages et inconvénients.

— Catégorie : Les tests peuvent être classés en tests dynamiques et tests statiques. Les tests dynamiques peuvent être automatisés, tandis que les tests statiques sont généralement effectués manuellement. Les tests dynamiques ont l'avantage d'être plus rapides et plus fiables, mais peuvent être coûteux à mettre en place. Les tests statiques, quant à eux, sont moins coûteux, mais nécessitent une intervention humaine pour les effectuer et peuvent être plus longs et moins fiables.

— Niveau : Les tests peuvent également être classés en fonction du niveau des composants téstées. Les tests de niveau unitaire sont effectués sur des composants individuels du logiciel comme par exemple des méthodes et des fonctions.Un peu plus haut tests d'intégration sont effectués lorsque plusieurs composants du logiciel sont combinés et testés ensemble. Les tests de système sont effectués sur le système dans son intégralité. Enfin, les tests d'acceptation sont effectués pour vérifier si le produit répond aux exigences des utilisateurs. Chacun de ces niveaux de test a des avantages et des inconvénients spécifiques en termes de coût, de rapidité et de fiabilité.

— Technique : Lorsqu'il s'agit de la technique de test, on distingue principalement les tests non fonctionnels, également appelés tests en boîte blanche, et les tests fonctionnels, appelés tests en boîte noire. La différence qui distingue les tests en boîte noire (BBT) des tests en boîte blanche (WBT) est la question de "perspective". Le BBT considère les objets à tester comme une boîte noire en se concentrant sur les relations entrée-sortie ou sur le comportement fonctionnel externe. Tandis que le WBT considère les objets comme une boîte transcparente où les détails

d'implémentation interne sont visibles et testés. Le BBT et le WBT peuvent également être comparés en fonction de la façon dont ils abordent les questions suivantes : objets, chronologie, focus sur les défauts, détection et correction des défauts. Le BBT est généralement effectué par des testeurs professionnels dédiés et peut également être effectué par des tiers dans un contexte vérification et validation indépendantes, tandis que le WBT est souvent effectué par les développeurs eux-mêmes (TIAN, 2005).

— Types : Enfin, il existe des tests spécifique comme les tests de performance, de sécurité ou d'utilisabilité. Ils sont effectué par des experts dans ces domaines (UMAR, 2020).

2.3 L'interêt des tests d'acceptation et l'importance de leur automatisation

Les tests d'acceptation visent à évaluer le logiciel du point de vue de l'utilisateur et se concentrent sur les scénarios d'utilisation, les séquences, les modèles et les fréquences associées. Ils sont généralement plus adaptés aux logiciels lourds et leurs composants en tant qu'ensemble. Ils ont une relation étroite avec les clients et les utilisateurs et sont effectués dans les phases finales de test. L'environnement de test est similaire à celui utilisé par l'utilisateur final.

	Tests d'acceptation
Catégorie	statique ou dynamique
Technique	boite noir
Responsable	les testeurs et les developpeurs
Quand ?	après le développement
Pourquoi ?	pour vérifier si le produit répond aux exigences des utilisateurs

L'automatisation implique le développement de scripts de test à l'aide de langages de programmation tels que Python, JavaScript ou Java afin que les cas de test puissent être exécutés automatiquement avec un minimum d'intervention humaine. Ceci a l'avantage de réduire l'effort humain et économiser de l'argent. Le logiciel d'automatisation peut également saisir des données de test dans le système, comparer les résultats attendus et réels et aussi générer des rapports de test détaillés (SHARMA, 2014). Le test automatisé nécessite des investissements considérables en argent et en ressources. Il est également possible d'enregistrer les suites de tests et de les rejouer au besoin. Une fois la suite de tests automatisée, aucune intervention humaine n'est nécessaire. Le but de l'automatisation est de réduire le nombre de cas de test à exécuter manuellement et non pas d'éliminer complètement les tests manuels. Les avantages des tests automatisés sont nombreux : la rapidité, l'efficacité, la répétabilité, la réutilisabilité, la programmabilité, la couverture complète et la fiabilité. Malgré tout ces avantages il n'est pas toujours nécessaire, approprié ou rentable d'automatiser ces tests, mais une analyse peut aider à

déterminer les bénéfices d'un test automatisé par rapport à un test manuel. La bonne gestion peut être obtenue en identifiant et en estimant les coûts et les avantages du test automatisé (HOFFMAN, 1999).

3 Les tests agiles

3.1 le testing agile et ses objectifs

Avant l'agilité, les tests étaient souvent organisés de manière plus traditionnelle, en suivant un modèle de développement de logiciel en cascade. Les tests étaient généralement planifiés en fin de cycle de développement et étaient souvent considérés comme une étape de vérification plutôt qu'une activité intégrée au processus de développement. S'ajoute à cela le temps mort des testeurs pendant les phases de développememnt.Ils étaient souvent effectués par une équipe distincte de testeurs et leur résultat était soumis à l'équipe de développement pour correction. Ce processus était souvent considéré comme inefficace et peu collaboratif.

