1. Introduction

Ces dernières années, la transformation digitale fut très importante notamment suite à la pandémie du Covid-19, qui a nécessité une adaptation informatique pour répondre aux nouveaux besoins du marché. En effet, le développement de la transformation digitale entraîna de nouvelles applications qui ont été créées avec une certaine précipitation dûe à la forte demande. Cela entraina donc un développement au détriment de la qualité des logiciels ou applications, qui enlève donc leur fiabilité et leur efficacité. Or, la qualité est un élément essentiel dans le domaine du digital et du numérique, qui pourrait entraîner des failles du système, des fuites de données ou encore des pannes. Également, le coût de développement devient de plus en plus important pour des applications qui ont autant de défauts. En plus de ces coûts de développement s'ajoutent des coûts de recherche et de corrections des failles et anomalies qui doivent être ajouté comme étape de développement. Alors que les organisations et entreprises deviennent dépendantes de ce système, une question se pose donc : comment développer en assurant une certaine qualité dans la sécurité et l'utilisation de ces logiciels ?

Suite à cette question, le développement d'une nouvelle solution est mise en avant. Avant la finition du logiciel ou de l'application, une vérification doit être mise en place. Vient alors la question de tests particuliers, que l'on appellera donc pas la suite tests unitaires.

Les tests unitaires sont des tests de logiciel dans lequel des unités ou des composants individuels sont testés. L'objectif est donc de valider chaque unité du code et de déterminer si elle fonctionne conformément aux attentes ou non.

Suite à cette solution, les Intelligences Artificielles ont également voulu s'adapter à la demande. En effet, les plateformes proposant des tests de logiciel sont mises à contribution dans cette démarche d'amélioration de la qualité des logiciels.

Dans un premier temps, nous allons présenter de manière plus précise les bases des tests unitaires, pour ensuite parler des bonnes pratiques pour les écrire de manière efficace. Nous verrons ensuite l'importance de la couverture de code. Puis, nous nous intéresserons aux différents frameworks de tests, pour finir sur la contribution des tests unitaires à la maintenabilité d'un projet.

2. Les bases de tests unitaires

Lorsque l'on développe un logiciel, il est important de vérifier le bon fonctionnement du code. Un test unitaire viendra ici vérifier le code et ainsi s'assurer qu'il réponde bien aux attentes du projet. Il consiste à décomposer les fonctionnalités du programme en testant chaque unité du fonctionnement individuellement, vérifiant ces unités une par une ce qui rend le code beaucoup plus fiable à la suite des tests. Ils garantissent donc une qualité du code et font partie intégrante du développement du logiciel.

Une des bonnes pratiques, et donc des bases du développement d'un logiciel est en faire une architecture particulière, c'est-à-dire qu'il fonctionne sous forme d'unités fonctionnelles. Il faut donc commencer par écrire le code dans ces unités respectives, pour ensuite écrire les tests unitaires de chacune. Ainsi, en exécutant les tests, nous obtenons des codes de

succès, d'alerte ou encore d'erreur. En cas d'échec, le code précisera le lieu de l'erreur ainsi que sa raison. Il sera donc simple d'apporter des corrections à l'unité et de recommencer les tests jusqu'au bon fonctionnement total du logiciel.

Les avantages des tests unitaires sont multiples. En effet, ils détectent les bugs de façon efficace, permettant ainsi de se concentrer sur la qualité du code. En vérifiant ainsi les erreurs d'entrée, de sortie ou de logique, les tests unitaires permettent de détecter chaque bug avant la mise en production du logiciel. Le code pourra donc être modifié et amélioré avant la finalisation du logiciel et son utilisation par les futurs utilisateurs.

Avec l'écriture des tests unitaires, la documentation est également très importante. Or, les tests unitaires sont une forme de documentation car explicatif de l'intégralité des petites unités du logiciel. Pour comprendre le comportement du code, nous pouvons donc lire les tests unitaires afin de modifier ce qu'il est nécessaire de modifier.

Les développeurs utilisent les tests unitaires à différentes étapes du développement du logiciel. En effet, il est possible de réaliser des tests avant la réalisation complète du logiciel, qui permettent de vérifier les exigences et de développer par la suite les dernières exigences du logiciel. Egalement, il est possible de réaliser les tests après avoir terminé une partie du code, et de procéder par étape, c'est-à-dire de faire bloc par bloc et test par test afin de voir les résultats plus rapidement et d'éviter une refonte ou une refactorisation plus importante du code suite aux résultats des tests.

En revanche, il y a des situations où les tests unitaires ne sont pas si avantageux. Ils ne sont donc pas toujours requis dans des situations particulières. Si nous prenons l'exemple d'un temps très court pour la réalisation d'un logiciel. En effet, la rédaction de tests unitaires peuvent prendre du temps et par conséquent retarder certains projets. Bien qu'ils soient gages de qualité, certaines organisations préfèrent s'en passer pour livrer plus tôt leur logiciel. La réalisation de tests unitaires entraînerait donc un temps supplémentaire à la livraison et par conséquent un budget en plus pour les réaliser.

