

Je Test !

Le test logiciel expliqué simplement, pour tous les niveaux ...



Zied HANNACHI

QA Tech Lead

Sommaire

- Objectifs
- Auteur
- L'importance du test logiciel : Pourquoi ce livre ?
- Qui bénéficiera de ce livre ?

Partie 1 : Le Test Manuel

- **Chapitre 1 : Stratégie de Test**
 - Définition et objectifs
 - Types de tests
 - Élaboration d'une stratégie adaptée
- **Chapitre 2 : Plan de Test**
 - Structure d'un plan de test
 - Exemple de plan de test pour une application mobile
- **Chapitre 3 : Rédaction des Cas de Test**
 - Éléments clés d'un cas de test
 - Techniques de conception de tests
 - Exemple : Test d'une fonctionnalité de connexion
- **Chapitre 4 : Jeux de Données**
 - Importance et création de jeux de données
 - Exemples de jeux de données réalistes
- **Chapitre 5 : Outils de Test Manuel**
 - Présentation des outils populaires
 - Comparaison des fonctionnalités
 - Exemples d'utilisation
- **Chapitre 6 : Rédaction des Rapports de Bugs**
 - Structure d'un bon rapport de bug
 - Outils de gestion des bugs
 - Exemple de rapport dans Jira
- **Chapitre 7 : Tests de Régression**
 - Définition et importance
 - Organisation et exécution des tests
 - Exemple de test de régression

Partie 2 : Le Test Automatique

- **Chapitre 8 : Introduction au Test Automatique**
 - Avantages et limites
 - Quand automatiser les tests ?
 - Différence entre outils et frameworks

- **Chapitre 9 : Selenium**
 - Présentation de Selenium WebDriver
 - Installation et configuration
 - Exemple de test automatisé
- **Chapitre 10 : Cucumber**
 - Introduction au BDD
 - Écriture de scénarios en Gherkin
 - Exemple de test d'une recherche
- **Chapitre 11 : Playwright**
 - Présentation et avantages
 - Automatisation multi-navigateurs
 - Exemple de navigation entre pages
- **Chapitre 12 : Cypress**
 - Introduction et spécificités
 - Configuration et exécution des tests
 - Exemple de test d'un formulaire
- **Chapitre 13 : Robot Framework**
 - Présentation et syntaxe
 - Automatisation des tests fonctionnels
 - Exemple de test de paiement
- **Chapitre 14 : Intégration Continue et Tests Automatisés**
 - Tests dans un pipeline CI/CD
 - Exemple d'intégration avec Jenkins

Partie 3 : Bonnes Pratiques et Conclusion

- **Chapitre 15 : Bonnes Pratiques en Test Logiciel**
 - Maintenir une suite de tests efficace
 - Gestion des données de test
 - Collaboration testeurs-développeurs
- **Chapitre 16 : Tendances Futures en Test Logiciel**
 - Impact de l'IA sur les tests
 - Évolution des outils et méthodologies

Objectifs

L'objectif de ce livre est de fournir une approche complète et accessible du test logiciel, qu'il soit manuel ou automatisé. Dans un monde où la rapidité et la qualité du développement sont essentielles, il devient crucial d'adopter des stratégies de test efficaces et adaptées aux besoins des projets.

Ce livre vise à :

- **Clarifier les fondamentaux** du test logiciel et son rôle dans le cycle de développement.
- **Aider à structurer une stratégie de test** en couvrant aussi bien les tests manuels qu'automatisés.
- **Guider la rédaction et l'exécution des tests** en tenant compte des meilleures pratiques et des défis courants.
- **Introduire les outils et frameworks clés** permettant d'optimiser et d'intégrer les tests dans un processus CI/CD.
- **Sensibiliser aux bonnes pratiques** pour garantir une qualité logicielle optimale et éviter les pièges courants.

Que vous soyez testeur, développeur ou responsable qualité, ce livre vous apportera une méthodologie claire et des outils concrets pour renforcer vos compétences et améliorer l'efficacité de vos tests.

Auteur



Zied Hannachi

QA Tech Lead, Formateur en Automatisation des Tests, Coach, Auditeur et Blogueur, Zied accompagne depuis plusieurs années les entreprises et les équipes dans l'amélioration de leur stratégie de test. Spécialiste des tests automatisés, il partage ses connaissances à travers des formations et des articles techniques pour aider les testeurs et développeurs à monter en compétence.