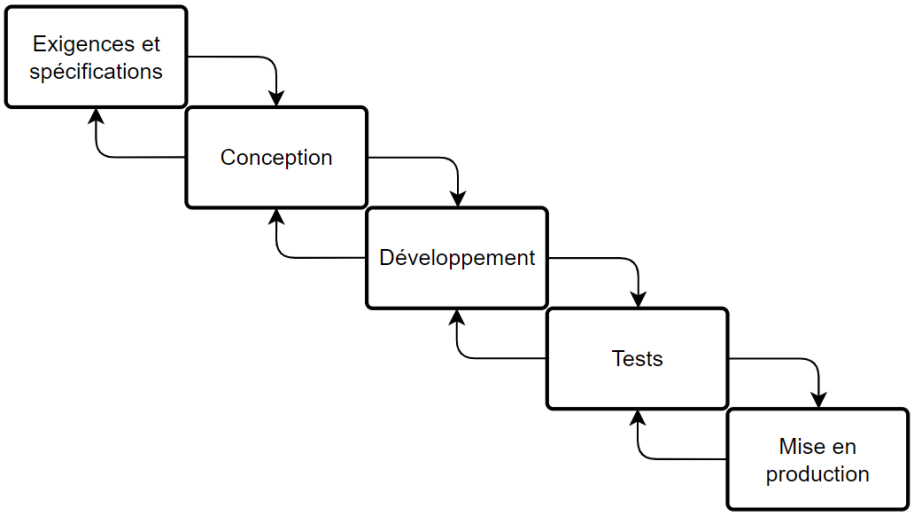


Figure I : Cycle de développement en cascade.

3.2 Le développement piloté par les tests

Avec l'émergence des méthodes agiles de nouveax méthodologie de travail sont apparues en rapport avec lestests. TDD, BDD et ATDD sont trois approches différentes. Il consistent à développer des logiciels qui visent à garantir que le logiciel répond aux exigences et se comporte comme prévu en se basant sur des une planification des tests plus précoces dans le cycle de développement.

TDD : Un processus de développement de logiciels où les développeurs écrivent les tests en premier et écrivent ensuite du code pour faire passer les tests. L'accent est mis sur l'écriture de tests automatisés qui valident le comportement d'unités individuelles de code.

BDD : Une extension du TDD qui se concentre sur l'écriture de tests à un niveau d'abstraction plus élevé. Il vise à décrire le comportement du système du point de vue de l'utilisateur final, ce qui rend les tests plus lisibles et compréhensibles pour les parties prenantes qui ne sont pas familières avec le code (MOE, 2019).

ATDD : Semblable à BDD, l'ATDD est une approche de développement où les critères d'acceptation pour une histoire utilisateur sont définis avant le début du développement et des tests automatisés sont écrits pour valider que l'histoire a été correctement implémentée. L'accent est mis sur l'implication des parties prenantes dans le processus de test pour garantir que les exigences ont été correctement implémentées (GÄRTNER, 2012).

Les scénarios BDD / ATDD sont souvent écrits en utilisant le format Gherkin "Given, When, Then" ou "Etant donné, Quand, alors ", qui décompose le scénario en trois parties distinctes : "Given" qui décrit le contexte de départ de la situation, "When" décrit l'action qui déclenche le comportement à tester, et "Then" décrit les résultats attendus suite à l'action décrite dans "When". Ce format compris par à la fois les supports technique et les éxperts métiers, permet de clarifier les attentes du comportement du système, de fournir une documentation claire et concise des tests, et peut être utilisé en conjonction avec des outils de test automatisé faciliter le processus de développement logiciel.

1 **Fonctionnalité**: météo de France
2 En tant que client, je veux pouvoir consulter la météo de France, afin de
3 pouvoir me préparer à la journée.
4 **Scenario**: consulter la météo de France
5 **Etant donné que** je suis sur la page d'accueil
6 **Quand**: je clique sur le bouton "météo de France"
7 **Alors**: je suis redirigé vers la page de la météo de France

Figure II : Exemple d'une fonctionnalité et d'un scénario BDD

3.3 les contrainte des tests agiles

Quoi qu'ils soient tres bénéfique pour l'entreprise, la mise en place des tests agiles peut présenter certains inconvénients et aussi des défis à relever.

— Coût élevé : la mise en place d'une infrastructure de test automatisé peut être coûteuse en termes de temps et de ressources.
— Temps de développement prolongé : l'automatisation des tests peut prendre beaucoup de temps et peut ralentir le processus de développement.
— Maintenance difficile : les tests automatisés peuvent nécessiter une maintenance constante pour s'assurer qu'ils restent pertinents et précis, ce qui peut être coûteux en temps et en ressources en effet une montée en version d'une application peut re-conceptions des scripts de tests.