Également, quand il s'agit d'une application orientée sur l'apparence, très utilisée aujourd'hui, et non sur la logique de code, il n'y aura pas besoin de tests unitaires.

L'écriture de tests peut également être impossible selon le logiciel et le code. En effet, les tests unitaires demandent des données fictives, données qui peuvent nécessiter une analyse syntaxique importante, ce qui peut prendre trop de temps afin de réaliser le projet.

Enfin, selon le projet, le logiciel peut changer de direction ou avoir des parties entières supprimées au cours d'un cycle de développement donné avec un temps limité. Si les exigences sont susceptibles de changer souvent, il n'y a aucune raison d'écrire des tests unitaires chaque fois qu'un bloc de code est développé.

3. Bonnes pratiques pour écrire des tests unitaires efficaces

Des bonnes pratiques existent quant à la réalisation de tests unitaires. En effet, pour faciliter leur utilisation et ainsi rendre le code plus fiable, des principes ont été créés afin de tirer le meilleur parti de ce processus.

Comme énoncé auparavant, les tests unitaires se font par bloc, ou unité de code. Ils sont personnalisés pour chacun d'entre eux. Or, cela entraîne une perte de temps considérable, alors chaque langage possède sa librairie ou son framework de tests, automatisant ainsi son utilisation et permettant ainsi un gain de temps. Par exemple, Python utilise pytest ou unittest, tandis que PHP utilise PHPUnit ou encore Java utilise JUnit.

Pour réaliser des tests unitaires fonctionnels et compréhensibles, il est important d'éviter les dépendances d'infrastructure. En effet, en induisant des dépendances, les tests deviennent plus difficiles à lire et les problèmes plus dures à résoudre. Ainsi, il est très important de rester dans son unité.

Pour continuer sur le sujet de l'automatisation, les tests unitaires doivent être déclenchés lors d'un événement précis du logiciel. En effet, si automatisé ils se déclencheront tout au long du processus et assure par conséquent la fiabilité demandée sur un logiciel. Nous pouvons les utiliser avant d'apporter des modifications à l'aide d'un logiciel de contrôle de version ou avant le déploiement de la mise à jour du logiciel.

Il faut également faire attention à n'avoir qu'une seule entrée ou sortie pour le test, afin d'éviter le cas où plusieurs instructions seront données pour un même test. Cela peut créer de la confusion quant à savoir laquelle est à l'origine du problème.

Les normes de dénomination des tests est un point important. Les noms doivent se composer en trois parties : le nom de la méthode testée, le scénario dans lequel la méthode est testée, le comportement attendu lorsque le scénario est appelé. L'objectif ici est donc de pouvoir comprendre le comportement du test unitaire, et ainsi de bien comprendre le problème si échec.

Enfin, les tests unitaires sont une étape très importante dans la création d'un logiciel. Malgré que certains projets ne nécessitent pas de tests unitaires comme les sites vitrines, ou bien omis par manque de temps, les tests unitaires permettent de résoudre les problèmes bien plus facilement, ce qui rend la construction d'un prototype ou d'un logiciel complet beaucoup plus facile à suivre et à maintenir.

4. Importance de la couverture de code

La couverture de code mesure le pourcentage de code source testé par les tests unitaires. Ce pourcentage est retranscrit par une métrique qui nous aide à évoluer la qualité du logiciel suite aux tests.

Les outils de couverture de code servent à vérifier quelles parties du programme ont été réellement testées lors de l'exécution d'une suite de tests. Dans un rapport de couverture, on peut retrouver plusieurs informations. Par exemple, la couverture des fonctions indique combien de fonctions ont été appelées au moins une fois. La couverture des instructions montre combien d'instructions du programme ont été exécutées. La couverture des branches correspond au nombre de choix testés dans les structures de contrôle comme les « if ». La couverture des conditions mesure combien de conditions logiques ont été vérifiées à la fois pour vrai et pour faux. Enfin, la couverture des lignes montre combien de lignes du

code source ont été testées. Ces différentes mesures sont généralement présentées sous forme de chiffres et de pourcentages afin de comparer le nombre d'éléments testés avec le nombre total d'éléments présents dans le code.

Ainsi, la couverture de code fournit un aperçu extrêmement puissant des risques. Elle peut être intégrée durant le processus d'intégration continue ou CI, c'est-à-dire durant la période pendant laquelle les développeurs intègrent leurs modifications à un code référentiel centralisé.

Quand on développe un logiciel, une question revient souvent : est-ce que j'ai vraiment fini de tester mon code ? C'est là que la couverture de code entre en jeu. Elle permet de mesurer à quel point nos tests sont complets, en vérifiant si chaque morceau du programme a été exécuté au moins une fois. En combinant différentes méthodes – comme l'analyse statique, les tests unitaires, les tests système ou encore les tests de performance – on s'assure d'avoir un code de meilleure qualité.