L'importance du test logiciel : Pourquoi ce livre ?

Le test logiciel est un pilier fondamental du développement informatique. Il permet de garantir la qualité, la fiabilité et la performance des applications avant leur mise en production. Pourtant, il est encore parfois sous-estimé ou mal compris.

Dans un monde où les cycles de développement s'accélèrent avec les méthodologies Agile et DevOps, l'automatisation des tests est devenue un enjeu stratégique. Ce livre vise à :

- Déconstruire les idées reçues sur le test logiciel et en démontrer l'importance.
- Offrir une approche structurée combinant test manuel et test automatique.
- Proposer des méthodologies et des outils adaptés aux besoins réels des projets.

L'objectif est de fournir aux lecteurs une base solide pour intégrer efficacement les tests dans le cycle de développement et améliorer la qualité des logiciels.

Qui bénéficiera de ce livre ?

Ce livre s'adresse à un large public, qu'il soit débutant ou expérimenté dans le domaine du test logiciel :

- **Testeurs manuels** souhaitant mieux structurer leur approche et découvrir l'automatisation.
- **Testeurs automatisés** cherchant à approfondir leurs connaissances sur les frameworks et bonnes pratiques.
- **Développeurs** souhaitant intégrer les tests dans leur workflow et comprendre leur impact sur la qualité logicielle.
- **Chefs de projet et responsables qualité** désireux d'adopter une stratégie de test efficace et cohérente avec les enjeux métiers.

Grâce à un langage clair et des exemples concrets, ce livre est conçu pour être accessible tout en couvrant les aspects avancés du test logiciel.

Partie 1 : Le Test Manuel

Chapitre 1 : Stratégie de Test

- **Définition et objectifs d'une stratégie de test**

Une stratégie de test est un document qui décrit l'approche globale pour tester un logiciel. Elle inclut les objectifs, les ressources, les outils, les environnements de test et les critères d'acceptation.

Exemple réel :

Pour une application e-commerce, la stratégie de test pourrait inclure :

- **Objectifs** : Vérifier que les utilisateurs peuvent parcourir les produits, ajouter des articles au panier, et finaliser leur achat.
 - **Types de tests** : Tests fonctionnels, tests de performance, tests de sécurité.
 - **Ressources** : Une équipe de 3 testeurs, un environnement de test avec une base de données de test.
 - **Critères d'acceptation** : Tous les cas de test doivent passer avec un taux de réussite de 95%.
- **Les différents types de tests**
 - **Tests unitaires** : Vérifier chaque composant individuellement.
 - **Tests d'intégration** : Vérifier l'interaction entre les composants.
 - **Tests système** : Vérifier le système dans son ensemble.
 - **Tests d'acceptation** : Vérifier que le système répond aux besoins des utilisateurs.
 - **Comment élaborer une stratégie de test adaptée au projet**
 - Identifier les risques et les priorités.
 - Définir les environnements de test.
 - Choisir les outils et les méthodologies.

Chapitre 2 : Plan de Test

- **Structure d'un plan de test**

Un plan de test comprend :

- **Objectifs** : Ce que vous voulez accomplir avec les tests.
 - **Périmètre** : Ce qui sera testé et ce qui ne le sera pas.
 - **Ressources** : Les personnes, les outils et les environnements nécessaires.
 - **Planning** : Les dates et les délais pour chaque phase de test.
- **Exemple de plan de test pour une application mobile**

Objectifs : Vérifier que l'application fonctionne correctement sur différents appareils et systèmes d'exploitation.

Périmètre : Tests fonctionnels, tests de performance, tests de compatibilité.

Ressources : 2 testeurs, 5 appareils mobiles, outils de test comme Appium.

Planning :

- Semaine 1 : Préparation des cas de test.
- Semaine 2 : Exécution des tests fonctionnels.
- Semaine 3 : Exécution des tests de performance.
- Semaine 4 : Revue et rapport final.

Chapitre 3 : Rédaction des Cas de Test

- **Les éléments clés d'un cas de test**
 - **Préconditions** : Ce qui doit être en place avant d'exécuter le test.
 - **Étapes** : Les actions à effectuer.
 - **Résultats attendus** : Ce qui devrait se produire après les actions.
- **Techniques de conception de tests**
 - **Partitionnement** : Diviser les données d'entrée en groupes logiques.
 - **Analyse des valeurs limites** : Tester les valeurs aux limites des domaines de données.
- **Exemple : Cas de test pour une fonctionnalité de connexion**

Préconditions : L'utilisateur est sur la page de connexion.