— Difficulté à tester des scénarios complexes : il peut être difficile de tester des scénarios complexes avec des tests automatisés, ce qui peut entraîner une couverture insuffisante.
— Dépendance de la qualité du code : la qualité des tests dépend directement de la qualité du code, donc si le code n'est pas correctement écrit, les tests peuvent ne pas être fiables.
— Dépendance des compétences techniques : les tests automatisés nécessitent des compétences techniques pour les écrire et les maintenir, ce qui peut limiter leur utilisation dans certaines équipes.

Cependant, il convient de noter que ces inconvénients peuvent être minimisés en mettant en place une bonne stratégie de test et en choisissant les outils appropriés pour les automatiser .

4 Exemple concret de mise en place des tests d'acceptation dans un environnement agile

Dans cette section nous allons voir comment mettre une stratégie de tests fonctionels d'acceptation dans une entreprise ainsi que les outils utilisés pour la mise en place de cette stratégie.

4.1 Etat de l'art des outils BDD ATD

Il existe plusieurs outils de tests d'acceptation qui sont disponibles sur le marché. Nous allons voir dans cette section les outils les plus utilisés et leurs avantages et inconvénients.

En termes de langages de programmation plusieurs incluent des bibliothéque et des frameork de tests ce qui veut dire que les tests ne seront pas développé de zéro. le choix d'un langage de programmation dépend donc de chaque entreprise et de chaque appllication en effet il est préferable de tester avec le meme langage de programmation avec laquelle l'application est écrite.

— Java avec Cucumber-JVM
— Java avec JBehave
— JavaScript avec Cucumber.js
— Python avec Behave
— .NET avec SpecFlow

Dans la majorité de ces outils les tests sont écrits dans un fichier de format ".feature" "fig" où à chaque étape on peut définir une action, une assertion ou une vérification.

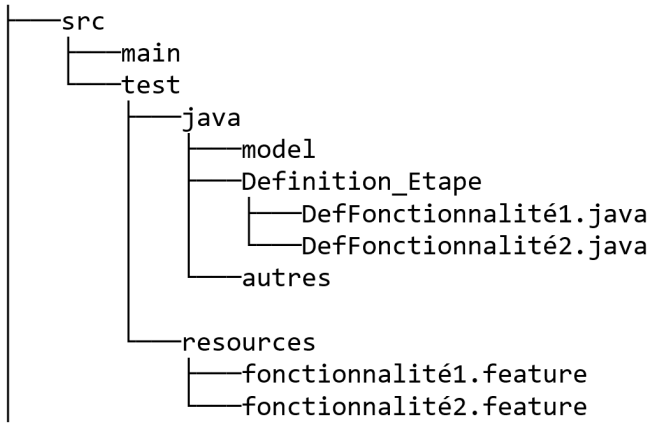
il existe aussi des outils de tests qui ne nécessite pas de langage de programmation. comme par exemple Selenium qui enregistre les actions de l'utilisateur et génère un script de test à partir de ces actions.

le code associé à un fichier de fonctionnalité (fichier.feature) est défini dans des classes d'étapes

didiées comme DefFonctionnali1 et DefFonctionnali2 dédiées. Ces classes contiennent les méthodes qui correspondent aux étapes décrites dans le fichier .feature. exemple :

```
@Given("je␣suis␣sur␣la␣page␣d'acceuil")
public void jeSuisSurLaPageDAcceuil() {
    \\ ici le code du test
}
```

c'est ainsi que sont associés les fichiers .feature et differentes classes et méthode java. Nous pourrons également rajouter des parametres ou des jeux de données dans les fichiers .feature (SMART, 2015). Apres l'éxecution des tests quelques outils integrent la possibilité de générer des rapports de tests qui permettent de visualiser les résultats des tests. Ces rapports sont souvent générés au format HTML pour la visualisation humaine ou au format xml ou json pour des traitement supplémentaire comme par exemple l'envoie vers un outils de gestion des projets.



4.2 Organisation des équipes et bonnes pratiques de tests

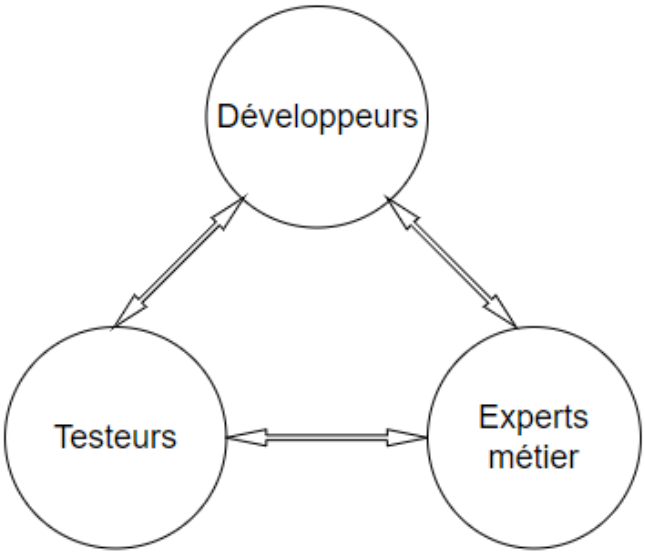


fig :comment les choses doibe

Ici je vais élaborer les points suivants (MARC HAGE CHAHINE, s. d.) :

