Définition

Une stratégie de tests est un plan détaillé qui définit comment les tests doivent être conçus, mis en place et exécutés pour valider le bon fonctionnement d'un Système ou d'une Application. Elle permet de définir les objectifs, les scénarios, les cas de test et les outils nécessaires pour assurer la qualité du système ou de l'application.

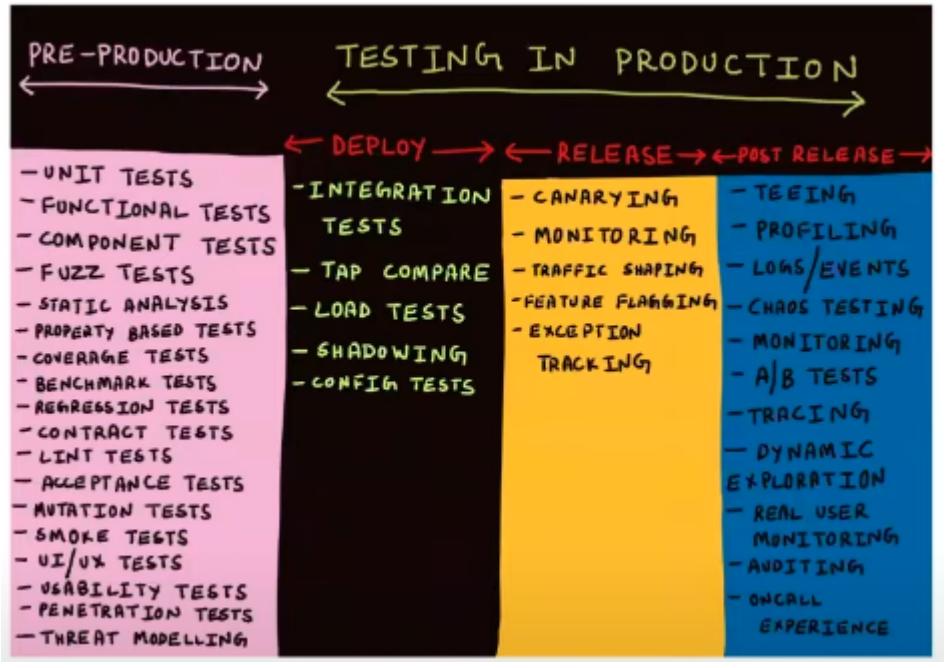
Les principales étapes d'une stratégie de tests incluent :

- La définition des objectifs de tests, c'est-à-dire les critères qui doivent être satisfaits pour que le système ou l'application soit considéré comme fonctionnel.
- La planification des tests, c'est-à-dire la définition des scénarios de test, des cas de test, des environnements de test et des ressources nécessaires.
- La conception des tests, c'est-à-dire la rédaction des scripts de test, la sélection des outils de test et la mise en place des environnements de test.
- L'exécution des tests, c'est-à-dire l'exécution des scripts de test et la collecte des résultats.
- L'analyse des résultats, c'est-à-dire l'analyse des résultats des tests pour déterminer si les objectifs de tests ont été atteints et pour identifier les éventuels défauts ou problèmes.
- La gestion des anomalies, c'est-à-dire la gestion des défauts ou problèmes identifiés lors des tests, y compris la correction, la retest et la documentation des anomalies.

Il existe différents types de tests qui peuvent être inclus dans une stratégie de tests, tels que les tests unitaires, les tests d'intégration, les tests fonctionnels, les tests de performance, les tests de sécurité, les tests de régression, etc. Il est important de choisir les types de tests appropriés en fonction des besoins de l'application ou du système.

En somme, une stratégie de tests est un plan détaillé qui définit comment les tests doivent être conçus, mis en place et exécutés pour valider le bon fonctionnement d'un système ou d'une application. Elle permet de définir les objectifs, les scénarios, les cas de test et les outils nécessaires pour assurer la qualité du système ou de l'application. Il est important de choisir les types de tests appropriés en fonction des besoins de l'application ou du système.

Le GOTO; quelque chose à atteindre : exemple :



Infos supplémentaires

Les tests dynamique :

- Test fonctionnel
- Test unitaire
- Property based testing
- Integration testing
- Smoke testing
- Sanity testing
- User acceptance testing (uat)

- System testing (ui testing)
- E2E testing
- Fuzz testing
- Model based testing
- Manual explorary tests
- Regulatory, compliance tests
- Intrusion testing
- Service level Fault injection testing
- Tests de scénarios d'utilisation
- Test de compatibilité
- Tests d'égalité, les Tests asynchrones, les Tests d'exception et les Tests de type

2 grands types de tests
Static

- Au moment de la compilation (Analysis (linter))
- Code quality : Lire le code source sans l'executer et qui check les problèmes
- Analyse du code sans execution du code

Dynamique
(ceux cités plus haut !)

On peut & on doit générer la doc via les tests E2E , Screenshot dans les artefacts

à chaque fois qu'on met à jour des tests, on met des mises à jour à la fois sur les tests, et les docs

→ *==ISTQB **- les normes des tests**==*

Black box : je vois rien dedans : API , interface utilisateur, boutton, capteur etc

- e2e
- intégration

White box : code source appel de méthode etc

- unitaire

Stack de test : code coverage : dans l'intégration continue : fail les tests si le code coverage descend
→ Envoyer en prod automatique dès qu'on a assez de confiance

Comment faire une bonne statégie de stest :
Cibler le Scénario essentiel

- Sorry-Cypress (pour cibler des tests end2end le plus rapidement possible & être sûr que les tests de scénario essentiels fonctionne à 100%)

Test en mode boite noir : ne pas savoir comment le code fonctionne - Iso prod

On met des minimums et pas de regressions de coverage pour les tests unitaires

Chaque bug corrigé mènent à un TNR

Dans le cas de grosse entreprise : on fait des tests via un expert fonctionnel du métier (un rh pour infotel et ARHI - puis on écrit les tests via Gherkin Language: Format, Syntax & Gherkin Test in Cucumber par exemple, ce qui permet de décrire les tests et de générer les documentations via les tests (Gherkin))

Liens et ressources liées

Related to :
SonarQube - Self-managed | SonarQube | Sonar

Stack de tests : React end-to-end testing with Jest and Puppeteer - LogRocket Blog (jest + puppetter)