Étapes :

 1. Saisir "testuser" dans le champ "Nom d'utilisateur".
 2. Saisir "password123" dans le champ "Mot de passe".
 3. Cliquer sur le bouton "Connexion".

Résultat attendu : L'utilisateur est redirigé vers la page d'accueil.

Chapitre 4 : Jeux de Données

- **Importance des jeux de données dans les tests**

Les jeux de données permettent de simuler des scénarios réels et de vérifier que le système gère correctement différentes entrées.
- **Création de jeux de données réalistes**
 - Utiliser des données anonymisées provenant de la production.
 - Générer des données aléatoires mais réalistes.
- **Exemple : Jeux de données pour tester un formulaire d'inscription**

Jeu de données 1 :

 - Nom : "Jean Dupont"
 - Email : "jean.dupont@example.com"
 - Mot de passe : "Password123!"

Jeu de données 2 :


 - Nom : "Marie Curie"
 - Email : "marie.curie@example.com"


- Mot de passe : "Marie123!"


Chapitre 5 : Outils de Test Manuel


- Présentation des outils populaires


 **TestRail** : Gestion des cas de test et des exécutions.

 **Zephyr** : Intégration avec Jira pour la gestion des tests.

 **Squash TM** : Outil de gestion de tests open-source avec une interface intuitive.

 **Xray** : Extension Jira permettant la gestion des tests manuels et automatisés avec une traçabilité avancée.

 **TestLink** : Outil open-source permettant la gestion des cas de test, des plans de test et des campagnes d'exécution.

 **Jira** : Outil de gestion de projet et de suivi des bugs, souvent utilisé en conjonction avec des plugins comme Xray et Zephyr pour la gestion des tests.

- Comparaison des fonctionnalités

Critères	TestRail	Xray	TestLink	Zephyr	Squash TM	Jira
Type d'outil	Gestion des tests	Extension Jira pour tests	Gestion des tests open source	Extension Jira pour tests	Gestion des tests open source	Gestion de projet (avec plugins pour tests)
Intégration avec Jira	Oui	Oui (natif)	Non	Oui (natif)	Oui	Oui
Support CI/CD	Oui	Oui	Limité	Oui	Oui	Avec plugins
Traçabilité des exigences	Oui	Oui	Oui	Oui	Oui	Avec plugins
Automatisation des tests	Intégration avec Selenium, JUnit, etc.	Oui	Non	Oui	Oui	Avec plugins

Interface utilisateur	Moderne et intuitive	Interface Jira	Simple mais datée	Interface Jira	Moderne	Interface Jira
Gestion des plans de test	Oui	Oui	Oui	Oui	Oui	Avec plugins
Gestion des campagnes de test	Oui	Oui	Oui	Oui	Oui	Avec plugins
Rapports et dashboards	Oui, détaillés	Oui, via Jira	Basique	Oui, via Jira	Oui, complets	Avec plugins
Modèle de licence	Payant	Payant (via Jira)	Open Source	Payant (via Jira)	Open Source	Payant (Jira)

- **Exemple d'utilisation de Squash TM pour gérer les cas de test**
 - Créer un projet dans Squash TM.
 - Ajouter des cas de test avec des étapes et des résultats attendus.
 - Exécuter les tests et enregistrer les résultats.
 - Générer des rapports détaillés pour les parties prenantes.
- **Exemple d'utilisation de Jira pour la gestion des tests**
 - Créer un projet dans Jira.
 - Utiliser le plugin Xray pour ajouter des cas de test.
 - Exécuter les tests et enregistrer les résultats directement dans Jira.
 - Suivre les bugs et les problèmes dans le même outil.

Chapitre 6 : Rédaction des Rapports de Bugs

- **Structure d'un bon rapport de bug**
 - **Titre** : Description concise du problème.
 - **Étapes pour reproduire** : Les actions nécessaires pour reproduire le bug.
 - **Résultat attendu** : Ce qui devrait se produire.
 - **Résultat obtenu** : Ce qui se produit réellement.
- **Outils de gestion des bugs**
 - **Jira** : Outil populaire pour le suivi des bugs.
 - **Bugzilla** : Outil open-source pour la gestion des bugs.
 - **Mantis** : Outil simple et efficace pour le suivi des problèmes.
- **Exemple : Rédaction d'un rapport de bug dans Jira**

Titre : "Le bouton de connexion ne fonctionne pas sur la page d'accueil."