Éviter de faire une surqualification. Sélectionner soigneusement les tests de régression. Maintenir la campagne de tests de régression. Mettre à jour les cas de tests dès qu'ils sont affectés par un nouveau développement ajouté au produit. Supprimer les tests qui se révèlent inutiles. Élargir les campagnes avec des tests couvrant de nouvelles fonctionnalités. Supprimer les tests les moins importants.

Lutter contre le paradoxe des pesticides en utilisant des tests exploratoires. Mettre en place des outils communs pour les tests. Comprendre l'application et savoir quoi automatiser. Choisir les bons outils d'automatisation. Tous les tests doivent être indépendants. Configurer des rapports détaillés d'automatisation des tests. Éviter les scénarios d'automatisation trop techniques et écrits par les développeurs. Les scénarios automatisés peuvent servir de documentation vivante pour l'application.

Outils de gestion des tests : Utiliser des outils de gestion des tests (Zephyr for Jira, TestRail, Xray). Ils intègrent un support de base pour BDD. Ils ne proposent pas d'intégration avec Git ou de documentation vivante.

4.3 Outils de gestion de projet

Dans un projet de test, les équipes de développement, de test et les experts métiers collaborent sur un outil de gestion de projet pour déterminer une stratégie de test. Chacun apporte son expertise en matière de développement dirigé par les tests (ATDD). Ils planifient les scénarios de test, les fonctionnalités à tester et les parties à automatiser. Une fois les fonctionnalités développées, les tests sont exportés vers

le dépôt de code, où ils sont exécutés, et les résultats sont examinés et revus sur l'outil de gestion de projet.

En plus de tester les nouvelles fonctionnalités, il est important de s'assurer que les modifications apportées n'ont pas affecté les fonctionnalités existantes. C'est ce que l'on appelle les tests de non-régression. Ces tests sont idéalement déjà développés, mais peuvent nécessiter une rectification. Dans ce cas, on utilise un outil d'intégration continue pour garantir la qualité du logiciel. .... Ici je vais expliquer comment on pourra lier le projet de test avec l'outil de gestion (exemple de Jira)

4.4 cycle de test et integration continue

Ici je vais parler des outils d'integration continue comme par exemple jenkins ou gitlab CI pour automatiser le lancement des tests apres les commits des develppeurs ou apres la création d'un nouveau ticket dans l'outil de gestion de projet.

5 Conclusion

ici la conclusion

References

AZEEM UDDIN, A. A. (2019). Importance of Software Testing in the Process of Software Development. *International Journal for Scientific Research and Development*. 6. ISSN : 2321-0613.

GÄRTNER, M. (2012). *ATDD by Example A Practical Guide to Acceptance Test-Driven Development*. Addison-Wesley, pp. 140-146. ISBN : 978-0-321-78415-5.

HOFFMAN, D. (1999). Cost Benefits Analysis of Test Automation. English, pp. 13.

MARC HAGE CHAHINE Bruno Legeard, J. V. Q. (s. d.). *Les tests logiciels en agile*. Français. Sous la dir. de le COMITÉ FRANÇAIS DU TEST LOGICIEL. ISBN : 978-2-9567490-0-4.

MOE, M. M. (juin 2019). Comparative Study of Test-Driven Development (TDD),Behavior-Driven Development (BDD) and Acceptance Test–Driven Development (ATDD). English. *SIAM Journal on Numerical Analysis. International Journal of Trend in Scientific Research and Development (IJTSRD)*. 3, pp. 231-232. ISSN : 2456 - 6470.

SHARMA, R. M. (juill. 2014). Quantitative Analysis of Automation and Manual Testing. English, pp. 252-253. ISSN : 9001:2008.

SMART, J. F. (2015). *BDD in Action,Behavior-Driven Development for the whole software lifecycle*. Sous la dir. de MANNING, pp. 144-170.

TIAN, J. (2005). *Software Quality Engineering, Testing Quality Assurance and quantifiable improvement*. Wilson, pp. 74-83. ISBN : 9780471722335, 0471722332.

UMAR, M. A. (juin 2020). Comprehensive Study of Software Testing: Categories, Levels, Techniques, and Types. English.