La couverture de code, c'est aussi un atout pour répondre à des exigences de conformité, notamment dans des secteurs stricts. Et surtout, elle aide à éviter les erreurs coûteuses. On le sait, les bugs découverts une fois l'application en production sont les plus pénibles à corriger, tant en temps qu'en argent. Du coup, repérer les parties du code non testées avant la mise en ligne, que ce soit directement sur l'ordinateur du développeur ou via un pipeline d'intégration continue, c'est vraiment crucial.

Pour les systèmes critiques, comme ceux liés à la sécurité, on peut même aller jusqu'à vérifier le code assembleur. Ça permet de respecter des normes hyper exigeantes, comme le DO-178C Niveau A, tout en gagnant du temps grâce à des outils automatisés qui évitent des semaines de boulot manuel. De la même manière, pour les applications embarquées, il faut tester le code à la fois sur l'environnement hôte et sur le matériel cible. Avec une instrumentation bien pensée, on peut mesurer la couverture de code de manière ultra-précise, que ce soit sur les instructions, les branches ou même les conditions logiques complexes.

5. Frameworks de tests (JUnit, NUnit…)

Avec le développement des tests unitaires, les organisations se sont vite rendu compte de la perte de temps qu'impliquerait leur réalisation. Une solution est donc arrivée au goût du jour : les frameworks de tests. Chaque langage de code en possède un ou plusieurs selon le langage.

En effet, JUnit est un framework de test pour le langage de programmation Java. Il se veut simple à utiliser, avec une fonction @Test suivi ensuite de la fonction que nous souhaitons tester. Il suffit juste de rajouter des assertions pour avoir les résultats attendus.

NUit quant à lui, est similaire à JUnit mais utilisé pour le .NET Framework. Utilisé très souvent avec du C#, .NET Framework prend en charge l'exécution de sites web, de services ou encore d'applications de bureau. C'est une implémentation multi plateforme permettant

d’exécuter les différents points énoncés. Les tests se définissent cette fois-ci par un ([Test]) avec des assertions pour les résultats. Le framework prend en charge l’exécution des tests en parallèle.

PHPUnit est un framework de test basé sur PHP, conçu pour la programmation de sites internet nécessitant une interaction avec l’utilisateur. Le test se définit par un @test et demande des assertions pour valider le comportement du code.

Bien que chaque langage possède son framework de test, il en existe énormément : MochaJS, fournissant une base de tests et fonctionnement sur le web aussi bien pour le backend que pour le frontend, Jasmine, automatisant le comportement d’un utilisateur et principalement axé sur le frontend ou encore Ava pouvant utiliser les tests unitaires simultanément sur du Javascript.

Ainsi, en utilisant un framework de tests, le gain de temps pour une organisation est significatif, permet de garder la fiabilité du code tout en maximisant le temps à disposition.

6. Contribution des tests unitaires à la maintenabilité

Les tests unitaires automatisés permettent un gain de temps important, mais participent également à la maintenabilité d’un logiciel. En effet, le test validant le bon fonctionnement d’une fonction, permettra par la suite de détecter si la logique du code s’applique toujours avec les mises à jour des librairies ou du langage en lui-même, permettant ainsi d’intervenir au plus vite afin de maintenir le code à jour.

7. Conclusion

En somme, les tests unitaires sont un pilier essentiel pour garantir la qualité et la fiabilité des logiciels, surtout dans un contexte où la transformation digitale pousse à développer vite. Ils permettent de détecter les erreurs tôt, d’améliorer la robustesse du code et de faciliter sa maintenance, tout en répondant aux exigences de conformité. Avec des frameworks comme JUnit, NUnit ou PHPUnit, on gagne du temps sans sacrifier la précision. Même si, dans certains cas, on peut être tenté de les zapper pour aller plus vite, intégrer les tests unitaires dans le processus de développement reste une stratégie gagnante pour livrer des applications solides et durables.

8. Source

https://webtech.fr/blog/pourquoi-la-qualite-des-logiciels-est-elle-importante/

https://learn.microsoft.com/fr-fr/visualstudio/test/unit-test-basics?view=vs-2022

https://aws.amazon.com/fr/what-is/unit-testing/

https://learn.microsoft.com/fr-fr/dotnet/core/testing/unit-testing-best-practices

https://www.atlassian.com/fr/continuous-delivery/software-testing/code-coverage#:~:text=Sten%20Pittet&text=La%20couverture%20de%20code%20est,de%20votre%20suite%20de%20tests.

https://fr.parasoft.com/learning-center/code-coverage-guide/

https://www.digischool.fr/cours/tests-et-validation-frameworks-de-tests-junit-nunit-phpunit-etc-ay30

https://dotnet.microsoft.com/fr-fr/learn/dotnet/what-is-dotnet-framework#:~:text=NET%20Framework%20est%20l'impl%C3%A9mentation,bien%20plus%20encore%20sur%20Windows.

https://geekflare.com/fr/javascript-unit-testing/