Étapes pour reproduire :

 1. Aller à la page d'accueil.
 2. Cliquer sur le bouton "Connexion".

Résultat attendu : La page de connexion devrait s'afficher.

Résultat obtenu : Rien ne se passe, le bouton ne fonctionne pas.

Chapitre 7 : Tests de Régression

- **Définition et importance des tests de régression**

Les tests de régression vérifient que les nouvelles modifications n'ont pas introduit de nouveaux bugs dans les fonctionnalités existantes.

- **Comment organiser et exécuter des tests de régression**

- Identifier les zones critiques à tester après chaque modification.
- Automatiser les tests de régression pour gagner du temps.

- **Exemple : Test de régression après une mise à jour de fonctionnalité**

Après une mise à jour du système de paiement, exécuter des tests sur :

- La fonctionnalité de paiement.
- La gestion des commandes.
- L'historique des transactions.

Partie 2 : Le Test Automatique

Chapitre 8 : Introduction au Test Automatique

- **Avantages et limites du test automatique**

- **Avantages** : Rapidité, reproductibilité, couverture de test étendue.
- **Limites** : Coût initial élevé, maintenance des scripts de test.

- **Quand automatiser les tests ?**

- Lorsque les tests sont répétitifs.
- Pour les tests de régression.
- Pour les tests de performance.

- **Différence entre les frameworks et les outils de test**

- **Outils** : Logiciels spécifiques pour exécuter des tests (Selenium, Cypress).
- **Frameworks** : Structures qui organisent les tests (Robot Framework, TestNG).

Critère	Selenium	Cypress	Playwright	TestNG	JUnit	Robot Framework	Cucumber
Type	Framework d'automatisation web	Framework de test end-to-end	Framework de test end-to-end	Framework de test unitaire	Framework de test unitaire	Framework de test basé sur mots-clés	Framework BDD (Behavior-Driven Development)
Langages supportés	Java, Python, C#, JavaScript, etc.	JavaScript, TypeScript	JavaScript, TypeScript	Java	Java	Python, Java, d'autres via	Java, JavaScript, Ruby, Kotlin, etc.

			Python, C#			bibliothèques	
Portée des tests	Web (multi-navigateurs)	Web (Chrome, Edge, Firefox expérimental)	Web (Chrome, Firefox, WebKit)	Unitaire et intégration	Unitaire et intégration	Web, API, Desktop, Mobile	Web, API, intégration
Facilité d'installation	Moyenne (nécessite WebDriver)	Facile (tout intégré)	Facile (tout intégré)	Facile	Facile	Moyenne (besoin de bibliothèques)	Facile (nécessite un framework d'automatisation)
Vitesse d'exécution	Moyenne	Rapide (DOM en mémoire)	Rapide (basé sur WebKit)	Rapide	Rapide	Moyenne (dépend du test)	Moyenne (dépend de l'automatisation)
Support du mobile	Oui (avec Appium)	Non	Non (mais expérimental)	Non	Non	Oui (via Appium, Selenium)	Non (dépend du framework sous-jacent)
Support des tests API	Non natif (via RestAssured, etc.)	Oui	Oui	Non natif	Non natif	Oui	Oui (avec des extensions)
Support des tests visuels	Non natif (via outils tiers)	Non natif (via outils tiers)	Oui (via Playwright Trace Viewer)	Non	Non	Oui (via bibliothèques)	Non natif (via outils tiers)
Parallélisation	Oui (via Grid/Test NG)	Oui	Oui	Oui	Oui	Oui	Oui (via framework sous-jacent)
Courbe d'apprentissage	Moyenne	Facile	Moyenne	Moyenne	Moyenne	Facile	Moyenne (dépend du langage et du framework utilisé)
BDD Support	Non natif (via Cucumber)	Non natif (via plugins)	Non natif (via plugins)	Non	Non	Non natif (via extensions)	Oui (conçu pour BDD)
Rapports intégrés	Non natif (via Allure, Extent)	Oui	Oui	Oui	Oui	Oui	Oui (via plugins comme Allure)

Écosystème & Communauté	Très large (leader du marché)	Large et en croissance	En forte croissance	Très large	Très large	Large et stable	Large et actif
------------------------------------	-------------------------------	------------------------	---------------------	------------	------------	-----------------	----------------

Chapitre 9 : Selenium



Présentation de Selenium WebDriver

Selenium est un outil open-source pour automatiser les tests sur les navigateurs web.

Installation et configuration

- Installer Selenium via pip : `pip install selenium`.
- Télécharger le WebDriver correspondant à votre navigateur.

- **Exemple de code : Automatisation d'un scénario de connexion**

```
python
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("https://www.example.com/login")
driver.find_element_by_id("username").send_keys("testuser")
driver.find_element_by_id("password").send_keys("password123")
driver.find_element_by_id("login-btn").click()
assert "Dashboard" in driver.title
driver.quit()
```

Chapitre 10 : Cucumber



Introduction à Cucumber et au Behavior-Driven Development (BDD)

Cucumber permet d'écrire des tests en langage naturel, facilitant la collaboration entre les développeurs et les non-développeurs.

Écriture de scénarios en Gherkin

- **Given** : Préconditions.
- **When** : Actions.
- **Then** : Résultats attendus.

- **Exemple de code : Test d'une fonctionnalité de recherche**

```
gherkin
Fonctionnalité: Recherche sur le site
Scénario: Rechercher un produit
    Étant donné que je suis sur la page d'accueil
    Quand je saisis "ordinateur portable" dans la barre de recherche
    Et que je clique sur "Rechercher"
    Alors je devrais voir une liste de produits correspondants
```

Chapitre 11 : Playwright



Présentation de Playwright et ses avantages

Playwright permet d'automatiser les tests sur plusieurs navigateurs (Chromium, Firefox, WebKit).

Automatisation des tests sur plusieurs navigateurs

- Utiliser Playwright pour exécuter les mêmes tests sur différents navigateurs.
- **Exemple de code : Test d'une navigation entre pages**

```
javascript
const { chromium } = require('playwright');

(async () => {
  const browser = await chromium.launch();
  const page = await browser.newPage();
  await page.goto('https://www.example.com');
  await page.click('text=About Us');
  await expect(page).toHaveTitle('About Us');
  await browser.close();
})();
```

Chapitre 12 : Cypress



Introduction à Cypress et ses spécificités

Cypress est un outil moderne pour les tests end-to-end, offrant une exécution rapide et un débogage facile.

Configuration et exécution des tests

- Installer Cypress via npm : `npm install cypress`.
- Exécuter les tests avec `npx cypress open`.
- **Exemple de code : Test d'un formulaire de contact**

```
javascript
describe('Contact Form Test', () => {
  it('Should submit the contact form', () => {
    cy.visit('https://www.example.com/contact');
    cy.get('#name').type('John Doe');
    cy.get('#email').type('john.doe@example.com');
    cy.get('#message').type('This is a test message');
    cy.get('#submit-btn').click();
    cy.contains('Thank you for your message!');
  });
});
```

Chapitre 13 : Robot Framework

Présentation de Robot Framework et sa syntaxe simple



Robot Framework est un framework open-source pour les tests automatisés, utilisant une syntaxe simple et lisible.

Automatisation des tests fonctionnels

- Écrire des tests en utilisant des mots-clés.
- **Exemple de code : Test d'une fonctionnalité de paiement**

```
robot
*** Settings ***
Library SeleniumLibrary

*** Test Cases ***
Paiement réussi
    Open Browser https://www.example.com chrome
    Input Text id=card-number 4111111111111111
    Input Text id=expiry-date 12/25
    Input Text id=cvv 123
    Click Button id=pay-button
    Page Should Contain Paiement réussi
    Close Browser
```

Chapitre 14 : Intégration Continue et Tests Automatisés



Jenkins est un outil d'intégration continue (CI) et de livraison continue (CD) open-source qui permet d'automatiser la construction, les tests et le déploiement des applications via des pipelines configurés par l'utilisateur.



CI/CD

GitLab CI est un service intégré à GitLab qui permet d'automatiser les processus de construction, de test et de déploiement des applications grâce à des pipelines définis dans un fichier « .gitlab-ci.yml ».



Azure DevOps

Azure DevOps est une suite d'outils de Microsoft qui couvre l'intégration continue, la gestion de projets, et la livraison continue, tout en facilitant la gestion du code, des tests et du déploiement.



GitHub Actions

GitHub Actions est une fonctionnalité intégrée à GitHub permettant d'automatiser des workflows CI/CD en utilisant des fichiers YAML pour configurer des actions déclenchées par des événements sur les dépôts.

- **Intégration des tests automatisés dans un pipeline CI/CD**
 - Utiliser Jenkins, GitLab CI, Azure DevOps ou GitHub Actions pour exécuter les tests automatiquement à chaque commit.
- **Exemple : Configuration d'un pipeline avec Jenkins et Selenium**

- Configurer un job Jenkins pour exécuter les tests Selenium à chaque build.

Prérequis :

- Jenkins installé et en cours d'exécution.
- Selenium WebDriver installé dans le projet.
- Un projet d'automatisation de tests existant (par exemple, un projet Maven/Gradle avec Selenium).

Étapes de configuration d'un pipeline Jenkins avec Selenium :

Créez un nouveau projet Jenkins :

- Ouvrez Jenkins.
- Cliquez sur **"New Item"**.
- Sélectionnez **"Pipeline"** et donnez un nom au projet, puis cliquez sur **OK**.

Configurez le Pipeline :

- Allez dans la section **Pipeline**.
- Dans **Définition**, choisissez **Pipeline script**.
- Ajoutez le script suivant dans le champ **Pipeline Script** :

```
groovy
pipeline {
    agent any

    environment {
        // Définir les variables d'environnement nécessaires
        SELENIUM_GRID_URL = "http://localhost:4444/wd/hub" // URL du
serveur Selenium Grid (exemple)
    }

    stages {
        stage('Checkout') {
            steps {
                // Vérification du code depuis le repository Git
                git 'https://votre-repository.git'
            }
        }

        stage('Install Dependencies') {
            steps {
                // Si vous utilisez Maven, installez les dépendances
                script {
                    sh 'mvn clean install'
                }
            }
        }

        stage('Run Tests') {
            steps {
                // Lancer les tests Selenium via Maven
                script {
                    sh 'mvn test -Dselenium.grid.url=$SELENIUM_GRID_URL'
```

```

    }
  }
}

stage('Post Results') {
  steps {
    // Si vous voulez générer un rapport de test, vous pouvez
le faire ici
    junit '**/target/test-*.xml' // Remplacez par le chemin de
vos rapports de test
  }
}

post {
  always {
    // Actions à exécuter après chaque exécution du pipeline (ex :
nettoyage)
    echo "Pipeline terminé"
  }
  success {
    // Actions à exécuter en cas de succès
    echo "Tests réussis"
  }
  failure {
    // Actions à exécuter en cas d'échec
    echo "Tests échoués"
  }
}
}

```

- **Comparaison des fonctionnalités**

Critère	Jenkins	GitLab CI/CD	GitHub Actions	Azure DevOps
Type	Serveur d'intégration et déploiement continu (CI/CD)	CI/CD intégré à GitLab	CI/CD intégré à GitHub	Plateforme DevOps complète (CI/CD, gestion de projet, reporting, etc.)
Hébergement	Auto-hébergé	Auto-hébergé & cloud (GitLab.com)	Cloud (GitHub) & Auto-hébergé	Cloud (Azure) & Auto-hébergé
Configuration	Basé sur des plugins, fichiers Jenkinsfile (Pipeline as Code)	Basé sur un fichier .gitlab-ci.yml	Basé sur des fichiers .yaml (workflows)	Basé sur des pipelines YAML
Facilité d'utilisation	Complexe (nécessite configuration manuelle)	Facile à modérée	Facile	Moyenne

Extensibilité	Très grande (via plugins)	Bonne (intégré avec GitLab)	Moyenne (GitHub Marketplace)	Bonne (extensions Marketplace)
Intégration Git	Oui, via plugins	Natif avec GitLab	Natif avec GitHub	Natif avec Azure Repos & GitHub
Parallélisation des jobs	Oui (via agents et configurations avancées)	Oui (via stages et jobs)	Oui (via workflows)	Oui (via pipelines YAML)
Support des runners/agents	Agents personnalisés (nécessite configuration)	GitLab Runners (auto-hébergés ou cloud)	GitHub Runners (cloud ou auto-hébergés)	Agents Microsoft-hosted ou auto-hébergés
Sécurité	Dépend des configurations et plugins	Bonne (intégrée à GitLab)	Bonne (GitHub offre des scans de sécurité)	Très bonne (intégration avec Azure Security)
Coût	Gratuit (open-source), mais coûteux en maintenance	Gratuit pour GitLab CE, payant pour fonctionnalités avancées	Gratuit avec limitations, payant pour plus de minutes	Gratuit pour certaines fonctionnalités, payant pour usages avancés
Intégration avec les Cloud Providers	AWS, Azure, GCP (via plugins)	AWS, Azure, GCP	AWS, Azure, GCP	Fortement intégré avec Azure
Support des tests automatisés	Oui (via plugins comme JUnit, Selenium, Cucumber, etc.)	Oui (tests intégrés dans CI/CD)	Oui (via workflows)	Oui (via pipelines et services Azure Test Plans)
Notifications & reporting	Notifications par email, Slack, Teams (via plugins)	Notifications intégrées	Notifications via GitHub	Notifications via Teams, Slack, email

Partie 4 : Le Test API

Chapitre 15 : Introduction au Test API



Postman : est un outil graphique très populaire utilisé pour tester les API REST. Il permet d'envoyer facilement des requêtes HTTP, d'inspecter les réponses, de gérer des environnements, et d'automatiser des suites de tests API avec des scripts JavaScript.



Karate: est un framework open-source de tests API basé sur Cucumber. Il permet d'écrire des scénarios de tests en langage Gherkin (Given/When/Then) tout en intégrant directement les requêtes HTTP, les assertions, et les tests de performance.



RestAssured: est une bibliothèque Java permettant de tester facilement les API REST. Elle fournit une syntaxe fluide pour envoyer des requêtes HTTP, valider les réponses et s'intègre facilement avec JUnit/TestNG dans des tests automatisés.



SoapUI : est un outil d'automatisation pour tester les services web SOAP et REST. Il permet de tester la sécurité, la charge et les performances des API via une interface graphique, avec la possibilité d'ajouter des assertions complexes sur les réponses.

- **Comparaison des fonctionnalités**

Critères	Postman	Karate	RestAssured	SoapUI
Type d'outil	Interface graphique (GUI)	Framework basé sur Java	Librairie Java	Interface graphique (GUI)
Utilisation principale	Tests API REST et SOAP	Tests API REST avec approche BDD	Tests API REST via code Java	Tests API REST et SOAP
Facilité d'utilisation	Très simple (No-code/Low-code)	Moyenne (DSL spécifique)	Requiert des compétences Java	Facile pour SOAP, plus complexe pour REST
Support des requêtes	REST, SOAP, GraphQL	REST	REST	REST, SOAP
Automatisation	Faible (via scripts Postman)	Native (BDD + Java)	Haute (intégré en Java)	Possible avec Groovy et des scripts
Intégration CI/CD	Via Newman (CLI)	Compatible avec CI/CD	Facilement intégrable (JUnit, TestNG)	Moins adapté pour CI/CD
Format des tests	Collections et scripts (JS)	Scénarios en Gherkin	Code Java (JUnit/TestNG)	XML/TestSteps

Gestion des assertions	JS (Chai, Mocha)	Assertions intégrées (Karate DSL)	Hamcrest, AssertJ	Assertions graphiques et scripts
Support des Mock Services	Oui (via Postman Mock Server)	Oui	Non natif	Oui
Courbe d'apprentissage	Faible	Moyenne	Élevée (Java requis)	Moyenne
Meilleur pour	Tests manuels et automatisés rapides	API Testing avec approche BDD	Tests API robustes en Java	Tests SOAP et REST avancés

Partie 4 : Conseils d'Optimisation

Chapitre 16 : Bonnes Pratiques en Test Logiciel

- **Comment maintenir une suite de tests efficace**
 - Réviser et mettre à jour régulièrement les cas de test.
 - Automatiser les tests répétitifs.
- **Gestion des données de test**
 - Utiliser des données anonymisées et réalistes.
 - Maintenir des jeux de données séparés pour différents environnements.
- **Collaboration entre développeurs et testeurs**
 - Intégrer les testeurs dès le début du projet.
 - Utiliser des outils de collaboration comme Jira ou Confluence.

Chapitre 17 : Tendances Futures en Test Logiciel

L'impact de l'Intelligence Artificielle (IA) et du Machine Learning (ML)

L'intelligence artificielle transforme profondément le domaine du test logiciel :

- Génération automatique de cas de test : à partir de l'analyse du comportement utilisateur ou de l'historique des défauts, l'IA peut proposer des cas de test intelligents, focalisés sur les zones à risque.
- Détection intelligente des anomalies : les modèles de machine learning permettent d'identifier des comportements inattendus dans les résultats de test, même sans règles prédéfinies.
- Optimisation des jeux de données : l'IA peut générer ou ajuster dynamiquement des jeux de données pour couvrir les cas les plus critiques.

Ces approches permettent d'accélérer les cycles de développement, d'augmenter la couverture de test, et de réduire les coûts d'assurance qualité.

Outils Innovants et Technologies Émergentes



Healenium

Healenium est une solution basée sur l'IA qui aide à auto-réparer les tests Selenium cassés suite à des changements dans le DOM (Dynamic Object Model).

En réutilisant des sélecteurs alternatifs ou des historiques de navigation, il permet de réduire considérablement le temps de maintenance des tests automatisés.



ZeroStep

ZeroStep est un outil open source qui permet d'améliorer la lisibilité des rapports de test et d'apporter une documentation claire et structurée des étapes de test, même lorsque le code d'automatisation est dense.

Il est particulièrement utile pour les équipes QA qui souhaitent suivre les bonnes pratiques BDD sans utiliser Gherkin.



Testim est un outil no-code basé sur l'IA pour la création de tests E2E. Il permet une automatisation rapide et intelligente avec apprentissage

automatique pour stabiliser les tests dans le temps.



Mabl est une plateforme de test intelligente qui combine exploration visuelle, tests API et UI, avec des capacités d'apprentissage automatique pour détecter les régressions visuelles ou fonctionnelles.



functionize

utilisateurs.

Functionize est une plateforme qui utilise le NLP (Natural Language Processing) pour permettre aux testeurs de créer des tests en langage naturel, tout en analysant les performances et les comportements

L'évolution des outils et des méthodologies

Tests basés sur le cloud : exécution simultanée sur des environnements multiples, réduisant les coûts et les délais de validation.

- **CI/CD & DevOps** : les tests sont intégrés dans les pipelines de développement, déclenchés à chaque changement de code pour un retour rapide.
- **Testing as Code (TaaC)** : les scripts de test sont versionnés avec le code applicatif, favorisant la collaboration et l'agilité.
- **Monitoring post-déploiement (Shift-right testing)** : les outils comme Datadog, Sentry ou Splunk permettent de surveiller le comportement de l'application en production, identifiant des problèmes non détectés pendant les tests.

Remerciements et Perspectives

Tout au long de ce livre, j'ai exploré les différentes facettes du test logiciel, des bases fondamentales aux méthodologies avancées d'automatisation. Mon objectif était de fournir un guide pratique et enrichissant pour toute personne souhaitant approfondir ses compétences en test logiciel et en automatisation.

Et après ?

L'univers du test logiciel est en perpétuelle évolution. Les outils, frameworks et méthodologies continueront d'évoluer pour répondre aux exigences toujours croissantes de l'industrie du logiciel. Nous vous encourageons à :

- Expérimenter les concepts et outils abordés dans ce livre.
- Rejoindre des communautés et forums pour échanger avec d'autres testeurs.
- Suivre les tendances et innovations du domaine.

Partager vos connaissances et vos expériences avec d'autres passionnés.

Nous espérons que ce livre vous a apporté de nouvelles perspectives et qu'il vous accompagnera dans votre parcours professionnel.

Bon courage et bonne exploration du monde du test logiciel !

Pour toute question ou retour d'expérience, n'hésitez pas à nous contacter sur nos plateformes respectives.

Rejoins la Communauté des Testeurs Passionnés !

Tu veux continuer à apprendre, partager, et évoluer dans le domaine du test logiciel ? Rejoins notre groupe sur les réseaux sociaux, découvre nos projets GitHub, nos vidéos sur YouTube et entraîne-toi sur notre plateforme dédiée pour pratiquer au quotidien.



Chaine YouTube :

<https://www.youtube.com/@testlogicielttn>



Groupe Facebook :

<https://www.facebook.com/groups/585214809554140/>



Groupe LinkedIn :

<https://www.linkedin.com/groups/12954419/>



Page Instagram :

<https://www.instagram.com/qarevolution/?igsh=ZDVtNHd3MW45MmEx#>



Plateforme d'entraînement :

<https://qatraining.fr/>



Projet Github :

<https://github.com/ziedhannachi>